ISSN NO. 2320-5407

Journal homepage: http://www.journalijar.com

INTERNATIONAL JOURNAL OF ADVANCED RESEARCH

The Theory of Perfect Learning

Thesis submitted to

University of Montreal

In partial fulfillment of the requirements

For the award of the degree of

DOCTOR OF PHILOSOPHY

BY

Nonvikan Karl-Augustt Alahassa

Under The Guidance of

Alejandro Murua,

Department of Mathematics and Statistics

University of Montreal

Canada

The Theory of Perfect Learning

Nonvikan Karl-Augustt Alahassa, Ph.D
alahassan@dms.umontreal.ca

December 2020

The Theory of Perfect Learning through The Shallow Potts Neural Network Mixture Model

A Novel Mixture Neural Network Model

Nonvikan Karl-Augustt Alahassa, Ph.D

A Thesis in Statistics

University of Montreal

Contents

Errors: All the main results have been verified by our supervisor Alejandro Murua, and some parts have already been published elsewhere in conferences circumstances, mainly in the Proceedings of the Edge Intelligence Workshop 2020 (Alahassa & Murua, 2020) and The Eighth Annual Canadian Statistics Student Conference (2020). There may be typos errors, for which we apologize and would be grateful to receive corrections at alahassan@dms.umontreal.ca, as there is ongoing Proofreading work for the paper.

keywords: Statistics, Mixture models, Potts model, Neural Networks, Probability and stochastic processes, Graphical models, Structured models, Invariance descriptors, Convolutional Neural Network, Artificial intelligence.

Acknowledgements: We would like to thank the following for contributions and suggestions: Alejandro Murua (My supervisor) who has been involved and has stayed completely determined relentlessly; Yoichi Mototake (from the Institute of Statistical Mathematics of Tokyo) that has also inspired me some researches directions when we met the first time at Neural Information Processing Systems 2019 Conference Annual Meeting – My application to this Conference was received at that time with **NeurIPS Travel Awards**. We would also like to thank IVADO and The Natural Sciences and Engineering Research Council of Canada (NSERC) for funding our PhD studies.

We are also grateful for all the support and guidance from the administration of the Department of Mathematics and Statistics of University of Montreal, as well as all the dedicated teachers who have supported us along the way.

A note about the supervisor. My (full) Professor Alejandro Murua has a very deep background in artificial intelligence and applied machine learning researches, that has inspired me so much. Here are a few works from him and some co-authors that the novice should read to further enrich his knowledge of the topics covered in this document: A 2D extended HMM for speech recognition (Li & Murua (1999)), Tensor train decompositions on recurrent networks (Murua et al. (2020)), Speech recognition using randomized relational decision trees (Amit & Murua (2001)), High-dimensional variable selection with the plaid mixture model for clustering (Chekouo & Murua (2018)), Kernel-based mixture models for classification (Murua & Wicker (2015)), Fast spatial inference in the homogeneous Ising model (Murua & Maitra (2017)), Classification and clustering of stop consonants via nonparametric transformations and wavelets (Gidas & Murua (1995)), Model-based clustering and data transformations for gene expression data (Yeung et al. (2001)), Method and system for enhanced data searching (Marchisio et al. (2008)), Hierarchical model-based clustering of large datasets through fractionation and refractionation (Tantrum et al. (2004)), Assessment and pruning of hierarchical model based clustering (Tantrum et al. (2003)), Probabilistic segmentation and intensity estimation for microarray images (Gottardo et al. (2006)), On Potts Model Clustering, Kernel K-Means and Density Estimation (Murua et al. (2008a)), The conditional-Potts clustering model (Murua & Wicker (2014a)), The penalized biclustering model and related algorithms (Chekouo & Murua (2015)), Functional connectivity mapping using the ferromagnetic Potts spin model (Stanberry et al. (2008a)), The gibbs-plaid biclustering model (Chekouo et al. (2015)), Semiparametric Bayesian regression via Potts model (Murua & Quintana (2017b)), Fast Approximate Complete-data k-nearest-neighbor Estimation (Murua & Wicker (2020)).

Conflict of Interest Declaration. This work is the accomplishment of four (4) years of arduous research on my doctoral thesis. During these years, I made some publications, but there is an essential part of my research which remained private because I had to collaborate with several companies of Canada such as (Robot) Advisors Inc., Quantolio Financial Technologies Inc., ApexMachina, Je rêve d'un Bureau (a French firm), Plant-E Corp, etc. However, I declare that all of the work presented in this paper is the unique culmination of my research at the University of Montreal with my research supervisor Alejandro Murua. *There is no conflict of interest with any of these private companies, and they are not involved in any way.*

Table 1: Key notations guideline

$\boldsymbol{\Theta}$	A model parameter
$\mathcal{B}$	A batch size
Ξ	An integer or binary number
∇	Derivative w.r.t to all dimensions
∂	A partial derivative
$\mathcal{D}$	A set of data
w.r.t	with respect to
ϵ	A very small real number
$\mathbb{E}_{p(\boldsymbol{z})}[\boldsymbol{X}]$	The expectation of X with respect to $p(\boldsymbol{z})$
$\boldsymbol{\pi}_{\xi}$	Probability of a component of label ξ
$\boldsymbol{\mathcal{A}}(Y)$	Expected value of information from random variable Y
$D_{KL}(p\|q)$	The Kullback-Leibler (KL) Divergence between p and q
$(a.e)$	Almost everywhere
ELBO	Evidence Lower Bound
MCMC	Markov Chain Monte Carlo
i.e	That is
e.g	For example
$\|\mathrm{S}\|$	Absolute value of S if S is a number, or Cardinal of S if S is a set
$\otimes$	Kronecker product
$\mathbb{\Lambda}$	A graph set of edges or an edge-set
Λ	Used mostly in chapter [4] as a model parameter
ρ_n	A random partition for n data points
σ	A bandwidth parameter or standard deviation
$\boldsymbol{\zeta}$	Chapter or Section adopted symbol for a representation. Called **Zeta**.
$\boldsymbol{\iota}$	Chapter or Section adopted symbol for a representation. Called **Iota**.
$\propto$	Proportionality symbol
$\approx$	Approximation symbol
$\subset$	Subset of
$\|\cdot\|$	A norm. $\|\cdot\|_F$ is the Frobenius norm
$\cup$	Union with
$d \leftarrow$ e	A symbol used to mean d is assigned the value of e
$d+=e$	Used to mean $d \leftarrow d+e$
$\mathbb{M}_{n\times q}$	The set of matrix with n rows and q columns
$y\|x$	y taken conditionally to x
$\mathcal{S}+$	Space of Positive Definite Matrices

Abstract. The perfect learning exists. We mean a learning model that can be generalized, and moreover, that can always fit perfectly the test data, as well as the training data. We have performed in this thesis many experiments that validate this concept in many ways. The tools are given through the chapters that contain our developments. The classical Multilayer Feedforward model has been re-considered and a novel N_k-architecture is proposed to fit any multivariate regression task. This model can easily be augmented to thousands of possible layers without loss of predictive power, and has the potential to overcome our difficulties simultaneously in building a model that has a good fit on the test data, and don't overfit. His hyper-parameters, the learning rate, the batch size, the number of training times (epochs), the size of each layer, the number of hidden layers, all can be chosen experimentally with cross-validation methods. There is a great advantage to build a more powerful model using mixture models properties. They can self-classify many high dimensional data in a few numbers of mixture components. This is also the case of the Shallow Gibbs Network model that we built as a Random Gibbs Network Forest to reach the performance of the Multilayer feedforward Neural Network in a few numbers of parameters, and fewer backpropagation iterations. To make it happens, we propose a novel optimization framework for our Bayesian Shallow Network, called the Double Backpropagation Scheme (DBS) that can also fit perfectly the data with appropriate learning rate, and which is convergent and universally applicable to any Bayesian neural network problem. The contribution of this model is broad. First, it integrates all the advantages of the Potts Model, which is a very rich random partitions model, that we have also modified to propose its Complete Shrinkage version using agglomerative clustering techniques. The model takes also an advantage of Gibbs Fields for its weights precision matrix structure, mainly through Markov Random Fields, and even has five (5) variants structures at the end: the Full-Gibbs, the Sparse-Gibbs, the Between layer Sparse Gibbs which is the B-Sparse Gibbs in a short, the Compound Symmetry Gibbs (CS-Gibbs in short), and the Sparse Compound Symmetry Gibbs (Sparse-CS-Gibbs) model. The Full-Gibbs is mainly to remind fully-connected models, and the other structures are useful to show how the model can be reduced in terms of complexity with sparsity and parsimony. All those models have been experimented, and the results arouse interest in those structures, in a sense that different structures help to reach different results in terms of Mean Squared Error (MSE) and Relative Root Mean Squared Error (RRMSE). For the Shallow Gibbs Network model, we have found the perfect learning framework : it is the $(l_1, \boldsymbol{\zeta}, \epsilon_{dbs}) - \textbf{DBS}$ configuration, which is a combination of the *Universal Approximation Theorem*, and the DBS optimization, coupled with the (*dist*)-Nearest Neighbor-(h)-Taylor Series-Perfect Multivariate Interpolation (*dist*-NN-(h)-TS-PMI) model [which in turn is a combination of the research of the Nearest Neighborhood for a good Train-Test association, the Taylor Approximation Theorem, and finally the Multivariate Interpolation Method]. It indicates that, with an appropriate number l_1 of neurons on the hidden layer, an optimal number ζ of DBS updates, an optimal DBS learnnig rate ϵ_{dbs}, an optimal distance $dist_{opt}$ in the research of the nearest neighbor in the training dataset for each test data x_i^{test}, an optimal order h_{opt} of the Taylor approximation for the Perfect Multivariate Interpolation (*dist*-NN-(h)-TS-PMI) model once the **DBS** has overfitted the training dataset, the train and the test error converge to zero (0). As the Potts Models and many random Partitions are based on a similarity measure, we open the door to find *sufficient* invariants descriptors in any recognition problem for complex objects such as image; using *metric* learning and invariance descriptor tools, to always reach 100% accuracy. This is also possible with invariant networks that are also universal approximators. Our work closes the gap between the theory and the practice in artificial intelligence, in a sense that it confirms that it is possible to learn with very small error allowed.

0.1 General Introduction

I would like to start this thesis with an uncommon question. What do you guess from the following lines of text?

```
 1N73LL1G3NC3
    15 7H3
  4B1L17Y
70 4D4P7  70
      CH4NG3.
     -73PH3N H4ZK1NG
```

This is the type of question that makes people think of how our mind learn and keep information. Everybody can still guess that the first line is the word "INTELLIGENCE". The other part is much hard for those who are reading that kind of sequence for the first time. I will reveal the answer to quick it off: *Intelligence is the ability to adapt to change - Stephen Hawking*[1]. We may not be right, some others have certainly found something else, better, but still rightful as language sentence. This question reveals the universal truth in statistical learning field: ***machines never mistake, they output what you have encoded, with the right comprehension system, with sufficient intelligencia***.

Nowadays, some of the most heard expression in the domain of computer science and technology is Artificial Intelligence (AI), divided into subfields such as intelligent agents, character recognition, human speech recognition, strategic games, graphs networks, natural language processing, machine learning, autonomously operating cars, reasoning, knowledge representation, planning, machine perception, statistical methods, computational intelligence, and traditional symbolic AI, search and mathematical optimization, artificial neural networks, information theory, intelligent assistants, learning heuristics, knowledge representation and reasoning and deep learning. The set of references in those fields is simply infinitely not limited: Alpaydin (2016), Mitchell et al. (2013), Charniak (1985), Nilsson (2014), McCarthy (1998), Russell & Norvig (2002), Gunning (2017), Minsky (1961), Fogel (1993), Goertzel & Pennachin (2007), Nilsson (2009), Negnevitsky (2005), Cohen (1995), Plant (2011), Haugeland (1989), Ginsberg (2012), McCarthy (1987), El Naqa & Murphy (2015), Mackenzie (2015), Lehr & Ohm (2017), Surden (2014), Mohri et al. (2018), Dey (2016), Beam & Kohane (2018), Wagstaff (2012), Doshi-Velez & Kim (2017), Wexler et al. (2019), Baştanlar & Özuysal (2014), Kononenko & Kukar (2007), Alpaydin (2004), Mohammed et al. (2016), Mitchell et al. (1986), Natarajan (2014), Flach (2012), Mitchell (1997), Chao (2011), Bonaccorso (2017), Chouldechova & Roth (2018), Ayodele (2010), Varshney (2019), Harrington (2012), Murdoch et al. (2019), Flach (2001), Thomaz et al. (2006), Carbonell et al. (1983), Jordan & Mitchell (2015), Lison (2015), Marsland (2015), Nadikattu (2018), Zinkevich (2017), Solomonoff (2006), Goodfellow et al. (2016), LeCun et al. (2015a), Deng & Yu (2014), Nikolenko et al. (2018), Kelleher (2019), Lauzon (2012), Hao et al. (2016), Sejnowski (2018), Wang & Raj (2017), Kim (2017), Kai et al. (2013), Du et al. (2016), Vargas et al. (2017), Ng et al. (2014).

With the evolution of computers, engineers started thinking of a way to let the computer behave like the human brain; and include this kind of intelligence in our everyday life. In the quest of the edge of this achievement, the generalization of statistical methods to machine learning problems has started

[1] This quote has been written in the leet format on the back of a T-shirt hold by Neil deGrasse Tyson in 2019.

with acknowledgements of proven effective results of statistical models and distributions when solving industrial and social problems. As Gaussian-like (Goodman (1963), Ahsanullah et al. (2014), Altman & Bland (1995), Patel & Read (1996), Roy (2003), Bryc (2012)), exponential-like (Marshall & Olkin (1967), Balakrishnan (2018), Verdu (1996), Epstein (1958), Mahdavi & Kundu (2017)), Benford law[2] (Miller (2015), Fewster (2009)) was useful in detection of fraud (Diekmann & Jann (2010), Kossovsky (2014), Tödter (2009), Diekmann (2007)) – in official statistics of a state's election returns (Mebane (2011), Deckert et al. (2011), Mebane & Kalinin (2009), Mebane Jr (2006), Alali & Romero (2013)), in accounting and financial data (Durtschi et al. (2004), Watrin et al. (2008), Nigrini (2017), Krakar & Žgela (2009)), in detection of winning offers for certain ebaY auctions which follow Benford's Law as well (Giles (2007), De Ceuster et al. (1998), Riccioni & Cerqueti (2018)), in money laundering (Badal-Valero et al. (2018)), in image forensics (Fu et al. (2007), Andriotis et al. (2013)), in neural networks (Busta & Weinberg (1998)), and for as many numerous examples in the field, it was obviously certain above all doubt that statistics would rise to the house of God (Hooke (1979), Hood & Jones (2003)) with eternal revolution (Salsburg (2001)).

Another famous example is the Boltzmann distribution. This distribution also called the *Maxwell–Boltzmann or Gibbs* distribution concerns the distribution of an amount of energy between identical but distinguishable particles. The distribution is expressed in the form introduced by (Landau & Evgeny (1980), McDowell (1999), Hernandez (2017), Russell (1996)). We may define it as:

$$q_i \propto e^{-\frac{H_i}{\kappa\eta}} \tag{1}$$

where H_i quantify the energy of the current state; q_i is a probability to be in state i; the constant κ is called or known as the Boltzmann constant and η the temperature of the thermodynamic system. The symbol $\propto$ denotes the proportionality symbol. It represents the probability for the distribution of the states in a system having different energies. The Maxwell-Boltzmann Distribution has already been used to describes the speed of particles in terms of distribution in an idealized gas. It plays an important role in understanding the kinetic, because this distribution forms the basis of the kinetic theory of gases at a certain temperature (Bhattacharyay (2019), Kalanov (2008), Liu et al. (2014), Brandenburger & Steverson (2019)). From this distribution, some interesting properties can be extracted for various applications (Nash (1982); Novak & Bortz (1970); Kozliak (2004); Richmond & Solomon (2001)). In fact, it has tons of application in almost every field in science: resource scheduling, the spatial dynamic of diffusion systems, chemical kinetics, Gibbs entropy and thermodynamic entropy, Quantum Science and Technology, lattice-Boltzmann networks, Boltzmann Workflow Generators, simulated annealing and parallel annealing algorithm, neuromorphic systems, Quantum machine learning, graphical models, probabilistic modelling (Liang et al. (2014), Ernst et al. (2019), Gao et al. (2019), Takeda et al. (2017), Rabbani & Babaei (2019), Aarts & Korst (1988), Neftci et al. (2014), Biamonte et al. (2017), Mühlenbein et al. (1999), Yunpeng et al. (2006))

The Boltzmann Distribution was a preliminary to the Boltzmann Machine. A Boltzmann machine is also known as an equivalent recurrent neural network (Medsker & Jain (2001)) – in binary decisions it

[2]Benford's law, the remarkable logarithmic dissemination of critical digits found within the late nineteenth century. Benford's law, moreover called the Newcomb–Benford law, the law of bizarre numbers, or the first-digit law, is an observation about the repetition of some digits in various real-life sets of numerical data... (Berger & Hill (2015), Berger et al. (2011), Burke & Kincanon (1991)).

has some bias in each node. Boltzmann machines can be assembled to derive more advanced systems such as deep belief networks (Hua et al. (2015), Hinton (2009), Tzortzis & Likas (2007)). Historically, the Boltzmann machine is constructed from a spin-glass model of Sherrington-Kirkpatrick's stochastic Ising Model (Sherrington & Kirkpatrick (1975)). Austrian scientist Ludwig Boltzmann was the first to introduce the Boltzmann distribution in the 20th century, but it was in 1985 that Stanford scientist Geoffroy Hinton and Terry Sejnowski (Hinton & Sejnowski (1983)) has performed advanced researches on this type of network. The success of those authors had come up in the 1980s with a new development of backpropagation (LeCun et al. (1988), Plaut & Hinton (1987), Hinton (2007)). A Boltzmann machine architecture is also equivalent to a known network such as stochastic Hopfield network with hidden units with much more applications and improved features (Barra et al. (2012), Hopfield (2007), Paik & Katsaggelos (1992), Young et al. (1997), Peng et al. (1996), Ramsauer et al. (2020)). More precisely, Boltzmann Machine is obtained by applying simulated annealing on discrete Hopfield network. It is closely related to the idea of a Hopfield network developed in the 1970s, and it relies on ideas from the world of thermodynamics.

A Boltzmann machine may be arranged in units with an assigned quantity called *energy* or *Hamiltonian* defined upon the network itself. Binary results are obtained from those units. It presents also some stochastic properties unlike some Hopfield nets, even though in practice, they both present similar energy distributions, as well as the Ising models (Cipra (1987), Glauber (1963), Pfeuty (1970), Brush (1967), Fredrickson & Andersen (1984), Kadowaki & Nishimori (1998),Joya et al. (2002), Aiyer et al. (1990), Wen et al. (2009), McCoy & Wu (2014)):

$$H = -\left(\sum_{i<j} \Delta_{ij} v_i v_j + \sum_i \delta_i v_i\right) \tag{2}$$

Where Δ_{ij} is to quantify the link, bond or connection between node j and node i, $v_i \in \{0, 1, 2, 3, ..., 100, ...\}$, is the state of unit i, $\{0, 1, 2, 3, ..., 100, ...\}$ a set of possible[3] states for v_i, δ_i represents a bias from unit i in the system ($-\delta_i$ is the activation threshold for the unit i, an external field). In a few theoretical terms, a Boltzmann Machine (BM) is also a probabilistic generative undirected graph model with Boltzmann equilibrium distribution (Upadhya & Sastry (2019), Yasuda (2018), Lauritzen & Richardson (2002)); those graph models satisfy Markov property (Andersson et al. (2001), Frydenberg (1990), Lauritzen et al. (2018)).

One particular example is the binary-Gaussian Boltzmann Machine which is treated as a hybrid model (Cho et al. (2013), Yamashita et al. (2014), Tan et al. (2019), Li et al. (2019d)), with applications (Pappagari et al. (2014), Choo & Lee (2018)). When the Boltzmann machine has a continuous vector Λ of variables (sometimes called *visible* variables), each of them having a Gaussian distribution conditionally to a given binary variable or latent configuration ζ, he is said to be a Gaussian Bernoulli Boltzmann machine written as follows:

$$P(\mathbf{\Lambda}) = \sum_{\zeta} P(\boldsymbol{\zeta}) P(\mathbf{\Lambda}|\boldsymbol{\zeta}) \tag{3}$$

[3]It may be infinite.

This model (in equation 3) can be seen as a Gaussian mixture model with a large number of components. In a purpose of continuous variable binarization, the Gaussian Bernoulli Boltzmann machine is frequently used (for example each data vector Λ, can be associated with its maximum a posteriori estimate). In speech processing applications for example, a model like this can be used as a Universal Background Model (Povey et al. (2008), Hasan & Hansen (2011), Morrison (2011), May et al. (2011), Xiong et al. (2006), Hasan et al. (2010), Snyder et al. (2015)), and it can process higher dimensional vectors with a lot of information than traditional mixture models.

One direct application of the Boltzmann Machine (BM) already quoted above is the Ising Model, which is a *type* of Restricted Boltzmann machines (RBMs), in which the number of states possible to unit i is limited to two (2): $v_i \in \{0, 1\}$. An extended application is the Potts Model, in which the number of possible state for unit i is finite and equal to q. The Potts model is a probabilistic system model created by particles which are data points, and a similarity measure controls their interactions. For a set of data $\{x_i\}_{1:N}$, with $x_i \in \mathbf{R}^d, i = 1, \ldots, N$, we denote $\Xi_{ki} = 1$ if x_i belongs to the k-th cluster, and set our Δ_{ij} in equation 2 to be $\Delta_{ij} = \Delta_{ij}(i, j, \Xi_{ki})$. We can see the product $v_i v_j$ as a similarity measure, and replace it by one more general notation $v_i v_j = s\left(x_i, x_j\right)$. The temperature η still has an influence on the distribution of the system (look for equation 1), and all external fields $-\delta_i$ in 2 are equal to zero. For each η, there is a probability $p_\eta\left(\{\Xi_{ki}\}\right)$ associated with each configuration of the system (Murua et al. (2008a)):

$$p_\eta\left(\{\Xi_{ki}\}\right) \propto \exp\left\{-\frac{1}{\eta} H\left(\{\Xi_{ki}\}\right)\right\} = \exp\left\{-\frac{1}{2\eta}\sum_{i=1}^{n}\sum_{j=1}^{n}\Delta_{ij}(i, j, \Xi_{ki}) s\left(x_i, x_j\right)\right\}$$

The Potts model has the property to return less-likely configurations that assign different labels (clusters) to observations that are similar (Ashkin & Teller (1943); Graner & Glazier (1992); Selke & Huse (1983)). Its applications extend to Some of the popular unsupervised learning methods are clustering, dimensionality reduction, image segmentation, association mining, anomaly detection and generative models (Li & Lowengrub (2014), Blatt et al. (1997); Reichardt & Bornholdt (2004); Kullmann et al. (2000); Grimmett (1994); Asikainen et al. (2003); Coniglio & Peruggi (1982); Georgii & Häggström (1996); Fortuin & Kasteleyn (1972); Murua & Wicker (2014a); Machta et al. (1996); Sweeny (1983); Cardy & Ziff (2003); Grimmett (2004); Tomita & Okabe (2001); Blatt et al. (1996b); Blatt et al. (1996a); Janke & Schakel (2004); Duminil-Copin et al. (2017)). Each of these techniques has a different pattern recognition objective such as identifying latent grouping, identifying latent space, finding irregularities in the data, density estimation or generating new samples from the data.

Taking advantage of the Potts models, and analogously to the stochastic gradient learning method for neural networks (Li & Liang (2018); Bottou (1991); Amari (1993); Bottou (2012); Zinkevich et al. (2010); Paine et al. (2013); Zhang et al. (2013)), we introduced the Shallow Gibbs Potts model, which in-place simulate a large number of clusters generated through random partitioning of the data to infer a similar standard stochastic *batch* learning (Fukumizu (1998); Lange et al. (2012); Sahoo et al. (2017); Hinton et al. (2012a); Runxuan (2005); Li et al. (2020); Hinton et al. (2012b)). The Shallow Gibbs Model is a Bayesian Gaussian mixture model, in which the components weights are generated with the Potts Clustering Model (the latent variable follows a Potts Model), the normal distribution is used for multivariate target prediction, and finally, the weights and biases of the feedforward network involved follow a Markov Random Field (MRF) with three (3) variants of structure investigated: a

sparse, compound symmetric and fully-connected precision matrix respectively. The hyper-parameters are backpropagated through a projection gradient-based learning method, and Monte Carlo sampling are used to obtained estimation of the predicted target via variational inference (Paisley et al. (2012); Mnih & Gregor (2014); Kingma & Welling (2013); Ranganath et al. (2014); Graves (2011); Blei et al. (2017); Salimans et al. (2015); Braun & McAuliffe (2010); Campbell & Li (2019); Campbell & Li (2019); Ranganath et al. (2013); Yin & Zhou (2018); Zhang et al. (2018a); Srivastava & Sutton (2017); Guo et al. (2016)).

Our model may be extended to a Multilayer Gibbs Potts Mixture of Neural Network (MGPM-NN), i.e with a multilayer neural network feature included, with potential research explorations, but in the scope of this thesis, we have restricted the model to a Shallow Network type, with a Bayesian framework. This type of model has already been used and studied in various contexts (Malach & Shalev-Shwartz (2019); Khor et al. (2019); Arjevani & Field (2020); Schindler et al. (2016); Kim & Gofman (2018); McDonnell et al. (2015); Zhang et al. (2017); Kaya et al. (2019); Zhou et al. (2019); Delalleau & Bengio (2011); Bianchini & Scarselli (2014)). In the Sparse model, you will find structures that reduce the Markov connections between the weights of the Neural Network in comparison to fully-connected one. To build our models, some sparse models were very inspirational in the literature (Kepner & Robinett (2019); de Dios & Bruna (2020); Liu et al. (2020); Bourely et al. (2017); Yang & Ma (2019); Mao et al. (2017); Gardner (1989); Glorot et al. (2011); Sun et al. (2016); Louizos et al. (2017); Srinivas et al. (2017)). In the quest of model parsimony, the compound symmetry was also helpful, as it is proven in the literature of mixed models (Wolfinger (1993)).

Our contributions, work and findings. The theory of perfect learning argues a universal model that can beat any type of statistical model given a suitable hyper-parameters input, set appropriately [see 3.10]. We have first introduced some statistical topics [see 1] such as mixture models [see 1.1], Cholesky decomposition [see 1.2.1], graphical models such as Markov Random Fields [see 1.2.2], Kullback-Leibler divergence [1.2.4], gradient and stochastic gradient optimization [see 1.2.3] and **Bernstein-von Mises** theorem for asymptotic convergence of Bayesian estimators [see 1.2.5]. Then, we have extended the Potts models to a novel version called Potts Model with Complete Shrinkage (PMCS) [see 2.3]. This novel framework constrains the size of the clusters which are the components of the partition. We have accomplished the shrinkage by adapting an agglomerative clustering algorithm (Kurita (1991)). There is also a doorway to find the conditional distribution of the bonds given the Shrinked Clusters passing by the distribution of the Potts Model Components size distribution itself using Frequency of frequencies distribution [see 2.6.1]. This last development may be computationally intractable for large datasets, but remains a land of rich mathematical developments. The main advantage of the Potts Model with Complete Shrinkage is that one can simulate random partitions with the Potts Model with a cluster size constraint, and even evaluate *a posteriori* the distribution of the components size. As described in section [see 2.3], we have found evidence of bell shape curve [figure 2.8, figure 2.20] of the constrained cluster size distribution of PMCS, when applied to some datasets taken from the multiple-output benchmark datasets available in the Mulan project website (Tsoumakas et al., 2020) [see Table 2.1].

In a second work, we have built a neural network model universally applicable for regression (even classification) purpose that can *beat* any other model in terms of prediction error or accuracy, random forest included (look in table 3.4). The model can be augmented sufficiently to thousands of layers [see 3.9], and a great number of epochs (training) times to reach a high level of precision. Finally, the

Shallow Gibbs Potts Neural Network model was truly effective in terms of Relative Root Mean Squared Error (RRMSE) against the multi-layer multi-target regression (MMR) Zhen et al. (2017) [see 4.6.1 and 4.1]. Moreover, the chapter allocated to this development has gone into depth about subjects that are important for the learning process: neural network types [3.2], activation functions [3.3], loss functions and its convexity issues [3.3, 3.4], the gradient and stochastic gradients application via backpropagation [3.5], the choice of the batch size, the learning rate schedule, and many more topics such as cross-validation, hold-out, the pruning regularization method [3.6], etc.

We have also shown with experimental trials as well as with mathematical theoretical grounded developments how to simplify in terms of sparsity and parsimony a Bayesian Neural Network, specifically the Shallow Gibbs Network [4.1] with complex structures on his weights distribution. The model weights and biases distribution is Gibbs-based because we have set up two independent fields: one on all the network weights $\mathrm{W} = (\mathrm{W}^{(1)}, \mathrm{W}^{(2)})$, and another one on all the network biases $b = (b^{(1)}, b^{(2)})$. The Sparse-Gibbs Network [see 4.1.1] simplify the model to to a sparsity ratio of $l_1(p+q)^2/(l_1(q+p))^2 = 1/l_1$, which is a very significant value of interest. One contribution of value is how we melt compound symmetry structure on Gaussian Markov Random Fields for the Shallow Gibbs Network in a much more various ways [see 4.1.2 and 4.1.2]. The interest in those structures is more valuable when come our experimental results where we have showed how different precision matrix structures help to reach different level of precision upon the predictions when it comes to the Mean Squared Error, as well as the Relative Root Mean Squared Error (RRMSE) [A, 4.6.1, and 4.6.1]. Beyond the Bayesian Variational Inference framework, that includes all our statistical developments for the model, passing by the approximated Bayes factors and Besag's related algorithm [see 4.4], and going through the Evidence lower Bound (ELBO) to maximize, in order to find the best solution to our Kullback Leibler Divergence optimization, and approximated using Monte Carlo methods (Blundell et al. (2015)). Not the least, the choice of our variational distribution family is not mean fields-based, as it preserves consistently the original structure of the model [4.4].

The shallow network is such a complex model, in a sense that regularisation has been proposed for its compound symmetry structure (the CS-Gibbs model) [4.4.1], and a substantive work has been done to keep the precision matrix positive definiteness property unchanged through the gradient updates which is more precisely an *Iterative projected gradient* (IPG) [see 4.4.1]. The model prediction scheme has also been completely built form the-top-to-bottom in our development [4.5], where it is also explained how the Shallow Gibbs model can be seen as Random Gibbs Network Forest model [4.5].

We have augmented the model (the Shallow Gibbs) with a Double Backpropagation Optimization Scheme called the ***DBS optimizer*** [see 4.5.1], that need to be fine-tuned with appropriate learning rate for the estimated prediction $\hat{\boldsymbol{y}}_{\text{est}}$ to converge. This optimizer has been derived from the Cholesky decomposition used to simulate the response variable $\boldsymbol{y}$ of the model. In fact, the **DBS** optimizer have experimental results of convergence and is convergent [see 4.6.2]. We are convinced that more mathematical developments on this optimizer will help to accommodate the best learning rate given the data set and the prediction scheme. In other words, we claim from our experiments that the **DBS** is a ***universal*** optimizer with a number of **DBS** updates that need to be determined. Our findings from the Potts Model with Complete Shrinkage, to the attractive results of the N_k Hidden Layer Multi-feedforward Neural Network [3.4] for multivariate regression models, all with the potentiality of the Shallow Gibbs Neural Network Potts Model based on mixture models, that comes up with its novel convergent **DBS** optimizer, coupled with our introduced framework in the last chapter that aims to search for the right similarity measure using *sufficient* invariant metric descriptor [6] to reach 100% accuracy in any recog-

nition task, combine everything we call under the modest appellation: **The Theory of The Perfect Learning (TTPL)**, for therefore it is already possible and at work. For the Shallow Gibbs Network model [4.1], we have found the Perfect learning as the $(l_1, \boldsymbol{\zeta}, \epsilon_{dbs}) - \textbf{DBS}$ configuration, where this configuration has to be coupled with our findings in chapter [5], i.e, the **(*dist*)-Nearest Neighbor-(h)-Taylor Series-Perfect Multivariate Interpolation (*dist*-NN-(h)-TS-PMI)** presented in [5.3], all summarized in the following result [4]:

$$\lim_{\boldsymbol{\iota}_{1,\text{opt}},\boldsymbol{\zeta}_{\text{opt}},\boldsymbol{\epsilon}_{\text{dbs,opt}},\text{dist}_{\text{opt}},\boldsymbol{h}_{\text{opt}}} (MSE^{\text{Train}}, MSE^{\text{Test}}) = (0,0) \tag{4}$$

where MSE^{Train}, MSE^{Test} are the Mean Squared Error of the train and test data respectively, $dist_{opt}$ is the optimal distance for the research of the nearest neighbor in the training dataset for each test data x_i^{test}, h_{opt} is the optimal order of the Taylor approximation for the Perfect Multivariate Interpolation (*dist*-NN-(h)-TS-PMI) model once the **DBS** has overfitted the training dataset. $\boldsymbol{\iota}_{1,\text{opt}}, \boldsymbol{\zeta}_{\text{opt}}$ are respectively the optimal number of hidden neurons (neurons on the hidden layer of the Shallow Net), and the optimal number of **DBS** updates, and finally $\boldsymbol{\epsilon}_{dbs,opt}$ which is the optimal DBS learning rate. $\boldsymbol{\epsilon}_{\text{dbs,opt}}$ integrates simultaneously the DBS learning rate vector for all the model parameters, the DBS learning rate for the training data, and the DBS learning rate for the test data.

To find the intuition of those mixed learning ingredients described in the previous paragraph, one needs to realize that Multivariate Interpolation using Taylor theorem is the *refined* generalization of simple Neighborhood Train-Test association [5.2], for the main advantage it offers. Therefore, it is not a coincidence that Taylor Approximation theorem is only defined in a certain vicinity or a given neighborhood set. Moreover, using the Multivariate version of Taylor Theorem [5.2.1], we can see that Taylor Series can help to approximate any (differentiable) function, analogiously to the *Universal Approximation Theorem* [see 4.6.1] for (continuous) functions. Using the assumption $(\mathfrak{F}_d)$ that :

There is an approximable differentiable function $\boldsymbol{\iota}$ such as: $\boldsymbol{\iota}(x_i^{test}) = (\psi_i^{test}, y_i^{test})$, where we already know in a perfect overfitting DBS configuration $(x_{j_1}^{train}, \psi_{j_1}^{train}, y_{j_1}^{train})$ [i.e $\boldsymbol{\iota}(x_{j_1}^{train}) = (\psi_{j_1}^{train}, y_{j_1}^{train})$] and $(x_{j_2}^{train}, \psi_{j_2}^{train}, y_{j_2}^{train})$ [i.e $\boldsymbol{\iota}(x_{j_2}^{train}) = (\psi_{j_2}^{train}, y_{j_2}^{train})$] for two training data j_1 and j_2, With the precision that the training data point j_2 and the test data i are taken in a very closed neighborhood of j_1.

where $\psi_{\cdot}^{\text{train}}$ is the parameter associated to $(x_{\cdot}^{\text{train}}, y_{\cdot}^{\text{train}})$ in the **DBS** optimization; we can sufficiently approximate $(\psi_i^{\text{test}}, y_i^{\text{test}})$, by applying an appropriate Data Augmentation for Empirical Differentiation (DAED) [5.3], which enables a Perfect Multivariate Interpolation model [5.2], the *dist*-NN-(h)-TS-PMI model.

This is a great revolution in *Statistical Learning Theory*, as we can sufficiently (i.e with $h \geq 2$) interpolate any machine learning regression problem, with an empirical differentiation form when we know the perfect parameters that fit the training data. **By then, the famous overfitting problem is no more an hindrance, but an extraordinary gateway to another type of *learning* method.**, as one can still generalize the model with appropriate use of the *Taylor Approximation Theorem*, as illustrated in section [5.3]. The conclusion of this part of the work is even powerful than experimental results, and can be dictated in one **universal *rule*** [as it is mathematically valid under assumption $(\mathfrak{F}_d)$]:

When we overfit, training data can be augmented, and any parametric model can sufficiently be differentiated to approximate test data solutions with small errors allowed (i.e, with almost perfect solutions).

The generalization power of this rule[4] is infinite, as the secret is to understand that every augmented data can also be sufficiently differentiated, as much as we want to overlap all the possible data Space domain:

$$x_{j_1}^{\text{aug-train}} = (x_{j_1}^{\text{train}}(1) + \delta, ..., x_{j_1}^{\text{train}}(q)) \longrightarrow \begin{cases} (x_{j_1}^{\text{train}}(1) + s_1 \cdot \delta, ..., x_{j_1}^{\text{train}}(q)) \\ (x_{j_1}^{\text{train}}(1), x_{j_1}^{\text{train}}(2) + s_2 \cdot \delta, ..., x_{j_1}^{\text{train}}(q)) \\ \cdots\cdots\cdots \\ (x_{j_1}^{\text{train}}(1), x_{j_1}^{\text{train}}(2), ..., x_{j_1}^{\text{train}}(q) + s_q \cdot \delta) \end{cases} \tag{5}$$

with $(s_1, s_2, ..., s_q)$ is a q-tuple of integers or rationals, where $x_{j_1}^{\text{aug-train}}$ is our augmented data in the training set, and $\delta \in \mathbb{R}$ a very small number. The re-training process has to be done progressively to obtain their perfect associated parameters [see 5.4]. To generalize the model to any parametric model, we have proposed an effective Generalized Double Back-Propagation for any parametric model, augmented with a differential and local neighborhood machine learning framework for almost sure convergence presented in section 8.0.6.

We have also discovered with Tiles, and the Turing Machines, and some notions of advanced mathematics, a system of creation of infinite machines capable of [LEARNING AND KNOWING EVERYTHING] (8.0.8).

Thesis chapter organization. The chapter-structure of the thesis is as follows: Chapter 1 is dedicated to those major topics and statistical ingredients that matter a lot in terms of learning, starting from mixture models to divergence metric and even Beirntein Von-Mises theorem for asymptotic convergence of Bayesian estimations. Chapter 2 presents the Potts models, the bonds approach which calls upon percolation theory, the Complete Shrinkage Potts models in which we have successfully constrained the minimum Potts Clusters size; all this with extended research on their distribution respectively (the Potts clusters size). Another major part of the document is deep learning and the classical neural network (feedforward approach), the how & the way it works effectively (chapter 3). We pursue the research by building a unique model based on the Shallow Neural Network Structure, a combination of Gaussian Markov Random Fields, a variational learning method that welcome a novel Bayesian framework: the Shallow Potts Neural Network Gibbs Model. You will see further development of the model in chapter 4. The Nearest Neighbor Multivariate Interpolation has been presented in chapter 5. Our interests went far to develop a generalized metric learning framework for the similarity measures (such as the one in Potts Model) which can be extended for fast image classification, segmentation and retrieval (chapter 6). A supplementary conceptual note about Convolutional Neural Network Gibbs Model has been proposed in chapter 7, and may further get improved in many ways. Essentially, we have introduced some novel architectures to explore [7.4.1] and have highlighted the importance of invariant networks [7.4.3] as universal approximators. This chapter is also an attempt to propose some directives to solve open

[4] Always under the validity of assumption $(\mathfrak{F}_d)$.

questions in chapter [6]. Finally, our concluding remarks, discussion notes and applications will appear in chapter 8.

The Shallow Gibbs model developed in chapter 4 [4] has an interesting application that we have presented in paragraph [8.0.7], titled ***Infinite Zelda Game*** as a note about how the model can also be applied in Artificial Intelligence Stochastic Games using the *dist*-NN-(h)-TS-PMI-$(l_1, \boldsymbol{\zeta}, \epsilon_{dbs}) - \mathbf{DBS}$ optimizer. Using the results from this thesis, we also contend that advances in learning theory would go very far if appropriate comprehension of shallow neural networks kernels is integrated.

Chapter 1

Statistical topics

1.1 Mixture Models

Mixture models, also known as mixtures of distributions, in particular the mixture of normal distributions, have been used extensively as models in a wide variety of important practical situations. They have been cited many years ago in the literature with the work of McLachlan & Peel (2004), Karl Pearson (Améndola et al., 2015), Tarter & Lock (1993), Li (1999), etc. The idea of using them was first popularized by Duda et al. (1973), in their seminal research paper: *Pattern Classification and Scene Analysis*.

They have already been applied to famous problems such as image retrieval (Permuter et al., 2003), image classification and segmentation (Permuter et al. (2006);Chen et al. (2010a); Shen (2006)), handwriting recognition (Bishop, 2006a), speaker identification, (Reynolds, 1995), state-space models (Lemke, 2006), volatility models (Brigo & Mercurio (2002); Alexander (2004); Wang (2001)), biometric verification (Stylianou et al., 2005), incomplete data (Dempster et al., 1977), topic modelling (Wang et al. (2017); Tong & Zhang (2016)), point set registration (Ravikumar et al. (2018), Ravikumar et al. (2018)), etc. For the following lines, Ξ is an integer ($\Xi \geq 1$).

Definition 1.1.1 (Deisenroth & Ong)
A Mixture model is a distribution obtained in the following form by a convex combination of Ξ *simple (base) distributions* $p(\boldsymbol{x})$*:*

$$p(\boldsymbol{x}) = \sum_{\xi=1}^{\Xi} \boldsymbol{\pi_\xi} p_\xi(\boldsymbol{x})$$

$$0 \leqslant \boldsymbol{\pi_\xi} \leqslant 1, \quad \sum_{\xi=1}^{\Xi} \boldsymbol{\pi_\xi} = 1$$

Where the $\boldsymbol{p_\xi}$ *are called the* components *with an assigned distribution family, as e.g : Laplace, Normal, Exponential, Cauchy, Uniform, logistics (Casella & Berger, 2002). The* $\boldsymbol{\pi_\xi}$ *are* called *the mixture weights or mixture proportions.*

Mixture models can also describe datasets with multiple clusters (McLachlan & Basford (1988); Govaert & Nadif (2003); Ng & McLachlan (2014); Coke & Tsao (2010); Verbeek (2004); Melnykov et al. (2010); Govaert & Nadif (2008); Khalidov et al. (2011)). Assume we are given a dataset $\mathfrak{X} =$

$\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N\} = (\boldsymbol{x}_i)_{1:N}$, where $\boldsymbol{x}_n, n = 1, ..., N$ are coming from an *i.i.d* distribution $p(x)$ which is unknown, a Mixture Clustering Model (MCM) of the data is obtained by finding for $\boldsymbol{x}_i$ its correct component p_ξ based on his posterior probability of being generated from this component; that is, the ξ-*th* cluster consists of those observations assigned to the ξ-*th* component $(\xi = 1, ..., \Xi)$. The following notes are dedicated to **Gaussian Mixture Models**, which are the most famous in the literature for their good generalization ability among many important probabilistic models.

1.1.1 Example of Gaussian Mixture Model (McLachlan & Basford, 1988)

Combining Ξ Gaussian distributions $\mathcal{N}(\cdot \mid \mu_\xi, \Sigma_\xi)$ in a convex way give rise to a mixture model called Gaussian mixture, for which we have :

$$\begin{gathered} p(\boldsymbol{x} \mid \boldsymbol{\theta}) = \sum_\xi p(x \mid z = \xi)\, p(z = \xi) = \sum_{\xi=1}^{\Xi} \pi_\xi \, \mathcal{N}\left(\boldsymbol{x} \mid \boldsymbol{\mu}_\xi, \boldsymbol{\Sigma}_\xi\right) \\ \text{with} \quad 0 \leqslant \pi_\xi \leqslant 1, \quad \sum_{\xi=1}^{\Xi} \pi_\xi = 1 \end{gathered} \tag{1.1}$$

where we defined z as a non-observed latent variable. $\mathcal{H} := \left\{\boldsymbol{\mu}_\xi, \boldsymbol{\Sigma}_\xi : \xi = 1, \ldots, \Xi\right\}$ gathers all the components parameters of the model. $\boldsymbol{\pi} = (\boldsymbol{\pi_1}, \boldsymbol{\pi_2}, ..., \boldsymbol{\pi_\xi})$ is the vector of the mixture proportions $\boldsymbol{\pi}_\xi$, where $\boldsymbol{\pi}_\xi$ is the probability to observe the ξ-th component. $\boldsymbol{\mu}_\xi$ and $\boldsymbol{\Sigma}_\xi$ are respectively the mean and the covariance of the ξ-th mixture component $\mathcal{N}\left(\boldsymbol{x} \mid \boldsymbol{\mu}_\xi, \boldsymbol{\Sigma}_\xi\right)$.

An illustration is given in Figure 1.1, displaying an example with a proper fitted GMM mixing weights (in orange) and GMM where we change the weights to be equal $\boldsymbol{\pi}_\xi = 1/6$ (in green) for a 6-Gaussian mixture fitted to the Galaxies dataset[1]. This dataset contains a numeric velocity vector in km/sec of 82 galaxies from 6 well-separated conic sections of the Corona Borealis field in an unfilled survey (Venables & Ripley, 2002).

As k-means, Gaussian mixture models also have clustering models properties for unlabeled data (Likas et al. (2003); Su & Dy (2007); Vermunt (2011); Hamerly & Elkan (2004); Kulis & Jordan (2011); Magidson & Vermunt (2002); Kodinariya & Makwana (2013)). However, the use of Gaussian mixture models over k-means has a few benefits. The key difference between the two is that k-means informs us what data point belongs to which cluster, but does not provide us with the likelihood that any of the data points is for a given cluster.

Maximum Likelihood and Issues Encountered

A natural idea is to maximize the likelohood for π_ξ, μ_ξ, and Σ_ξ, with $\xi = 1, \ldots, \Xi$. Assuming that the observations $(x_i)_{1:N}$ are independent, the log-likelihood is:

$$\ell(\mathcal{H}, \boldsymbol{\pi}; \boldsymbol{x}) = \sum_{i=1}^{N} \log p(x_i) = \underbrace{\sum_{n=1}^{N} \log \sum_{\xi=1}^{\Xi} \pi_\xi \mathcal{N}\left(\boldsymbol{x}_n \mid \boldsymbol{\mu}_\xi, \boldsymbol{\Sigma}_\xi\right)}_{=:\mathcal{L}} \tag{1.2}$$

[1] Look for the Galaxies dataset on this link `https://stat.ethz.ch/R-manual/R-devel/library/MASS/html/galaxies.html`

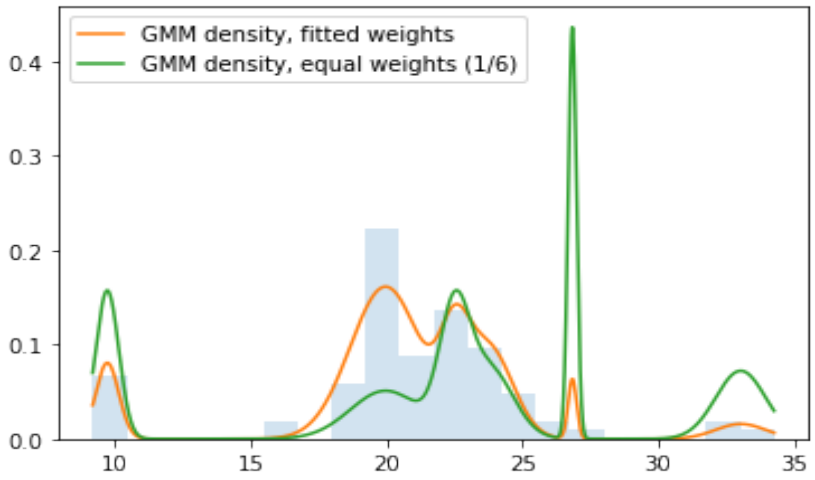


Figure 1.1: Example of a 6-Gaussian Mixture Model (GMM) fitted with the Galaxies Dataset, fitted GMM weights (in orange) and equal weights $\pi_\xi = 1/6$ (in green); histogram in blue.

where x_i is the data vector for observation i, and $p(\cdot)$ is as defined above. Because of the sum of terms inside the logarithm, an algorithm to maximize $\ell(\mathcal{H}, \boldsymbol{\pi}; \boldsymbol{x})$ is difficult to implement numerically. This is mainly due to the $\boldsymbol{\pi}_\xi$ which are unknown. The appropriate maximum likelihood suffers then from some issues. In the number of issues with Maximum Likelihood for Gaussian Mixture Models, we have :

- Non-identifiability: there are many maximums for the likelihood; hence there are not unique, since permuting the names of the components will deliver a diverse set of parameters values that fit the data fairly as well (Kim & Lindsay (2015); Kéry (2018); Jiang & Tanner (1999); Iannario (2010)). Let $(\boldsymbol{\iota}(1), \boldsymbol{\iota}(2), ..., \boldsymbol{\iota}(\xi))$ denote a permutation of the integers $1, \ldots, \xi$. In other words, with regard to any permutation of the parameters positions, the probability likelihood stays invariant. With $(\mathcal{H}, \boldsymbol{\pi}) = \{(\mathcal{H}_1, \boldsymbol{\pi}_1), (\mathcal{H}_2, \boldsymbol{\pi}_2), ..., (\mathcal{H}_k, \boldsymbol{\pi}_\xi)\}$, it means:

$$\ell(\mathcal{H}, \boldsymbol{\pi}; \boldsymbol{x}) = \ell(\boldsymbol{\iota}(\mathcal{H}, \boldsymbol{\pi}); \boldsymbol{x}) \tag{1.3}$$

Let consider this case of a mixture model with two mean parameters $\mathcal{M}_1$ and $\mathcal{M}_2$, for $x = (x_i)_{1:N}$, with a base normal distribution $\mathcal{N}(x_n \mid \mathcal{M}_1, \boldsymbol{\sigma})$ and $\mathcal{N}(x_n \mid \mathcal{M}_2, \boldsymbol{\sigma})$ as for example, with a common scale parameter $\boldsymbol{\sigma} > 0$:

$$p(x \mid \boldsymbol{\rho}, \mathcal{M}_1, \mathcal{M}_2, \boldsymbol{\sigma}) = \prod_{n=1}^{N} \left(\boldsymbol{\rho} \cdot p(x_n \mid \mathcal{M}_1, \boldsymbol{\sigma}) + (1 - \boldsymbol{\rho}) \cdot p(x_n \mid \mathcal{M}_2, \boldsymbol{\sigma})\right)$$

where $\boldsymbol{\rho} \in [0, 1]$. The issue here is exchangeability of the mixture components, because:

$$p(\boldsymbol{\rho}, \mathcal{M}_1, \mathcal{M}_2, \sigma \mid y) = p((1 - \boldsymbol{\rho}), \mathcal{M}_2, \mathcal{M}_1, \sigma \mid y)$$

The problem is compounded by the number Ξ of components in the mixture, mostly when it grows, leading to $\Xi!$ identical Log-likelihood (and posterior) maxima (Stephens, 2000).

- Existence of other local maxima: Even aside from re-labellings, there is often more than one local maximum of the likelihood (Seidel & Ševčíková (2004); Jin et al. (2016); Srebro (2007)). Finding one of the global maxima (or at least a good local maximum) may require searching for the maximum from many different starting points (Ueda et al. (1999); Botev & Kroese (2004)). As for example, suppose when $\Xi > 1$, there exist a component ξ, and some data item, i, for which we have $0 < \pi_\xi < 1, \mu_\xi = x_i$, and $\Sigma_\xi = 0$. This gives a limitless spike of likelihood thickness at a certain point (the actual global maximum will be at a point with infinite likelihood), while other points have non-zero probability density from other components. Because of this problem, we need to try as many starting points as needed to find a good local maximum that may not be one where some Σ_ξ are closed to zero.

Expectation Maximization algorithm

As explained above, there are several issues to find the parameters of a mixture model passing by the likelihood maximization. When the mixture is finite, there is a simpler approach, known as the *Expectation Maximization (EM) algorithm* (Dempster et al. (1977), Dinov (2008), Day (1969)). It has the property to be simple to implement (easy to actualize through updates), and exceptionally stable.

Definition 1.1.2 (GMM Expected Complete Log-Likelihood)
In the case z is still a latent unobserved *variable, we define the following quantity as the Expected Complete log-likelihood for all the data $(x_i, z_i)_{1:N}$, which is as follows:*

$$\mathbb{E}[\boldsymbol{I}_c(\mathcal{H}; \boldsymbol{x}, \boldsymbol{z})] = \sum_1^N \mathbb{E}_{p(\boldsymbol{z})}[\log p\left(\boldsymbol{z}_i \mid \pi\right)] + \sum_1^N \mathbb{E}_{p(\boldsymbol{z})}[\log p\left(\boldsymbol{x}_i \mid \boldsymbol{z}_i, \mu, \Sigma\right)] \tag{1.4}$$

where $\mathbb{E}_{p(\boldsymbol{z})}[\boldsymbol{X}]$ is the expectation of X with respect to $p(\boldsymbol{z})$.

We consider the unobserved latent variable $\boldsymbol{z}$ in equation (1.1), for a set of N data $(x_i)_{1:N}$, we maximize 1.4 and solve:

$$\max_{\mathcal{H}} \sum_{i=1}^N E_{p(\boldsymbol{z})}[\log p\left(\boldsymbol{x}_i, \boldsymbol{z}_i \mid \mathcal{H}\right)]$$

utilizing this specific EM iterative method to numerically approximate the maximum-likelihood. For a Gaussian mixture with Σ_ξ, the desire maximization (EM) calculation can be depicted as follows, where it interchanges between step (E) and (M) until it converges:

1. Choose a random Gaussian parameters $\mathcal{H}$,
2. Repeat the two steps $a)$ and $b)$ below until convergence:
   a) Expectation (E) Step: Calculate the $r_{i\xi}$ value using the existing values of the parameters,i.e, compute the $r_{i\xi} = \mathrm{p}(\mathrm{z_i} = \xi \mid \mathrm{x}_i, \mathcal{H})$ of components for all data x_i. The $r_{i\xi}$ are called the *responsibilities*, by applying Bayes' Rule, we easily have:

$$r_{i\xi} = \mathrm{p}(\mathrm{z_i} = \xi \mid \mathrm{x}_i, \mathcal{H}) = \frac{\pi_\xi N\left(x_i \mid \mu_\xi, \Sigma_\xi\right)}{\sum_{\xi'} \pi_{\xi'} N\left(x_i \mid \mu_{\xi'}, \Sigma_{\xi'}\right)} \tag{1.5}$$

As you can notice, the likelihood of data x_i is proportional to the responsibility $r_{i\xi}$ of the ξ-th mixture component:

$$p\left(\boldsymbol{x}_i \mid \pi_\xi, \boldsymbol{\mu}_\xi, \boldsymbol{\Sigma}_\xi\right) = \pi_\xi \mathcal{N}\left(\boldsymbol{x}_i \mid \boldsymbol{\mu}_\xi, \boldsymbol{\Sigma}_\xi\right)$$

b) Maximisation (M) Step: Re-estimate the parameters using the current responsibilities, by applying a weighted average:

$$\pi_\xi = \frac{1}{N}\sum_i r_{i\xi}, \quad \mu_\xi = \sum_i r_{i\xi} x_i / \sum_i r_{i\xi}, \quad \boldsymbol{\Sigma}_\xi = \frac{1}{N_\xi}\sum_{i=1}^{N} r_{i\xi}\left(\boldsymbol{x}_i - \boldsymbol{\mu}_\xi\right)\left(\boldsymbol{x}_i - \boldsymbol{\mu}_\xi\right)^\top$$

Initializations can be random, or based on an initial guess You may start with some initial guess (perhaps random) of the parameter values, or perhaps with some initial guess of the responsibilities $r_{i\xi}$ (in which case you start with an M step). Continue interchanging and executing E and M steps until there is little change. Now, we need to justify the reason why the updates are set as they are in the *EM* algorithm Maximisation (M) Step.

Theorem 1.1.1 (Update of the GMM Means)
The update of the mean parameters $\boldsymbol{\mu}_\xi, \xi = 1, \ldots, \Xi$, of the GMM is given by:

$$\boldsymbol{\mu}_\xi^{new} = \frac{\sum_{n=1}^{N} r_{n\xi}\boldsymbol{x}_n}{\sum_{n=1}^{N} r_{n\xi}}$$

where the responsibilities $r_{n\xi}$ are defined in equation 1.5

Proof 1.1.1 (Deisenroth & Ong)
The gradient of the log-likelihood with respect to the mean parameters $\mu_\xi, \xi = 1, \ldots, \Xi$, requires us to compute the partial derivative

$$\begin{aligned}\frac{\partial p\left(x_i \mid \mathcal{H}\right)}{\partial \mu_\xi} &= \sum_{j=1}^{\Xi} \pi_j \frac{\partial \mathcal{N}\left(x_i \mid \mu_j, \Sigma_j\right)}{\partial \mu_\xi} = \pi_k \frac{\partial \mathcal{N}\left(x_i \mid \mu_\xi, \Sigma_\xi\right)}{\partial \mu_\xi} \\ &= \pi_\xi \left(x_i - \mu_\xi\right)^\top \Sigma_\xi^{-1} \mathcal{N}\left(x_i \mid \mu_\xi, \Sigma_\xi\right)\end{aligned}$$

where we exploited that only the ξ-th mixture component depends on μ_ξ.

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial \boldsymbol{\mu}_\xi} &= \sum_{i=1}^{N} \frac{\partial \log p\left(\boldsymbol{x}_i \mid \mathcal{H}\right)}{\partial \boldsymbol{\mu}_\xi} = \sum_{n=1}^{N} \frac{1}{p\left(x_i \mid \mathcal{H}\right)} \frac{\partial p\left(\boldsymbol{x}_i \mid \mathcal{H}\right)}{\partial \boldsymbol{\mu}_\xi} \\ &= \sum_{i=1}^{N} \left(\boldsymbol{x}_i - \boldsymbol{\mu}_\xi\right)^\top \boldsymbol{\Sigma}_\xi^{-1} \underbrace{\frac{\pi_\xi \mathcal{N}\left(\boldsymbol{x}_i \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_\xi\right)}{\sum_{j=1}^{\Xi} \pi_j \mathcal{N}\left(x_i \mid \mu_j, \Sigma_j\right)}}_{=r_{i\xi}} \\ &= \sum_{i=1}^{N} r_{i\xi} \left(\boldsymbol{x}_i - \boldsymbol{\mu}_\xi\right)^\top \boldsymbol{\Sigma}_\xi^{-1}\end{aligned}$$

We now solve for μ_ξ^{new} *so that* $\frac{\partial \mathcal{L}\left(\mu_\xi^{new}\right)}{\partial \mu_\xi} = \mathbf{0}^\top$ *and obtain:*

$$\sum_{i=1}^{N} r_{i\xi} \boldsymbol{x}_i = \sum_{i=1}^{N} r_{i\xi} \boldsymbol{\mu}_\xi^{new} \iff \boldsymbol{\mu}_\xi^{new} = \frac{\sum_{i=1}^{N} r_{i\xi} \boldsymbol{x}_i}{\sum_{i=1}^{N} r_{i\xi}}$$

This concludes the proof of Theorem 1.1.1.

Theorem 1.1.2 (Updates of the GMM Covariances)
The update of the covariance parameters $\boldsymbol{\Sigma}_\xi, \xi = 1, \ldots, \Xi$ *of the GMM is given by*

$$\boldsymbol{\Sigma}_\xi^{new} = \frac{1}{N_\xi} \sum_{i=1}^{N} r_{i\xi} \left(\boldsymbol{x}_i - \boldsymbol{\mu}_\xi\right) \left(\boldsymbol{x}_i - \boldsymbol{\mu}_\xi\right)^\top$$

where N_ξ *are defined as:*

$$N_\xi := \sum_{i=1}^{N} r_{i\xi}$$

Proof 1.1.2 (Deisenroth & Ong)
Our approach is to compute the partial derivatives of the log-likelihood $\mathcal{L}$ *with respect to the covariances* $\boldsymbol{\Sigma}_\xi$*, set them to* 0*, and solve for* Σ_ξ*. We start with our general approach*

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{\Sigma}_\xi} = \sum_{i=1}^{N} \frac{\partial \log p\left(\boldsymbol{x}_i \mid \mathcal{H}\right)}{\partial \boldsymbol{\Sigma}_\xi} = \sum_{i=1}^{N} \frac{1}{p\left(x_i \mid \mathcal{H}\right)} \frac{\partial p\left(\boldsymbol{x}_i \mid \mathcal{H}\right)}{\partial \boldsymbol{\Sigma}_\xi}$$

We then obtain

$$\begin{aligned}\frac{\partial p\left(\boldsymbol{x}_i \mid \mathcal{H}\right)}{\partial \boldsymbol{\Sigma}_\xi} &= \frac{\partial}{\partial \boldsymbol{\Sigma}_\xi} \left(\pi_\xi (2\pi)^{-\frac{D}{2}} \det\left(\boldsymbol{\Sigma}_\xi\right)^{-\frac{1}{2}} \exp\left(-\frac{1}{2}\left(\boldsymbol{x}_i - \boldsymbol{\mu}_\xi\right)^\top \boldsymbol{\Sigma}_\xi^{-1} \left(\boldsymbol{x}_i - \boldsymbol{\mu}_\xi\right)\right)\right) \\ &= \pi_\xi (2\pi)^{-\frac{D}{2}} \left[\frac{\partial}{\partial \boldsymbol{\Sigma}_\xi} \det\left(\boldsymbol{\Sigma}_\xi\right)^{-\frac{1}{2}} \exp\left(-\frac{1}{2}\left(\boldsymbol{x}_i - \boldsymbol{\mu}_\xi\right)^\top \boldsymbol{\Sigma}_\xi^{-1} \left(\boldsymbol{x}_i - \boldsymbol{\mu}_\xi\right)\right)\right. \\ &\quad \left. + \det\left(\boldsymbol{\Sigma}_k\right)^{-\frac{1}{2}} \frac{\partial}{\partial \boldsymbol{\Sigma}_\xi} \exp\left(-\frac{1}{2}\left(\boldsymbol{x}_i - \boldsymbol{\mu}_\xi\right)^\top \boldsymbol{\Sigma}_\xi^{-1} \left(\boldsymbol{x}_i - \boldsymbol{\mu}_\xi\right)\right)\right]\end{aligned}$$

We now use the identities:

$$\frac{\partial}{\partial \boldsymbol{\Sigma}_\xi} \det\left(\boldsymbol{\Sigma}_\xi\right)^{-\frac{1}{2}} = -\frac{1}{2}\det\left(\boldsymbol{\Sigma}_\xi\right)^{-\frac{1}{2}}\boldsymbol{\Sigma}_\xi^{-1}$$
$$\frac{\partial}{\partial \boldsymbol{\Sigma}_\xi}\left(\boldsymbol{x}_i-\boldsymbol{\mu}_\xi\right)^\top \boldsymbol{\Sigma}_\xi^{-1}\left(\boldsymbol{x}_i-\boldsymbol{\mu}_\xi\right) = -\boldsymbol{\Sigma}_\xi^{-1}\left(\boldsymbol{x}_i-\boldsymbol{\mu}_\xi\right)\left(\boldsymbol{x}_i-\boldsymbol{\mu}_\xi\right)^\top\boldsymbol{\Sigma}_\xi^{-1}$$

and obtain the desired partial derivative required as:

$$\frac{\partial p\left(\boldsymbol{x}_i \mid \mathcal{H}\right)}{\partial \boldsymbol{\Sigma}_\xi} = \pi_\xi \mathcal{N}\left(\boldsymbol{x}_i \mid \boldsymbol{\mu}_\xi, \boldsymbol{\Sigma}_\xi\right)\cdot\left[-\frac{1}{2}\left(\boldsymbol{\Sigma}_\xi^{-1}-\boldsymbol{\Sigma}_\xi^{-1}\left(\boldsymbol{x}_i-\boldsymbol{\mu}_\xi\right)\left(\boldsymbol{x}_i-\boldsymbol{\mu}_\xi\right)^\top\boldsymbol{\Sigma}_\xi^{-1}\right)\right]$$

Putting everything together, the partial derivative of the log-likelihood with respect to Σ_ξ is given by :

$$\begin{aligned}
\frac{\partial \mathcal{L}}{\partial \boldsymbol{\Sigma}_\xi} &= \sum_{i=1}^{N}\frac{\partial \log p\left(\boldsymbol{x}_i \mid \mathcal{H}\right)}{\partial \boldsymbol{\Sigma}_\xi} = \sum_{n=1}^{N}\frac{1}{p\left(x_i \mid \mathcal{H}\right)}\frac{\partial p\left(\boldsymbol{x}_i \mid \mathcal{H}\right)}{\partial \boldsymbol{\Sigma}_\xi} \\
&= \sum_{i=1}^{N}\underbrace{\frac{\pi_\xi \mathcal{N}\left(\boldsymbol{x}_i \mid \boldsymbol{\mu}_\xi, \boldsymbol{\Sigma}_\xi\right)}{\sum_{j=1}^{\Xi}\pi_j \mathcal{N}\left(x_i \mid \mu_j, \Sigma_j\right)}}_{=r_{i\xi}}\cdot\left[-\frac{1}{2}\left(\boldsymbol{\Sigma}_\xi^{-1}-\boldsymbol{\Sigma}_\xi^{-1}\left(\boldsymbol{x}_i-\boldsymbol{\mu}_\xi\right)\left(\boldsymbol{x}_i-\boldsymbol{\mu}_\xi\right)^\top\boldsymbol{\Sigma}_\xi^{-1}\right)\right] \\
&= -\frac{1}{2}\sum_{i=1}^{N} r_{i\xi}\left(\boldsymbol{\Sigma}_\xi^{-1}-\boldsymbol{\Sigma}_\xi^{-1}\left(\boldsymbol{x}_i-\boldsymbol{\mu}_\xi\right)\left(\boldsymbol{x}_i-\boldsymbol{\mu}_\xi\right)^\top\boldsymbol{\Sigma}_\xi^{-1}\right) \\
&= -\frac{1}{2}\boldsymbol{\Sigma}_\xi^{-1}\sum_{i=1}^{N} r_{i\xi} + \frac{1}{2}\boldsymbol{\Sigma}_\xi^{-1}\left(\sum_{i=1}^{N} r_{i\xi}\left(\boldsymbol{x}_i-\boldsymbol{\mu}_\xi\right)\left(\boldsymbol{x}_i-\boldsymbol{\mu}_\xi\right)^\top\right)\boldsymbol{\Sigma}_\xi^{-1}
\end{aligned}$$

Setting this partial derivative to 0, we obtain the result:

$$\begin{aligned}
N_\xi\boldsymbol{\Sigma}_\xi^{-1} &= \boldsymbol{\Sigma}_\xi^{-1}\left(\sum_{i=1}^{N} r_{i\xi}\left(\boldsymbol{x}_i-\boldsymbol{\mu}_\xi\right)\left(\boldsymbol{x}_i-\boldsymbol{\mu}_\xi\right)^\top\right)\boldsymbol{\Sigma}_\xi^{-1} \\
\iff N_\xi\boldsymbol{I} &= \left(\sum_{i=1}^{N} r_{i\xi}\left(\boldsymbol{x}_i-\boldsymbol{\mu}_\xi\right)\left(\boldsymbol{x}_i-\boldsymbol{\mu}_\xi\right)^\top\right)\boldsymbol{\Sigma}_\xi^{-1}
\end{aligned}$$

By solving for $\boldsymbol{\Sigma}_\xi$, we obtain

$$\boldsymbol{\Sigma}_\xi^{new} = \frac{1}{N_\xi}\sum_{n=1}^{N} r_{n\xi}\left(\boldsymbol{x}_n-\boldsymbol{\mu}_\xi\right)\left(\boldsymbol{x}_n-\boldsymbol{\mu}_\xi\right)^\top$$

This gives us a simple update rule for Σ_ξ for $\xi = 1, \ldots, \Xi$ and proves Theorem 1.1.2.

Theorem 1.1.3 (Update of the GMM Mixture Weights)
The mixture weights π_k of the GMM are updated as:

$$\pi_\xi^{new} = \frac{N_\xi}{N}, \quad \xi = 1, \ldots, \Xi$$

where N is the number of data points and N_ξ is defined in Theorem 1.1.2.

Proof 1.1.3 (Deisenroth & Ong)
To find the partial derivative of the log-likelihood with respect to the weight parameters $\pi_\xi, \xi = 1, \ldots, \Xi$, we account for the constraint $\sum_\xi \pi_\xi = 1$ by using Lagrange multipliers (Rockafellar, 1993). The Lagrangian is:

$$\mathfrak{L} = \mathcal{L} + \lambda \left(\sum_{\xi=1}^{\Xi} \pi_\xi - 1 \right) = \sum_{i=1}^{N} \log \sum_{k=1}^{\Xi} \pi_\xi \mathcal{N}\left(\boldsymbol{x}_i \mid \boldsymbol{\mu}_\xi, \boldsymbol{\Sigma}_\xi\right) + \lambda \left(\sum_{\xi=1}^{\Xi} \pi_\xi - 1 \right)$$

where $\mathcal{L}$ is the log-likelihood defined in equation 1.2 and the second term encodes for the equality constraint that all the mixture weights need to sum up to 1. We obtain the partial derivative with respect to π_ξ as:

$$\begin{aligned} \frac{\partial \mathfrak{L}}{\partial \pi_\xi} &= \sum_{i=1}^{N} \frac{\mathcal{N}\left(\boldsymbol{x}_i \mid \boldsymbol{\mu}_\xi, \boldsymbol{\Sigma}_\xi\right)}{\sum_{j=1}^{\Xi} \pi_j \mathcal{N}\left(\boldsymbol{x}_i \mid \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j\right)} + \lambda \\ &= \frac{1}{\pi_\xi} \underbrace{\sum_{i=1}^{N} \frac{\pi_\xi \mathcal{N}\left(\boldsymbol{x}_i \mid \boldsymbol{\mu}_\xi, \boldsymbol{\Sigma}_\xi\right)}{\sum_{j=1}^{\Xi} \pi_j \mathcal{N}\left(\boldsymbol{x}_i \mid \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j\right)}}_{=N_\xi} + \lambda = \frac{N_\xi}{\pi_\xi} + \lambda \end{aligned}$$

and the partial derivative with respect to the Lagrange multiplier λ as:

$$\frac{\partial \mathfrak{L}}{\partial \lambda} = \sum_{\xi=1}^{\Xi} \pi_\xi - 1$$

Setting both partial derivatives to 0 (a necessary condition for optimum) yields the system of equations:

$$\begin{aligned} \pi_\xi &= -\frac{N_\xi}{\lambda} \\ 1 &= \sum_{\xi=1}^{K} \pi_\xi \end{aligned}$$

Solving for π_ξ, we obtain:

$$\sum_{\xi=1}^{\Xi} \pi_\xi = 1 \iff -\sum_{\xi=1}^{\Xi} \frac{N_\xi}{\lambda} = 1 \iff -\frac{N}{\lambda} = 1 \iff \lambda = -N$$

This allows us to substitute $-N$ for λ to obtain:

$$\pi_\xi^{\text{new}} = \frac{N_\xi}{N}$$

which gives us the update for the weight parameters π_ξ and proves Theorem 1.1.3.

Illustrative Python Example of *EM* algorithm at work

To help us illustrate the *EM* algorithm, we first generate some observations from a mixture of Gaussians. For this we choose three (4) mixture components, with the following settings:

$$\pi_1 = \frac{1}{4}, \boldsymbol{\mu}_1 = (1, 9), \boldsymbol{\Sigma}_1 = \begin{pmatrix} 0.5 & 0 \\ 0 & 0.2 \end{pmatrix}$$
$$\pi_2 = \frac{1}{3}, \boldsymbol{\mu}_1 = (1, 1), \boldsymbol{\Sigma}_2 = \begin{pmatrix} 0.92 & 0.38 \\ 0.39 & 0.91 \end{pmatrix}$$
$$\pi_3 = \frac{1}{4}, \boldsymbol{\mu}_1 = (8, 1), \boldsymbol{\Sigma}_3 = \begin{pmatrix} 0.2 & 0 \\ 0 & 0.2 \end{pmatrix}$$
$$\pi_4 = \frac{1}{6}, \boldsymbol{\mu}_1 = (8, 10), \boldsymbol{\Sigma}_4 = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

Now, we generate $N = 300$ data from the three (4) components, and randomly. Once our data $(x_i)_{1:N}$ is generated, we plot the result in a 2D plot (in figure 1.2):

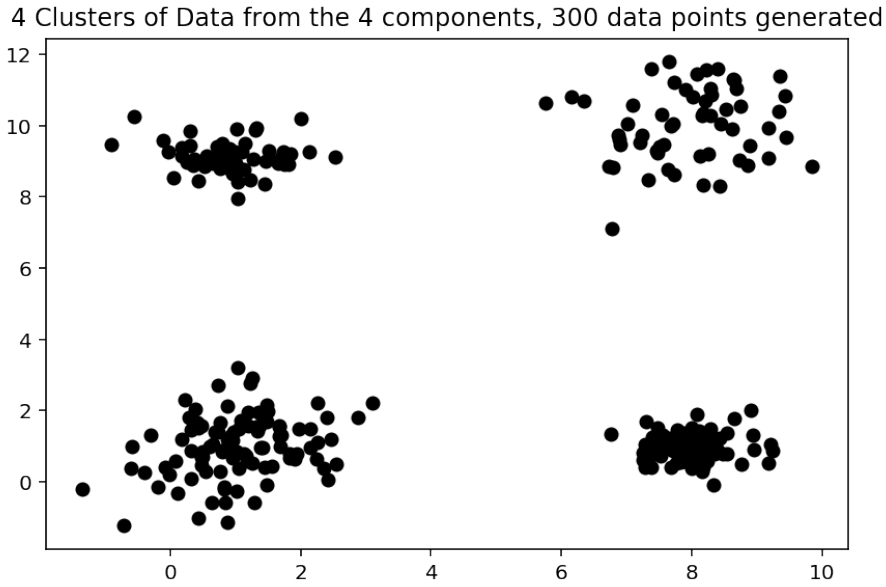


Figure 1.2: 4 Clusters of Data from Our 4 components set for EM illustration

EM algorithm *tip*: In this simple case, you can take the empirical covariances of the data to be the initial covariances in each cluster, as well as the four(4) points $[\boldsymbol{\mu}_1, \boldsymbol{\mu}_2, \boldsymbol{\mu}_3, \boldsymbol{\mu}_4]$ to be the initial cluster means. On the other hand, using K-means as a first step and using the means and the covariances from such clusters to initialise the EM algorithm works.

Based on what is known as a contour map, one way to show a two-dimensional Gaussian distribution is what our simulated data looks like for a single iteration of the EM algorithm (figure 1.3).

After 12 iterations, we have reached convergence, but we have increased iterations to 50 (figure 1.4).

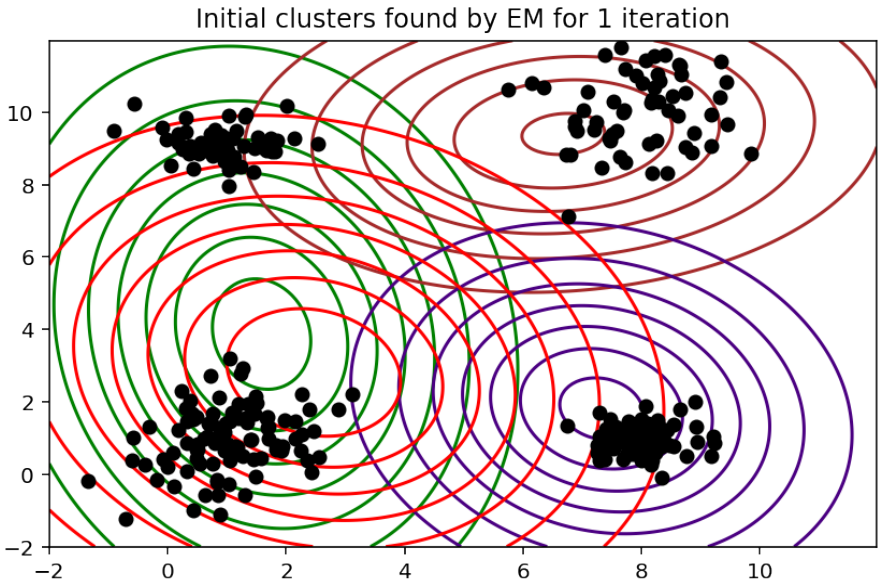


Figure 1.3: Initial GMM Clustering for the simulated data, iteration=1

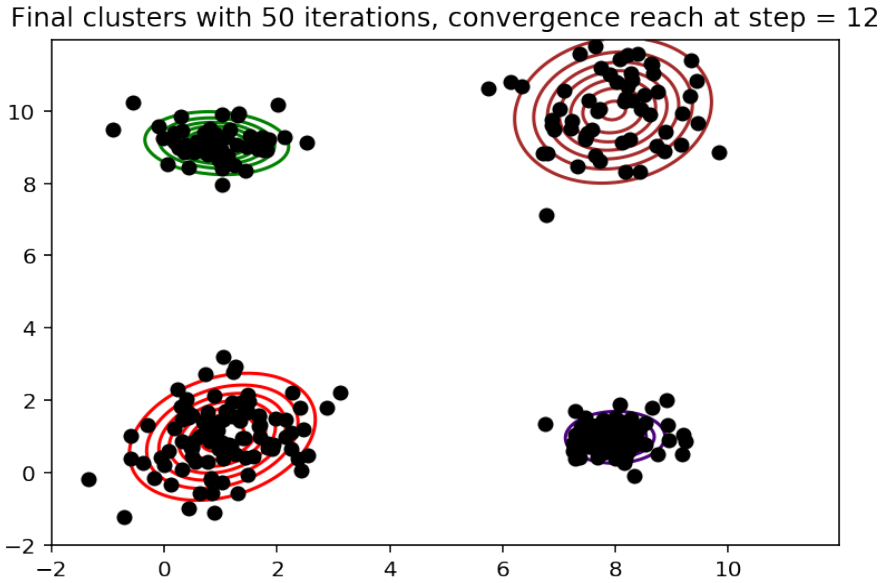


Figure 1.4: GMM Clustering for the simulated data, EM algorithm, iteration=50

The odds and the ends: how do you choose Ξ?

The Curse of dimensionality: One downside of GMMs is that there are heaps of parameters to learn, therefore they may require loads of data and an increased number of parameters to get great results. An unconstrained model with τ -mixtures (or simply τ clusters) and κ dimensional data involves fitting $\kappa \times \kappa \times \tau + \kappa \times \tau + \tau$ parameters (τ covariance matrices each of size $\kappa \times \kappa$, plus τ mean vectors of length κ, plus a weight vector of length τ). That could be a problem for datasets with a large number of dimensions (e.g. text data - Hansen et al. (2000); Bingham & Mannila (2001); Fattah & Ren (2009)), since with the number of parameters required generally is as the square of the dimension, it might immediately get difficult to find an adequate amount of data to make good inferences. (Steinley & Brusco (2011); Huber & Hanebeck (2008); Meinicke & Ritter (2001); Kim et al. (2015)).

If you choose Ξ too small, you'll underfit the data, whereas if you choose it too large, you can overfit. One solution is to choose it using a validation set. I.e., you may choose the value of Ξ which maximizes the average log-likelihood on the validation set (Grimm et al. (2017)). There are many other approaches available to find Ξ (Melnykov & Melnykov (2012); Figueiredo & Jain (2002); Figueiredo et al. (1999); Zhang et al. (2004); Pernkopf & Bouchaffra (2005); Yang et al. (2012); Najar et al. (2019); Zivkovic & van der Heijden (2004); McLachlan et al. (1999); Chen et al. (2004)).

Similar to K-means, GMM requires the user to specify the number of components (clusters) before training the model. Here, we can use the Aikaki Information Criterion (AIC). or the Bayesian Information Criterion (BIC) to aid us in this decision (Steele & Raftery (2010); Soromenho (1994); Agusta & Dowe (2003); Brame et al. (2006); Huang et al. (2017); Hirai & Yamanishi (2013); Chen & Khalili (2009); Kim & Seo (2014); Hirai & Yamanishi (2011)). Let $\mathfrak{M}$ be the maximum value of the likelihood function for the model, $\boldsymbol{e}$ be the number of estimated parameters in the model and N be the total number of data points. Then the AIC value $\mathfrak{C}_{AIC}$ of the model is the following:

$$\mathfrak{C}_{AIC} = 2 \cdot \boldsymbol{e} - 2 \cdot \ln(\mathfrak{M})$$

And the BIC value $\mathfrak{C}_{BIC}$ is denoted as:

$$\mathfrak{C}_{BIC} = -2 \cdot \ln(\mathfrak{M}) + \boldsymbol{e} \cdot \ln(N)$$

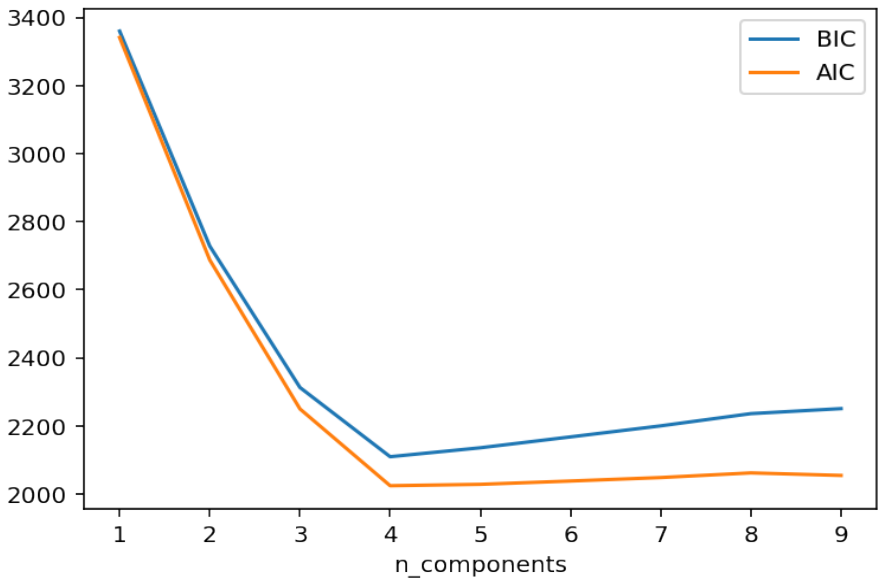


Figure 1.5: GMM AIC & BIC for the simulated data, Scikit Learn GMM, iteration=1000

For both evaluation criterion, AIC and BIC propose four (4) components (figure 1.5). There is also a very elegant framework called Bayesian Mixture Models (Nasserinejad et al. (2017); Elguebaly & Bouguila (2011); Miller & Harrison (2018); Frühwirth-Schnatter & Pyne (2010)) which automatically adjusts the model complexity without requiring a validation set, but that is beyond the scope of this *EM* illustration. We will still grasp your attention in the following section to Dirichlet Mixture Models which is one of those Bayesian Models.

1.1.2 Notes on Dirichlet Mixture Models

The mixture models discussed above have their Bayesian version in which the mixture proportions and the parameters of the components have all an assigned prior distribution: they become random variables. The Dirichlet Mixture Models (DMM) represent one particular case of Bayesian mixture models (Moser et al. (2015); Svensén & Bishop (2005); Medvedovic et al. (2004); Lartillot & Philippe (2004); Do et al. (2005)) in which the mixture proportion π_k (or the mixture weights) the weights are typically viewed as a K-dimensional random vector drawn from a Dirichlet distribution (the conjugate prior of the categorical distribution), and the parameters will be distributed according to their respective conjugate priors (Steck & Jaakkola (2002), Tu (2014)).

Definition 1.1.3 (Darroch & Ratcliff (1971); Ongaro & Migliorati (2013); Lin (2016); Geiger & Heckerman (1996))
Let $Y^k = [Y_1, \ldots, Y_k]$ *be a vector with* k *components, where* $Y_i \geq 0$ *for* $i = 1, 2, \ldots, k$ *and* $\sum_{i=1}^{k} Y_i = 1$. *Also, let* $\mathcal{E}^k = [\mathcal{E}_1, \mathcal{E}_2, \ldots \ \mathcal{E}_k]$, *where* $\alpha_i > 0$ *for each i. Then the Dirichlet probability density function is:*

$$f\left(y^k\right) = \frac{\Gamma\left(\mathcal{E}_0\right)}{\prod_{i=1}^{k} \Gamma\left(\mathcal{E}_i\right)} \prod_{i=1}^{k} y_i^{\mathcal{E}_i - 1} \tag{1.6}$$

where $\mathcal{E}_0 = \sum_{i=1}^{k} \mathcal{E}_i, y_i > 0, y_1 + \cdots + y_{k-1} < 1$ *and* $y_k = 1 - y_1 - \cdots - y_{k-1}$. *We denote this distribution by* $\mathrm{Dir}\left(\mathcal{E}_1, \mathcal{E}_2, \ldots, \mathcal{E}_k\right)$.

In practice, we have two types of priors implementation based on Dirichlet representation of the components weights: a **finite mixture model with Dirichlet distribution** and an **infinite mixture model with the Dirichlet Process** (Antoniak (1974)). In practice, the Dirichlet Process inference algorithm is approximated and uses a truncated distribution with a fixed maximum number of components (called the *Stick-breaking representation*). The number of components used almost always depends on the data.

Since the Dirichlet distribution is a multi-dimensional Beta distribution, the stick-breaking approach (Ishwaran & Zarepour (2002); Lee & MacEachern (2020); Zarepour & Al Labadi (2012); Ghahramani et al. (2010); Broderick et al. (2012); Griffin & Steel (2011); Griffin & Steel (2006)) can be used for generating Dirichlet random variables. The general principle is to first consider a stick with a length of 1., then split it using a suitable Beta distribution to stick it into two parts, and hold one piece of stick. Then, properly split the remaining stick into two pieces (Paisley (2010)). Repeat this process until there are k pieces of the stick. This stick-breaking method generates a random vector $(V_1, \ldots, V_i, \ldots, V_k)$, which is distributed as a Dirichlet distribution $\mathrm{Dir}(\mathcal{E}_1, \ldots, \mathcal{E}_k)$, where V_i is the length of the i^{th} piece of the original stick. Mathematically, we generate a random vector $(V_1, \ldots, V_k)$ as follows:

- Simulate a random variate $U_j \sim \mathrm{Beta}\left(\mathcal{E}_j, \sum_{i=j+1}^{k} \mathcal{E}_i\right)$, where $j = 1, \ldots k-1$. When $j = 1$, we have $U_1 \sim \mathrm{Beta}\left(\mathcal{E}_1, \sum_{i=2}^{k} \mathcal{E}_i\right)$. The first piece of the stick has length $1 \cdot U_1$, such that the length of the remaining stick is $1 - U_1$. Also, set $V_1 = U_1$.
- When $j = 2$, we have $U_2 \sim \mathrm{Beta}\left(\mathcal{E}_2, \sum_{i=3}^{k} \mathcal{E}_i\right)$. The second piece of the stick has length $(1 - U_1) U_2$, such that the length of the remaining stick is $(1 - U_1) - (1 - U_1) U_2 = (1 - U_1)(1 - U_2)$. Also, set $U_2 = (1 - U_1) U_2$.
- When $j = k-1$, we have $U_{k-1} \sim \mathrm{Beta}(\mathcal{E}_{k-1}, \mathcal{E}_k)$. The $(k-1)^{th}$ piece of the stick has length $U_{k-1} \prod_{j=1}^{k-2} (1 - U_j)$, such that the length of the remaining stick is $\prod_{j=1}^{k-1} (1 - U_j)$. Also, set $V_{k-1} = U_{k-1} \prod_{j=1}^{k-2} (1 - U_j)$ Note that the k^{th} piece of the stick has length $\prod_{j=1}^{k-1} (1 - U_j)$ and set $V_k = \prod_{j=1}^{k-1} (1 - U_j)$.

The Dirichlet process is a prior probability distribution on partition clusters when the number of partitions is said to be infinite or not finite. Variational techniques have this advantage, compared to a finite Gaussian mixture model, to help to integrate this prior structure into Gaussian mixture models (GMM) at almost no penalty in inference time. In other words, most of the time, variational inference algorithms are applied to Dirichlet infinite mixture models for the practice of their estimation.

Mathematical formulation of the Dirichlet Mixture Models

Let us consider the mixture of Ξ Gaussian models:

$$q(\boldsymbol{x} \mid \Pi, \mathrm{M}, \Sigma) = \sum_{\xi=1}^{\Xi} \pi_\xi N\left(\boldsymbol{x} \mid \boldsymbol{\mu}_\xi, \boldsymbol{\Sigma}_\xi^{-1}\right)$$

where $\Pi = (\pi_1, \ldots, \pi_\Xi)$, $\mathrm{M} = (\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_\Xi)$, $\Sigma = (\boldsymbol{\Sigma}_1, \ldots, \boldsymbol{\Sigma}_\Xi)$, and $N\left(\boldsymbol{x} \mid \boldsymbol{\mu}, \boldsymbol{\Sigma}^{-1}\right)$ denotes the Gaussian density with expectation $\boldsymbol{\mu}$ and variance-covariance matrix $\boldsymbol{\Sigma}^{-1}$ (or precision matrix $\boldsymbol{\Sigma}$:

$$N\left(\boldsymbol{x} \mid \boldsymbol{\mu}, \boldsymbol{\Sigma}^{-1}\right) = \frac{\sqrt{\det(\boldsymbol{\Sigma})}}{(2\pi)^{d/2}} \exp\left(-\frac{1}{2}(\boldsymbol{x} - \boldsymbol{\mu})^\top \boldsymbol{\Sigma} (\boldsymbol{x} - \boldsymbol{\mu})\right)$$

For training samples $\mathcal{D} = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\}$ drawn independently from the true density $p(\boldsymbol{x})$, the likelihood $p(\mathcal{D} \mid \Pi, \mathrm{M}, \Sigma)$ is given by:

$$p(\mathcal{D} \mid \Pi, \mathrm{M}, \Sigma) = \prod_{i=1}^{n} q\left(\boldsymbol{x}_i \mid \Pi, \mathrm{M}, \Sigma\right)$$

For mixing weights Π, the symmetric Dirichlet distribution (Ng et al. (2011); Bela et al. (2010) ; Good et al. (1976)) is considered as the prior probability[2]:

$$p\left(\Pi; \alpha_0\right) = \mathrm{Dir}\left(\boldsymbol{\Pi}; \alpha_0\right) \propto \prod_{\xi=1}^{\Xi} \pi_{\xi}^{\alpha_0 - 1}$$

where $\mathrm{Dir}(\boldsymbol{\Pi}; \alpha)$ denotes the symmetric Dirichlet density with concentration parameter α [Orhan (2012a), Orhan (2012b)]. For Gaussian expectations M and Gaussian precision matrices Σ, the product of the normal distribution and the Wishart distribution (Srivastava (1965); Haff (1979)), called the normal-Wishart distribution, is considered as the prior probability:

$$\begin{aligned} p\left(\mathrm{M}, \Sigma; \beta_0, \boldsymbol{\Psi}_0, \nu_0\right) &= \prod_{\xi=1}^{\Xi} N\left(\boldsymbol{\mu}_{\xi} \mid \mathbf{0}, \left(\beta_0 \boldsymbol{\Sigma}_{\xi}\right)^{-1}\right) \text{Wishart}\left(\boldsymbol{\Sigma}_{\xi}; \boldsymbol{\Psi}_0, \nu_0\right) \\ &\propto \prod_{\xi=1}^{\Xi} \det\left(\boldsymbol{\Sigma}_{\xi}\right)^{\frac{\nu_0 - d}{2} - 1} \exp\left(-\frac{\beta_0}{2} \boldsymbol{\mu}_{\xi} \boldsymbol{\Sigma}_{\xi} \boldsymbol{\mu}_{\xi} - \frac{1}{2} \operatorname{tr}\left(\boldsymbol{\Psi}_0^{-1} \boldsymbol{\Sigma}_{\xi}\right)\right) \end{aligned}$$

where $\text{Wishart}(\boldsymbol{\Sigma}; \boldsymbol{\Psi}, \nu)$ denotes the Wishart density with ν degrees of freedom:

$$\text{Wishart}(\boldsymbol{\Sigma}; \boldsymbol{\Psi}, \nu) = \frac{\det(\boldsymbol{\Sigma})^{\frac{\nu - d - 1}{2}} \exp\left(-\frac{1}{2} \operatorname{tr}\left(\boldsymbol{\Psi}^{-1} \boldsymbol{\Sigma}\right)\right)}{\det(2\boldsymbol{\Psi})^{\frac{\nu}{2}} \Gamma_d\left(\frac{\nu}{2}\right)}$$

Here, d denotes the dimensionality of input $\boldsymbol{x}$ and $\Gamma_d(\cdot)$ denotes the d-dimensional gamma function defined by:

$$\Gamma_d\left(\frac{\nu}{2}\right) = \int_{S_d^+} \det(\boldsymbol{S})^{\frac{\nu - d - 1}{2}} \exp(-\operatorname{tr}(\boldsymbol{S})) \mathrm{d}\boldsymbol{S}$$

where $\mathbb{S}_d^+$ denotes the set of all $d \times d$ positive symmetric matrices. Note that the above normal-Wishart distribution is conjugate for the multivariate normal distribution with unknown expectation and unknown precision matrix (Sugiyama (2015);Hutter (2008)).

Proposition 1.1.1 (Hutter (2008) & Sugiyama (2015))
By the Bayes theorem, the posterior probability $p(\Pi, \mathrm{M}, \Sigma \mid \mathcal{D})$ *of the Dirichelet Mixture Model defined above is given as:*

$$p\left(\Pi, \mathrm{M}, \Sigma \mid \mathcal{D}; \alpha_0, \beta_0, \boldsymbol{\Psi}_0, \nu_0\right) = \frac{p(\mathcal{D} \mid \Pi, \mathrm{M}, \Sigma) p\left(\Pi; \alpha_0\right) p\left(\mathrm{M}, \Sigma; \beta_0, \boldsymbol{\Psi}_0, \nu_0\right)}{p\left(\mathcal{D}; \alpha_0, \beta_0, \boldsymbol{\Psi}_0, \nu_0\right)} \tag{1.7}$$

[2]Analogous to multinomial distribution to the binomial distribution, Dirichlet is the multinomial version for the beta distribution.

where $p\left(\mathcal{D}; \alpha_0, \beta_0, \mathbf{\Psi}_0, \nu_0\right)$ is the marginal probability of the data $\mathcal{D}$ itself with respect to (w.r.t) all other parameters.

The proof is trivial based on the Bayes theorem, and this posterior distribution is not computationally tractable (Sabourin & Naveau, 2014). We can have many practical approximation inference methods for this infinite mixture model: Markov chain sampling method (Neal, 2000), variational inference techniques Kurihara et al. (2007) for the Dirichlet process, etc. We will limit this presentation about infinite (and finite) Dirichlet Mixture Models to this simple introduction for the scope of this thesis.

Extension to other Dirichlet Mixture Models (DMM)

They are other types of Dirichlet Mixture Models (DMM) commonly known as Dirichlet Hidden Markov Mixture Models (DHMM) that you may want to investigate. They have four (3) principal characteristics :

1. The Dirichlet process is still a prior probability distribution on the mixture weights of the components;

2. Markov models – These are used to model sequences where the future state depends only on the current state and not any past states. (memoryless processes): the latent variable z follows a Markov model (Davis (2018), Fink (2014), Bahl et al. (1992), Fosler-Lussier (1998)).

3. Hidden Markov models – Used to model processes where the true state is unobserved (hidden) but there are observable factors that give us useful information to guess the true state: because the latent variable z is hidden for the observable $(x_i)_{1:N}$, the latter are said to follow a Hidden Markov Model (HMM) – (Brown et al. (1993); Vaičiulytė & Sakalauskas (2020); Beal et al. (2001); Chen et al. (2015b); Fuse & Kamiya (2017); Ko et al. (2015); Nasfi et al. (2020); Moon et al. (2010); Brown et al. (1993); Xu et al. (2008); Torbati & Picone (2015); Bastani et al. (2014)).

1.2 Major learning ingredients

1.2.1 Cholesky Decomposition

For symmetric, positive definite matrices, the Cholesky decomposition or Cholesky factorization is useful in practice to generate samples from a Gaussian-*like* distribution.

Theorem 1.2.1 (Cholesky Decomposition)
A symmetric, positive definite. matrix A can be factorized into a product $A = \mathcal{Z}\mathcal{Z}^\top$, described as:

$$\begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{nn} \end{bmatrix} = \begin{bmatrix} l_{11} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ l_{n1} & \cdots & l_{nn} \end{bmatrix} \begin{bmatrix} l_{11} & \cdots & l_{n1} \\ \vdots & \ddots & \vdots \\ 0 & \cdots & l_{nn} \end{bmatrix}$$

where $\mathcal{Z}$ is a lower triangular matrix with positive diagonal elements, $\mathcal{Z}$ is called the Cholesky factor of A, and $\mathcal{Z}$ is unique.

Application to Gaussian distribution sampling. The following example is from Williams & Rasmussen (2006). For a Gaussian variable $y \sim N(\eta, \kappa_0)$, the multivariate normal distribution has a joint probability density given by:

$$p\left(y \mid \eta, \kappa_0\right) = (2\pi)^{-d/2} \left|\kappa_0\right|^{-1/2} \exp\left(-\frac{1}{2}(y-\eta)^T \kappa_0^{-1}(y-\eta)\right)$$

where $\eta \in \mathbb{R}^d$ is the mean vector and $\kappa_0 \in M_d(\mathbb{R})$ is the (symmetric, positive definite) covariance matrix. To get some samples of y from this distribution, apply the following steps:

- Compute the Cholesky decomposition: we want to compute the Cholesky decomposition of the covariance matrix κ_0. That is, we want to find a lower triangular matrix $\mathcal{Z} \in M_d(\mathbb{R})$ such that $\kappa_0 = \mathcal{Z}\mathcal{Z}^T$. Matrix $\mathcal{Z}$ will be useful in a further step.
- Generate Independent Samples $v \sim N(0, I)$;
- Compute $y = \eta + \mathcal{Z}v$. The variable $y = \eta + \mathcal{Z}v$ has a multivariate normal distribution since is a linear combination of independent normally distributed variables. Moreover,

  $$E[y] = E[\eta + \mathcal{Z}v] = \eta + \mathcal{Z}E[v] = \eta$$

  and

  $$E\left[(y-\eta)^T (y-\eta)\right] = \kappa_0$$

1.2.2 Markov Random Fields

A graphical probabilistic model is a graphical representation used to expresses the conditional dependency between random variables. A graphical model has two components in it: vertices and edges. The vertices indicate the state of the random variables and the edge indicates the direction of connections (or transformations).

Markov Random Fields

A set of random variables having Markov property and described by an undirected graph is referred to as Markov Random Field (MRF) or Markov network. In other words, a random field is said to be a Markov random field if it satisfies Markov property.

Definition 1.2.1 (Simon (2011); Li (1994); Geman & Graffigne (1986); Spitzer (1971))
A Markov Random Field (MRF) is a graph $\mathcal{G} = (\mathcal{V}, \mathbb{\Lambda})$, where $\mathcal{V} = \{1, 2, \ldots, N\}$ is the set of nodes, each of which is associated with a random variable $(\mathrm{RV}), u_j$, for $j = 1 \ldots N$. $\mathbb{\Lambda}$ is the set of edges of the graph.

- *The neighbourhood of node i, denoted $\mathcal{N}_i$, is the set of nodes to which i is adjacent; i.e., $j \in \mathcal{N}_i$ if and only if $(i, j) \in \mathbb{\Lambda}$.*
- *The Markov Random field satisfies*

  $$p\left(u_i \mid \{u_j\}_{j \in \mathcal{V} \setminus i}\right) = p\left(u_i \mid \{u_j\}_{j \in \mathcal{N}_i}\right)$$

$\mathcal{N}_i$ is often called the Markov blanket of node i.

We denote the set of all cliques in the graph $\mathcal{G} = (\mathcal{V}, \mathbb{\Lambda})$ as C, with a clique $c_i \in C$ meaning any fully connected subset of the graph comprising its nodes. The joint probability for the Markov Random Field (MRF) variables $\mathbf{u} = \{u_1, \dots u_N\}$ is represented by a Gibbs Field (Cressie & Lele (1992)) and can be written as the product of clique potentials ϕ_i :

$$P(\mathbf{u}) = \frac{1}{Z} \prod_{c_i \in C} \phi_i (c_i) \tag{1.8}$$

with $\phi_i (c_i)$ the i th clique potential, a function only of the values of the members of the clique in c_i. Each potential function ϕ_i must be positive. A normalization constant Z is required in to create a valid probability distribution $Z = \sum_x \prod_{c \in C} \phi_i (c_i)$. Here follows an example from Bagnell (2020).

Clique Potentials as Energy Functions. Often, clique potentials of the form $\phi_i (c_i) = \exp(-f(c_i))$ are used, with $f_i (c_i)$ an energy function over values of c_i (Spitzer (1971); Sherman (1973); Preston (1973); Clifford (1990)). The energy assigned by the function $f_i (c_i)$ is an indicator of the likelihood of the corresponding relationships within the clique, with a higher energy configuration having lower probability and vice-versa. If this is the case, equation 1.8 can be written as:

$$P(\mathbf{x}) \quad = \quad \frac{1}{Z} \exp \left[- \sum_{c_i \in C} f_i (c_i) \right] \tag{1.9}$$

For example, we can write energy functions over the cliques in the graph from Figure 1.6.
Let $f_1 (\{x_1, x_2, x_3\}) = x_1^2 + (x_2 - 5x_3 - 3)^2$, and $f_2 (\{x_3, x_4, x_5\}) = (x_3 - x_4)^2 + (x_3 + x_4 + x_5)^2$

Then the joint probability can be written as:

$$P(\mathbf{x}) = \frac{1}{Z} \exp \left[- \left(x_1^2 + (x_2 - 5x_3 - 3)^2 \right) - \left((x_3 - x_4)^2 + (x_3 + x_4 + x_5)^2 \right) \right]$$

Gaussian Markov Random Fields

Definition 1.2.2 (Gaussian Markov random field)
Let the neighbours $\mathcal{N}_i$ to a point s_i be the points $\{s_j, j \in \mathcal{N}_i\}$ that are "close" to s_i. A Gaussian distribution $x = (x_1, x_2, ..., x_N) \sim N(\mu, \Sigma)$ that satisfies

$$p(x_i \mid \{x_j : j \neq i\}) = p(x_i \mid \{x_j : j \in \mathcal{N}_i\})$$

is a Gaussian Markov random field.

An example commonly used in practice is the Multivariate Gaussian (MRF) distribution defined on vector $w = (vec(w^1)^T, vec(w^2)^T, ..., vec(w^k)^T)^T$, with $i = 1, ..., k$, $w^i \in \mathbb{M}_{l_{i-1} \times l_i}$ (with $l_0 = q$),

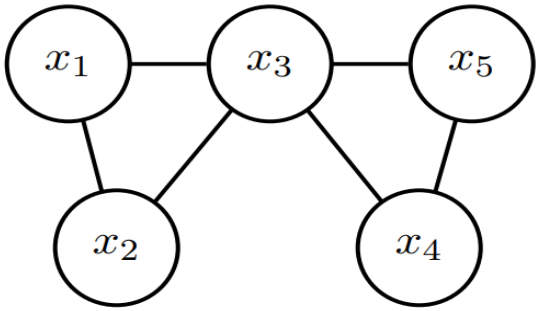


Figure 1.6: A simple Markov Random Field

$vec(w^i) \in \mathbb{R}_{l_{i-1} \times l_i}$, $\kappa_w = \sum_{i=1}^{k} l_{i-1} \times l_i$. The mean is $\mu_=(\mu_1^T, \mu_2^T, ..., \mu_k^T)$, let's say $\mu_k = \mathbb{E}[vec(w^k)]$), with precision matrix $\mathcal{Q}$, and it has the following density :

$$p(w|\mathcal{Q}) = (\frac{1}{2\pi})^{\kappa_w/2} det(\mathcal{Q})^{\frac{1}{2}} \exp(-\frac{1}{2}(w-\mu)^T \mathcal{Q}(w-\mu))$$

The precision matrix $\mathcal{Q}$ is sparse:

$$\mathcal{Q}_{ij} = 0 \text{ if } j \notin \mathcal{N}_i$$

The matrix block $\mathcal{Q}_{ij}$ is null if i and j are not neighbors, and we have:

$$\mathbb{E}[vec(w^i)|vec(w^{-i})] = \mu_i + \mathcal{Q}_{ii}^{-1} \sum_{j:j\sim i} \mathcal{Q}_{ij} vec(w^j)$$

where $\sim$ means *w.r.t.*[3] the connected weights, and w^{-i} is the set of all weights matrices without w^i.

The Ising Model

The Ising model is also a Markov random field having binary variables and pairwise potential functions. It has since been used in a broad range of application domains including economics, social networks, computer vision, biology, and signal processing to explain the phenomenon of phase transition (Brush (1967)). Progress has been made to understand its inference framework (computing the partition function and sampling) (Bresler (2015), Jerrum & Sinclair (1993), Sinclair et al. (2014), Goldberg et al. (2003)).

The Ising model is made of magnetic node with two states dipole moments: $+1$ or -1. It has a network graph with the following Hamiltonian of pair-wise interacting spins:

$$\mathcal{H}_{\text{Hamiltonian}}(v) = -\sum_i \mathcal{Q}_i v_i - \sum_{i,j} \mathcal{Q}_{i,j} v_i v_j$$

[3] with respect to.

where $(\mathcal{Q}_{i,j})_{ij}$ represent interactions, and $(\mathcal{Q}_i)_i$ (external) magnetic fields. This model is a Gibbs Field based on equation [1.9] when a Boltzmann distribution is applied to the Hamiltonian.

Note that, more broadly, terms of interaction $\mathcal{Q}_{i,j}$ can be more complex, namely, the following:

$$\mathcal{H}(v) = -\sum_i \mathcal{Q}_i v_i - \sum_{i,j} \mathcal{Q}_{i,j} v_i v_j - \sum_{i,j,k} \mathcal{Q}_{i,j,k} v_i v_j v_k - \cdots$$

One example of such a complex network is the Hinton restricted Boltzmann machine (RBM) which introduced the hidden layer of neurons which that dramatically improved the performance of the network for learning purposes (Salakhutdinov et al. (2007)).

1.2.3 Gradient, Stochastic Gradient and batch learning

Gradient Descent

As you might know, the gradient of a function at a point provides the direction of the steepest *ascent* of this function at the given point. The negative of the gradient then gives the direction of the steepest *descent*. Therefore, the direction of the steepest descent is given by the negative of the gradient. A gradient can therefore be ascent or descent. Both have the property of maximization (ascent) or minimization (descent) depending on the direction of the optimization problem.

Gradient Descent is an algorithm to minimize a function $J(\theta)$. The idea is that for the current value of θ, you calculate the gradient of $J(\theta)$, then take a small step in direction of negative gradient to minimize the function, and repeat until convergence. The update equation is as follow:

$$\theta^{\text{new}} = \theta^{\text{old}} - \epsilon \nabla_\theta J(\theta) \tag{1.10}$$

Where ϵ is the step size or the learning rate. But the problem is that $J(\theta)$ might be very expensive to compute sometimes. The inclusion of the word *stochastic* simply means that some random samples from the training data are chosen in each run to update the parameter during optimisation, within the framework of gradient descent. In other words, Stochastic Gradient Descent (SGD) repeatedly sample a set of data and update the parameter after each *random* selection.

> In Gradient Descent or Batch Gradient Descent, we use the whole training data per epoch whereas, in Stochastic Gradient Descent, we use only single training example per epoch and Mini-batch Gradient Descent lies in between of these two extremes, in which we can use a mini-batch (small portion) of training data per epoch [Look for Section 3.6 for more implementation details].

Stochastic gradient

The Stochastic gradient can be also ascent or descent. We will illustrate the standard properties of this algorithm using the descent configuration, because both (ascent & descent) have similar characteristics (or identical features).

The stochastic descent of the gradient (often shortened as SGD) is a stochastic approximation of the method of gradient descent to minimise an objective descent. A functionality that is written as a sum of distinguishable functions. The term stochastic here applies to the fact that we understand that we do not exactly know the gradient, but rather know a chaotic estimate of it instead.

This paragraph may require further readings about the Evidence Lower Bound (ELBO) in section 4.4 where more details have been proposed. Following Bottou (2012), the stochastic gradient descent algorithm replaces the gradient by an estimate :

$$\boldsymbol{\lambda}^{n+1} = \boldsymbol{\lambda}^{n} - \epsilon_n \boldsymbol{Z}\left(\boldsymbol{\lambda}^{n}; \boldsymbol{\xi}^{n}\right)$$

where we describe the estimate of the gradient by $Z\left(\boldsymbol{\lambda}^{n}; \boldsymbol{\xi}^{n}\right)$, with the optimized parameter λ, emphasizing the stochastic nature through the random vector $\boldsymbol{\xi}^{n}$. A class of possibilities are given by

$$Z\left(\boldsymbol{\lambda}^{n}; \boldsymbol{\xi}^{n}\right) = \left[\frac{1}{n_t} \sum_{i \in \mathcal{S}_t} \nabla ELBO_i\left(\boldsymbol{\lambda}^{n}\right)\right]$$

where $\mathcal{S}_t \subset \{1, \ldots, n_o\}$ and $n_t = |\mathcal{S}_t|$ gives the number of observations to base the estimate of the gradient on. In our case, we have set $n_t \leq 2$ enabling high-speed computation, but requiring many iterations ≥ 2. We will consider the following assumptions on $\{\epsilon_n\}$:

$$\begin{array}{ll} \epsilon_n > 0 & (A-1) \\ \sum_{n=1}^{\infty} \epsilon_n = \infty & (A-2) \\ \sum_{n=1}^{\infty} \epsilon_n^2 < \infty & (A-3) \end{array}$$

Theorem 1.2.2 (Quasimartingale convergence theorem (Robbins & Siegmund, 1971) and (Fisk, 1965))
If $(X_n)_{n=1}^{\infty}$ is a positive stochastic process, and

$$\sum_{n=1}^{\infty} \mathbb{E}\left[\left(\mathbb{E}\left[X_{n+1} \mid \mathcal{F}_n\right] - X_n\right) \mathbb{1}_{\{\mathbb{E}[X_{n+1}|\mathcal{F}_n] - X_n > 0\}}\right] < \infty$$

then $X_n \to X_\infty$ almost surely on a filtered probability space $\left(\Omega, \mathcal{F}, (\mathcal{F}_n)_{n=0}^{\infty}, \mathbb{P}\right)$, with

$$\mathcal{F}_n = \sigma\left(X_m \mid m \leq \hat{n}\right)$$

Now, let $C : \mathbb{R}^k \longrightarrow \mathbb{R}$ be differentiable. How do solutions $s : \mathbb{R} \to \mathbb{R}^k$ to the following Ordinary differential Equation (ODE) behave:

$$\frac{d}{dt} s(t) = -\nabla C(s(t)) \text{ or in a discretized form: } \frac{s_{n+1} - s_n}{\epsilon_n} = -\nabla C\left(s_n\right)$$

The discretization uses Forward Euler discretisation method (Villatoro & Ramos, 1999).

Proposition 1.2.1 ((Robbins & Monro, 1951) and (Bach, 2018))
If $s_{n+1} = s_n - \epsilon_n H_n\left(s_n\right)$, with $H_n\left(s_n\right)$ an unbiased estimator for $\nabla C\left(s_n\right)$, and C satisfies:

1. *C has a unique minimiser $x^\star$*
2. *$\forall \epsilon > 0, \inf_{\|x-x^\star\|_2^2 > \epsilon} \langle x - x^\star, \nabla C(x) \rangle > 0$,*
3. *$\mathbb{E}\left[\|H_n(x)\|_2^2\right] \leq A + B\|x - x^\star\|_2^2$ for some $A, B \geq 0$ independent of n,*

then subject to $(A-2)$ and $(A-3)$ we have $s_n \to x^\star$.

Proof. The proof structure broadly follows Bottou (1998) paper.

Step 1 **Define Lyapunov sequence**: $h_n = \|s_n - x^\star\|_2^2$. h_n is not guaranteed to be decreasing. The main Idea from (Bottou (1998)) is: h_n may fluctuate, but if we can show that the cumulative 'up' movements aren't too big, we can still prove convergence of h_n.

Step 2 **Consider the h_n variations**: $h_{n+1} - h_n$.

$$\begin{aligned} h_{n+1} - h_n &= \langle s_{n+1} - x^\star, s_{n+1} - x^\star \rangle - \langle s_n - x^\star, s_n - x^\star \rangle \\ &= \langle s_{n+1}, s_{n+1} \rangle - \langle s_n, s_n \rangle - 2\langle s_{n+1} - s_n, x^\star \rangle \\ &= \langle s_n - \epsilon_n H_n(s_n), s_n - \epsilon_n H_n(s_n) \rangle - \langle s_n, s_n \rangle + 2\epsilon_n \langle H_n(s_n), x^\star \rangle \\ &= -2\epsilon_n \langle s_n - x^\star, H_n(s_n) \rangle + \epsilon_n^2 \|H_n(s_n)\|_2^2 \end{aligned}$$

So

$$\mathbb{E}\left[h_{n+1} - h_n \mid \mathcal{F}_n\right] = -2\epsilon_n \langle s_n - x^\star, \nabla C(s_n) \rangle + \epsilon_n^2 \mathbb{E}\left[\|H_n(s_n)\|_2^2 \mid \mathcal{F}_n\right]$$

Step 3 **Show that** h_n converge almost surely:

Assuming $\mathbb{E}\left[\|H_n(x)\|_2^2\right] \leq A + B\|x - x^\star\|_2^2$, we get:

$$\begin{aligned} & h_{n+1} - h_n \leq -2\epsilon_n \langle s_n - x^\star, H_n(s_n) \rangle + \epsilon_n^2 (A + Bh_n) \\ \implies \quad & h_{n+1} - \left(1 + \epsilon_n^2 B\right) h_n \leq -2\epsilon_n \langle s_n - x^\star, H_n(s_n) \rangle + \epsilon_n^2 A \\ & \leq \epsilon_n^2 A \end{aligned}$$

Condition (2) simply states that the opposite of the gradient $-\nabla_x C(x)$ always points towards the minimum x^*. This is also a convexity criterion that ensures that the term $\langle s_n - x^\star, H_n(s_n) \rangle$ is always negative. We have:

$$\forall \epsilon > 0, \inf_{\|x-x^\star\|_2^2 > \epsilon} \langle x - x^\star, \nabla C(x) \rangle > 0 \implies \langle s_n - x^\star, H_n(s_n) \rangle > 0$$

So,we get:

$$h_{n+1} - \left(1 + \epsilon_n^2 B\right) h_n \leq \epsilon_n^2 A$$

Introduce the series $\mu_n = \prod_{i=1}^{n-1} \frac{1}{1+\epsilon_i^2 B} \overset{n\to\infty}{\longrightarrow} \mu_\infty$, and $h'_n = \mu_n h_n$ Get:

$$\mathbb{E}\left[h'_{n+1} - h'_n \mid \mathcal{F}_n\right] \leq \epsilon_n^2 \mu_n A$$
$$\Longrightarrow \mathbb{E}\left[\left(h'_{n+1} - h'_n\right) \mathbb{1}_{\mathbb{E}\left[h'_{n+1} - h'_n \mid \mathcal{F}_n\right] > 0} \mid \mathcal{F}_n\right] \leq \epsilon_n^2 \mu_n A$$

$$\left\{ \begin{array}{l} \mu_n \leq \frac{1}{1+\epsilon_1^2 B} \\ \sum_{n=1}^{\infty} \epsilon_n^2 < \infty \end{array} \right. \Longrightarrow \text{[Quasimartingale convergence] } \textbf{Theorem 1.} \Longrightarrow (h'_n)_{n=1}^{\infty} \text{ converges a.s.}$$
$$\Longrightarrow (h_n)_{n=1}^{\infty} \text{ converges a.s .}$$

Step 4 **Show that h_n must converge to 0**:

From previous calculations:

$$\mathbb{E}\left[h_{n+1} - \left(1 + \epsilon_n^2 B\right) h_n \mid \mathcal{F}_n\right] = -2\epsilon_n \left\langle s_n - x^\star, \nabla C\left(s_n\right)\right\rangle + \epsilon_n^2 A$$

$(h_n)_{n=1}^{\infty}$ converges, so the sequence in the first member of the equation above is summable a.s. . Because $\sum_{n=1}^{\infty} \epsilon_n^2 < \infty$, so right term $(\epsilon_n^2 A)$ is summable a.s., so the left term side is also summable a.s. :

$\sum_{n=1}^{\infty} \epsilon_n \left\langle s_n - x^\star, \nabla C\left(s_n\right)\right\rangle < \infty$ almost surely, and $\epsilon_n \left\langle s_n - x^\star, \nabla C\left(s_n\right)\right\rangle \longrightarrow 0$ almost surely

We can conclude that $(h_n)_{n=1}^{\infty}$ converge to zero, and this forces also to have: $\left\langle s_n - x^\star, \nabla C\left(s_n\right)\right\rangle \to 0$ almost surely .

We can reach this conclusion because we know that h_n converges. Reasoning by contradiction: let us assume that $h_n = \|s_n - x^\star\|_2^2$ converges to a value greater than zero and therefore, after a certain time, remains greater than some $\epsilon > 0$. Assumption (2) implies that $\left\langle s_n - x^\star, \nabla C\left(s_n\right)\right\rangle > 0$, and then it remains greater than a strictly positive quantity. Since this would cause the sum $(\sum_{n=1}^{\infty} \epsilon_n \left\langle s_n - x^\star, \nabla C\left(s_n\right)\right\rangle)$ to diverge (but, this is not the case: $\sum_{n=1}^{\infty} \epsilon_n \left\langle s_n - x^\star, \nabla C\left(s_n\right)\right\rangle < \infty$), we can conclude on that h_n converges to zero. We get by then: $s_n \to x^\star$ (and simultaneously $\left\langle s_n - x^\star, \nabla C\left(s_n\right)\right\rangle \to 0$), which ends the proof.■

■

Application on our stochastic gradient algorithm: If C is the ELBO function [Section 4.4] averaged across a data set, the true gradient is of the form:

$$\nabla C(\lambda^t) = \frac{1}{N} \sum_{i \in \mathcal{S}_t} \nabla ELBO_i\left(\boldsymbol{\lambda}^n\right)$$

and an approximation is formed by subsampling:

$$\widehat{\nabla C}(\lambda^n) = \frac{1}{n_t} \sum_{i \in \mathcal{S}_t} \nabla ELBO_i\left(\boldsymbol{\lambda}^n\right) \quad \left(S_t \sim \text{Uniforme(subsets of size } \mathrm{n}_t)\right)$$

The only difference between the Stochastic Gradient Ascent (SGA) algorithm versus the Stochastic Gradient Descent (SGD), is that we want to maximize the ELBO function. For that purpose, we simply reformat maximizing the ELBO as minimizing its negative to apply the convergence theorem. In our case:

i. The learning rate vector $\alpha_t = (\alpha_t^1, \alpha_t^2, \alpha_t^3)$ defined in section **7.1** (α_t^1 for the weights, α_t^2 for the biases and α_t^3 for the covariance matrix Σ) is set to be decreasing to make conditions [$(\mathrm{A}-1), (\mathrm{A}-2)$, and $(\mathrm{A}-3)$] hold in our experiments :

$$\alpha_t^1 = \alpha_t^2 = \alpha_t^3 = \frac{10^{-50}}{t}$$

ii. The Evidence Lower Bound (ELBO) is continuous and differentiable in every point, and is bounded by $\log p(y)$ (Yang, 2017). So in Proposition [see 1.2.1], the condition (3) always holds for the ELBO. But, it is not the case for condition (1) and (2) because of non-convexity of the ELBO. In their seminal paper from 1951, Robbins and Monro showed that the stochastic optimization will converge to a local optimum in our case (Robbins & Monro (1951)). So, it requires good initialization.

In practice, the quality of the approximation depends on the variance of the estimator of $Z\left(\boldsymbol{\lambda}^n; \boldsymbol{\xi}^n\right)$ (Johnson & Zhang, 2013). The main advantage of this algorithm is that one can even estimate the model with a cluster of small size, and still get good estimations. The Robbins-Siegmund theorem (Robbins & Siegmund, 1971) provides the means to establish almost sure convergence under conditions including $(A-2)$, also in cases where the loss function is non-smooth (Saad, 1998).

The size of the subset used to measure the gradient can be considered in the same manner as we think of *sample size* in simple estimation problem. Big mini-batch sizes can have reliable gradient forecasts, reducing the parameters update variances. Small mini-batches, by comparison, are easy to estimate.

1.2.4 Notes on Divergence Metrics for Distributions

In statistics, the principle of divergence in machine learning goes back to the term *entropy*, defined as *amount of information*. In practice, for a random variable Y, with $\{y_1, ..., y_n\}$ possible values, we can define entropy as the expected value of information $\boldsymbol{\mathcal{A}}(Y)$ computed as follows (MacKay (2003); Bishop (2006a)):

$$\boldsymbol{\mathcal{A}}(Y) = E[I(Y)] = E[-\log(p(Y))] = -\sum_{i=1}^{n} P\left(y_i\right) \log P\left(y_i\right)$$

where $I(Y)$ is the *Fisher* information of Y.

There is another advanced feature for entropy called *cross-entropy*, applied to measure the distance between two probability distributions p and q, denoted as $\boldsymbol{\mathcal{A}}(p, q)$. The cross-entropy is defined as :

$$\boldsymbol{\mathcal{A}}(p, q) \quad = \quad \boldsymbol{\mathcal{A}}(p) + D_{KL}(p \| q) \tag{1.11}$$

$$D_{KL}(p \| q) \quad = \quad KL(p(y), q(y)) = E_p(y) \left\{ \log \left(\frac{p(y)}{q(y)} \right) \right\} \tag{1.12}$$

where $D_{KL}(p\|q)$ is the Kullback-Leibler Divergence (KL) [Kullback & Leibler (1951)] between p and q, sometimes denoted $KL(p\|q)$ as well in the literature. From a **Bayesian** perspective, the KL divergence is the information obtained as we switch from a previous distribution Q to a posterior distribution P, and when the gain is null or negligible, it is equal to zero (0).

From a **Bayesian** perspective, the KL divergence is the information gained when we move from a prior distribution Q to a posterior distribution P, and is equal to zero (0), when the gain is null, or negligible.

Let's dive deep into the KL divergence

The following lines explain some properties of the KL divergence and its forward and reverse form respectively.

$D_{KL}(p\|q) \in [0, \infty]$, and is not symmetric (neither a distance metric, because it does not satisfy the triangle inequality), that is (Amari, 2007):

$$D_{KL}(p\|q) \neq D_{KL}(q\|p)$$

As a consequence, we have an option between two possible targets to refine when attempting to approximate p with another $q_{\boldsymbol{\theta}}$ distribution.

1. Optimize $\arg\min_\theta D_{KL}\left(p\|q_{\boldsymbol{\theta}}\right)$ called ***the forward KL***;
2. Optimize $\arg\min_\theta D_{KL}\left(q_{\boldsymbol{\theta}}\|p\right)$ called ***the reverse KL***.

The two opposite targets, as it turns out, really trigger different kinds of approximations. It is also not possible to do both in reality, because you are constrained by the topic you are attempting to fix, and the optimization facilities you have. A good resource is this paper from Huszár (2015) where they go more in detail investigating this.

Also, the KL divergence is reduced when p and q are the same almost everywhere (a.e) :

$$D_{KL}(p\|q) = 0 \iff p \overset{a.e.}{=} q$$

Finally, the p support has to exchange points in the q support in order for the KL divergence to be finite, whether it is the forward or the reverse KL. In the case of forward KL as for example, if a point y exists with $q(y) = 0$ but $p(y) > 0$, then $D_{KL}(p\|q) = \infty$.

A matter of smoothness and symmetry: the Jensen-Shannon Divergence

The comparability of two distributions is also calculated by the Jensen-Shannon Divergence (JSD). There is no symmetrical Kullback-Leibler Divergence (KLD), and this is where the JSD comes to the rescue. The JSD variant of the Kullback-Liebler Divergence, or $KL(p, q)$, is symmetrical and smooth.

Assuming the Kullback-Leibler Divergence between $p(y)$ and $q(y)$ in 1.12, the Jensen-Shannon Divergence (JSD) symmetrizes and smooths $KL(p(y), q(y))$ by:

$$JSD(p(y), q(y)) = \frac{1}{2}D\left(p(y), \frac{1}{2}p(y) + q(y)\right) + \frac{1}{2}D\left(q(y), \frac{1}{2}p(y) + q(y)\right)$$

For further research explorations, one can inspect other divergence metrics as well. The scope is not limited, as for example, the square root of the Jensen-Shannon divergence (JSD) (Fuglede & Topsoe (2004)) presents interesting properties. As it satisfies metric axioms between any two probability densities, and represents another difference by itself, including several variations of Jensen-Shannon metrics (Yamano, 2019).

Historically introduced in Wong & You (1985) work, the Jensen-Shannon divergence can be interpreted as the total KL divergence to the average distribution $\frac{p+q}{2}$. His implementation is heavy; depending on the application, the Kullback-Leibler divergence is always prefered, like in *Variational Bayesian Inference* Blei et al. (2017) for example.

1.2.5 Never forget Bayes if your frequent likelihood get hard...

Haugh (2017) has inspired me to add some notes about the possibility to switch from classical/frequentist likelihood to Bayesian learning whenever the time, because of Bernstein-von Mises theorem of asymptotic equivalence between Bayesian estimation, and Maximum Likelihood Estimation (MLE) under suitable assumptions.

Let Φ be our parameter of interest. We assume Φ is a variable with $\pi(\Phi)$ as its distribution. There is also a random vector, Y. with likelihood $p(\mathbf{y} \mid \mathbf{\Phi})$. Both have a joint distribution presented as follows:

$$p(\mathbf{\Phi}, \mathbf{y}) = \pi(\mathbf{\Phi}) p(\mathbf{y} \mid \mathbf{\Phi})$$

An integral upon all possible values of Φ lead to the marginal distribution of Y:

$$p(\mathbf{y}) = \int_{\Phi} \pi(\mathbf{\Phi}) p(\mathbf{y} \mid \mathbf{\Phi}) d\mathbf{\Phi}$$

Using Bayes theorem, this gives rise to the posterior distribution $\pi(\mathbf{\Phi} \mid \mathbf{y})$ of Φ:

$$\pi(\mathbf{\Phi} \mid \mathbf{y}) = \frac{\pi(\mathbf{\Phi}) p(\mathbf{y} \mid \mathbf{\Phi})}{p(\mathbf{y})} = \frac{\pi(\mathbf{\Phi}) p(\mathbf{y} \mid \mathbf{\Phi})}{\int_{\mathbf{\Phi}} \pi(\mathbf{\Phi}) p(\mathbf{y} \mid \mathbf{\Phi}) d\mathbf{\Phi}} \propto \pi(\mathbf{\Phi}) p(\mathbf{y} \mid \mathbf{\Phi})$$

Theorem 1.2.3 (Bernstein-von Mises)
Under suitable assumptions and for sufficiently large sample sizes, the posterior distribution of Φ *is approximately normal with mean equal to the true value of* Φ *and variance equal to the inverse of the Fisher information matrix (Le Cam (1986)).*

The Bernstein-von Mises theorem means that asymptotically, there exist identical distributional sample properties for Bayesian and MLE estimators, which is, in fact, a normal distribution.

Chapter 2

The Potts Model with Complete Shrinkage

2.1 The Potts Clustering

The Potts Clustering is a random partition model for clustering with the prior distribution on partitions ρ_n being the Potts Model.

Let $\mathfrak{D} = \{x_i \in \mathbf{R}^p, i = 1, ..., n\}$ be our data, i.e, the observations form the vertices of a graph (as in the super-paramagnetic clustering framework). Let us denote this data graph by $(\mathfrak{G}(\mathfrak{D}), \mathbb{\Lambda}(\mathfrak{G}))$, where $\mathbb{\Lambda}(\mathfrak{G})$, the edge-set is composed of pairs of nearest neighbors, i.e:

$$\mathbb{\Lambda}(\mathfrak{G}) = \{(x, y) \in \mathfrak{D}^2, k_{xy}(\sigma) = k_\sigma(x, y) > 0\}$$

where k_{xy} is said to be the similarities between the neighboring points x and y (k_{xy} a Mercel kernel in (x, y) given a bandwidth parameter σ). If x and y are neighboring points, we will write $x \sim y$.

In Potts clustering, we assign labels $i \in \{1, ..., q\}$ to each observation $x_i, i = 1, ..., n$, so that observations similar to each other are likely to be assigned the same label. Denoting $z_{si} = 1$ if x_i has been assigned to the sth label, and zero, otherwise, the model density is given by:

$$p(\{z_{si}\}|\sigma, X, \beta, q) = Z^{-1} \exp\Big\{ -\beta \sum_{x_i \sim x_j} k_\sigma(x_i, x_j)(1 - \delta(x_i, x_j))\Big\}$$

where $\beta = \frac{1}{T}$ is the inverse temperature parameter, $\delta(x_i, x_j) = \sum_{s=1}^{q} z_{si} z_{sj} = 1$ if x_i and x_j have the same label assignment.

2.1.1 The Bernouilli bonds

Let's introduce *percolation* (Duminil-Copin, 2016).

Definition 2.1.1 (Percolation configuration)
A percolation configuration $b = (b_{ij} : (x_i, x_j) \in \mathbb{\Lambda}(\mathfrak{G}))$ is an element of $\{0, 1\}^{\mathbb{\Lambda}(\mathfrak{G})}$. If $b_{ij} = 1$, the edge (x_i, x_j) is said to be frozen (open), otherwise (x_i, x_j) is said to be not frozen (closed).

A percolation model is given by a family of probability measures on percolation configurations. Giving The Kastaleyn-Fortuin mapping, which establishes a connection between a particular percolation model and a limit of the Potts model (Hu, 1987), Bernouilli bonds are introduced to match the percolation configuration for the Potts clustering. For an edge $(x_i, x_j) \in \mathbb{A}(\mathcal{G})$, the bond b_{ij} becomes frozen with conditional probability (given δ_{ij}):

$$p_{ij} = p(b_i j|\delta_{ij}) = \delta_{ij}\big(1 - \exp\{-\beta k_{ij}(\sigma)\}\big)$$

where $\delta_{ij} = \delta(x_i, x_j)$ and $k_{ij}(\sigma) = k_\sigma(x_i, x_j)$.

This is known as the Fortuin-Kasteleyn-Swendsen-Wang model (Sokal, 1997a).

Maximal connected components (clusters) are obtained by finding all observations in a frozen path (Murua & Quintana, 2017b),i.e, each cluster is then identified by those unique observations among a given frozen path. Isolated observations form a cluster of size equal to 1. This is a major drawback, because in real applications (biomedical datasets (Hu et al., 2018a), finance, computer science, engineering (Ganganath et al., 2014)), it is often essential to obtain clusters of sufficient sample size to make the clustering result meaningful and interpretable for subsequent analysis.

Because the bonds probabilities are conditioned by the label assignment process, it is obvious that the clustering is influenced by q, the number of labels. The larger q is, the more subsets (of data) of unique label are generated at random.

There is a need to control even the mean size of all subsets (of data) of unique label, after **initial label assignment**. We only care about the initial labelling because for further steps (in the case of Swenden-Wang algorithm) [Borgs et al. (2012); Salas & Sokal (1997); Häggkvist et al. (2004); Häggström et al. (2002); Johansson & Pistol (2011); Martinelli et al. (1990); Ding & Barbu (2015); Häggkvist et al. (2004)], each connected subset is assigned the same color label uniformly at random and independently from each other, given the bonds $\{b_{ij}\}$.

2.2 Notes on Standard Application: Random Partitions Models

It is well-known that a random measure in Bayesian non-parametrics induces a distribution over random partitions. Many Random partition models do exist with multiple applications (Dahl et al. (2017); Dahl (2008); Loschi & Cruz (2005); Betancourt et al. (2020); Di Benedetto et al. (2017); McCullagh (2011); Zanella et al. (2015); Stam (1983)). Some random partitions are implied by the Dirichlet process (DP) prior $p(\pi_n)$ (Blackwell et al. (1973); Ferguson (1973); Antoniak (1974); Müller & Quintana (2010a)). The most famous random partitions model, is the one of Müller & Quintana (2010a), which introduced a *cohesion* measure :

$$P(\pi_n = \{S_1, \ldots, S_{\Xi}\}) = K \prod_{\xi=1}^{\Xi} c(S_\xi) \tag{2.1}$$

where π_n is a partition of the objects in a family of subsets $S_1, S_2, \ldots, S_{\Xi}$ of $S_0 = \{1, 2, \ldots, n\}$ and $c(S)$

is a non-negative cohesion that is specified for each subset of S_0, $\boldsymbol{\Xi} = |\rho|$ is the number of partitions. Here, the normalizing constant $K = \sum_{\rho \in \mathcal{P}} \Pi_{\boldsymbol{\xi}=1}^{\boldsymbol{\Xi}} c\left(S_{\boldsymbol{\xi}}\right)$, where $\mathcal{P}$ is the set of all possible partitions into non-empty sets.

> As a reminder : Cohesion is the measure of the strength of the functional relationship of the elements in each subset that then controls the partition of subsets that can be roughly thought of as a probability (Page et al. (2019); Müller et al. (2013)).

A popular choice is $c(S) = m(|S| - 1)!$ where m is a precision parameter and $|S|$ is the number of elements in S. It follows that the resulting probability model for π_n is

$$P\left(\pi_n\right) = \frac{m^{\boldsymbol{\Xi}-1} \prod_{\boldsymbol{\xi}=1}^{\boldsymbol{\Xi}} \left(n_\xi - 1\right)!}{\prod_{i=1}^{n}(m+i-1)}$$

where $n_\xi = |S_\xi|$ is the number of elements in cluster j that is known as the Dirichlet process (DP) random partition.

Remark (Dahl et al. (2009))
The connection between product partition models and Dirichlet process mixture (DPM) models was first shown by Quintana & Iglesias (2003). The proof is obvious. take the equation 2.1, and replace $c(S) = m(|S| - 1)!$*, we get :*

$$P\left(\pi_n = \{S_1, \ldots, S_{\boldsymbol{\xi}}\}\right) = K \prod_{\boldsymbol{\xi}=1}^{\boldsymbol{\Xi}} m(|S_{\boldsymbol{\xi}}| - 1)! = m^{\boldsymbol{\Xi}} \cdot K \prod_{\boldsymbol{\xi}=1}^{\boldsymbol{\Xi}} (|S_{\boldsymbol{\xi}}| - 1)!$$

And it is easy to find the right K*, that will make* $P\left(\rho_n = \{S_1, \ldots, S_{\boldsymbol{\Xi}}\}\right)$ *a probability:*

$$K = \prod_{i=1}^{n} (m + i - 1)$$

.

Among many related random partition models, we have :

1. Product partition models (PPM) [Hartigan (1990); Barry & Hartigan (1992); Dahl et al. (2009); Loschi & Cruz (2005); Loschi & Cruz (2002); Quintana & Iglesias (2003)] is a special case. These model assume that observations in different elements of a random partition of the data are independent. So if the probability distribution for the random partitions is in a product form prior to obtaining observations, it is also then in product form after obtaining the observations (Jordan et al. (2007)).

   Definition 2.2.1 (Product Partition Model)
   Together with independent sampling across clusters, a $\boldsymbol{PPM}$ *can be described as (Quintana & Iglesias (2003); Blackwell et al. (1973); Pitman (1996); Dahl et al. (2009)):*

$$\mathcal{P}\left(\mathbf{y} \mid \pi_n\right) = \{S_1, \ldots, S_{\boldsymbol{\Xi}}\}) \propto \prod_{\boldsymbol{\xi}=1}^{\boldsymbol{\Xi}} c\left(S_\xi\right) P\left(\mathbf{y}_s\right)$$

2. Product partition models with a covariate-dependent extension (PPMx) proposed by Müller et al. (2011a), Dunson & Park (2008), and Dahl(2008). This PPM version uses covariates approach, with preditors dependent random probability measures. In this application, covariates are available and are used to a *priori* inform the clustering. This leads to random clustering models indexed by covariates, i.e., regression models with the outcome being a partition of the experimental units. There are many PPMx variants : Fung (2012), Quintana (2010); Quintana et al. (2020); Blei & Frazier (2011); Jo et al. (2015); Barcella et al. (2017); Ferreira et al. (2014); Page & Quintana (2018). One standard application of the Potts Model, is in fact to build random clustering covariates model with regression. The main example is the model of Murua & Quintana (2017b). We introduce briefly the scope of their model: suppose we have a set of n data available, and that each individual $i \in [n]$ in a given sample is associated with a p -dimensional vector y_i of responses of interest and a q-dimensional covariate vector x_i. Suppose also that the interest lies in studying the relationship between y_i and x_i, and in particular, in predicting the response y_{n+1} associated with a covariate vector x_{n+1} of a future individual. Let $p\left(y_i \mid x_i, \Phi_i\right)$ be a likelihood model stating the relationship between the i th response and the associated covariate vector. The covariate dependent random partition model with a hierarchical structure for these data is as follow:

$$
\begin{gathered}
y_1, \ldots, y_n \mid \rho_n, \boldsymbol{\Phi}_1^*, \ldots, \boldsymbol{\Phi}_{k_n}^* \overset{\text{ind}}{\sim} p\left(\boldsymbol{y}_i \mid \boldsymbol{x}_i, \boldsymbol{\Phi}_{s_i}^*\right) \\
\boldsymbol{\Phi}_1^*, \ldots, \boldsymbol{\Phi}_{k_n}^* \overset{\text{iid.}}{\sim} p(\boldsymbol{\Phi}) \text{ and } \rho_n \sim p\left(\rho_n \mid \boldsymbol{x}^n\right)
\end{gathered}
$$

Here ρ_n is a partition of $[n]$ into k_n subsets. Also, $s_1, \ldots, s_n$ are cluster membership indicators such that $s_i = j$ if the i th individual belongs to the j th cluster. In addition $\Phi_i = \boldsymbol{\Phi}_{s_i}^*$ for all $i \in [n]$. Model (1) groups in cluster j those individuals having identical parameter value Φ_j^*. Individuals within this cluster are conditionally iid given Φ_j^*. What make the model particular here is that $\rho_n \sim p\left(\rho_n \mid \boldsymbol{x}^n\right)$ is a Potts Model. This implies the existence of auxiliary binary variables, the so-called bonds $\boldsymbol{b} = \{b_{ij}\}$, so that:

$$
p\left(\rho_n \mid x^n\right) = \sum_{\boldsymbol{b} \Rightarrow \rho_n} p\left(\boldsymbol{b} \mid \boldsymbol{x}^n\right)
$$

They then apply the Metropolis-Hastings (MH) algorithm to sample parameters from this posterior, by choosing an efficient MH proposal distribution and they obtain consistent improvements compared to the results found in the literature. The model simultaneously allows for explicit estimation of the number of clusters, and for good responses predictions (Murua & Quintana (2017b)).

2.3 The Potts Clustering Model with Complete Shrinkage

One of the difficulties encountered when sampling partitions with the Swenden-Wang algorithm Swendsen & Wang (1987) for the random bond Potts Models as done by Murua & Quintana (2017a), is the distribution of the temperature T of the Potts Model in its probabilistic form as a system of particles (data points) and their interactions given by the similarity measure. The distribution of the system depends on the temperature T. For each T there is a probability $p_T\left(\{z_{kl}\}\right)$ associated with each configuration of the system's labels:

$$p_T\left(\{z_{ki}\}\right) \propto \exp\left\{-\frac{1}{T}H\left(\{z_{ki}\}\right)\right\} = \exp\left\{-\frac{1}{2T}\sum_{i=1}^{n}\sum_{j=1}^{n}\left(1-\delta_{ij}\right)s\left(x_i,x_j\right)\right\}$$

where $\delta_{ij} = \sum z_{kl}z_{kj}$ between observations i and j equals one if they are assigned to the same cluster k, and zero otherwise. $s\left(x_i,x_j\right)$ is the similarity measure, and finally $z_{ki} = 1$ if observation i belongs to cluster k. Then, as proposed by Murua & Quintana (2017a) (by introducing a set of latent variables, the bonds $\boldsymbol{b}$), the bond $b_{ij} = 1$ is said to be frozen if $b_{ij} = 1$ and $\alpha_{ij} = \delta_{ij} = 1$, that is, the points x_i and x_j are neighbors and have the same label. Otherwise, the bond is not frozen. The bond b_{ij} becomes frozen with probability $p_{ij} = 1 - \exp\left\{-\beta\kappa_{ij}(\sigma)\right\}$.

In our case, the similarities between pairs of covariate vectors are defined by $\kappa_{ij} = K\left(\boldsymbol{x}_i,\boldsymbol{x}_j\right)$ ($K(\cdot,\cdot)$ is a Mercel Kernel). As usual, we have assumed that K is a function of the distances $\left\|\boldsymbol{x}_i - \boldsymbol{x}_j\right\|$, of the form $\kappa_{ij} = \kappa_{ij}(\sigma) = K\left(\left\|x_i - x_j\right\|/\sigma\right)$, where $\sigma > 0$ is a bandwidth parameter.

But, the drawback of the bonds approach is the increasing number of small clusters generated in a given partition. To deal will it, we apply a modified agglomerative clustering approach (Kurita, 1991) by merging all small clusters of size $\leq \mathbf{h}$ with their closest cluster in terms of minimal distance respectively, where $\mathbf{h}$ is an integer greater or equal to 2. The algorithm uses a technique in which distances of all pairs of observations are stored. Then the nearest cluster (with size $\geq \mathbf{h}$) is given by the cluster with the closest node in terms of minimal distance to the cluster to be merged. This is what we call Potts Clustering with Complete Shrinkage (PCCS), and **this is the main contribution of this chapter**.

2.4 Effective Python Implementation

The fully implemented architecture and Valid Python Code of the Potts Clustering with Complete Shrinkage is available with comments on our github repository under GNU General Public license $\boldsymbol{v3.0}$. Please Send a request of access on the link : `https://github.com/kgalahassa/Potts_Clustering_Models_COmplete_Shrinkage` or write directly to alahassan@dms.umontreal.ca for more details.

The full descriptive code of the Potts Clustering with Complete Shrinkage (PCCS) requires at least 100 to 200 lines to present the complete algorithm; the reason why we prefer to (simply) distribute directly the code through github.

2.5 Experiments

We have performed experiments with ten (10) datasets taken from the multiple-output benchmark datasets available in the Mulan project website (Tsoumakas et al., 2020). The datasets are shown in Table 2.1.

With those data described above, we have generated a set of Potts partitions in a number b, with $b = 50, 100$, with a shrinkage constraint = 2 (a ***minimum*** of two (2) observations per cluster in each partition is required). We focus attention on four (4) characteristics that can help to compare the performance of the algorithm from one given dataset to one another:

Table 2.1: Summary of the ten (10) datasets taken from the Mulan project.

Dataset	Domain	Number of Instances	Number of features	Response variable dimension (targets)
Andromeda	Water	49	30	6
Slump	Concrete	103	7	3
EDM	Machining	154	16	2
ATP7D	Forecast	296	211	6
ATP1D	Forecast	337	411	6
Jura	Geology	359	15	3
Online sales	Forecast	639	401	12
ENB	Buildings	768	8	2
Water quality	Biology	1 060	14	16
SCPF	Forecast	1 137	23	3

- The number of data per cluster (NumbDC): this corresponds to the list of cluster size per partition;
- The Mean Number of data per cluster and per 100 individuals (MeanNumbC100): we take the mean of the list of the number of data per cluster, and divide by 100 [shown in table 2.2].
- Number of cluster per partition (NumbCPar): this corresponds to a list of the number of clusters per partition taken through all the partitions;
- The Mean Number of cluster per partition (MeanNumbCPar) [shown in table 2.2];

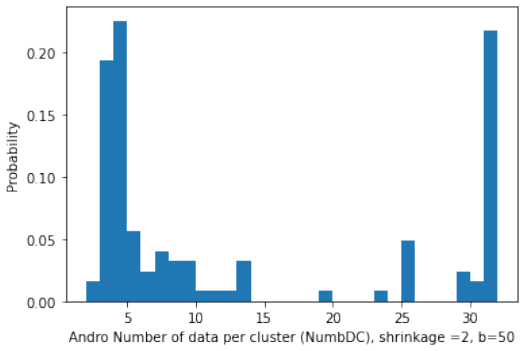
And finally a histogram to illustrate:

- The number of data per cluster (NumbDC);
- The number of cluster per partition (NumbCPar);

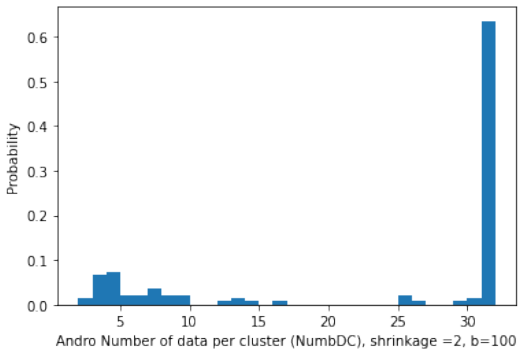
As for example, this histogram has Bell Shape Normal Curve for EDM number of cluster per partition [$b = 100$, shrinkage $= 2$] (figure 2.8) for Water Quality number of cluster per partition [$b = 100$, shrinkage $= 2$](figure 2.20).

Table 2.2: Summary of clustering statistics for the Mulan Project Dataset.

	$b = 50$		$b = 100$	
Dataset	MeanNumbC100	MeanNumbCPar	MeanNumbC100	MeanNumbCPar
Andromeda	0.132	2.431	0.229	1.396
Slump	0.301	2.294	0.405	1.702
EDM	0.491	1.356	0.0490	1.356
ATP7D	296	211	1.123	1.356
ATP1D	0.918	1.653	1.361	1.653
Jura	0.775	3.098	0.881	2.723
Online sales	1.617	2.647	2.862	1.495
ENB	1.691	3.039	2.495	2.059
Water quality	0.056	126.196	0.057	125.495
SCPF	0.351	2.705	0.473	2.009

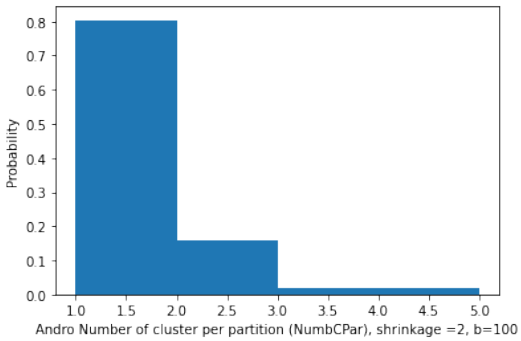


(a) $b = 50$

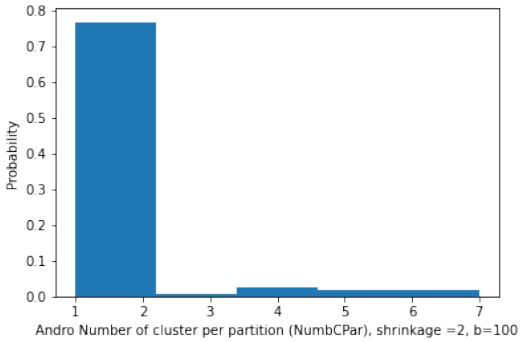


(b) $b = 100$

Figure 2.1: Andromeda, Number of data per cluster (NumbDC), shrinkage $= 2$

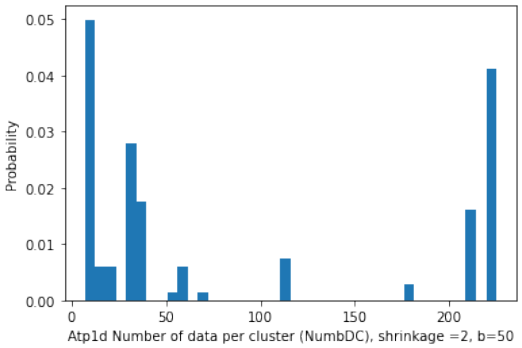


(a) $b = 50$

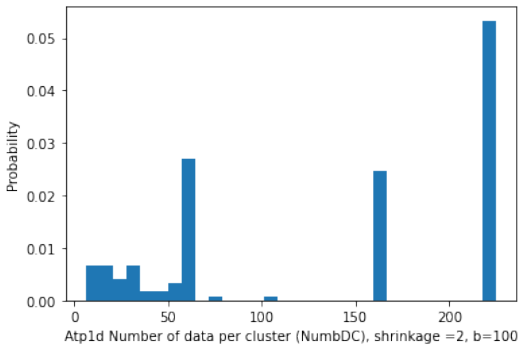


(b) $b = 100$

Figure 2.2: Andromeda, Number of cluster per partition (NumbCPar), shrinkage $= 2$

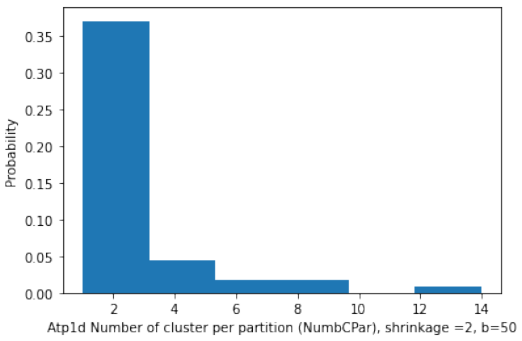


(a) $b = 50$

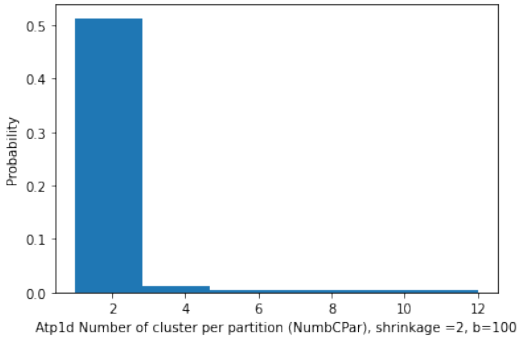


(b) $b = 100$

Figure 2.3: Atp1d, Number of data per cluster (NumbDC), shrinkage $= 2$

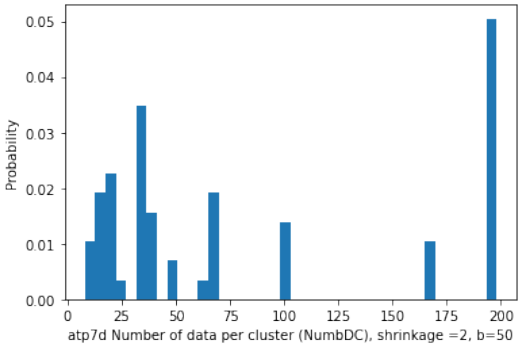


(a) $b = 50$

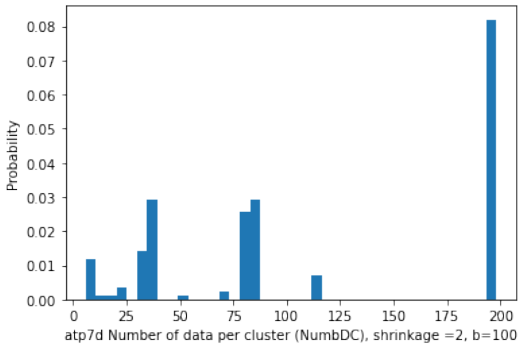


(b) $b = 100$

Figure 2.4: Atp1d, Number of cluster per partition (NumbCPar), shrinkage $= 2$

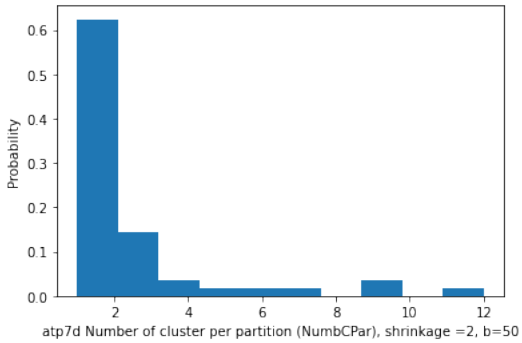


(a) $b = 50$

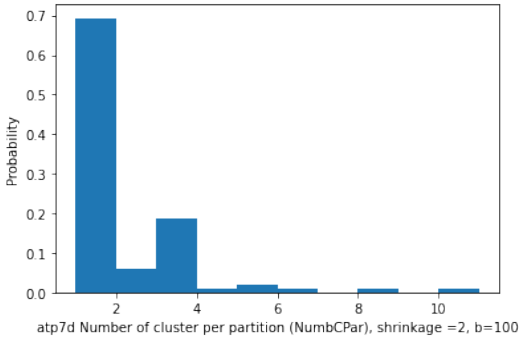


(b) $b = 100$

Figure 2.5: Atp7d, Number of data per cluster (NumbDC), shrinkage $= 2$

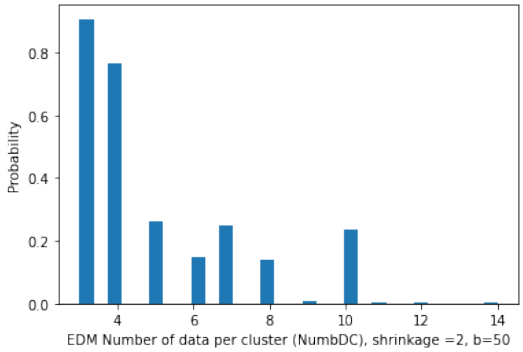


(a) $b = 50$

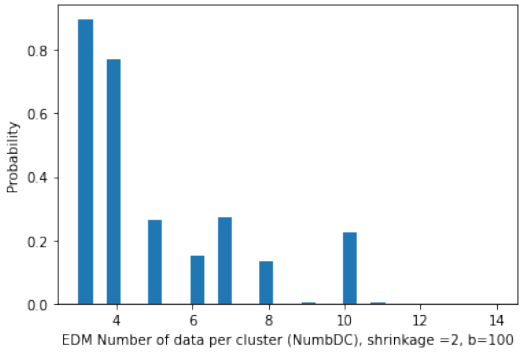


(b) $b = 100$

Figure 2.6: Atp1d, Number of cluster per partition (NumbCPar), shrinkage $= 2$

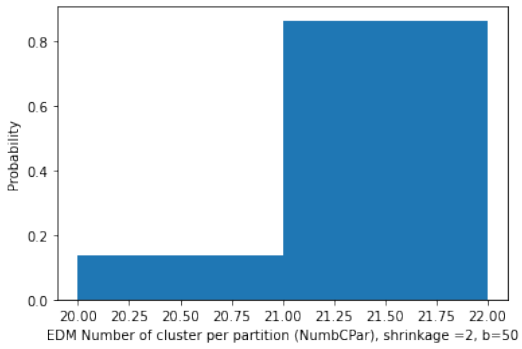


(a) $b = 50$

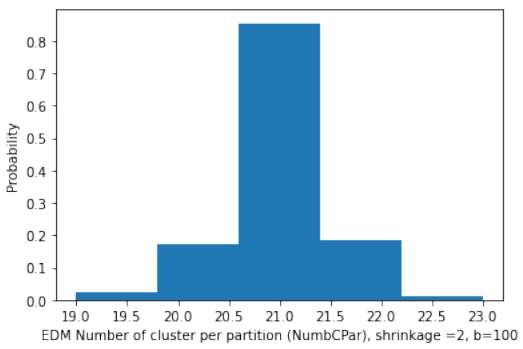


(b) $b = 100$

Figure 2.7: EDM, Number of data per cluster (NumbDC), shrinkage $= 2$

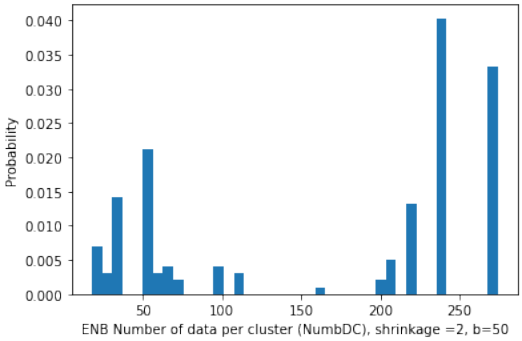


(a) $b = 50$

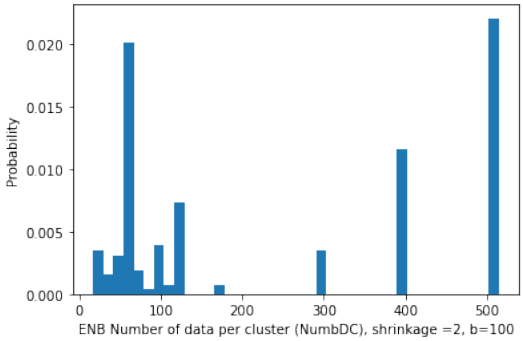


(b) $b = 100$

Figure 2.8: EDM, Number of cluster per partition (NumbCPar), shrinkage $= 2$

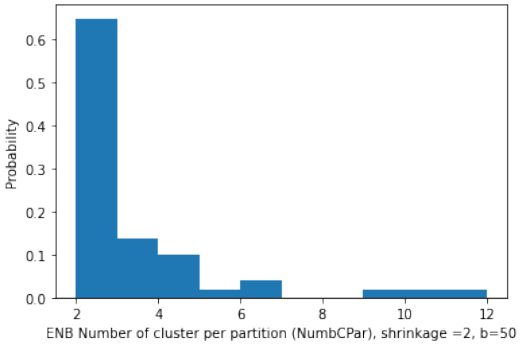


(a) $b = 50$

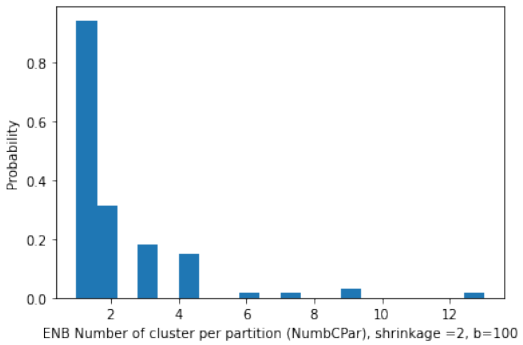


(b) $b = 100$

Figure 2.9: ENB, Number of data per cluster (NumbDC), shrinkage $= 2$

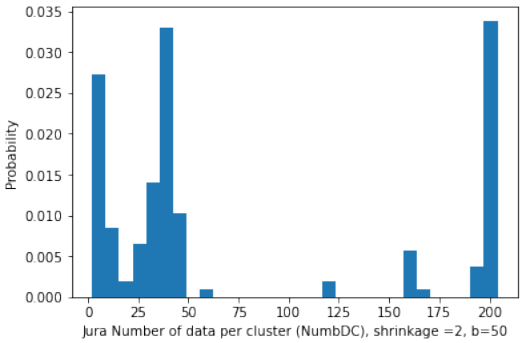


(a) $b = 50$

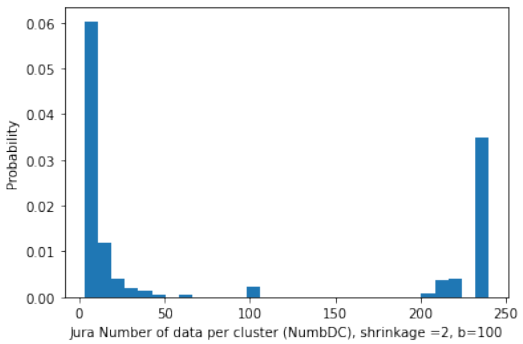


(b) $b = 100$

Figure 2.10: ENB, Number of cluster per partition (NumbCPar), shrinkage $= 2$

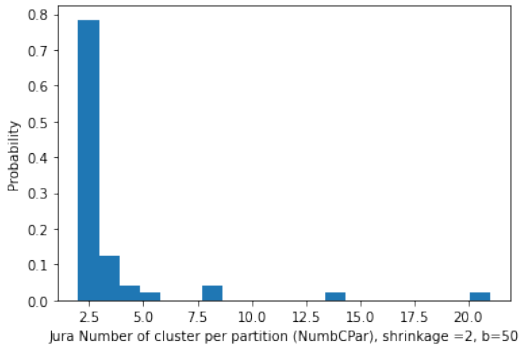


(a) $b = 50$

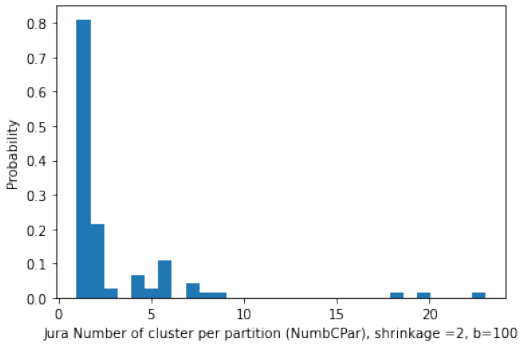


(b) $b = 100$

Figure 2.11: Jura, Number of data per cluster (NumbDC), shrinkage $= 2$

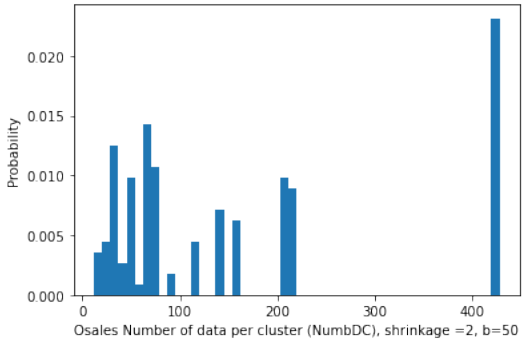


(a) $b = 50$

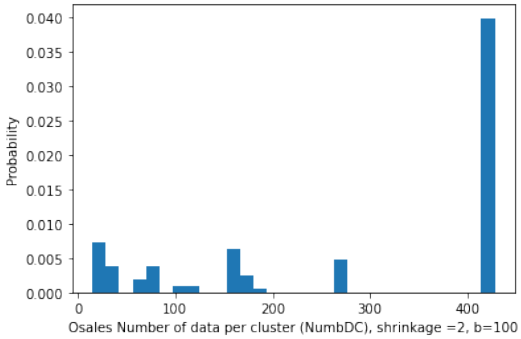


(b) $b = 100$

Figure 2.12: Jura, Number of cluster per partition (NumbCPar), shrinkage $= 2$

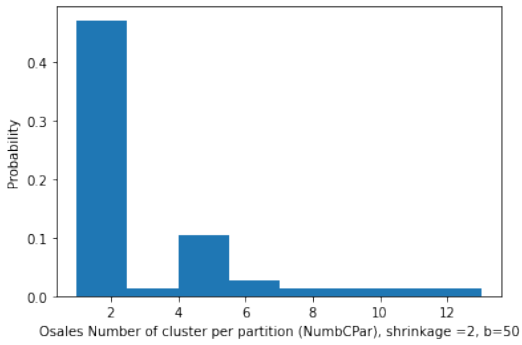


(a) $b = 50$

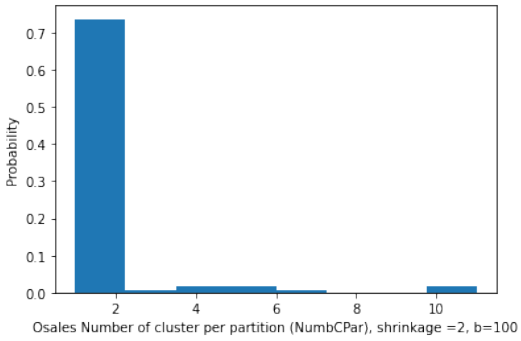


(b) $b = 100$

Figure 2.13: Online sales, Number of data per cluster (NumbDC), shrinkage $= 2$

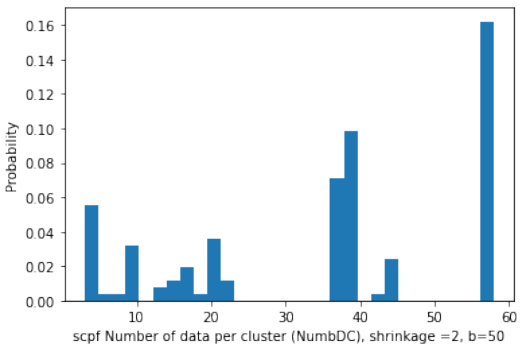


(a) $b = 50$

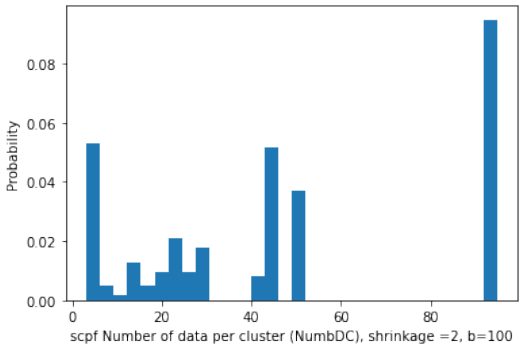


(b) $b = 100$

Figure 2.14: Online sales, Number of cluster per partition (NumbCPar), shrinkage $= 2$

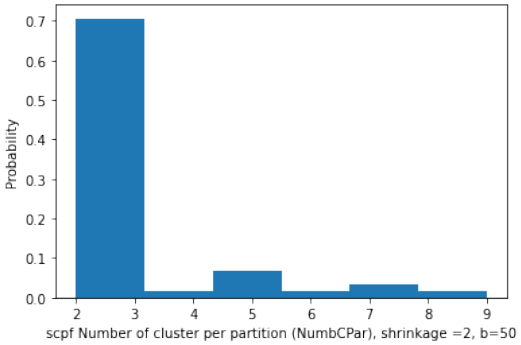


(a) $b = 50$

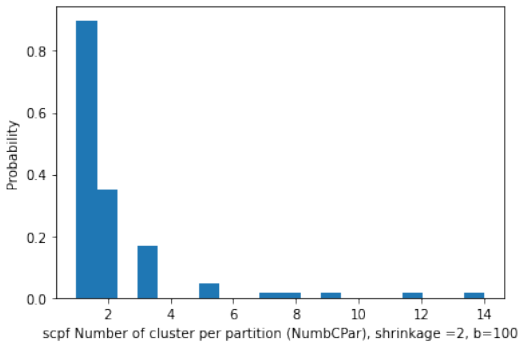


(b) $b = 100$

Figure 2.15: scpf, Number of data per cluster (NumbDC), shrinkage $= 2$

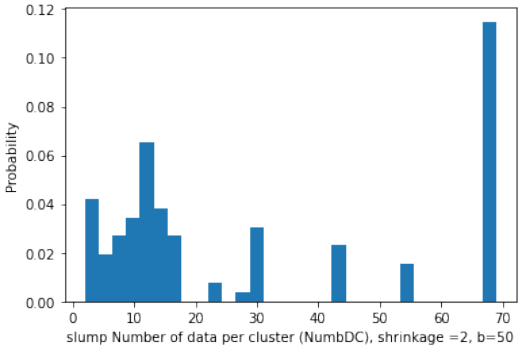


(a) $b = 50$

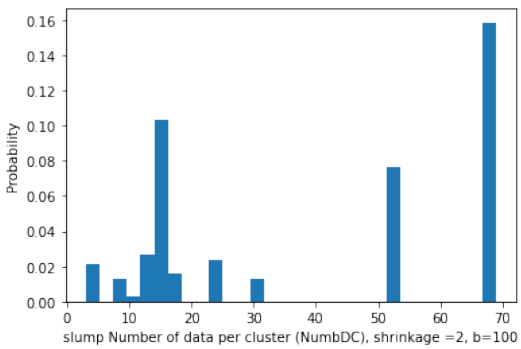


(b) $b = 100$

Figure 2.16: scpf, Number of cluster per partition (NumbCPar), shrinkage $= 2$

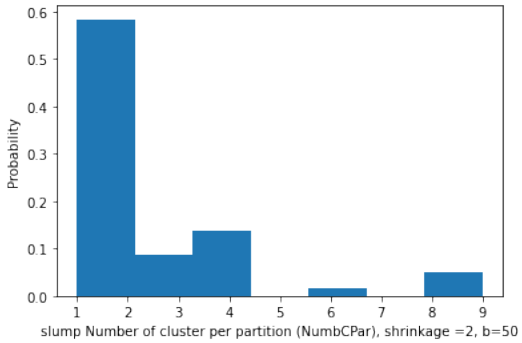


(a) $b = 50$

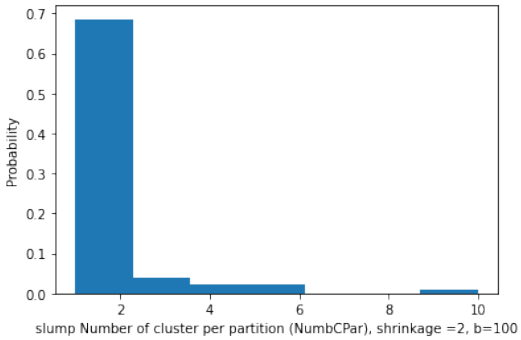


(b) $b = 100$

Figure 2.17: slump, Number of data per cluster (NumbDC), shrinkage $= 2$

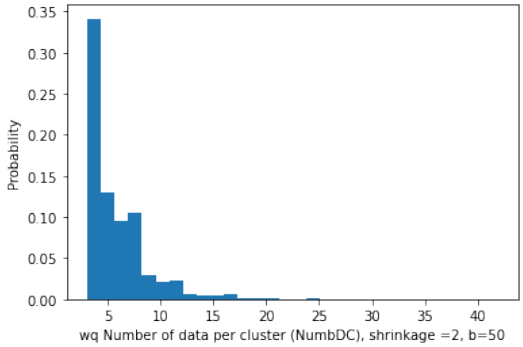


(a) $b = 50$

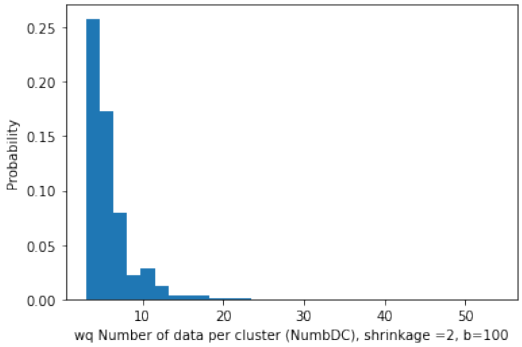


(b) $b = 100$

Figure 2.18: slump, Number of cluster per partition (NumbCPar), shrinkage $= 2$

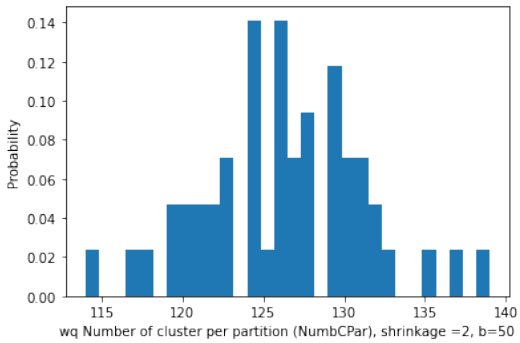


(a) $b = 50$

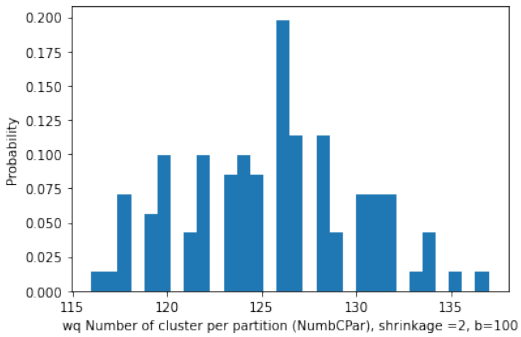


(b) $b = 100$

Figure 2.19: Water Quality, Number of data per cluster (NumbDC), shrinkage $= 2$



(a) $b = 50$



(b) $b = 100$

Figure 2.20: Water Quality, Number of cluster per partition (NumbCPar), shrinkage $= 2$

2.6 Extended Research on the Components size distribution

2.6.1 Frequency of frequencies distribution

The objective of the Potts clustering algorithm is to find $R(1 \leq R \leq n)$ clusters, $\mathcal{C} = \{\mathcal{C}_1, \mathcal{C}_2, ..., \mathcal{C}_R\}$. Each clustering specifies a sequence of cluster sizes, namely, letting $c_h = |C_h|$ be the size of the h-th cluster, then the sequence of cluster sizes is $[c_1, c_2, ..., c_R]$. Let denote $u_i \in \{1, ..., R\}$, the cluster observation x_i is assigned to, $c_h = \sum_{i=1}^{n} \delta(u_i = h)$ the number of observations in cluster h, and $m_v = \sum_{h=1}^{R} \delta(c_h = v)$ the number of clusters of size v, where here $\delta(x) = 1$ if condition x is satisfied, and $\delta(x) = 0$ otherwise. Thus by definition (Zhou et al., 2017), we have:

$$R = \sum_{i=1}^{n} m_i \text{ and } n = \sum_{i=1}^{n} i m_i$$

For example, if $n = 14$, $(u_1, ..., u_{14}) = (1, 2, 3, 4, 5, 5, 6, 6, 6, 6, 7, 7, 7, 7)$ as $(c_1, ..., c_7) = (1, 1, 1, 1, 2, 4, 4)$, we have $\{m_1, m_2, m_4\} = \{4, 1, 2\}$ and $m_i = 0$ for $i \notin \{1, 2, 4\}$.

As a reminder, R is a random variable. We refer the count vector $\mathcal{M} = (\{m_v\})_v$ as the frequency of frequencies (FoF) vector, the distribution of which is commonly referred to as the FoF distribution (Good, 1953).

2.6.2 Objective

Our objective is to find the FoF distribution of clusters count vector given the bonds probabilities. Knowing the graph G and its edges set $E(G)$, with $\delta_{ij} = 1$ if i and j are connected, this is express as :

$$FoF(\{m_v\}|\{\delta_{ij}\}; E(G))$$

And then derive the conditional distribution of bonds given the constraint cluster size condition $\mathcal{S}_c = \{m_v = 0, \text{ for } v \leq S_c\}$, where S_c is the minimum cluster size we want.

This is express as :

$$p(b_{ij}|\{z_{si}\}, \mathcal{S}_c)$$

Remark
As highlighted above, q influences the clustering. One may be interested in :

1. *The FoF distribution $\mathcal{W} = (\{\omega_v\})_v$ for the count vector of all subsets (of data) of unique label after the first step of label assignment when q is given. This will provide a way to condition the*

label assignment process with constraint subset size condition $\mathcal{S}_d = \{\omega_v = 0,$ for $v \leq S_d\}$, where S_d is the minimum subset size we want. Let $\{\hat{z}_{si}\}$ be the conditioned label assignment.

2. *And then search for the conditional distribution of bonds given the constraint subset size condition $\mathcal{S}_d$ and the constraint cluster size condition $\mathcal{S}_c$. This is expressed as :*

$$p(b_{ij}|\mathcal{S}_d, \mathcal{S}_c) = p(b_i j|\{\hat{z}_{si}\}, \mathcal{S}_c)$$

3. *Finally, one can insert a prior $p(\sigma, T)$ as done by Murua & Wicker (2014a) and access the MAP for the model [but not required].*

2.6.3 Methodology and combinatorial approach to the count vector

We present a mathematical framework to compute the distribution of the count vector $\mathcal{W} = (\{\omega_v\})_v$.

The FoF distribution for all subsets (of data) of unique label (at initial labelling)

Initial labelling may have a great influence on the clustering process. To find the FoF distribution for all subsets (of data) of unique label (at initial labelling) when q is given, the Potts Model can be related to many-body quantum systems problem, and describe in terms of a probability distribution over the quantum states (Tong (2012) and Tong (2006)).

Knowing the probability of a given labelling configuration $\{z_{si}\}$ as $p(\{z_{si}\}|\sigma, X, \beta, q)$, one can infer $p(\{\omega_v\}|\sigma, X, \beta, q)$. Each state configuration induces a sequence of subsets (of data) of unique label. Let $[L_1, L_2, ..., L_e]$, and $[l_1, l_2, ..., l_e]$ the sequence of subsets of data (of unique label) and their $\#$-cardinal respectively. $z_i \in \{1, ..., e\}$ represents the subset of observation i, such that :

$$p(\{z_{si}\}|\sigma, X, \beta, q) = p(\{z_i\}|\sigma, X, \beta, q)$$

Let denote V the index set of $\{\omega_v\}$ we have necessarily $\#V \leq n$.

$$\begin{aligned} p(\{\omega_v\}|\sigma, X, \beta, q) &= p(\omega_1, \omega_2, ..., \omega_v, ..., \omega_n, \sum_v \omega_v = e|\sigma, X, \beta, q) \\ &= \sum_{Hot_w(i_i, i_2, ..., i_e)=1} p(l_1 = i_1, l_2 = i_2, ..., l_e = i_e|\sigma, X, \beta, q) \\ &= \sum_{\substack{Hot_w(i_i, i_2, ..., i_e) = 1 \\ Hat_w(z_1, z_2, ..., z_n) = 1}} p(z_1, z_2, ..., z_n|\sigma, X, \beta, q)) \end{aligned} \tag{2.2}$$

with

$$Hot_v^w = \Big\{(i_1, i_2, ..., i_k, ..., i_e) \in \mathbf{N}^*, 1 \leq i_k \leq n, \text{ and } \sum \delta(i_k = v) = \omega_v \text{ for all } \omega_v \in (\omega_1, \omega_2, ..., \omega_v)\Big\}$$

,

$$Hat_v^w = \left\{(z_1, z_2, ..., z_k, ..., z_n) \in \mathbf{N}^*, 1 \leq z_k \leq e \text{ and } \sum \delta(z_k = j) = i_j, \text{ for all } i_j \in (i_i, i_2, ..., i_e)\right\}$$

and

$$Hot_w(i_i, i_2, ..., i_e) = \begin{cases} 1 \text{ if } (i_i, i_2, ..., i_e) \in Hot_v^w \\ 0 \text{ otherwise} \end{cases}$$

$$Hat_w(z_1, z_2, ..., z_k, ..., z_n) = \begin{cases} 1 \text{ if } (z_1, z_2, ..., z_k, ..., z_n) \in Hat_v^w \\ 0 \text{ otherwise} \end{cases}$$

Based on previous computation, it is easy to get the conditioned label assignment distribution including the subset size constraint $\mathcal{S}_d$:

$$\begin{aligned} p(\{z_{si}\}|\sigma, X, \beta, q, S_d) &= p(\{z_i\}|\sigma, X, \beta, q, S_d) \\ &= \frac{p(\{z_i\}, S_d|\sigma, X, \beta, q)}{p(S_d|\sigma, X, \beta, q)} \\ &= \frac{p(\{z_i\}, S_d|\sigma, X, \beta, q)}{p(\{\omega_v = 0, \text{ for } v \leq S_d\}|\sigma, X, \beta, q)} \end{aligned} \tag{2.3}$$

The advantage of this conditioned distribution is that even if q is larger, subsets of significant size (size larger than in S_d constraint) can be generated at first iteration. As we know how to compute $p(\{z_i\}|\sigma, X, \beta, q)$ [by coming back to $p(\{z_{si}\}|\sigma, X, \beta, q, S_d)$] with the Hamiltonian quantity, it is easy to deduce also $p(\{z_i\}, S_d|\sigma, X, \beta, q)$. Because $\{z_i\}$ induced by $\{z_{si}\}$ give a subsets sequence configuration, it is simple to understand the combination : $\left\{\{z_i\}, S_d\right\}$, which is equal to $\left\{\{z_{si}\}, \{\omega_v = 0, \text{ for } v \leq S_d\}\right\}$ and evaluate the expression above. Of course, the condition $\left\{\{z_{si}\}, \{\omega_v = 0, \text{ for } v \leq S_d\}\right\}$ can be fitted by combinatorial *search* on computer if the dataset is not too large.

FoF distribution for the Potts clusters

The following calculations are based on section 2.6.1 notations. Given the symmetric matrix M_b of bonds probabilities $\{p_{ij}\}$, one can infer a combinatorial approach to get the FoF distribution of clusters count vector. For example, if $n = 3$ and M_b defined as follows :

$$M_b = \begin{pmatrix} 0 & p_{12} & p_{13} \\ p_{12} & 0 & p_{23} \\ p_{13} & p_{23} & 0 \end{pmatrix}$$

we can see that

$$p(u_1 = 1, u_2 = 1, u_3 = 2) = p_{12}(1 - p_{23})(1 - p_{13})$$

So clearly, for R clusters, the configuration $(u_1, ..., u_n)$ with $u_i \in \{1, 2, ..., R\}$ defines a set $\mathcal{P}_f$ of disjoints frozen paths configurations for which (respective) probabilities can be easily computed.

$$\begin{aligned} p(\{m_v\}|\sigma, X, \beta, q) &= p(m_1, m_2, ..., m_v, ..., m_n, \sum_v m_v = R|\sigma, X, \beta, q) \\ &= \sum_{Hot_m(i_i, i_2, ..., i_R)=1} p(n_1 = i_1, n_2 = i_2, ..., n_R = i_R|\sigma, X, \beta, q) \\ &= \sum_{\substack{Hot_m(i_i, i_2, ..., i_R) = 1 \\ Hat_m(u_1, u_2, ..., u_n) = 1}} p(u_1, u_2, ..., u_n|\sigma, X, \beta, q)) \end{aligned} \tag{2.4}$$

with

$$Hot_v^m = \left\{(i_1, i_2, ..., i_k, ..., i_R) \in \mathbf{N}^*, 1 \leq i_k \leq n, \text{ and } \sum \delta(i_k = v) = m_v \text{ for all } m_v \in (m_1, m_2, ..., m_v)\right\}$$

,

$$Hat_v^m = \left\{(u_1, u_2, ..., u_k, ..., u_n) \in \mathbf{N}^*, 1 \leq u_k \leq R \text{ and } \sum \delta(u_k = j) = i_j, \text{ for all } i_j \in (i_i, i_2, ..., i_R)\right\}$$

and

$$Hot_w(i_i, i_2, ..., i_e) = \begin{cases} 1 \text{ if } (i_i, i_2, ..., i_e) \in Hot_v^w \\ 0 \text{ otherwise} \end{cases}$$

$$Hat_w(z_1, z_2, ..., z_k, ..., z_n) = \begin{cases} 1 \text{ if } (z_1, z_2, ..., z_k, ..., z_n) \in Hat_v^w \\ 0 \text{ otherwise} \end{cases}$$

Here, $p(u_1, u_2, ..., u_n|\sigma, X, \beta, q))$ is evaluated among the set $\mathcal{P}_f$ of all disjoints frozen paths configurations induced by $(u_1, ..., u_n)$ with $u_i \in \{1, 2, ..., R\}$.

We can then simply update the bonds distribution :

$$\begin{aligned} p(\{b_{ij}\}|\sigma, X, \beta, q, S_c) &= \frac{p(\{b_{ij}\}, S_c|\sigma, X, \beta, q)}{p(S_c|\sigma, X, \beta, q)} \\ &= \frac{p(\{b_{ij}\}, S_c|\sigma, X, \beta, q)}{p(\{m_v = 0, \text{ for } v \leq S_c\}|\sigma, X, \beta, q)} \end{aligned} \tag{2.5}$$

Remark

1. *The probability $p(\{b_{ij}\}|\sigma, X, \beta, q, S_c)$ is implicitly given by the matrix of bonds probabilities M_b, and its exact expression can also be obtained analytically.*

2. *The matrix of bonds probabilities M_b is mainly influenced by the label assignment, because of δ_{ij} in expression of p_{ij} ($p_{ij} = p(b_{ij}|\delta_{ij}) = \delta_{ij}\big(1 - \exp\{-\beta k_{ij}(\sigma)\}\big)$). $k_{ij}(\sigma)$ is always the same for any i and j.*

From previous remark, we will describe in next section how to compute $p(\{b_{ij}\}, S_c|\sigma, X, \beta, q)$, to find a fully explicit expression of $p(\{b_{ij}\}|\sigma, X, \beta, q, S_c)$, which is here the conditional bonds distribution.

2.6.4 The conditional bonds distribution (given the size constraint)

Computation method

In previous bonds evaluation approach, we start with all p_{ij} evaluation and comparison to a given threshold (let say 0.5 for example). In this approach, we automatically select the most probable $\{b_{ij}\}$ configuration. For our context of conditional bonds, evaluation of all single p_{ij} is not *useful* to access the right frozen state that fit the cluster size constraint for all bonds.

To access $p(\{b_{ij}\}, S_c|\sigma, X, \beta, q)$, the best method here is to build a list of all $\{b_{ij}\}$ configurations (w.r.t $b_{ij} = 0$ if x_i and x_j are not neighbors) that satisfy the cluster size constraint $\mathcal{S}_c = \{m_v = 0, \text{ for } v \leq S_c\}$ (by a combinatorial search), and choose the most highly probable configuration among those selected given a label assignement.

1. There is a fast computation algorithm to find the list of all bonds $\{b_{ij}\}$ configurations (w.r.t $b_{ij} = 0$ if x_i and x_j are not neighbors) that meet the cluster size constraint $\mathcal{S}_c$. This list will be called the S_c-list, and the list of all bonds $\{b_{ij}\}$ configurations (w.r.t $b_{ij} = 0$ if x_i and x_j are not neighbors) will be called the $G-$list (where notation G is set to signify : general or global).

2. The S_c-list should be built at the beginning, and the respective probability of each of its element evaluate for each label assignment (as a reminder matrix M_b changes for each label assignment).

2.6.5 Fast-Algorithm to find S_c-list

1. For each $\{b_{ij}\}$ configuration, we use the Hoshen-Kopelman (HK) algorithm (Li, 2011) to connect frozen spin pairs into a path of frozen bonds. This step helps in finding all clusters given the bonds configuration.

2. All frozen paths have many *extended*-version, and *shrunk*-version. The frozen path is said to be *extended* if it is augmented with other vertices. When the path is reduced to less number of vertices (contains fewer vertices), it is said to be *shrunk*. When the frozen path is simultaneously *shrunk* and *extended* (i.e some old vertices are replaced with newer vertices), it is said to be *mixed*.

3. Let denote $G-$-List$[i]$ the ith element from the G-list. To avoid searching for all frozen paths for all $\{b_{ij}\}$ configurations, here is an algorithm to go through the list and search for all $\{b_{ij}\}$

configurations that fit the clusters size condition:

S_c-**algorithm**

---

1 Starting at $G-$list[1], Go through G-list and find with HK algorithm the first configuration $G-$list$[i_k]$ for which the small frozen path $SP_f[i_k]$ has a length upper than in condition S_c; add $G-$list$[i_k]$ to S_c-list.

2 In the next step, use $SP_f[i_k]$ to make a selection of good candidates:

- Add to S_c-list all configurations that contain either a *shrunk*, *extended* or *mixed* version of $SP_f[i_k]$ that fit the cluster size condition $\mathcal{S}_c$. Remove those one from G-list.
- Delete from G-list all configurations that contains either a *shrunk* or *mixed* version of $SP_f[i_k]$ that don't fit the cluster size condition $\mathcal{S}_c$.

3 Move to the next $G-$list$[i_{k+1}]$ that fit the condition, and repeat step [2] by using $SP_f[i_{k+1}]$.

4 Repeat steps [1], [2] and [3] for any other element of $G-$list that fit the clusters size condition, until done!

2.6.6 Finding the most probable configuration among S_c-list given a label assignment

Here we describe a computation method to select the most probable configuration (in S_c-list) given a label assignment.

1. Most importantly, **given a label assignment**, we need to delete from S_c-list all configurations for which there exist any $\delta_{ij} = 0$, and $\big(1 - \exp\{-\beta k_{ij}(\sigma)\}\big)$ suggest $b_{ij} = 1$. In fact, we generate in practice the bonds with $\big(1 - \exp\{-\beta k_{ij}(\sigma)\}\big)$, and check for the state of δ_{ij} to confirm that the label assignment is the same.

   Let call $\tilde{S}_c$-list the adjusted S_c-list which includes this modification.

2. The Bonds matrix probabilities $M_b = (p_{ij})$ with $p_{ij} = p(b_{ij}|\delta_{ij}) = \delta_{ij}\big(1 - \exp\{-\beta k_{ij}(\sigma)\}\big)$, can be seen as a upper triangular matrix re-written as **an element by element** product of two upper triangular matrix with **diagonal elements set to zero** :

$$M_b = \tilde{\mathbf{P}} \circ \tilde{\delta}$$

   with $\tilde{\mathbf{P}} = (\tilde{p}_{ij})$, $\tilde{p}_{ij} = 1 - \exp\{-\beta k_{ij}(\sigma)\}$, and $\tilde{\delta} = (\delta_{ij})$. Only matrix $\tilde{\delta}$ change after each label assignment.

   The upper triangular bonds matrix $B = (b_{ij})$ with **diagonal elements set to zero** can also be introduced for a given configuration $\{b_{ij}\}$ in $\tilde{S}_c$-list.

The probability of a given configuration in $\tilde{S}_c$-list is the product of all elements in upper right-corner of the following triangular matrix (excluding diagonal elements):

$$[\tilde{P} \circ \tilde{\delta}]^B + [(1 - \tilde{P}) \circ \tilde{\delta}]^{1-B}$$

for which each non-null element is computed as :

$$\left(\tilde{p}_{ij} \cdot \delta_{ij}\right)^{b_{ij}} + \left(\{1 - \tilde{p}_{ij}\} \cdot \delta_{ij}\right)^{1-b_{ij}}$$

We have then :

$$p(\{b_{ij}\}, \{b_{ij}\} \in \tilde{S}_c - \text{list}|\sigma, X, \beta, q, S_c) = \frac{PROD_{i\in\{1,\ldots,n-1\},j>i}\left([\tilde{P} \circ \tilde{\delta}]^B + [(1 - \tilde{P}) \circ \tilde{\delta}]^{1-B}\right)}{p(\{m_v = 0, \text{ for } v \leq S_c\}|\sigma, X, \beta, q)} \tag{2.6}$$

where $PROD_{i\in\{1,\ldots,n-1\},j>i}$ describe the product of all elements in the upper right-corner of the matrix.

3. The most probable configuration can then be selected based on this computation. This probability in 2.6 is computationally intractable for large datasets but remains **a contribution of this research**, as it can be used for further calculus, or mathematics developments.

Chapter 3

Deep learning and the Classical Neural Networks

3.1 The General Multi-Layer FeedForward Neural Network: Definitions & Concepts

Definition 3.1.1 (Multilayer Neural Network Model)
We consider first a general case where the space for all observations $\mathcal{D}_n$ consists of n points:

$$\mathcal{D}_n = \big(\boldsymbol{x}, \boldsymbol{y}\big) = \big\{(\boldsymbol{x}_1, \boldsymbol{y}_1), \ldots, (\boldsymbol{x}_{n-1}, \boldsymbol{y}_{n-1}), (\boldsymbol{x}_n, \boldsymbol{y}_n); \boldsymbol{x}_i \in \mathbb{R}^{p_1 \times p_2}; \boldsymbol{y}_i \in \mathbb{R}^{q_1 \times q_2}; p_1, p_2, q_1, q_2 \in \mathbb{N}\big\}$$

For a k-layers network (with multiple hidden layers), neural network model is called multi-layer *network, and the architecture of the network is designed as follows:*

$$\boldsymbol{x} = \mathrm{h}_0 \qquad : \text{the input layer, } \boldsymbol{x} \in \mathbb{M}_{n \times q}$$

$$\mathrm{h}_k = \boldsymbol{b}_k + g_k(\mathrm{h}_{k-1})\boldsymbol{\mathcal{W}}_k \tag{3.1}$$

Equation (4.1) defines the feed-forward function; and $\mathrm{h}_1 = \boldsymbol{b}_1 + h_0\boldsymbol{\mathcal{W}}_1$ is the first layer, where the symbol $+$ has to be seen here as $\oplus$ the Kronecker sum, or a broadcast sum in a proper term. In other words, the estimated output $\hat{\boldsymbol{y}}$ obtained on from the neurons $(O)_{1:l_k}$ (illustrated in figure 3.1) from the model is given by :

$$\begin{aligned} \hat{\boldsymbol{y}} &= \hat{\mathrm{h}}_k = \hat{\boldsymbol{b}}_k + g_k(\hat{\mathrm{h}}_{k-1})\hat{\boldsymbol{\mathcal{W}}}_k = g_k(\hat{\boldsymbol{b}}_{k-1} + g_k(\hat{\mathrm{h}}_{k-2})\hat{\boldsymbol{\mathcal{W}}}_{k-1})\hat{\boldsymbol{\mathcal{W}}}_k \\ &= f_\psi(\boldsymbol{x}) \end{aligned} \tag{3.2}$$

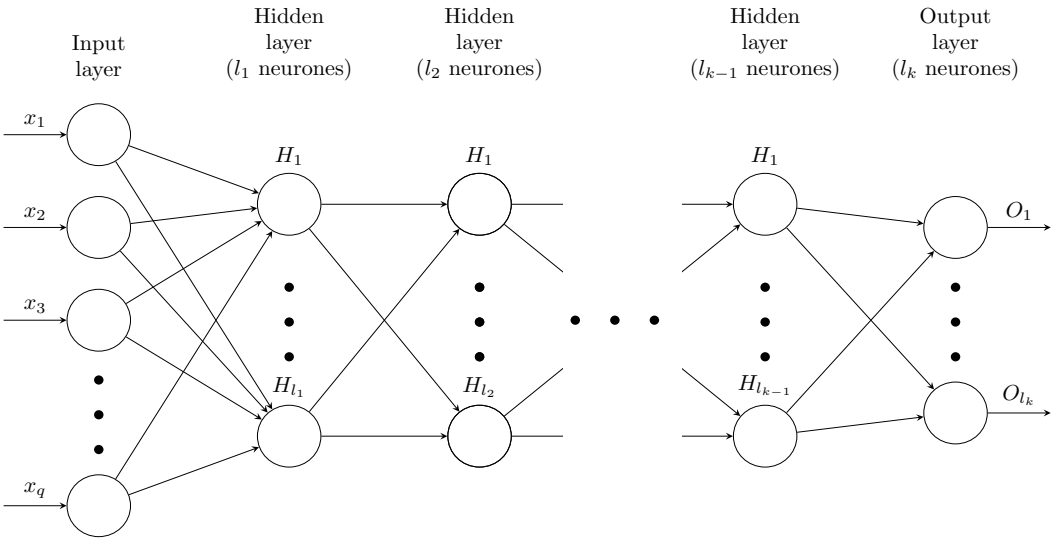


Figure 3.1: Multi-layer feed-forward neural network

where $g_k(x)$ represents the activation function of the k^{th} layer, f_ψ is the whole network function, with $\boldsymbol{\psi} = (\boldsymbol{\mathcal{W}}, \boldsymbol{b})$, $\boldsymbol{\mathcal{W}} = (\boldsymbol{\mathcal{W}}_1, \boldsymbol{\mathcal{W}}_2, ..., \boldsymbol{\mathcal{W}}_k)$ the weights matrix, and $\boldsymbol{b} = (\boldsymbol{b}_1, \boldsymbol{b}_2, ..., \boldsymbol{b}_k)$ the biases of the model. $\boldsymbol{\mathcal{W}}_k \in \mathbb{R}^{l_{k-1} \times l_k}$ may be a random matrix in a Bayesian configuration, and finally, l_k is the number of neurons of the k^{th} layer ($l_0 = q$).

This model that we introduce can learn a non-linear function approximator for either *classification* or *regression* (Hassoun et al. (1995)). In terms of composition, it is different from many known models, because, between the input and the output layer, there can be one or more non-linear layers, called *hidden* layers (figure 3.1). The main difference with a *Shallow*-Neural Network is that it is built up with k-layers (with multiple hidden layers, ($k > 2$)), hence the name *Multi-layer*-FeedForward Neural Network Model.

For further purpose, we will also define an error function, $\boldsymbol{\mathcal{J}}(\boldsymbol{\mathcal{W}}, \boldsymbol{b})$, which represents the error between the desired output $\boldsymbol{y}$ and the calculated output $\hat{\boldsymbol{y}}$ of the neural network on input $\boldsymbol{x}$ for a set of input-output pairs $(\boldsymbol{x}, \boldsymbol{y}) \in \mathcal{D}_n$ and a particular value of the parameters $(\boldsymbol{\mathcal{W}}, \boldsymbol{b})$.

> Any hidden layer of a neural network is thus simply a stack of models to the previous layer. The individual neuron acts like its own model in the layer, whose outputs feed additional neurons into another model stack (each successive hidden layer may have many more neurons in the neural network).

3.1.1 The Hold on Neural networks and the WHY

Deep learning (DL) is a new area and little is understood about an optimal representations of data, loss functions, and the techniques for data preparation. Binary signals, for example, may be interpreted as binary or one-hot vectors, modulated (complex) symbols or integers, and the optimal representation can depend on the architecture of the Neural Network, learning purpose, and loss function, among other variables (O'Shea & Hoydis, 2017). For this reason, researches are growing in this direction and many

novices are increasing their interest.

One appeal of neural networks is their ability to derive high-level, dynamic abstractions and representations in the face of vast quantities of data via a suitable procedure (Najafabadi et al., 2015). Neural networks can be quickly built, due to the rapid development and wide availability of data, and have become an appropriate predictive platform for national intelligence, cyber protection, fraud prevention, marketing, and medical knowledge issues (Chen & Lin, 2014).

In particular, in neural network architectures, the explicit nonlinear relationship is very useful in constructing a general mapping model without defining any functional types in advance (Chang & Su, 1995). Chang & Su (1995) showed that while the statistical regression approach involves careful model selection to perform well by statistical testing methods, many neural network architectures perform well while carefully gathering training data.

Bansal et al. (1993) have demonstrated a distinction of regression analysis and neural network analysis, discussed a real-world example from the field of finance. They observed that, as data quality deteriorated, neural net-based predictions appeared to be more stable than linear regression estimates. They are not only capable of capturing non-linear phenomena, but have demonstrated strong success in their application in finance. (see Chen et al. (2003) and Zhang et al. (2007)).

Neural networks may have many benefits over traditional models of regression. Without the need for a complete definition of the model, they are claimed to own the property to *learn* information from a collection of data; they are also claimed to be able to see beyond noise and distortion (Marquez et al., 1991). In theory, a sufficiently large neural network can learn *any* function (see the Universal Approximation Theorem with paper).

Hornik (1991) paper *"Approximation Capabilities of Multilayer Feedforward Networks"* demonstrates that "standard multilayer feedforward networks with as few as a single hidden layer and arbitrary bounded and nonconstant activation function are universal approximators with respect to $L^P(\mu)$ performance criteria, for arbitrary finite input environment measures μ, provided only that sufficiently many hidden units are available." In other words, the assumption that the activation function is bounded and non-constant is enough to approximate almost any function, provided that we can use as many hidden units in the neural network as we want.

Neural network models are then an opportunity to explore: their conjunction with statistical modelling gives classical model families a strong extension, as the later offer a systematic way of obtaining good initializations (Ciampi & Lechevallier, 1997).

3.2 Classification of neural networks

1. Multilayer Perceptron – It is simply a feedforward artificial neural network model, the original name for the one we introduced at the beginning of this chapter, it maps sets of input data onto a set of appropriate outputs with multiple referenced applications : Noriega (2005); Gardner & Dorling (1998); Pal & Mitra (1992); Ramchoun et al. (2016); Attali & Pagès (1997); Attali & Pagès (1997); Chaudhuri & Bhattacharya (2000); Murtagh (1991); Parlos et al. (1994); Suykens & Vandewalle (1999); Suykens & Vandewalle (1999); Sivaram & Hermansky (2011).

2. Memory network: networks with context memory. Memory Networks combine inference components with a memory component. Recurrent Neural Networks or network with memory are referenced here : Dyer et al. (2016); Mikolov et al. (2011); Zaremba et al. (2014); Medsker & Jain (2001)), Long Short Term Memory Network are such an example (Malhotra et al. (2015); Tai et al. (2015); Fischer & Krauss (2018); Sak et al. (2014); Zhu et al. (2015); Bakker (2001); Kalchbrenner et al. (2015); Graves (2012); Hochreiter & Schmidhuber (1997). These networks have many favourable features, including better features, Memory power, high noise robustness, increased memory overloading robustness, and enhanced memory retention during learning. (Fuster (1997); Cheng et al. (2016); Wang et al. (2018); Sukhbaatar et al. (2015); Kumar et al. (2016); Pan et al. (2017); Soda et al. (2004); Weston et al. (2014); Chunseong Park et al. (2017)).

3. Dynamic neural network:

   It is important to split these complex networks into two categories: those with only feed-forward connections and those with feedback or recurring connections. This separation may intercept the previous category as well. A dynamic neural network, in practice, consists of a series of interconnected neurons that communicate continuously. Applications and examples are numerous: Tayarani-Bathaie et al. (2014); Yang & Ni (2005); Shaw et al. (1997); Sanchez & Perez (1999); Ito et al. (2006); Ishak et al. (2003); Ishak et al. (2003); Mordjaoui et al. (2017); Hatti & Tioursi (2009); Yu & Li (2001); Yu & Li (2001); Jiang & Adeli (2005); Shoaib et al. (2016); Hosoda et al. (2013); Leven & Levine (1996); Sholahudin & Han (2016); Ghiassi et al. (2005). Dynamic networks are generally more powerful than static networks (although somewhat more difficult to train). Because dynamic networks have memory, they can be trained to learn sequential or time-varying patterns. This has applications in such disparate areas as prediction in financial markets (Roman & Jameel, 1996), channel equalization in communication systems (Feng et al., 2003), phase detection in power systems (Kamwa et al., 1996), sorting (Rahman et al., 2004), fault detection (Gan & Danai, 1999), speech recognition (Robinson, 1994), and even the prediction of protein structure in genetics (Pollastri et al., 2002). You can find a discussion of many more dynamic network applications in (Medsker & Jain, 1999).

4. Radial Basis Network – A radial basis function network is an artificial neural network. It uses radial basis functions as activation functions. References and applications are available as well: Yeung et al. (2007); Yeung et al. (2007); Chen et al. (1991); Venkatesan & Anitha (2006); Orr et al. (1996); Jang & Sun (1993); Sudheer & Jain (2003); Bishop (1991); Leung et al. (2001); Dash et al. (2000); Hwang & Bang (1997); Vt & Shin (1994); Vt & Shin (1994); Yingwei et al. (1998); Bors & Pitas (1996). One example of Radial Basis Function (RBF) is as follow:

$$\text{activation}\,(\zeta) = \exp\left(s^* \log(\text{altitude}) \; - \zeta\right)$$

   Where s is here the *fan-in* of each unit in the layer, that is the number of other units feeding into that unit, excluding bias, and the altitude is a positive number stored in Neuron or NeuralLayer or NeuralNetwork. The default is altitude=1 with the activation function decreasing to a basic $\exp(-\zeta)$ for that amount.

5. Other types of networks: networks that operate similar to neuronal (mathematical functions) and synaptic states (linking neurons) with the additional feature that these networks also incorporate the concept of time into their operating model (Liu & Perez (2017)). Such networks impose

synaptic controls with specific gates. One example is the Gate Recurrent Units Network (Costa et al. (2017); Yao et al. (2015); Xu et al. (2017); Dey & Salemt (2017); Chung et al. (2015); Tang et al. (2015); Chung et al. (2014)).

For the scope of this research, we will limit our presentation to the Multilayer Perceptron type, commonly known as the feedforward artificial neural network model as introduced and presented here with his multilayer feature version.

3.3 Activation function and Loss function types

We want to find the set of weights (remember that a weight is found in each connecting line between any two components in a neural network) and biases (each neuron has a bias) that reduce our cost function - where the cost function is an estimate of how incorrect our predictions are relative to the target result.

As a reminder :

Definition 3.3.1 (Activation functions)
*The activation function is a mathematical "*gate*", a logical "*gate*" based on a law or threshold, it may be as basic as a phase function that turns the neuron output on and off.*

Here is a following list of some activation functions you may find in the literature (Agostinelli et al. (2014); Agostinelli et al. (2014)):

1. Rectified Linear Units (ReLU) : $\phi(\mathbf{v}) = \max(0, b + \mathbf{v}'\mathbf{W})$. This function (figure 3.2) is computationally efficient—allows the network to converge very quickly [Jin et al. (2015); Hussain & Jeong (2016); Wu et al. (2014); Jiang et al. (2018b); Le et al. (2015); Arora et al. (2016); Agarap (2018); Hara et al. (2015)]. But, ReLU is Non-linear—although it looks like a linear function, and has a derivative function and allows for backpropagation (Baldassi et al. (2019); Dolezel et al. (2019); Wagner & McComb (2019); Sigtia & Dixon (2014); Xiong et al. (2014)).

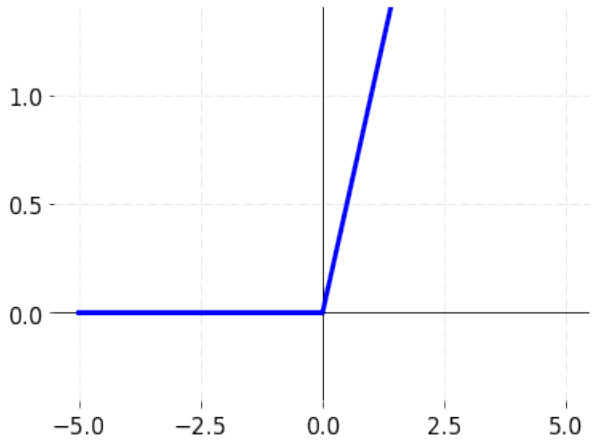


Figure 3.2: ReLU Activation function

2. Sigmoid Function called S-shaped functions: $\phi(\mathbf{v}) = \frac{1}{1+e^{-(b+\mathbf{v}'\mathbf{w})}}$. It is one of the most widely used non-linear activation function. The main reason why we use sigmoid function is because it exists between 0 to 1 (figure 3.3). It is also particularly used for models where the likelihood has to be predicted as an output. This function is distinguishable, and it is often referred to as a logistic function and its graph curve appears S-shaped. It is used in situations like making the final decision in a network with the binary classification layer (Sutskever & Hinton (2008); Panicker & Babu (2012); Han & Moraga (1995); Ngah et al. (2016); Uykan (2013); Kondo (2006); Tommiska (2003); Kros et al. (2006); Lin & Wang (2008); Menon et al. (1996); Kilian & Siegelmann (1993); Tsai et al. (2015); Yamanashi et al. (2012); Shibata & Ito (1999); Wanto et al. (2017); Elfwing et al. (2018); Ito (1991)).

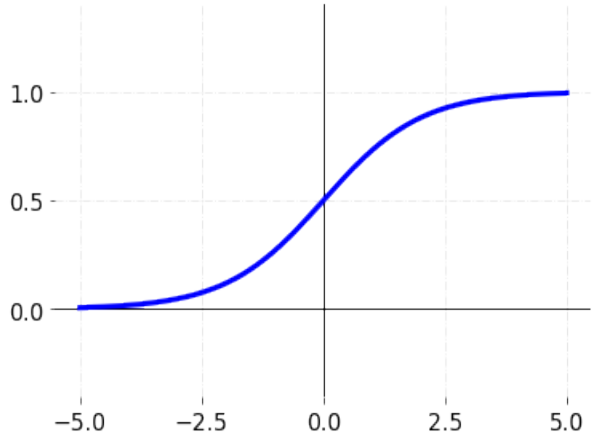


Figure 3.3: Sigmoid Activation function

3. Tangent Hyperbolic activation: $\phi(\mathbf{v}) = \frac{e^{(b+\mathbf{v}'\mathbf{w})}-e^{-(b+\mathbf{v}'\mathbf{w})}}{e^{(b+\mathbf{v}'\mathbf{w})}+e^{-(b+\mathbf{v}'\mathbf{w})}}$. Tanh can be thought of as a scaled sigmoid (it is a Bipolar Sigmoid logistic function), and is right to model inputs that have strongly negative, neutral, and strongly positive values (Shamsi et al. (2015); Namin et al. (2009); Zamanlooy & Mirhassani (2013); Roy et al. (2019); Lin & Wang (2008); Mathias & Rech (2012); Anastassiou (2011b); Anastassiou (2011a)). He is plotted in figure 3.4.

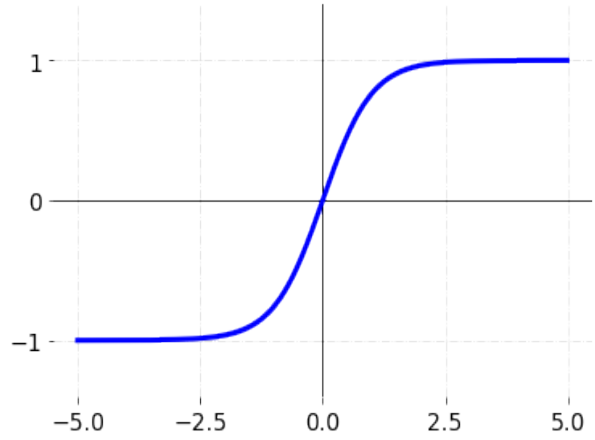


Figure 3.4: Tanh Activation function

4. Softmax output activation:

$$\text{softmax}(\mathbf{x}) = \left\{ \frac{e^{sx_1}}{\sum_{k=1}^{n} e^{sx_k}}, \frac{e^{sx_2}}{\sum_{k=1}^{n} e^{sx_k}}, \ldots, \frac{e^{sx_n}}{\sum_{k=1}^{n} e^{sx_k}} \right\}$$

Softmax $(\mathbf{x})$ is only defined for vector operations, it cannot take a scalar input. Softmax $(\mathbf{x})$ converts an input vector $\mathbf{x}$ with n elements into a probability distribution with n elements. Softmax guarantees each term to be positive by exponentiating any term in the input vector. By summing all exponentiated elements and dividing by that sum, and by assigning each element in the output vector to be the percentage that the input element added to the sum, softmax calculates a probability distribution. The scalar s is just a stretch parameter, it controls the variance of the distribution, or how "*spread out*" the distribution is (Kanai et al. (2018); de Brébisson & Vincent (2015); Chen et al. (2018); Qin et al. (2019); Su & Xu (2015); Tüske et al. (2015); Rimer & Martinez (2004); Liu et al. (2016); Yuan (2016); Wu et al. (2016); Gold et al. (1996)).

A Multilayer Perceptron can support multi-class classification by applying Softmax as the output function (Ranjan et al. (2017); Memisevic et al. (2010); Jiang et al. (2018a); Liang et al. (2017)). The raw output passes through the softmax function and determines each class probability. The most higher value is rounded to 1, the other values are rounded to 0. The index where the value is 1 represent the allocated class of that sample for the expected output class of the sample.

Some functions are, as you will read above, devoted only to the output layer. Output neurons are simply dependent on basic activation functions in parallel to input and hidden neurons, depending on the type of models required. The activation of one neuron determines the value of the corresponding input field, and the activation of the output neuron is determined by the model purpose (classification, regression, multi-classification, etc.). In networks with *supervised* learning, the normalised values of the respective reference fields are correlated with the computed activity of the output neurons. The difference between the neuron's activation and the normalized target field determines what is known as the *prediction error* (Hastie et al., 2009).

The activation functions play a crucial role in the preparation of Neural Networks Models. For this purpose, to develop effective and performing training, these roles represent a key subject in the deep learning environment. The trick of these techniques is to allow a stable activation feature to be learned, preventing the absence of gradient problems (Maguolo et al., 2019).

Table 3.1: Frequently used activation functions

Name	$[\sigma(\zeta)]_\mathbf{i}$	Range
linear	ζ_i	$(-\infty, \infty)$
ReLU	$\max(0, \zeta_i)$	$[0, \infty)$
tanh	$\tanh(\zeta_i)$	(-1,1)
sigmoid	$\frac{1}{1+e_i^{-\zeta_i}}$	(0,1)
softmax	$\frac{e^{\zeta_i}}{\sum_j e^{\zeta_j}}$	(0,1)

Small non-linearities in activation functions create bad local minima in neural networks (Yun et al., 2018). Specifically, Yun et al. (2018) have shown that for ReLU(-like) networks, and for almost all

practical datasets, there exist infinitely many local minima. But also, the choice of the right *loss* function, is a true matter of practice.

The loss function in Neural Networks Models

The most famous loss function in neural network is the Mean Squared Error (MSE), or L2 loss (table 3.2). It is more often used as regression loss that is computed by taking the average squared difference between actual $(\boldsymbol{w})$ and predicted observations $(\boldsymbol{z})$ (Zainuddin & Pauline (2008); Guresen et al. (2011); Cheng et al. (2005); Arsad et al. (2013); Xu (1993)).

Cross-Entropy: Cross-entropy loss, or log-loss, measures the performance of a classification model whose output is a probability value between 0 and 1 (Zhang et al. (2018c); Aurelio et al. (2019); Wiesler et al. (2013); Cao et al. (2018); Li et al. (2019b); Martinez & Stiefelhagen (2018); Semenov et al. (2019); Szita & Lörincz (2006); Nasr et al. (2002); Amos & Yarats (2020); Shan & Fang (2020); Xie et al. (2019); Fu et al. (2019); Panchapagesan et al. (2016); Golik et al. (2013); Li et al. (2019c); Zhang & Sabuncu (2018); Bosman et al. (2020); Hu et al. (2018b); Kline & Berardi (2005)). As the expected probability diverges from the real mark, cross-entropy loss increases. A perfect model would have a log-loss of 0. If $\boldsymbol{\Xi} > 2$ (i.e. multiclass classification), we calculate a separate loss for each class label per observation and sum the result as follows[1]:

$$-\sum_{\boldsymbol{\xi}=1}^{\Xi} \boldsymbol{e}_{\boldsymbol{\zeta},\boldsymbol{\xi}} \log\left(\boldsymbol{p}_{\boldsymbol{\zeta},\boldsymbol{\xi}}\right)$$

- $\boldsymbol{\Xi}$ is the number of classes;
- $\boldsymbol{e}_{\boldsymbol{\zeta},\boldsymbol{\xi}}$ is a binary indicator $(0$ or 1 $)$ if class label $\boldsymbol{\xi}$ is the correct classification for observation $\boldsymbol{\zeta}$
- $\boldsymbol{p}_{\boldsymbol{\zeta},\boldsymbol{\xi}}$ is the predicted probability that observation $\boldsymbol{\zeta}$ is of class $\boldsymbol{\xi}$

In binary classification, where the number of classes $\boldsymbol{\Xi}$ equals 2 , cross-entropy can be calculated as:

$$-(\boldsymbol{e}\log(\boldsymbol{p}) + (1-\boldsymbol{e})\log(1-\boldsymbol{p}))$$

Table 3.2: List of loss functions

Name	$l(\mathbf{w},\mathbf{z})$
MSE	$\|\mathbf{w}-\mathbf{z}\|_2^2$
Categorical cross-entropy	$-\sum_j \mathbf{w}_j \log\left(\mathbf{z}_j\right)$

3.4 Notes on The Geometry of the loss function

Training on the neural network depends on our ability to find "good" minimizers or a loss function which is strongly non-convex. It is well known that some network architectures require simpler training

[1]We took this definition from the Python PyPy Manual as it is well illustrative of how to compute the cross-entropy.

loss functions, and well-chosen training parameters (batch size, learning rate, optimizer) generate better generalising minimizers.

However, the reasons for these variations, and their effect on the underlying loss surface, are not well known. Using a variety of findings from research papers, we discuss the convexity nature of neural loss functions and the impacts of the loss surface on generalisation in this section.

3.4.1 The non-convexity problem

It is important to understand the loss surface of deep Neural Network (DNNs) under practical conditions to provide principled ways of modelling proper models (Falas & Stafylopatis, 1999). Non-convexity is one of the key problems in the analysis of neural networks, because of the loss function that may have multiple poor local minimums. As an example of neural networks for binary classification tasks, Liang et al. (2018) have shown that any local minimum becomes a global minimum per area (called region) after adding one special neuron with a skip relation to the output, or one special neuron per layer.

It is then necessary for the development of improved optimization algorithms to understand the geometry of neural network loss function surfaces, to create a theoretical understanding of why deep learning works. Using the distribution of the Hessian matrix eigenvalues, Pennington & Bahri (2017) have studied the geometry of the loss function at critical points of varying energy. They present an analytical structure and a collection of random matrix theory methods that allow, under a set of simplifying assumptions, to compute an approximation of this distribution. The parameter domain of the loss surface can be decomposed into regions in which activation values are consistent.

They found that the loss surface has similar properties to that of linear neural networks when decomposed into sub-region, where any local minimum is a global minimum. This means that every differentiable local minimum is the global minimum of the corresponding region. When all hidden layers are wider than the input or output layers, Laurent & Brecht (2018) prove that any local minimum of a deep linear network under differentiable convex loss is global.

3.4.2 Level Sets

The loss surface $\mathcal{J} = F(\Theta)$ of a given model can be expressed in terms of its level sets Ω_ζ, which contain for each energy level ζ all parameters Θ yielding a loss smaller or equal than ζ.

The loss feature involves weak local solutions very often (minima or maxima). It is connected to the form or geometry of the level sets, or their topology essentially. By approximating the geodesics of each level set beginning at two random boundary points, Freeman & Bruna (2016) has implemented an effective algorithm to estimate the geometric regularity of these level sets. His goal was to verify this intuition about, as their properties were so rich to understand all irregularities in the loss function.

The local conditioning of the loss surface and the model architecture that prevents the existence of bad local minima is the immediate follow-up query that then decides the convergence of algorithms in operation. A first question Freeman & Bruna (2016) have addressed concerns the topology of these level sets, i.e. under which conditions they are connected. Connected level sets mean that a descent path can always be found at each energy level, and therefore there can be no poor (weak) local minima. In absence of nonlinearities, deep (linear) networks have connected level sets (Kawaguchi, 2016). In

presence of non-linearity, this connection is not a guarantee.

This result justifies the research for the right loss function for a given model, and the use of gradients and backpropagation optimisation methods in neural networks.

3.5 The Science of Gradients and Backpropagation in Deep learning

> The training process of a neural network, at a high level, is like that of many other data science models - define a cost function and use gradient descent optimization to minimize it[a].
>
> [a]This is a quote from Tony Yiu, in a web-blog titled : "Understanding Neural Networks" in 2019.

Generally, if we try to find the minimum of a function, the derivative is set to zero and the parameters are solved. However, it turns out that it is difficult sometimes to get a closed-form solution for the functions of neural network loss gradients, especially for all $\mathcal{W}$ and $\boldsymbol{b}$. Instead, we search for a minimum using the iterative method called *gradient descent*. When attempting to find the minimum of a function, gradient descent works: it begins at a random position in the parameter space and then iteratively decreases the $\mathcal{J}$ error until it reaches a local minimum. It defines the direction of the steepest descent at each step of the iteration and takes a step in that direction. This process is depicted as the *backpropagation* principle (Rumelhart et al. (1986); Hecht-Nielsen (1992)) in the following graph in which there is a set of 5 five neural network nodes $w_i, w_{i+1}, w_{i+2}, w_{i+3}, w_{i+4}$, all set in forward pass as in figure :

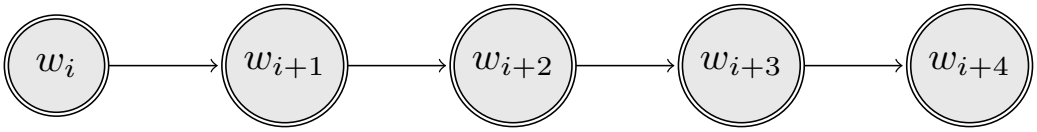


Figure 3.5: A simple graph

By the chain rule, we have

$$\frac{\partial w_{i+4}}{\partial w_i} = \frac{\partial w_{i+4}}{\partial w_{i+1}} \cdot \frac{\partial w_{i+1}}{\partial w_i} = \frac{\partial w_{i+4}}{\partial w_{i+2}} \cdot \frac{\partial w_{i+2}}{\partial w_{i+1}} \cdot \frac{\partial w_{i+1}}{\partial w_i} = \frac{\partial w_{i+4}}{\partial w_{i+3}} \cdot \frac{\partial w_{i+3}}{\partial w_{i+2}} \cdot \frac{\partial w_{i+2}}{\partial w_{i+1}} \cdot \frac{\partial w_{i+1}}{\partial w_i}$$

As we can see, in order to compute the gradient of w_{i+4} with respect to w_i, we can start at w_{i+4} a go backwards towards w_i, computing the gradient of every node's output with respect to its input along the way until we reach w_i. Then, we multiply them all together.

Now consider the following scenario in figure 3.6. In this case, w_i contributes to w_{i+4} along two paths: The path $w_i, w_{i+1}, w_{i+3}, w_{i+4}$ and the path $w_i, w_{i+2}, w_{i+3}, w_{i+4}$. Hence, we have:

Proposition 3.5.1
The total derivative of w_{i+4} with respect to w_i in figure 3.6 is given by:

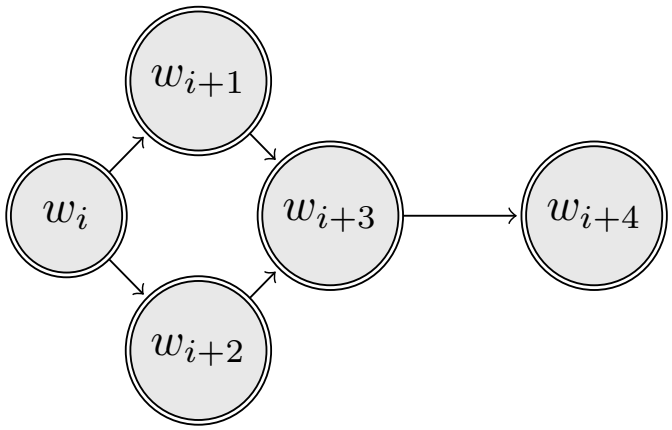


Figure 3.6: A simple graph

$$\begin{aligned}\frac{\partial w_{i+4}}{\partial w_i} = \frac{\partial w_{i+4}}{\partial w_{i+3}} \cdot \frac{\partial w_{i+3}}{\partial w_i} &= \frac{\partial w_{i+4}}{\partial w_{i+3}} \cdot \left(\frac{\partial w_{i+3}}{\partial w_{i+1}} \cdot \frac{\partial w_{i+1}}{\partial w_i} + \frac{\partial w_{i+3}}{\partial w_{i+2}} \cdot \frac{\partial w_{i+2}}{\partial w_i} \right) \\ &= \frac{\partial w_{i+4}}{\partial w_{i+3}} \cdot \frac{\partial w_{i+3}}{\partial w_{i+1}} \cdot \frac{\partial w_{i+1}}{\partial w_i} + \frac{\partial w_{i+4}}{\partial w_{i+3}} \cdot \frac{\partial w_{i+3}}{\partial w_{i+2}} \cdot \frac{\partial w_{i+2}}{\partial w_i}\end{aligned} \tag{3.3}$$

This gives us an intuition for the general algorithm that computes the gradient of the loss function of neural network with respect to any node: we perform what is known as a backwards *breadth-first search* starting from the loss node itself (Beamer et al. (2012); Bundy & Wallen (1984); Zhou & Hansen (2006)). At each node w_i that we visit, we compute the gradient of the loss with respect to w_i itself and all his connected nodes.

Application to neural network optimization: the victory of backpropagation

The goal of the training process is to minimize the loss $\mathcal{L}(\boldsymbol{\Theta})$ with respect to the parameters in $\boldsymbol{\Theta}$. The most popular algorithm to find good sets of parameters $\boldsymbol{\Theta}$ is stochastic gradient descent (SGD) which starts with some random initial values of $\boldsymbol{\Theta} = \boldsymbol{\Theta}_0$ and then updates $\boldsymbol{\Theta}$ iteratively as:

$$\boldsymbol{\Theta}_{t+1} = \boldsymbol{\Theta}_t - \eta \nabla_{\boldsymbol{\Theta}} \tilde{\mathcal{L}} \left(\boldsymbol{\Theta}_t \right) \tag{3.4}$$

where $\eta > 0$ is the learning rate and $\tilde{\mathcal{L}}(\boldsymbol{\Theta})$ is an approximation of the loss function which is computed for a random minibatch of training examples $\mathcal{S}_t \subset \{1, 2, \ldots, S\}$ of size $\mathcal{B}_t$ at each iteration. By choosing $|S_t| = \mathcal{B}_t$ small compared to S, the gradient computation complexity is significantly reduced while still reducing weight update variance. By applying this to our neural network defined in [3.1.1], and using the partial derivative on the cost function $\mathcal{J}(\boldsymbol{\mathcal{W}}, \boldsymbol{b})$, one can compute the gradient in which the weight and bias have to descent to minimize it.

$$\boldsymbol{\mathcal{W}}_{ij}^{(l)} = \boldsymbol{\mathcal{W}}_{ij}^{(l)} - \alpha \frac{\partial}{\partial \boldsymbol{\mathcal{W}}_{ij}^{(l)}} \mathcal{J}(\boldsymbol{\mathcal{W}}, \boldsymbol{b})$$

$$\boldsymbol{b}_i^{(l)} = \boldsymbol{b}_i^{(l)} - \alpha \frac{\partial}{\partial \boldsymbol{b}_i^{(l)}} \mathcal{J}(\boldsymbol{\mathcal{W}}, \boldsymbol{b})$$

Where α determine the amount of the gradient to be used, called learning rate, and discussed more formally further [3.6.2]. The backpropagation principle allows us to compute $\frac{\partial}{\partial \boldsymbol{b}_i^{(l)}} \mathcal{J}(\mathcal{W}, \boldsymbol{b})$ at every layer of the neural network using proposition 3.5.1. We then have a rough idea how to minimize $\mathcal{J}$ the loss function of our model (Yu et al. (2002); Wilamowski et al. (2001); Abid et al. (2001)):

Theorem 3.5.1 (Neural Network Error Function Derivatives)
The derivation of the backpropagation algorithm begins by applying the chain rule in equation 3.3 to the error function $\mathcal{J}(\mathcal{W}, \boldsymbol{b})$ partial derivatives:

$$\frac{\partial \mathcal{J}(\mathcal{W}, \boldsymbol{b})}{\partial \mathcal{W}_k} = \frac{\partial \mathcal{J}(\mathcal{W}, \boldsymbol{b})}{\partial a_k} \frac{\partial a_k}{\partial \mathcal{W}_k}$$

where a_k is the activation (product-sum plus bias) of node j in layer k before it is passed to generate the next layer output.

Proposition 3.5.2 (Backpropagation algorithm & Convergence, Wu et al. (2008))
The following Backpropagation algorithm :

1. *Start with some values for $\mathcal{W}$ and $\boldsymbol{b}$ taken at random;*
2. *Compute the gradients of $\mathcal{J}$ with respect to $\mathcal{W}$ and $\boldsymbol{b}$*
3. *Take a small step along the direction of the negative gradient of $\mathcal{J}$ using equation 3.4 update.*
4. *Go back to 2*

is convergent under the conditions of proposition 1.2.1.

3.6 Talks on Deep Learning common regularization methods

One thing to remember is that while we demonstrated the presence of spurious local minima in the entire parameter space, things will vary in small parameter space sets, for example, by adding regularizers (Yun et al., 2018).

In a machine learning algorithm, a common approach to minimise overfitting is to use a regularisation concept that penalises large weights (L2) or non-sparse weights (L1), etc. There are three other methods much more with the same objective that should be known:

1. Cross-Validation: it is a multiple-round validation in its simplest form, where we leave a sample as in-time for validation and rest the other parts for model training. A higher fold cross-validation is favoured to ensure a lower variance.

2. Early Stopping: Early stopping guidelines offer instructions about how many iterations can be carried out before the model starts to *over-fit*.

3. Pruning: When building Classification And Regression Trees (CART) model templates, pruning is used extensively. It essentially excludes the nodes that give no predictive strength to the output

at hand. (Ghasemain et al. (2020); Pham et al. (2019); Martínez-Muñoz et al. (2008); Cui & Feng (2019); Frank (2000)).

A Couple of Hyper-parameters that needs to be tuned

A number of hyperparameters, such as the number of hidden neurons or layers (Yuan et al. (2003)), and training iterations (called *epochs*) and the batch size, need to be tuned with the MultiLayer Perceptron. The batch size parameter $\mathcal{B}$ is one of the main ones of the hyper-parameters that will be tuned as you train a Stochastic Gradient Descent mini-batch neural network (SGD)[see Section 1.2.3]. The most simple hyper-parameter search approach is to do a grid search to find a pair that lets the network converge w.r.t the learning rate and batch size. (Neelakantan et al. (2015); Shang & Wah (1996); Hinton et al. (2012b); Fukumizu (1998); Liang et al. (2006); Masters & Luschi (2018); Bottou & Cun (2003); Sahoo et al. (2017); Murata (1998)).

Definition 3.6.1 (The Batch Size)
The batch size $\mathcal{B}$ defines how many examples you are looking at before updating the weight. The smaller it is, the noisier the training signal will be, the greater it will be, the longer it will take for each step to measure the gradient.

Another major parameter for the SGD described above is the learning rate:

Definition 3.6.2 (The learning rate schedule)
This function $\epsilon(s) : \mathbb{N} \to \mathbb{R}$ is called the learning rate schedule.

The parameter $\epsilon(s)$ can be interpreted as a function of the count of epochs s and is the learning rate (Darken & Moody (1991); Darken et al. (1992)). If you want to fix the learning rate, just take epsilon as a constant function. One famous application of the learning rate schedule is the Adam Optimizer (Kingma & Ba (2014)).

It's important to see the relationship between batch gradient descent, online SGD, and mini-batch SGD to understand what the batch size should be. Here is the general formula for the mini-batch SGD weight update step, which is a generalisation of all three kinds of it (Tsoi (1997); Bengio (2012); Ventresca & Tizhoosh (2009); Bottou (1991); Du et al. (2019); Andrychowicz et al. (2016); Hochreiter et al. (2001)).

$$\mathbf{\Theta}_{s+1} \leftarrow \mathbf{\Theta}_s - \epsilon(s)\frac{1}{\mathcal{B}}\sum_{b=0}^{\mathcal{B}-1}\frac{\partial \mathcal{L}\left(\mathbf{\Theta}, \mathbf{m}_b\right)}{\partial \mathbf{\Theta}}$$

where $\mathbf{m}_b$ is a subset of the training data. More details follow:

1. Batch gradient descent, $\mathcal{B} = |\mathbf{m}_b|$; In this case, The batch size $\mathcal{B}$ is just the cardinality of $\mathbf{m}_b$: $\mathcal{B} = |\mathbf{m}_b|$. SGD converges faster than standard Batch gradient descent, because after looking at a randomly chosen subset of the training set, it changes the weights. (Lin & Zhou (2017); Li et al. (2014a); Li & Orabona (2019)).

2. For online stochastic gradient descent: $\mathcal{B} = 1$, For Mini-batch stochastic gradient descent: $\mathcal{B} > 1$ and $\mathcal{B} < N_b$ (N_b is set for the size of the whole training data),

Using the gradients of the whole dataset x, batch gradient descent updates the weights $\boldsymbol{\Theta}$; while SGD updates the weights using the gradient average for the mini-batch $\mathbf{m}_b$. The predicted value of this stochastic gradient approximation used in Online SGD refers to the deterministic gradient used in the descent of the batch gradient.

Theorem 3.6.1 (Expected Online SGD)
In Online SGD, we have :

$$\mathbb{E}_{\mathbf{m}\sim U}\left[\nabla \mathcal{L}_{SGD}(\boldsymbol{\Theta}, \mathbf{m})\right] = \nabla \mathcal{L}(\boldsymbol{\Theta}, \mathbf{x})$$

where $\mathbf{m} \sim U$ *means* $\mathbf{m}$ *is taken uniformly over the data set.*

Proof 3.6.1
We show this as follows:

$$\begin{aligned}
\mathbb{E}_{\mathbf{m}\sim U}\left[\nabla \mathcal{L}_{SGD}(\boldsymbol{\Theta}, \mathbf{m})\right] &= \nabla \mathbb{E}_{n\sim U}\left[\mathcal{L}_{SGD}(\boldsymbol{\Theta}, \mathbf{m})\right] \\
&= \nabla \sum_{i=1}^{N} P(\mathbf{m} = i)\mathcal{L}_{SGD}(\boldsymbol{\Theta}, i) \\
&= \nabla \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}_{SGD}(\boldsymbol{\Theta}, i) \\
&= \nabla \mathcal{L}(\boldsymbol{\Theta}, \mathbf{x})
\end{aligned}$$

because $\mathcal{L}_{SGD}(\boldsymbol{\Theta}, i)$ *is the log-likelihood evaluated at the data point* i*. This ends the proof.*

> Each time we take a sample and update our weights it is called a mini-batch. Each time we run through the entire dataset, it's called an *epoch*.

Let's assume we have a data vector called $\mathbf{x} \in \mathbb{R}^D$, an initial weight vector that represents the parameters of our neural network, $\boldsymbol{\Theta}_0 \in \mathbb{R}^S$, and a loss function called $\mathcal{L}(\boldsymbol{\Theta}, \mathbf{x})$ that we use to optimize the network. If we have Ξ training examples and a batch size of $\mathcal{B}$, then we can split those training examples into $\boldsymbol{\iota}$ mini-batches:

$$\boldsymbol{\iota} = \lceil \Xi / \mathcal{B} \rceil$$

We should say, for convenience, that Ξ is uniformly divisible by the $\mathcal{B}$. But, where this is not the case, because sometimes it is not, as a function of its scale, proper weight should be allocated to each mini-batch.

Proposition 3.6.1 (Iterative SGD with $\mathfrak{O}$ epochs)

An iterative algorithm for SGD with $\mathfrak{O}$ epochs is given below:

$$
\begin{aligned}
& s \leftarrow 0 \\
& \textit{while } s < \mathfrak{O} \\
& \quad \boldsymbol{\Theta}_{s+1} \leftarrow \boldsymbol{\Theta}_s - \epsilon(s) \frac{1}{\mathfrak{B}} \sum_{b=0}^{\mathfrak{B}-1} \frac{\partial \mathcal{L}\left(\boldsymbol{\Theta}, \mathbf{m}_b\right)}{\partial \boldsymbol{\Theta}} \\
& \quad s \leftarrow s + 1
\end{aligned}
$$

> Note: The dataset is shuffled by most SGD implementations and then the examples get loaded into memory in order to be interpreted.

This iterative algorithm is improved in Adam optimizer (Kingma & Ba (2014)) presented as follows:

1. Initialization : $\boldsymbol{\Theta}_0$: Initial parameter vector, $b_0 \leftarrow 0$ (Initialize 1^{st} moment vector), $a_0 \leftarrow 0$ (Initialize 2^{nd} moment vector), $s \leftarrow 0$ (Initialize timestep);
2. $s \leftarrow s + 1$
3. Get gradients w.r.t. stochastic objective at timestep s: $g_s \leftarrow \nabla_{\boldsymbol{\Theta}} ; \tilde{\mathcal{L}}\left(\boldsymbol{\Theta}_{s-1}\right)$
4. Update biased first moment estimate: $b_s \leftarrow e_1 \cdot b_{s-1} + (1 - e_1) \cdot g_s$
5. Update biased second raw moment estimate: $a_s \leftarrow e_2 \cdot a_{s-1} + (1 - e_2) \cdot g_s^2$;
6. Compute bias-corrected first moment estimate: $\widehat{b}_s \leftarrow b_s / \left(1 - e_1^s\right)$;
7. Compute bias-corrected second raw moment estimate: $\widehat{a}_s \leftarrow a_s / \left(1 - e_2^s\right)$;
8. Update parameters: $\boldsymbol{\Theta}_s \leftarrow \boldsymbol{\Theta}_{s-1} - \zeta \cdot \widehat{b}_s / \left(\sqrt{\widehat{a}_s} + \epsilon\right)$;

where $e_1, e_2 \in [0, 1)$ are the exponential decay rates for the moment estimates. They recommand a good default settings for the tested machine learning problems are $\zeta = 0.001$ (ζ is the stepsize here), $e_1 = 0.9, e_2 = 0.999$ and $\epsilon = 10^{-8}$.

Hold-out and Cross-validation

Hold-out is when the dataset is broken into a series of 'train' and 'test.' When the dataset is arbitrarily broken up into 'k' classes, cross-validation or 'k-fold cross-validation' is. As the evaluation set, one of the classes is used and the others are used as the testing set. On the training package, the model is conditioned and graded on the test set. Then, before each particular category is used as the test set, the procedure is replicated (Cooil et al. (1987), Berrar (2019), Hawkins et al. (2003)).

Cross-validation is typically the chosen approach because it eliminates the bias in model estimation and allows it the ability to practice on several splits of train evaluations. (Bengio & Grandvalet (2004), Jung & Hu (2015), Blum et al. (1999), Yadav & Shukla (2016)). This gives you a clearer idea of how well on unseen data the model can do. Hold-out, on the other hand, depends on only one split train test, which makes the score of the hold-out system based on how the knowledge is separated into train

and test sets. A web-blogger on machine learning topics [2] once said: Bear in mind that it takes more processing resources and time to run than using the holdout process, as cross-validation requires many train-test breaks. You will get more into it by reading this book titled ***Note on Free Lunches and Cross-Validation*** written by Goutte (1997) which introduces The "no-free-lunch" theorems, and share knowledge of matter on the cross-validation method.

3.7 A short example: The How it works

This short illustrative example comes from Scikit Learn Python Library User Guide[3].

Given a set of training examples $(\boldsymbol{u}_1, \boldsymbol{v}_1), (\boldsymbol{u}_2, \boldsymbol{v}_2), \ldots, (\boldsymbol{u}_n, \boldsymbol{v}_n)$ where $\boldsymbol{u}_i \in \mathbf{R}^n$ and $\boldsymbol{v}_i \in \{0, 1\}$, a one hidden layer one hidden neuron MLP learns the function $f(\boldsymbol{u}) = \boldsymbol{\mathcal{W}}_2 g\left(\boldsymbol{\mathcal{W}}_1^T \boldsymbol{u} + \boldsymbol{b}_1\right) + \boldsymbol{b}_2$ where $\boldsymbol{\mathcal{W}}_1 \in \mathbf{R}^m$ and $\boldsymbol{\mathcal{W}}_2, \boldsymbol{b}_1, \boldsymbol{b}_2 \in \mathbf{R}$ are model parameters. $\boldsymbol{\mathcal{W}}_1, \boldsymbol{\mathcal{W}}_2$ Represents the weights, respectively, of the input layer and hidden layer; and $\boldsymbol{b}_1, \boldsymbol{b}_2$ represent the bias applied to the hidden layer and output layer, respectively. $g(\cdot) : R \rightarrow R$ is the activation function set as the hyperbolic tan . It is given as,

$$g(\boldsymbol{z}) = \frac{e^{\boldsymbol{z}} - e^{-\boldsymbol{z}}}{e^{\boldsymbol{z}} + e^{-\boldsymbol{z}}}$$

For binary classification, $f(\boldsymbol{u})$ passes through the logistic function $g(\boldsymbol{z}) = 1/\left(1 + e^{-\boldsymbol{z}}\right)$ to receive values of output between zero and one. A 0.5 threshold will allocate samples of outputs greater than or equal to 0.5 to the positive class, and the remainder to the negative class.

If there are more than two classes, $f(\boldsymbol{u})$, instead of passing through the logistic feature, it passes through the softmax function, which is written as a vector of size n classes [see 3.3]:

$$\text{softmax}(\boldsymbol{z})_i = \frac{\exp\left(\boldsymbol{z}_i\right)}{\sum_{l=1}^{k} \exp\left(\boldsymbol{z}_l\right)}$$

where $\boldsymbol{z}_i$ represents the i-th element of the input to softmax, which corresponds to class i, and K is the number of classes. The consequence is a vector representing the probabilities of each class having the sample $\boldsymbol{u}$. The class with the highest likelihood is the output. The output stays as $f(\boldsymbol{u})$ in regression; thus, the output activation function is just the identity function. Depending on the problem form, MLP uses various loss functions. In regression, the output remains as $f(\boldsymbol{u})$; therefore, output activation function is just the identity function. MLP uses different loss functions depending on the problem type. Cross-Entropy is the loss function for classification, which in binary cases is given as:

$$\text{Loss}(\hat{\boldsymbol{v}}, \boldsymbol{v}, \boldsymbol{\mathcal{W}}) = -\boldsymbol{v} \ln \hat{\boldsymbol{v}} - (1 - \boldsymbol{v}) \ln(1 - \hat{\boldsymbol{v}}) + \alpha \|\boldsymbol{\mathcal{W}}\|_2^2$$

Where $\alpha \|\boldsymbol{\mathcal{W}}\|_2^2$ is an L2-regularization (aka punishment) word that penalises complex models, and $\alpha > 0$ is a non-negative penalty magnitude regulation hyperparameter. MLP uses the Square Error loss function to regress; written as:

[2] Eijaz Allibhai.

[3] https://scikit-learn.org/stable/modules/neural_networks_supervised.html

$$\text{Loss}(\hat{\boldsymbol{v}}, \boldsymbol{v}, \boldsymbol{\mathcal{W}}) = \frac{1}{2}\|\hat{\boldsymbol{v}} - \boldsymbol{v}\|_2^2 + \frac{\alpha}{2}\|\boldsymbol{\mathcal{W}}\|_2^2$$

The multi-layer perceptron (MLP) minimises the loss function by continuously updating these weights starting from initial random weights. A backward pass propagates it from the output layer to the previous layers after calculating the loss, supplying each weight parameter with an update value intended to reduce the loss. The gradient $\nabla\text{Loss}_{\boldsymbol{\mathcal{W}}}$ of the weight loss is determined in the gradient descent and removed from $\boldsymbol{\mathcal{W}}$. This is, more formally, expressed as:

$$\boldsymbol{\mathcal{W}}^{i+1} = \boldsymbol{\mathcal{W}}^{i} - \epsilon\nabla\text{Loss}^{i}_{\boldsymbol{\mathcal{W}}}$$

where i is the iteration step, and ϵ is the learning rate with a value larger than 0. The algorithm stops when the full number of iterations is reached by a preset; or when the loss improvement is below a fixed, limited number.

3.8 Extended Notes on Pruning Method

Pruning is a strategy guided by a pruning saliency, which heuristically approximates the change in the loss related with the removal of explicit weights. Many pruning signals have been proposed, but the performance of each heuristic relies upon each specific neural network. Pruning was achieved specifically by using magnitude as a saliency approximation to assess the weights that are less useful, the intuition is that lower weights magnitude has a smaller performance effect, and thus if pruned, they are less likely to have an effect on the model predictions. In **Optimal Brain Damage**, LeCun et al. (1990) suggested that various metrics could be a better estimate of saliency, and has proposed to use the second derivative of the objective function as a pruning measure with respect to the parameters.

A widely cited paper has been released suggesting a three-step procedure to prune neural networks (Han et al. (2015)):

1. Train a dense network;
2. Using magnitude as a surrogate for saliency, prune the less meaningful weights;
3. Retrain the network to fine-tune the remaining weights links.

Surprisingly, they showed that not only was this approach equal to standard deep learning models, but it was able to achieve much greater precision.

Weight pruning techniques: The Case of the Optimal Brain Damage (OBD) Method

In the OBD process, pruning is based on the network weights saliency measure as introduced above. Let $\mathcal{J}$ be the cost function that is used for network learning process, $\boldsymbol{\mathcal{W}}_i$ be a generic network weight. The saliency of $\boldsymbol{\mathcal{W}}_i$ defined by the OBD method is

$$S\left(\boldsymbol{\mathcal{W}}_i\right) \approx \frac{\partial^2 \mathcal{J}}{\partial \boldsymbol{\mathcal{W}}_i^2}\boldsymbol{\mathcal{W}}_i^2$$

Note that $\partial^2 \mathcal{J}/\partial \mathcal{W}_i^2$ is a diagonal element of the Hessian matrix. The OBD system pruning procedure is as follows: first, the network is trained and the saliency of all the weights are determined using 3.8; then the weights with the smallest saliency are removed and the network with reduced size is retrained. Before having a final pruned network model, the process may have to be repeated several times (Cornelius, 1997).

Pruning with Graphical Models

We will be dealing with binary random variables throughout this part. As Johnson et al. (2015), we write $P(\boldsymbol{y})$ denoting the distribution of probability of a set of random variables $\boldsymbol{y} = (\boldsymbol{y}_1, \ldots, \boldsymbol{y}_n)$ We work with undirected graphs, unless specified otherwise, $\mathcal{G} = (\mathbb{\Gamma}, \mathbb{\Lambda})$ with vertex (or node) set $\mathbb{\Gamma}$ and edges $\{i, j\} \in \mathbb{\Lambda} \subset \begin{pmatrix} \mathbb{\Gamma} \\ 2 \end{pmatrix}$. For vertices $i, j \in \mathbb{\Gamma}$ we write $\mathcal{G} + ij$ to denote the graph $(\mathbb{\Gamma}, \mathbb{\Lambda} \cup \{i, j\})$. A pairwise graphical model is a probability distribution $P(\boldsymbol{y}) = P(\boldsymbol{y}_1, \ldots, \boldsymbol{y}_n)$ that is defined on a graph $\mathcal{G} = (\mathbb{\Gamma}, \mathbb{\Lambda})$ with vertices $\mathbb{\Gamma} = \{1, \ldots, n\}$ as:

$$P(\boldsymbol{y}) \propto \prod_{i \in V} \psi_i(\boldsymbol{y}_i) \prod_{\{i,j\} \in E} \psi_{ij}(\boldsymbol{y}_i, \boldsymbol{y}_j) \propto \exp\left\{ \sum_{i \in V} f_i(\boldsymbol{y}_i) + \sum_{\{i,j\} \in E} f_{ij}(\boldsymbol{y}_i, \boldsymbol{y}_j) \right\} \tag{3.5}$$

where $\psi_i, \psi_{ij} \geq 0$ are non-negative node and edge compatibility functions. For positive ψ's, we may also represent $P(\boldsymbol{y})$ as a Gibbs distribution with potentials $f_i = \log \psi_i$ and $f_{ij} = \log \psi_{ij}$.

Using this graphical model to prune neural network weights $(\mathcal{W}_{ij})_{ij}$ means that you set :

$$\hat{\mathcal{W}}_{ij} = \begin{cases} \mathcal{W}_{ij}, & \text{with } P(\mathcal{W}_{ij}) \\ 0, & \text{otherwise} \end{cases}$$

where $\hat{\mathcal{W}}_{ij}$ is a pruned weight, $\mathcal{W}_{ij}$ is a real weight before pruning, and $P(\mathcal{W}_{ij})$ the probability of keeping a weight, which follows a binary graphical model on the set $\mathcal{N}_i$ of all $\mathcal{W}_{ij}$ neighbors in the network. It means that $\mathcal{W}_{ij}$ is pruned if the restricted graph generated with $\{\mathcal{W}_{ij}\} \cup \mathcal{N}_i$ is in a certain state. One particular example of interest would be the Ising model case. As a reminder, an Ising model on binary random variables $\boldsymbol{y} = (\boldsymbol{y}_1, \ldots, \boldsymbol{y}_n)$ and graph $\mathcal{G} = (\mathbb{\Gamma}, \mathbb{\Lambda})$ is the probability distribution defined by

$$P(\boldsymbol{y}) = \frac{1}{Z(\theta)} \exp\left\{ \sum_{i \in V} \theta_i \boldsymbol{y}_i + \sum_{\{i,j\} \in E} \theta_{ij} \boldsymbol{y}_i \boldsymbol{y}_j \right\}$$
$$Z(\theta) = \sum_{\boldsymbol{y}} \exp\left\{ \sum_{i \in V} \theta_i \boldsymbol{y}_i + \sum_{\{i,j\} \in E} \theta_{ij} \boldsymbol{y}_i \boldsymbol{y}_j \right\}$$

where $\boldsymbol{y}_i \in \{-1, 1\}$. The partition function $Z(\theta)$ serves to normalize the probability distribution.

3.9 Effective Python Implementation of the model

The full Python Valid Code of the Multilayer Feed Forward Neural Network is available with comments on our github repository under GNU General Public license $\boldsymbol{v3.0}$. Please Send a request of access on the link : `https://github.com/kgalahassa/MultilayerFeedFeedForward-NN/upload/main` or write directly to alahassan@dms.umontreal.ca for more details. In the following algorithm description, the symbol (\) is a line break.

Algorithm 1 Efficient N_k H-layer FeedForward Neural Network Architecture

procedure HLAYERFFNN(x, y, Decay, LearningRate, Epochs, BatchSize) ▷ The N_k H-layer FFNN

▷ Using Cross-Validation heuristic, split (x, y) in training and testing data.

Create xtraindata & ytraindata ; Create xtestdata & ytestdata

▷ Prepare the placeholders for tensorflow

```
Xfill = Placeholder(Float32, shape= [None, xtraindata.shape[1]])
# Predictions will go here
Yfill = Placeholder(Float32, shape=[None, ytraindata.shape[1]])
```

Create a pruning variable $pkeep$; $p \leftarrow$ xtraindata.shape[1]; Set layers size $(l_1, l_2, ..., l_k)$.

for $i \leftarrow 1$ to $NHLayer$ **do**

$L_1 \leftarrow l_1; W_1 \leftarrow Var([p, L_1]); B_1 = Var([L_1])$;

Y_1 = Relu(Matmul(Xfill, W_1) + B_1); Y_{1d} = Pruning(Y_1, pkeep)

$L_2 \leftarrow l_2; W_2 \leftarrow Var([L_1, L_2]); B_2 = Var([L_2])$;

Y_2 = Relu(Matmul(Y_{1d}, W_2) + B_2); Y_{2d} = Pruning(Y_2, pkeep) ▷ Continue up to L_k layers

$L_k \leftarrow l_k; W_k \leftarrow Var([L_{k-1}, L_k]); B_k = Var([L_k])$;

Y_k = Relu(Matmul($Y_{(k-1)d}$, W_k) + B_k); Y_{kd} = Pruning(Y_k, pkeep)

end for

if Decay ==Constant **then**

LR = LearningRate

else

Set $MinLR$ = max_learning_rate (0.003); Set $MaxLR$ =min_learning_rate (0.0001);

Set DS = decay_speed (100.0)

LR = $MinLR$ + ($MaxLR$ - $MinLR$) * Exp(-i/DS)

end if

Require: Set the final layer $Y\_$ = Matmul(Y_{kd}, W_{k+1}) + B_{k+1}; Set Error = $MSE(Yfill, Y\_)$; Set Adam Stochastic Optimizer

Train_step = Train.AdamOptimizer(lr).minimize(Error)

lr is a Placeholder for your learning rate.

for $i \leftarrow 1$ to Epochs **do**

Shuffle the data to select random minibatch

$j = 1$

▷ You may shuffle using a shuffling function.

X_datasf, Y_datasf = Shuffle(X_datatrain, Y_datatrain)

for BatchIndex in MiniBatches **do** ▷ There come the batch learning

X_batch,Y_batch = X_datasf[BatchIndex],Y_datasf[BatchIndex];

$j+ = 1$

_,TrainLoss = RunFFNN([Train_step, MSE], feed_dict= { Xfill: X_batch, Yfill: Y_batch, pkeep: 0.75, lr: LR }) ▷ You may evaluate test error at each step. ▷ You can add Stoploss, StoppingCriteria.

end for

end for

end procedure

3.10 Performance and Comparison with Random Forest

We have performed experiments with five (5) multivariate multiple regression datasets taken from the multiple-output benchmark datasets available in the Mulan project website (Tsoumakas et al., 2020).

Mulan Project DataSets

The Mulan project dataset (Tsoumakas et al., 2020) is shown in Table 2.1. We benchmark our results in terms of Mean Squared Error (MSE) with Random Forest (RF) (Pal, 2005) in two scenarios:

- $RandF_1$: number_estimators $= 10$, maximum_depth $= 30$;
- $RandF_2$: number_estimators $= 5$, maximum_depth $= 10$.

Random Forest *max_depth* represents the depth of each tree in the forest. The deeper the tree, the more splits it has and it captures more information about the data.

Random Forest *number_estimators* represents the number of trees in the forest. Usually the higher the number of trees the better to learn the data.

In this experiments, the test set always represents 20% of the whole data. To compare our models in terms of the performance of the feedforward neural network (FFNN), we build two architectures: a one-hidden-layer FFNN, and a four-hidden-layer FFNN. We will refer to these networks as 1 H-layer FFNN (20 hidden neurons maximum) and 4 H-layer FFNN, respectively. We set the size of the four hidden layers as follows: $l_1 = 200, l_2 = 160, l_3 = 100$, and $l_4 = 30$.. For both models, epoch times was fixed to 3500, and learning rate $\epsilon = learning\_rate = 0.00010013298299992624$. The results are presented in table 3.3 and 3.4.

Table 3.3: Summary of Train MSE statistics for five (5) Mulan Project Dataset

	FFNN		Benchmark Models	
Dataset	1-H-layer FFNN	4 H-layer FFNN	$RandF_1$	$RandF_2$
Slump	253.523	103.492	20.329	16.407
EDM	0.184	0.124	0.043	0.032
Jura	35.784	20.846	5.873	5.713
Water quality	1.158	1.085	0.517	0.225
SCPF	13701.031	194.873	1745.208	836.529

Table 3.4 is just a confirmation of how powerful neural network can be at work. The result for EDM dataset can be updated using a more powerful architecture: simply imagine what we could get with hundred layers more added to the system, and a training time (epochs) well adapted using suitable hyper-parameters heuristics and validation methods... Finally, to beat random forest on EDM, we have simply augmented the four hidden layers as follows with one supplementary layer as follow: $l_1 = 500, l_2 = 260, l_3 = 150, l_4 = 40, l_5 = 40$, with a training time (epochs) equal to 7240. The learning rate

Table 3.4: Summary of Test MSE statistics for five (5) Mulan Project Dataset

	FFNN		Benchmark Models	
Dataset	1-H-layer FFNN	4 H-layer FFNN	$RandF_1$	$RandF_2$
Slump	**116.125**	269.500	137.809	137.296
EDM	0.225	0.202	0.185	**0.140**
Jura	**20.204**	22.592	87.179	79.277
Water quality	1.391	**1.313**	1.444	1.388
SCPF	**407.353**	453.266	1087.353	1198.551

stays the same. We get with this final architecture: Train MSE $= 0.033$, and Test MSE $= 0.133$. Our results have been cross-validated, and this model (with N_k H-layer FFNN) is universally unbeatable. The N_k H-layer FFNN model is the *one-of-a-kind* model that will transform neural networks statistical learning forever.

3.11 The odds and the Even of neural networks

Neural networks are more stable and can be used for problems of regression as well as grouping. With a huge number of inputs such as photos for example, neural networks are also good for the nonlinear predictors (Howard (2013); Kanellopoulos & Wilkinson (1997); Li et al. (2014b); Park et al. (2004)). With any number of inputs and layers, neural networks can operate. A well-known researcher in neural networks once said that: ***"A neural network is the second fastest way to solve any problem. The best way is to really consider the problem"***.

Chapter 4

Shallow Potts Neural Network Mixture Models

We introduce a novel ensemble learning approach which combines random partitions models with a non-parametric predictor such as Multilayer feedforward networks. Neural networks are known as universal approximators (Hornik et al. (1989) and Cybenko (1989)), and are very well suited to explore other learning methods. We combine them with Potts clustering models to create a bagging-boosting-averaging-like learning framework where several estimates from each random partition are aggregated into one prediction. Our approach carries out the balance between overfitting and model stability in presence of higher dimensional data. More precisely, our model merges Potts model in a *multivariate multiple regression* task with Bayesian deep learning models to produce the ensemble learning method. Potts model clustering has been introduced to the statistical community by Murua et al. (2008b). The model is applied to the set of covariate values, formalizing a proximity co-clustering. The method has been used and proven to be effective (see Murua & Wicker (2014b); Murua & Quintana (2017c)).

The model called *Structured Potts Shallow Gibbs Neural Network* will be a hierarchical Bayesian model where we train individual neural nets to specialize on sub-groups (latent clusters components) while still being informed about representations of the overall population. Our Potts neural network model differs from those of Kanter (1988) and Philipsen & Cluitmans (1993), which is a generalization of the Ising neural network. We call it a structured one, because we integrate the structured correlations among the weights (and offsets) of the network (Sun et al., 2017) through a Markov Random Fields (MRF) process. Bayesian learning allows the opportunity to quantify posterior uncertainty on neural networks (NNs) model parameters. We can specify priors to inform and constrain our models and get structured uncertainty estimation.

4.0.1 Efficiency of regression clustering

Regression clustering is a learning algorithm that allows multiple regression settings to be clustered where you have a dependent Y vector. And one or two autonomous (independent) variables, the X's. Recovering the implicit partitioning of observations is the problem of regression clustering: the algorithm then divides the data into two or more clusters and performs an independent multiple regression within each cluster on the data. Many algorithms have emerged to run this approach in the past (see

Späth (1979) and H.-J. (1985)). The ultimate aim is to discover classes without oversight before regression is implemented in each class. Nowadays, many variations have been proposed in machine learning; clustered linear regression[1] (CLR) is an example (Ari & Güvenir, 2002).

Regression clustering has shown a significant predictive advantage in several tasks: clustered regression trees (Torgo & da Costa, 2000), ridge regression clustering (Nagpal et al., 2013), and non-parametric Bayesian clustering regression (see Müller et al. (2008), Yang et al. (2014), and Murua & Quintana (2017c)). The combination of clustering and regression approaches will potentially minimise the possible problems of predictive performance efficacy due to data heterogeneity (Chen et al., 2013). Using a clustered method, the data subsets are created with a degree of homogeneity that improves the accuracy of prediction.

4.0.2 Combination of neural network regression and Potts clustering model

Potts clustering has the ability to integrate multiple partitions from different cluster solutions to improve the robustness, stability, scalability of the clustering algorithms (Murua et al., 2008c). As shown by Murua & Quintana (2017c), Potts clustering can be a good prior for statistical models such as regression. Our model is a *multivariate multiple regression clustering* one, aiming at combining the Potts clustering model with a non-linear multivariate multiple regression tool: neural networks. The first main advantage of the combination of neural networks regression and Potts clustering models, is that a few of the drawbacks of linear regression can be overcome by using artificial neural network (ANN), and *clustered* models can improve prediction accuracy. Their combination presents other key advantages that make them most suitable for certain problems and situations:

1. An artificial neural network is capable of fitting and modelling non-linear and complicated relationships, which is really important since many of the relationships between inputs and outputs are both non-linear and complex in real life.

2. An artificial neural network is a good predictor: it can also infer unseen relations on unseen data after the necessary fitting step from the initial inputs and their relationships, thereby allowing the model to predict on unseen data.

3. Extension of Potts model to various forecasting problems: in addition to exploring a graph-based consensus clustering[2] (GCC) to find cluster structures from heterogeneous data, the model proposed here offer a novel opportunity to couple neural network model with random partition models in diverse machine learning tasks, such as multivariate multiple regression (precisely: when Y response is a **vector**). In fact, Potts clustering as used here, is a *random partition model*, explicitly, a clustering model with prior distribution on partitions (Müller & Quintana, 2010a). Section 4.2 describes Potts clustering in detail.

However, ANN is a black box learning technique that can not interpret input-output relationships and can not cope with uncertainties. Using the Bayesian framework as done by Murua & Quintana (2017c) can overcome uncertainties issue, and the results can be easily compared. The ***Bayesian*** technique is highly important, as traditional neural network training involves a lot of labelled data to monitor the

[1] Clustered linear regression (CLR) is a modern machine learning algorithm that, by partitioning training space into subspaces, increases the precision of classical linear regression.

[2] Potts clustering is based on a consensus clustering approach (Blatt et al., 1996a).

possibility of overfitting. And when it comes to real-world regression assignments, the task gets more complicated. Such exercises (regression) also have less training data to use, which also makes it easy for neural networks to get stuck in overfitting.

A principled approach for solving this problem is Bayesian Neural Networks (see Vehtari & Lampinen (1999), Bishop (1997)). Prior distributions are placed on the neural network weights in Bayesian Neural Networks to consider the model uncertainty. One can fit a predictor by doing Bayesian inference on the weights, which both matches the training data and knows about the volatility of its own estimation on the test data (Blundell et al., 2015).

4.1 Shallow Gibbs networks

Let $\mathcal{D} = \{(y_i, x_i) : i = 1, \ldots, n\}$ denote the complete data, where $\boldsymbol{x}^n = \{x_1, \ldots, x_n\} \subset \mathbb{R}^q$ is the set of input vectors or covariables, and $\boldsymbol{y}^n = \{y_1, \ldots, y_n\} \subset \mathbb{R}^p$, the set of associated responses. The matrix of covariables will be denote by $\boldsymbol{X} = (x_1|x_2|\cdots|x_n)^T \in \mathbb{M}_{n\times q}$ (here and throghout the chapter, the symbol ^{T} will indicate matrix transposition). Our shallow network is, as in the general case, a feedforward network given by the equations

$$h^{(k)} = b^{(k)} + g_{k-1}(h^{(k-1)})W^{(k)}, \qquad k = 0, 1, 2, \tag{4.1}$$

where the layer 0 corresponds to the input data $h^0 = x, b^{(0)} = 0, g_0(x) = x$, the first layer corresponds to the hidden layer with vector output $h^{(1)}$, and the output layer corresponds to the network's predicted values $h^{(2)} = \hat{y}$. The parameters $\{(b^{(k)}, W^{(k)}) : k = 1, 2\}$ are, respectively, the matrices of offsets, also known as biases, and weights. The vector $g_{k-1}(h^{(k-1)})$ denotes a linear (k=2) or non-linear (k=1) function, such as the identity or a logistic sigmoid, that is applied element-wise. The top layer output $h^{(2)}$ is used for making a prediction and is combined with the supervised target y into a loss function $L(h^{(2)}, y)$, which is typically convex in $h^{(2)} = b^{(2)} + g_1(h^{(1)})W^{(2)}$. As in any regression model, the network's predicted value $h^{(2)}$ is an estimate of $f(y) = \mathbb{E}(y|x)$. Our model may be seen as a simplified feedforward network where the weight matrices are constrained to obey a Gibbs distribution whose neighborhood structure is explained below. Figure 4.1 sketches our network architecture.

Let $\psi = (b^{(1)}, W^{(1)}, b^{(2)}, W^{(2)})$ be the parameters of the network. Suppose that the hidden layer contains l_1 nodes. Then the parameter dimensions are as follows: $b^{(1)} \in \mathbb{R}^{l_1}$, $W^{(1)} \in \mathbb{M}_{q\times l_1}$, $b^{(2)} \in \mathbb{R}^p$, and $W^{(2)} \in \mathbb{M}_{l_1\times p}$. To ease the exposure of our model, we will use the notation, $l_0 = q$, $l_2 = p$. Let $\Sigma \in \mathbb{M}_{p\times p}$ be the variance-covariance matrix of y. We suppose that $y|x, \psi, \Sigma$ is distributed as a multivariate normal distribution with mean $f(y) = f_\psi(y) = \mathbb{E}(y|x, \psi)$, and variance Σ. That is, $p(y|x, \psi, \Sigma) = (2\pi)^{-p/2}|\Sigma|^{-1/2}\exp\{-1/2(y - f_\psi(x))'\Sigma^{-1}(y - f_\psi(x))\}$.

Let $w^{(1)} = \mathrm{vec}(W^{(1)})$ be the $q \times l_1$-dimensional vector of stacked columns of the weight matrix $W^{(1)}$. The vector $w^{(2)} = \mathrm{vec}(W^{(2)}) \in \mathbb{R}^{l_1\times p}$ is defined similarly. Let $w = (w^{(1)}, w^{(2)})$ be the vectorized version of the weight matrices $(W^{(1)}, W^{(2)})$. Our model is set into a Bayesian framework by setting a Gaussian Markov random field prior for ψ. The complete model is the similar to the one described in (Sun et al., 2017). But ours presents significant differences in the actual architecture, because we set up two independent fields: one on all the network weights $w = (w^{(1)}, w^{(2)})$, and another one on all the network biases $b = (b^{(1)}, b^{(2)})$. We assume that both random fields are zero-mean Gaussian fields.

Let $\Omega \in \mathbb{M}_{(q+p)l_1\times(q+p)l_1}$ and $\Gamma \in \mathbb{M}_{(l_1+p)\times(l_1+p)}$ be the precision matrices associated with the random field of the weights, and the biases, respectively. Here we assume that the biases are independent, so

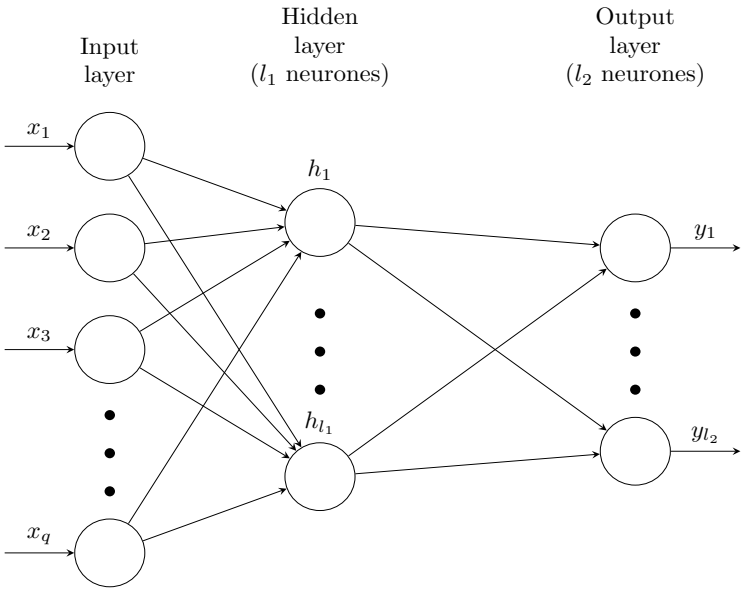


Figure 4.1: Shallow feedforward neural network.

that Γ is a block-diagonal matrix with blocks $\Gamma_1 \in \mathbb{M}_{l_1 \times l_1}$, and $\Gamma_2 \in \mathbb{M}_{p \times p}$. The corresponding prior densities are given by:

$$p(w|\Omega) = (2\pi)^{-\frac{l_1(p+q)}{2}} \det(\Omega)^{\frac{1}{2}} \exp\big(-\frac{1}{2} w^T \Omega w\big), \qquad \text{for the weights, and} \tag{4.2}$$

$$p(b|\Gamma) = (2\pi)^{-\frac{l_1+p}{2}} \det(\Gamma_1)^{\frac{1}{2}} \det(\Gamma_2)^{\frac{1}{2}} \exp\big(-\frac{1}{2}\big\{(b^{(1)})^T \Gamma_1 b^{(1)} + (b^{(2)})^T \Gamma_2 b^{(2)}\big\}\big), \tag{4.3}$$

for the biases. Our framework for this part is completed by assuming an Inverse-Wishart(Λ, ν) prior for the covariance matrix Σ. Here $\Lambda \in \mathbb{M}_{p \times p}$ is the prior scale matrix, and $\nu > 0$ is the prior degrees of freedom.

4.1.1 The sparse-Gibbs network

We define the neighborhood of the random field based on the nodes of the hidden layer. All weights coming or going out of the same node are considered neighbor weights. That is, weights $W^{(1)}_{ij}$ and $W^{(2)}_{st}$ are neighbors if and only if $j = s$, $j, s \in \{1, \ldots, l_1\}$, and weights $W^{(1)}_{ij}$ and $W^{(1)}_{st}$ are neighbors if and only if $j = t$, $j, t \in \{1, \ldots, l_1\}$, and weights $W^{(2)}_{ij}$ and $W^{(2)}_{st}$ are neighbors if and only if $j = t$, $j, t \in \{1, \ldots, p\}$. The matrix Ω is sparse, and composed of l_1 blocks of size $q \times q$, p blocks of size $l_1 \times l_1$, and l_1 blocks of size $q \times p$ as well as the associated transposed blocks of size $p \times q$. Note that the total number of non-zero elements in Ω is $l_1(q^2 + p^2 + 2pq) = l_1(q + p)^2$, which corresponds to a sparsity ratio of $l_1(p + q)^2/(l_1(q + p))^2 = 1/l_1$. This rate is very significative even for small values of l_1.

Let $V_{i\bullet} \in \mathbb{M}_{q \times q}$ be the block of the precision matrix associated with column i of $W^{(1)}$, $i = 1, \ldots, l_1$, $V_{\bullet j} \in \mathbb{M}_{l_1 \times l_1}$ be the block of the precision matrix associated with column j of $W^{(2)}$, $j = 1, \ldots, p$, and $V_{ii} \in \mathbb{M}_{q \times p}$ be the block of the precision matrix associated with column i of $W^{(1)}$, and row i of $W^{(2)}$,

$i = 1, \dots, l_1$. Let also $e_{i;l_1} \in \mathbb{M}_{1\times l_1}$ be the ith row of the $l_1 \times l_1$ identity matrix. Then

$$\Omega = \begin{pmatrix} V_{1\bullet} & 0 & \cdots & 0 & V_{11} \otimes e_{1;l_1} \\ 0 & V_{2\bullet} & \cdots & 0 & V_{22} \otimes e_{2;l_1} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & V_{l_1\bullet} & V_{l_1 l_1} \otimes e_{l_1;l_1} \\ & & & & V_{\bullet 1} \quad 0 \quad \cdots \quad 0 \\ & & & & 0 \quad V_{\bullet 2} \quad \cdots \quad 0 \\ V_{11}^T \otimes e_{1;l_1}^T & \cdots & V_{l_1 l_1}^T \otimes e_{l_1;l_1}^T & & \vdots \quad \vdots \quad \ddots \quad \vdots \\ & & & & 0 \quad 0 \quad \cdots \quad V_{\bullet p} \end{pmatrix} \tag{4.4}$$

where the symbol $\otimes$ stands for the Kronecker product. We will refer to this simplified network as the sparse-Gibbs network, as opposed to the full Gibbs network that imposed no constraints on Ω.

4.1.2 Compound symmetry Gibbs network

Note that the above matrix consists of four blocks, that is $\Omega = \begin{pmatrix} \Omega_1 & \Omega_{12} \\ \Omega_{12}^T & \Omega_2 \end{pmatrix}$, with $\Omega_1 \in \mathbb{M}_{ql_1\times ql_1}$, $\Omega_2 \in \mathbb{M}_{pl_1\times pl_1}$, and $\Omega_{12} \in \mathbb{M}_{ql_1\times pl_1}$. Also, if all matrices $V_{i\bullet}$ were equal, then we could write $\Omega_1 = I_{l_1} \otimes V_{1\bullet}$, where I_r denotes the identity matrix in $\mathbb{M}_{r\times r}$. Similarly, if all matrices $V_{\bullet j}$ were the same, we could write $\Omega_2 = I_p \otimes V_{\bullet 1}$. We could also suppose that all matrices V_{ii} are equal. These simplifications would greatly reduce the number of parameters defining the network.

A less stringent Gaussian random field model consists of replacing I_{l_1} and I_p in the above Kronecker products for more general matrices $U_1 \in \mathbb{M}_{l_1\times l_1}$, $U_2 \in \mathbb{M}_{p\times p}$. This is the second model we consider. However, we simplify it by constraining the inverse matrices of U_1, U_2, $V_{1\bullet}$, and $V_{\bullet 1}$ to have compound symmetry. That is, the inverse matrices of these matrices have to be of the form $\kappa_1(1-\kappa_2)I_{l_1} + \kappa_1\kappa_2 \mathbb{1}\mathbb{1}^T$, for $\kappa_1 > 0$, and $\kappa_2 \in [-1, 1]$, where $\mathbb{1}$ stands for the column matrix of all elements equal to 1. It can easily be shown that this constraint implies matrices of the form $aI_{l_1} + b\mathbb{1}\mathbb{1}^T$, with $(a, b) \in \mathbb{R}^+ \times \mathbb{R}$. The matrices V_{ii} are not constrained. We will refer to this network architecture as sparse compound symmetry Gibbs network or sparse-CS-Gibbs network for short.

Two less structured models. More complex models can be conceived by (a) changing the neighborhood in the Gibbs network, and by allowing the diagonal blocks Ω_{11} and Ω_{22} to be unconstrained. In the Gibbs network this corresponds to consider that all weights coming from the same layer are neighbors, while still keeping the neighborhood restriction on weights coming to and going out of the same node. We will refer to this model as the between-layer sparse-Gibbs network. The second extended model correspond to the compound symmetry network in which the off-diagonal block Ω_{12} is set to be of the form $U \otimes V$, for matrices $U \in \mathbb{M}_{l_1\times l_1}$, and $V \in \mathbb{M}_{q\times p}$, or, as we have actually implemented in our model, for matrices $U \in \mathbb{M}_{l_1\times p}$, and $V \in \mathbb{M}_{q\times l_1}$. Note that the latter matrix structure is always sparser than the former one when $l_1 \in [0, \min\{p, q\}) \cup (\max\{p, q\}, +\infty)$. We will refer to this architecture as the compound-symmetry Gibbs network or CS-Gibbs for short.

4.2 The random-Potts partition model

The purpose of creating the sparse networks of the preceding section is to be able to train the networks with a fraction of the data used by a classical network, As mentioned earlier, our prediction model is a finite but very large mixture of shallow and sparse networks. This is the result of using a non-parametric Bayesian model for prediction. The networks are trained on a subset of data that share similar characteristics in their features. Classical Bayesian non-parametric models rely on the Dirichlet process. However, these processes do not necessarily look at the data features to create clusters or partitions of the data. Recent work has overcome this limitation Müller & Quintana (2010b); Müller et al. (2011b). In particular, the random-Potts partition model introduced in Murua & Quintana (2017c) stresses the importance of data similarities in the resulting partitions by guiding the random partition process through feature characteristics. These are incorporated in the model through a kernel which in turns induces the partition probabilities. The random-Potts partition model may be seen as a stochastic version of the label propagation[3] approach (Tibély & Kertész, 2008).

The random-Potts partition model is a probabilistic model defined on a graph. Keeping the notation introduced before, each data point x_i defines a vertex in a graph whose edges are pre-specified. For example, Murua & Quintana (2017c) consider a k-nearest-neighbor graph. That is, a graph for which there is an edge between data points x_i and x_j if and only if either x_j is one of the k most similar data points to x_i, or x_i is one of the k most similar data points to x_j. Associated with any data point in the graph there is a color label $z_i \in \{1, \ldots, r\}$. Usually r is chosen to be between 10 and 30 (Blatt et al., 1996c; Stanberry et al., 2008b). The set of color labels $\boldsymbol{z}^n = (z_1, \ldots, z_n)$ follows a Potts model, and thus it has probability mass function (pmf)

$$p(\boldsymbol{z}^n|\boldsymbol{x}^n, \beta, \sigma) \propto \exp\{-\beta \sum_{i\sim j} J_{ij}(\sigma)(1 - \mathbf{I}[z_i = z_j])\},$$

where $J_{ij}(\sigma) = J(x_i, x_j; \sigma)$ is the kernel evaluated at (x_i, x_j), $\mathbf{I}[\cdot] = 1$ is the condition between brackets is satisfied, and is zero otherwise, and where the notation $i \sim j$ means that x_i and x_j are neighbors. In the Physics literature the parameter β is referred to as the inverse-temperature. In practice, one chooses the kernel so as to penalize color labelings that do not assign the same color to very similar data points. For example, one can work with $J_{ij} = J_{ij}(\sigma) = J(\|x_i - x_j\|/\sigma)$, where $\|\cdot\|$ denotes the distance (e.g., Euclidean), and $\sigma > 0$ is a bandwidth parameter. The partitions are generated by iteratively deleting edges and inserting deleted edges at random. This is done through a set of random binary variables $\boldsymbol{b} = \{b_{ij}\}$ named the *bonds*. Let $p_{ij} = 1 - \exp(-\beta J_{ij}(\sigma)\mathbf{I}[z_i = z_j, i \sim j])$. Each bond b_{ij} is generated independently of the other bonds as a $\text{Bernoulli}(p_{ij})$. If $i \sim j$, the edge between x_i and x_j remains in the graph with probability p_{ij} (i.e., $b_{ij} = 1$), and it is deleted from the graph with probability $1 - p_{ij}$ (i.e., $b_{ij} = 0$). A random partition ρ_n is generated as the (randomly generated) connected components $\rho_n = \{S_1, ..., S_{k_n}\}$ of this new random graph. In what follows, we will refer to the components of a

[3] A fast algorithm for finding communities in a graph is the Mark Propagation Algorithm (LPA). It defines certain communities as its guide using the network structure alone, and does not require a predefined objective feature or prior group knowledge. LPA is a relatively new algorithm, and was only proposed by Raghavan et al. (2007), in *"Near linear time algorithm to detect community structures in large-scale networks"*. It functions by distributing labels across the network and building communities focused on this label dissemination mechanism. There are some references online : Xie & Szymanski (2011), Xie & Szymanski (2013), Gregory (2010). The paper of Tibely and Kertesz (2008) demonstrates an equivalence of the label propagation method of community detection and Potts model approach.

partition as clusters. It can be shown that this procedure is governed by the random cluster pmf (Sokal, 1997b)

$$p(\boldsymbol{b}|\boldsymbol{x}^n) = [\prod_{b_{ij}=1, i\sim j} p_{ij}][\prod_{b_{ij}=0, i\sim j} (1-p_{ij})]r^{k_n}.$$

Colors are also randomly assigned once the bonds have been chosen. Each connected component is assigned a color uniformly at random. This process of sampling bonds and colors alternatively is known as the Swendsen-Wang algorithm. It is a special case of the Markov chain Monte Carlo (MCMC) stochastic algorithm. Its goal is to generate samples from the random-Potts partition model. This is achieved when the two sampling steps, bonds and colors, have been applied a sufficiently large number of times. The distribution of the random-Potts partitions is given by

$$p(\rho_n|\boldsymbol{x}^n) = \sum_{\mathbf{b}\Rightarrow\rho_n} p(\boldsymbol{b}|\boldsymbol{x}^n) \tag{4.5}$$

where the notation $\boldsymbol{b} \Rightarrow \rho_n$ means that the partition ρ_n arises from the connected components of the associated graph with edges given by the bonds $\boldsymbol{b}$ (see (Murua & Quintana, 2017c) for more in-depth overview). The calculation in the above equation 4.5 might be intractable. Fortunately, we can sample partitions with MCMC methods without knowing these quantities exactly. These will be detailed below in Section 4.5.

4.2.1 Some practical considerations

For our experiments of Section 4.6, we use the Gaussian kernel $J_{ij}(\sigma) = \exp(-\frac{1}{2\sigma^2}\|x_i - x_j\|^2)$, which is the most popular kernel choice for the Potts or super-paramagnetic clustering model (Blatt et al., 1996a,c; Murua et al., 2008b). Although the scale parameter σ may be estimated through a Bayesian stochastic procedure (Murua & Wicker, 2014b), we prefer to use its common estimator which is given by the average distances $\hat{\sigma}^2 = \sum_{i<j}\|x_i - x_j\|^2 / \binom{n}{2}$. Murua & Wicker (2014b) show that the optimal scale is close to this simple estimator. To simplify the computations involved with the Potts model, we restricted the data neighborhood to a k-nearest-neighbors graph; that is, only the k-nearest-neighbors of a given point x_i are considered as neighbors of x_i. This implies that we set $J_{ij}(\sigma) = \exp(-\frac{1}{2\sigma^2}\|x_i - x_j\|^2)$, if x_j is one of the k-nearest-neighbors of x_i, or if x_i is one of the k-nearest-neighbors of x_j, and set $J_{ij}(\sigma) = 0$, otherwise.

One of the main issues encountered with the Potts model is the choice of the inverse temperature parameter β. This parameter controls the cluster sizes in the partitions. A value too low of β produces too many small clusters, while a value too high produces very few and large clusters. Because a shallow Gibbs network is going to be fitted for each cluster of the partition, we would rather not have too small size clusters. Therefore, a small value of β is preferred. In our experiments, for each dataset, we selected a value of β that produced large enough clusters. In general, Murua & Wicker (2014b) gave a simple procedure to find a nearly optimal value of β. In our case, we just chose a value slightly smaller than the value suggested so as to ensure large cluster sizes.

4.3 The shallow Potts Gibbs-network mixture model

Combining the shallow Gibbs networks introduced in Section 4.1 with the random-Potts partition model described in the previous section we obtained the shallow Potts Gibbs-network mixture model. This

model will be referred to as Potts Gibbs-network for short, since "Potts" already conveys the idea of mixture.

Keeping the notation introduced in the previous section, ρ_n denotes a partition of the data, $\psi_\sigma = (w, b, \Sigma)$ denotes the parameters of a shallow network, and $(\Omega, \Gamma, \Lambda, \nu)$ denotes the parameters defining the prior on ψ_σ. For a given partition $\rho_n = (S_1, \ldots, S_{k_n})$, the variables $s_1, \ldots, s_n$ will denote the cluster membership of the data points. That is, $s_i = \ell$ if and only if $x_i \in S_\ell$, $i = 1, \ldots, n$, $\ell = 1, \ldots, k_n$. Note that k_n, the number of clusters or connected components of ρ_n, is a random variable. Proceeding as in (Murua & Quintana, 2017c), we assume a hierarchical random partition model for the data

$$y_1, ..., y_n | \boldsymbol{x}^n, \rho_n, \psi^*_{\sigma 1}, ..., \psi^*_{\sigma k_n} \overset{\text{ind}}{\sim} p(y_i | x_i, \psi^*_{\sigma s_i}) \tag{4.6}$$

$$\psi^*_{\sigma 1}, ..., \psi^*_{\sigma k_n} \overset{\text{ind}}{\sim} p(\psi_\sigma | \Omega, \Gamma, \Lambda, \nu), \text{ and } \rho_n | \boldsymbol{x}^n \sim p(\rho_n | \boldsymbol{x}^n), \tag{4.7}$$

where $(\psi^*_{\sigma 1}, \ldots, \psi^*_{\sigma k_n})$ denote the unique values of $(\psi_{\sigma 1}, \ldots, \psi_{\sigma n})$ given the k_n-cluster partition ρ_n. Thus, $\psi^*_{\sigma i}$ are the parameters of the shallow Gibbs network associated with the i-th data cluster S_i of ρ_n. The notation $\overset{\text{ind}}{\sim}$ indicates that the variables are independent.

As suggested in (Murua & Quintana, 2017c, Section 2.2), avoiding reversible jumps moves (Green, 1995), and following Besag (2004), we can sample from the posterior of ψ_σ by MCMC using these two moves:
(A) Sample a proposed partition $\rho'_n \sim p(\rho'_n | \boldsymbol{x}^n)$, and evaluate

$$\alpha = p(\boldsymbol{y}^n | \rho'_n, \boldsymbol{x}^n) / p(\boldsymbol{y}^n | \rho_n, \boldsymbol{x}^n); \tag{4.8}$$

if $\alpha \geq 1$, accept the proposed partition; otherwise, accept the proposed partition with probability α.
(B) If the proposed partition is accepted, sample (and accept) a new set of parameters $\boldsymbol{\psi}^{*'}_\sigma = (\psi'_{\sigma 1}, \ldots, \psi'_{\sigma k'_n})$ from the posterior given the partition $p(\boldsymbol{\psi}_\sigma | \boldsymbol{y}^n, \boldsymbol{x}^n, \rho'_n)$, where $\rho'_n = (S'_1, \ldots, S'_{k'_n})$. If the proposed partition is not accepted, then keep the current partition ρ_n and the corresponding parameters $\boldsymbol{\psi}^*_\sigma = (\psi_{\sigma 1}, \ldots, \psi_{\sigma k_n})$.

We will refer to the above steps as Besag's algorithm. To use this scheme, we need to be able (i) to sample from the posteriors $p(\boldsymbol{\psi}_\sigma | \boldsymbol{y}^n, \boldsymbol{x}^n, \rho_n)$, and (ii) to compute the ratio α. In general, the probabilities $p(\boldsymbol{y}^n | \rho'_n, \boldsymbol{x}^n)$ are intractable. In fact, computing α is equivalent to computing Bayes factors. This algorithm works exactly only for certain problems such as the multivariate multiple regression with conjugate priors used in (Murua & Quintana, 2017c). In fact, if $p(\boldsymbol{\psi}_\sigma | \boldsymbol{y}^n, \boldsymbol{x}^n, \rho_n)$ is known, we can compute the marginal probabilities using the identity

$$p(\boldsymbol{y}^n | \rho_n, \boldsymbol{x}^n) = p(\boldsymbol{y}^n | \boldsymbol{\psi}_\sigma, \rho_n, \boldsymbol{x}^n) p(\boldsymbol{\psi}_\sigma | \rho_n) / p(\boldsymbol{\psi}_\sigma | \boldsymbol{y}^n, \boldsymbol{x}^n, \rho_n). \tag{4.9}$$

In our case, the posteriors $p(\psi_{\sigma j} | \boldsymbol{y}^n, \boldsymbol{x}^n, \rho_n)$ are proportional to

$$p(T_j | S_j, \psi_{\sigma j})\, p(w_j, b_j, \Sigma_j | \Omega, \Gamma, \Lambda, \nu) = \prod_{x_i \in S_j} p(y_i | S_j, \psi_{\sigma j})\, p(w_j, b_j, \Sigma_j | \Omega, \Gamma, \Lambda, \nu),$$

where $T_j = \{y_i : x_i \in S_j\}$. The normalizing constants of these posteriors are not known because of the non-linearity in the neural network likelihood $p(y_i | S_j, \psi_{\sigma j})$. To avoid integrating over the posterior distribution so as to compute the normalizing constants, it is common to use Markov Chain Monte Carlo (MCMC) methods (Neal, 1996; Rasmussen, 1995). Another line of research for posterior inference uses

stochastic gradient Markov Chain Monte Carlo (Chen et al., 2015a). However, one major drawback of sampling, is that it is often very slow, especially for high-dimensional models (Robert & Casella, 2005). Faster and easy to parallelize methods may be drawn from Bayesian variational inference algorithms. These have been developed to be as flexible as MCMC (Zhai et al., 2012). Therefore, this is the route we follow to overcome the computational burden of posterior and Bayes factors calculation.

4.4 Bayesian variational inference

In this section we explain how we find a Bayesian variational approximation of the posterior distributions. We choose a family of distributions of the model parameters $\psi_\sigma = (w, b, \Sigma)$ indexed by a hyper-parameter λ, $Q(\psi_\sigma; \lambda)$. The idea is to approximate the posterior of ψ_σ with its closest distribution from this family. This corresponds to an optimization over the hyper-parameter λ. In this section, we suppose that we work with a given cluster S of the partition ρ_n, and its associated set of response variables T. The core idea is to minimize over λ the Kullback-Leibler divergence (KL) between the posterior and $Q(\psi_\sigma; \lambda)$. That is, we need to minimize

$$\mathrm{KL}(Q||p) = \int Q(\psi_\sigma; \lambda) \log\{Q(\psi_\sigma; \lambda)/p(\psi_\sigma|T, S, \rho_n)\}.$$

The optimal value λ_{opt} gives rise to the approximating *variational posterior distribution* $Q(\psi_\sigma; \lambda_{opt})$.

Approximated Bayes factors and Besag's algorithm. In practice, one obtains k_n such variational posteriors $Q(\psi_{\sigma j}; \lambda_j)$, $j = 1, \ldots, k_n$ for a given partition ρ_n. These are the distributions that we use to replaced the intractable posteriors $p(\boldsymbol{\psi}_\sigma|\boldsymbol{y}^n, \boldsymbol{x}^n, \rho_n)$ in:
(i) STEP B of Besag's algorithm (see previous section) to sample the parameters $\boldsymbol{\psi}_\sigma$; and
(ii) STEP A, to estimate the Bayes factor α. More specifically, let, respectively, $\widehat{\boldsymbol{\psi}}_\sigma = (\hat{\psi}_{\sigma 1}, \ldots, \hat{\psi}_{\sigma k_n})$ and $\widehat{\boldsymbol{\psi}}'_\sigma = (\hat{\psi}'_{\sigma 1}, \ldots, \hat{\psi}_{\sigma k'_n})$ be the maximum a posteriori (MAP) estimators of $\boldsymbol{\psi}_\sigma$ and $\boldsymbol{\psi}'_\sigma$. We use the estimate

$$\hat{\alpha} = \frac{p(\boldsymbol{y}^n|\widehat{\boldsymbol{\psi}}'_\sigma, \rho'_n, \boldsymbol{x}^n) p(\widehat{\boldsymbol{\psi}}'_\sigma|\rho'_n)}{p(\boldsymbol{y}^n|\widehat{\boldsymbol{\psi}}_\sigma, \rho_n, \boldsymbol{x}^n) p(\widehat{\boldsymbol{\psi}}_\sigma|\rho_n)} \frac{\prod_{j=1}^{k_n} Q(\hat{\psi}_{\sigma j}|\lambda_j)}{\prod_{j=1}^{k'_n} Q(\hat{\psi}'_{\sigma,j}|\lambda'_j)}.$$

The ELBO. Note that the optimal value is the solution of the optimization problem

$$\begin{aligned} \lambda_{opt} &= \arg\min_\lambda \mathrm{KL}(Q(\psi_\sigma; \lambda) \| p(\psi_\sigma|T, S, \rho_n)\big) \\ &= \arg\min_\lambda \int Q(\psi_\sigma; \lambda) \log \frac{Q(\psi_\sigma; \lambda)}{p(\psi_\sigma|\rho_n) p(T|\psi_\sigma, S, \rho_n)} d\psi_\sigma \\ &= \arg\max_\lambda \mathbb{E}_{Q(\psi_\sigma;\lambda)}\big(\log p(T|\psi_\sigma, S, \rho_n)\big) - \mathrm{KL}\big(Q(\psi_\sigma; \lambda) \| p(\psi_\sigma|\rho_n)\big). \end{aligned} \tag{4.10}$$

where the term to be maximized is known as the *evidence lower bound* or ELBO:

$$\mathrm{ELBO}(\lambda) = \mathbb{E}_{Q(\psi_\sigma;\lambda)}\big(\log p(T|\psi_\sigma, S, \rho_n)\big), - \mathrm{KL}\big(Q(\psi_\sigma; \lambda) \| p(\psi_\sigma|\rho_n)\big) \tag{4.11}$$

The negative of this quantity is also known as the description length cost, or the variational free energy (see, for example, Friston et al. (2007)). The direct optimisation of this quantity instead of the Kullback-Leibler divergence was introduced by Hinton & van Camp (1993) (see also, Graves (2011)). In general,

the ELBO cannot be computed exactly. The expectations are usually approximated through Monte Carlo methods (Blundell et al., 2015).

Choice of variational family. If the likelihood $p(T|\psi_\sigma, S, \rho_n)$ were linear in the weight and bias parameters, the posterior of the parameters would be conjugate to the prior (Murua & Quintana, 2017c). Inspired by this observation, we consider $\lambda = (\lambda_w, \lambda_b, \lambda_\sigma)$ and the family of conjugate distributions to the prior

$$Q(\psi_\sigma; \lambda) = Q(w; \lambda_w)Q(b; \lambda_b)Q(\Sigma; \lambda_\sigma).$$

More specifically, we suppose that $Q(w; \lambda_w)$ is of the same form as the prior $p(w|\Omega)$. That is, $Q(w; \lambda_w)$ is a zero-mean Gaussian random field with covariance structure dictated by the Gibbs network induced by Ω. Thus, if the Gibbs field given by the prior is a compound symmetry Gibbs network, then so it is $Q(w; \lambda_w)$, etc. Similarly, $Q(b; \lambda_b)$ is a zero-mean Gaussian field with block diagonal covariance $\lambda_b = \mathrm{diag}(\lambda_{b1}, \lambda_{b2})$ just as Γ. Also, just as the prior, the variational posterior approximation $Q(\Sigma; \lambda_\sigma)$ is assumed to be an Inverse-Wishart$(\nu_\sigma, \Lambda_\sigma)$. With this family of distributions, the ELBO becomes

$$\begin{aligned} \mathrm{ELBO}(\lambda) \;=\; & \mathbb{E}_{Q(\psi_\sigma;\lambda)}\big(\log p(T|\psi_\sigma, S, \rho_n)\big) - \mathrm{KL}\big(Q(w; \lambda_w)\|p(w|\rho_n)\big) \\ & - \mathrm{KL}\big(Q(b; \lambda_b)\|p(b|\rho_n)\big) - \mathrm{KL}\big(Q(\Sigma; \lambda_\sigma)\|p(\Sigma|\rho_n)\big) \end{aligned} \tag{4.12}$$

It is not difficult to show that the Kullback-Leibler divergences between the two sets of Gaussian distributions are

$$\begin{aligned} \mathrm{KL}\big(Q(w; \lambda_w)\|p(w|\rho_n)\big) &= \frac{1}{2}\left\{\mathrm{trace}\big(\Omega^{-1}\lambda_w\big) - (q+p)l_1 + \log\big(\det(\Omega)/\det(\lambda_w)\big)\right\}, \\ \mathrm{KL}\big(Q(b; \lambda_b)\|p(b|\rho_n)\big) &= \frac{1}{2}\left\{\mathrm{trace}\big(\Gamma^{-1}\lambda_b\big) - (l_1+p) + \log\big(\det(\Gamma)/\det(\lambda_b)\big)\right\}. \end{aligned}$$

For the last divergence we have the following result.

Proposition 4.4.1
The Kullback-Leibler divergence $\mathrm{KL}\big(Q(\Sigma; \lambda_\sigma)\|p(\Sigma|\rho_n)\big)$ *between the two inverse-Wisharts* $Q(\Sigma; \lambda_\sigma)$ *and* $p(\Sigma|\rho_n)$*, is given by*

$$\frac{\nu}{2}\log\left(\frac{\det(\Lambda_\sigma)}{\det(\Lambda)}\right) + \frac{\nu_\sigma}{2}\mathrm{trace}\big(\Lambda\Lambda_\sigma^{-1}\big) - (\nu - \nu_\sigma)\frac{\psi_p\big(\nu_\sigma/2\big)}{2} - \nu_\sigma\frac{p}{2} + \log\left(\frac{\Gamma_p\big(\nu/2\big)}{\Gamma_p\big(\nu_\sigma/2\big)}\right), \tag{4.13}$$

where $\Gamma_p(\cdot)$ *stands for the* p*-multivariate gamma function, and* $\psi_p(\cdot)$ *denotes the derivative of the* p*-multivariate gamma function (i.e., the multivariate digamma function).*

Proof: By definition $\mathrm{KL}\big(Q(\Sigma;\lambda_\sigma)\|p(\Sigma|\rho_n)\big)$ is equal to

$$
\begin{aligned}
&-\frac{(\nu_\sigma+p+1)}{2}\mathbb{E}_{Q(\Sigma;\lambda_\sigma)}\big(\log\det(X)\big)-\frac{1}{2}\operatorname{trace}\big(\Lambda_\sigma\mathbb{E}_{Q(\Sigma;\lambda_\sigma)}(X^{-1})\big)+\frac{\nu_\sigma}{2}\log\det(\Lambda_\sigma)\\
&\qquad-\frac{\nu_\sigma p}{2}\log(2)-\log\big(\Gamma_p(\nu_\sigma/2)\big)+\frac{(\nu+p+1)}{2}\mathbb{E}_{Q(\Sigma;\lambda_\sigma)}\big(\log\det(X)\big)\\
&\quad+\frac{1}{2}\operatorname{trace}\big(\Lambda\mathbb{E}_{Q(\Sigma;\lambda_\sigma)}(X^{-1})\big)-\frac{\nu}{2}\log\det(\Lambda)+\frac{\nu p}{2}\log(2)+\log\big(\Gamma_p(\nu/2)\big)\\
&\qquad=\frac{(\nu-\nu_\sigma)}{2}\mathbb{E}_{Q(\Sigma;\lambda_\sigma)}\big(\log\det(X)\big)+\frac{1}{2}\operatorname{trace}\big([\Lambda-\Lambda_\sigma]\mathbb{E}_{Q(\Sigma;\lambda_\sigma)}(X^{-1})\big)\\
&\qquad\qquad+\frac{\nu_\sigma}{2}\log\det(\Lambda_\sigma)-\frac{\nu}{2}\log\det(\Lambda)+(\nu-\nu_\sigma)\frac{p\log(2)}{2}+\log\left(\frac{\Gamma_p(\nu/2)}{\Gamma_p(\nu_\sigma/2)}\right),
\end{aligned}
$$

where X denotes a positive definite matrix. Note that when $X\sim$ Inverse-Wishart$(\nu_\sigma,\Lambda_\sigma)$, its inverse $X^{-1}\sim$ Wishart$(\nu_\sigma,\Lambda_\sigma^{-1})$. Therefore,

$$
\begin{aligned}
\mathbb{E}_{Q(\Sigma;\lambda_\sigma)}(X^{-1})&=\nu_\sigma\Lambda_\sigma^{-1},\ \text{and (Bishop, 2006b, pp. 693)}\\
\mathbb{E}_{Q(\Sigma;\lambda_\sigma)}\big(\log\det(X)\big)&=-\psi_p\big(p/2\big)-p\log(2)-\log\det(\Lambda_\sigma^{-1}).
\end{aligned}
$$

Putting all together, we obtain the desired result (4.13).■

The optimal solution is found using the stochastic gradient ascent variational Bayes algorithm (Paisley et al., 2012; Kucukelbir et al., 2014; Ye et al., 2020; Duchi et al., 2011). To obtain the gradient of ELBO(λ), the gradients of the Kullback-Leibler divergence terms are needed. In practice, these derivatives are usually computed through automatic differentiation. That is, calculating numerically the value of the derivatives (e.g., using Chebyshev polynomial approximation (Press et al., 1996, Ch. 5.7)). These methods achieve very good accuracies (Bartholomew-Biggs et al., 2000).

The integrated log-likelihood, the first term in (4.12), is intractable. We use Monte Carlo sampling to estimate it. However, we only need to estimate the score (Kingma et al., 2015; Mohamed et al., 2019),

$$
\frac{\partial}{\partial\lambda}\left(\mathbb{E}_{Q(\psi_\sigma;\lambda)}\big(\log p(T|\psi_\sigma,S,\rho_n)\big)\right)=\mathbb{E}_{Q(\psi_\sigma;\lambda)}\left(\log p(T|\psi_\sigma,S,\rho_n)\frac{\partial Q(\psi_\sigma;\lambda)}{\partial\lambda}\right).
$$

The integral in the right-hand-side of the above equation is estimated by Monte Carlo sampling of ψ_σ from $Q(\psi_\sigma;\lambda)$, which is a multivariate Gaussian-inverse-Wishart distribution.

4.4.1 Regularization on the CS-Gibbs model

For the compound symmetry Gibbs network model described at the end of in Section 4.1.2, the off-diagonal block $\Omega_{12}=U\otimes V$, for $U\in\mathbb{M}_{l_1\times p}$, and $V\in\mathbb{M}_{q\times l_1}$. These matrices are unconstrained. For the variational approximation, we suppose that the corresponding block of the precision matrix has the same structure. Let $U_{12}\in\mathbb{M}_{l_1\times p}$, and $V_{12}\in\mathbb{M}_{q\times l_1}$ be the corresponding matrices in the variational approximation.

Note that the our prior for the weights correspond to a Gaussian graphical model (Giudici & Green, 1999; Rajaratnam et al., 2008). The usual procedure to find sparse covariance models for Gaussian random fields is the so-called graphical lasso (Friedman et al., 2008). This method is a penalized

log-likelihood in which the size of the entries of Ω are penalized, so as to render this matrix sparse. Translating this method to our already sparse model leads to the penalization of the L_1-norm of the matrices U_{12} and V_{12}. That is, we consider the maximization of the penalized log-likelihood

$$\log p(T|\psi_\sigma, S, \rho_n) - \beta_u \|U_{12}\|_1 - \beta_v \|V_{12}\|_1,$$

where β_u and β_v are the regularization constants. Introducing the variational approximation, we have

$$\begin{aligned}\mathbb{E}_{Q(\psi_\sigma;\lambda)}\big(\log p(T|\psi_\sigma, S, \rho_n) - \beta_u \|U_{12}\|_1 - \beta_v \|V_{12}\|_1\big) \\ = \mathrm{ELBO}(\lambda) + \mathrm{KL}\left(Q\left(\psi_\sigma;\lambda\right) \| p\left(\psi_\sigma \mid T, S, \rho_n\right)\right) - \beta_u \|U_{12}\|_1 - \beta_v \|V_{12}\|_1.\end{aligned}$$

Because $\mathrm{KL}\left(Q\left(\psi_\sigma;\lambda\right) \| p\left(\psi_\sigma \mid T, S, \rho_n\right)\right) \geq 0$, we have

$$\mathrm{ELBO}(\lambda) - \beta_u \|U_{12}\|_1 - \beta_v \|V_{12}\|_1 \leq \mathbb{E}_{Q(\psi_\sigma;\lambda)}\big(\log p(T\|\psi_\sigma, S, \rho_n)\big) - \beta_u \|U_{12}\|_1 - \beta_v \|V_{12}\|_1,$$

and we can think of the right-hand-side of this expression as a lower bound for the expected graphical-lasso log-likelihood function. The goal is then to solve the maximizition problem

$$\lambda^* = \arg\max_\lambda \big\{\mathrm{ELBO}(\lambda) - \beta_u \|U_{12}\|_1 - \beta_v \|V_{12}\|_1.\big\}$$

This result may also be embedded in a special case of sparse variational inference (Campbell & Beronov, 2019) or *Generalized ELBO with Constrained Optimization* (Rezende & Viola, 2018), where we build a regularized framework for our compound symmetry Gibbs network in other to set sparsity constraints.

The optimization problem is not straightforward to solve. First, the L_1-norm is non differentiable. And more importantly, the solution for the covariance matrix must still be a positive definite matrix. There are several algorithms designed specially to solve the usual graphical lasso model (Mazumder & Hastie, 2015). But, they are not directly applicable to our model. We have addressed these two issues. In this section, we only explain how we have modified the maximization problem for this special case of our model. The problem dealing with the positive-definiteness of the matrix applies to all our models. It is addressed in the next section.

To deal with the non-differentiability of the L_1 norm, we have implemented two modifications to the original problem. The first one consists of replacing the absolute values of the elements of the two matrices $|u|$ by a smooth version of them. These are given by a smooth approximation (Schmidt et al., 2007) of the absolute value $|u|_\alpha = (1/\alpha)\{\log(1+\exp(-\alpha u)) + \log(1+\exp(\alpha u))\}$, for a given constant α. It can be easily seen that as the value of α increases $|u|_\alpha$ approaches $|u|$. In fact, Schmidt et al. (2007) show that $|\,|u| - |u|_\alpha\,| \leq \frac{2\log(2)}{\alpha}$. Therefore, the constant α must be fixed to a large enough value (a value of about 10^6 is large enough as suggested in (Schmidt et al., 2007), but smaller values also give good approximations). Our second modification is to replace the L_1-norm by the Frobenius (that is, Euclidean) norm $\|\cdot\|$ of the matrices U_{12} and V_{12}. This modification can be thought of as a group-lasso version of the graphical lasso. Unfortunately, the Frobenius norm is also non-differentiable. Instead, we consider the squared of the Frobenius norm, which is a smooth function. In particular, it is easily seen that its derivative is given by $\nabla \|U\|_F^2 = \frac{\partial}{\partial U} \operatorname{trace}\left(U^T U\right) = 2U$.

The parameters β_u and β_v must be chosen adequately. We perform K-fold cross-validation to establish appropriate values for them. We stress that this is done during the fitting (training) stage of the model for each cluster of the partitions ρ_n. Because it is not possible to do cross-validation over the entire

space of possible values, we set a sequence of possible values for $\beta = (\beta_u, \beta_v)$. Let $\{\beta^{(m)} : m = 1, \ldots, M\}$ be this sequence. The procedure is as follows: Divide the training data (T, S) into K disjoint approximately equal size subsets $\mathcal{D}_\ell$, $\ell = 1, \ldots, K$. For each $m \in \{1, \ldots, M\}$, and $\ell \in \{1, \ldots, K\}$, set $\mathcal{D}_{-\ell} = \cup_{j \neq \ell} \mathcal{D}_j$, and fit the model with the data in $\mathcal{D}_{-\ell}$ and parameters $\beta^{(m)}$. Let $\{\hat{y}_i^{(\ell,m)} : y_i \in T\}$ be the adjusted values of the model. Compute $\mathrm{CV}(\beta^{(m)}) = \sum_{\ell=1}^{K} \sum_{y_i \in \mathcal{D}_\ell} \|y_i - \hat{y}_i^{(\ell,m)}\|^2$. Let $\hat{\beta} \in \{\beta^{(m)} : m = 1, \ldots, M\}$ be the parameter values that minimizes $\mathrm{CV}(\beta^{(m)})$. Fit the model with all data (T, S) and parameters $\hat{\beta}$. In our implementation we have use $M \geq 100$. The β values of the sequence were chosen automatically by the two-dimensional Golden-Section Search (GSS) algorithm (Chang, 2009). To initialize the search, several random values for each component of the vector β were drawn. Each draw is a value chosen uniformly at random from one of the intervals in the collection $\{[\epsilon + 100\ell, \epsilon + 100(\ell + 1)] : \ell = 0, 1, \ldots, \ell_{\max}\}$, where, in practice, $\ell_{\max} \leq 100$, and $\epsilon \leq 1/10000$. For each draw, the interval was also chosen uniformly at random. Technically, the GSS is based on the fact that the minimum lies within the interval defined by the two points adjacent to the last observed value. The method operates by successively narrowing the range of values on the specified search space, which makes it relatively slow, but very robust. It finds the best extrema $(\beta_u^{opt}, \beta_v^{opt})$ after going through all the regions. GSS presents an oracle complexity which converges to an ϵ-accurate solution at a linear rate, also known as geometric or exponential convergence in the numerical literature (Bertolazzi, 2008; Sebastien, 2013). In fact, GSS makes $O(\log(1/\epsilon))$ calls to an *Oracle Query Optimizer* (Wijesiriwardana & Firdhous, 2019) to compute the optimum.

4.4.2 Keeping positive definiteness on the precision Matrix

Let λ_w be the precision matrix associated with the variational density approximation to the posterior of the weight parameters w. The gradient of the ELBO with respect to the variational precision matrix λ_w, denoted $\nabla \mathrm{ELBO}(\lambda_w)$, is computed component-wise or block-wise in all sparse structures. For example, in the Sparse-Gibbs network where the precison matrix λ_w has the form given by (4.4), we compute each gradient $\nabla \mathrm{ELBO}(V_{i\bullet})$, $\nabla \mathrm{ELBO}(V_{\bullet j})$, and $\nabla \mathrm{ELBO}(V_{ii})$ separately in order to build the full update of λ_w. The same differentiation principle is applied for the sparse compound symmetry Gibbs model (sparse-CS-Gibbs network) where we constrain $U_1, U_2, V_{1\bullet}$, and $V_{\bullet 1}$ to have compound symmetry.

One of the issues that arised on the estimation of λ_w is how to ensure in practice that the matrix is kept positive definite during the gradient updates. Contrary to graphical lasso (Mazumder & Hastie, 2015), the positive definiteness property can be lost. There is no known algorithm that can ensure this property for our models while optimizing the matrix. To find a very good or optimal solution might require heavy exploration of the matrix space at each gradient update. Hence, it might be impractical to optimize the ELBO over λ_w with simultaneous constraints on both the sparse structure, and the positive definiteness property. Given the complexity of the model, and from a statistical viewpoint, it might be sufficient to obtain an approximate solution for λ_w that is both sparse as imposed, and positive definite. Therefore, betting on this observation, we based our matrix updates on the search for the nearest positive definite matrix (Higham, 1988). An simple procedure to find the nearest positive definite matrix $\hat{\lambda}_w$, where proximity is measure by the Frobenius norm, is based on the spectral decomposition of the matrix (Jewbali & Ore, 2009). We first determine the spectral decomposition of $\lambda_w = QDQ^T$, where Q is an unitary matrix, and D is a diagonal matrix with the eigenvalues of λ_w. An estimate of the nearest-positive definite matrix to λ_w is given by $\check{\lambda}_w = Q\check{D}Q^T$, where $\check{D}$ is the diagonal matrix D modified with all negative eigenvalues set to a small positive constant. However, $\check{\lambda}_w$ might not be the nearest positive

definite matrix. The optimal small constant needs to be searched. One algorithm that does this search is the Iterative Spectral Method (Marée, 2012). Unfortunately, this and variants of this algorithm are costly, taking order $O(d^3)$ operations for a $d \times d$ matrix.

Two practical ways to verify if the above approximation is necessary, are, first to try a Cholesky decomposition of λ_w after each update, or to apply the well-known Sylvester's criterion (Gilbert, 1991) to each update. The latter verifies if all leading principal minors of λ_w are positive. This criterion is doubly usefull, since it also helps to build the nearest symmetric positive definite matrix, with the imposed model structure. The minors can be used for non-positive definiteness correction. In fact, the kth principal minor is computed based on the $(k-1)$th principal minor augmented with the corresponding reduced kth column and row. If the $(k-1)$th principal minor is positive, and the kth principal minor is not, then one can make this latter positive with some slight modifications to the kth column and/or row. This can be done sequentially until all principal minors has been computed. However, one drawback of this method, besides its computational cost, is that it does not always guarantee to get the nearest positive definite matrix in terms of the Frobenius norm.

In our implementation, we use the *nearest Positive Definite Matrix computation,* which is mathematically equivalent to the gradient projection method known as *Projected gradient updates* (Cruz et al., 2011; Hassani et al., 2017). The combination of the gradient updates and the projection into the nearest positive definite matrix space is called *Iterative projected gradient* (IPG). This iterates between calculating the gradient and calculating the projection into the positive definite matrix space $\mathcal{S}+$. More explicitly, at iteration k, the IPG computes

$$\lambda_w^{(k)} = \mathcal{P}_{\mathcal{S}+}\left(\lambda_w^{(k-1)} + \alpha_k \nabla \,\mathrm{ELBO}\left(\lambda_w^{(k-1)}\right)\right) \tag{4.14}$$

where, α_k is the step size, and $\mathcal{P}_{\mathcal{S}+}$ denotes the Euclidean projection into the positive definite matrix space, that is,

$$\mathcal{P}_{\mathcal{S}+}(\lambda_w) = \mathrm{argmin}_{A\in\mathcal{S}+} \|A - \lambda_w\|_F$$

where, $\|\cdot\|_F$ stands for the Frobenius norm. To compute $\mathcal{P}_{\mathcal{S}+}(\lambda_w)$, we have applied an approximation by matrices positive semi-definite on the subspace $\mathcal{S}+$ as detailed in (Hayden & Wells, 1988). From properties of the Frobenius norm, we have that:

$$\|A - \lambda_w\|_F^2 = \|B - A + C\|_F^2 = \|B - \lambda_w\|_F^2 + \|C\|_F^2$$

with :

$$B = \frac{\lambda_w + {\lambda_w}^T}{2} \quad \text{and} \quad C = \frac{\lambda_w - {\lambda_w}^T}{2}$$

Thus we minimize $\|A - \lambda_w\|_F$ by minimizing $\|B - A\|_F$. Now, our approximation in the subspace $\mathcal{S}+$, goes as it follows:

- Take $A \longleftarrow B = \frac{\lambda_w + {\lambda_w}^T}{2}$. This is a transformation technique to force A to have the same properties as the symmetric matrix B.
- Compute the spectrale decomposition of A, let say $A = U\Lambda U^T$ where U is orthogonal and $\Lambda = \mathrm{diag}\left[\lambda_1, \lambda_2, \ldots, \lambda_n\right]$. Then the unique best approximation A^+ to A is given by:

$$A^+ = U\Lambda_+ U^T$$

where Λ_+ is obtained from Λ by replacing each negative eigenvalue by a number $\boldsymbol{a} > 0$. We denote the new matrix as $A(a)$. So if there are no negative eigenvalues, A is taken as our approximation.

- If A as negative eigenvalues, we generate multiple values[4] of $\boldsymbol{a}$, to find $\boldsymbol{a}_{opt}$, the optimal value of a.

$$\mathcal{P}_{\mathbb{S}+}(\lambda_w) = \text{argmin}_{A\in\mathbb{S}+} \|A - \lambda_w\|_F = A(\boldsymbol{a}_{opt})$$

However, we have also found that this projection was not robust. Our algorithm has then been reinforced with the **mbend** package from Nilforooshan (2020), which increase robustness by adding a weight matrix to A^+ (Jorjani et al., 2003), or by replacing each of the m negative eigenvalues (λ_i) with $\rho\left(s-\lambda_i\right)^2 / \left(100s^2+1\right)$, where ρ is the smallest positive eigenvalue and $s = 2\sum_{i=1}^{m}\lambda_i$. This is called methode $LRS14$ in their R package Nilforooshan & Nilforooshan (2020). To obtain the (ultimate) best solution $\mathcal{P}_{\mathbb{S}+}(\lambda_w)$, those procedures are repeated until convergence is reached.

4.5 Predictive Posterior

In this section we describe how we estimate the predictive posterior distribution. That is, the estimation of $p(y^{(n+1)}|x^{(n+1)}, \boldsymbol{y}^n, \boldsymbol{x}^n)$ for a new data item $x^{(n+1)}$, which is the goal of our Potts Gibbs-network. Since the mixture is over all possible partitions ρ_{n+1} of the data $\boldsymbol{x}^n \cup \{x^{(n+1)}\}$, we have

$$p(y^{(n+1)}|x^{(n+1)}, \boldsymbol{y}^n, \boldsymbol{x}^n) = \sum_{\rho_{n+1}} p(y^{(n+1)}|x^{(n+1)}, \boldsymbol{y}^n, \boldsymbol{x}^n, \rho_{n+1}) p(\rho_{n+1}|x^{(n+1)}, \boldsymbol{y}^n, \boldsymbol{x}^n).$$

As in (Murua & Quintana, 2017c), we consider the collection $\mathcal{A}(\rho_{n+1})$ of all partitions ρ_n of $\boldsymbol{x}^n$ giving rise to the partition ρ_{n+1} of $\boldsymbol{x}^n \cup \{x^{(n+1)}\}$ by generating appropriate bonds b_{n+1} in the extended graph with vertices $\boldsymbol{x}^n$ and $x^{(n+1)}$. The extended graph includes all edges of the the original graph plus the edges added to link the new data point. (e.g., if the graph is a k-nearest-neighbors graph, then all edges between $x^{(n+1)}$ and its k nearest neighbors are added). Elements in the collection $\mathcal{A}(\rho_{n+1})$ will be denoted as $[\rho_n, b_{n+1}]$ to make clear that ρ_{n+1} can be generated from ρ_n and the bonds b_{n+1}. Then, we can write

$$p(\rho_{n+1}|x^{(n+1)}, \boldsymbol{y}^n, \boldsymbol{x}^n) = \sum_{[\rho_n, b_{n+1}]} p(\rho_n|\boldsymbol{y}^n, \boldsymbol{x}^n) p(b_{n+1}|\rho_n, x^{(n+1)}, \boldsymbol{x}^n).$$

Therefore, the predictive posterior $p(y^{(n+1)}|x^{(n+1)}, \boldsymbol{y}^n, \boldsymbol{x}^n)$ is the mixture:

$$\sum_{[\rho_n, b_{n+1}]} p(y^{(n+1)}|x^{(n+1)}, \boldsymbol{y}^n, \boldsymbol{x}^n, [\rho_n, b_{n+1}]) p(\rho_n|\boldsymbol{y}^n, \boldsymbol{x}^n) p(b_{n+1}|\rho_n, x^{(n+1)}, \boldsymbol{x}^n). \tag{4.15}$$

This expression is intractable since the number of possible partitions of $\boldsymbol{x}^n$ is the well-known Bell number B_n. Fortunately, having sampled partitions and parameters from the posterior of ρ_n and $\boldsymbol{\psi}_\sigma$, we can use these samples to estimate the predictive posterior as follows. Let $\{(\rho_{n,\ell}, \boldsymbol{\psi}_{\sigma,\ell}) : \ell = 1, ..., M\}$ be

[4] In Python, it is implemented in Scipy library as the Golden Section Search technique.

a sample from the posterior distribution. Also, let $\{b_{n+1,\ell} : \ell = 1, ..., M\}$ be a sample of bonds linking the sampled partitions ρ_n to $x^{(n+1)}$. The estimate is

$$\hat{p}(y^{(n+1)}|x^{(n+1)}, \boldsymbol{y}^n, \boldsymbol{x}^n) = \frac{1}{M}\sum_{\ell=1}^{M} p(y^{(n+1)}|x^{(n+1)}, \boldsymbol{y}^n, \boldsymbol{x}^n, [\rho_{n,\ell}, b_{n+1,\ell}]). \tag{4.16}$$

The terms in the sum are

$$p(y^{(n+1)}|x^{(n+1)}, \boldsymbol{y}^n, \boldsymbol{x}^n, [\rho_n, b_{n+1}]) = \frac{p(y^{(n+1)}, \boldsymbol{y}^n|x^{(n+1)}, \boldsymbol{x}^n, [\rho_n, b_{n+1}])}{p(\boldsymbol{y}^n|x^{(n+1)}, \boldsymbol{x}^n, [\rho_n, b_{n+1}])} \tag{4.17}$$

$$= \frac{p(y^{(n+1)}, \boldsymbol{y}^n|x^{(n+1)}, \boldsymbol{x}^n, [\rho_n, b_{n+1}])}{\int_t p(t, \boldsymbol{y}^n|x^{(n+1)}, \boldsymbol{x}^n, [\rho_n, b_{n+1}])\, dt}. \tag{4.18}$$

Let $\bar{\jmath}$ be the cluster in which $x^{(n+1)}$ falls in the partition ρ_{n+1}, and suppose that this cluster is $\bar{S} = S_{\bar{\jmath}} \cup \{x^{(n+1)}\}$, with $S_{\bar{\jmath}} \subset \rho_n$. Let $\bar{T} = T_{\bar{\jmath}} \cup \{y^{(n+1)}\}$. Note that

$$p(y^{(n+1)}, \boldsymbol{y}^n|x^{(n+1)}, \boldsymbol{x}^n, [\rho_n, b_{n+1}]) = p(\bar{T}|\bar{S}, \rho_{n+1}) \prod_{j \neq \bar{\jmath}} p(T_j|S_j, \rho_n),$$

and similarly

$$\int_t p(t, \boldsymbol{y}^n|x^{(n+1)}, \boldsymbol{x}^n, [\rho_n, b_{n+1}]) = \left(\int_t p(\{T_{\bar{\jmath}} \cup \{t\}|\bar{S}, \rho_{n+1})\, dt\right)\left(\prod_{j \neq \bar{\jmath}} p(T_j|S_j, \rho_n)\right).$$

Hence all the terms cancel in (4.17) except for the term associated with $x^{(n+1)}$. Therefore

$$p(y^{(n+1)}|x^{(n+1)}, \boldsymbol{y}^n, \boldsymbol{x}^n, [\rho_{n,j}, b_{n+1,j}]) = \frac{p(\bar{T}|\bar{S}, \rho_{n+1})}{\int_t p(\{T_{\bar{\jmath}} \cup \{t\}|\bar{S}, \rho_{n+1})\, dt}.$$

Using any value of $\psi_{\sigma\bar{\jmath}}$ in expression (4.9), we get

$$\hat{p}(y^{(n+1)}|x^{(n+1)}, \boldsymbol{y}^n, \boldsymbol{x}^n, [\rho_n, b_{n+1}]) \approx \frac{p(\bar{T}|\psi_{\sigma\bar{\jmath}}, \bar{S}, \rho_{n+1})p(\psi_{\sigma\bar{\jmath}}|\rho_{n+1})}{p(\psi_{\sigma\bar{\jmath}}|\bar{T}, \bar{S}, \rho_{n+1}) \int_t \frac{p(T_{\bar{\jmath}} \cup \{t\}|\psi_{\sigma\bar{\jmath}}, \bar{S}, \rho_{n+1})p(\psi_{\sigma\bar{\jmath}}|\rho_{n+1})}{p(\psi_{\sigma\bar{\jmath}}|T_{\bar{\jmath}} \cup \{t\}, \bar{S}, \rho_{n+1})}\, dt}$$

$$= \frac{p(\bar{T}|\psi_{\sigma\bar{\jmath}}, \bar{S}, \rho_{n+1})}{p(\psi_{\sigma\bar{\jmath}}|\bar{T}, \bar{S}, \rho_{n+1}) \int_t \frac{p(T_{\bar{\jmath}} \cup \{t\}|\psi_{\sigma\bar{\jmath}}, \bar{S}, \rho_{n+1})}{p(\psi_{\sigma\bar{\jmath}}|T_{\bar{\jmath}} \cup \{t\}, \bar{S}, \rho_{n+1})}\, dt}.$$

Remark

In particular,

$$\hat{p}(y^{(n+1)}|x^{(n+1)}, \boldsymbol{y}^n, \boldsymbol{x}^n, [\rho_n, b_{n+1}]) = \mathbb{E}_{p(\psi_{\sigma\bar{\jmath}}|\bar{T}, \bar{S}, \rho_{n+1})}\left\{\frac{p(\bar{T}|\psi_{\sigma\bar{\jmath}}, \bar{S}, \rho_{n+1})p(\psi_{\sigma\bar{\jmath}}|\rho_{n+1})}{p(\psi_{\sigma\bar{\jmath}}|\bar{T}, \bar{S}, \rho_{n+1})}\right\} \times$$

$$\left(\int_t \mathbb{E}_{p(\psi_{\sigma\bar{\jmath}}|\bar{T}, \bar{S}, \rho_{n+1})}\left[\frac{p(T_{\bar{\jmath}} \cup \{t\}|\psi_{\sigma\bar{\jmath}}, \bar{S}, \rho_{n+1})p(\psi_{\sigma\bar{\jmath}}|\rho_{n+1})}{p(\psi_{\sigma\bar{\jmath}}|T_{\bar{\jmath}} \cup \{t\}, \bar{S}, \rho_{n+1})}\, dt\right]\right)^{-1}.$$

In practice, we suppose that we can approximate $p(\psi_{\sigma\bar{\jmath}}|T_{\bar{\jmath}} \cup \{t\}, \bar{S}, \rho_{n+1})$ by $p(\psi_{\sigma,\bar{\jmath}}|T_{\bar{\jmath}}, S_{\bar{\jmath}}, \rho_n)$, and that, in turn, this last posterior is well approximated by the variational posterior approximation $Q(\psi_{\sigma\bar{\jmath}}; \lambda_{\bar{\jmath}})$.

Then each one of the expectations can be approximated by Monte Carlo, using our samples $\{\psi_{\sigma\bar{j}_\ell,\ell}\}_{\ell=1}^{M}$ *from the variational posterior, where* $\bar{j}_\ell$ *denotes the cluster in* $\rho_{n+1,\ell}$ *in which* $x^{(n+1)}$ *falls,* $\ell = 1, \ldots, M$. *This yields*

$$\hat{p}(y^{(n+1)}|x^{(n+1)}, \boldsymbol{y}^n, \boldsymbol{x}^n, [\rho_n, b_{n+1}]) = \left(\sum_{\ell=1}^{M} \frac{p(\bar{T}_\ell|\psi_{\sigma\bar{j}_\ell,\ell}, \bar{S}_\ell, \rho_{n+1,\ell})p(\psi_{\sigma\bar{j}_\ell,\ell}|\rho_{n+1,\ell})}{Q(\psi_{\sigma\bar{j}_\ell,\ell}; \lambda_{\bar{j}_\ell,\ell})}\right) \times \left(\int_t \left[\sum_{\ell=1}^{M} \frac{p(T_{\bar{j}_\ell} \cup \{t\}|\psi_{\sigma\bar{j}_\ell,\ell}, \bar{S}_\ell, \rho_{n+1,\ell})p(\psi_{\sigma\bar{j}_\ell,\ell}|\rho_{n+1,\ell})}{Q(\psi_{\sigma\bar{j}_\ell,\ell}; \lambda_{\bar{j}_\ell,\ell})}\, dt\right]\right)^{-1},$$

with $\bar{S}_\ell = S_{\bar{j}_\ell,\ell} \cup \{x^{(n+1)}\}$, *and* $\bar{T}_\ell = T_{\bar{j}_\ell,\ell} \cup \{y^{(n+1)}\}$. *The integral over t, which is a univariate integral, may be computed numerically, for example, using Romberg's algorithm (Li & Hu, 2017).*

In particular, the value of the prediction $\mathbb{E}(y|x^{(n+1)}, \boldsymbol{y}^n, \boldsymbol{x}^n)$ may be estimated by the quantity:

$$\frac{1}{M}\sum_{\ell=1}^{M} \mathbb{E}(y|\psi_{\sigma\bar{j}_\ell}, x^{(n+1)}) = \frac{1}{M}\sum_{\ell=1}^{M} \left[b_\ell^{(2)} + W_\ell^{(2)} g_1\left(b_\ell^{(1)} + W_\ell^{(1)} x^{(n+1)}\right)\right].$$

The Shallow Potts as Random Gibbs Network Forest. There is an *intuitive* mathematical equivalence between *Random Forest* and the Shallow Gibbs. Random Forest is an ensemble learning : a machine learning method that combines many simple learners – base models– to construct an optimized predictive model (*powerful* model).

Random-forest does both row sampling and column sampling with Decision tree as a base learner (Liu et al. (2012); Ali et al. (2012); Rigatti (2017); Liu (2014); Schonlau & Zou (2020); Probst et al. (2019); Svetnik et al. (2003)). Analogically, the Shallow Gibbs does random clusters and rather than column sampling, it projects the data $\boldsymbol{x}$ into a fewer dimensional space on the shallow layer (one hidden layer with *very* few reduced number of neurons).

The variance will decrease in Random Forests, as you increase the number of base learners $\boldsymbol{\tau}$. When you reduce $\boldsymbol{\tau}$, there is a rise in variance. For the whole process, however, bias remains unchanged. It is *commonly* said: Random forest= Decision Trees (as a base learner)+ bagging (Row sampling with replacement)+ feature bagging (column sampling) + aggregation (mean/median, majority vote) [Skurichina & Duin (2001); Munson & Caruana (2009); Biau & Devroye (2010); Sutton et al. (2005); Strobl et al. (2009); Kotsiantis (2011); Panov & Džeroski (2007)]. In the Shallow Potts, when you increase the number of base learners (the number neurons one the hidden layer), the train and test errors increase. To reduce the erros in Shallow Gibbs, we *double* backpropagate it accross the hyper-parameters and parameters of the model [See 4.5.1], it becomes then *powerful* as the N_k Multilayer feedforward neural network, described in section 3.10.

The Shallow Potts as a Mixture Model (As a reminder of some practical computations) Computing expectation $\mathbb{E}(y^{(n+1)}|x^{(n+1)}, \boldsymbol{y}^n, \boldsymbol{x}^n)$ in a proper way would require an exact probability for each partition in 4.15. The value of the prediction $\mathbb{E}(y^{(n+1)}|x^{(n+1)}, \boldsymbol{y}^n, \boldsymbol{x}^n)$ need to adjusted with an estimate of $p(\rho_n|\boldsymbol{y}^n, \boldsymbol{x}^n)p(b_{n+1}|\rho_n, x^{(n+1)}, \boldsymbol{x}^n)$: it simply means that each partition has a probability that need

to be taken into account in the final estimation, plus an estimation of the conditional probability of the new bonds (b_{n+1}). To solve this problem, we have approximated $\hat{p}(\rho_n|\boldsymbol{y}^n, \boldsymbol{x}^n) \approx 1/M$, and have introduced a smoothing parameter $0 \leq \delta \leq 1$ ($\delta = \hat{p}(b_{n+1}|\rho_n, x^{(n+1)}, \boldsymbol{x}^n)$) that will represent an estimate of this probability. This is not the optimal way, in a sense that $\hat{p}(b_{n+1}|\rho_n, x^{(n+1)}, \boldsymbol{x}^n)$ is different from a partition to another one. But, il will help out to transform the final prediction to:

$$\hat{y} \approx \frac{\delta}{M}\mathbb{E}(y^{(n+1)}|, x^{(n+1)}, \boldsymbol{y}^n, \boldsymbol{x}^n)$$

where M is the number of partitions generated during the training step.

4.5.1 On Double backpropagation

Double backpropagation has various used cases. Under the name double backpropagation, the concept of penalizing the output gradient with respect to the input was introduced (Ross & Doshi-Velez, 2018), and later used to locate large connected areas of the error function called flat minima. The ultimate aim was the development of their models generalization, which is our goal here. It has effectively improved generalization performance [Drucker & Le Cun (1992), Drucker & Le Cun (1991)], and has also been applied to other adversarial models ((Seck et al., 2019), (Sun et al., 2020)), in character recognition (Kamruzzaman & Syed (1998), Kamruzzaman et al. (1997)), in robustness and saliency map interpretability (Etmann, 2019). We present in the following line our novel double backpropagation scheme (DBS).

Our Proposed Double backpropagation scheme Similarly to our *Iterative projected gradient* (IPG) applied to our model variational parameter λ_w, we found that the Mean Squared Error (MSE) of our regression model can also be back-propagated [section 3.5] *w.r.t* to each of the model parameter, i.e $\psi = (b^{(1)}, W^{(1)}, b^{(2)}, W^{(2)})$, the main parameters of the network, with $b^{(1)} \in \mathbb{R}^{l_1}$, $W^{(1)} \in \mathbb{M}_{q\times l_1}$, $b^{(2)} \in \mathbb{R}^p$, and $W^{(2)} \in \mathbb{M}_{l_1\times p}$, $l_0 = q$, $l_2 = p$, $\Sigma \in \mathbb{M}_{p\times p}$ being the variance-covariance matrix of y. First, we know that $y|x, \psi, \Sigma$ is distributed as a multivariate normal distribution with mean $f(x) = f_\psi(x) = \mathbb{E}(y|x, \psi)$, and variance Σ. That is, $p(y|x, \psi, \Sigma) = (2\pi)^{-p/2}|\Sigma|^{-1/2}\exp\{-1/2(y - f_\psi(x))'\Sigma^{-1}(y - f_\psi(x))\}$. Then, by applying the sampling method described in the Cholesky decomposition section 1.2.1, we have :

$$\hat{y}_{est} = f_\psi(x) + L \cdot u = [b^{(2)} + g_1(b^{(1)} + xW^{(1)})W^{(2)}] + L \cdot u \tag{4.19}$$

such that $L \in M_d(\mathbb{R})$ is a lower triangular matrix such that $\Sigma = LL^T$, and $u \sim N(0, I)$. The *double* backpropagation scheme for the Shallow Gibbs – **which is another main contribution of this research** – goes as follows:

1. Using the IPG (in 4.14), apply backpropagation method on hyper-parameter λ_w to reduce its Kullback-Leibler (KL) estimation error. Once done, generate an estimate $\hat{\boldsymbol{\psi}}_0$ of $\psi = (b^{(1)}, W^{(1)}, b^{(2)}, W^{(2)})$, using Monte Carlo sampling method from the variational distribution of the parameters.

2. Use equation 4.19 to backpropagate the $MSE(y_i - \hat{y}_i) = \|y_i - \hat{y}_{est,i}\|^2$ to update $\hat{\boldsymbol{\psi}}_0$ in the Potts cluster and **per observation** as follows:

$$\hat{\boldsymbol{\psi}}_{1,i} \longleftarrow \hat{\boldsymbol{\psi}}_0 - \epsilon_{\psi,0}\frac{\partial MSE(y_i - \hat{y}_{est,i})}{\partial\psi} \tag{4.20}$$

$$\hat{\boldsymbol{\psi}}_{t,i} \longleftarrow \hat{\boldsymbol{\psi}}_{t-1,i} - \epsilon_{\psi,t-1}\frac{\partial MSE(y_i - \hat{y}_{est,i})}{\partial \psi} \tag{4.21}$$

where $\epsilon_{\psi,t}$ is the learning rate schedule [See 3.6.2] for this gradient update of ψ at step t, and $\hat{\boldsymbol{\psi}}_{t,i}$ is the value of ψ at iteration t for observation i. Equivalently, for the i-th data x_i in the training data, it corresponds to the changes:

$$\begin{aligned}
b^{(1)}_{i,t} &= b^{(1)}_{i,t-1} - \epsilon_{b^{(1)}_{i,t-1}} \cdot 2I^i_{l_1\times l_1} g_1^{'}(b^{(1)}_{i,t-1} + x_i W^{(1)}_{i,t-1})W^{(2)}_{i,t-1} \star [f_{\psi_{i,t-1}}(x_i) - y_i]^T \\
b^{(2)}_{i,t} &= b^{(2)}_{i,t-1} - \epsilon_{b^{(2)}_{i,t-1}} \cdot 2I^i_{p\times p}[f_{\psi_{i,t-1}}(x_i) - y_i]^T \qquad (4.22)\\
W^{(1)}_{i,t} &= W^{(1)}_{i,t-1} - \epsilon_{W^{(1)}_{i,t-1}} \cdot 2\boldsymbol{x}_i^T[g_1^{'}(b^{(1)}_{i,t-1} + x_i W^{(1)}_{i,t-1})]W^{(2)}_{i,t-1}[f_{\psi_{i,t-1}}(x_i) - y_i]^T \qquad (4.23)\\
W^{(2)}_{i,t} &= W^{(2)}_{i,t-1} - \epsilon_{W^{(2)}_{i,t-1}} \cdot 2[g_1(b^{(1)}_{i,t-1} + x_i W^{(1)}_{i,t-1})]^T[f_{\psi_{i,t-1}}(x_i) - y_i]^T \qquad (4.24)\\
\Sigma_{i,t} &= \Sigma_{i,t-1} - \epsilon_{\Sigma_{i,t-1}} \cdot 2[(L^T + L)^{-1}u_{i,t}][f_{\psi_{i,t-1}}(x_i) - y_i]^T \qquad (4.25)\\
\hat{y}_{est,(i,t)} &= f_{\psi_{i,t}}(x) + L_{i,t}\cdot u_{i,t} = [b^{(2)}_{i,t} + g_1(b^{(1)}_{i,t} + xW^{(1)}_{i,t})W^{(2)}_{i,t}] + L_{i,t}\cdot u_{i,t} \qquad (4.26)
\end{aligned}$$

where $[z]^T$ means the translate of z, $g_1^{'}$ the derivative of g_1, $\psi_{i,t} = (W^{(1)}_{i,t}, W^{(2)}_{i,t}, b^{(1)}_{i,t}, b^{(2)}_{i,t}, \Sigma_{i,t})$ represent the change for ψ for the $i-$th training observation at step t; $\epsilon_{\psi_{i,t}} = (\epsilon_{W^{(1)}_{i,t}}, \epsilon_{W^{(2)}_{i,t}}, \epsilon_{b^{(1)}_{i,t}}, \epsilon_{b^{(2)}_{i,t}}, \epsilon_{\Sigma_{i,t}})$ is the learning rate of those gradients updates; $L_{i,t}$ the Choleski factor at step t from $\Sigma_{i,t}$; and finally $u_{i,t} \sim N(0, I)$. The changes for the $i-$th data in the testing set, is the change of its nearest neighbor in the current cluster *w.r.t* to the Euclidean distance [5].

The product symbol $\star$ means that the vector $[f_{\psi_{i,t-1}}(x_i) - y_i]^T$ will multiply each line of the final computation on his left. We will be calling the optimization scheme above as the Double Backpropagation Scheme – ***DBS optimizer***. The DBD is an universal Bayesian neural network optimizer: given an initial $\psi_{i,0)} = (W^{(1)}_{i,0}, W^{(2)}_{i,0}, b^{(1)}_{i,0}, b^{(2)}_{i,0}, \Sigma_{i,0})$, and $\hat{y}_{est,(i,0)}$, it can make $\hat{y}_{est,(i,t)}$ converge to the right response value y_i with appropriate learning rate schedule. To find this appropriate learning schedule is an open work, and one can look for its convergence properties using the stochastic gradient learning convergence theorems, to show how $\hat{y}_{est,(i,t)}$ converges, and his speed of convergence. We present a Generalized DBS in our concluding remarks in the last chapter.

4.6 Experimental evaluation

In this section, we study the performance of the Potts Gibbs-network mixture model, Potts-Gibbs-NM for short, on real datasets. As a baseline, we compare the performance of Potts-Gibbs-NM with that of the classical feed-forward neural network, and to the multi-layer multi-target regression (MMR) model of Zhen et al. (2017). The performance of the models is measured on datasets from the Mulan Project (Tsoumakas et al., 2011).

[5] We have updated this choice during our experiments.

As described above, there are four variants of the Potss-Gibbs-NM model: the fully-connected Gibbs network or full Gibbs network (Full-Gibbs, for short), the between-layer sparse Gibbs network (B-Sparse-Gibbs), the sparse shallow Gibbs network (Sparse-Gibbs), the sparse compound symmetry Gibbs network (CS-Sparse-Gibbs), and the compound symmetry Gibbs network (CS-Gibbs). The activation function used for all the neural networks in this study is a smooth version of the Rectified Linear Unit or ReLU, namely, $\log(\exp(x)+1) \approx \max\{0, x\}$ (Lee et al., 2019). All the code used in the experiments can be downloaded from the first author's `github` site (Alahassa, 2020).

To appreciate how we tweak each of our model, you need to pay attention on at least 13 important factors:

- NumberPartitions (NP): an integer for the number of Potts model partitions generated;
- Minimum Potts Cluster Size (MPCS): an integer for the minimum size of each cluster in all partitions generated;
- NHLayer (NH): an integer for the number of neurons on the hidden layer (l_1);
- epoch_times (ET): an integer for the number of epochs training times;
- learning_rate (LR): an integer for the learning rate of parameters
- Number_EpLogLike (NE): Number of times we simulate λ_w to estimate the Expected-loglikelihood Score inside the KL to optimize.
- batch_psize (BS): an integer for the proportion of batch training data;
- Simulate_W_b_Pred (SWbP): After estimation of variational λ_w, Number of times We simulate W and b from to get a sample of (W,b) ready to be used for sampling the predicted test data from the model.
- Pred_Simulate_Ytest (PSY): Number of times we simulate Ytest given [each] W and b from our previous sample of (W, b). It means, for each W,b, we sample "Pred_Simulate_Ytest" times the Ytest predicted response data from the model.
- Simulate_Proba_Partition (SPP): After estimation of variational λ_w, Number of times We simulate W and b from to get a sample of (W, b) ready to be used to estimate the probability of acceptation each partition from the model.
- The smooth δ applied to smooth slightly every prediction [see 4.5]
- The number of times we backpropagate for the updates $\psi_{i,t} = (W_{i,t}^{(1)}, W_{i,t}^{(2)}, b_{i,t}^{(1)}, b_{i,t}^{(2)}, \Sigma_{i,t})$ for all the model parameters [see 4.5.1].
- All the DBS parameters (learning rate, number of DBS updates).

The experiments were performed on 5 multivariate multiple regression datasets (Slump, EDM, Jura, Water Quality, and SCPF) taken from the multiple-output benchmark datasets available in the Mulan project website (Tsoumakas et al., 2020). The datasets are shown in Table 2.1.

Evaluation metrics. To directly benchmark with state-of-the-art algorithms, we measure the performance of our Potts neural Gibbs networks with the Mean Squared Error (MSE), and the commonly-used Relative Root Mean Squared Error (RRMSE)

$$\text{RRMSE} = \sqrt{\frac{\sum_{(x_i,y_i)\in\mathcal{D}}(\hat{y}_i - y_i)^2}{\sum_{(x_i,y_i)\in\mathcal{D}}(\hat{Y} - y_i)^2}}$$

where (x_i, y_i), the i-th sample in the testing-set $\mathcal{D}$, is composed of features x_i and ground truth y_i; $\hat{y}_i$ is the model prediction of y_i; and $\hat{Y}$ is the average of the adjusted values $\hat{y}$ over the training-set. We take the average RRMSE across all the response dimensions (target variables) within the testing-set $Dtest$. A lower aRRMSE indicates better performance. The MMR model has already substantially outperformed the best results from state-of-the-art algorithms on most of these eleven datasets. But, one counterfactual thing we have noticed is that the Relative Root Mean Squared Error (RRMSE) is not a measure of goodness of fit.

THE SIZES OF THE TESTING DATA IS 20% OF THE WHOLE DATASET EACH TIME.

The Shallow Gibbs Model is also comparable in section 3.10.

4.6.1 The Results

The MMR Relative Root Mean Squared Error (RRMSE) is beatable, but RRMSE is still not the best... Table 4.1 displays our multi-target prediction performance results on the 3 datasets from the Mulan project (Slump, EDM and SCPF). In the last column of the table you can compare our results with the multi-layer multi-target regression (MMR) RRMSE of Zhen et al. (2017). As you can see, when running the B-Sparse Gibbs model on Slump dataset, we have achieved a better RRMSE of 45.64 in comparison to the MMR model (58.70). This is true for the SCPF dataset, with a better RRMSE of 48.07 in comparison to the MMR model (81.20). Our goal with table 4.1 is to exhibit some datasets for which the RRMSE was low in comparison to the MMR model, but the MSE for the test data was not attractive in comparison to table 3.4. In other words, we have achieved better Relative Root Mean Squared Error predictions, but the Mean Squared Error of our predictions need more work or fine tuning to be competitive.

Table 4.1: Average root mean squared errors RRMSE (%) and and mean squared error MSE Results with the shallow Gibbs Networks

	Shallow Gibbs settings								Measure of good fit S-Gibbs			MMR
Data sets	NH	ET	LR	NE	BS	SWbP	PSY	SPP	RMSE	RMSE	RRMSE	RRMSE
									Train	Test	-	-
Slump (FG)	1	1	$10e(-3)$	5	0.8	5	5	5	1433.99	482.73	**45.64**	58.70
EDM	1	1	$10e(-4)$	5	0.8	5	5	5	7.24	**1.27**	87.93	71.6
SCPF (BG)	1	1	$10e(-4)$	5	0.8	5	5	5	470.14	147.33	**48.07**	81.20

**We have generated 5 Potts partitions with a shrinkage constraint of 5 for each of the dataset, RMSE=Root Mean Squared Error, RRMSE=Relative Root Mean Squared Error, FG= Full Gibbs, BG = B-Sparse Gibbs, No DBS optimization and no δ-smoothing is applied to the data*

A broad view of the Model in terms of Mean Squared Error. We have performed many experiments before adapting and fixing our model parameters and hyper-parameters initialization included. You can check for some tables in appendix chapter [A] for some other results that may bring information about how the model behaves. As a reminder, our goal is to keep the model as simple as possible and propose an efficient estimation that can compete the Classical N_k Hidden Layer Multifeedforward Model in [3.10], which has much more parameters, up to 100 per layer sometimes. The proposed Potts-Gibbs-NM models has not achieved attractive Mean Squared Error compare to the N_k Hidden Layer Multi-feedforward until we have added the **DBS** optimizer [See 4.6.2].

Some Comments about the model structures. We have designed the sparse and B-sparse Gibbs network for training with a small number of observations. As a side-effect, our sparse networks are also computationally fast to train. In fact, we observed in practice that in general, and in a relative comparison to the fully-connected Gibbs network (given the machine used), the sparse network structures speed up training to 2.5x$\geq$ (for sparse-Gibbs) and 1.5x$\geq$ (for B-sparse-Gibbs), all while doing a single training (epochs) at a time, specially for Slump dataset [see 4.3, 4.6, 4.4].

When we look at the sparse network structures, we also observe that the gradients oscillate or vanish less than in the fully-connected and compound symmetry Gibbs Networks. Moreover, we can argue (generally saying) that the sparse structured networks is the most effective at reducing the RRMSE than the fully-connected Gibbs Network, more effective than the compound symmetry Gibbs Network variants [4.3, 4.6, 4.4]. Of course, it depends on the dataset, and this finding is not consistent. However, it is impressive that sparse models can match and beat the performance of more dense networks with fewer weight parameters. Probably, one of the reasons for this behavior is that sparse neural network models are known to capture features useful to uncover broader and more general aspects of the data, resulting in better (learning) prediction (Liu, 2020). As many other works have shown (Srinivas et al., 2017), our work shows that sparse learning in a very complex structures is possible.

The compound symmetry structure imposed on the precision matrix Ω is conceived so as to simplify the model in terms of parsimony. As opposed to the fully-connected Gibbs network for which the unstructured precision matrix specifies no patterns in the weight spatial connection (that is, the precision matrix is completely general), the compound symmetry structure specifies that weights on the same layer have homogeneous (or nearly equal) covariances, and homogeneous (or nearly equal) variances. In practice, this saves a lot in terms of number of parameters, leading to training times up to more faster than the fully-connected network in some cases. Obviously, these results cannot be over-generalized too. There is an open door for research about conceiving a good Bayesian tests (Mulder & Fox, 2013), or other model selection mechanisms, to choose the best suitable model among the shallow Gibbs networks for a given dataset.

Table 4.2: Average root mean squared errors RRMSE (%) and and mean squared error MSE Results with the shallow Gibbs Networks: Sparse-Gibbs

	Shallow Gibbs settings								Measure of fitness S-Gibbs			Time (sec.)
Data sets	NH	ET	LR	NE	BS	SWbP	PSY	SPP	RMSE	RMSE	RRMSE	Time
									Train	Test	-	
Slump	1	1	$10e(-3)$	5	0.8	5	5	5	67.36	**67.36**	99.34	116.12
Jura	1	3	$10e(-3)$	2	0.3	5	5	5	32.94	**35.99**	100.0	1385.67
SCPF	1	1	$10e(-4)$	5	0.8	5	5	5	111.82	**41.30**	116.57	633.94

**We have generated 5 Potts partitions with a shrinkage constraint of 5, RMSE=Root Mean Squared Error, RRMSE=Relative Root Mean Squared Error; DBS epoch times =3, DBS learning rate =* $10e-3$*, and no smoother* δ *is applied*

Table 4.3: Average root mean squared errors RRMSE (%) and and mean squared error MSE Results with the shallow Gibbs Networks: B-Sparse Gibbs

	Shallow Gibbs settings								Measure of fitness S-Gibbs			Time (sec.)
Data sets	NH	ET	LR	NE	BS	SWbP	PSY	SPP	RMSE	RMSE	RRMSE	Time
									Train	Test	-	
Slump	1	1	$10e(-3)$	2	0.8	5	5	5	66.32	**67.23**	99.65	825.62
Jura	1	3	$10e(-2)$	2	0.3	5	5	5	32.57	**35.69**	103.05	2133.79
SCPF	1	1	$10e(-4)$	5	0.8	5	5	5	111.82	**40.25**	136.69	950.19

**We have generated 5 Potts partitions with a shrinkage constraint of 5, RMSE=Root Mean Squared Error, RRMSE=Relative Root Mean Squared Error; DBS epoch times =3, DBS learning rate =* $10e-5$*, and no smoother* δ *is applied*

Table 4.4: Average root mean squared errors RRMSE (%) and and mean squared error MSE Results with the shallow Gibbs Networks: Full-Gibbs

	Shallow Gibbs settings								Measure of fitness S-Gibbs			Time (sec.)
Data sets	NH	ET	LR	NE	BS	SWbP	PSY	SPP	RMSE	RMSE	RRMSE	Time
									Train	Test	-	
Slump	1	1	$10e(-5)$	2	0.8	5	5	5	70.06	**70.68**	98.46	2156.10
Jura	1	3	$10e(-2)$	5	0.8	5	5	5	32.70	**35.70**	110.26	5349.34
SCPF	1	1	$10e(-4)$	5	0.8	5	5	5	111.82	**31.30**	186.87	428.97

**We have generated 5 Potts partitions with a shrinkage constraint of 5, RMSE=Root Mean Squared Error, RRMSE=Relative Root Mean Squared Error; DBS epoch times =3, DBS learning rate =* $10e-5$*, and no smoother* δ *is applied*

Table 4.5: Average root mean squared errors RRMSE (%) and and mean squared error MSE Results with the shallow Gibbs Networks: Sparse-CS-Gibbs

	Shallow Gibbs settings								Measure of fitness S-Gibbs			Time (sec.)
Data sets	NH	ET	LR	NE	BS	SWbP	PSY	SPP	RMSE	RMSE	RRMSE	Time
									Train	Test	-	
Slump	1	1	$10e(-5)$	2	0.8	5	5	5	66.90	**68.21**	100.64	1102.48
Jura	1	3	$10e(-2)$	5	0.8	5	5	5	32.94	**38.39**	190.26	2929.34
SCPF	1	1	$10e(-4)$	5	0.8	5	5	5	111.82	**51.89**	171.82	430.84

**We have generated 5 Potts partitions with a shrinkage constraint of 5, RMSE=Root Mean Squared Error, RRMSE=Relative Root Mean Squared Error; DBS epoch times =3, DBS learning rate =* $10e-5$*, and no smoother* δ *is applied*

Table 4.6: Average root mean squared errors RRMSE (%) and and mean squared error MSE Results with the shallow Gibbs Networks: CS-Gibbs

	Shallow Gibbs settings								Measure of fitness S-Gibbs			Time (sec.)
Data sets	NH	ET	LR	NE	BS	SWbP	PSY	SPP	RMSE	RMSE	RRMSE	Time
									Train	Test	-	
Slump	1	1	$10e(-5)$	2	0.8	5	5	5	70.06	**70.68**	98.46	456.10
Jura	1	3	$10e(-2)$	5	0.8	5	5	5	32.70	**45.70**	110.26	1349.34
SCPF	1	1	$10e(-4)$	5	0.8	5	5	5	111.82	**41.30**	116.57	798.09

**We have generated 5 Potts partitions with a shrinkage constraint of 5, RMSE=Root Mean Squared Error, RRMSE=Relative Root Mean Squared Error; DBS epoch times =3, DBS learning rate =* $10e-5$*, and no smoother* δ *is applied*

4.6.2 Convergence of the DBS optmizer.

We have found experimentally that the **DBS** optimizer is convergent. We have choosen **Slump** dataset to illustrate this finding, look for table [4.7] to be convainced.

To reach this convergence, we have identify the need to modify this optimizer with a slight correction. Mainly $\frac{\partial MSE(y_i - \hat{y}_{est,i})}{\partial \hat{y}_{est,i}}$ has to integrate the updates system as follows:

$$\hat{\boldsymbol{y}}_{est,(i,t)} \longleftarrow f_{\psi_{i,t}}(x) + L_{i,t} \cdot u_{i,t} - \epsilon_{\hat{\boldsymbol{y}}_{est,(i,t)}} \frac{\partial MSE(y_i - \hat{y}_{est,i})}{\partial \hat{\boldsymbol{y}}_{est,i}} \tag{4.27}$$

where $\frac{\partial MSE(y_i - \hat{y}_{est,(i,t)})}{\partial \hat{\boldsymbol{y}}_{est,i}} = 2 * (y_i - \hat{\boldsymbol{y}}_{est,(i,t)})$.

But, at step t in the implementation, $2 * (y_i - \hat{\boldsymbol{y}}_{est,(i,t)})$ is replaced by $2 * (y_i - \hat{\boldsymbol{y}}_{est,(i,t-1)})$. At initial step, $\hat{\boldsymbol{y}}_{est,(i,0)}$ is computed using the model equation and $\hat{\boldsymbol{\psi}}_0$.

So each training data has its own learning rate which is here set as $\epsilon_{\hat{\boldsymbol{y}}_{est,(i,t)}}$. This is valuable for each test data as follows:

$$\hat{\boldsymbol{y}}^{\text{test}}_{est,(i,t)} \longleftarrow f_{\psi_{i,t}}(x^{\text{test}}) + L_{i,t} \cdot u_{i,t} - \epsilon^{\text{test}}_{\hat{\boldsymbol{y}}_{est,(i,t)}} \frac{\partial MSE(y_i - \hat{y}_{est,i})}{\partial \hat{\boldsymbol{y}}_{est,i}} \tag{4.28}$$

where for the k-th test data x^{test}_k the changes $f_{\psi_{i,t}}, L_{i,t}, u_{i,t}$ are taken from the j-th training data y_j which verify:

$$j^{\text{choosen}} = \underset{x_j \in \text{ Training Set}}{\arg\min} \text{ Mean}(x_j - x^{test}_k) \tag{4.29}$$

where the operation Mean(u) for vector u is taken upon all dimension of u. In our experiments with **Slump** dataset, we have choosen $\epsilon^{\text{test}}_{\hat{\boldsymbol{y}}_{est,(i,t)}}$ and $\epsilon_{\hat{\boldsymbol{y}}_{est,(i,t)}}$ to be double time larger than:

$$\epsilon_{\psi_{i,t}} = (\epsilon_{W^{(1)}_{i,t}}, \epsilon_{W^{(2)}_{i,t}}, \epsilon_{b^{(1)}_{i,t}}, \epsilon_{b^{(2)}_{i,t}}, \epsilon_{\Sigma_{i,t}})$$

In equation [4.31], $\frac{\partial MSE(y_i - \hat{y}_{est,i})}{\partial \hat{\boldsymbol{y}}_{est,i}}$ has to be approximated as :

$$\frac{\partial MSE(y_i - \hat{y}_{est,i})}{\partial \hat{\boldsymbol{y}}_{est,i}} \approx 2 * (y^{\text{train}}_{j^{\text{choosen}}} - \hat{\boldsymbol{y}}^{\text{train}}_{est,(j^{\text{choosen}},t)}). \tag{4.30}$$

When the training error was getting lower, with a not decreasing test error, we double again $\epsilon^{\text{test}}_{\hat{\boldsymbol{y}}_{est,(i,t)}}$ compared to $\epsilon^{\text{train}}_{\hat{\boldsymbol{y}}_{est,(i,t)}}$ specifically. The convergence property of the **DBS** optimizer in equation [4.31]

for the training data is not surprising due to the convergence of gradient descent [Lee et al. (2016); Ruder (2016)] and Stochastic Gradient Descent properties for the Mean Squared Error $MSE(y_i - \hat{y}_{est,i}) = \|y_i - \hat{y}_{est,i}\|_F^2$ which is convex in $\hat{y}_{est,i}$. However, because of the approximation [4.30], this convergence is not reached with the same *speed* for the test data, but the model still do converge. In their case, to reduce test errors to (zero), a last ingredient needs to be applied, called *Universal Approximation Theorem*:

Proposition 4.6.1 (Guilhoto (2018))
A Neural Network with a single hidden layer and sufficiently many nodes is capable of approximating any continuous function.

Intuitively, it means that we need to increase the number of neurons within the hidden layer, to reach a good prediction for the test data. This is also valid if the model was multi-layer architecture. To understand this intuition, look for equation [4.31], and beware that when convergence is reached for the training data, we have :

$$\frac{\partial MSE(y - \hat{y}_{est,(i,t)})}{\partial \hat{\boldsymbol{y}}_{est}} = 2 * (y_i - \hat{\boldsymbol{y}}_{est,(i,t)}) = 0$$

and equation [4.31] for the test data x^{test} becomes:

$$\hat{\boldsymbol{y}}^{\text{test}}_{est,(i,t)} \longleftarrow f_{\psi_{i,t}}(x^{\text{test}}) + L_{i,t} \cdot u_{i,t} \tag{4.31}$$

We simply need then to find the right (neural) machine that will associate with perfection x^{test} to y^{test}, as:

$$\hat{y}^{\text{test}}_{est,(i)} = f_{\psi_i}(x) + L_i \cdot u_i = [b_i^{(2)} + g_1(b_i^{(1)} + x_i^{\text{test}} W_i^{(1)}) W_i^{(2)}] + L_i \cdot u_i$$

Or simply, very like an encoder or neural cryptography machine [Ruttor et al. (2007), Ruttor et al. (2004), Volná (2000), Kanter & Kinzel (2003), Pattanayak & Ludwig (2017), Hadke & Kale (2016), Sharma et al. (2019), Kinzel & Kanter (2002a), Kinzel & Kanter (2002b), Dong & Huang (2019), Volna et al. (2012), Klimov et al. (2002), Godhavari et al. (2005), Blackledge et al. (2015), Baird et al. (2005), Behrmann et al. (2019)]:

$$x_i^{\text{test}} \underset{y_i^{\text{test}}}{\longleftrightarrow} \psi_i^{\text{test}} = (W_i^{(1)}, W_i^{(2)}, b_i^{(1)}, b_i^{(2)}, \Sigma_i) \tag{4.32}$$

To obtain this machine ψ_i^{test} which has to encode as much as information to be efficient, we need to extract it from its associate training data in equation[6] [4.29], and to make it robust, the *Universal Approximation Theorem* [see 4.6.1] requires $[W_i^{(1)}, W_i^{(2)}, b_i^{(1)}]$ to be high dimensional[7], i.e. an appropriate number of neurons on the hidden layer. This last requirement transforms the **DBS** optimizer into another **DBS** optimizer [the combination of *Universal Approximation Theorem* and **DBS** optimization] with an

[6] This equation can be improved too for better Train-Test association.
[7] Where the model can also be multi-layer as well.

optimal number l_1 of neurons on the hidden layer [see 4.1] of the Shallow net, an adapted number $\boldsymbol{\zeta}$ of **DBS** optimization, an optimal DBS learning rate ϵ_{dbs}, called the $(l_1, \boldsymbol{\zeta}, \epsilon_{dbs})-$**DBS**, and which, combined with findings of chapter [5], i.e, the ***(dist)*-Nearest Neighbor-(h)-Taylor Series-Perfect Multivariate Interpolation (*dist*-NN-(h)-TS-PMI)** presented in [5.3], is the Perfect fit[8] (or the Perfect learning) for the Shallow Gibbs Network, summarized in equation [4.33]:

$$\lim_{\boldsymbol{l}_{1,\text{opt}},\boldsymbol{\zeta}_{\text{opt}},\boldsymbol{\epsilon}_{\text{dbs,opt}},\text{dist}_{\text{opt}},\boldsymbol{h}_{\text{opt}}} (MSE^{\text{Train}}, MSE^{\text{Test}}) = (0,0) \tag{4.33}$$

where MSE^{Train}, MSE^{Test} are the Mean Squared Error of the train and test data respectively, $dist_{opt}$ is the optimal distance for the research of the nearest neighbor in the training dataset for each test data x_i^{test}, h_{opt} is the optimal order of the Taylor approximation for the Perfect Multivariate Interpolation (*dist*-NN-(h)-TS-PMI) model once the $(l_1, \boldsymbol{\zeta}, \epsilon_{dbs}) -$ **DBS** has overfitted the training dataset. $\boldsymbol{l}_{1,\text{opt}}, \boldsymbol{\zeta}_{\text{opt}}$ are respectively the optimal number of hidden neurons, and the optimal number of **DBS** updates. ϵ_{dbs} integrates simultaneously the DBS learning rate vector for all the model parameters, the DBS learning rate for the training data, and the DBS learning rate for the test data. $\boldsymbol{\epsilon}_{\text{dbs,opt}}$ is the optimal one. We have performed two more experiments to confirm this fact, summarized in table [4.8]. Table [4.8] may be treated as over-fitting, but the advantage of this finding is a confirmation of the convergence $(\downarrow 0)$ of the with $(l_1, \boldsymbol{\zeta}, \epsilon_{dbs}) -$ **DBS**, where we may applied a simple optimization –[Grid Search or Golden Section Search method]– to find the perfect parametrization of $(l_1, \boldsymbol{\zeta})$ and the required learning rate ϵ_{dbs}.

We have also found in practice, that when l_1 and $\boldsymbol{\zeta}$ are too large, the model may diverge. So, it is important to find the right ratio $\boldsymbol{\zeta}/l_1$ (with appropriate **DBS** learning rate).

When there is enough training data available, another way to increase the convergence speed for the test data is to modify the criteria used in optimization [4.29], and use a distance $dist$ for which each test data x_k^{test} is ensured to find an associate x_i^{train} in the training data with :

$$dist(x_k^{\text{test}}, x_i^{\text{train}}) \leq \varepsilon \tag{4.34}$$

where ε is a very small number.

Model Execution Time. The results we have presented above are relatively comparable in terms of execution times, simply because during all the simulations and model trials, we have used different machines with different capacities, sometimes not comparable in terms of computation and degree of availability to run large or huge multiple simulations at a time [look in table 4.9]. Simply to say, our time comparison should be taken as relative for it require precaution and regards about the machine used to run the model, not absolute. Also, whatever the machine being used, increasing the training times (ET==epoch_times) for the neural networks in each cluster of the Shallow Gibbs model is costly in CPUs [Figure 4.2]. The CPUs available are overfull in less than 20 minutes of execution. To solve this problem, we have applied ***Reusable Process Pool Executor*** which is a novel framework that combines threading and multiprocessing primitives for robust concurrent futures (McCullagh (2017)).

[8] Chapter [5] is added to reinforce the $(l_1, \boldsymbol{\zeta}, \epsilon_{dbs}) -$ **DBS** model.

Table 4.7: Average root mean squared errors RRMSE (%) and and mean squared error MSE Results with the shallow Gibbs Networks on Slump dataset with DBS optimization

Shallow Gibbs settings with DBS applied on Slump dataset					Measure of fitness Sparse-Gibbs with Slump dataset			Time (sec.)
NH	ET	LR	DBS	DBS learning rate	RMSE	RMSE	RRMSE	Time
					Train	Test	-	
1	1	$10e(-5)$	3	$10e(-3)$	66.60	**67.36**	99.34	116.12
1	1	$10e(-5)$	10	$10e(-3)$	46.35	**50.06**	99.15	100.02
1	1	$10e(-5)$	30	$10e(-3)$	43.56	**49.13**	102.85	4239.41
1	1	$10e(-5)$	30	$10e(-3)$	43.56	**49.13**	102.85	4239.41
1	1	$10e(-5)$	30	$\mathbf{10e(-6)}$	47.66	**53.13**	102.51	4300.36
1	1	$10e(-5)$	50	$10e(-3)$	14.20	**30.24**	130.59	9358.63
1	1	$10e(-5)$	70	$10e(-3)$	9.20	**27.19**(*)	128.11	10609.91

**We have generated 5 Potts partitions with a shrinkage constraint of 5, RMSE=Root Mean Squared Error, RRMSE=Relative Root Mean Squared Error; DBS epoch times =3,* $NE = 2, BS = 0.8$ *,* $SWbP = PSY = SPP = 5$*, DBS learning rate =* $10e - 2$*, and no smoother* δ *is applied; test data DBS learning rate =* $2 * (10e - 2)$*, (*) the DBS learning rate of the test data is twice the learning rate of the training data*

Table 4.8: Average root mean squared errors RRMSE (%) and and mean squared error MSE Results with the shallow Gibbs Networks on Slump dataset with DBS optimization

Shallow Gibbs settings with DBS applied on Slump dataset					Measure of fitness Sparse-Gibbs with Slump dataset			Time (sec.)
NH	ET	LR	DBS	DBS learning rate	RMSE	RMSE	RRMSE	Time
					Train	Test	-	
6	1	$10e(-5)$	3	$10e(-2)$	2.05	**26.30**(*)	126.20	6814.34
10	1	$10e(-5)$	40	$10e(-2)$	3.11	**25.89**(*)	127.26	5894.04

**We have generated 5 Potts partitions with a shrinkage constraint of 5, RMSE=Root Mean Squared Error, RRMSE=Relative Root Mean Squared Error; DBS epoch times =3,* $NE = 2, BS = 0.8$ *,* $SWbP = PSY = SPP = 5$*, DBS learning rate =* $10e - 2$*, and no smoother* δ *is applied; test data DBS learning rate =* $2 * (10e - 2)$*, (*) the DBS learning rate of the test data is twice the learning rate of the training data*

Table 4.9: Machines used during our simulation and their main characteristics

Machine Host	Cpus	C.L.	Mem	Mem L	GPU	G. Nom	CPU S	CPU M
simulation7	24	10	251	133	-	-	13030	312720
simulation8	24	12	251	111	-	-	13030	312720
venice	12	8	15	8	0	GTX 1050	15971	191652
jupiter	8	5	11	6	-	-	5200	41600
fox	12	7	15	5	0	GTX 1050	15971	191652
acapulco	12	3	15	2	0	GTX 1050	15971	191652
panthere	12	8	15	2	0	GTX 1050	15971	191652
lion	12	0	15	1	0	GTX 1050	15971	191652
jaguar	12	7	15	1	0	GTX 1050	15971	191652

Cpus	=>	Cores	C.L.	=>	Free Cpu
Mem	=>	Total Memory	Mem L	=>	Available Memory (Go)
GPU	=>	GPUs used	G. Nom	=>	GPU model
CPU S	=>	Cpu Mark single thread	CPU M	=>	Cpu Mark total multi threads

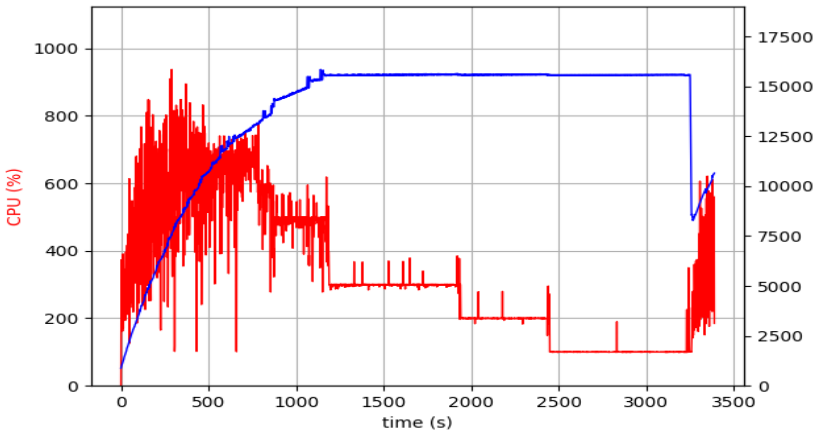


Figure 4.2: Percentage of CPUs charged by the Shallow Gibbs during execution for a training times ≥ 10. The right axis is RAM memory in Mo.

Chapter 5

Nearest Neighbor Multivariate Interpolation (NNMI)

In chapter [4], equations [4.29] and [4.34] indicate the need for an adapted Train-test association function [4.29] for the prediction of $\boldsymbol{y}^{\text{test}}$ when we know $\boldsymbol{x}^{\text{test}}$. This form of data closest neighborhood association is one limitation of the **DBS** optimizer, since it *simply* implies that we are still searching for the right *learner*.

5.1 Interpolation as a Machine *learner*

You can notice with table [4.8], that in few iterations, the **DBS** can overfit the training data. The main question then is: knowing a perfect relation [4.32] for any train data, as $(x_j^{\text{train}}, \psi_j^{\text{train}})$:

$$x_j^{\text{train}} \overset{\boldsymbol{\iota}}{\longleftrightarrow} \psi_j^{\text{train}} = (W_j^{(1)}, W_j^{(2)}, b_j^{(1)}, b_j^{(2)}, \Sigma_j), y_j^{\text{train}} \tag{5.1}$$

where $\boldsymbol{\iota}$ is our denoted *gate* function, how can we find the perfect ψ_i^{test} for the test data as:

$$x_i^{\text{test}} \overset{\boldsymbol{\iota}}{\longleftrightarrow} \psi_i^{\text{test}} = (W_i^{(1)}, W_i^{(2)}, b_i^{(1)}, b_i^{(2)}, \Sigma_i), y_i^{\text{test}} \tag{5.2}$$

Or simply said, is Nearest Neighborhood enough to find $\psi_i^{\text{test}}, y_i^{\text{test}}$, for a given x_i^{test}?

This is where Multivariate Interpolation come into action, as the later can be taken also as a machine *learner*, because it can refine Nearest Neighborhood Train-Test association [4.29]. To perceive this, we present another perspective of machine learning problems with the following described properties about interpolation.

Linear interpolation usually requires two data points (u_a, v_a) and (v_b, v_b), and at the point (u, v), the interpolation equation is given by:

$$v = v_a + (v_b - v_a)\frac{u - u_a}{u_b - u_a} \tag{5.3}$$

Equation [5.3] specifies that the current line slope between (u_a, v_a) and (u, v) is equivalent to the line slope between (u_a, v_a) and (u_b, v_b). The method have been used since a while in the literature [Joarder (2001), Dressler (2009), Otsuki et al. (2004), Guo-qiang et al. (2004), Kay (1983), Bourke (1999)]. When we know u, interpolation is useful to estimate v. But, it may be a poor estimator sometimes [Maeland (1988), Dumitru et al. (2013)].

To generalize equation [5.3] to a learning problem, remember the Taylor's theorem for a multivariate function $\boldsymbol{\iota}$ in functions analysis theory [Giaquinta & Modica (2010), Browder (2012), Zorich (2016), Choquet-Bruhat et al. (1982), Pettersson (1992), Barbarossa & Manzonetto (2019), Cheney & Kincaid (2012), Canuto & Tabacco (2015)]. We already know the best linear approximation to $\boldsymbol{\iota}$. It involves the derivative $\boldsymbol{D\iota}(\mathbf{a})$ such as:

$$\boldsymbol{\iota}(\mathbf{x}) \approx \boldsymbol{\iota}(\mathbf{a}) + \boldsymbol{D\iota}(\mathbf{a}) \circ (\mathbf{x} - \mathbf{a}) \tag{5.4}$$

where $D\boldsymbol{\iota}(\mathbf{a})$ is the matrix of partial derivatives of $\boldsymbol{\iota}$ evaluated in the neighborhood of $\mathbf{a}$, and $\circ$ is the dot product between both vectors $\boldsymbol{D\iota}(\mathbf{a})$ and $(\mathbf{x} - \mathbf{a})$. This approximation is linear and represents the first-order Taylor polynomial [Reimer (2012), Guessab et al. (2006), Phillips (2003), Dikusar (2016), Neumaier (2003), Cohen & Tan (2012), Smyth (2014)]. Proposition [5.2.1] is the generalization of approximation [5.4].

5.2 Multivariate Interpolation

Proposition 5.2.1 (Multivariate version of Taylor theorem (Qian (2011)))
Let $\boldsymbol{\iota} : \mathbf{R}^n \to \mathbf{R}$ *be a* $m \geq \Xi \geq 2$ *-times-differentiable function in a certain vicinity* D *of the point* $\mathbf{u_0} = (u_{01}, ..., u_{0n}) \in \mathbf{R}^n$. *Then, for any* $v = (v_1, ..., v_n)$, *we have :*

$$\begin{aligned}
&\boldsymbol{\iota}(v_1, \ldots, v_n) = \boldsymbol{\iota}(u_{01}, \ldots, u_{0n}) + \sum_{i=1}^{n} \frac{\partial \boldsymbol{\iota}}{\partial u_i}\Big|_{(u_{01},\ldots,u_{0n})} (v_i - u_{0i}) \\
&+ \frac{1}{2!}\sum_{i=1}^{n}\sum_{j=1}^{n} \frac{\partial^2 \boldsymbol{\iota}}{\partial u_i \partial u_j}\Big|_{(u_{01},\ldots,u_{0n})} (v_i - u_{0i})(u_j - v_{0j}) \\
&+ \cdots + R_m(v - u_0)
\end{aligned}$$

where $R_m(v - u_0)$ *is a reminder function of* $v - u_0$.

Remark
It is not a coincidence that Taylor Approximation theorem is only defined in a certain vicinity or a given neighborhood set! One of the main contribution of this chapter is to understand that : Multivariate Interpolation using Taylor theorem is the refined *generalization of simple Neighborhood Train-Test association. Notice in equation [5.4], if* $\boldsymbol{a} = \boldsymbol{x}^{train}$ *and* $\boldsymbol{x} = \boldsymbol{x}^{test}$, *we have* $\boldsymbol{\iota}(\boldsymbol{x}^{test}) \approx \boldsymbol{\iota}(\boldsymbol{x}^{train})$,

when we suppose $\boldsymbol{x}^{train} \approx \boldsymbol{x}^{test}$. *To avoid that simple approximation, we add more differential terms, as exposed in Proposition [5.2.1].*

Can we overfit with proud? It is important to also understand the case $\boldsymbol{\iota} : \mathbb{R}^q \to \mathbb{R}^m$ in [5.1], and how we expand $D\boldsymbol{\iota}(\boldsymbol{\zeta})$ at $\zeta = x_0^{\text{train}}$. When $\boldsymbol{\iota}$ is multivariate, we will use Proposition [5.2.1], or in a one order [5.4], to find [5.2] per component using the assumption $(\mathcal{F}_d)$ that :

There is an approximable differentiable function $\boldsymbol{\iota}$ *such as:* $\boldsymbol{\iota}(x_i^{test}) = (\psi_i^{test}, y_i^{test})$, *where we already know in a perfect overfitting DBS configuration* $(x_{j_1}^{train}, \psi_{j_1}^{train}, y_{j_1}^{train})$ *[i.e* $\boldsymbol{\iota}(x_{j_1}^{train}) = (\psi_{j_1}^{train}, y_{j_1}^{train})$*] and* $(x_{j_2}^{train}, \psi_{j_2}^{train}, y_{j_2}^{train})$ *[i.e* $\boldsymbol{\iota}(x_{j_2}^{train}) = (\psi_{j_2}^{train}, y_{j_2}^{train})$*] for two training data* j_1 *and* j_2, *With the precision that the training data point* j_2 *and the test data* i *are taken in a very closed neighborhood of* j_1.

In each Potts partition ρ_n clusters, this is applicable. Suppose $x_{j_1}^{\text{train}} = (x_{j_1}^{\text{train}}(1), ..., x_{j_1}^{\text{train}}(q))$, and $x_{j_2}^{\text{train}} = (x_{j_2}^{\text{train}}(1), ..., x_{j_2}^{\text{train}}(q))$, with the precision that the training data point j_2 is a taken in a very closed neighborhood of j_2 (potential example is: the associate Potts cluster of j_1 given a partition ρ_n). In practice, for $x = x_j^{\text{test}}$ and $a = x_{j_1}^{\text{train}}$, we compute $\boldsymbol{D\iota}(\mathbf{a}) \circ (\mathbf{x} - \mathbf{a})$ in approximation [5.4] as:

$$D\mathcal{C}_{\boldsymbol{\iota} j} \quad (x_{j_1}^{\text{train}}) \quad \circ (x_j^{\text{test}} - x_{j_1}^{\text{train}}) = \tag{5.5}$$
$$\sum_{k=1}^{q} \quad \frac{\mathcal{I}(x_{j_1}^{\text{train}}(k) \neq x_{j_2}^{\text{train}}(k))(x_j^{\text{test}}(k) - x_{j_2}^{\text{train}})(k)}{x_{j_1}^{\text{train}}(k) - x_{j_2}^{\text{train}}(k) + \mathcal{I}(x_{j_1}^{\text{train}}(k) = x_{j_2}^{\text{train}}(k))} [\mathcal{C}_{\boldsymbol{\iota} j_1}^{(1)} - \mathcal{C}_{\boldsymbol{\iota} j_2}^{(1)}]$$

where $\mathcal{C}_{\boldsymbol{\iota}} \in \{W^{(1)}, W^{(2)}, b^{(1)}, b^{(2)}, \Sigma, y^{\text{train}}\}$, $\mathcal{I}(A)$ equal 1 if condition A is verified. Those constraints $\mathcal{I}(x_{j_1}^{\text{train}}(k)! = x_{j_2}^{\text{train}}(k))$ and $\mathcal{I}(x_{j_1}^{\text{train}}(k) = x_{j_2}^{\text{train}}(k))$ are useful to indicate that we do not want the partial derivatives to be computed if there are no differentiation available for the k th dimension. Further, we shall reduced the Taylor series in [5.2.1] to integrate all those constraints, and the final computation will be called *reduced* derivative.

To use this technique of Reduced Taylor Series Multivariate Interpolation in [5.5] with the research of the nearest neighbor in the train dataset for a new $\boldsymbol{x}^{\text{test}}$, is what we call the **(*dist*)-Nearest Neighbor-(h)-Taylor Series-Reduced Multivariate Interpolation (*dist*-NN-(h)-TS-RMI)** in reference to previous existent work [Rukundo & Cao (2012)], and it can reduce (possibly) test error, specially when we overfit sometimes. In *dist*-NN-(h)-TS-RMI, *dist* is set for the Nearest Neighbor distance applied [e.g 4.29], and h is the order of computed Taylor approximation in [5.5]. Assumption [5.2] denoted $(\mathcal{F}_d)$ may not be valid, but the method may be effective for some datasets; or another situation that could arise is that the closest data point j_2 to j_1 in a very small neighborhood set may not exist in the training data. Some experiments are required to confirm in which case the proposed *dist*-NN-(h)-TS-RMI model is a *good* learner.

Taylor Theorem can approximate any (differentiable) function. Two questions of interest may arise then:

$\mathfrak{Q}_1$ **What is the statistical distribution of any *dist*-Nearest Neighbor-(h)-Interpolated-Reduced**

response variable y, as the parameters of the model are shifted (by interpolation and reduction) ?

$\mathfrak{Q}_2$ Analogious to the *Universal Approximation Theorem* [see 4.6.1], can higher order Taylor Theorem solve any machine learining problem?

Taylor Approximation is the future of Machine learning.

We can use Taylor Theorem [5.2.1] to develop higher order ($h \geq 2$) Multivariate Interpolation. In fact, Taylor Series can also be used to approximate any function. Formula [5.5] also opens the door to empirical derivatives computation. Empirical differentiation reduces or deletes the terms where differentiation is not applicable. This is done to signify that the higher-order differentiated parts represent a *small* correction of the Nearest Neighbor associated values. This technique is valid because the reminder terms in proposition [5.2.1] get smaller when the differentiation order increases.

This is a great revolution in *Statistical Learning Theory*, as we can sufficiently (i.e $h \geq 2$) interpolate any machine learning regression problem, with the *reduced* differentiation form when we know the perfect parameters that fit the training data. By then, the famous **overfitting** problem will not appear to be an hindrance anymore, but an extraordinary gateway to another type of *learning* method, which is therefore the research of the perfect fit for both the training and the test data, using as e.g the proposed *dist*-NN-(h)-TS-RMI model.

5.3 Data Augmentation for Empirical Differentiation (DAED)

This section is a trial to answer question $\mathfrak{Q}_2$, as Taylor Approximation can truly solve any machine learning problem. We illustrate this fact using data augmentation. To make the assumption $(\mathfrak{F}_d)$ valid, we need to create more samples from existent training data.

To understand this intuition, remember that the partial derivative of a function $\boldsymbol{\iota}(x_1, \ldots, x_n)$ in the direction x_i at the point $(e_1, \ldots, e_n)$ is defined by:

$$\frac{\partial \boldsymbol{\iota}}{\partial x_i}(e_1, \ldots, e_n) = \lim_{\delta \to 0} \frac{\boldsymbol{\iota}(e_1, \ldots, e_i + \delta, \ldots, e_n) - \boldsymbol{\iota}(e_1, \ldots, e_i, \ldots, e_n)}{\delta} \tag{5.6}$$

with $\delta \in \mathbb{R}$ has to be a very small real number. Because the closed neighborhood training data j_2 for j_1 in the differentiation computation [5.5] is not available in practice in the training data, we can create more data as follows, to compute derivative [5.6] and [5.5] with almost exact precision:

$$x_{j_1}^{\text{train}} = (x_{j_1}^{\text{train}}(1), ..., x_{j_1}^{\text{train}}(q)) \longrightarrow \begin{cases} (x_{j_1}^{\text{train}}(1) + \delta, ..., x_{j_1}^{\text{train}}(q)) \\ (x_{j_1}^{\text{train}}(1), x_{j_1}^{\text{train}}(2) + \delta, ..., x_{j_1}^{\text{train}}(q)) \\ \cdots\cdots\cdots \\ (x_{j_1}^{\text{train}}(1), x_{j_1}^{\text{train}}(2), ..., x_{j_1}^{\text{train}}(q) + \delta) \end{cases} \tag{5.7}$$

where in [5.7], δ is added on each component of $x_{j_1}^{\text{train}}$ in each of his augmented version. When we

have the model $\boldsymbol{\iota}$ that overfit [e.g the **DBS** optimizer], we have to re-train the model with the augmented data [5.7] to find [5.8]:

$$\begin{cases} \boldsymbol{\iota}(x_{j_1}^{\text{train}}(1), ..., x_{j_1}^{\text{train}}(q)) \\ \boldsymbol{\iota}(x_{j_1}^{\text{train}}(1) + \delta, ..., x_{j_1}^{\text{train}}(q)) \\ \boldsymbol{\iota}(x_{j_1}^{\text{train}}(1), x_{j_1}^{\text{train}}(2) + \delta, ..., x_{j_1}^{\text{train}}(q)) \\ \qquad\qquad \cdots\cdots\cdots \\ \boldsymbol{\iota}(x_{j_1}^{\text{train}}(1), x_{j_1}^{\text{train}}(2), ..., x_{j_1}^{\text{train}}(q) + \delta) \end{cases} \tag{5.8}$$

This data augmentation framework, the DAED, transforms the *dist*-NN-(h)-TS-RMI model in [5.2] into another one, which is the : **(*dist*)-Nearest Neighbor-(h)-Taylor Series-Perfect Multivariate Interpolation (*dist*-NN-(h)-TS-PMI)**. We have also find out that in practice, the reduced model *dist*-NN-(h)-TS-RMI was not precised, and not effective differentiation approach. We have run some experiments with Slump dataset, we still found the **DBS** as a model that overfit the training data, but the reduced *dist*-NN-(h)-TS-RMI model can not generalize very well. The Perfect Multivariate Interpolation method, let say the *dist*-NN-(h)-TS-PMI) model, is recommended.

Using limit [5.6], and a re-trained model where we know [5.8], we can compute:

$$\frac{\partial \boldsymbol{\iota}}{\partial x_j(k)} \left(x_j^{\text{train}}(1), \ldots, x_j^{\text{train}}(q) \right) \approx \frac{\boldsymbol{\iota}\left(x_j^{\text{train}}(1), \ldots, x_j^{\text{train}}(k) + \delta, \ldots, x_j^{\text{train}}(q) \right) - \boldsymbol{\iota}\left(x_j^{\text{train}}(1), \ldots, x_j^{\text{train}}(k), \ldots, x_j^{\text{train}}(q) \right)}{\delta} \tag{5.9}$$

with a very small δ [e.g 10^{-10}, $10^{-}50$], to get :

$$\begin{aligned} D\boldsymbol{\iota}(x_j^{\text{train}}) \quad = \quad & \left(\frac{\partial \boldsymbol{\iota}}{\partial x_j(1)} \left(x_j^{\text{train}}(1), \ldots, x_j^{\text{train}}(q) \right), \right. \\ & \left. \ldots, \frac{\partial \boldsymbol{\iota}}{\partial x_j(k)} \left(x_j^{\text{train}}(1), \ldots, x_j^{\text{train}}(q) \right), \ldots, \frac{\partial \boldsymbol{\iota}}{\partial x_j(q)} \left(x_j^{\text{train}}(1), \ldots, x_j^{\text{train}}(q) \right) \right) \end{aligned} \tag{5.10}$$

$D\boldsymbol{\iota}(x_j^{\text{train}})$ presented above is the best mathematical and empirical Multivariate function Derivative that you can compute for $\boldsymbol{\iota}$ and evaluate at x_j^{train}.

We suppose that the augmented training data in [5.7], being denoted $x_j^{\text{aug-train}}$, are taken in very close neighborhood of $x_{j_1}^{\text{train}} = (x_{j_1}^{\text{train}}(1), ..., x_{j_1}^{\text{train}}(q))$. So, **before the re-training step**, to compute the associated $y_j^{\text{aug-train}}$ for each $x_j^{\text{aug-train}}$, we may use $\psi_{j_1}^{\text{train}}$, as illustrated in [5.1].

Iterative Multivariate Interpolation Method. We want to highlight a strong link between Gradient descent and Taylor Approximation [HAMMER (2017)]. There is then possibility to refine the Perfect Multivariate Interpolation in [5.3], so that it can be iterated, most likely in a Gradient Descent Optimization Scheme, e.g illustrated by Lydia & Francis (2019).

5.4 Generalization Method

In the previous section, we see how data can be augmented. Now, it is time to reveal how we generalize the model in order to make any test data to have its associate neighbor in the training data. The secret is to understand that every augmented data can also be sufficiently differentiated, as much as we want to overlap all the possible data Space domain:

$$x_{j_1}^{\text{aug-train}} = (x_{j_1}^{\text{train}}(1) + \delta, ..., x_{j_1}^{\text{train}}(q)) \longrightarrow \begin{cases} (x_{j_1}^{\text{train}}(1) + s_1 \cdot \delta, ..., x_{j_1}^{\text{train}}(q)) \\ (x_{j_1}^{\text{train}}(1), x_{j_1}^{\text{train}}(2) + s_2 \cdot \delta, ..., x_{j_1}^{\text{train}}(q)) \\ ..... \\ (x_{j_1}^{\text{train}}(1), x_{j_1}^{\text{train}}(2), ..., x_{j_1}^{\text{train}}(q) + s_q \cdot \delta) \end{cases} \tag{5.11}$$

with $(s_1, s_2, ..., s_q)$ is a q-tuple of integers or rationals. The re-training process has to be done progressively to obtain their perfect associated parameters. The last, but not the least, is that each novel augmented data can again be differentiated as done in [5.11], (again and again) with no end.

We also found that in practice, this generalisation method is effective for slump dataset, but it takes many data augmentation iterations over the training data to reach convergence for the test data.

Chapter 6

Generalization of similarity measure using Metric Learning

We will end this dissertation with a short introduction to an idea that has come to us during this thesis: to generalize the Potts Models ***similarity measure*** using any type of metric learning framework. Yet, it is truly possible, and I will dedicate some lines to explain this research possibility for the need to highlight an *open work* or an open *research problem* of interest.

> In general, Metric Learning refers to algorithms that learn an acceptable function of similarity in the feature space that ensures that large values (resp. small values) are preserved for inter-class separability (resp. intra-class separability) (Gao et al. (2014); Hoi et al. (2010); Lee et al. (2008); Hoi et al. (2006); Chang & Yeung (2007); Cao et al. (2020)).

We will discuss the role of the geometric invariance concept in image recognition, and review the classical and recent literature[1] on features invariance. Invariants help solve major problems of object recognition. For instance, because of the distinct point of view from which they were obtained, different images of the same object frequently differ from each other. Popular methods therefore need to find the correct point of view to fit the two images, a difficult problem that can require looking for all possible points of view and/or finding point correspondences in a high dimensional space (Adhan & Pintavirooj (2014); Tosranon et al. (2009); Ollivier et al. (2017); Sun et al. (2015); Harrison & Estabrook (1971); Tannenbaum (2006); Bottasso et al. (2002); Houtappel et al. (1965); Alghoniemy & Tewfik (2004); Mundy et al. (1992); Biess et al. (2011); Romano et al. (2018); Mundy (2006); Farah et al. (1994); Mei et al. (2018); Mundy et al. (1992); Sun & Cai (2009); Maweheb et al. (2016); Zisserman et al. (1995); Bonmassar & Schwartz (1994); Giles & Maxwell (1987); Gross & Latecki (1995)). Shape descriptors, calculated from the geometry of the shape, are geometric invariants that remain unchanged under geometric transformations, such as changing the point of view. Another significant class of geometric changes for which invariance is useful is object deformation (Flusser et al. (2009)). In the next section, we introduce the problem of occlusions, clutter and noise in image that affect recognition accuracy.

[1] The figures that appear in this review are extracted from the cited papers in each respective section.

6.1 The Problem of Occlusions, Clutter and Noise

Most invariants from an image are less effective when certain parts of it are inaccessible. This is true in presence of occlusions or clutters. The downside to most tests of similarity is that occlusions are not robust (Steger, 2002). Such occlusions or clutter are not identified by several neural networks. One possible explanation is that it is irrational to conclude that all possible occlusion patterns can be found during the training step (Qi et al. (2018); Kurenkov et al. (2020); Li et al. (2017); Bejjani et al. (2020)). Those occlusions are so numerous that it is not possible to detect all their variants during the neural network training. Therefore, this becomes a major problem in machine learning tasks, especially in computer vision, because we certainly can not presume that the data from the training and test are sampled from the same underlying distribution.

Many neural network architectures have been introduced to classify picture data information beneath those distortions (Kortylewski et al. (2020), Spoerer et al. (2017), Huang & Murphy (2015), Svirsky & Sharf (2020)), but their recognition precision is still not perfect. In addition, Volokitin et al. (2017) reveal that the training of cluttered picture models does not render models resilient to the configurations of clutter and flankers not used in training. They also find that it is prohibitively costly to train a model to be resilient to general clutter. In the next section, we present geometric invariance, and how it can help to overcome clutters and occlusions in MNIST digit data (Cohen et al. (2017); Yadav & Bottou (2019); Wu & Zhang (2010); Deng (2012)).

6.2 Geometric Invariance

A suitable choice of object invariants accomplishes high-efficiency detection of two-dimensional objects. The role of the principle of general invariance in object recognition has been discussed in the work of Weiss (1993). Invariants are designed to help overcome big object recognition issues (Zisserman et al. (1995); Cao et al. (2019); Cao (2003); Lou et al. (2008); Hong (1991); Alghoniemy & Tewfik (2004); Mundy et al. (1992)).

We need to retrieve numerous invariants, given a picture of a digit, that will serve as final descriptors. Given another picture of the same object, contrasting from the primary by, e.g., perspective or viewpoint, we need to retrieve the same descriptors. To do so, the influence of the transformations that gave rise to the variations between the images must be eliminated in some way. By using *sufficient* invariants, this task can be greatly facilitated. The word *sufficient* has to be taken as *sufficient statistics* (Ay et al. (2015)). Much works have been initiated in that sense (Doretto & Yao (2010); Masci et al. (2014); Rothganger et al. (2006); Arbter et al. (1990); Kazhdan et al. (2002); Kokkinos et al. (2012); Liao et al. (2013); Dubey et al. (2015); Kokkinos & Yuille (2008); Chen et al. (2010b); Ling & Jacobs (2005); Abdel-Hakim & Farag (2006)). When treating clustering of images like MNIST data, a fundamental question immediately arises: when do we decide with a *sufficient* invariant that two images come from the same digit, even though they are different?

6.3 Invariance descriptors and related works

There are many ways of removing changes between pictures. The easiest thing is to perform any possible transformation of the image to see if the other images fit any of its transformed versions. For instance,

in Ballard & Brown (1982), a template format and a given picture are assumed to vary as it were by translation, and the format is moved pixel by pixel over the picture until it finds a coordinate match. However, the search space becomes disproportionately broad when more complex transformations are involved, such as rotation, projection, etc. *Invariant* features can therefore be used to decrease the search space (Lowe, 1985). In the following, we present briefly some of those invariant features, and how they are computed.

6.3.1 Holographic Nearest Neighbor (HNN)

Holographic nearest neighbour (HNN) is an algorithm for object recognition with characteristics based on centroid moment of inertia and topological image characteristics. Under translation, rotation and scaling conditions, Torres-Mendez et al. (2000) used HNN to implement a new approach for object recognition that has been applied to character recognition. The invariant properties of the normalised moment of inertia are taken into account in the preprocessing phase (Ma et al. (2019); Liu et al. (2013); Lu & Xia (2007)). They also suggested a new encoding of an object that defines its topological features. The vectors obtained during the preprocessing stage were then used as inputs to the algorithm of the Holographic Nearest Neighbor (HNN) (Lu & Xia (2007)).

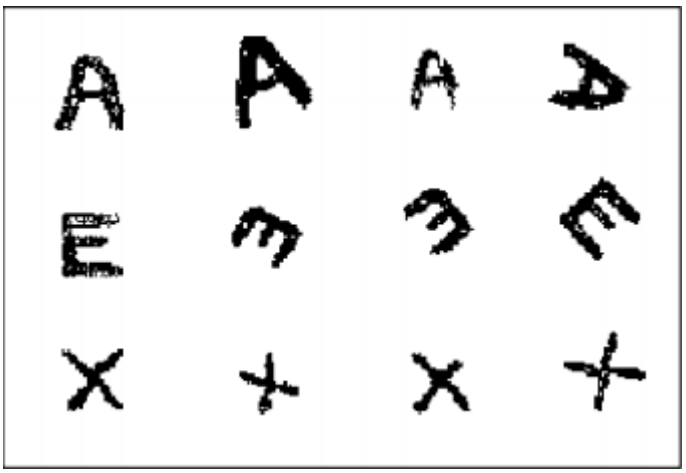


Figure 6.1: Letters A, E, and X are identified with 100% accuracy under some degrees of noise

Torres-Mendez et al. (2000) used 3 sets of invariant features:

1. Normalized central moment of inertia,
2. radial coding,
3. and differential radial coding;

Radial Coding and Computation Explained

The strategy for achieving the radial coding of a binary 2D object can be summed up as follows (Torres-Mendez et al., 2000):

1. **The centroid of the image is determined using its moment of inertia.**

In general, the moment of inertia quantifies the inertia of a rotating object considering its mass distribution. The moment of inertia is normally calculated by dividing the object into κ small pieces of mass $\eta_1, \eta_2, \ldots, \eta_\kappa$. Each piece is at a distance, $g_1, g_2, \ldots, g_\kappa$, from the axis of rotation. The moment $\boldsymbol{\iota}_m$ of inertia of the object is:

$$\boldsymbol{\iota}_m = \eta_1 g_1^2 + \eta_2 g_2^2 + \ldots + \eta_\kappa g_\kappa^2$$

A two-dimensional image is not a mass object, but can be represented by a continuous function $\mathrm{f(u, v)}$, where each image pixel can be considered as a mass particle equal to the pixel intensity value. The moment of inertia with respect to the image centroid (central moment of inertia) for binary images is:

$$\boldsymbol{\iota}_m = \sum_{i=1}^{\kappa} (\text{distance})_i^2 = \sum_{i=1}^{\kappa} \left((u_i - U_u)^2 + (v_i - V_v)^2 \right)$$

where U_u, V_v are the coordinates of the image centroid, u_i, v_i the image pixel coordinates of the object and κ the overall pixel count (its total number).

2. **Generate around the centroid some concentric circles equidistantly in a number of J.**

3. Count the number of changes in intensity for each circle: $(0$ to 1 or 1 to $0)$ that occur in the image, this is Θ_j.

4. For each circle, obtain the two largest ares that are not part of the object (we assume a known value for object and background). Measure each arc by counting the number of pixels, obtain the difference and divide by the size of the circle.

   Obtain the two largest circular segment (arcs) for each circle that are not part of the image. The number of pixels will represent each arc measure, then compute : $O_j = (o_1 - o_2) / o_\Phi$, where o_Φ is the length of the circle itself, o_1 is the length of the arc with the highest measure, and o_2 is the length of the second largest arc.

   The radial coding can be represented by the following vector:

   $$\Theta_1, \Theta_2, \ldots, \Theta_J, O_1, O_2, \ldots, O_J$$

   considering j circles. Θ_i is positive integer and O_i is a real value in $[0, 1]$.

Torres-Mendez et al. (2000) in their recognition stage uses the Holographic Nearest Neighbor (HNN) algorithm that is based on the principles of the Holographic Associative Memory (Sutherland (1992)). All these features are invariant to translation, rotation and scaling. Their recognition was obtained with almost 100% accuracy on images.

6.3.2 Shape Context

In the characterization of Belongie et al. (2002), an image can be viewed as a point set of infinite elements, for an object form is be a finite projection those points. A shape is defined then more technically by a distinct set of points sampled from the internal or external contours of the object. As detected by

an edge detector, these can be obtained as positions of edge pixels, giving a set $\mathcal{D} = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\}$, $\boldsymbol{x}_i \in \mathbb{R}^2$, of n points.

They find the *best* matching point $\boldsymbol{y}_j$ on the second shape for every point $\boldsymbol{x}_i$ on the first shape using *bipartite graph-matching* (Riesen et al., 2007) and *shape context*. Their methodology was productive and only 63 out of 10000 were wrongly classified for all of the MNIST test digits data using their process.

Figure 6.2: Examples of two digits that are handwritten. These two images are very different in terms of pixel-to-pixel comparisons, but the forms can appear to be identical to the human observer.

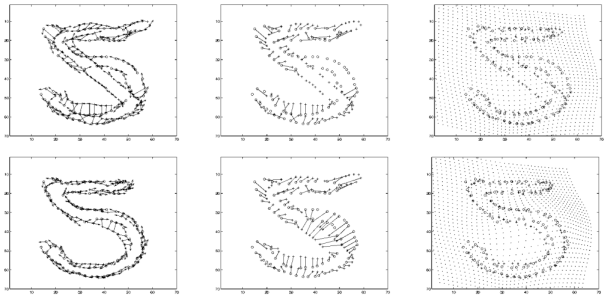


Figure 6.3: Illustration of the matching procedure applied to Figure 6.2 example.

6.3.3 Hough transformations features

When a large number of parameters are involved, hashing strategies such as the Hough conversion become necessary. It is a class of algorithms for the identification of edge-based objects, also called *Generalised Hough Transform* (GHT) (Ballard (1981); Ballard (1981); Samal & Edwards (1997); Illingworth & Kittler (1988)). Approaches of this sort have the benefit of being resistant to both occlusion and clutter.

Hough transform is an efficient method of detecting graphics specific targets; such that it can detect straight lines, circles, ellipses, parabolas and many other analytical graphs (Nair & Saunders Jr (1996), Yip (2000), Srihari & Govindaraju (1989), Pedersen (2007), Dahyot (2008), Goldenshluger et al. (2004)). The main downside of the transformation of Hough is the discretization of space, as well as the measurement process of the method that has some limitations, such as weak detection results due to high-intensity noise, a large amount of calculation, and even a large storage resources, that is in demand for the detection task, etc. (Illingworth & Kittler (1988), Mukhopadhyay & Chaudhuri (2015), Yuen et al. (1990), Hassanein et al. (2015)). Unfortunately, for the edge directions is a complicated and costly processing scheme for the generalised GHT, as it often needs extremely precise estimates to

determine if an object is present inside the image, and to determine its pose. For large models, this issue is particularly serious. Even in low noise images, the necessary precision is typically not achievable (Suetake et al. (2006)), since the discretization of the image contributes to errors in the edge direction that are already too large for the GHT (Steger, 2002).

A tool to deal with such cases has also been introduced by Kimura & Watanabe (2002): it extends the Generalized Hough Transform (GHT) under noise and occlusions to be an affine-invariant shape detector. In order to bring the direct computation for six parameters of an affine transformation, this process, called the affine-GHT, utilises pairwise parallel tangents and fundamental properties of an affine transformation. Experimental findings indicate that the proposed approach works quickly and effectively. More interesting results are presented in the work of Ecabert & Thiran (2004), Zhu et al. (2009), Hahmann et al. (2015), Tsai (1997), Ulrich et al. (2003).

6.3.4 Fourier descriptors

For the identification of two-dimensional related forms, Fourier descriptors (FDs) are shape-based characteristics. In object recognition and image processing, this method is used to represent a segment boundary shape in an image.

Suppose that the boundary of a given shape in 2-dimension contains Π pixels, and that the coordinate of the $\boldsymbol{s}$-th pixel on this boundary is $\boldsymbol{u}[\boldsymbol{s}]$ and $\boldsymbol{v}[\boldsymbol{s}]$. The Fourier Descriptor $(\mathcal{F}_d)$ of this shape boundary is defined as the Direct Fourier Transform $\mathcal{T}_{\mathcal{F}_d}[\boldsymbol{z}[\boldsymbol{s}]]$ of $\boldsymbol{z}[\boldsymbol{s}] = \boldsymbol{u}[\boldsymbol{s}] + j \cdot \boldsymbol{v}[\boldsymbol{s}]$ expressed as:

$$\mathcal{T}_{\mathcal{F}_d}[\boldsymbol{z}[\boldsymbol{s}]] = \frac{1}{\Pi} \sum_{\boldsymbol{s}}^{\Pi-1} \boldsymbol{z}[\boldsymbol{s}] e^{-\frac{j2\pi}{\Pi} sq} \quad (q = 0, \cdots, \Pi - 1)$$

In this Fourier transform, the first terms are much interesting and serve as the base ingredients of a given descriptor. For recognition tasks, this type of object descriptor is useful because of its most attractive properties, in the sense that it can be built to be independent of scaling, translation, or rotation. In Invariant Object Recognition, boundary-based analysis using Direct Fourier transformations has been suggested as an alternative (IOR) (Gonzalez (1987); Kauppinen et al. (1995)). For algorithms based on this type of processing, effectively called Fourier descriptors, invariance is obtained by normalising the image shape frequency representation.

6.4 Other Invariance researches

6.4.1 Chord distribution

The goal with chord distribution is to show an example of the construction of an algorithm for rapid identification also for highly complex objects (Mingqiang et al., 2011). Chord background analysis refers to finding the distribution of all chord lengths in a given form in various directions. Figure 6.4 shows an example of chords in a direction $\boldsymbol{\zeta} = \boldsymbol{\theta}$ vectorized by a set of lines $\mathfrak{C}(\boldsymbol{\rho}, \boldsymbol{\theta})$ defined by polar coordinates:

$$\boldsymbol{\rho} = \boldsymbol{x} \cos(\boldsymbol{\theta} - \pi/2) + \boldsymbol{y} \sin(\boldsymbol{\theta} - \pi/2), \boldsymbol{\theta} \in [0, \pi], \text{ and } \boldsymbol{\rho} \in (-\infty, \infty)$$

where $(\boldsymbol{x}, \boldsymbol{y})$ are pixels coordinates. For invariance, Chord algorithms are used to describe a comprehensive part of object recognition; precisely, they are used as an invariant of surfaces, representing a whole. They were used around the object as a boundary length invariant (Cao (2011); You & Jain (1984); Smith & Jain (1982); Marshall (1989)). The current example has been extracted from those existent researches on the topic.

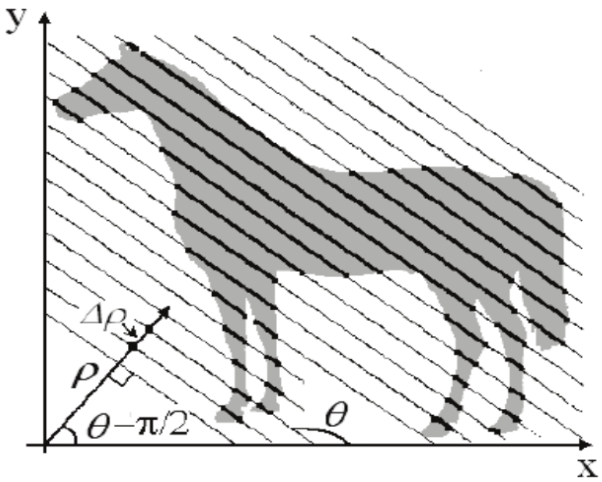


Figure 6.4: Representation of chords in direction $\boldsymbol{\theta}$ with the interval $\boldsymbol{\Delta}\rho$. The bold lines are the chords of the shape.

The interval, in order to catch the specifics of a shape, is $\boldsymbol{\Delta\rho}$ of $\boldsymbol{\rho}$, i.e. there should not be a great gap between two parallel chords. In practice, $\boldsymbol{\Delta}\rho = \boldsymbol{\mathcal{F}_{\max}}/(50 \sim 100)$, Where $\boldsymbol{\mathcal{F}_{\max}}$ is the length of a shape longest axis.

6.4.2 Moment invariants method

The research of moment invariants and their applications in image processing was first proposed in (Hu, 1962), where seven moment invariants in 2D were given. To detect symmetry for shape analysis and object recognition, directional moments (DMs) have been described in (Li & Li, 2017). They prove that by solving a trigonometric method derived from the DMs, identification of reflection symmetry can be achieved in a simple way, and if no reflection invariants are equal to 0, there is no symmetry between the shapes.

Khotanzad & Hong (1990) provided Zernike moments where rotation invariant elements are added. They are the magnitudes of a series of complex orthogonal moments. By first normalising the image with respect to these parameters using its regular geometric moments, scale and translation invariance are obtained. In their work, a systematic reconstruction-based approach is generated to determine the highest order of Zernike moments needed in a classification problem. In its relation to the original, the accuracy of the restored image is tested. More moments are used before the image reconstructed from them is fairly close to the original image. Using clean and noisy images from a 26-class character data set and a 4-class lake data set, the process is tested. Mainly, the supremacy of Zernike moment characteristics is experimentally confirmed over periodic moments and moment invariants.

Others researchers have proposed other measures such as inner-distance (Ling & Jacobs, 2007), bis-

pectrum invariants (Negrinho & Aguiar, 2013), Möbius invariants (Marsland et al., 2016), but those strategies will not be effective for some shape like letters or digits. Especially, illumination invariants (Alferez & Wang, 1999) will be unuseful here because MNIST are grey scale images.

6.5 Open Research Framework

As Torres-Mendez et al. (2000) obtained almost 100% accuracy on their set of images, and Belongie et al. (2002) method was defeated only 63 times out of 10000: is it that possible to combine multiple/more invariants methods to reach a 100% accuracy without any possible error? In other words, is that possible to build an engine that can auto-classify automatically any digit with a given *perfect* combination of invariance descriptors with no *error* allowed?

As a basic example (not to be counted among the *best* necessarily), we combine here the HNN from Torres-Mendez et al. (2000), and softmax activation function (Gold et al. (1996)) to introduce a modified *classifier* that we call the Holographic Nearest Neighbor Class of a digit.

Holographic Nearest Neighbor Class of a digit

Let $\mathcal{C}_i = \{c_0, ..., c_9\}$ represent the set of all possible classes for a given misclassified digit d_j.

As MNIST uses 60000 digits images for training, all categorized in 10 classes (6000 per class), we compute the Holographic Nearest Neighbor (HNN) of a class given a digit d_j by:

$$HNN(d_j, c_i) = \sum_{d_i \in c_i} HNN(d_i, d_j)$$

where d_i represent a digit in the training class c_i, and $HNN(d_i, d_j)$ using the features described in the work of Torres-Mendez et al. (2000). The computation above gives rise to an *estimate* of the probability for digit d_j to belong to the class c_i:

$$s\left(d_j, c_i\right) = \frac{e^{HNN(d_j, c_i)}}{\sum_{j=0}^{9} e^{HNN(d_j, c_i)}}$$

Thanks to the properties of the softmax function, the outputs $s\left(d_j, c_i\right)$ are always in the range [0,1] and add up to 1, hence, forming a probability distribution. Finally, let us introduce our newest measure:

Definition 6.5.1 (Holographic Nearest Neighbor Class of a digit - The HNNC)
Holographic Nearest Neighbor Class (HNNC) of a digit d_j, is computed as:

$$HNNC(d_j) = \arg\max_{c_i} s\left(d_j, c_i\right)$$

Beyond that scope, our Main (General) Open Questions from this literature review are :

- What are the metrics that should be learned for automatic object recognition in a given retrieval problem?

- What is the right combination of invariance descriptors to reach a 100% accuracy in a given recognition problem?
- Like fingerprints are unique and identifiable for everyone, is it that possible that each digit from a given source is unique and identifiable through a unique combination of invariance descriptors?

Chapter 7

Convolutional Neural Network Gibbs Model

In this chapter, we propose to augment the Shallow Gibbs Model (SGM) using Convolutional Neural Networks (CNNs) with image classification applications. This work is presented here as a *conceptual note*, because the model may get improved in many ways. We wish to keep it in the body of the thesis because of its contributing lines, as it proposes some directives to solve open questions in chapter [6]. We have introduced some novel architectures to explore [7.4.1] and have highlighted the importance of invariant networks [7.4.3] as universal approximators.

With a lot of applications in computer vision, image classification is a fundamental research subject. Due to its supervised feature, most image classification tasks perform training and testing using pre-labeled datasets. In real-world situations, however, images need to be grouped from more abstract definitions into labels. Typically, those labels definitions come from a particular domain, and certain rules describe them. We call this type of problem *domain-specific* classification of images. A common challenge in image classification is the lack of labelled images for training, especially for *domain-specific* images. Analogously to the Shallow Gibbs Model [4], we want to build here an ensemble learner by incorporating many Convolutional Neural Network (CNN) as base learners, in specific using some special filters, mainly the (similarity) invariants descriptor [see 6], and some reduced configuration: Quantized - Pruned - Binarized CNN. **In real life problems, assumption $(\mathcal{F}_d)$ in [8.0.6] may not be valid, or it may be difficult to explore all the Space domain to reach convergence**. When it is the case, as for *domain-specific* images (as e.g), it is better to explore other learning methods to find a perfect learning framework for those classification tasks.

7.0.1 Simple talks about Convolution

From **The Mind's Machine, Foundations of Brain and Behavior** (Watson & Breedlove (2012)), The transfer of signals from our eyes to our brain, involving numerous intermediate functions and convoluted pathways, is truly a complicated process. Suppose that each of our eyes represents a function [ι_1 for the left one, and ι_2 for the right one] of the image of reality. The following equation [7.1] with $v, u \in \Omega$ is called convolution between both eyes given an input v:

$$\boldsymbol{\iota_1 \star \iota_2(v) = \int_{\Omega} \iota_1(u)\iota_2(v-u)du} \tag{7.1}$$

where $\star$ is set for the convolution operation here. You may understand convolution $\iota_1 \star \iota_2(v)$ as the way both eyes join their effort to detect one image v. At the first sight of v, each eye $\iota \in \{\iota_1, \iota_2\}$ retina sends signals known as optic nerves through a series of neuronal axons. The key "connectors" between the retina and the brain act as these nerves. Each optic nerve begins with its corresponding eye and continues until a part of the brain called the thalamus is reached. Suppose in a simple configuration that eye ι_1 only perceives part u of image v, and ι_2 perceives part $v - u$. Whatever the amount of sight u perceived by ι_1, both eyes always *synchronize* and sum out what they see in $\iota_1(u)\iota_2(v-u)$, for all possible values of $u \in \Omega$. This leads to the integration [7.1], commonly called *convolution*.

Convolution in discrete perspective. For a moment, suppose that both eyes catch up a discrete view of the reality. The convolution is:

$$\iota_1 \star \iota_2(v) = \sum_{u=-\infty}^{\infty} \iota_1(u)\iota_2(v-u)$$

2D convolution. More than one dimension space can also be convoluted. For example, two-dimensional (2D) convolutions are applicable in the processing of images. If an image is a signal $\iota_1(u, v)$, where (u, v) is the position of a pixel, and $\iota_2(u, v)$ is the kernel of a filter, the convolution is:

$$(\iota_1 * \iota_2)(u, v) = \int_u \int_v \iota_1(\alpha, \beta)\iota_2(u - \alpha, v - \beta)\partial\beta\partial\alpha$$

For example, with a Gaussian Kernel, this operation is smoothing, since we are integrating a small region of the image with a 'normal' filter at each (u, v) location. [Chang & Sha (2016), Cheng & Parhi (2020)].

7.1 Convolutional neural networks (CNNs)

Graphical Representation

Convolutional neural networks (CNNs) is a special architecture of artificial neural networks, proposed by LeCun et al. (1995). LeCun called this model **LeNet5** [Leray (2000), LeCun et al. (2015b)]. CNNs use some features of the visual cortex. Image classification is one of the most common implementations of this architecture. CNNs have a layer of data for the input phase, a layer of output, and a hidden layers. The secret layers typically consist of convolutional layers, layers of ReLU, layers of pooling, and layers that are completely connected.

1. A convolution process is introduced from the input layer by convolutional layers. The basic information from the previous layer is passed on to the next layer. Neurons only receive feedback from a subarea of the previous layer in a convolutional layer. [Zeiler & Fergus (2014), Kuo (2016), LeCun et al. (1995), O'Shea & Nash (2015), Bouvrie (2006), Albawi et al. (2017)]

2. Pooling unites the outputs of neuron clusters in the next layer into a single neuron.

3. Every neuron in one layer is connected to every neuron in the next layer by completely linked layers. Each neuron receives input from every part of the previous layer in a completely connected layer (fully connected layer).

Some experiments have been done in the past proving CNNs really effective in the field of image classification tasks (Sultana et al. (2018), Sermanet et al. (2012), Simonyan & Zisserman (2014)).

A more formalized definition for CNNs

Here follows a short definition of Convolutional Neural Network [Mairal et al. (2014), Wu (2017), Albawi et al. (2017)]:

In general, a convolutional layer consists of a set of F filter weights $\mathbf{H}^f \in \mathbb{R}^{z \times w}$, $f = 1, \ldots, F$, F is called the depth, which generate each a so-called feature map $\mathbf{U}^f \in \mathbb{R}^{n' \times m'}$ from an input matrix $\mathbf{V} \in \mathbb{R}^{n \times m}$ according to the following convolution:

$$\mathbf{U}^f_{i,j} = \sum_{k=0}^{z-1} \sum_{\ell=0}^{w-1} \mathbf{H}^f_{z-k,w-\ell} \mathbf{V}_{1+s(i-1)-k,1+s(j-1)-\ell}$$

where $s \geq 1$ is an integer parameter called stride, $n' = 1 + \left\lfloor \frac{n+z-2}{s} \right\rfloor$ and $m' = 1 + \left\lfloor \frac{m+w-2}{s} \right\rfloor$, and it is assumed that $\mathbf{V}$ is padded with zeros, i.e., $\mathbf{V}_{i,j} = 0$ for all $i \notin [1, n]$ and $j \notin [1, m]$. It is possible to decrease the performance measurements by either increasing the s stride or by adding a pooling layer. The pooling layer partitions $\mathbf{U}$ into regions $p \times p$ for each of which a single output value is computed, e.g. a maximum or average value, or $L2$ -norm.

7.2 Pruned and Quantized CNNs for sparsity and model compression

It takes a huge dataset to use CNN for classification, which improves accuracy. A downside, however, is that it produces several model parameters that increase the cost of computation and require high memory bandwidth. We optimize the model in two ways. The first is network optimization to narrow down the network size by pruning redundant links and quantifying the weights and fusing the neural networks.

7.2.1 Pruning a Convolutionnal Neural Network: computation speed and model size reduction

This is a model compression type. It reduces the number of synaptic ties to other neurons in order to reduce the total amount of data. Weights similar to zero are usually omitted. This will help remove redundant connection ties for tasks such as classification with small accuracy drops (Shawahna et al., 2018). We will focus here on pruning entire filters in convolutional layers.

This has a cool side affect of also reducing memory. As observed in Molchanov et al. (2016) paper, the deeper the layer, the more it will get pruned. This implies a lot of pruning of the last convolutionary layer, and a lot of neurons from the fully connected layer after it will also be discarded. Another

alternative would be to reduce the weights in each filter or delete a particular dimension of a single kernel while pruning the convolution filters. You can end up with sparse filters, but having a computational rate is not trivial. Latest works support "Structured sparsity" instead of pruning whole filters [Ding et al. (2018), You et al. (2019), Li et al. (2016), Luo et al. (2017), He et al. (2020), Huang et al. (2018), He et al. (2018)]. One important thing some of these papers demonstrate is that they get results that are much better than training a smaller network from scratch by training and then pruning a larger network, particularly in the case of transfer learning.

Li et al. (2016) supports the pruning of full convolution filters. Pruning a filter with an index of k affects the layer in which it resides, and the layer below. All input channels in the k index of the following layer will have to be deleted, as they will no longer exist after pruning. If the following layer is a fully linked layer, and the size of that channel's feature map is $M \times N$, then $M \times N$ neurons are excluded from the fully linked layer. In this work, the neuron ranking is fairly basic. This is the L1 norm for each filter's weight. They rank all the filters at each pruning iteration, prune the lowest ranking filters globally across all the layers, retrain and repeat.

A similar work has been proposed by Anwar et al. (2017), but the ranking is much more nuanced. They retain a set of N particle filters that reflect the pruning of N convolutional filters. A score dependent on the network precision on a validation set is allocated to each particle when the filter represented by the particle has not been masked out. Then, new pruning masks are sampled based on the new ranking. They used a small validation collection to calculate the particle scores because running this method is heavy.

7.2.2 Other Proof-based and advanced Pruning methods

1. Adversarial Neural Pruning (Madaan et al. (2019))
2. Network Pruning via Transformable Architecture Search (Dong & Yang (2019))
3. Self-Adaptive Network Pruning (Chen et al. (2019))
4. Structured Pruning of Large Language Models (Wang et al. (2019))

7.3 Quantized Convolutional Neural Networks

To get the neural network to an acceptable size while still maintaining high-performance precision, the quantization method is often applied. For edge applications, where the memory size and number of computations are necessarily limited, this is particularly important. In such applications, the model parameters are held in the local memory to prevent time-consuming transfers using PCIe ((Peripheral Component Interconnect Express) or other interconnection interfaces in order to achieve better performance [Gschwend (2020), Wang et al. (2020), Morcel et al. (2019), Ding et al. (2019a), Ding et al. (2019b), Blott et al. (2018), Shu et al. (2019)].

The method of approximating a neural network using floating-point numbers (FTP32) through a neural network of low-bit width numbers (INT8) is carried out for the reduction. This significantly decreases both the need for memory and the cost of computing using neural networks. We slightly lose accuracy and precision by quantizing the model. However, for most applications there is no need for a 32-bit

floating point. Research into the use of lower-precision numerical formats has been motivated by the need for reduced bandwidth and machine specifications of deep learning models [Fromm et al. (2018), Zhuang et al. (2019), Cai et al. (2018), Zhuang et al. (2018)]. It has been extensively demonstrated that 8-bit integers (or INT8) can be used to represent weights and activations without incurring substantial loss of accuracy. An active area of research that has also shown tremendous success is the use of even lower bit widths, such as 4 or 2, or even 1-bit (Zhou et al., 2016).

If binary $(-1, 1)$ or ternary $(-1, 0, 1)$ are weighted using 2-bits, then convolution and fully-connected layers can only be determined by adding and subtracting, completely removing multiplications. If activations are also binary, it is possible to delete additions in favor of bitwise operations (Rastegari et al., 2016). High performance hardware is typically indispensable for the implementation of CNN models because of the high computational complexity, which prevents their further extensions. Banner et al. (2019) present the first realistic approach to quantization of 4-bit post-training: it does not include training the quantized model (fine-tuning), nor does it require the entire dataset to be usable. They also accomplished the quantization of all activations and Weights and three complementary methods have been suggested to reduce the quantization error of the tensor stage, two of which achieve a closed-form analytical solution.

7.3.1 Binarized Neural Networks (BNNs)

In this part, we review Binarized Neural Networks (BNNs).

> Binarisation is the most extreme form of network quantization. Binarization is a 1-bit quantization where only two possible values are possible for data. In general, for those two values, -1 and +1 were used. In some cases presented in the literature, quantized networks using the values -1 and 0 and +1 are not binary but ternary. [Simons & Lee (2019), Seo et al. (2016), Prost-Boucle et al. (2018)].

The BNNs deep neural models are networks which, instead of full accuracy values, use binary values for activations and weights. With binary values, BNNs can use bitwise operations to perform computations, which decreases execution time. The model sizes of BNNs are much smaller than their complete counterparts in precision. While the accuracy of a BNN model is typically less than full precision models, on larger datasets such as ImageNet (Simons & Lee, 2019), BNNs have closed the accuracy gap and are becoming more accurate models. As a result of their bitwise performance, BNNs are also good candidates for deep learning implementations on Field Programmable Gate Arrays (FPGAs) and ASICs. Those machine are very famous due to their bitcoin and cryptocurrencies mining power (Tu et al., 2019).

A method has been implemented to train Binarized Neural Networks (BNNs) with binary weights and runtime activations (Courbariaux et al., 2016). The binary weights and activations are used to measure the parameter gradients at train-time. BNNs greatly decrease memory size and are accessed during the forward transfer, as they replace most arithmetic operations with bit-wise operations, which are supposed to dramatically increase power performance. They performed two sets of experiments on the Torch7 and Theano systems to verify the efficacy of BNNs. On both, BNNs over the MNIST, CIFAR-10 and SVHN datasets achieved almost state-of-the-art results.

BNNs provide simple, compact descriptions and, thus, they have a wide variety of low-power com-

puter applications. Icarte et al. (2019) got a model-based approach to train BNNs using constraints. Programming (CP), mixed-integer (MIP) and CP/MIP programming, and hybrids approaches were all investigated. They formulate the issue of training as seeking a set of weights which correctly classify the instances of the training set while optimizing objective functions proposed in the literature as proxies in a quest of generalization. Their experimental findings on the MNIST digit recognition dataset indicate that BNNs based on a hybrid approach generalize better than those obtained from state-of-the-art models when training data is minimal.

7.4 Our CNN architecture

Let us assume a CNN architecture with Ξ convolutional layers, denoted as network ($\mathcal{C}_{NN}$). Assume $\mathcal{L}_i$ to be the i^{th} layer and $i \in [1, 2, \ldots \Xi]$. The layer $\mathcal{L}_i$ has n_i filters which gives the n_i feature-maps that are used as input for the next layer. The set of filters at layer $\mathcal{L}_i$ is denoted as $\mathcal{F}_{\mathcal{L}_i}$ where $\mathcal{F}_{\mathcal{L}_i} = [_{1,2}, \ldots, _{n_i}]$. Similarly, the feature maps at layer $\mathcal{L}_i$ are represented as $\mathcal{M}_{\mathcal{L}_i} = [m_1, m_2, \ldots, m_{n_i}]$.

Each feature map m_i is of dimension (e_ξ, d_ξ), where e_ξ, d_ξ are height and width, respectively, of the feature map. Therefore the shape of $\mathcal{M}_{\mathcal{L}_i}$ is (e_ξ, d_ξ, n_i).

7.4.1 Our 4-type CNN Gibbs Model

Our main CNN network has feature maps $\mathcal{M}_{\mathcal{L}_i}$ on each layer $\mathcal{L}_i$ that we replace by weighted feature maps, and the weights are given by three (3) variants weights type for the network ($\mathcal{C}_{NN}$) (explained in the next section). Let $\mathbf{W}_{\mathcal{L}_i} = [\boldsymbol{W}_1, \boldsymbol{W}_2, \ldots, \boldsymbol{W}_{n_i}]$ be the weights of layer $\mathcal{L}_i$ given by the $\mathcal{C}_{NN}$ network. Then, the $\mathcal{L}_i^{th}$ layer feature maps of $\mathcal{C}_{NN}$ is replaced as:

$$\mathcal{M}_w = [\boldsymbol{W}_1 m_1, \boldsymbol{W}_2 m_2, \ldots, \boldsymbol{W}_{n_i} m_{n_i}]$$

Here $m_1, m_2, \ldots, m_{n_i}$ are the feature maps at layer $\mathcal{L}_i$ in the original network. Now, our objective is to optimize the network such that most of the $\boldsymbol{W}_i$ are close to zero, without sacrificing the accuracy. The modified network is obtained using the following approaches :

- The Pruned CNN: $m_i \in \{0, 1\}$ with higher probability on 0. The probability is drawn with a beta law with hierarchical Bayesian approach.
- The Mean and Contrast CNN: $m_i \in \{0, 1\}$ with lower probability on 0. The probability is still drawn with a beta law with hierarchical Bayesian approach.
- Pruned CNN with Ising Models as illustrated in section [3.8], this approach is a very rich land of statistical exploration.
- The Quantized CNN: $m_i \in \{0, 1\}$ with uniform probability. In this architecture, each scalar v_i of $\boldsymbol{W}_i$ is transformed into the most closed number in $\pm 2^s$ format, for bitwidth standard computations. This is much more closed to the IEEE-754 standard presented in (Al-Hami et al., 2018).

We plan to test the effectiveness of the CNN Gibbs Model with CIFAR dataset described below.

7.4.2 The CIFAR-10 Photo classification dataset for experiments

The CIFAR-10 dataset is used for our experiments. It consists of color images in 10 classes of 60000 of 32×32 pixels, with 6000 images per class. The dataset is divided into five training batches and one test batch, each with 10000 photos. There are 50000 training images and 10000 test images. There are precisely 1000 randomly-selected photos from each class in the test batch. The training batches comprise the remaining images in random order, but more images from one class than another might be present in certain training batches. The training batches between them contain precisely 5000 images from each class. Krizhevsky et al. (2009) explains more the dataset and the methods used in much greater detail when collecting it.

7.4.3 Two additional layers and Invariants Networks as (universal) approximators

As we are working to make the CNN Gibbs Model a very powerful model, one of our main contribution in this chapter is to realize that **neural networks can be built as invariant networks for universal approximation purpose**.

First, we will add after the input layer, a matching layer using bipartite graph-matching as presented in shape context section 6.3.2. A second layer after the matching layer will extract different invariant features coupled with the existent ones in the image: such as: Holographic Nearest Neighbor (HNN) features [see 6.3.1], Hough transformation features [6.3.3], Fourier descriptors [see 6.3.4], Chord distribution features [see 6.4.1], Moment invariant features [6.4.2].

Finally, we will integrate results from [Maron et al. (2018), Yarotsky (2018), Azizian & Lelarge (2020), Dym & Maron (2020), Keriven & Peyré (2019), Maehara & NT (2019), Maron et al. (2019)] which highlight some invariant networks as universal approximators. As for example, Maron et al. (2019) exposes :

- G-Invariant networks;
- (hyper-) graph networks;

that are all universal. To the best of our knowledge, few of those networks have truly been implemented in practice.

Chapter 8

Concluding Remarks, Discussion notes & Applications

8.0.1 On the mixture models.

In the statistical literature, the significant role of finite mixture models is expanded by the ever-increasing pace at which publications on their implementations appear. Many more combinations of such (mixture) versions are also available. They are fundamentally different from the latent distribution type (discrete or continuous) or family (exponential family most of the time), to their estimation method (Bayesian posterior likelihood or maximum likelihood). The finite mixture models are easy to implement because of various criterion that help to choose the right number of components: Akaike's Information Criterion (AIC), Bayesian information criterion (BIC), informational and entropy criterion, etc. (Steele & Raftery (2010); Bozdogan (1993); Hawkins et al. (2001); Biernacki et al. (1999); Naik et al. (2007); McLachlan & Rathnayake (2014); Celeux & Soromenho (1996); Lo et al. (2001)).

Algorithms that work for that purpose (Expectation Maximization –the EM particularly) rely on the fact that, there exists a strongly consistent asymptotically efficient local maximizer in the interior of the parameter space (Zhao et al. (2020); Archambeau et al. (2003); Yu et al. (2018); Jordan & Xu (1995); Ma et al. (2000); Clark & Vo (2007); Xu & Jordan (1996)). Therefore, it is recommended to search the best local maximum in the unconstrained parameter space and then to check that the obtained solution indeed corresponds to a local maximum and is not on its way to infinity. This check can be difficult due to the presence of so-called spurious local maxima which should be ignored.

A natural Bayesian approach for mixture models with an unknown number of components is to take the usual finite mixture model with symmetric Dirichlet weights, and put a prior on the number of components—that is also called sometimes: *to use a mixture of finite mixtures (MFM)* (Miller & Harrison (2012); Miller & Harrison (2018); Geng & Hu (2020); Geng (2017)).

8.0.2 Extension to a Mixture of music composers.

Our interest into music goes way back to time series analysis (ALAHASSA (2018)). Like in time series, music is essentially a subject of interest because of its special *harmonics*. Music automated generation is one excellent study for mixture models that we would like to explore in the future for this

reason. The motive is to use the potential of machine learning architectures and mixture models to learn musical types from random musical corpora automatically and then to create samples from the predicted distribution.

A question of interest: Is-it that possible that each musical type is a combination of specific mixture of waves distributions? Are those mixtures all approximable by a reduced neural network kernel, being Shallow or Deep ?

The applications are vast enough and can reach the domain of piano music generation, and automatic realistic video contents creations as well. In practice, this is done by applying a mixture of composers (Balkema & van der Heijden (2010); Engels et al. (2015); Boulanger-Lewandowski et al. (2012); Mao et al. (2018); Williams et al. (2019); Dieleman et al. (2018); Moerchen et al. (2006); Suprem & Ruprem (2013); Ramalingam & Krishnan (2006); Williams et al. (2020); Coviello et al. (2012); Li et al. (2019a)). This may require the utilization of hidden mixture (Markov) models, as much as much more capitalization of deep neural networks and *alike* models (Deng et al. (2013); Docevski et al. (2018); Židek (2020); Simões (2019)). One expectation of this challenge is to produce by itself sounding music with original compositions that still respects the rules and classic patterns. It will also help music creators to automatically correct/adjust their composition.

8.0.3 The Multilayer feedforward Neural Network.

Deep learning neural networks are trained using the stochastic gradient descent optimization algorithm (Li & Orabona (2019)). The learning rate is a hyperparameter that controls how much to change the model in response to the estimated error each time the model weights are updated (Gotmare et al. (2018)). Choosing the learning rate is challenging as a value too small may result in a long training process that could get stuck, whereas a value too large may result in learning a sub-optimal set of weights too fast or an unstable training process that fails to converge (Darken et al. (1992); Magoulas et al. (1999); Behera et al. (2006); Jacobs (1988)). For our experiments in section, the learning rate was small, large learning rates result in unstable training and tiny rates result in a failure to train sometimes (Wu et al. (2019)). A good batch size is also a matter of importance (Smith et al. (2017)). Those rules have become more important in a variational inference scheme (Ranganath et al. (2013)).

With the recent advanced in neural networks, the choice of the loss function is simplified. The cross-entropy is mostly used for classification tasks where the output is a probability value between 0 and 1 (Zhang et al. (2018c)). One may investigate other entropy criteria such as the Shannon entropy (Silva et al. (2005)). The Mean Squared Error is frequently used for regression tasks, but depending on the choice has to be validated (Rady (2011)) because of its potential impact on the final results. This is valid for all neural networks classifiers (Falas & Stafylopatis (1999)). Activation functions are also a subject of matter. You can choose your activation functions experimentally (Karlik & Olgac (2011); Ramachandran et al. (2017); Sharma (2017); Zhang et al. (2018b); Nwankpa et al. (2018); Hayou et al. (2019)). This is our main recommendation which is also valuable for all initialization (Hayou et al. (2018)).

Our results from table 3.4 prove that we can reach any sufficiently good level of prediction by adapting an N_k H-Layer-Neural Network model. This architecture is *universal* and represents one *state-of-the-art* model in **multivariate regression**. The experimental results are impressive because we can now (and

only) step to how to find the good model in statistical learning only by simply augmenting/reducing the chosen number of layers, the number of training times (epochs), the size of the batch learning process, the number of neurons on each layer, etc., to reach any desired precision level. This can be generalized to classification task using the softmax activation function (Memisevic et al. (2010); Duan et al. (2003); Liang et al. (2017)).

8.0.4 Notes on the Potts Models with Complete Shrinkage.

The Potts model is frequently used as random partitions models where covariates may be included [See 2.2]. Those models with covariates are called Product partition models with a covariate-dependent extension (PPMx) mainly introduced by Müller et al. (2011a). The Shallow Gibbs Model is based on a PPMx from Murua & Quintana (2017b) which has the form:

$$\begin{gathered} y_1, \ldots, y_n \mid \rho_n, \boldsymbol{\psi}_1^*, \ldots, \boldsymbol{\psi}_{k_n}^* \overset{\text{ind}}{\sim} p\left(\boldsymbol{y}_i \mid \boldsymbol{x}_i, \boldsymbol{\psi}_{s_i}^*\right) \\ \boldsymbol{\psi}_1^*, \ldots, \boldsymbol{\psi}_{k_n}^* \overset{\text{iid}}{\sim} p(\boldsymbol{\psi}) \text{ and } \rho_n \sim p\left(\rho_n \mid \boldsymbol{x}^n\right) \end{gathered}$$

where $p\left(y_i \mid x_i, \psi_i\right)$ is the likelihood model stating the relationship between the i th response and the associated covariate vector x_i. The Potts clusters in this last model are simulated using the Swendsen-Wang algorithm (Wang & Swendsen (1990), Barbu & Zhu (2003), Galanis et al. (2019)). One may be confronted to the cluster size distribution during simulation, because the *bonds* based version of the Potts Model [see 2.1.1] has the drawback to simulate small clusters that are not preferable in some circumstances, as for example to run or apply some neural networks models locally on the clusters themselves (using cluster adaptive training scheme as a specific example – Gales (2001)). One theoretical approach is to search for the distribution of the components size in the Potts Model to simulate a conditional distribution: a bonds distribution with constraints or restrictions on the clusters minimum size. This can be done using the frequency of frequencies distribution [see 2.6.1], specifically the count vector $\mathcal{M} = (\{m_v\})_v$ of the clusters also called the frequency of frequencies (FoF) vector, the distribution of which is also commonly referred to as the FoF distribution in short appellation. Once you find the FoF distribution of clusters count vector given the bonds probabilities, expressed as :

$$FoF(\{m_v\}|\{\delta_{ij}\}; E(G))$$

with the graph G and its edges set $E(G)$ ($\delta_{ij} = 1$ if observations i and j are connected), you may derive the conditional distribution of bonds given the constraint cluster size condition $\mathcal{S}_c = \{m_v = 0, \text{ for } v \leq S_c\}$, where S_c is the minimum cluster size we expect. This requires that we explore all configurations of $\mathcal{S}_c$ [see 2.6.4], for which we have proposed a fast algorithm. This may almost be intractable for our usual computers (only for for large datasets), even to find the most probable configuration among S_c-list given a label assignment. So, we have just limited this research to simply present this conditional bonds distribution we were looking for in [see 2.6.6]:

$$p(\{b_{ij}\}, \{b_{ij}\} \in \tilde{S}_c - \text{list}|\sigma, X, \beta, q, S_c) \quad = \quad \frac{PROD_{i\in\{1,\ldots,n-1\}, j>i}\left([\tilde{P} \circ \tilde{\delta}]^B + [(1-\tilde{P}) \circ \tilde{\delta}]^{1-B}\right)}{p(\{m_v = 0, \text{ for } v \leq S_c\}|\sigma, X, \beta, q)} \tag{8.1}$$

where we are given a q-state, a bandwidth σ, X is set for our covariates, and β the inverse of the system temperature, and the other parameters are reformulated quantities presented in section 2.6. Another practical approach is to modify slightly the Swendsen-Wang algorithm to insert some clusters constraints. This what we have achieved as a novel Potts Model, called the Potts Clustering with Complete Shrinkage (PCCS) [see 2.3]. In this approach, to deal with the increasing number of small clusters generated in a given partition, we apply a modified agglomerative clustering approach (Kurita, 1991) by merging all small clusters of size $\leq \mathbf{h}$ with their closest cluster in terms of minimal distance respectively, where $\mathbf{h}$ is an integer greater or equal to 2. The algorithm uses a technique in which distances of all pairs of observations are stored. Then the nearest cluster (with size $\geq \mathbf{h}$) is given by the cluster with the closest node in terms of minimal distance to the cluster to be merged. This approach is truly effective; it helps to control the clusters size, and we have found evidence of bell shape curve [figure 2.8, figure 2.20] of the constrained cluster size distribution of PCCS, when applied to some datasets taken from the multiple-output benchmark datasets available in the Mulan project website (Tsoumakas et al., 2020) [see Table 2.1].

8.0.5 Concluding remarks on the Shallow Gibbs Structure.

We present in chapter 4 a framework to build a new structured *Potts clustered Gibbs Multivariate Regression model* which is properly say is Random Gibbs Neural Network Forest (Barber & Bishop (1997), Barber & Bishop (1998)), by combining structured precision matrixes, Potts Neural Network Regression (SPNNR), variational learning and backpropagation. You may call the model under many appellations: Shallow Gibbs Network, Random Gibbs Network Forest, Shallow Potts Neural Network, the Potts-Gibbs-NM model etc. The model has four variants: the fully-connected Gibbs network or full Gibbs network (Full-Gibbs, for short), the between-layer sparse Gibbs network (B-Sparse-Gibbs), the sparse shallow Gibbs network (Sparse-Gibbs), the sparse compound symmetry Gibbs network (CS-Sparse-Gibbs), and the compound symmetry Gibbs network (CS-Gibbs). As literature references to many alike sparse-structured or similar models, you may look for the work from: Ionescu et al. (2015), Kepner & Robinett (2019), Wainwright (2014), Louizos & Welling (2016), Paul et al. (2016), Schwing & Urtasun (2015), Ardakani et al. (2016), Mazza-Anthony (2019), with an exception of the Compound symmetry Gibbs network, which is a particular *innovation* of our research. We can even say that one main innovation of this research is the introduction of *compound symmetry* precision matrix on the weights on our Bayesian Shallow neural networks. The effectiveness of our results will definitely increase interests in those types of matrices.

The neural network (our base learner) weights are Gaussian Markov Random Field distributed. To offer flexibility in terms of model structure, the precision matrix has been modified to infer three variants of analysis : sparse analysis, compound symmetry analysis and fully-connected framework. Those models have the properties to adapt themselves easily to the data in a few iterations during the learning process. The model is mounted in three stages: a set of data $\boldsymbol{y}^n = \{y_1, \ldots, y_n\} \subset \mathbb{R}^p$ with associate covariables ; each $\boldsymbol{y}_i$ following a Gaussian distribution $p(y|x, \psi, \Sigma) = (2\pi)^{-p/2}|\Sigma|^{-1/2}\exp\{-1/2(y - f_\psi(x))'\Sigma^{-1}(y - f_\psi(x))\}$., with covariance Σ and mean $f_\psi(x)$ which represent specifically a neural network function of the covariable $\boldsymbol{x}$. The weights and the biases of the model vectorized as $\psi = (b^{(1)}, W^{(1)}, b^{(2)}, W^{(2)})$ are high dimensional and their size can be increased at will. But, keep in mind that the goal of the Shallow Gibbs Network is to build a simplified model with few neurons and better mean *squared* error in a *very* short training time (reduced number of epoch ≤ 10).

First, we define the Kullback-Leibler of the model variational distribution [See 4.4] that will help to simulate variational parameters. This approach is very rich in terms of properties and convergence (da Rocha et al. (2011), Wiegerinck & Kappen (2000), Challis & Barber (2013)); even though it suffers from many drawbacks as the positive definiteness of your precision/covariances matrices during gradients updates [see 4.4.2]. Gradients updates have good advantages such as a fact approximations in a few iterative steps, mainly if you tweak appropriately the learning rate. When you applied an approximation for your parameters or hyper-parameters during the updates, you step forward the ***Iterative Projected Gradient*** (IPG) updates approach (Cruz et al. (2011)) which converge also if the projection space is an optimal subspace. As the case of PSD - Positive Semi Definite - approximation is current in literature, we can index this paper ***"Approximation by matrices positive semidefinite on a subspace"*** from Hayden & Wells (1988) that was a great help in building our model.

It is no need to say that the *double* backpropagation scheme in [see 4.5.1] is truly a requirement to reach an oustanding result. The secret behind is that the second backpropagation system represents itself another *neural network* by its own. In other words, given a set of initial predictions, for the data $(\boldsymbol{x}, \boldsymbol{y})$ and an estimated/or random values for $\psi = (b^{(1)}, W^{(1)}, b^{(2)}, W^{(2)})$ and Σ, it reduces naturally the model error by backpropagating under a suitable learning rate schedule. Our experiments and results were not only efficient in comparison to the MMR model Zhen et al. (2017), but also was impressive against Murua & Quintana (2017c) model, which was also a benchmark during all our supervision time with him as ***Professor and Mentor***.

The four (5) models developed in this framework include a clustered neural network regression via a Potts model. Monte Carlo simulations are limited to run posterior simulation for these models, and variational inference was the last resort. This novel representation (framework) aims to capture also more complex pattern in small datasets with high dimensional features. We have increased relative prediction power compare to simple prediction, and offered to the novice an adapted regression model to heterogeneous data, that can surpass the N_k H-layer Feed Forward Neural Network (FFNN) [see 3.10] and shall over-*become* the next revolution of neural networks. And finally, we have found for the Shallow Gibbs Network model [4.1], the perfect learning configuration is: the *dist*-NN-(h)-TS-PMI)-$(l_1, \boldsymbol{\zeta}, \epsilon_{dbs})$ − **DBS**, which is a combination of the *Universal Approximation Theorem*, and the DBS optimization, all coupled with the (*dist*)-Nearest Neighbor-(h)-Taylor Series-Perfect Multivariate Interpolation (*dist*-NN-(h)-TS-PMI). It indicates that, with an optimal number l_1 of neurons on the hidden layer, an adapted $\boldsymbol{\zeta}$ of DBS updates, an optimal DBS learning rate vector ϵ_{dbs}, an optimal distance *dist* in the research of the nearest neighbor in the training dataset for each test data x_i^{test}, an optimal order h of the Taylor approximation for the Perfect Multivariate Interpolation (*dist*-NN-(h)-TS-PMI) model once the **DBS** has overfitted the training dataset, the train and the test error converge to zero (0). ϵ_{dbs} integrates simultaneously the DBS learning rate vector for all the model parameters, the DBS learning rate for the training data, and the DBS learning rate for the test data.

This finding is a great revolution in all fields and subfields of *Statistical Learning Theory*, from the now to the forever. The generalization power of this model [under assumption $(\mathcal{F}_d)$] is infinite, as the secret is to understand that every augmented data can also be sufficiently differentiated, as much as we want to overlap all the possible data Space domain:

$$x_{j_1}^{\text{aug-train}} = (x_{j_1}^{\text{train}}(1) + \delta, ..., x_{j_1}^{\text{train}}(q)) \longrightarrow \begin{cases} (x_{j_1}^{\text{train}}(1) + s_1 \cdot \delta, ..., x_{j_1}^{\text{train}}(q)) \\ (x_{j_1}^{\text{train}}(1), x_{j_1}^{\text{train}}(2) + s_2 \cdot \delta, ..., x_{j_1}^{\text{train}}(q)) \\ ..... \\ (x_{j_1}^{\text{train}}(1), x_{j_1}^{\text{train}}(2), ..., x_{j_1}^{\text{train}}(q) + s_q \cdot \delta) \end{cases} \tag{8.2}$$

with $(s_1, s_2, ..., s_q)$ is a q-tuple of integers or rationals, where $x_{j_1}^{\text{aug-train}}$ is our augmented data in the training set, and $\delta \in \mathbb{R}$ a very small number. The re-training process has to be done progressively to obtain their perfect associated parameters [see 5.4].

8.0.6 A Generalized Double Back-Propagation Scheme (GDBS) for any parametric model

We propose an effective Generalized Double Back-Propagation for any parametric model, augmented with a differential and local neighborhood machine learning framework for almost sure convergence. As an extension of section [4.5.1], we propose a general double back-propagation scheme (GDBS) using the Mean Squared Error (MSE), and for any (parametric) model with parameter ψ as :

$$\hat{y}_{est,i} = f_{\psi}(x_i) \tag{8.3}$$

1. Using the equation (in 8.3), apply any suitable optimization framework to obtain a general estimate $\hat{\boldsymbol{\psi}}_0$ of ψ for all the model (in a first step).

2. Use again equation 8.3 to backpropagate the $MSE(y_i - \hat{y}_i) = \frac{1}{n}\|y_i - \hat{y}_{est,i}\|^2$ to update $\hat{\boldsymbol{\psi}}_0$ per observation as follows:

$$\hat{\boldsymbol{\psi}}_{1,i} \longleftarrow \hat{\boldsymbol{\psi}}_0 - \epsilon_{\psi,0} \frac{\partial MSE(y_i - \hat{y}_{est,i})}{\partial \psi} \tag{8.4}$$

$$\hat{\boldsymbol{\psi}}_{t,i} \longleftarrow \hat{\boldsymbol{\psi}}_{t-1,i} - \epsilon_{\psi,t-1} \frac{\partial MSE(y_i - \hat{y}_{est,i})}{\partial \psi} \tag{8.5}$$

   where $\epsilon_{\psi,t}$ is the learning rate schedule [See 3.6.2] for this gradient update of ψ at step t, and $\hat{\boldsymbol{\psi}}_{t,i}$ is the value of ψ at iteration t for observation i.

To reach convergence, above assignments have to integrate updates for y as follows:

$$\hat{\boldsymbol{y}}_{est,(i,t)} \longleftarrow f_{\psi_{i,t}}(x) + L_{i,t} \cdot u_{i,t} - \epsilon_{\hat{\boldsymbol{y}}_{est,(i,t)}} \frac{\partial MSE(y_i - \hat{y}_{est,i})}{\partial \hat{\boldsymbol{y}}_{est,i}} \tag{8.6}$$

where $\frac{\partial MSE(y_i - \hat{y}_{est,(i,t)})}{\partial \hat{\boldsymbol{y}}_{est,i}} = 2 * (y_i - \hat{\boldsymbol{y}}_{est,(i,t)})$.

But, at step t in the implementation, $2 * (y_i - \hat{\boldsymbol{y}}_{est,(i,t)})$ is replaced by $2 * (y_i - \hat{\boldsymbol{y}}_{est,(i,t-1)})$. At initial step, $\hat{\boldsymbol{y}}_{est,(i,0)}$ is computed using the model equation and $\hat{\boldsymbol{\psi}}_0$.

So each training data has its own learning rate which is here set as $\epsilon_{\hat{\boldsymbol{y}}_{est,(i,t)}}$. This is valuable for each test data as follows:

$$\hat{\boldsymbol{y}}^{\text{test}}_{est,(i,t)} \longleftarrow f_{\psi_{i,t}}(x^{\text{test}}) + L_{i,t} \cdot u_{i,t} - \epsilon^{\text{test}}_{\hat{\boldsymbol{y}}_{est,(i,t)}} \frac{\partial MSE(y_i - \hat{y}_{est,i})}{\partial \hat{\boldsymbol{y}}_{est,i}} \tag{8.7}$$

where for the k-th test data x_k^{test} the changes $f_{\psi_{i,t}}$ are taken from the j-th training data y_j which verify:

$$j^{\text{choosen}} = \underset{x_j \in \text{ Training Set}}{\arg\min} \quad \text{Mean}(x_j - x_k^{test}) \tag{8.8}$$

where the operation Mean(u) for vector u is taken upon all dimension of u. The criteria used in optimization [4.29] can be modify for a distance $dist$ for which each test data x_k^{test} is ensured to find an associate x_i^{train} in the training data with :

$$dist(x_k^{\text{test}}, x_i^{\text{train}}) \leq \varepsilon \tag{8.9}$$

where ε is a very small number. This presented framework will be called $(\boldsymbol{\zeta}, \epsilon_{dbs}) - \textbf{GDBS}$, and augmented with the data Augmentation for Empirical Differentiation (DAED) framework in section 5.3, shall be called the *dist*-NN-(h)-TS-PMI-$(l_1, \boldsymbol{\zeta}, \epsilon_{dbs}) - \textbf{GDBS}$. When the model is truly differentiable, with assumption $(\mathcal{F}_d)$ being valid, the learning with this model is almost surely perfect.

8.0.7 The Infinite Zelda Stochastic Game.

This is a stochastic game framework we have derived for a direct application scheme of the Potts Shrinkage model coupled with the Shallow Potts Gibbs Models. We all know that the next generation of games will come from artificial intelligence (AI) networks (Hsu (2004), Hassabis (2017), DeCoste (1997), Granter et al. (2017)). The core idea of this part is inspired from Bowling & Veloso (2002) that have described a scalable learning algorithm for stochastic games, and mainly the equivalence between Potts Model and percolation (Essam (1979), Ding et al. (2012), Kemppainen & Smirnov (2019)). We present here the Infinite Zelda Game (IZG) that rely on the learning process of the Shallow Gibbs Network (SGN) model.

The random partitions can be seen as random graphs. Each partition from the random bond Potts models used for its simulation presents some connected components and isolated elements. With a given subset of data $\{(x_i, y_i)\}_{1 \leq i \leq N}$, the graph generated at each step is equivalent to a network graph built from pair interactions between the neighbors. Regardless of the connectivity structure, the graph has v^N state (s) if each data point is characterized by v possible states (or spins). Each of the v states are represented graphically by a color. At each step (a new distinct *generated* partition from a previous one), y_i can

move from state v_1 to v_2, thanks to the clustering process, and its estimate from the Shallow Gibbs Network (SGN) is denoted $\hat{y}_i$.

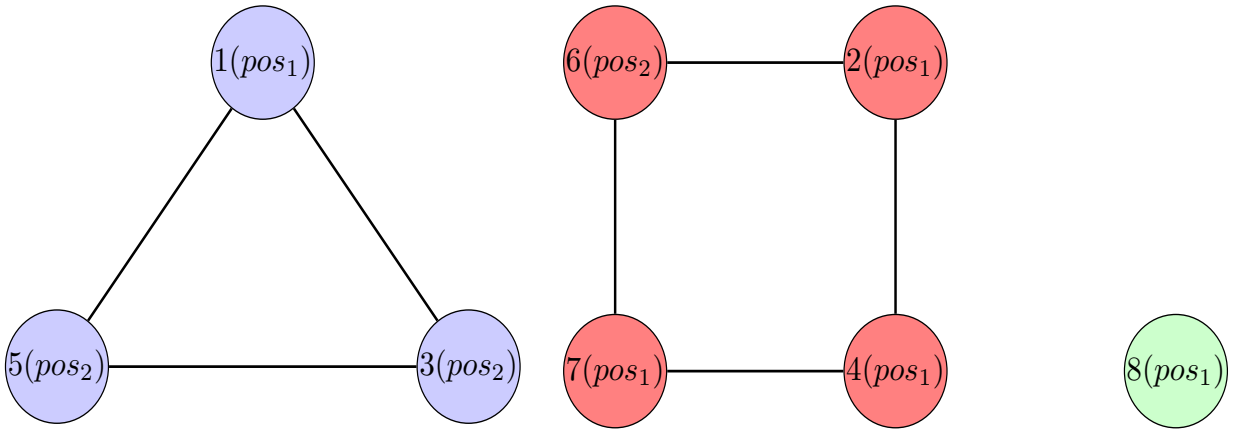


Figure 8.1: A 3-component Potts graph of size 8. Shrinkage constraint is reduced to 1.

The green colored circle shall indicate isolated points, and the same colored squared indicates a set of points all connected forming a component. Given the size of the hidden layer of the Shallow Gibbs Network (SGN), there two possibilities (denoted pos_1 and pos_2 and detailed in the following) [see figure 8.1]:

i. Estimated $\hat{y}_i$ is close (in distance metric) to real y_i more than its assigned cluster S_i estimated mean $\frac{1}{s_i}\sum_{j\in S_i} y_j$, with $|S_i| = s_i$. This possibility is called pos_1.

ii. Estimated $\hat{y}_i$ is not close (in distance metric) to y_i more than its its assigned cluster S_i mean $\frac{1}{s_i}\sum_{j\in S_i} y_j$, with $|S_i| = s_i$. This possibility is called pos_2.

We set each available dataset $\mathcal{D}_s$ to represent a Mansion (the training part). In each Mansion, the player has to maximize its coins collection to be eligible to move out to another Mansion. Each generated partition proposes a conditional estimate $\hat{y}_i$ for $y_i \in \mathbf{Y} = \{y_i, 1 \leq i \leq q\}$ through the Shallow Gibbs Network (SGN). The player has to find *by mouse or finger click* in the network graph, those colored points in the position pos_2. Obviously, the isolated points (in white color) are always in position pos_2.

To facilitate the game, we always declare the number b of data points in position pos_2, and ask the player to find them in b_e trials with $b + 5 = b_e$. Because the partitions are generated by Swenden-Wang cuts (Barbu & Zhu (2003)), the game is set for the player to accumulate coins as many as he detects the right squares in position pos_2, in few trials, and in a given recorded time t_e.

The Goal of the Infinite Zelda Game (IZG): "Win or Learn More" The goal of the Infinite Zelda Game (IZG) is to maximize your coins through the mansions. The more you win, the quicker you can move to another mansion. One more trick is to be done each time before generating a partition (a new network graph): the player has to choose the number of *dist*-NN-(h)-TS-PMI-$(l_1, \boldsymbol{\zeta}, \epsilon_{dbs})-$**DBS** updates for the Shallow Gibbs Network (SGN) characterising its ***power level***. By default, the power is set to $\mathbf{5}$, and he has to buy more coins to set more power.

The more *dist*-NN-(h)-TS-PMI-$(l_1, \boldsymbol{\zeta}, \epsilon_{dbs})$ − **DBS** updates you set, the more you may win at each partition configuration, because it reduces the number of data points (y_i) in position pos_2 with a Potts Shrinkage constraint $\geq m \geq 5$: the more ***power level*** you set, the more you reduce the number of positions pos_2 to find. The Infinite Zelda Game (IZG) for the Shallow Gibbs Network (SGN) can be summarized in the WoLM principle **("Win or Learn More")** which encourages convergence.

8.0.8 Other potential researches.

Metric learning is the doorway to find for any recognition or classification task a *sufficient* invariant descriptor, that can resists to any deformation such as an occlusion, clutter, etc. One may need to combine multiple metrics to reach this goal, such as: Holographic Nearest Neighbor (HNN) [see 6.3.1], Shape Context [see 6.3.2], Hough transformations features [6.3.3], Fourier descriptors [see 6.3.4], Chord distribution [see 6.4.1], Moment invariants method [6.4.2], Holographic Nearest Neighbor Class of a digit [6.5]. Is it that possible: ***like fingerprints are unique and identifiable for everyone, that each digit from a given source is unique and identifiable through a unique combination of invariance descriptors ?*** Our question stays opened with potential researches, with great advantages for the future of Artificial Intelligence.

We are also doing some experiments with Convolutional Neural Networks presented in chapter 7 [see 7]. This is a conceptual note, that may further get improved in many ways. Essentially, we have introduced some novel architectures to explore [7.4.1] and have highlighted the importance of invariant networks [7.4.3] as universal approximators. This chapter is also an attempt to propose some directives to solve open questions in chapter [6].

Random Neural Networks and Stationary Architectures. We are reviewing the work of Erol Gelenbe, a pionner in random neural networks. Random connections in neural networks have also raised interest because of their Universal approximation properties [Gelenbe et al. (2006), Gelenbe (1993), Gelenbe et al. (1999a), Gelenbe et al. (1999b)]. The goal of our research is to develop approximate stationary architectures during a learning process [Gelenbe (1990), Bakırcıoğlu & Koçak (2000)].

Graph Theory, Lie Groups, Non-Commutative algebras, and Infinite Representations for Deep learning architectures: discovery of an infinite machine The progression of Deep learning has been dependent upon its architecture all those years of its edge development. We found that using Graph Theory, Lie Groups, Non-Commutative algebras, and some infinite representations, different high level architectures arise easily that may reinforce existent ones. The proposed new representations come from complex objects accessible through formalized algebraic theory of graphs, and evidence from exploratory analysis, including some known geometric structures. From now, it is possible to extend deep learning architectures to such an infinite scale of embedded networks that they may have simultaneously an infinite number of backwards and forwards feed connections based on those novel representations and their respective patterns. We also found that fully-connected networks, stacked networks, convolutional networks, fractal networks or even spherical networks are just some derivatives from those complex representations. The Cartan–Killing classification of Lie groups [Procesi (2007)], and Non-commutative Penrose tilling objects [Akiyama & Arnoux (2017)], Cayley graphs [Alon & Roichman (1994), Heydemann (1997), Kelarev (2002), Lakshmivarahan et al. (1993)], Metatron Cube and HyperCube patterns (Knill & Slavkovsky (2013)), as well as Olde Grafham Geometric tiles were all useful to illustrate our findings. For example, Lie groups are useful as they propose equivariant

transformations to neural networks (Finzi et al. (2020)), stacking is the method to get fractal recursive networks (Larsson et al., 2016), and tiling helps to generate Turing machines (Demaine et al. (2012), Glassner (1998), Gopinath et al. (2011), Toth et al. (1987)) This research is completely novel in his approaches, but can be seen as a much more general and deeper version of the work of Ngiam et al. (2010), applicable to any neural network.

Syracuse as a Machine Learning Problem. We have initiated a work on Syracuse problem in 2017 (Alahassa, 2017), with partial results (the proof in the document is not complete). We are currently proceeding reviews to answer the question if deep learning or machine learning can help to solve the collatz conjecture, because this may be possible [Koch et al. (2020), Jiang (2021)].

Bibliography

Aarts, E., & Korst, J. (1988). Simulated annealing and boltzmann machines.

Abdel-Hakim, A. E., & Farag, A. A. (2006). Csift: A sift descriptor with color invariant characteristics. In *2006 IEEE computer society conference on computer vision and pattern recognition (CVPR'06)*, vol. 2, (pp. 1978–1983). Ieee.

Abid, S., Fnaiech, F., & Najim, M. (2001). A fast feedforward training algorithm using a modified form of the standard backpropagation algorithm. *IEEE Transactions on neural networks*, *12*(2), 424–430.

Adhan, S., & Pintavirooj, C. (2014). Alphabetic hand sign interpretation using geometric invariance. In *The 7th 2014 Biomedical Engineering International Conference*, (pp. 1–4). IEEE.

Agarap, A. F. (2018). Deep learning using rectified linear units (relu). *arXiv preprint arXiv:1803.08375*.

Agostinelli, F., Hoffman, M., Sadowski, P., & Baldi, P. (2014). Learning activation functions to improve deep neural networks. *arXiv preprint arXiv:1412.6830*.

Agusta, Y., & Dowe, D. L. (2003). Unsupervised learning of correlated multivariate gaussian mixture models using mml. In *Australasian Joint Conference on Artificial Intelligence*, (pp. 477–489). Springer.

Ahsanullah, M., Kibria, B. G., & Shakil, M. (2014). Normal distribution. In *Normal and Student´ st Distributions and Their Applications*, (pp. 7–50). Springer.

Aiyer, S. V., Niranjan, M., & Fallside, F. (1990). A theoretical investigation into the performance of the hopfield model. *IEEE transactions on neural networks*, *1*(2), 204–215.

Akiyama, S., & Arnoux, P. (2017). Substitution and tiling dynamics: Introduction to self-inducing structures.

Al-Hami, M., Pietron, M., Casas, R., Hijazi, S., & Kaul, P. (2018). Towards a stable quantized convolutional neural networks: an embedded perspective. In *10th International conference on agents and artificial intelligence (ICAART)*, vol. 2, (pp. 573–580).

Alahassa, K.-A. (2020).
URL `http://my-github.com/kgalahassa`

Alahassa, N. K.-A. (2017). Arithmétique cantus, graphe de syracuse, et preuve de la conjecture de collatz.

ALAHASSA, N. K.-A. (2018). P-lag ratio analysis for sound pattern detection in time series.

Alahassa, N. K.-A., & Murua, A. (2020). Shallow structured potts neural network regression (s-spnnr). *Proceedings of the Edge Intelligence Workshop 2020, Les Cahiers du GERAD G–2020–23*. URL `https://www.gerad.ca/en/papers/G-2020-23-EIW03`

Alali, F. A., & Romero, S. (2013). Benford's law: Analyzing a decade of financial data. *Journal of Emerging Technologies in Accounting*, *10*(1), 1–39.

Albawi, S., Mohammed, T. A., & Al-Zawi, S. (2017). Understanding of a convolutional neural network. In *2017 International Conference on Engineering and Technology (ICET)*, (pp. 1–6). IEEE.

Alexander, C. (2004). Normal mixture diffusion with uncertain volatility: Modelling short-and long-term smile effects. *Journal of Banking & Finance*, *28*(12), 2957–2980.

Alferez, R., & Wang, Y.-F. (1999). Geometric and illumination invariants for object recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *21*(6), 505–536.

Alghoniemy, M., & Tewfik, A. H. (2004). Geometric invariance in image watermarking. *IEEE transactions on image processing*, *13*(2), 145–153.

Ali, J., Khan, R., Ahmad, N., & Maqsood, I. (2012). Random forests and decision trees. *International Journal of Computer Science Issues (IJCSI)*, *9*(5), 272.

Alon, N., & Roichman, Y. (1994). Random cayley graphs and expanders. *Random Structures & Algorithms*, *5*(2), 271–284.

Alpaydin, E. (2004). Machine learning. *The New AI*.

Alpaydin, E. (2016). *Machine learning: the new AI*. MIT press.

Altman, D. G., & Bland, J. M. (1995). Statistics notes: the normal distribution. *Bmj*, *310*(6975), 298.

Amari, . N. H., S. (2007). *Methods of Information Geometry*. American Mathematical Society.

Amari, S.-i. (1993). Backpropagation and stochastic gradient descent method. *Neurocomputing*, *5*(4-5), 185–196.

Améndola, C., Faugere, J.-C., & Sturmfels, B. (2015). Moment varieties of gaussian mixtures. *arXiv preprint arXiv:1510.04654*.

Amit, Y., & Murua, A. (2001). Speech recognition using randomized relational decision trees. *IEEE transactions on speech and audio processing*, *9*(4), 333–341.

Amos, B., & Yarats, D. (2020). The differentiable cross-entropy method. In *International Conference on Machine Learning*, (pp. 291–302). PMLR.

Anastassiou, G. A. (2011a). Multivariate hyperbolic tangent neural network approximation. *Computers & Mathematics with Applications*, *61*(4), 809–821.

Anastassiou, G. A. (2011b). Univariate hyperbolic tangent neural network approximation. *Mathematical and Computer Modelling*, *53*(5-6), 1111–1132.

Andersson, S. A., Madigan, D., & Perlman, M. D. (2001). Alternative markov properties for chain graphs. *Scandinavian journal of statistics*, *28*(1), 33–85.

Andriotis, P., Oikonomou, G., & Tryfonas, T. (2013). Jpeg steganography detection with benford's law. *Digital Investigation*, *9*(3-4), 246–257.

Andrychowicz, M., Denil, M., Gomez, S., Hoffman, M. W., Pfau, D., Schaul, T., Shillingford, B., & De Freitas, N. (2016). Learning to learn by gradient descent by gradient descent. In *Advances in neural information processing systems*, (pp. 3981–3989).

Antoniak, C. E. (1974). Mixtures of dirichlet processes with applications to bayesian nonparametric problems. *The annals of statistics*, (pp. 1152–1174).

Anwar, S., Hwang, K., & Sung, W. (2017). Structured pruning of deep convolutional neural networks. *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, *13*(3), 1–18.

Arbter, K., Snyder, W. E., Burkhardt, H., & Hirzinger, G. (1990). Application of affine-invariant fourier descriptors to recognition of 3-d objects. *IEEE Transactions on pattern analysis and machine intelligence*, *12*(7), 640–647.

Archambeau, C., Lee, J. A., Verleysen, M., et al. (2003). On convergence problems of the em algorithm for finite gaussian mixtures. In *ESANN*, vol. 3, (pp. 99–106).

Ardakani, A., Condo, C., & Gross, W. J. (2016). Sparsely-connected neural networks: towards efficient vlsi implementation of deep neural networks. *arXiv preprint arXiv:1611.01427*.

Ari, B., & Güvenir, H. (2002). Clustered linear regression. *Knowledge-Based Systems*, *15*(3), 169 – 175.

Arjevani, Y., & Field, M. (2020). Symmetry & critical points for a model shallow neural network. *arXiv preprint arXiv:2003.10576*.

Arora, R., Basu, A., Mianjy, P., & Mukherjee, A. (2016). Understanding deep neural networks with rectified linear units. *arXiv preprint arXiv:1611.01491*.

Arsad, P. M., Buniyamin, N., et al. (2013). A neural network students' performance prediction model (nnsppm). In *2013 IEEE International Conference on Smart Instrumentation, Measurement and Applications (ICSIMA)*, (pp. 1–5). IEEE.

Ashkin, J., & Teller, E. (1943). Statistics of two-dimensional lattices with four components. *Physical Review*, *64*(5-6), 178.

Asikainen, J., Aharony, A., Mandelbrot, B., Rausch, E., & Hovi, J.-P. (2003). Fractal geometry of critical potts clusters. *The European Physical Journal B-Condensed Matter and Complex Systems*, *34*(4), 479–487.

Attali, J.-G., & Pagès, G. (1997). Approximations of functions by a multilayer perceptron: a new approach. *Neural networks*, *10*(6), 1069–1081.

Aurelio, Y. S., de Almeida, G. M., de Castro, C. L., & Braga, A. P. (2019). Learning from imbalanced data sets with weighted cross-entropy function. *Neural Processing Letters*, *50*(2), 1937–1949.

Ay, N., Jost, J., Vân Lê, H., & Schwachhöfer, L. (2015). Information geometry and sufficient statistics. *Probability Theory and Related Fields*, *162*(1-2), 327–364.

Ayodele, T. O. (2010). Machine learning overview. *New Advances in Machine Learning*, (pp. 9–19).

Azizian, W., & Lelarge, M. (2020). Characterizing the expressive power of invariant and equivariant graph neural networks. *arXiv preprint arXiv:2006.15646*.

Bach, F. (2018). Statistical machine learning and convex optimization.

Badal-Valero, E., Alvarez-Jareño, J. A., & Pavía, J. M. (2018). Combining benford's law and machine learning to detect money laundering. an actual spanish court case. *Forensic science international*, *282*, 24–34.

Bagnell, D. (2020). Gibbs fields and markov random fields. *Statistical Techniques in Robotics (16-831, F10)*, *7*.

Bahl, L. R., Desouza, P. V., Mercer, R. L., & Picheny, M. A. (1992). Feneme-based markov models for words. US Patent 5,165,007.

Baird, L., Smalenberger, D., & Ingkiriwang, S. (2005). One-step neural network inversion with pdf learning and emulation. In *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.*, vol. 2, (pp. 966–971). IEEE.

Bakırcıoğlu, H., & Koçak, T. (2000). Survey of random neural network applications. *European journal of operational research*, *126*(2), 319–330.

Bakker, B. (2001). Reinforcement learning with long short-term memory. *Advances in neural information processing systems*, *14*, 1475–1482.

Balakrishnan, K. (2018). *Exponential distribution: theory, methods and applications*. Routledge.

Baldassi, C., Malatesta, E. M., & Zecchina, R. (2019). Properties of the geometry of solutions and capacity of multilayer neural networks with rectified linear unit activations. *Physical review letters*, *123*(17), 170602.

Balkema, W., & van der Heijden, F. (2010). Music playlist generation by assimilating gmms into soms. *Pattern Recognition Letters*, *31*(11), 1396–1402.

Ballard, D. H. (1981). Generalizing the hough transform to detect arbitrary shapes. *Pattern recognition*, *13*(2), 111–122.

Ballard, D. H., & Brown, C. M. (1982). Computer vision, article, 4 pages prentice-hall. *Englewood Cliffs, New Jersey, believed to be published more than one year prior to the filing date of the present application*.

Banner, R., Nahshan, Y., & Soudry, D. (2019). Post training 4-bit quantization of convolutional networks for rapid-deployment. In *Advances in Neural Information Processing Systems*, (pp. 7948–7956).

Bansal, A., Kauffman, R. J., & Weitz, R. R. (1993). Comparing the modeling performance of regression and neural networks as data quality varies: a business value approach. *Journal of Management Information Systems*, *10*(1), 11–32.

Barbarossa, D., & Manzonetto, G. (2019). About the power of taylor expansion. In *3rd International Workshop on Trends in Linear Logic and Applications*.

Barber, D., & Bishop, C. (1997). Ensemble learning for multi-layer networks. *Advances in neural information processing systems*, *10*, 395–401.

Barber, D., & Bishop, C. M. (1998). Ensemble learning in bayesian neural networks. *Nato ASI Series F Computer and Systems Sciences*, *168*, 215–238.

Barbu, A., & Zhu, S.-C. (2003). Graph partition by swendsen-wang cuts. In *null*, (p. 320). IEEE.

Barcella, W., De Iorio, M., & Baio, G. (2017). A comparative review of variable selection techniques for covariate dependent dirichlet process mixture models. *Canadian Journal of Statistics*, *45*(3), 254–273.

Barra, A., Bernacchia, A., Santucci, E., & Contucci, P. (2012). On the equivalence of hopfield networks and boltzmann machines. *Neural Networks*, *34*, 1–9.

Barry, D., & Hartigan, J. A. (1992). Product partition models for change point problems. *The Annals of Statistics*, (pp. 260–279).

Bartholomew-Biggs, M., Brown, S., Christianson, B., & Dixon, L. (2000). Automatic differentiation of algorithms. *Journal of Computational and Applied Mathematics*, *124*(1), 171 – 190. Numerical Analysis 2000. Vol. IV: Optimization and Nonlinear Equations.

Bastani, V., Marcenaro, L., & Regazzoni, C. (2014). Unsupervised trajectory pattern classification using hierarchical dirichlet process mixture hidden markov model. In *2014 IEEE International Workshop on Machine Learning for Signal Processing (MLSP)*, (pp. 1–6). IEEE.

Başt anlar, Y., & Özuysal, M. (2014). Introduction to machine learning. In *miRNomics: MicroRNA Biology and Computational Analysis*, (pp. 105–128). Springer.

Beal, M., Ghahramani, Z., & Rasmussen, C. (2001). The infinite hidden markov model. *Advances in neural information processing systems*, *14*, 577–584.

Beam, A. L., & Kohane, I. S. (2018). Big data and machine learning in health care. *Jama*, *319*(13), 1317–1318.

Beamer, S., Asanovic, K., & Patterson, D. (2012). Direction-optimizing breadth-first search. In *SC'12: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, (pp. 1–10). IEEE.

Behera, L., Kumar, S., & Patnaik, A. (2006). On adaptive learning rate that guarantees convergence in feedforward networks. *IEEE transactions on neural networks*, *17*(5), 1116–1125.

Behrmann, J., Grathwohl, W., Chen, R. T., Duvenaud, D., & Jacobsen, J.-H. (2019). Invertible residual networks. In *International Conference on Machine Learning*, (pp. 573–582). PMLR.

Bejjani, W., Agboh, W. C., Dogar, M. R., & Leonetti, M. (2020). Occlusion-aware search for object retrieval in clutter. *arXiv preprint arXiv:2011.03334*.

Bela, A., Frigyik, A., & Gupta, M. (2010). Introduction to the dirichlet distribution and related processes. *Department of Electrical Engineering, University of Washington*.

Belongie, S., Malik, J., & Puzicha, J. (2002). Shape matching and object recognition using shape contexts. *IEEE transactions on pattern analysis and machine intelligence*, *24*(4), 509–522.

Bengio, Y. (2012). Practical recommendations for gradient-based training of deep architectures. In *Neural networks: Tricks of the trade*, (pp. 437–478). Springer.

Bengio, Y., & Grandvalet, Y. (2004). No unbiased estimator of the variance of k-fold cross-validation. *Journal of machine learning research*, *5*(Sep), 1089–1105.

Berger, A., & Hill, T. P. (2015). *An introduction to Benford's law*. Princeton University Press.

Berger, A., Hill, T. P., et al. (2011). A basic theory of benford's law. *Probability Surveys*, *8*, 1–126.

Berrar, D. (2019). Cross-validation. *Encyclopedia of Bioinformatics and Computational Biology*, *1*, 542–545.

Bertolazzi, E. (2008). One-dimensional minimization. Lecture Notes, Università di Trento.

Besag, J. (2004). Markov Chain Monte Carlo Methods for Statistical Inference. Tech. rep., Department of Statistics, University of Washington, Seattle, USA.

Betancourt, B., Zanella, G., & Steorts, R. C. (2020). Random partition models for microclustering tasks. *arXiv preprint arXiv:2004.02008*.

Bhattacharyay, A. (2019). Equilibrium of a brownian particle with coordinate dependent diffusivity and damping: Generalized boltzmann distribution. *Physica A: Statistical Mechanics and its Applications*, *515*, 665–670.

Biamonte, J., Wittek, P., Pancotti, N., Rebentrost, P., Wiebe, N., & Lloyd, S. (2017). Quantum machine learning. *Nature*, *549*(7671), 195–202.

Bianchini, M., & Scarselli, F. (2014). On the complexity of neural network classifiers: A comparison between shallow and deep architectures. *IEEE transactions on neural networks and learning systems*, *25*(8), 1553–1565.

Biau, G., & Devroye, L. (2010). On the layered nearest neighbour estimate, the bagged nearest neighbour estimate and the random forest method in regression and classification. *Journal of Multivariate Analysis*, *101*(10), 2499–2518.

Biernacki, C., Celeux, G., & Govaert, G. (1999). An improvement of the nec criterion for assessing the number of clusters in a mixture model. *Pattern Recognition Letters*, *20*(3), 267–272.

Biess, A., Flash, T., & Liebermann, D. G. (2011). Riemannian geometric approach to human arm dynamics, movement optimization, and invariance. *Physical Review E*, *83*(3), 031927.

Bingham, E., & Mannila, H. (2001). Random projection in dimensionality reduction: applications to image and text data. In *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, (pp. 245–250).

Bishop, C. (1991). Improving the generalization properties of radial basis function neural networks. *Neural computation*, *3*(4), 579–588.

Bishop, C. M. (1997). Bayesian Neural Networks. *Journal of the Brazilian Computer Society*, *4*.

Bishop, C. M. (2006a). *Pattern recognition and machine learning*. springer.

Bishop, C. M. (2006b). *Pattern Recognition and Machine Learning*. Springer.

Blackledge, J., Bezobrazov, S., & Tobin, P. (2015). Cryptography using artificial intelligence. In *2015 International Joint Conference on Neural Networks (IJCNN)*, (pp. 1–6). IEEE.

Blackwell, D., MacQueen, J. B., et al. (1973). Ferguson distributions via pólya urn schemes. *The annals of statistics*, *1*(2), 353–355.

Blatt, M., Wiseman, S., & Domany, E. (1996a). Clustering data through an analogy to the potts model. In *Advances in Neural Information Processing Systems*, (pp. 416–422).

Blatt, M., Wiseman, S., & Domany, E. (1996b). Superparamagnetic clustering of data. *Physical review letters*, *76*(18), 3251.

Blatt, M., Wiseman, S., & Domany, E. (1996c). Superparamagnetic clustering of data. *Physical Review Letters*, *76*, 3251–3254.

Blatt, M., Wiseman, S., & Domany, E. (1997). Data clustering using a model granular magnet. *Neural Computation*, *9*(8), 1805–1842.

Blei, D. M., & Frazier, P. I. (2011). Distance dependent chinese restaurant processes. *Journal of Machine Learning Research*, *12*(8).

Blei, D. M., Kucukelbir, A., & McAuliffe, J. D. (2017). Variational inference: A review for statisticians. *Journal of the American statistical Association*, *112*(518), 859–877.

Blott, M., Preußer, T. B., Fraser, N. J., Gambardella, G., O'brien, K., Umuroglu, Y., Leeser, M., & Vissers, K. (2018). Finn-r: An end-to-end deep-learning framework for fast exploration of quantized neural networks. *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, *11*(3), 1–23.

Blum, A., Kalai, A., & Langford, J. (1999). Beating the hold-out: Bounds for k-fold and progressive cross-validation. In *Proceedings of the twelfth annual conference on Computational learning theory*, (pp. 203–208).

Blundell, C., Cornebise, J., Kavukcuoglu, K., & Wierstra, D. (2015). Weight uncertainty in neural networks. In *Proceedings of the 32Nd International Conference on International Conference on Machine Learning - Volume 37*, ICML'15, (pp. 1613–1622). JMLR.org.
URL `http://dl.acm.org/citation.cfm?id=3045118.3045290`

Bonaccorso, G. (2017). *Machine learning algorithms*. Packt Publishing Ltd.

Bonmassar, G., & Schwartz, E. L. (1994). Geometric invariance in space-variant vision systems: the exponential chirp transform. In *Proceedings of the 12th IAPR International Conference on Pattern Recognition, Vol. 2-Conference B: Computer Vision & Image Processing.(Cat. No. 94CH3440-5)*, (pp. 204–207). IEEE.

Borgs, C., Chayes, J. T., & Tetali, P. (2012). Tight bounds for mixing of the swendsen–wang algorithm at the potts transition point. *Probability Theory and Related Fields*, *152*(3-4), 509–557.

Bors, A. G., & Pitas, I. (1996). Median radial basis function neural network. *IEEE transactions on Neural Networks*, *7*(6), 1351–1364.

Bosman, A. S., Engelbrecht, A., & Helbig, M. (2020). Visualising basins of attraction for the cross-entropy and the squared error neural network loss functions. *Neurocomputing*.

Botev, Z., & Kroese, D. P. (2004). Global likelihood optimization via the cross-entropy method with an application to mixture models. In *Proceedings of the 2004 Winter Simulation Conference, 2004.*, vol. 1. IEEE.

Bottasso, C. L., Borri, M., & Trainelli, L. (2002). Geometric invariance. *Computational mechanics*, *29*(2), 163–169.

Bottou, L. (1991). Stochastic gradient learning in neural networks. *Proceedings of Neuro-Nımes*, *91*(8), 12.

Bottou, L. (1998). Online learning and stochastic approximations. *On-line learning in neural networks*, *17*(9), 142.

Bottou, L. (2012). Stochastic gradient descent tricks. In *Neural networks: Tricks of the trade*, (pp. 421–436). Springer.

Bottou, L., & Cun, Y. (2003). Large scale online learning. *Advances in neural information processing systems*, *16*, 217–224.

Boulanger-Lewandowski, N., Bengio, Y., & Vincent, P. (2012). Modeling temporal dependencies in high-dimensional sequences: Application to polyphonic music generation and transcription. *arXiv preprint arXiv:1206.6392*.

Bourely, A., Boueri, J. P., & Choromonski, K. (2017). Sparse neural networks topologies. *arXiv preprint arXiv:1706.05683*.

Bourke, P. (1999). Interpolation methods. *Miscellaneous: projection, modelling, rendering*, *1*.

Bouvrie, J. (2006). Notes on convolutional neural networks.

Bowling, M., & Veloso, M. (2002). Scalable learning in stochastic games. In *AAAI Workshop on Game Theoretic and Decision Theoretic Agents*, (pp. 11–18).

Bozdogan, H. (1993). Choosing the number of component clusters in the mixture-model using a new informational complexity criterion of the inverse-fisher information matrix. In *Information and classification*, (pp. 40–54). Springer.

Brame, R., Nagin, D. S., & Wasserman, L. (2006). Exploring some analytical characteristics of finite mixture models. *Journal of Quantitative Criminology*, *22*(1), 31–59.

Brandenburger, A., & Steverson, K. (2019). Axioms for the boltzmann distribution. *Foundations of physics*, *49*(5), 444–456.

Braun, M., & McAuliffe, J. (2010). Variational inference for large-scale models of discrete choice. *Journal of the American Statistical Association*, *105*(489), 324–335.

Bresler, G. (2015). Efficiently learning ising models on arbitrary graphs. In *Proceedings of the forty-seventh annual ACM symposium on Theory of computing*, (pp. 771–782).

Brigo, D., & Mercurio, F. (2002). Lognormal-mixture dynamics and calibration to market volatility smiles. *International Journal of Theoretical and Applied Finance*, *5*(04), 427–446.

Broderick, T., Jordan, M. I., Pitman, J., et al. (2012). Beta processes, stick-breaking and power laws. *Bayesian analysis*, *7*(2), 439–476.

Browder, A. (2012). *Mathematical analysis: an introduction*. Springer Science & Business Media.

Brown, M., Hughey, R., Krogh, A., Mian, I. S., Sjölander, K., & Haussler, D. (1993). Using dirichlet mixture priors to derive hidden markov models for protein families. In *Ismb*, vol. 1, (pp. 47–55).

Brush, S. G. (1967). History of the lenz-ising model. *Reviews of modern physics*, *39*(4), 883.

Bryc, W. (2012). *The normal distribution: characterizations with applications*, vol. 100. Springer Science & Business Media.

Bundy, A., & Wallen, L. (1984). Breadth-first search. In *Catalogue of artificial intelligence tools*, (pp. 13–13). Springer.

Burke, J., & Kincanon, E. (1991). Benford's law and physical constants: the distribution of initial digits. *American Journal of Physics*, *59*(10), 952.

Busta, B., & Weinberg, R. (1998). Using benford's law and neural networks as a review procedure. *Managerial Auditing Journal*.

Cai, Y., Tang, T., Xia, L., Cheng, M., Zhu, Z., Wang, Y., & Yang, H. (2018). Training low bitwidth convolutional neural network on rram. In *2018 23rd Asia and South Pacific Design Automation Conference (ASP-DAC)*, (pp. 117–122). IEEE.

Campbell, T., & Beronov, B. (2019). Sparse variational inference: Bayesian coresets from scratch. In *Advances in Neural Information Processing Systems*, (pp. 11461–11472).

Campbell, T., & Li, X. (2019). Universal boosting variational inference. In *Advances in Neural Information Processing Systems*, (pp. 3484–3495).

Canuto, C., & Tabacco, A. (2015). *Mathematical analysis II*, vol. 85. Springer.

Cao, F. (2003). *Geometric curve evolution and image processing*. Springer Science & Business Media.

Cao, J., Su, Z., Yu, L., Chang, D., Li, X., & Ma, Z. (2018). Softmax cross entropy loss with unbiased decision boundary for image classification. In *2018 Chinese Automation Congress (CAC)*, (pp. 2028–2032). IEEE.

Cao, R., Zhang, Q., Zhu, J., Li, Q., Li, Q., Liu, B., & Qiu, G. (2020). Enhancing remote sensing image retrieval using a triplet deep metric learning network. *International Journal of Remote Sensing*, *41*(2), 740–751.

Cao, T. P. (2011). *Object recognition*. BoD–Books on Demand.

Cao, Y., Yang, Z., Wang, H., Peng, X., Gao, C., & Li, Y. (2019). Template matching based on geometric invariance in deep neural network. *IEEE Access*, *7*, 82174–82182.

Carbonell, J. G., Michalski, R. S., & Mitchell, T. M. (1983). An overview of machine learning. In *Machine learning*, (pp. 3–23). Elsevier.

Cardy, J., & Ziff, R. M. (2003). Exact results for the universal area distribution of clusters in percolation, ising, and potts models. *Journal of statistical physics*, *110*(1-2), 1–33.

Casella, G., & Berger, R. L. (2002). *Statistical inference*, vol. 2. Duxbury Pacific Grove, CA.

Celeux, G., & Soromenho, G. (1996). An entropy criterion for assessing the number of clusters in a mixture model. *Journal of classification*, *13*(2), 195–212.

Challis, E., & Barber, D. (2013). Gaussian kullback-leibler approximate inference. *The Journal of Machine Learning Research*, *14*(1), 2239–2286.

Chang, C. A., & Su, C.-T. (1995). A comparison of statistical regression and neural network methods in modeling measurement errors for computer vision inspection systems. *Computers & industrial engineering*, *28*(3), 593–603.

Chang, H., & Yeung, D.-Y. (2007). Kernel-based distance metric learning for content-based image retrieval. *Image and Vision Computing*, *25*(5), 695–703.

Chang, J., & Sha, J. (2016). An efficient implementation of 2d convolution in cnn. *IEICE Electronics Express*, (pp. 13–20161134).

Chang, Y.-C. (2009). N-dimension golden section search: Its variants and limitations. In *2009 2nd International Conference on Biomedical Engineering and Informatics*, (pp. 1–6). IEEE.

Chao, W.-L. (2011). Machine learning tutorial. *Digital Image and Signal Processing*.

Charniak, E. (1985). *Introduction to artificial intelligence*. Pearson Education India.

Chaudhuri, B., & Bhattacharya, U. (2000). Efficient training and improved performance of multilayer perceptron in pattern classification. *Neurocomputing*, *34*(1-4), 11–27.

Chekouo, T., & Murua, A. (2015). The penalized biclustering model and related algorithms. *Journal of Applied Statistics*, *42*(6), 1255–1277.

Chekouo, T., & Murua, A. (2018). High-dimensional variable selection with the plaid mixture model for clustering. *Computational Statistics*, *33*(3), 1475–1496.

Chekouo, T., Murua, A., Raffelsberger, W., et al. (2015). The gibbs-plaid biclustering model. *The Annals of Applied Statistics*, *9*(3), 1643–1670.

Chen, A.-S., Leung, M. T., & Daouk, H. (2003). Application of neural networks to an emerging financial market: forecasting and trading the taiwan stock index. *Computers & Operations Research*, *30*(6), 901–923.

Chen, C., Ding, N., & Carin, L. (2015a). On the convergence of stochastic gradient mcmc algorithms with high-order integrators. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'15, (pp. 2278–2286). Cambridge, MA, USA: MIT Press. URL `http://dl.acm.org/citation.cfm?id=2969442.2969494`

Chen, H., Chen, J., & Kalbfleisch, J. D. (2004). Testing for a finite mixture model with two components. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, *66*(1), 95–115.

Chen, H. H., Wang, P. F., Sung, C. T., Yeh, Y. R., & Lee, Y. J. (2013). Energy disaggregation via clustered regression models: A case study in the convenience store. In *2013 Conference on Technologies and Applications of Artificial Intelligence*, (pp. 37–42).

Chen, J., Adebomi, O. E., Olusayo, O. S., & Kulesza, W. (2010a). The evaluation of the gaussian mixture probability hypothesis density approach for multi-target tracking. In *2010 IEEE International Conference on Imaging Systems and Techniques*, (pp. 182–185). IEEE.

Chen, J., & Khalili, A. (2009). Order selection in finite mixture models with a nonsmooth penalty. *Journal of the American Statistical Association*, *104*(485), 187–196.

Chen, J., Tian, J., Lee, N., Zheng, J., Smith, R. T., & Laine, A. F. (2010b). A partial intensity invariant feature descriptor for multimodal retinal image registration. *IEEE Transactions on Biomedical Engineering*, *57*(7), 1707–1718.

Chen, J., Zhu, Z., Li, C., & Zhao, Y. (2019). Self-adaptive network pruning. In *International Conference on Neural Information Processing*, (pp. 175–186). Springer.

Chen, L., Zhou, M., Su, W., Wu, M., She, J., & Hirota, K. (2018). Softmax regression based deep sparse autoencoder network for facial emotion recognition in human-robot interaction. *Information Sciences*, *428*, 49–61.

Chen, S., Cowan, C. F., & Grant, P. M. (1991). Orthogonal least squares learning algorithm for radial basis function networks. *IEEE Transactions on neural networks*, *2*(2), 302–309.

Chen, T., Cullen, R. M., & Godwin, M. (2015b). Hidden markov model using dirichlet process for de-identification. *Journal of biomedical informatics*, *58*, S60–S66.

Chen, X.-W., & Lin, X. (2014). Big data deep learning: challenges and perspectives. *IEEE access*, *2*, 514–525.

Cheney, E. W., & Kincaid, D. R. (2012). *Numerical mathematics and computing*. Cengage Learning.

Cheng, C., Chau, K., Sun, Y., & Lin, J. (2005). Long-term prediction of discharges in manwan reservoir using artificial neural network models. In *International Symposium on Neural Networks*, (pp. 1040–1045). Springer.

Cheng, C., & Parhi, K. K. (2020). Fast 2d convolution algorithms for convolutional neural networks. *IEEE Transactions on Circuits and Systems I: Regular Papers*, *67*(5), 1678–1691.

Cheng, J., Dong, L., & Lapata, M. (2016). Long short-term memory-networks for machine reading. *arXiv preprint arXiv:1601.06733*.

Cho, K. H., Raiko, T., & Ilin, A. (2013). Gaussian-bernoulli deep boltzmann machine. In *The 2013 International Joint Conference on Neural Networks (IJCNN)*, (pp. 1–7). IEEE.

Choo, S., & Lee, H. (2018). Learning framework of multimodal gaussian–bernoulli rbm handling real-value input data. *Neurocomputing*, *275*, 1813–1822.

Choquet-Bruhat, Y., DeWitt-Morette, C., de Witt, C., Bleick, M. D., & Dillard-Bleick, M. (1982). *Analysis*. Gulf Professional Publishing.

Chouldechova, A., & Roth, A. (2018). The frontiers of fairness in machine learning. *arXiv preprint arXiv:1810.08810*.

Chung, J., Gulcehre, C., Cho, K., & Bengio, Y. (2014). Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*.

Chung, J., Gulcehre, C., Cho, K., & Bengio, Y. (2015). Gated feedback recurrent neural networks. In *International conference on machine learning*, (pp. 2067–2075).

Chunseong Park, C., Kim, B., & Kim, G. (2017). Attend to you: Personalized image captioning with context sequence memory networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, (pp. 895–903).

Ciampi, A., & Lechevallier, Y. (1997). Statistical models as building blocks of neural networks. *Communications in Statistics - Theory and Methods*, *26*(4), 991–1009.

Cipra, B. A. (1987). An introduction to the ising model. *The American Mathematical Monthly*, *94*(10), 937–959.

Clark, D., & Vo, B.-N. (2007). Convergence analysis of the gaussian mixture phd filter. *IEEE Transactions on Signal Processing*, *55*(4), 1204–1212.

Clifford, P. (1990). Markov random fields in statistics. *Disorder in physical systems: A volume in honour of John M. Hammersley*, *19*.

Cohen, G., Afshar, S., Tapson, J., & Van Schaik, A. (2017). Emnist: Extending mnist to handwritten letters. In *2017 International Joint Conference on Neural Networks (IJCNN)*, (pp. 2921–2926). IEEE.

Cohen, M. A., & Tan, C. O. (2012). A polynomial approximation for arbitrary functions. *Applied Mathematics Letters*, *25*(11), 1947–1952.

Cohen, P. R. (1995). *Empirical methods for artificial intelligence*, vol. 139. MIT press Cambridge, MA.

Coke, G., & Tsao, M. (2010). Random effects mixture models for clustering electrical load series. *Journal of time series analysis*, *31*(6), 451–464.

Coniglio, A., & Peruggi, F. (1982). Clusters and droplets in the q-state potts model. *Journal of Physics A: Mathematical and General*, *15*(6), 1873.

Cooil, B., Winer, R. S., & Rados, D. L. (1987). Cross-validation for prediction. *Journal of Marketing Research*, *24*(3), 271–279.

Cornelius, L. (1997). *Neural Network Systems Techniques and Applications. Volume 1. Algorithms and Architectures*, vol. 7. Academic Press, 1st edition ed.

Costa, R., Assael, I. A., Shillingford, B., de Freitas, N., & Vogels, T. (2017). Cortical microcircuits as gated-recurrent neural networks. In *Advances in neural information processing systems*, (pp. 272–283).

Courbariaux, M., Hubara, I., Soudry, D., El-Yaniv, R., & Bengio, Y. (2016). Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1. *arXiv preprint arXiv:1602.02830*.

Coviello, E., Vaizman, Y., Chan, A. B., & Lanckriet, G. R. (2012). Multivariate autoregressive mixture models for music auto-tagging. In *ISMIR*, (pp. 547–552).

Cressie, N., & Lele, S. (1992). New models for markov random fields. *Journal of Applied probability*, (pp. 877–884).

Cruz, J. B., Pérez, L. L., & Melo, J. (2011). Convergence of the projected gradient method for quasi-convex multiobjective optimization. *Nonlinear Analysis: Theory, Methods & Applications*, *74*(16), 5268–5273.

Cui, X., & Feng, W.-c. (2019). Iterative machine learning (iterml) for effective parameter pruning and tuning in accelerators. In *Proceedings of the 16th ACM International Conference on Computing Frontiers*, (pp. 16–23).

Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, *2*(4), 303–314.

da Rocha, J. C. F., Guimarães, A. M., & Kozlowski Jr, V. (2011). Convergence of iterative algorithms for learning bayesian networks. *Iberoamerican Journal of Applied Computing*, *1*(2).

Dahl, D. B. (2008). Distance-based probability distribution for set partitions with applications to bayesian nonparametrics. *JSM Proceedings. Section on Bayesian Statistical Science, American Statistical Association, Alexandria, Va*.

Dahl, D. B., Day, R., & Tsai, J. W. (2017). Random partition distribution indexed by pairwise information. *Journal of the American Statistical Association*, *112*(518), 721–732.

Dahl, D. B., et al. (2009). Modal clustering in a class of product partition models. *Bayesian Analysis*, *4*(2), 243–264.

Dahyot, R. (2008). Statistical hough transform. *IEEE Transactions on pattern analysis and machine intelligence*, *31*(8), 1502–1509.

Darken, C., Chang, J., Moody, J., et al. (1992). Learning rate schedules for faster stochastic gradient search. In *Neural networks for signal processing*, vol. 2. Citeseer.

Darken, C., & Moody, J. E. (1991). Note on learning rate schedules for stochastic optimization. In *Advances in neural information processing systems*, (pp. 832–838).

Darroch, J., & Ratcliff, D. (1971). A characterization of the dirichlet distribution. *Journal of the American Statistical Association*, *66*(335), 641–643.

Dash, P., Mishra, S., & Panda, G. (2000). A radial basis function neural network controller for upfc. *IEEE Transactions on Power Systems*, *15*(4), 1293–1299.

Davis, M. H. (2018). *Markov models & optimization*. Routledge.

Day, N. E. (1969). Estimating the components of a mixture of normal distributions. *Biometrika*, *56*(3), 463–474.

de Brébisson, A., & Vincent, P. (2015). An exploration of softmax alternatives belonging to the spherical loss family. *arXiv preprint arXiv:1511.05042*.

De Ceuster, M. J., Dhaene, G., & Schatteman, T. (1998). On the hypothesis of psychological barriers in stock markets and benford's law. *Journal of Empirical Finance*, *5*(3), 263–279.

de Dios, J., & Bruna, J. (2020). On sparsity in overparametrised shallow relu networks. *arXiv preprint arXiv:2006.10225*.

Deckert, J., Myagkov, M., & Ordeshook, P. C. (2011). Benford's law and the detection of election fraud. *Political Analysis*, *19*(3), 245–268.

DeCoste, D. (1997). The future of chess-playing technologies and the significance of kasparov versus deep blue. In *Deep Blue Versus Kasparov: The Significance for Artificial Intelligence*, (pp. 9–13).

Deisenroth, A. A. F. M. P., & Ong, C. S. (????). Mathematics for machine learning. 2019. *url: https://mml-book. github.io*, (p. 407).

Delalleau, O., & Bengio, Y. (2011). Shallow vs. deep sum-product networks. In *Advances in neural information processing systems*, (pp. 666–674).

Demaine, E. D., Demaine, M. L., Fekete, S. P., Patitz, M. J., Schweller, R. T., Winslow, A., & Woods, D. (2012). One tile to rule them all: simulating any turing machine, tile assembly system, or tiling system with a single puzzle piece. *arXiv preprint arXiv:1212.4756*.

Dempster, A. P., Laird, N. M., & Rubin, D. B. (1977). Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society: Series B (Methodological)*, *39*(1), 1–22.

Deng, L. (2012). The mnist database of handwritten digit images for machine learning research [best of the web]. *IEEE Signal Processing Magazine*, *29*(6), 141–142.

Deng, L., Hinton, G., & Kingsbury, B. (2013). New types of deep neural network learning for speech recognition and related applications: An overview. In *2013 IEEE international conference on acoustics, speech and signal processing*, (pp. 8599–8603). IEEE.

Deng, L., & Yu, D. (2014). Deep learning: methods and applications. *Foundations and trends in signal processing*, *7*(3–4), 197–387.

Dey, A. (2016). Machine learning algorithms: a review. *International Journal of Computer Science and Information Technologies*, *7*(3), 1174–1179.

Dey, R., & Salemt, F. M. (2017). Gate-variants of gated recurrent unit (gru) neural networks. In *2017 IEEE 60th international midwest symposium on circuits and systems (MWSCAS)*, (pp. 1597–1600). IEEE.

Di Benedetto, G., Caron, F., & Teh, Y. W. (2017). Non-exchangeable random partition models for microclustering. *arXiv preprint arXiv:1711.07287*.

Diekmann, A. (2007). Not the first digit! using benford's law to detect fraudulent scientific data. *Journal of Applied Statistics*, *34*(3), 321–329.

Diekmann, A., & Jann, B. (2010). Benford's law and fraud detection: Facts and legends. *German economic review*, *11*(3), 397–401.

Dieleman, S., van den Oord, A., & Simonyan, K. (2018). The challenge of realistic music generation: modelling raw audio at scale. *Advances in Neural Information Processing Systems*, *31*, 7989–7999.

Dikusar, N. (2016). Higher-order polynomial approximation. *Mathematical Models and Computer Simulations*, *8*(2), 183–200.

Ding, C., Wang, S., Liu, N., Xu, K., Wang, Y., & Liang, Y. (2019a). Req-yolo: A resource-aware, efficient quantization framework for object detection on fpgas. In *Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, (pp. 33–42).

Ding, C., Wang, Y., & Li, Y. (2012). Potts and percolation models on bowtie lattices. *Physical Review E*, *86*(2), 021125.

Ding, L., & Barbu, A. (2015). Scalable subspace clustering with application to motion segmentation. *Current Trends in Bayesian Methodology with Applications*, (p. 267).

Ding, W., Huang, Z., Huang, Z., Tian, L., Wang, H., & Feng, S. (2019b). Designing efficient accelerator of depthwise separable convolutional neural network on fpga. *Journal of Systems Architecture*, *97*, 278–286.

Ding, X., Ding, G., Han, J., & Tang, S. (2018). Auto-balanced filter pruning for efficient convolutional neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32.

Dinov, I. D. (2008). Expectation maximization and mixture modeling tutorial.

Do, K.-A., Müller, P., & Tang, F. (2005). A bayesian mixture model for differential gene expression. *Journal of the Royal Statistical Society: Series C (Applied Statistics)*, *54*(3), 627–644.

Docevski, M., Zdravevski, E., Lameski, P., & Kulakov, A. (2018). Towards music generation with deep learning algorithms.

Dolezel, P., Honc, D., & Stursa, D. (2019). Predictive controller based on feedforward neural network with rectified linear units. In *Proceedings of the Computational Methods in Systems and Software*, (pp. 1–12). Springer.

Dong, T., & Huang, T. (2019). Neural cryptography based on complex-valued neural network. *IEEE Transactions on Neural Networks and Learning Systems*.

Dong, X., & Yang, Y. (2019). Network pruning via transformable architecture search. In *Advances in Neural Information Processing Systems*, (pp. 760–771).

Doretto, G., & Yao, Y. (2010). Region moments: Fast invariant descriptors for detecting small image structures. In *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, (pp. 3019–3026). IEEE.

Doshi-Velez, F., & Kim, B. (2017). Towards a rigorous science of interpretable machine learning. *arXiv preprint arXiv:1702.08608*.

Dressler, M. (2009). Art of surface interpolation. *Technical University of Liberec Faculty of Mechatronics and Interdisciplinary Engineering Studies: Czech Republic*.

Drucker, H., & Le Cun, Y. (1991). Double backpropagation increasing generalization performance. In *IJCNN-91-Seattle International Joint Conference on Neural Networks*, vol. 2, (pp. 145–150). IEEE.

Drucker, H., & Le Cun, Y. (1992). Improving generalization performance using double backpropagation. *IEEE Transactions on Neural Networks*, *3*(6), 991–997.

Du, S., Lee, J., Li, H., Wang, L., & Zhai, X. (2019). Gradient descent finds global minima of deep neural networks. In *International Conference on Machine Learning*, (pp. 1675–1685). PMLR.

Du, X., Cai, Y., Wang, S., & Zhang, L. (2016). Overview of deep learning. In *2016 31st Youth Academic Annual Conference of Chinese Association of Automation (YAC)*, (pp. 159–164). IEEE.

Duan, K., Keerthi, S. S., Chu, W., Shevade, S. K., & Poo, A. N. (2003). Multi-category classification by soft-max combination of binary classifiers. In *International Workshop on Multiple Classifier Systems*, (pp. 125–134). Springer.

Dubey, S. R., Singh, S. K., & Singh, R. K. (2015). Rotation and scale invariant hybrid image descriptor and retrieval. *Computers & Electrical Engineering*, *46*, 288–302.

Duchi, J., Hazan, E., & Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research*, *12*(Jul), 2121–2159.

Duda, R. O., Hart, P. E., et al. (1973). *Pattern classification and scene analysis*, vol. 3. Wiley New York.

Duminil-Copin, H. (2016). *Graphical representations of lattice spin models: cours Peccot, College de France : janvier-fevrier 2015*. Spartacus IDH.
URL `https://books.google.ca/books?id=OPFbswEACAAJ`

Duminil-Copin, H., Sidoravicius, V., & Tassion, V. (2017). Continuity of the phase transition for planar random-cluster and potts models with $1 \leq q \leq 4$. *Communications in Mathematical Physics*, *349*(1), 47–107.

Dumitru, P. D., Plopeanu, M., & Badea, D. (2013). Comparative study regarding the methods of interpolation. *Recent advances in geodesy and Geomatics engineering*, *1*, 45–52.

Dunson, D. B., & Park, J.-H. (2008). Kernel stick-breaking processes. *Biometrika*, *95*(2), 307–323.

Durtschi, C., Hillison, W., & Pacini, C. (2004). The effective use of benford's law to assist in detecting fraud in accounting data. *Journal of forensic accounting*, *5*(1), 17–34.

Dyer, C., Kuncoro, A., Ballesteros, M., & Smith, N. A. (2016). Recurrent neural network grammars. *arXiv preprint arXiv:1602.07776*.

Dym, N., & Maron, H. (2020). On the universality of rotation equivariant point cloud networks. *arXiv preprint arXiv:2010.02449*.

Ecabert, O., & Thiran, J.-P. (2004). Adaptive hough transform for the detection of natural shapes under weak affine transformations. *Pattern Recognition Letters*, *25*(12), 1411–1419.

El Naqa, I., & Murphy, M. J. (2015). What is machine learning? In *Machine Learning in Radiation Oncology*, (pp. 3–11). Springer.

Elfwing, S., Uchibe, E., & Doya, K. (2018). Sigmoid-weighted linear units for neural network function approximation in reinforcement learning. *Neural Networks*, *107*, 3–11.

Elguebaly, T., & Bouguila, N. (2011). Bayesian learning of finite generalized gaussian mixture models on images. *Signal Processing*, *91*(4), 801–820.

Engels, S., Tong, T., & Chan, F. (2015). Automatic real-time music generation for games. In *Eleventh Artificial Intelligence and Interactive Digital Entertainment Conference*.

Epstein, B. (1958). The exponential distribution and its role in life testing. Tech. rep., WAYNE STATE UNIV DETROIT MI.

Ernst, O. K., Bartol, T. M., Sejnowski, T. J., & Mjolsness, E. (2019). Learning moment closure in reaction-diffusion systems with spatial dynamic boltzmann distributions. *Physical Review E*, *99*(6), 063315.

Essam, J. (1979). Potts models, percolation, and duality. *Journal of Mathematical Physics*, *20*(8), 1769–1773.

Etmann, C. (2019). *Double Backpropagation with Applications to Robustness and Saliency Map Interpretability*. Ph.D. thesis, Universität Bremen.

Falas, T., & Stafylopatis, A.-G. (1999). The impact of the error function selection in neural network-based classifiers. In *IJCNN'99. International Joint Conference on Neural Networks. Proceedings (Cat. No. 99CH36339)*, vol. 3, (pp. 1799–1804). IEEE.

Farah, M. J., Rochlin, R., & Klein, K. L. (1994). Orientation invariance and geometric primitives in shape recognition. *Cognitive Science*, *18*(2), 325–344.

Fattah, M. A., & Ren, F. (2009). Ga, mr, ffnn, pnn and gmm based models for automatic text summarization. *Computer Speech & Language*, *23*(1), 126–144.

Feng, J., Tse, C. K., & Lau, F. C. (2003). A neural-network-based channel-equalization strategy for chaos-based communication systems. *IEEE transactions on circuits and systems I: fundamental theory and applications*, *50*(7), 954–957.

Ferguson, T. S. (1973). A bayesian analysis of some nonparametric problems. *The annals of statistics*, (pp. 209–230).

Ferreira, J. A., Loschi, R. H., & Costa, M. A. (2014). Detecting changes in time series: A product partition model with across-cluster correlation. *Signal processing*, *96*, 212–227.

Fewster, R. M. (2009). A simple explanation of benford's law. *The American Statistician*, *63*(1), 26–32.

Figueiredo, M. A., Leitão, J. M., & Jain, A. K. (1999). On fitting mixture models. In *International Workshop on Energy Minimization Methods in Computer Vision and Pattern Recognition*, (pp. 54–69). Springer.

Figueiredo, M. A. T., & Jain, A. K. (2002). Unsupervised learning of finite mixture models. *IEEE Transactions on pattern analysis and machine intelligence*, *24*(3), 381–396.

Fink, G. A. (2014). *Markov models for pattern recognition: from theory to applications*. Springer Science & Business Media.

Finzi, M., Stanton, S., Izmailov, P., & Wilson, A. G. (2020). Generalizing convolutional neural networks for equivariance to lie groups on arbitrary continuous data. In *International Conference on Machine Learning*, (pp. 3165–3176). PMLR.

Fischer, T., & Krauss, C. (2018). Deep learning with long short-term memory networks for financial market predictions. *European Journal of Operational Research*, *270*(2), 654–669.

Fisk, D. L. (1965). Quasi-martingales. *Transactions of the American Mathematical Society*, *120*(3), 369–389.

Flach, P. (2012). *Machine learning: the art and science of algorithms that make sense of data*. Cambridge University Press.

Flach, P. A. (2001). On the state of the art in machine learning: A personal review. *Artificial Intelligence*, *131*(1-2), 199–222.

Flusser, J., Zitova, B., & Suk, T. (2009). *Moments and moment invariants in pattern recognition*. John Wiley & Sons.

Fogel, D. B. (1993). Evolving artificial intelligence.

Fortuin, C. M., & Kasteleyn, P. W. (1972). On the random-cluster model: I. introduction and relation to other models. *Physica*, *57*(4), 536–564.

Fosler-Lussier, E. (1998). Markov models and hidden markov models: A brief tutorial. *International Computer Science Institute*.

Frank, E. (2000). *Pruning decision trees and lists*. Ph.D. thesis, Citeseer.

Fredrickson, G. H., & Andersen, H. C. (1984). Kinetic ising model of the glass transition. *Physical review letters*, *53*(13), 1244.

Freeman, C. D., & Bruna, J. (2016). Topology and geometry of half-rectified network optimization. *arXiv preprint arXiv:1611.01540*.

Friedman, J., Hastie, T., & Tibshirani, R. (2008). Sparse inverse covariance estimation with the graphical lasso. *Biostatistcs*, *9*(3), 432–441.

Friston, K. J., Mattout, J., Trujillo-Barreto, N. J., Ashburner, J., & Penny, W. D. (2007). Variational free energy and the laplace approximation. *NeuroImage*, *34 1*, 220–34.

Fromm, J., Patel, S., & Philipose, M. (2018). Heterogeneous bitwidth binarization in convolutional neural networks. *arXiv preprint arXiv:1805.10368*.

Frühwirth-Schnatter, S., & Pyne, S. (2010). Bayesian inference for finite mixtures of univariate and multivariate skew-normal and skew-t distributions. *Biostatistics*, *11*(2), 317–336.

Frydenberg, M. (1990). The chain graph markov property. *Scandinavian Journal of Statistics*, (pp. 333–353).

Fu, D., Shi, Y. Q., & Su, W. (2007). A generalized benford's law for jpeg coefficients and its applications in image forensics. In *Security, Steganography, and Watermarking of Multimedia Contents IX*, vol. 6505, (p. 65051L). International Society for Optics and Photonics.

Fu, H., Chi, Y., & Liang, Y. (2019). Local geometry of cross entropy loss in learning one-hidden-layer neural networks. In *2019 IEEE International Symposium on Information Theory (ISIT)*, (pp. 1972–1976). IEEE.

Fuglede, B., & Topsoe, F. (2004). Jensen-shannon divergence and hilbert space embedding. In *International Symposium onInformation Theory, 2004. ISIT 2004. Proceedings.*, (p. 31). IEEE.

Fukumizu, K. (1998). Effect of batch learning in multilayer neural networks. *Gen*, *1*(04), 1E–03.

Fung, L. H. (2012). *Model-based Clustering with Network Covariates by Combining a Modified Product Partition Model with Hidden Markov Random Field*. Ph.D. thesis, Chinese University of Hong Kong.

Fuse, T., & Kamiya, K. (2017). Statistical anomaly detection in human dynamics monitoring using a hierarchical dirichlet process hidden markov model. *IEEE Transactions on Intelligent Transportation Systems*, *18*(11), 3083–3092.

Fuster, J. M. (1997). Network memory. *Trends in neurosciences*, *20*(10), 451–459.

Galanis, A., Štefankovič, D., & Vigoda, E. (2019). Swendsen-wang algorithm on the mean-field potts model. *Random Structures & Algorithms*, *54*(1), 82–147.

Gales, M. J. (2001). Multiple-cluster adaptive training schemes. In *2001 IEEE International Conference on Acoustics, Speech, and Signal Processing. Proceedings (Cat. No. 01CH37221)*, vol. 1, (pp. 361–364). IEEE.

Gan, C., & Danai, K. (1999). Fault diagnosis of the ifac benchmark problem with a model-based recurrent neural network. In *Proceedings of the 1999 IEEE International Conference on Control Applications (Cat. No. 99CH36328)*, vol. 2, (pp. 1755–1760). IEEE.

Ganganath, N., Cheng, C.-T., & Tse, C. K. (2014). Data clustering with cluster size constraints using a modified k-means algorithm. Institute of Electrical and Electronics Engineers.

Gao, X., Gallicchio, E., & Roitberg, A. E. (2019). The generalized boltzmann distribution is the only distribution in which the gibbs-shannon entropy equals the thermodynamic entropy. *The Journal of chemical physics*, *151*(3), 034113.

Gao, X., Hoi, S. C., Zhang, Y., Wan, J., & Li, J. (2014). Soml: Sparse online metric learning with application to image retrieval.

Gardner, E. (1989). Optimal basins of attraction in randomly sparse neural network models. *Journal of Physics A: Mathematical and General*, *22*(12), 1969.

Gardner, M. W., & Dorling, S. (1998). Artificial neural networks (the multilayer perceptron)—a review of applications in the atmospheric sciences. *Atmospheric environment*, *32*(14-15), 2627–2636.

Geiger, D., & Heckerman, D. (1996). A characterization of the dirichlet distribution with application to learning bayesian networks. In *Maximum entropy and Bayesian methods*, (pp. 61–68). Springer.

Gelenbe, E. (1990). Stability of the random neural network model. *Neural computation*, *2*(2), 239–247.

Gelenbe, E. (1993). Learning in the recurrent random neural network. *Neural computation*, *5*(1), 154–164.

Gelenbe, E., Mao, Z.-H., & Li, Y.-D. (1999a). Function approximation with spiked random networks. *IEEE Transactions on Neural Networks*, *10*(1), 3–9.

Gelenbe, E., Mao, Z.-H., & Li, Y.-D. (2006). Function approximation by random neural networks with a bounded number of layers. In *Computer System Performance Modeling In Perspective: A Tribute to the Work of Prof Kenneth C Sevcik*, (pp. 35–58). World Scientific.

Gelenbe, E., Mao, Z.-W., & Li, Y.-D. (1999b). Approximation by random networks with bounded number of layers. In *Neural Networks for Signal Processing IX: Proceedings of the 1999 IEEE Signal Processing Society Workshop (Cat. No. 98TH8468)*, (pp. 166–175). IEEE.

Geman, S., & Graffigne, C. (1986). Markov random field image models and their applications to computer vision. In *Proceedings of the international congress of mathematicians*, vol. 1, (p. 2). Berkeley, CA.

Geng, J. (2017). Bayesian models for capturing heterogeneity in discrete data.

Geng, J., & Hu, G. (2020). Mixture of finite mixtures model for basket trial. *arXiv preprint arXiv:2011.04135*.

Georgii, H.-O., & Häggström, O. (1996). Phase transition in continuum potts models. *Communications in mathematical physics*, *181*(2), 507–528.

Ghahramani, Z., Jordan, M., & Adams, R. P. (2010). Tree-structured stick breaking for hierarchical data. *Advances in neural information processing systems*, *23*, 19–27.

Ghasemain, B., Asl, D. T., Pham, B. T., Avand, M., Nguyen, H. D., & Janizadeh, S. (2020). Shallow landslide susceptibility mapping: A comparison between classification and regression tree and reduced error pruning tree algorithms. *Vietnam Journal of Earth Sciences*.

Ghiassi, M., Saidane, H., & Zimbra, D. (2005). A dynamic artificial neural network model for forecasting time series events. *International Journal of Forecasting*, *21*(2), 341–362.

Giaquinta, M., & Modica, G. (2010). *Mathematical analysis: An introduction to functions of several variables*. Springer Science & Business Media.

Gidas, B., & Murua, A. (1995). Classification and clustering of stop consonants via nonparametric transformations and wavelets. In *1995 International Conference on Acoustics, Speech, and Signal Processing*, vol. 1, (pp. 872–875). IEEE.

Gilbert, G. T. (1991). Positive definite matrices and sylvester's criterion. *The American Mathematical Monthly*, *98*(1), 44–46.

Giles, C. L., & Maxwell, T. (1987). Learning, invariance, and generalization in high-order neural networks. *Applied optics*, *26*(23), 4972–4978.

Giles, D. E. (2007). Benford's law and naturally occurring prices in certain ebay auctions. *Applied Economics Letters*, *14*(3), 157–161.

Ginsberg, M. (2012). *Essentials of artificial intelligence*. Newnes.

Giudici, P., & Green, P. (1999). Decomposable graphical Gaussian model determination. *Biometrika*, *86*(4), 785–801.

Glassner, A. (1998). Penrose tiling. *IEEE Computer Graphics and Applications*, *18*(4), 78–86.

Glauber, R. J. (1963). Time-dependent statistics of the ising model. *Journal of mathematical physics*, *4*(2), 294–307.

Glorot, X., Bordes, A., & Bengio, Y. (2011). Deep sparse rectifier neural networks. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, (pp. 315–323).

Godhavari, T., Alamelu, N., & Soundararajan, R. (2005). Cryptography using neural network. In *2005 Annual IEEE India Conference-Indicon*, (pp. 258–261). IEEE.

Goertzel, B., & Pennachin, C. (2007). *Artificial general intelligence*, vol. 2. Springer.

Gold, S., Rangarajan, A., et al. (1996). Softmax to softassign: Neural network algorithms for combinatorial optimization. *Journal of Artificial Neural Networks*, *2*(4), 381–399.

Goldberg, L. A., Jerrum, M., & Paterson, M. (2003). The computational complexity of two-state spin systems. *Random Structures & Algorithms*, *23*(2), 133–154.

Goldenshluger, A., Zeevi, A., et al. (2004). The hough transform estimator. *The Annals of Statistics*, *32*(5), 1908–1932.

Golik, P., Doetsch, P., & Ney, H. (2013). Cross-entropy vs. squared error training: a theoretical and experimental comparison. In *Interspeech*, vol. 13, (pp. 1756–1760).

Gonzalez, R. C. (1987). P. wintz digital image processing. *Addision-Wesley Publishing Company*, (pp. 275–281).

Good, I. J. (1953). The population frequencies of species and the estimation of population parameters. *Biometrika*, *40*(3-4), 237–264.

Good, I. J., et al. (1976). On the application of symmetric dirichlet distributions and their mixtures to contingency tables. *The Annals of Statistics*, *4*(6), 1159–1189.

Goodfellow, I., Bengio, Y., Courville, A., & Bengio, Y. (2016). *Deep learning*, vol. 1. MIT press Cambridge.

Goodman, N. R. (1963). Statistical analysis based on a certain multivariate complex gaussian distribution (an introduction). *The Annals of mathematical statistics*, *34*(1), 152–177.

Gopinath, R., Zheng, Y., Vadvalkar, M., Lin, J., & Lin, L. (2011). Computing with tiles.

Gotmare, A., Keskar, N. S., Xiong, C., & Socher, R. (2018). A closer look at deep learning heuristics: Learning rate restarts, warmup and distillation. *arXiv preprint arXiv:1810.13243*.

Gottardo, R., Besag, J., Stephens, M., & Murua, A. (2006). Probabilistic segmentation and intensity estimation for microarray images. *Biostatistics*, *7*(1), 85–99.

Goutte, C. (1997). Note on free lunches and cross-validation. *Neural Computation*, *9*(6), 1245–1249.

Govaert, G., & Nadif, M. (2003). Clustering with block mixture models. *Pattern Recognition*, *36*(2), 463–473.

Govaert, G., & Nadif, M. (2008). Block clustering with bernoulli mixture models: Comparison of different approaches. *Computational Statistics & Data Analysis*, *52*(6), 3233–3245.

Graner, F., & Glazier, J. A. (1992). Simulation of biological cell sorting using a two-dimensional extended potts model. *Physical review letters*, *69*(13), 2013.

Granter, S. R., Beck, A. H., & Papke Jr, D. J. (2017). Alphago, deep learning, and the future of the human microscopist. *Archives of pathology & laboratory medicine*, *141*(5), 619–621.

Graves, A. (2011). Practical variational inference for neural networks. In J. Shawe-Taylor, R. S. Zemel, P. L. Bartlett, F. Pereira, & K. Q. Weinberger (Eds.) *Advances in Neural Information Processing Systems 24*, (pp. 2348–2356). Curran Associates, Inc.

Graves, A. (2012). Long short-term memory. In *Supervised sequence labelling with recurrent neural networks*, (pp. 37–45). Springer.

Green, P. J. (1995). Reversible jump Markov chain Monte Carlo computation and Bayesian model determination. *Biometrika*, *82*(4), 711–732.

Gregory, S. (2010). Finding overlapping communities in networks by label propagation. *New journal of Physics*, *12*(10), 103018.

Griffin, J. E., & Steel, M. F. (2011). Stick-breaking autoregressive processes. *Journal of econometrics*, *162*(2), 383–396.

Griffin, J. E., & Steel, M. J. (2006). Order-based dependent dirichlet processes. *Journal of the American statistical Association*, *101*(473), 179–194.

Grimm, K. J., Mazza, G. L., & Davoudzadeh, P. (2017). Model selection in finite mixture models: A k-fold cross-validation approach. *Structural Equation Modeling: A Multidisciplinary Journal*, *24*(2), 246–256.

Grimmett, G. (1994). Potts models and random-cluster processes with many-body interactions. *Journal of Statistical Physics*, *75*(1-2), 67–121.

Grimmett, G. (2004). The random-cluster model. In *Probability on discrete structures*, (pp. 73–123). Springer.

Gross, A., & Latecki, L. (1995). Digital geometric invariance and shape representation. In *Proceedings of International Symposium on Computer Vision-ISCV*, (pp. 121–126). IEEE.

Gschwend, D. (2020). Zynqnet: An fpga-accelerated embedded convolutional neural network. *arXiv preprint arXiv:2005.06892*.

Guessab, A., Nouisser, O., & Schmeisser, G. (2006). Multivariate approximation by a combination of modified taylor polynomials. *Journal of Computational and Applied Mathematics*, *196*(1), 162–179.

Guilhoto, L. F. (2018). An overview of artificial neural networks for mathematicians.

Gunning, D. (2017). Explainable artificial intelligence (xai). *Defense Advanced Research Projects Agency (DARPA), Web*, *2*(2).

Guo, F., Wang, X., Fan, K., Broderick, T., & Dunson, D. B. (2016). Boosting variational inference. *arXiv preprint arXiv:1611.05559*.

Guo-qiang, Y., Wan-jin, H., Wen-cai, L., et al. (2004). Linear interpolation method for processing the test data of five-hole probes [j]. *Journal of Engineering for Thermal Energy and Power*, *5*.

Guresen, E., Kayakutlu, G., & Daim, T. U. (2011). Using artificial neural network models in stock market index prediction. *Expert Systems with Applications*, *38*(8), 10389–10397.

Hadke, P. P., & Kale, S. G. (2016). Use of neural networks in cryptography: A review. In *2016 World Conference on Futuristic Trends in Research and Innovation for Social Welfare (Startup Conclave)*, (pp. 1–4). IEEE.

Haff, L. (1979). An identity for the wishart distribution with applications. *Journal of Multivariate Analysis*, *9*(4), 531–544.

Häggkvist, R., Rosengren, A., Andrén, D., Kundrotas, P., Lundow, P. H., & Markström, K. (2004). A monte carlo sampling scheme for the ising model. *Journal of statistical physics*, *114*(1-2), 455–480.

Häggström, O., Jonasson, J., Lyons, R., et al. (2002). Coupling and bernoullicity in random-cluster and potts models. *Bernoulli*, *8*(3), 275–294.

Hahmann, F., Böer, G., Gabriel, E., Meyer, C., & Schramm, H. (2015). A shape consistency measure for improving the generalized hough transform. *Proc. VISAPP*.

Hamerly, G., & Elkan, C. (2004). Learning the k in k-means. In *Advances in neural information processing systems*, (pp. 281–288).

HAMMER, F. G. H. (2017). Taylor expansion, gradient descent and newton's method in machine learning.

Han, J., & Moraga, C. (1995). The influence of the sigmoid function parameters on the speed of backpropagation learning. In *International Workshop on Artificial Neural Networks*, (pp. 195–201). Springer.

Han, S., Pool, J., Tran, J., & Dally, W. (2015). Learning both weights and connections for efficient neural network. In *Advances in neural information processing systems*, (pp. 1135–1143).

Hansen, L. K., Sigurdsson, S., Kolenda, T., Nielsen, F. A., Kjems, U., & Larsen, J. (2000). Modeling text with generalizable gaussian mixtures. In *2000 IEEE International Conference on Acoustics, Speech, and Signal Processing. Proceedings (Cat. No. 00CH37100)*, vol. 6, (pp. 3494–3497). IEEE.

Hao, X., Zhang, G., & Ma, S. (2016). Deep learning. *International Journal of Semantic Computing*, *10*(03), 417–439.

Hara, K., Saito, D., & Shouno, H. (2015). Analysis of function of rectified linear unit used in deep learning. In *2015 International Joint Conference on Neural Networks (IJCNN)*, (pp. 1–8). IEEE.

Harrington, P. (2012). *Machine learning in action*. Manning Publications Co.

Harrison, B. K., & Estabrook, F. B. (1971). Geometric approach to invariance groups and solution of partial differential systems. *Journal of Mathematical Physics*, *12*(4), 653–666.

Hartigan, J. A. (1990). Partition models. *Communications in statistics-Theory and methods*, *19*(8), 2745–2756.

Hasan, T., & Hansen, J. H. (2011). A study on universal background model training in speaker verification. *IEEE Transactions on Audio, Speech, and Language Processing*, *19*(7), 1890–1899.

Hasan, T., Lei, Y., Chandrasekaran, A., & Hansen, J. H. (2010). A novel feature sub-sampling method for efficient universal background model training in speaker verification. In *2010 IEEE International Conference on Acoustics, Speech and Signal Processing*, (pp. 4494–4497). IEEE.

Hassabis, D. (2017). Artificial intelligence: Chess match of the century. *Nature*, *544*(7651), 413–414.

Hassanein, A. S., Mohammad, S., Sameer, M., & Ragab, M. E. (2015). A survey on hough transform, theory, techniques and applications. *arXiv preprint arXiv:1502.02160*.

Hassani, H., Soltanolkotabi, M., & Karbasi, A. (2017). Gradient methods for submodular maximization. In *Advances in Neural Information Processing Systems*, (pp. 5841–5851).

Hassoun, M. H., et al. (1995). *Fundamentals of artificial neural networks*. MIT press.

Hastie, T., Tibshirani, R., & Friedman, J. (2009). Overview of supervised learning. In *The elements of statistical learning*, (pp. 9–41). Springer.

Hatti, M., & Tioursi, M. (2009). Dynamic neural network controller model of pem fuel cell system. *International Journal of Hydrogen Energy*, *34*(11), 5015–5021.

Haugeland, J. (1989). *Artificial intelligence: The very idea*. MIT press.

Haugh, M. (2017). Mcmc and bayesian modeling. *IEOR E4703 Monte-Carlo Simulation, Columbia University*.

Hawkins, D. M., Basak, S. C., & Mills, D. (2003). Assessing model fit by cross-validation. *Journal of chemical information and computer sciences*, *43*(2), 579–586.

Hawkins, D. S., Allen, D. M., & Stromberg, A. J. (2001). Determining the number of components in mixtures of linear models. *Computational Statistics & Data Analysis*, *38*(1), 15–48.

Hayden, T. L., & Wells, J. (1988). Approximation by matrices positive semidefinite on a subspace. *Linear Algebra and its Applications*, *109*, 115–130.

Hayou, S., Doucet, A., & Rousseau, J. (2018). On the selection of initialization and activation function for deep neural networks. *arXiv preprint arXiv:1805.08266*.

Hayou, S., Doucet, A., & Rousseau, J. (2019). On the impact of the activation function on deep neural networks training. *arXiv preprint arXiv:1902.06853*.

He, Y., Ding, Y., Liu, P., Zhu, L., Zhang, H., & Yang, Y. (2020). Learning filter pruning criteria for deep convolutional neural networks acceleration. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, (pp. 2009–2018).

He, Y., Kang, G., Dong, X., Fu, Y., & Yang, Y. (2018). Soft filter pruning for accelerating deep convolutional neural networks. *arXiv preprint arXiv:1808.06866*.

Hecht-Nielsen, R. (1992). Theory of the backpropagation neural network. In *Neural networks for perception*, (pp. 65–93). Elsevier.

Hernandez, H. (2017). Standard maxwell-boltzmann distribution: Definition and properties. *ForsChem Research Reports*, *2*, 2017–2.

Heydemann, M.-C. (1997). Cayley graphs and interconnection networks. In *Graph symmetry*, (pp. 167–224). Springer.

Higham, N. J. (1988). Computing a nearest symmetric positive semidefinite matrix. *Linear algebra and its applications*, *103*, 103–118.

Hinton, G. (2007). How to do backpropagation in a brain. In *Invited talk at the NIPS'2007 deep learning workshop*, vol. 656.

Hinton, G., Srivastava, N., & Swersky, K. (2012a). Lecture 6a overview of mini–batch gradient descent. *Coursera Lecture slides https://class. coursera. org/neuralnets-2012-001/lecture,[Online*.

Hinton, G., Srivastava, N., & Swersky, K. (2012b). Neural networks for machine learning lecture 6a overview of mini-batch gradient descent. *Cited on*, *14*(8).

Hinton, G. E. (2009). Deep belief networks. *Scholarpedia*, *4*(5), 5947.

Hinton, G. E., & Sejnowski, T. J. (1983). Optimal perceptual inference. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, vol. 448. Citeseer.

Hinton, G. E., & van Camp, D. (1993). Keeping the neural networks simple by minimizing the description length of the weights. In *Proceedings of the Sixth Annual Conference on Computational Learning Theory*, COLT '93, (pp. 5–13). New York, NY, USA: ACM.
URL http://doi.acm.org/10.1145/168304.168306

Hirai, S., & Yamanishi, K. (2011). Efficient computation of normalized maximum likelihood coding for gaussian mixtures with its applications to optimal clustering. In *2011 IEEE International Symposium on Information Theory Proceedings*, (pp. 1031–1035). IEEE.

Hirai, S., & Yamanishi, K. (2013). Efficient computation of normalized maximum likelihood codes for gaussian mixture models with its applications to clustering. *IEEE Transactions on Information Theory*, *59*(11), 7718–7727.

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, *9*(8), 1735–1780.

Hochreiter, S., Younger, A. S., & Conwell, P. R. (2001). Learning to learn using gradient descent. In *International Conference on Artificial Neural Networks*, (pp. 87–94). Springer.

Hoi, S. C., Liu, W., & Chang, S.-F. (2010). Semi-supervised distance metric learning for collaborative image retrieval and clustering. *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)*, *6*(3), 1–26.

Hoi, S. C., Liu, W., Lyu, M. R., & Ma, W.-Y. (2006). Learning distance metrics with contextual constraints for image retrieval. In *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, vol. 2, (pp. 2072–2078). IEEE.

Hong, Z.-Q. (1991). Algebraic feature extraction of image for recognition. *Pattern recognition*, *24*(3), 211–219.

Hood, C., & Jones, D. (2003). Chapter four: The extent to which "statistics are signs from god". In *Accident And Design*, (pp. 98–100). Routledge.

Hooke, R. (1979). Getting people to use statistics as god and sir ronald fisher intended. *The American Statistician*, *34*, 39–42.

Hopfield, J. J. (2007). Hopfield network. *Scholarpedia*, *2*(5), 1977.

Hornik, K. (1991). Approximation capabilities of multilayer feedforward networks. *Neural networks*, *4*(2), 251–257.

Hornik, K., Stinchcombe, M., & White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural Networks*, *2*(5), 359 – 366.

Hosoda, C., Tanaka, K., Nariai, T., Honda, M., & Hanakawa, T. (2013). Dynamic neural network reorganization associated with second language vocabulary acquisition: a multimodal imaging study. *Journal of Neuroscience*, *33*(34), 13663–13672.

Houtappel, R., Van Dam, H., & Wigner, E. (1965). The conceptual basis and use of the geometric invariance principles. *Reviews of Modern Physics*, *37*(4), 595.

Howard, A. G. (2013). Some improvements on deep convolutional neural network based image classification. *arXiv preprint arXiv:1312.5402*.

Hsu, F.-H. (2004). *Behind Deep Blue: Building the computer that defeated the world chess champion*. Princeton University Press.

Hu, C.-K. (1987). Exact cluster size distributions and mean cluster sizes for the q-state bond-correlated percolation model. *Journal of Physics A: Mathematical and General*, *20*(18), 6617.

Hu, C. W., Li, H., & Qutub, A. A. (2018a). Shrinkage clustering: a fast and size-constrained clustering algorithm for biomedical applications. *BMC bioinformatics*, *19*(1), 19.

Hu, K., Zhang, Z., Niu, X., Zhang, Y., Cao, C., Xiao, F., & Gao, X. (2018b). Retinal vessel segmentation of color fundus images using multiscale convolutional neural network with an improved cross-entropy loss function. *Neurocomputing*, *309*, 179–191.

Hu, M.-K. (1962). Visual pattern recognition by moment invariants. *IRE transactions on information theory*, *8*(2), 179–187.

Hua, Y., Guo, J., & Zhao, H. (2015). Deep belief networks and deep learning. In *Proceedings of 2015 International Conference on Intelligent Computing and Internet of Things*, (pp. 1–4). IEEE.

Huang, J., & Murphy, K. (2015). Efficient inference in occlusion-aware generative models of images. *arXiv preprint arXiv:1511.06362*.

Huang, Q., Zhou, K., You, S., & Neumann, U. (2018). Learning to prune filters in convolutional neural networks. In *2018 IEEE Winter Conference on Applications of Computer Vision (WACV)*, (pp. 709–718). IEEE.

Huang, T., Peng, H., & Zhang, K. (2017). Model selection for gaussian mixture models. *Statistica Sinica*, (pp. 147–169).

Huber, M. F., & Hanebeck, U. D. (2008). Progressive gaussian mixture reduction. In *2008 11th International Conference on Information Fusion*, (pp. 1–8). IEEE.

Hussain, F., & Jeong, J. (2016). Efficient deep neural network for digital image compression employing rectified linear neurons. *Journal of Sensors*, *2016*.

Huszár, F. (2015). How (not) to train your generative model: Scheduled sampling, likelihood, adversary? *arXiv preprint arXiv:1511.05101*.

Hutter, M. (2008). Introduction to statistical machine learning. *Machine Learning Summer School-MLSS-2008*, (pp. 2–15).

Hwang, Y.-S., & Bang, S.-Y. (1997). An efficient method to construct a radial basis function neural network classifier. *Neural networks*, *10*(8), 1495–1503.

H.-J., M. (1985). Spath, h.: Cluster dissection and analysis: theory, fortran programs, examples. (translator: Johannes goldschmidt.) ellis horwood ltd wiley, chichester 1985. 226 pp. £25. *Biometrical Journal*, *28*(2), 182–182.

Iannario, M. (2010). On the identifiability of a mixture model for ordinal data. *Metron*, *68*(1), 87–94.

Icarte, R. T., Illanes, L., Castro, M. P., Cire, A. A., McIlraith, S. A., & Beck, J. C. (2019). Training binarized neural networks using mip and cp. In *International Conference on Principles and Practice of Constraint Programming*, (pp. 401–417). Springer.

Illingworth, J., & Kittler, J. (1988). A survey of the hough transform. *Computer vision, graphics, and image processing*, *44*(1), 87–116.

Ionescu, C., Vantzos, O., & Sminchisescu, C. (2015). Training deep networks with structured layers by matrix backpropagation. *arXiv preprint arXiv:1509.07838*.

Ishak, S., Kotha, P., & Alecsandru, C. (2003). Optimization of dynamic neural network performance for short-term traffic prediction. *Transportation Research Record*, *1836*(1), 45–56.

Ishwaran, H., & Zarepour, M. (2002). Exact and approximate sum representations for the dirichlet process. *Canadian Journal of Statistics*, *30*(2), 269–283.

Ito, M., Noda, K., Hoshino, Y., & Tani, J. (2006). Dynamic and interactive generation of object handling behaviors by a small humanoid robot using a dynamic neural network model. *Neural Networks*, *19*(3), 323–337.

Ito, Y. (1991). Representation of functions by superpositions of a step or sigmoid function and their applications to neural network theory. *Neural Networks*, *4*(3), 385–394.

Jacobs, R. A. (1988). Increased rates of convergence through learning rate adaptation. *Neural networks*, *1*(4), 295–307.

Jang, J.-S., & Sun, C.-T. (1993). Functional equivalence between radial basis function networks and fuzzy inference systems. *IEEE transactions on Neural Networks*, *4*(1), 156–159.

Janke, W., & Schakel, A. M. (2004). Geometrical vs. fortuin–kasteleyn clusters in the two-dimensional q-state potts model. *Nuclear Physics B*, *700*(1-3), 385–406.

Jerrum, M., & Sinclair, A. (1993). Polynomial-time approximation algorithms for the ising model. *SIAM Journal on computing*, *22*(5), 1087–1116.

Jewbali, A., & Ore, R. T. I. (2009). Finding the nearest positive definite matrix for input to semi-automatic variogram fitting (varfit_lmc). *Centre for Computational Geostatistics*, *11*, 402.

Jiang, M., Liang, Y., Feng, X., Fan, X., Pei, Z., Xue, Y., & Guan, R. (2018a). Text classification based on deep belief network and softmax regression. *Neural Computing and Applications*, *29*(1), 61–70.

Jiang, S. G. (2021). Collatz total stopping times with neural networks. *World Scientific Research Journal*, *7*(1), 296–301.

Jiang, W., & Tanner, M. A. (1999). On the identifiability of mixtures-of-experts. *Neural Networks*, *12*(9), 1253–1258.

Jiang, X., & Adeli, H. (2005). Dynamic wavelet neural network for nonlinear identification of highrise buildings. *Computer-Aided Civil and Infrastructure Engineering*, *20*(5), 316–330.

Jiang, X., Pang, Y., Li, X., Pan, J., & Xie, Y. (2018b). Deep neural networks with elastic rectified linear units for object recognition. *Neurocomputing*, *275*, 1132–1139.

Jin, C., Zhang, Y., Balakrishnan, S., Wainwright, M. J., & Jordan, M. I. (2016). Local maxima in the likelihood of gaussian mixture models: Structural results and algorithmic consequences. In *Advances in neural information processing systems*, (pp. 4116–4124).

Jin, X., Xu, C., Feng, J., Wei, Y., Xiong, J., & Yan, S. (2015). Deep learning with s-shaped rectified linear activation units. *arXiv preprint arXiv:1512.07030*.

Jo, S., Lee, J., Page, G., Quintana, F., Trippa, L., & Müller, P. (2015). Spatial species sampling and product partition models. In *Nonparametric Bayesian Inference in Biostatistics*, (pp. 359–375). Springer.

Joarder, A. H. (2001). Six ways to look at linear interpolation. *International Journal of Mathematical Education in Science and Technology*, *32*(6), 932–937.

Johansson, J., & Pistol, M.-E. (2011). Microcanonical entropy of the infinite-state potts model. *Physics Research International*, *2011*.

Johnson, J. K., Oyen, D., Chertkov, M., & Netrapalli, P. (2015). Learning planar ising models. *arXiv preprint arXiv:1502.00916*.

Johnson, R., & Zhang, T. (2013). Accelerating stochastic gradient descent using predictive variance reduction. In *Advances in neural information processing systems*, (pp. 315–323).

Jordan, C., Livingstone, V., & Barry, D. (2007). Statistical modelling using product partition models. *Statistical Modelling*, *7*(3), 275–295.

Jordan, M. I., & Mitchell, T. M. (2015). Machine learning: Trends, perspectives, and prospects. *Science*, *349*(6245), 255–260.

Jordan, M. I., & Xu, L. (1995). Convergence results for the em approach to mixtures of experts architectures. *Neural networks*, *8*(9), 1409–1431.

Jorjani, H., Klei, L., & Emanuelson, U. (2003). A simple method for weighted bending of genetic (co) variance matrices. *Journal of dairy science*, *86*(2), 677–679.

Joya, G., Atencia, M., & Sandoval, F. (2002). Hopfield neural networks for optimization: study of the different dynamics. *Neurocomputing*, *43*(1-4), 219–237.

Jung, Y., & Hu, J. (2015). A k-fold averaging cross-validation procedure. *Journal of nonparametric statistics*, *27*(2), 167–179.

Kadowaki, T., & Nishimori, H. (1998). Quantum annealing in the transverse ising model. *Physical Review E*, *58*(5), 5355.

Kai, Y., Lei, J., Yuqiang, C., & Wei, X. (2013). Deep learning: yesterday, today, and tomorrow. *Journal of computer Research and Development*, *50*(9), 1799.

Kalanov, T. Z. (2008). Modern analysis of the boltzmann distribution. *Galilean Electrodynamics (be published)*.

Kalchbrenner, N., Danihelka, I., & Graves, A. (2015). Grid long short-term memory. *arXiv preprint arXiv:1507.01526*.

Kamruzzaman, J., Kumagai, Y., & Aziz, S. M. (1997). Character recognition by double backpropagation neural network. In *TENCON'97 Brisbane-Australia. Proceedings of IEEE TENCON'97. IEEE Region 10 Annual Conference. Speech and Image Technologies for Computing and Telecommunications (Cat. No. 97CH36162)*, vol. 1, (pp. 411–414). IEEE.

Kamruzzaman, J., & Syed, A., Mahfuzul (1998). A neural network based character recognition system using double backpropagation. *Malaysian Journal of Computer Science*, *11*(1), 58–64.

Kamwa, I., Grondin, R., Sood, V., Gagnon, C., Nguyen, V. T., & Mereb, J. (1996). Recurrent neural networks for phasor detection and adaptive identification in power system control and protection. *IEEE Transactions on Instrumentation and Measurement*, *45*(2), 657–664.

Kanai, S., Fujiwara, Y., Yamanaka, Y., & Adachi, S. (2018). Sigsoftmax: Reanalysis of the softmax bottleneck. *Advances in Neural Information Processing Systems*, *31*, 286–296.

Kanellopoulos, I., & Wilkinson, G. G. (1997). Strategies and best practice for neural network image classification. *International Journal of Remote Sensing*, *18*(4), 711–725.

Kanter, I. (1988). Potts-glass models of neural networks. *Phys. Rev. A*, *37*, 2739–2742.
URL `https://link.aps.org/doi/10.1103/PhysRevA.37.2739`

Kanter, I., & Kinzel, W. (2003). The theory of neural networks and cryptography. In *The Physics of Communication*, (pp. 631–642). World Scientific.

Karlik, B., & Olgac, A. V. (2011). Performance analysis of various activation functions in generalized mlp architectures of neural networks. *International Journal of Artificial Intelligence and Expert Systems*, *1*(4), 111–122.

Kauppinen, H., Seppanen, T., & Pietikainen, M. (1995). An experimental comparison of autoregressive and fourier-based descriptors in 2d shape classification. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *17*(2), 201–207.

Kawaguchi, K. (2016). Deep learning without poor local minima. In *Advances in neural information processing systems*, (pp. 586–594).

Kay, S. (1983). Some results in linear interpolation theory. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, *31*(3), 746–749.

Kaya, Y., Hong, S., & Dumitras, T. (2019). Shallow-deep networks: Understanding and mitigating network overthinking. In *International Conference on Machine Learning*, (pp. 3301–3310). PMLR.

Kazhdan, M., Chazelle, B., Dobkin, D., Finkelstein, A., & Funkhouser, T. (2002). A reflective symmetry descriptor. In *European Conference on Computer Vision*, (pp. 642–656). Springer.

Kelarev, A. V. (2002). On undirected cayley graphs. *Australasian Journal of Combinatorics*, *25*, 73–78.

Kelleher, J. D. (2019). *Deep learning*. Mit Press.

Kemppainen, A., & Smirnov, S. (2019). Conformal invariance of boundary touching loops of fk ising model. *Communications in Mathematical Physics*, *369*(1), 49–98.

Kepner, J., & Robinett, R. (2019). Radix-net: Structured sparse matrices for deep neural networks. In *2019 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, (pp. 268–274). IEEE.

Keriven, N., & Peyré, G. (2019). Universal invariant and equivariant graph neural networks. *arXiv preprint arXiv:1905.04943*.

Kéry, M. (2018). Identifiability in n-mixture models: A large-scale screening test with bird data. *Ecology*, *99*(2), 281–288.

Khalidov, V., Forbes, F., & Horaud, R. (2011). Conjugate mixture models for clustering multimodal data. *Neural Computation*, *23*(2), 517–557.

Khor, H.-Q., See, J., Liong, S.-T., Phan, R. C., & Lin, W. (2019). Dual-stream shallow networks for facial micro-expression recognition. In *2019 IEEE International Conference on Image Processing (ICIP)*, (pp. 36–40). IEEE.

Khotanzad, A., & Hong, Y. H. (1990). Invariant image recognition by zernike moments. *IEEE Transactions on pattern analysis and machine intelligence*, *12*(5), 489–497.

Kilian, J., & Siegelmann, H. T. (1993). On the power of sigmoid neural networks. In *Proceedings of the sixth annual conference on Computational learning theory*, (pp. 137–143).

Kim, D., & Lindsay, B. G. (2015). Empirical identifiability in finite mixture models. *Annals of the Institute of Statistical Mathematics*, *67*(4), 745–772.

Kim, D., & Seo, B. (2014). Assessment of the number of components in gaussian mixture models in the presence of multiple local maximizers. *Journal of Multivariate Analysis*, *125*, 100–120.

Kim, D. E., & Gofman, M. (2018). Comparison of shallow and deep neural networks for network intrusion detection. In *2018 IEEE 8th Annual Computing and Communication Workshop and Conference (CCWC)*, (pp. 204–208). IEEE.

Kim, P. (2017). Deep learning. In *MATLAB Deep Learning*, (pp. 103–120). Springer.

Kim, S. M., Do, T. T., Oechtering, T. J., & Peters, G. (2015). On the entropy computation of large complex gaussian mixture distributions. *IEEE Transactions on Signal Processing*, *63*(17), 4710–4723.

Kimura, A., & Watanabe, T. (2002). An extension of the generalized hough transform to realize affine-invariant two-dimensional (2d) shape detection. In *Object recognition supported by user interaction for service robots*, vol. 1, (pp. 65–69). IEEE.

Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Kingma, D. P., Salimans, T., & Welling, M. (2015). Variational dropout and the local reparameterization trick. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, & R. Garnett (Eds.) *Advances in Neural Information Processing Systems 28*, (pp. 2575–2583). Curran Associates, Inc.

Kingma, D. P., & Welling, M. (2013). Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*.

Kinzel, W., & Kanter, I. (2002a). Interacting neural networks and cryptography. In *Advances in solid state physics*, (pp. 383–391). Springer.

Kinzel, W., & Kanter, I. (2002b). Neural cryptography. In *Proceedings of the 9th International Conference on Neural Information Processing, 2002. ICONIP'02.*, vol. 3, (pp. 1351–1354). IEEE.

Klimov, A., Mityagin, A., & Shamir, A. (2002). Analysis of neural cryptography. In *International Conference on the Theory and Application of Cryptology and Information Security*, (pp. 288–298). Springer.

Kline, D. M., & Berardi, V. L. (2005). Revisiting squared-error and cross-entropy functions for training neural network classifiers. *Neural Computing & Applications*, *14*(4), 310–318.

Knill, O., & Slavkovsky, E. (2013). Illustrating mathematics using 3d printers. *arXiv preprint arXiv:1306.5599*.

Ko, S. I., Chong, T. T., Ghosh, P., et al. (2015). Dirichlet process hidden markov multiple change-point model. *Bayesian Analysis*, *10*(2), 275–296.

Koch, C., Sultanow, E., & Cox, S. (2020). Divisions by two in collatz sequences: A data science approach.

Kodinariya, T. M., & Makwana, P. R. (2013). Review on determining number of cluster in k-means clustering. *International Journal*, *1*(6), 90–95.

Kokkinos, I., Bronstein, M., & Yuille, A. (2012). Dense scale invariant descriptors for images and surfaces.

Kokkinos, I., & Yuille, A. (2008). Scale invariance without scale selection. In *2008 IEEE Conference on Computer Vision and Pattern Recognition*, (pp. 1–8). IEEE.

Kondo, T. (2006). Revised gmdh-type neural network algorithm with a feedback loop identifying sigmoid function neural network. In *Proceedings of the ISCIE International Symposium on Stochastic Systems Theory and its Applications*, vol. 2006, (pp. 137–142). The ISCIE Symposium on Stochastic Systems Theory and Its Applications.

Kononenko, I., & Kukar, M. (2007). *Machine learning and data mining*. Horwood Publishing.

Kortylewski, A., Liu, Q., Wang, H., Zhang, Z., & Yuille, A. (2020). Combining compositional models and deep networks for robust object classification under occlusion. In *The IEEE Winter Conference on Applications of Computer Vision*, (pp. 1333–1341).

Kossovsky, A. E. (2014). *Benford's Law: Theory, the General Law of Relative Quantities, and Forensic Fraud Detection Applications*, vol. 3. World Scientific.

Kotsiantis, S. (2011). Combining bagging, boosting, rotation forest and random subspace methods. *Artificial intelligence review*, *35*(3), 223–240.

Kozliak, E. I. (2004). Introduction of entropy via the boltzmann distribution in undergraduate physical chemistry: A molecular approach. *Journal of chemical education*, *81*(11), 1595.

Krakar, Z., & Žgela, M. (2009). Application of benford's law in payment systems auditing. *Journal of Information and Organizational Sciences*, *33*(1), 39–51.

Krizhevsky, A., Hinton, G., et al. (2009). Learning multiple layers of features from tiny images.

Kros, J. F., Lin, M., & Brown, M. L. (2006). Effects of the neural network s-sigmoid function on kdd in the presence of imprecise data. *Computers & operations research*, *33*(11), 3136–3149.

Kucukelbir, A., Ranganath, R., Gelman, A., & Blei, D. (2014). Fully automatic variational inference of differentiable probability models. In *NIPS Workshop on Probabilistic Programming*.

Kulis, B., & Jordan, M. I. (2011). Revisiting k-means: New algorithms via bayesian nonparametrics. *arXiv preprint arXiv:1111.0352*.

Kullback, S., & Leibler, R. (1951). On information and sufficiency. *Ann. Math. Stat*, *22*, 79–86.

Kullmann, L., Kertesz, J., & Mantegna, R. (2000). Identification of clusters of companies in stock indices via potts super-paramagnetic transitions. *Physica A: Statistical Mechanics and its Applications*, *287*(3-4), 412–419.

Kumar, A., Irsoy, O., Ondruska, P., Iyyer, M., Bradbury, J., Gulrajani, I., Zhong, V., Paulus, R., & Socher, R. (2016). Ask me anything: Dynamic memory networks for natural language processing. In *International conference on machine learning*, (pp. 1378–1387).

Kuo, C.-C. J. (2016). Understanding convolutional neural networks with a mathematical model. *Journal of Visual Communication and Image Representation*, *41*, 406–413.

Kurenkov, A., Taglic, J., Kulkarni, R., Dominguez-Kuhne, M., Garg, A., Martín-Martín, R., & Savarese, S. (2020). Visuomotor mechanical search: Learning to retrieve target objects in clutter. *arXiv preprint arXiv:2008.06073*.

Kurihara, K., Welling, M., & Teh, Y. W. (2007). Collapsed variational dirichlet process mixture models. In *IJCAI*, vol. 7, (pp. 2796–2801).

Kurita, T. (1991). An efficient agglomerative clustering algorithm using a heap. *Pattern Recognition*, *24*(3), 205–209.

Lakshmivarahan, S., Jwo, J.-S., & Dhall, S. K. (1993). Symmetry in interconnection networks based on cayley graphs of permutation groups: A survey. *Parallel Computing*, *19*(4), 361–407.

Landau, L. D., & Evgeny, L. (1980). *Course of theoretical physics: statistical physics*. Elsevier Butterworth Heinemann.

Lange, S., Gabel, T., & Riedmiller, M. (2012). Batch reinforcement learning. In *Reinforcement learning*, (pp. 45–73). Springer.

Larsson, G., Maire, M., & Shakhnarovich, G. (2016). Fractalnet: Ultra-deep neural networks without residuals. *arXiv preprint arXiv:1605.07648*.

Lartillot, N., & Philippe, H. (2004). A bayesian mixture model for across-site heterogeneities in the amino-acid replacement process. *Molecular biology and evolution*, *21*(6), 1095–1109.

Laurent, T., & Brecht, J. (2018). Deep linear networks with arbitrary loss: All local minima are global. In *International conference on machine learning*, (pp. 2902–2907).

Lauritzen, S., Sadeghi, K., et al. (2018). Unifying markov properties for graphical models. *The Annals of Statistics*, *46*(5), 2251–2278.

Lauritzen, S. L., & Richardson, T. S. (2002). Chain graph models and their causal interpretations. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, *64*(3), 321–348.

Lauzon, F. Q. (2012). An introduction to deep learning. In *2012 11th International Conference on Information Science, Signal Processing and their Applications (ISSPA)*, (pp. 1438–1439). IEEE.

Le, Q. V., Jaitly, N., & Hinton, G. E. (2015). A simple way to initialize recurrent networks of rectified linear units. *arXiv preprint arXiv:1504.00941*.

Le Cam, L. M. (1986). *On the Bernstein-von Mises theorem*. Department of Statistics, University of California.

LeCun, Y., Bengio, Y., & Hinton, G. (2015a). Deep learning. *nature*, *521*(7553), 436–444.

LeCun, Y., Bengio, Y., et al. (1995). Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, *3361*(10), 1995.

LeCun, Y., Denker, J. S., & Solla, S. A. (1990). Optimal brain damage. In *Advances in neural information processing systems*, (pp. 598–605).

LeCun, Y., Touresky, D., Hinton, G., & Sejnowski, T. (1988). A theoretical framework for backpropagation. In *Proceedings of the 1988 connectionist models summer school*, vol. 1, (pp. 21–28). CMU, Pittsburgh, Pa: Morgan Kaufmann.

LeCun, Y., et al. (2015b). Lenet-5, convolutional neural networks. *URL: http://yann. lecun. com/exdb/lenet*, *20*(5), 14.

Lee, J., & MacEachern, S. N. (2020). A new proof of the stick-breaking representation of dirichlet processes. *Journal of the Korean Statistical Society*, (pp. 1–6).

Lee, J., Shridhar, K., Hayashi, H., Iwana, B. K., Kang, S., & Uchida, S. (2019). Probact: A probabilistic activation function for deep neural networks. *arXiv preprint arXiv:1905.10761*.

Lee, J. D., Simchowitz, M., Jordan, M. I., & Recht, B. (2016). Gradient descent only converges to minimizers. In *Conference on learning theory*, (pp. 1246–1257).

Lee, J.-E., Jin, R., & Jain, A. K. (2008). Rank-based distance metric learning: An application to image retrieval. In *2008 IEEE Conference on Computer Vision and Pattern Recognition*, (pp. 1–8). IEEE.

Lehr, D., & Ohm, P. (2017). Playing with the data: what legal scholars should learn about machine learning. *UCDL Rev.*, *51*, 653.

Lemke, W. (2006). *Term structure modeling and estimation in a state space framework*, vol. 565. Springer Science & Business Media.

Leray, P. (2000). Quelques types de réseaux de neurones la rétropropagation. *Fond de cours*.

Leung, H., Lo, T., & Wang, S. (2001). Prediction of noisy chaotic time series using an optimal radial basis function neural network. *IEEE Transactions on Neural Networks*, *12*(5), 1163–1172.

Leven, S. J., & Levine, D. S. (1996). Multiattribute decision making in context: A dynamic neural network methodology. *Cognitive Science*, *20*(2), 271–299.

Li, D., Chen, X., Zhang, Z., & Huang, K. (2017). Learning deep context-aware features over body and latent parts for person re-identification. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, (pp. 384–393).

Li, E., & Li, H. (2017). Reflection invariant and symmetry detection. *arXiv preprint arXiv:1705.10768*.

Li, H., Kadav, A., Durdanovic, I., Samet, H., & Graf, H. P. (2016). Pruning filters for efficient convnets. *arXiv preprint arXiv:1608.08710*.

Li, J. (2011). *Potts model clustering for discovering patterns of epigenetic marks*. Ph.D. thesis, Rutgers University-Graduate School-New Brunswick.

Li, J., & Murua, A. (1999). A 2d extended hmm for speech recognition. In *1999 IEEE International Conference on Acoustics, Speech, and Signal Processing. Proceedings. ICASSP99 (Cat. No. 99CH36258)*, vol. 1, (pp. 349–352). IEEE.

Li, J., Xing, F., Liu, Y., & Liu, Z. (2020). Backward-link computational imaging using batch learning networks. *Neural Computing and Applications*, (pp. 1–13).

Li, J. F., & Lowengrub, J. (2014). The effects of cell compressibility, motility and contact inhibition on the growth of tumor cell clusters using the cellular potts model. *Journal of theoretical biology*, *343*, 79–91.

Li, M., Zhang, T., Chen, Y., & Smola, A. J. (2014a). Efficient mini-batch training for stochastic optimization. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, (pp. 661–670).

Li, Q., Cai, W., Wang, X., Zhou, Y., Feng, D. D., & Chen, M. (2014b). Medical image classification with convolutional neural network. In *2014 13th International Conference on Control Automation Robotics & Vision (ICARCV)*, (pp. 844–848). IEEE.

Li, Q. J. (1999). *Estimation of mixture models*. Yale University.

Li, S. Z. (1994). Markov random field models in computer vision. In *European conference on computer vision*, (pp. 361–370). Springer.

Li, T., Choi, M., Fu, K., & Lin, L. (2019a). Music sequence prediction with mixture hidden markov models. In *2019 IEEE International Conference on Big Data (Big Data)*, (pp. 6128–6132). IEEE.

Li, X., Chang, D., Tian, T., & Cao, J. (2019b). Large-margin regularized softmax cross-entropy loss. *IEEE Access*, *7*, 19572–19578.

Li, X., & Orabona, F. (2019). On the convergence of stochastic gradient descent with adaptive stepsizes. In *The 22nd International Conference on Artificial Intelligence and Statistics*, (pp. 983–992). PMLR.

Li, X., Yu, L., Chang, D., Ma, Z., & Cao, J. (2019c). Dual cross-entropy loss for small-sample fine-grained vehicle classification. *IEEE Transactions on Vehicular Technology*, *68*(5), 4204–4212.

Li, Y., & Liang, Y. (2018). Learning overparameterized neural networks via stochastic gradient descent on structured data. *Advances in Neural Information Processing Systems*, *31*, 8157–8166.

Li, Z., Cai, X., Liu, Y., & Zhu, B. (2019d). A novel gaussian–bernoulli based convolutional deep belief networks for image feature extraction. *Neural Processing Letters*, *49*(1), 305–319.

Li, Z.-H., & Hu, W.-Z. (2017). A high-precision digital integrator based on the romberg algorithm. *Review of Scientific Instruments*, *88*(4), 045111.

Liang, N.-Y., Huang, G.-B., Saratchandran, P., & Sundararajan, N. (2006). A fast and accurate online sequential learning algorithm for feedforward networks. *IEEE Transactions on neural networks*, *17*(6), 1411–1423.

Liang, S., Sun, R., Lee, J. D., & Srikant, R. (2018). Adding one neuron can eliminate all bad local minima. In *Advances in Neural Information Processing Systems*, (pp. 4350–4360).

Liang, X., Chen, H., & Lozano, J. A. (2014). A boltzmann-based estimation of distribution algorithm for a general resource scheduling model. *IEEE Transactions on Evolutionary Computation*, *19*(6), 793–806.

Liang, X., Wang, X., Lei, Z., Liao, S., & Li, S. Z. (2017). Soft-margin softmax for deep classification. In *International Conference on Neural Information Processing*, (pp. 413–421). Springer.

Liao, K., Liu, G., & Hui, Y. (2013). An improvement to the sift descriptor for image representation and matching. *Pattern Recognition Letters*, *34*(11), 1211–1220.

Likas, A., Vlassis, N., & Verbeek, J. J. (2003). The global k-means clustering algorithm. *Pattern recognition*, *36*(2), 451–461.

Lin, C.-W., & Wang, J.-S. (2008). A digital circuit design of hyperbolic tangent sigmoid function for neural networks. In *2008 IEEE International Symposium on Circuits and Systems*, (pp. 856–859). IEEE.

Lin, J. (2016). *On the dirichlet distribution*. Ph.D. thesis, Master's thesis, Department of Mathematics and Statistics, Queens University ....

Lin, J., & Zhou, D.-X. (2017). Online learning algorithms can converge comparably fast as batch learning. *IEEE transactions on neural networks and learning systems*, *29*(6), 2367–2378.

Ling, H., & Jacobs, D. W. (2005). Deformation invariant image matching. In *Tenth IEEE International Conference on Computer Vision (ICCV'05) Volume 1*, vol. 2, (pp. 1466–1473). IEEE.

Ling, H., & Jacobs, D. W. (2007). Shape classification using the inner-distance. *IEEE transactions on pattern analysis and machine intelligence*, *29*(2), 286–299.

Lison, P. (2015). An introduction to machine learning. *Language Technology Group (LTG)*, *1*(35).

Liu, F., & Perez, J. (2017). Gated end-to-end memory networks. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*, (pp. 1–10).

Liu, F.-H., Gao, Y.-Q., & Li, B.-C. (2014). Comparing two-boltzmann distribution and tsallis statistics of particle transverse momentums in collisions at lhc energies. *The European Physical Journal A*, *50*(8), 123.

Liu, Q., Zhao, L. M., & Zhang, L. J. (2013). Image feature extraction of moment of inertia based on otsu threshold segmentation. In *Advanced Materials Research*, vol. 756, (pp. 3157–3161). Trans Tech Publ.

Liu, S. (2020). Learning sparse neural networks for better generalization. In *29th International Joint Conference on Artificial Intelligence-17th Pacific Rim International Conference on Artificial Intelligence.*.

Liu, S., Van der Lee, T., Yaman, A., Atashgahi, Z., Ferraro, D., Sokar, G., Pechenizkiy, M., & Mocanu, D. C. (2020). Topological insights in sparse neural networks. *arXiv preprint arXiv:2006.14085*.

Liu, W., Wen, Y., Yu, Z., & Yang, M. (2016). Large-margin softmax loss for convolutional neural networks. In *ICML*, vol. 2, (p. 7).

Liu, Y. (2014). Random forest algorithm in big data environment. *Computer Modelling & New Technologies*, *18*(12A), 147–151.

Liu, Y., Wang, Y., & Zhang, J. (2012). New machine learning algorithm: Random forest. In *International Conference on Information Computing and Applications*, (pp. 246–252). Springer.

Lo, Y., Mendell, N. R., & Rubin, D. B. (2001). Testing the number of components in a normal mixture. *Biometrika*, *88*(3), 767–778.

Loschi, R., & Cruz, F. (2002). An analysis of the influence of some prior specifications in the identification of change points via product partition model. *Computational Statistics & Data Analysis*, *39*(4), 477–501.

Loschi, R. H., & Cruz, F. R. (2005). Extension to the product partition model: computing the probability of a change. *Computational Statistics & Data Analysis*, *48*(2), 255–268.

Lou, L., Zhang, F.-M., Xu, C., Li, F., & Xue, M.-G. (2008). Automatic registration of aerial image series using geometric invariance. In *2008 IEEE International Conference on Automation and Logistics*, (pp. 1198–1203). IEEE.

Louizos, C., & Welling, M. (2016). Structured and efficient variational deep learning with matrix gaussian posteriors. In *International Conference on Machine Learning*, (pp. 1708–1716).

Louizos, C., Welling, M., & Kingma, D. P. (2017). Learning sparse neural networks through l_0 regularization. *arXiv preprint arXiv:1712.01312*.

Lowe, D. (1985). Perceptual organization and vi-sual recognition", kluwer academic publishers, boston.

Lu, F., & Xia, S. (2007). Microscopic image mosaicing algorithm based on normalized moment of inertia. In *International Conference on Intelligent Computing*, (pp. 1010–1017). Springer.

Luo, J.-H., Wu, J., & Lin, W. (2017). Thinet: A filter level pruning method for deep neural network compression. In *Proceedings of the IEEE international conference on computer vision*, (pp. 5058–5066).

Lydia, A., & Francis, S. (2019). Adagrad—an optimizer for stochastic gradient descent. *Int. J. Inf. Comput. Sci.*, *6*(5).

Ma, C., Lv, X., & Ao, J. (2019). Normalized moment of inertia-based detection algorithm for copy-paste image tampering. *International Journal of Pattern Recognition and Artificial Intelligence*, *33*(06), 1954023.

Ma, J., Xu, L., & Jordan, M. I. (2000). Asymptotic convergence rate of the em algorithm for gaussian mixtures. *Neural Computation*, *12*(12), 2881–2907.

Machta, J., Choi, Y., Lucke, A., Schweizer, T., & Chayes, L. (1996). Invaded cluster algorithm for potts models. *Physical Review E*, *54*(2), 1332.

MacKay, D. J. (2003). *Information theory, inference and learning algorithms*. Cambridge university press.

Mackenzie, A. (2015). The production of prediction: What does machine learning want? *European Journal of Cultural Studies*, *18*(4-5), 429–445.

Madaan, D., Shin, J., & Hwang, S. J. (2019). Adversarial neural pruning with latent vulnerability suppression. *arXiv preprint arXiv:1908.04355*.

Maehara, T., & NT, H. (2019). A simple proof of the universality of invariant/equivariant graph neural networks. *arXiv preprint arXiv:1910.03802*.

Maeland, E. (1988). On the comparison of interpolation methods. *IEEE transactions on medical imaging*, *7*(3), 213–217.

Magidson, J., & Vermunt, J. (2002). Latent class models for clustering: A comparison with k-means. *Canadian Journal of Marketing Research*, *20*(1), 36–43.

Magoulas, G. D., Vrahatis, M. N., & Androulakis, G. S. (1999). Improving the convergence of the backpropagation algorithm using learning rate adaptation methods. *Neural Computation*, *11*(7), 1769–1796.

Maguolo, G., Nanni, L., & Ghidoni, S. (2019). Ensemble of convolutional neural networks trained with different activation functions. *arXiv preprint arXiv:1905.02473*.

Mahdavi, A., & Kundu, D. (2017). A new method for generating distributions with an application to exponential distribution. *Communications in Statistics-Theory and Methods*, *46*(13), 6543–6557.

Mairal, J., Koniusz, P., Harchaoui, Z., & Schmid, C. (2014). Convolutional kernel networks. *Advances in neural information processing systems*, *27*, 2627–2635.

Malach, E., & Shalev-Shwartz, S. (2019). Is deeper better only when shallow is good? In *Advances in Neural Information Processing Systems*, (pp. 6429–6438).

Malhotra, P., Vig, L., Shroff, G., & Agarwal, P. (2015). Long short term memory networks for anomaly detection in time series. In *Proceedings*, vol. 89, (pp. 89–94). Presses universitaires de Louvain.

Mao, H., Han, S., Pool, J., Li, W., Liu, X., Wang, Y., & Dally, W. J. (2017). Exploring the regularity of sparse structure in convolutional neural networks. *arXiv preprint arXiv:1705.08922*.

Mao, H. H., Shin, T., & Cottrell, G. (2018). Deepj: Style-specific music generation. In *2018 IEEE 12th International Conference on Semantic Computing (ICSC)*, (pp. 377–382). IEEE.

Marchisio, G. B., Koperski, K., Liang, J., Murua, A., Nguyen, T., Tusk, C., Dhillon, N. S., & Pochman, L. (2008). Method and system for enhanced data searching. US Patent 7,398,201.

Marée, S. (2012). *Correcting non positive definite correlation matrices*. BSc Thesis Applied Mathematics, TU Delft.

Maron, H., Ben-Hamu, H., Shamir, N., & Lipman, Y. (2018). Invariant and equivariant graph networks. *arXiv preprint arXiv:1812.09902*.

Maron, H., Fetaya, E., Segol, N., & Lipman, Y. (2019). On the universality of invariant networks. In *International conference on machine learning*, (pp. 4363–4371). PMLR.

Marquez, L., Hill, T., Worthley, R., & Remus, W. (1991). Neural network models as an alternative to regression. In *System Sciences, 1991. Proceedings of the Twenty-Fourth Annual Hawaii International Conference on*, vol. 4, (pp. 129–135). IEEE.

Marshall, A. W., & Olkin, I. (1967). A multivariate exponential distribution. *Journal of the American Statistical Association*, *62*(317), 30–44.

Marshall, S. (1989). Review of shape coding techniques. *Image and vision computing*, *7*(4), 281–294.

Marsland, S. (2015). *Machine learning: an algorithmic perspective*. CRC press.

Marsland, S., McLachlan, R. I., et al. (2016). Möbius invariants of shapes and images. *SIGMA. Symmetry, Integrability and Geometry: Methods and Applications*, *12*, 080.

Martinelli, F., Olivieri, E., & Scoppola, E. (1990). Rigorous analysis of low-temperature stochastic ising models: metastability and exponential approach to equilibrium. *EPL (Europhysics Letters)*, *12*(3), 223.

Martinez, M., & Stiefelhagen, R. (2018). Taming the cross entropy loss. In *German Conference on Pattern Recognition*, (pp. 628–637). Springer.

Martínez-Muñoz, G., Hernández-Lobato, D., & Suárez, A. (2008). An analysis of ensemble pruning techniques based on ordered aggregation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *31*(2), 245–259.

Masci, J., Migliore, D., Bronstein, M. M., & Schmidhuber, J. (2014). Descriptor learning for omnidirectional image matching. In *Registration and Recognition in Images and Videos*, (pp. 49–62). Springer.

Masters, D., & Luschi, C. (2018). Revisiting small batch training for deep neural networks. *arXiv preprint arXiv:1804.07612*.

Mathias, A. C., & Rech, P. C. (2012). Hopfield neural network: The hyperbolic tangent and the piecewise-linear activation functions. *Neural Networks*, *34*, 42–45.

Maweheb, S., Malek, S., & Faouzi, G. (2016). Geometric invariance in digital imaging for the preservation of cultural heritage in tunisia. *Digital Applications in Archaeology and Cultural Heritage*, *3*(4), 99–107.

May, T., Van de Par, S., & Kohlrausch, A. (2011). Noise-robust speaker recognition combining missing data techniques and universal background modeling. *IEEE Transactions on Audio, Speech, and Language Processing*, *20*(1), 108–121.

Mazumder, R., & Hastie, T. (2015). The graphical lasso: new insights and alternatives. *Electronic Journal of Statisitcs*, *6*, 2125–2149.

Mazza-Anthony, C. A. (2019). *Structured Sparsity and Precision Matrix Estimation*. Ph.D. thesis, McGill University.

McCarthy, J. (1987). Generality in artificial intelligence. *Communications of the ACM*, *30*(12), 1030–1035.

McCarthy, J. (1998). What is artificial intelligence?

McCoy, B. M., & Wu, T. T. (2014). *The two-dimensional Ising model*. Courier Corporation.

McCullagh, P. (2011). Random permutations and partition models.

McCullagh, P. (2017). Robustifying concurrent.futures with loky.
URL `https://tommoral.github.io/talks/pyparis17/#41`

McDonnell, M. D., Tissera, M. D., Vladusich, T., van Schaik, A., & Tapson, J. (2015). Fast, simple and accurate handwritten digit classification by training shallow neural network classifiers with the 'extreme learning machine'algorithm. *PloS one*, *10*(8), e0134254.

McDowell, S. A. (1999). A simple derivation of the boltzmann distribution. *Journal of chemical education*, *76*(10), 1393.

McLachlan, G. J., & Basford, K. E. (1988). *Mixture models: Inference and applications to clustering*, vol. 38. M. Dekker New York.

McLachlan, G. J., & Peel, D. (2004). *Finite mixture models*. John Wiley & Sons.

McLachlan, G. J., Peel, D., Basford, K. E., & Adams, P. (1999). The emmix software for the fitting of mixtures of normal and t-components. *Journal of Statistical Software*, *4*(2).

McLachlan, G. J., & Rathnayake, S. (2014). On the number of components in a gaussian mixture model. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, *4*(5), 341–355.

Mebane, W. R. (2011). Comment on "benford's law and the detection of election fraud". *Political Analysis*, *19*(3), 269–272.

Mebane, W. R., & Kalinin, K. (2009). Comparative election fraud detection. In *APSA 2009 Toronto Meeting Paper*.

Mebane Jr, W. R. (2006). Election forensics: Vote counts and benford's law. In *Summer Meeting of the Political Methodology Society, UC-Davis, July*, (pp. 20–22).

Medsker, L., & Jain, L. C. (1999). *Recurrent neural networks: design and applications*. CRC press.

Medsker, L. R., & Jain, L. (2001). Recurrent neural networks. *Design and Applications*, *5*.

Medvedovic, M., Yeung, K. Y., & Bumgarner, R. E. (2004). Bayesian mixture model based clustering of replicated microarray data. *Bioinformatics*, *20*(8), 1222–1232.

Mei, L., Guo, X., & Yin, W. (2018). Learning geometric invariance features and discrimination representation for image classification via spatial transform network and xgboost modeling. In *Proceedings of the 7th International Conference on Informatics, Environment, Energy and Applications*, (pp. 222–226).

Meinicke, P., & Ritter, H. (2001). Resolution-based complexity control for gaussian mixture models. *Neural computation*, *13*(2), 453–475.

Melnykov, V., Maitra, R., et al. (2010). Finite mixture models and model-based clustering. *Statistics Surveys*, *4*, 80–116.

Melnykov, V., & Melnykov, I. (2012). Initializing the em algorithm in gaussian mixture models with an unknown number of components. *Computational Statistics & Data Analysis*, *56*(6), 1381–1395.

Memisevic, R., Zach, C., Pollefeys, M., & Hinton, G. E. (2010). Gated softmax classification. *Advances in neural information processing systems*, *23*, 1603–1611.

Menon, A., Mehrotra, K., Mohan, C. K., & Ranka, S. (1996). Characterization of a class of sigmoid functions with applications to neural networks. *Neural Networks*, *9*(5), 819–835.

Mikolov, T., Kombrink, S., Burget, L., Černockỳ, J., & Khudanpur, S. (2011). Extensions of recurrent neural network language model. In *2011 IEEE international conference on acoustics, speech and signal processing (ICASSP)*, (pp. 5528–5531). IEEE.

Miller, J. W., & Harrison, M. T. (2012). Posterior consistency for the number of components in a finite mixture. In *NIPS Workshop on Modern Nonparametric Machine Learning*, (pp. 1–5).

Miller, J. W., & Harrison, M. T. (2018). Mixture models with a prior on the number of components. *Journal of the American Statistical Association*, *113*(521), 340–356.

Miller, S. J. (2015). *Benford's Law*. Princeton University Press.

Mingqiang, Y., Kidiyo, K., & Joseph, R. (2011). Chord context algorithm for shape feature extraction. *Object Recognition*, (p. 65).

Minsky, M. (1961). Steps toward artificial intelligence. *Proceedings of the IRE*, *49*(1), 8–30.

Mitchell, R., Michalski, J., & Carbonell, T. (2013). *An artificial intelligence approach*. Springer.

Mitchell, T. M. (1997). Does machine learning really work? *AI magazine*, *18*(3), 11–11.

Mitchell, T. M., Carbonell, J. G., & Michalski, R. S. (1986). *Machine learning: a guide to current research*, vol. 12. Springer Science & Business Media.

Mnih, A., & Gregor, K. (2014). Neural variational inference and learning in belief networks. *arXiv preprint arXiv:1402.0030*.

Moerchen, F., Mierswa, I., & Ultsch, A. (2006). Understandable models of music collections based on exhaustive feature generation with temporal statistics. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, (pp. 882–891).

Mohamed, S., Rosca, M., Figurnov, M., & Mnih, A. (2019). Monte carlo gradient estimation in machine learning. *arXiv preprint arXiv:1906.10652*.

Mohammed, M., Khan, M. B., & Bashier, E. B. M. (2016). *Machine learning: algorithms and applications*. Crc Press.

Mohri, M., Rostamizadeh, A., & Talwalkar, A. (2018). *Foundations of machine learning*. MIT press.

Molchanov, P., Tyree, S., Karras, T., Aila, T., & Kautz, J. (2016). Pruning convolutional neural networks for resource efficient inference. *arXiv preprint arXiv:1611.06440*.

Moon, T., Erk, K., & Baldridge, J. (2010). Crouching dirichlet, hidden markov model: Unsupervised pos tagging with context local tag generation. In *Proceedings of the 2010 conference on empirical methods in natural language processing*, (pp. 196–206).

Morcel, R., Hajj, H., Saghir, M. A., Akkary, H., Artail, H., Khanna, R., & Keshavamurthy, A. (2019). Feathernet: An accelerated convolutional neural network design for resource-constrained fpgas. *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, *12*(2), 1–27.

Mordjaoui, M., Haddad, S., Medoued, A., & Laouafi, A. (2017). Electric load forecasting by using dynamic neural network. *International journal of hydrogen energy*, *42*(28), 17655–17663.

Morrison, G. S. (2011). A comparison of procedures for the calculation of forensic likelihood ratios from acoustic–phonetic data: Multivariate kernel density (mvkd) versus gaussian mixture model–universal background model (gmm–ubm). *Speech Communication*, *53*(2), 242–256.

Moser, G., Lee, S. H., Hayes, B. J., Goddard, M. E., Wray, N. R., & Visscher, P. M. (2015). Simultaneous discovery, estimation and prediction analysis of complex traits using a bayesian mixture model. *PLoS Genet*, *11*(4), e1004969.

Mühlenbein, H., Mahnig, T., & Rodriguez, A. O. (1999). Schemata, distributions and graphical models in evolutionary optimization. *Journal of Heuristics*, *5*(2), 215–247.

Mukhopadhyay, P., & Chaudhuri, B. B. (2015). A survey of hough transform. *Pattern Recognition*, *48*(3), 993–1010.

Mulder, J., & Fox, J.-P. (2013). Bayesian tests on components of the compound symmetry covariance matrix. *Statistics and Computing*, *23*(1), 109–122.

Müller, P., & Quintana, F. (2010a). Random partition models with regression on covariates. *Journal of statistical planning and inference*, *140*(10), 2801–2808.

Müller, P., & Quintana, F. (2010b). Random partition models with regression on covariates. *Journal of Statistical Planning and Inference*, *140*(10).

Müller, P., Quintana, F., & Rosner, G. (2008). Bayesian clustering with regression. Tech. rep., Working paper series - European Central Bank.

Müller, P., Quintana, F., & Rosner, G. L. (2011a). A product partition model with regression on covariates. *Journal of Computational and Graphical Statistics*, *20*(1), 260–278.

Müller, P., Quintana, F. A., & Rosner, G. L. (2011b). A Product Partition Model with Regression on Covariates. *Journal of Computational and Graphical Statistics*, *20*(1), 260–278.

Müller, P., Rodriguez, A., et al. (2013). Random partition models. In *Nonparametric Bayesian Inference*, (pp. 87–92). IMS and ASA.

Mundy, J. L. (2006). Object recognition in the geometric era: A retrospective. In *Toward category-level object recognition*, (pp. 3–28). Springer.

Mundy, J. L., Zisserman, A., et al. (1992). *Geometric invariance in computer vision*, vol. 92. MIT press Cambridge, MA.

Munson, M. A., & Caruana, R. (2009). On feature selection, bias-variance, and bagging. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, (pp. 144–159). Springer.

Murata, N. (1998). A statistical study of on-line learning. *Online Learning and Neural Networks. Cambridge University Press, Cambridge, UK*, (pp. 63–92).

Murdoch, W. J., Singh, C., Kumbier, K., Abbasi-Asl, R., & Yu, B. (2019). Interpretable machine learning: definitions, methods, and applications. *arXiv preprint arXiv:1901.04592*.

Murtagh, F. (1991). Multilayer perceptrons for classification and regression. *Neurocomputing*, *2*(5-6), 183–197.

Murua, A., & Maitra, R. (2017). Fast spatial inference in the homogeneous ising model. *arXiv preprint arXiv:1712.02195*.

Murua, A., & Quintana, F. A. (2017a). Sem iparametric bayesian regression via potts model. *Journal of Computational and Graphical Statistics*, *26*(2), 265–274.
URL https://doi.org/10.1080/10618600.2016.1172015

Murua, A., & Quintana, F. A. (2017b). Semiparametric bayesian regression via potts model. *Journal of Computational and Graphical Statistics*, *26*(2), 265–274.

Murua, A., & Quintana, F. A. (2017c). Semiparametric bayesian regression via potts model. *Journal of Computational and Graphical Statistics*, *26*(2), 265–274.
URL https://doi.org/10.1080/10618600.2016.1172015

Murua, A., Ramakrishnan, R., Li, X., Yang, R. H., & Nia, V. P. (2020). Tensor train decompositions on recurrent networks. *arXiv preprint arXiv:2006.05442*.

Murua, A., Stanberry, L., & Stuetzle, W. (2008a). On potts model clustering, kernel k-means and density estimation. *Journal of Computational and Graphical Statistics*, *17*(3), 629–658.

Murua, A., Stanberry, L., & Stuetzle, W. (2008b). On potts model clustering, kernel k-means, and density estimation. *Journal of Computational and Graphical Statistics*, *17*(3), 629–658.

Murua, A., Stanberry, L., & Stuetzle, W. (2008c). On potts model clustering, kernel k means and density estimation. *Journal of Computational and Graphical Statistics*, *17*, 629–658.

Murua, A., & Wicker, N. (2014a). The conditional-potts clustering model. *Journal of Computational and Graphical Statistics*, *23*(3), 717–739.

Murua, A., & Wicker, N. (2014b). The conditional-potts clustering model. *Journal of Computational and Graphical Statistics*, *23*(3), 717–739.
URL https://doi.org/10.1080/10618600.2013.837828

Murua, A., & Wicker, N. (2015). Kernel-based mixture models for classification. *Computational Statistics*, *30*(2), 317–344.

Murua, A., & Wicker, N. (2020). Fast approximate complete-data k-nearest-neighbor estimation. *Austrian Journal of Statistics*, *49*(2), 18–30.

Nadikattu, R. R. (2018). Fundamental applications of machine learning across the globe. *International Journal of Creative Research Thoughts (IJCRT), ISSN*.

Nagpal, G., Uddin, M., & Kaur, A. (2013). Estimating project development effort using clustered regression approach. In *Computer Science & Information Technology*, vol. 3, (pp. 493–507).

Naik, P. A., Shi, P., & Tsai, C.-L. (2007). Extending the akaike information criterion to mixture regression models. *Journal of the American Statistical Association*, *102*(477), 244–254.

Nair, P., & Saunders Jr, A. (1996). Hough transform based ellipse detection algorithm. *Pattern Recognition Letters*, *17*(7), 777–784.

Najafabadi, M. M., Villanustre, F., Khoshgoftaar, T. M., Seliya, N., Wald, R., & Muharemagic, E. (2015). Deep learning applications and challenges in big data analytics. *Journal of Big Data*, *2*.

Najar, F., Bourouis, S., Bouguila, N., & Belghith, S. (2019). Unsupervised learning of finite full covariance multivariate generalized gaussian mixture models for human activity recognition. *Multimedia Tools and Applications*, *78*(13), 18669–18691.

Namin, A. H., Leboeuf, K., Muscedere, R., Wu, H., & Ahmadi, M. (2009). Efficient hardware implementation of the hyperbolic tangent sigmoid function. In *2009 IEEE International Symposium on Circuits and Systems*, (pp. 2117–2120). IEEE.

Nasfi, R., Amayri, M., & Bouguila, N. (2020). A novel approach for modeling positive vectors with inverted dirichlet-based hidden markov models. *Knowledge-Based Systems*, *192*, 105335.

Nash, L. K. (1982). On the boltzmann distribution law. *Journal of Chemical Education*, *59*(10), 824.

Nasr, G. E., Badr, E., & Joun, C. (2002). Cross entropy error function in neural networks: Forecasting gasoline demand. In *FLAIRS conference*, (pp. 381–384).

Nasserinejad, K., van Rosmalen, J., de Kort, W., & Lesaffre, E. (2017). Comparison of criteria for choosing the number of classes in bayesian finite mixture models. *PloS one*, *12*(1), e0168838.

Natarajan, B. K. (2014). *Machine learning: a theoretical approach*. Elsevier.

Neal, R. M. (1996). *Bayesian Learning for Neural Networks*. Secaucus, NJ, USA: Springer-Verlag New York, Inc.

Neal, R. M. (2000). Markov chain sampling methods for dirichlet process mixture models. *Journal of computational and graphical statistics*, *9*(2), 249–265.

Neelakantan, A., Vilnis, L., Le, Q. V., Sutskever, I., Kaiser, L., Kurach, K., & Martens, J. (2015). Adding gradient noise improves learning for very deep networks. *arXiv preprint arXiv:1511.06807*.

Neftci, E., Das, S., Pedroni, B., Kreutz-Delgado, K., & Cauwenberghs, G. (2014). Event-driven contrastive divergence for spiking neuromorphic systems. *Frontiers in neuroscience*, *7*, 272.

Negnevitsky, M. (2005). *Artificial intelligence: a guide to intelligent systems*. Pearson education.

Negrinho, R., & Aguiar, P. (2013). "shape representation via symmetric polynomials: a complete invariant inspired by the bispectrum. *Submitted for publication*.

Neumaier, A. (2003). Taylor forms—use and limits. *Reliable computing*, *9*(1), 43–79.

Ng, A., Ngiam, J., Foo, C. Y., & Mai, Y. (2014). Deep learning. *CS229 Lecture Notes*, (pp. 1–30).

Ng, K. W., Tian, G.-L., & Tang, M.-L. (2011). *Dirichlet and related distributions: Theory, methods and applications*, vol. 888. John Wiley & Sons.

Ng, S.-K., & McLachlan, G. J. (2014). Mixture models for clustering multilevel growth trajectories. *Computational Statistics & Data Analysis*, *71*, 43–51.

Ngah, S., Bakar, R. A., Embong, A., & Razali, S. (2016). Two-steps implementation of sigmoid function for artificial neural network in field programmable gate array. *ARPN Journal of Engineering and Applied Sciences*, *11*(7), 4882–4888.

Ngiam, J., Chen, Z., Chia, D., Koh, P., Le, Q., & Ng, A. (2010). Tiled convolutional neural networks. *Advances in neural information processing systems*, *23*, 1279–1287.

Nigrini, M. J. (2017). Audit sampling using benford's law: a review of the literature with some new perspectives. *Journal of emerging technologies in accounting*, *14*(2), 29–46.

Nikolenko, S., Kadurin, A., & Arkhangelskaya, E. (2018). Deep learning. *SPb.: Peter*.

Nilforooshan, M. A. (2020). mbend: an r package for bending non-positive-definite symmetric matrices to positive-definite. *BMC genetics*, *21*(1), 1–8.

Nilforooshan, M. A., & Nilforooshan, M. M. A. (2020). Package 'mbend'.

Nilsson, N. J. (2009). *The quest for artificial intelligence*. Cambridge University Press.

Nilsson, N. J. (2014). *Principles of artificial intelligence*. Morgan Kaufmann.

Noriega, L. (2005). Multilayer perceptron tutorial. *School of Computing. Staffordshire University*.

Novak, J., & Bortz, A. B. (1970). The evolution of the two-dimensional maxwell-boltzmann distribution. *American Journal of Physics*, *38*(12), 1402–1406.

Nwankpa, C., Ijomah, W., Gachagan, A., & Marshall, S. (2018). Activation functions: Comparison of trends in practice and research for deep learning. *arXiv preprint arXiv:1811.03378*.

Ollivier, Y., Arnold, L., Auger, A., & Hansen, N. (2017). Information-geometric optimization algorithms: A unifying picture via invariance principles. *The Journal of Machine Learning Research*, *18*(1), 564–628.

Ongaro, A., & Migliorati, S. (2013). A generalization of the dirichlet distribution. *Journal of Multivariate Analysis*, *114*, 412–426.

Orhan, E. (2012a). Bayesian statistics: Dirichlet processes.

Orhan, E. (2012b). *Dirichlet processes*. Ph.D. thesis, PhD thesis, Rochester University, 2012.(Cited on page 46.).

Orr, M. J., et al. (1996). Introduction to radial basis function networks.

O'Shea, K., & Nash, R. (2015). An introduction to convolutional neural networks. *arXiv preprint arXiv:1511.08458*.

Otsuki, T., Ogino, H., Ide, S., & Chiba, T. (2004). Curve interpolation method. US Patent 6,823,234.

O'Shea, T., & Hoydis, J. (2017). An introduction to deep learning for the physical layer. *IEEE Transactions on Cognitive Communications and Networking*, *3*(4), 563–575.

Page, G. L., & Quintana, F. A. (2018). Calibrating covariate informed product partition models. *Statistics and Computing*, *28*(5), 1009–1031.

Page, G. L., Quintana, F. A., & Dahl, D. B. (2019). Spatio-temporal random partition models. *arXiv preprint arXiv:1912.11542*.

Paik, J. K., & Katsaggelos, A. K. (1992). Image restoration using a modified hopfield network. *IEEE Transactions on image processing*, *1*(1), 49–63.

Paine, T., Jin, H., Yang, J., Lin, Z., & Huang, T. (2013). Gpu asynchronous stochastic gradient descent to speed up neural network training. *arXiv preprint arXiv:1312.6186*.

Paisley, J. (2010). A simple proof of the stick-breaking construction of the dirichlet process. *Technical Report, Princeton University, Department of Computer Science, Tech. Rep.*.

Paisley, J., Blei, D., & Jordan, M. (2012). Variational bayesian inference with stochastic search. *arXiv preprint arXiv:1206.6430*.

Pal, M. (2005). Random forest classifier for remote sensing classification. *International journal of remote sensing*, *26*(1), 217–222.

Pal, S. K., & Mitra, S. (1992). Multilayer perceptron, fuzzy sets, classifiaction.

Pan, B., Li, H., Zhao, Z., Cao, B., Cai, D., & He, X. (2017). Memen: Multi-layer embedding with memory networks for machine comprehension. *arXiv preprint arXiv:1707.09098*.

Panchapagesan, S., Sun, M., Khare, A., Matsoukas, S., Mandal, A., Hoffmeister, B., & Vitaladevuni, S. (2016). Multi-task learning and weighted cross-entropy for dnn-based keyword spotting. In *Interspeech*, vol. 9, (pp. 760–764).

Panicker, M., & Babu, C. (2012). Efficient fpga implementation of sigmoid and bipolar sigmoid activation functions for multilayer perceptrons. *IOSR Journal of Engineering*, *2*(6), 1352–1356.

Panov, P., & Džeroski, S. (2007). Combining bagging and random subspaces to create better ensembles. In *International Symposium on Intelligent Data Analysis*, (pp. 118–129). Springer.

Pappagari, R. R., Nayak, S., & Murty, K. S. R. (2014). Unsupervised spoken word retrieval using gaussian-bernoulli restricted boltzmann machines. In *Fifteenth Annual Conference of the International Speech Communication Association*.

Park, S. B., Lee, J. W., & Kim, S. K. (2004). Content-based image classification using a neural network. *Pattern Recognition Letters*, *25*(3), 287–300.

Parlos, A. G., Fernandez, B., Atiya, A. F., Muthusami, J., & Tsai, W. K. (1994). An accelerated learning algorithm for multilayer perceptron networks. *IEEE Transactions on Neural Networks*, *5*(3), 493–497.

Patel, J. K., & Read, C. B. (1996). *Handbook of the normal distribution*, vol. 150. CRC Press.

Pattanayak, S., & Ludwig, S. A. (2017). Encryption based on neural cryptography. In *International Conference on Health Information Science*, (pp. 321–330). Springer.

Paul, D., Wang, L., et al. (2016). Discussion of "estimating structured high-dimensional covariance and precision matrices: Optimal rates and adaptive estimation". *Electronic Journal of Statistics*, *10*(1), 74–80.

Pedersen, S. J. K. (2007). Circular hough transform. *Aalborg University, Vision, Graphics, and Interactive Systems*, *123*(6).

Peng, M., Gupta, N. K., & Armitage, A. F. (1996). An investigation into the improvement of local minima of the hopfield network. *Neural networks*, *9*(7), 1241–1253.

Pennington, J., & Bahri, Y. (2017). Geometry of neural network loss surfaces via random matrix theory. In *International Conference on Machine Learning*, (pp. 2798–2806).

Permuter, H., Francos, J., & Jermyn, I. (2006). A study of gaussian mixture models of color and texture features for image classification and segmentation. *Pattern Recognition*, *39*(4), 695–706.

Permuter, H., Francos, J., & Jermyn, I. H. (2003). Gaussian mixture models of texture and colour for image database retrieval. In *2003 IEEE International Conference on Acoustics, Speech, and Signal Processing, 2003. Proceedings.(ICASSP'03).*, vol. 3, (pp. III–569). IEEE.

Pernkopf, F., & Bouchaffra, D. (2005). Genetic-based em algorithm for learning gaussian mixture models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *27*(8), 1344–1348.

Pettersson, R. (1992). Stratonovich–taylor expansion and numerical methods. *Stochastic Analysis and Applications*, *10*(5), 603–612.

Pfeuty, P. (1970). The one-dimensional ising model with a transverse field. *ANNALS of Physics*, *57*(1), 79–90.

Pham, B. T., Prakash, I., Singh, S. K., Shirzadi, A., Shahabi, H., Bui, D. T., et al. (2019). Landslide susceptibility modeling using reduced error pruning trees and different ensemble techniques: Hybrid machine learning approaches. *Catena*, *175*, 203–218.

Philipsen, W., & Cluitmans, L. (1993). Using a genetic algorithm to tune potts neural networks. In *Artificial Neural Nets and Genetic Algorithms*, (pp. 650–657). Springer.

Phillips, G. M. (2003). *Interpolation and approximation by polynomials*, vol. 14. Springer Science & Business Media.

Pitman, J. (1996). Some developments of the blackwell-macqueen urn scheme. *Lecture Notes-Monograph Series*, (pp. 245–267).

Plant, R. (2011). An introduction to artificial intelligence. In *32nd Aerospace Sciences Meeting and Exhibit*, (p. 294).

Plaut, D. C., & Hinton, G. E. (1987). Learning sets of filters using back-propagation. *Computer Speech & Language*, *2*(1), 35–61.

Pollastri, G., Przybylski, D., Rost, B., & Baldi, P. (2002). Improving the prediction of protein secondary structure in three and eight classes using recurrent neural networks and profiles. *Proteins: Structure, Function, and Bioinformatics*, *47*(2), 228–235.

Povey, D., Chu, S. M., & Varadarajan, B. (2008). Universal background model based speech recognition. In *2008 IEEE International Conference on Acoustics, Speech and Signal Processing*, (pp. 4561–4564). IEEE.

Press, W. H., Teukolsky, S. A., Vetterling, W. T., & Flannery, B. P. (1996). *Numerical recipes in C*. Cambridge University Press.

Preston, C. J. (1973). Generalized gibbs states and markov random fields. *Advances in Applied probability*, *5*(2), 242–261.

Probst, P., Wright, M. N., & Boulesteix, A.-L. (2019). Hyperparameters and tuning strategies for random forest. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, *9*(3), e1301.

Procesi, C. (2007). Lie groups. an approach through invariants and representations. *Bull. Amer. Math. Soc*.

Prost-Boucle, A., Bourge, A., & Pétrot, F. (2018). High-efficiency convolutional ternary neural networks with custom adder trees and weight compression. *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, *11*(3), 1–24.

Qi, Y., Zhang, S., Qin, L., Huang, Q., Yao, H., Lim, J., & Yang, M.-H. (2018). Hedging deep features for visual tracking. *IEEE transactions on pattern analysis and machine intelligence*, *41*(5), 1116–1130.

Qian, Z. (2011). High order directional derivative and the simple form of multivariate taylor theorem. *Journal of Heze University*, (2), 4.

Qin, Z., Kim, D., & Gedeon, T. (2019). Rethinking softmax with cross-entropy: Neural network classifier as mutual information estimator. *arXiv preprint arXiv:1911.10688*.

Quintana, F. A. (2010). Linear regression with a dependent skewed dirichlet process. *Chilean Journal of Statistics*, *1*, 35–49.

Quintana, F. A., & Iglesias, P. L. (2003). Bayesian clustering and product partition models. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, *65*(2), 557–574.

Quintana, F. A., Mueller, P., Jara, A., & MacEachern, S. N. (2020). The dependent dirichlet process and related models. *arXiv preprint arXiv:2007.06129*.

Rabbani, A., & Babaei, M. (2019). Hybrid pore-network and lattice-boltzmann permeability modelling accelerated by machine learning. *Advances in water resources*, *126*, 116–128.

Rady, H. A. K. (2011). Shannon entropy and mean square errors for speeding the convergence of multilayer neural networks: A comparative approach. *Egyptian Informatics Journal*, *12*(3), 197–209.

Raghavan, U. N., Albert, R., & Kumara, S. (2007). Near linear time algorithm to detect community structures in large-scale networks. *Physical review E*, *76*(3), 036106.

Rahman, S., et al. (2004). A neural network with o (n) neurons for ranking n numbers in o (1/n) time. *IEEE Transactions on Circuits and Systems I: Regular Papers*, *51*(10), 2044–2051.

Rajaratnam, B., Massam, H., & Carvalho, C. M. (2008). Flexible covariance estimation in graphical Gaussian models. *The Annals of Statistics*, *36*(6), 2818–2849.

Ramachandran, P., Zoph, B., & Le, Q. V. (2017). Searching for activation functions. *arXiv preprint arXiv:1710.05941*.

Ramalingam, A., & Krishnan, S. (2006). Gaussian mixture modeling of short-time fourier transform features for audio fingerprinting. *IEEE Transactions on Information Forensics and Security*, *1*(4), 457–463.

Ramchoun, H., Idrissi, M. A. J., Ghanou, Y., & Ettaouil, M. (2016). Multilayer perceptron: Architecture optimization and training. *IJIMAI*, *4*(1), 26–30.

Ramsauer, H., Schäfl, B., Lehner, J., Seidl, P., Widrich, M., Gruber, L., Holzleitner, M., Pavlović, M., Sandve, G. K., Greiff, V., et al. (2020). Hopfield networks is all you need. *arXiv preprint arXiv:2008.02217*.

Ranganath, R., Gerrish, S., & Blei, D. (2014). Black box variational inference. In *Artificial intelligence and statistics*, (pp. 814–822). PMLR.

Ranganath, R., Wang, C., David, B., & Xing, E. (2013). An adaptive learning rate for stochastic variational inference. In *International Conference on Machine Learning*, (pp. 298–306). PMLR.

Ranjan, R., Castillo, C. D., & Chellappa, R. (2017). L2-constrained softmax loss for discriminative face verification. *arXiv preprint arXiv:1703.09507*.

Rasmussen, C. E. (1995). A practical monte carlo implementation of bayesian learning. In *Proceedings of the 8th International Conference on Neural Information Processing Systems*, NIPS'95, (pp. 598–604). Cambridge, MA, USA: MIT Press.
URL `http://dl.acm.org/citation.cfm?id=2998828.2998913`

Rastegari, M., Ordonez, V., Redmon, J., & Farhadi, A. (2016). Xnor-net: Imagenet classification using binary convolutional neural networks. In *European conference on computer vision*, (pp. 525–542). Springer.

Ravikumar, N., Gooya, A., Çimen, S., Frangi, A. F., & Taylor, Z. A. (2018). Group-wise similarity registration of point sets using student's t-mixture model for statistical shape models. *Medical image analysis*, *44*, 156–176.

Reichardt, J., & Bornholdt, S. (2004). Detecting fuzzy community structures in complex networks with a potts model. *Physical Review Letters*, *93*(21), 218701.

Reimer, M. (2012). *Multivariate polynomial approximation*, vol. 144. Birkhäuser.

Reynolds, D. (1995). Rc rose published a paper,". *Robust test-independent speaker identification using Gaussian mixture Speaker models." IEEE Transaction on Speech Audio Processing*, *3*, 72–83.

Rezende, D. J., & Viola, F. (2018). Generalized elbo with constrained optimization, geco. In *Workshop on Bayesian Deep Learning, NeurIPS*.

Riccioni, J., & Cerqueti, R. (2018). Regular paths in financial markets: Investigating the benford's law. *Chaos, Solitons & Fractals*, *107*, 186–194.

Richmond, P., & Solomon, S. (2001). Power laws are disguised boltzmann laws. *International Journal of Modern Physics C*, *12*(03), 333–343.

Riesen, K., Neuhaus, M., & Bunke, H. (2007). Bipartite graph matching for computing the edit distance of graphs. In *International Workshop on Graph-Based Representations in Pattern Recognition*, (pp. 1–12). Springer.

Rigatti, S. J. (2017). Random forest. *Journal of Insurance Medicine*, *47*(1), 31–39.

Rimer, M., & Martinez, T. (2004). Softprop: softmax neural network backpropagation learning. In *2004 IEEE International Joint Conference on Neural Networks (IEEE Cat. No. 04CH37541)*, vol. 2, (pp. 979–983). IEEE.

Robbins, H., & Monro, S. (1951). A stochastic approximation method. *The annals of mathematical statistics*, (pp. 400–407).

Robbins, H., & Siegmund, D. (1971). A convergence theorem for non negative almost supermartingales and some applications. In *Optimizing methods in statistics*, (pp. 233–257). Elsevier.

Robert, C. P., & Casella, G. (2005). *Monte Carlo Statistical Methods (Springer Texts in Statistics)*. Secaucus, NJ, USA: Springer-Verlag New York, Inc.

Robinson, A. J. (1994). An application of recurrent nets to phone probability estimation. *IEEE transactions on Neural Networks*, *5*(2), 298–305.

Rockafellar, R. T. (1993). Lagrange multipliers and optimality. *SIAM review*, *35*(2), 183–238.

Roman, J., & Jameel, A. (1996). Backpropagation and recurrent neural networks in financial analysis of multiple stock market returns. In *Proceedings of HICSS-29: 29th Hawaii International Conference on System Sciences*, vol. 2, (pp. 454–460). IEEE.

Romano, G., Barretta, R., & Diaco, M. (2018). A geometric rationale for invariance, covariance and constitutive relations. *Continuum Mechanics and Thermodynamics*, *30*(1), 175–194.

Ross, A., & Doshi-Velez, F. (2018). Improving the adversarial robustness and interpretability of deep neural networks by regularizing their input gradients. In *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32.

Rothganger, F., Lazebnik, S., Schmid, C., & Ponce, J. (2006). 3d object modeling and recognition using local affine-invariant image descriptors and multi-view spatial constraints. *International journal of computer vision*, *66*(3), 231–259.

Roy, D. (2003). The discrete normal distribution. *Communications in Statistics-theory and Methods*, *32*(10), 1871–1883.

Roy, S. K., Manna, S., Dubey, S. R., & Chaudhuri, B. B. (2019). Lisht: Non-parametric linearly scaled hyperbolic tangent activation function for neural networks. *arXiv preprint arXiv:1901.05894*.

Ruder, S. (2016). An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*.

Rukundo, O., & Cao, H. (2012). Nearest neighbor value interpolation. *arXiv preprint arXiv:1211.1768*.

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning representations by back-propagating errors. *nature*, *323*(6088), 533–536.

Runxuan, Z. (2005). Efficient sequential and batch learning artificial neural network methods for classification problems. *Singapore*, *2*, 825–845.

Russell, D. K. (1996). The boltzmann distribution. *Journal of Chemical Education*, *73*(4), 299.

Russell, S., & Norvig, P. (2002). Artificial intelligence: a modern approach.

Ruttor, A., Kinzel, W., & Kanter, I. (2007). Dynamics of neural cryptography. *Physical Review E*, *75*(5), 056104.

Ruttor, A., Kinzel, W., Shacham, L., & Kanter, I. (2004). Neural cryptography with feedback. *Physical Review E*, *69*(4), 046110.

Saad, D. (1998). Online algorithms and stochastic approximations. *Online Learning*, *5*, 6–3.

Sabourin, A., & Naveau, P. (2014). Bayesian dirichlet mixture model for multivariate extremes: a re-parametrization. *Computational Statistics & Data Analysis*, *71*, 542–567.

Sahoo, D., Pham, Q., Lu, J., & Hoi, S. C. (2017). Online deep learning: Learning deep neural networks on the fly. *arXiv preprint arXiv:1711.03705*.

Sak, H., Senior, A. W., & Beaufays, F. (2014). Long short-term memory recurrent neural network architectures for large scale acoustic modeling.

Salakhutdinov, R., Mnih, A., & Hinton, G. (2007). Restricted boltzmann machines for collaborative filtering. In *Proceedings of the 24th international conference on Machine learning*, (pp. 791–798).

Salas, J., & Sokal, A. D. (1997). Dynamic critical behavior of the swendsen-wang algorithm: The two-dimensional three-state potts model revisited. *Journal of statistical physics*, *87*(1-2), 1–36.

Salimans, T., Kingma, D., & Welling, M. (2015). Markov chain monte carlo and variational inference: Bridging the gap. In *International Conference on Machine Learning*, (pp. 1218–1226).

Salsburg, D. (2001). *The lady tasting tea: How statistics revolutionized science in the twentieth century*. Macmillan.

Samal, A., & Edwards, J. (1997). Generalized hough transform for natural shapes. *Pattern Recognition Letters*, *18*(5), 473–480.

Sanchez, E. N., & Perez, J. P. (1999). Input-to-state stability (iss) analysis for dynamic neural networks. *IEEE Transactions on circuits and systems I: Fundamental Theory and Applications*, *46*(11), 1395–1398.

Schindler, A., Lidy, T., & Rauber, A. (2016). Comparing shallow versus deep neural network architectures for automatic music genre classification. In *FMT*, (pp. 17–21).

Schmidt, M., Fung, G., & Rosales, R. (2007). Fast optimization methods for l_1 regularization: A comparative study and two new approaches. In J. N. Kok, J. Koronacki, R. L. d. Mantaras, S. Matwin, D. Mladenič, & A. Skowron (Eds.) *Machine Learning: ECML 2007*, (pp. 286–297). Berlin, Heidelberg: Springer Berlin Heidelberg.

Schonlau, M., & Zou, R. Y. (2020). The random forest algorithm for statistical learning. *The Stata Journal*, *20*(1), 3–29.

Schwing, A. G., & Urtasun, R. (2015). Fully connected deep structured networks. *arXiv preprint arXiv:1503.02351*.

Sebastien, B. (2013). Orf523: Oracle complexity, large-scale optimization.

Seck, I., Loosli, G., & Canu, S. (2019). L 1-norm double backpropagation adversarial defense. *arXiv preprint arXiv:1903.01715*.

Seidel, W., & Ševčíková, H. (2004). Types of likelihood maxima in mixture models and their implication on the performance of tests. *Annals of the Institute of Statistical Mathematics*, *56*(4), 631–654.

Sejnowski, T. J. (2018). *The deep learning revolution*. Mit Press.

Selke, W., & Huse, D. A. (1983). Interfacial adsorption in planar potts models. *Zeitschrift für Physik B Condensed Matter*, *50*(2), 113–116.

Semenov, A., Boginski, V., & Pasiliao, E. L. (2019). Neural networks with multidimensional cross-entropy loss functions. In *International Conference on Computational Data and Social Networks*, (pp. 57–62). Springer.

Seo, J., Yu, J., Lee, J., & Choi, K. (2016). A new approach to binarizing neural networks. In *2016 International SoC Design Conference (ISOCC)*, (pp. 77–78). IEEE.

Sermanet, P., Chintala, S., & LeCun, Y. (2012). Convolutional neural networks applied to house numbers digit classification. In *Proceedings of the 21st International Conference on Pattern Recognition (ICPR2012)*, (pp. 3288–3291). IEEE.

Shamsi, J., Amirsoleimani, A., Mirzakuchaki, S., Ahmade, A., Alirezaee, S., & Ahmadi, M. (2015). Hyperbolic tangent passive resistive-type neuron. In *2015 IEEE International Symposium on Circuits and Systems (ISCAS)*, (pp. 581–584). IEEE.

Shan, B., & Fang, Y. (2020). A cross entropy based deep neural network model for road extraction from satellite images. *Entropy*, *22*(5), 535.

Shang, Y., & Wah, B. W. (1996). Global optimization for neural network training. *Computer*, *29*(3), 45–54.

Sharma, K., Aggarwal, A., Singhania, T., Gupta, D., & Khanna, A. (2019). Hiding data in images using cryptography and deep neural network. *arXiv preprint arXiv:1912.10413*.

Sharma, S. (2017). Activation functions in neural networks. *Towards Data Science*, *6*.

Shaw, A. M., Doyle III, F. J., & Schwaber, J. S. (1997). A dynamic neural network approach to nonlinear process modeling. *Computers & chemical engineering*, *21*(4), 371–385.

Shawahna, A., Sait, S. M., & El-Maleh, A. (2018). Fpga-based accelerators of deep learning networks for learning and classification: A review. *IEEE Access*, *7*, 7823–7859.

Shen, J. J. (2006). A stochastic-variational model for soft mumford-shah segmentation. *International Journal of Biomedical Imaging*, *2006*.

Sherman, S. (1973). Markov random fields and gibbs random fields. *Israel Journal of Mathematics*, *14*(1), 92–103.

Sherrington, D., & Kirkpatrick, S. (1975). Solvable model of a spin-glass. *Physical review letters*, *35*(26), 1792.

Shibata, K., & Ito, K. (1999). Gauss-sigmoid neural network. In *IJCNN'99. International Joint Conference on Neural Networks. Proceedings (Cat. No. 99CH36339)*, vol. 2, (pp. 1203–1208). IEEE.

Shoaib, M., Shamseldin, A. Y., Melville, B. W., & Khan, M. M. (2016). A comparison between wavelet based static and dynamic neural network approaches for runoff prediction. *Journal of hydrology*, *535*, 211–225.

Sholahudin, S., & Han, H. (2016). Simplified dynamic neural network model to predict heating load of a building using taguchi method. *Energy*, *115*, 1672–1678.

Shu, C., Pang, W., Liu, H., & Lu, S. (2019). High energy efficiency fpga-based accelerator for convolutional neural networks using weight combination. In *2019 IEEE 4th International Conference on Signal and Image Processing (ICSIP)*, (pp. 578–582). IEEE.

Sigtia, S., & Dixon, S. (2014). Improved music feature learning with deep neural networks. In *2014 IEEE international conference on acoustics, speech and signal processing (ICASSP)*, (pp. 6959–6963). IEEE.

Silva, L. M., de Sá, J. M., & Alexandre, L. A. (2005). Neural network classification using shannon's entropy. In *ESANN*, (pp. 217–222). Citeseer.

Simões, J. M. d. C. (2019). *Deep Learning for Dynamic Music Generation*. Ph.D. thesis, Universidade de Coimbra.

Simon, J. (2011). Prince: Computer vision: Models, learning, and inference.

Simons, T., & Lee, D.-J. (2019). A review of binarized neural networks. *Electronics*, *8*(6), 661.

Simonyan, K., & Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.

Sinclair, A., Srivastava, P., & Thurley, M. (2014). Approximation algorithms for two-state anti-ferromagnetic spin systems on bounded degree graphs. *Journal of Statistical Physics*, *155*(4), 666–686.

Sivaram, G. S., & Hermansky, H. (2011). Sparse multilayer perceptron for phoneme recognition. *IEEE Transactions on Audio, Speech, and Language Processing*, *20*(1), 23–29.

Skurichina, M., & Duin, R. P. (2001). Bagging and the random subspace method for redundant feature spaces. In *International Workshop on Multiple Classifier Systems*, (pp. 1–10). Springer.

Smith, S. L., Kindermans, P.-J., Ying, C., & Le, Q. V. (2017). Don't decay the learning rate, increase the batch size. *arXiv preprint arXiv:1711.00489*.

Smith, S. P., & Jain, A. K. (1982). Chord distributions for shape matching. *Computer Graphics and Image Processing*, *20*(3), 259–271.

Smyth, G. K. (2014). Polynomial approximation. *Wiley StatsRef: Statistics Reference Online*.

Snyder, D., Garcia-Romero, D., & Povey, D. (2015). Time delay deep neural network-based universal background models for speaker recognition. In *2015 IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU)*, (pp. 92–97). IEEE.

Soda, G., Usai, A., & Zaheer, A. (2004). Network memory: The influence of past and current networks on performance. *Academy of Management Journal*, *47*(6), 893–906.

Sokal, A. (1997a). Monte carlo methods in statistical mechanics: foundations and new algorithms. In *Functional integration*, (pp. 131–192). Springer.

Sokal, A. (1997b). Monte Carlo methods in statistical mechanics: foundations and new algorithms. In *Functional integration (Cargèse, 1996)*, vol. 361 of *NATO Adv. Sci. Inst. Ser. B Phys.*, (pp. 131–192). Plenum, New York.

Solomonoff, R. J. (2006). Machine learning-past and future. *Dartmouth, NH, July*.

Soromenho, G. (1994). Comparing approaches for testing the number of components in a finite mixture model. *Computational Statistics*, *9*(1), 65–78.

Späth, H. (1979). Algorithm 39 clusterwise linear regression. *Computing*, *22*(4), 367–373. URL `https://doi.org/10.1007/BF02265317`

Spitzer, F. (1971). Markov random fields and gibbs ensembles. *The American Mathematical Monthly*, *78*(2), 142–154.

Spoerer, C. J., McClure, P., & Kriegeskorte, N. (2017). Recurrent convolutional neural networks: a better model of biological object recognition. *Frontiers in psychology*, *8*, 1551.

Srebro, N. (2007). Are there local maxima in the infinite-sample likelihood of gaussian mixture estimation? In *International Conference on Computational Learning Theory*, (pp. 628–629). Springer.

Srihari, S. N., & Govindaraju, V. (1989). Analysis of textual images using the hough transform. *Machine vision and Applications*, *2*(3), 141–153.

Srinivas, S., Subramanya, A., & Venkatesh Babu, R. (2017). Training sparse neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, (pp. 138–145).

Srivastava, A., & Sutton, C. (2017). Autoencoding variational inference for topic models. *arXiv preprint arXiv:1703.01488*.

Srivastava, M. S. (1965). On the complex wishart distribution. *The Annals of mathematical statistics*, *36*(1), 313–315.

Stam, A. (1983). Generation of a random partition of a finite set by an urn model. *Journal of Combinatorial Theory, Series A*, *35*(2), 231–240.

Stanberry, L., Murua, A., & Cordes, D. (2008a). Functional connectivity mapping using the ferromagnetic potts spin model. *Human brain mapping*, *29*(4), 422–440.

Stanberry, L., Murua, A., & Cordes, D. (2008b). Functional connectivity mapping using the ferromagnetic Potts spin model. *Human Brain Mapping*, *29*, 422–440.

Steck, H., & Jaakkola, T. (2002). On the dirichlet prior and bayesian regularization. *Advances in neural information processing systems*, *15*, 713–720.

Steele, R. J., & Raftery, A. E. (2010). Performance of bayesian model selection criteria for gaussian mixture models. *Frontiers of statistical decision making and bayesian analysis*, *2*, 113–130.

Steger, C. (2002). Occlusion, clutter, and illumination invariant object recognition. *International Archives of Photogrammetry Remote Sensing and Spatial Information Sciences*, *34*(3/A), 345–350.

Steinley, D., & Brusco, M. J. (2011). Evaluating mixture modeling for clustering: Recommendations and cautions. *Psychological Methods*, *16*(1), 63.

Stephens, M. (2000). Dealing with label switching in mixture models. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, *62*(4), 795–809.

Strobl, C., Malley, J., & Tutz, G. (2009). An introduction to recursive partitioning: rationale, application, and characteristics of classification and regression trees, bagging, and random forests. *Psychological methods*, *14*(4), 323.

Stylianou, Y., Pantazis, Y., Calderero, F., Larroy, P., Severin, F., Schimke, S., Bonal, R., Matta, F., & Valsamakis, A. (2005). Gmm-based multimodal biometric verification. In *eINTERFACE'05-Summer Workshop on Multimodal Interfaces*.

Su, H., & Xu, H. (2015). Multi-softmax deep neural network for semi-supervised training. In *Sixteenth Annual Conference of the International Speech Communication Association*.

Su, T., & Dy, J. G. (2007). In search of deterministic methods for initializing k-means and gaussian mixture clustering. *Intelligent Data Analysis*, *11*(4), 319–338.

Sudheer, K., & Jain, S. (2003). Radial basis function neural network for modeling rating curves. *Journal of Hydrologic Engineering*, *8*(3), 161–164.

Suetake, N., Uchino, E., & Hirata, K. (2006). Generalized fuzzy hough transform for detecting arbitrary shapes in a vague and noisy image. *Soft Computing*, *10*(12), 1161–1168.

Sugiyama, M. (2015). *Introduction to statistical machine learning*. Morgan Kaufmann.

Sukhbaatar, S., Weston, J., Fergus, R., et al. (2015). End-to-end memory networks. In *Advances in neural information processing systems*, (pp. 2440–2448).

Sultana, F., Sufian, A., & Dutta, P. (2018). Advancements in image classification using convolutional neural network. In *2018 Fourth International Conference on Research in Computational Intelligence and Communication Networks (ICRCICN)*, (pp. 122–129). IEEE.

Sun, C., & Cai, R. (2009). Document image registration using geometric invariance and hausdorff distance. In *2009 First International Workshop on Education Technology and Computer Science*, vol. 2, (pp. 725–728). IEEE.

Sun, C., Chen, S., & Huang, X. (2020). Double backpropagation for training autoencoders against adversarial attack. *arXiv preprint arXiv:2003.01895*.

Sun, S., Chen, C., & Carin, L. (2017). Learning Structured Weight Uncertainty in Bayesian Neural Networks. In A. Singh, & J. Zhu (Eds.) *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, vol. 54 of *Proceedings of Machine Learning Research*, (pp. 1283–1292). Fort Lauderdale, FL, USA: PMLR.
URL `http://proceedings.mlr.press/v54/sun17b.html`

Sun, W., Shao, S., Zhao, R., Yan, R., Zhang, X., & Chen, X. (2016). A sparse auto-encoder-based deep neural network approach for induction motor faults classification. *Measurement*, *89*, 171–178.

Sun, Z., Huang, Y., Yu, W., Zhang, M., Shui, R., & Gao, T. (2015). How to break the configuration of moving objects? geometric invariance in visual working memory. *Journal of experimental psychology: human perception and performance*, *41*(5), 1247.

Suprem, A., & Ruprem, M. (2013). A new composition algorithm for automatic generation of thematic music from the existing music pieces. In *Proceedings of the World Congress on Engineering and Computer Science*, vol. 2, (pp. 23–25).

Surden, H. (2014). Machine learning and law. *Wash. L. Rev.*, *89*, 87.

Sutherland, J. G. (1992). The holographic neural method. *Fuzzy, holographic and parallel intelligence*, (pp. 30–63).

Sutskever, I., & Hinton, G. E. (2008). Deep, narrow sigmoid belief networks are universal approximators. *Neural computation*, *20*(11), 2629–2636.

Sutton, C., Sindelar, M., & McCallum, A. (2005). Feature bagging: Preventing weight undertraining in structured discriminative learning. *Center for Intelligent Information Retrieval, U. of Massachusetts*.

Suykens, J. A., & Vandewalle, J. (1999). Training multilayer perceptron classifiers based on a modified support vector method. *IEEE transactions on Neural Networks*, *10*(4), 907–911.

Svensén, M., & Bishop, C. M. (2005). Robust bayesian mixture modelling. *Neurocomputing*, *64*, 235–252.

Svetnik, V., Liaw, A., Tong, C., Culberson, J. C., Sheridan, R. P., & Feuston, B. P. (2003). Random forest: a classification and regression tool for compound classification and qsar modeling. *Journal of chemical information and computer sciences*, *43*(6), 1947–1958.

Svirsky, Y., & Sharf, A. (2020). A non-linear differentiable cnn-rendering module for 3d data enhancement. *IEEE Transactions on Visualization and Computer Graphics*.

Sweeny, M. (1983). Monte carlo study of weighted percolation clusters relevant to the potts models. *Physical Review B*, *27*(7), 4445.

Swendsen, R. H., & Wang, J.-S. (1987). Nonuniversal critical dynamics in monte carlo simulations. *Physical review letters*, *58*(2), 86.

Szita, I., & Lörincz, A. (2006). Learning tetris using the noisy cross-entropy method. *Neural computation*, *18*(12), 2936–2941.

Tai, K. S., Socher, R., & Manning, C. D. (2015). Improved semantic representations from tree-structured long short-term memory networks. *arXiv preprint arXiv:1503.00075*.

Takeda, Y., Tamate, S., Yamamoto, Y., Takesue, H., Inagaki, T., & Utsunomiya, S. (2017). Boltzmann sampling for an xy model using a non-degenerate optical parametric oscillator network. *Quantum Science and Technology*, *3*(1), 014004.

Tan, K., Wu, F., Du, Q., Du, P., & Chen, Y. (2019). A parallel gaussian–bernoulli restricted boltzmann machine for mining area classification with hyperspectral imagery. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, *12*(2), 627–636.

Tang, D., Qin, B., & Liu, T. (2015). Document modeling with gated recurrent neural network for sentiment classification. In *Proceedings of the 2015 conference on empirical methods in natural language processing*, (pp. 1422–1432).

Tannenbaum, A. (2006). *Invariance and system theory: Algebraic and geometric aspects*, vol. 845. Springer.

Tantrum, J., Murua, A., & Stuetzle, W. (2003). Assessment and pruning of hierarchical model based clustering. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, (pp. 197–205).

Tantrum, J., Murua, A., & Stuetzle, W. (2004). Hierarchical model-based clustering of large datasets through fractionation and refractionation. *Information Systems*, *29*(4), 315–326.

Tarter, M. E., & Lock, M. D. (1993). *Model-free curve estimation*, vol. 56. CRC Press.

Tayarani-Bathaie, S. S., Vanini, Z. S., & Khorasani, K. (2014). Dynamic neural network-based fault diagnosis of gas turbine engines. *Neurocomputing*, *125*, 153–165.

Thomaz, A. L., Breazeal, C., Barto, A. G., & Picard, R. (2006). Socially guided machine learning.

Tibély, G., & Kertész, J. (2008). On the equivalence of the label propagation method of community detection and a potts model approach. *Physica A: Statistical Mechanics and its Applications*, *387*(19), 4982–4984.

Tödter, K.-H. (2009). Benford's law as an indicator of fraud in economics. *German Economic Review*, *10*(3), 339–351.

Tomita, Y., & Okabe, Y. (2001). Probability-changing cluster algorithm for potts models. *Physical Review Letters*, *86*(4), 572.

Tommiska, M. (2003). Efficient digital implementation of the sigmoid function for reprogrammable logic. *IEE Proceedings-Computers and Digital Techniques*, *150*(6), 403–411.

Tong, D. (2006). Lectures on quantum field theory. *Part III Cambridge University Mathematics Tripos, Michaelmas*.

Tong, D. (2012). Statistical physics. *Part II Cambridge University Mathematics Tripos, Michaelmas*.

Tong, Z., & Zhang, H. (2016). A text mining research based on lda topic modelling. In *International Conference on Computer Science, Engineering and Information Technology*, (pp. 201–210).

Torbati, A. H. H. N., & Picone, J. (2015). A doubly hierarchical dirichlet process hidden markov model with a non-ergodic structure. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, *24*(1), 174–184.

Torgo, L., & da Costa, J. P. (2000). Clustered multiple regression. In H. A. L. Kiers, J.-P. Rasson, P. J. F. Groenen, & M. Schader (Eds.) *Data Analysis, Classification, and Related Methods*, (pp. 217–222). Berlin, Heidelberg: Springer Berlin Heidelberg.

Torres-Mendez, L. A., Ruiz-Suarez, J. C., Sucar, L. E., & Gomez, G. (2000). Translation, rotation, and scale-invariant object recognition. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, *30*(1), 125–130.

Tosranon, P., Sanpanich, A., Bunluechokchai, C., & Pintavirooj, C. (2009). Gaussian curvature-based geometric invariance. In *2009 6th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology*, vol. 2, (pp. 1124–1127). IEEE.

Toth, L. F., et al. (1987). Branko grunbaum and gc shephard, tilings and patterns. *Bulletin (New Series) of the American Mathematical Society*, *17*(2), 369–372.

Tsai, C.-H., Chih, Y.-T., Wong, W. H., & Lee, C.-Y. (2015). A hardware-efficient sigmoid function with adjustable precision for a neural network system. *IEEE Transactions on Circuits and Systems II: Express Briefs*, *62*(11), 1073–1077.

Tsai, D.-M. (1997). An improved generalized hough transform for the recognition of overlapping objects. *Image and Vision computing*, *15*(12), 877–888.

Tsoi, A. C. (1997). Gradient based learning methods. In *International School on Neural Networks, Initiated by IIASS and EMFCSC*, (pp. 27–62). Springer.

Tsoumakas, G., Spyromitros-Xioufis, E., Vilcek, J., & Vlahavas, I. (2011). Mulan: A java library for multi-label learning. *Journal of Machine Learning Research*, *12*(Jul), 2411–2414.

Tsoumakas, G., Spyromitros-Xioufis, E., Vilcek, J., & Vlahavas, I. (2020). Datasets from mulan: A java library for multi-label learning.
URL `http://mulan.sourceforge.net/datasets-mtr.html`

Tu, S. (2014). The dirichlet-multinomial and dirichlet-categorical models for bayesian inference. *Computer Science Division, UC Berkeley*.

Tu, Y., Sadiq, S., Tao, Y., Shyu, M.-L., & Chen, S.-C. (2019). A power efficient neural network implementation on heterogeneous fpga and gpu devices. In *2019 IEEE 20th International Conference on Information Reuse and Integration for Data Science (IRI)*, (pp. 193–199). IEEE.

Tüske, Z., Tahir, M. A., Schlüter, R., & Ney, H. (2015). Integrating gaussian mixtures into deep neural networks: Softmax layer with hidden variables. In *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, (pp. 4285–4289). IEEE.

Tzortzis, G., & Likas, A. (2007). Deep belief networks for spam filtering. In *19th IEEE International Conference on Tools with Artificial Intelligence (ICTAI 2007)*, vol. 2, (pp. 306–309). IEEE.

Ueda, N., Nakano, R., Ghahramani, Z., & Hinton, G. E. (1999). Smem algorithm for mixture models. In *Advances in neural information processing systems*, (pp. 599–605).

Ulrich, M., Steger, C., & Baumgartner, A. (2003). Real-time object recognition using a modified generalized hough transform. *Pattern Recognition*, *36*(11), 2557–2570.

Upadhya, V., & Sastry, P. (2019). An overview of restricted boltzmann machines. *Journal of the Indian Institute of Science*, *99*(2), 225–236.

Uykan, Z. (2013). Fast-convergent double-sigmoid hopfield neural network as applied to optimization problems. *IEEE transactions on neural networks and learning systems*, *24*(6), 990–996.

Vaičiulytė, J., & Sakalauskas, L. (2020). Recursive parameter estimation algorithm of the dirichlet hidden markov model. *Journal of Statistical Computation and Simulation*, *90*(2), 306–323.

Vargas, R., Mosavi, A., & Ruiz, R. (2017). Deep learning: a review. *Advances in Intelligent Systems and Computing*.

Varshney, K. R. (2019). Trustworthy machine learning and artificial intelligence. *XRDS: Crossroads, The ACM Magazine for Students*, *25*(3), 26–29.

Vehtari, A., & Lampinen, J. (1999). Bayesian neural networks for industrial applications. In *SMCia/99 Proceedings of the 1999 IEEE Midnight - Sun Workshop on Soft Computing Methods in Industrial Applications (Cat. No.99EX269)*, (pp. 63–68).

Venables, W., & Ripley, B. (2002). Modern applied statistics (fourth s., editor) new york.

Venkatesan, P., & Anitha, S. (2006). Application of a radial basis function neural network for diagnosis of diabetes mellitus. *Current Science*, *91*(9), 1195–1199.

Ventresca, M., & Tizhoosh, H. R. (2009). Improving gradient-based learning algorithms for large scale feedforward networks. In *2009 International Joint Conference on Neural Networks*, (pp. 3212–3219). IEEE.

Verbeek, J. (2004). *Mixture models for clustering and dimension reduction*. Ph.D. thesis.

Verdu, S. (1996). The exponential distribution in information theory. *Problemy peredachi informatsii*, *32*(1), 100–111.

Vermunt, J. K. (2011). K-means may perform as well as mixture model clustering but may also be much worse: Comment on steinley and brusco (2011).

Villatoro, F. R., & Ramos, J. I. (1999). On the method of modified equations. i: Asymptotic analysis of the euler forward difference method. *Applied mathematics and computation*, *103*(2-3), 111–139.

Volná, E. (2000). Using neural network in cryptography. In *The State of the Art in Computational Intelligence*, (pp. 262–267). Springer.

Volna, E., Kotyrba, M., Kocian, V., & Janosek, M. (2012). Cryptography based on neural network. In *ECMS*, (pp. 386–391).

Volokitin, A., Roig, G., & Poggio, T. A. (2017). Do deep neural networks suffer from crowding? In *Advances in Neural Information Processing Systems*, (pp. 5628–5638).

Vt, S. E., & Shin, Y. C. (1994). Radial basis function neural network for approximation and estimation of nonlinear stochastic dynamic systems. *IEEE transactions on neural networks*, *5*(4), 594–603.

Wagner, K. H., & McComb, S. (2019). Optical rectifying linear units for back-propagation learning in a deep holographic convolutional neural network. *IEEE Journal of Selected Topics in Quantum Electronics*, *26*(1), 1–18.

Wagstaff, K. (2012). Machine learning that matters. *arXiv preprint arXiv:1206.4656*.

Wainwright, M. J. (2014). Structured regularizers for high-dimensional problems: Statistical and computational issues. *Annual Review of Statistics and Its Application*, *1*, 233–253.

Wang, B., Liakata, M., Zubiaga, A., & Procter, R. (2017). A hierarchical topic modelling approach for tweet clustering. In *International Conference on Social Informatics*, (pp. 378–390). Springer.

Wang, D., Xu, K., Guo, J., & Ghiasi, S. (2020). Dsp-efficient hardware acceleration of convolutional neural network inference on fpgas. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, *39*(12), 4867–4880.

Wang, H., & Raj, B. (2017). On the origin of deep learning. *arXiv preprint arXiv:1702.07800*.

Wang, J. (2001). Generating daily changes in market variables using a multivariate mixture of normal distributions. In *Proceeding of the 2001 Winter Simulation Conference (Cat. No. 01CH37304)*, vol. 1, (pp. 283–289). IEEE.

Wang, J.-S., & Swendsen, R. H. (1990). Cluster monte carlo algorithms. *Physica A: Statistical Mechanics and its Applications*, *167*(3), 565–579.

Wang, S., Mazumder, S., Liu, B., Zhou, M., & Chang, Y. (2018). Target-sensitive memory networks for aspect sentiment classification. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, (pp. 957–967).

Wang, Z., Wohlwend, J., & Lei, T. (2019). Structured pruning of large language models. *arXiv preprint arXiv:1910.04732*.

Wanto, A., Windarto, A. P., Hartama, D., & Parlina, I. (2017). Use of binary sigmoid function and linear identity in artificial neural networks for forecasting population density. *IJISTECH (International Journal of Information System & Technology)*, *1*(1), 43–54.

Watrin, C., Struffert, R., & Ullmann, R. (2008). Benford's law: an instrument for selecting tax audit targets? *Review of managerial science*, *2*(3), 219.

Watson, N. V., & Breedlove, S. M. (2012). *The mind's machine: Foundations of brain and behavior.*. Sinauer Associates.

Weiss, I. (1993). Geometric invariants and object recognition. *International Journal of Computer 11263on*, *10*(3), 207–231.

Wen, U.-P., Lan, K.-M., & Shih, H.-S. (2009). A review of hopfield neural networks for solving mathematical programming problems. *European Journal of Operational Research*, *198*(3), 675–687.

Weston, J., Chopra, S., & Bordes, A. (2014). Memory networks. *arXiv preprint arXiv:1410.3916*.

Wexler, J., Pushkarna, M., Bolukbasi, T., Wattenberg, M., Viégas, F., & Wilson, J. (2019). The what-if tool: Interactive probing of machine learning models. *IEEE transactions on visualization and computer graphics*, *26*(1), 56–65.

Wiegerinck, W., & Kappen, B. (2000). Approximations of bayesian networks through kl minimisation. *New Generation Computing*, *18*(2), 167–175.

Wiesler, S., Li, J., & Xue, J. (2013). Investigations on hessian-free optimization for cross-entropy training of deep neural networks. In *Interspeech*, (pp. 3317–3321).

Wijesiriwardana, C., & Firdhous, M. (2019). An innovative query tuning scheme for large databases. In *2019 International Conference on Data Science and Engineering (ICDSE)*, (pp. 154–159). IEEE.

Wilamowski, B. M., Iplikci, S., Kaynak, O., & Efe, M. O. (2001). An algorithm for fast convergence in training neural networks. In *IJCNN'01. International Joint Conference on Neural Networks. Proceedings (Cat. No. 01CH37222)*, vol. 3, (pp. 1778–1782). Ieee.

Williams, C. K., & Rasmussen, C. E. (2006). *Gaussian processes for machine learning*, vol. 2. MIT press Cambridge, MA.

Williams, D., Hodge, V. J., Gega, L., Murphy, D., Cowling, P. I., & Drachen, A. (2019). Ai and automatic music generation for mindfulness. In *2019 AES International Conference on Immersive and Interactive Audio: Creating the Next Dimension of Sound Experience*. York.

Williams, D. A., Hodge, V. J., Wu, C.-Y., et al. (2020). On the use of ai for generation of functional music to improve mental health. *Frontiers in Artificial Intelligence*.

Wolfinger, R. (1993). Covariance structure selection in general mixed models. *Communications in statistics-Simulation and computation*, *22*(4), 1079–1106.

Wong, A. K., & You, M. (1985). Entropy and distance of random graphs with application to structural pattern recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, (5), 599–609.

Wu, J. (2017). Introduction to convolutional neural networks. *National Key Lab for Novel Software Technology. Nanjing University. China*, *5*, 23.

Wu, M., & Zhang, Z. (2010). Handwritten digit classification using the mnist data set. *Course project CSE802: Pattern Classification & Analysis*.

Wu, W., Zhang, N., Li, Z., Li, L., & Liu, Y. (2008). Convergence of gradient method with momentum for back-propagation neural networks. *Journal of Computational Mathematics*, (pp. 613–623).

Wu, Y., Li, J., Kong, Y., & Fu, Y. (2016). Deep convolutional neural network with independent softmax for large scale face recognition. In *Proceedings of the 24th ACM international conference on Multimedia*, (pp. 1063–1067).

Wu, Y., Liu, L., Bae, J., Chow, K.-H., Iyengar, A., Pu, C., Wei, W., Yu, L., & Zhang, Q. (2019). Demystifying learning rate policies for high accuracy training of deep neural networks. In *2019 IEEE International Conference on Big Data (Big Data)*, (pp. 1971–1980). IEEE.

Wu, Y., Zhao, H., & Zhang, L. (2014). Image denoising with rectified linear units. In *International Conference on Neural Information Processing*, (pp. 142–149). Springer.

Xie, J., & Szymanski, B. K. (2011). Community detection using a neighborhood strength driven label propagation algorithm. In *2011 IEEE Network Science Workshop*, (pp. 188–195). IEEE.

Xie, J., & Szymanski, B. K. (2013). Labelrank: A stabilized label propagation algorithm for community detection in networks. In *2013 IEEE 2nd Network Science Workshop (NSW)*, (pp. 138–143). IEEE.

Xie, Z., Huang, Y., Zhu, Y., Jin, L., Liu, Y., & Xie, L. (2019). Aggregation cross-entropy for sequence recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, (pp. 6538–6547).

Xiong, S., Guo, W., & Liu, D. (2014). The vietnamese speech recognition based on rectified linear units deep neural network and spoken term detection system combination. In *The 9th International Symposium on Chinese Spoken Language Processing*, (pp. 183–186). IEEE.

Xiong, Z., Zheng, T. F., Song, Z., Soong, F., & Wu, W. (2006). A tree-based kernel selection approach to efficient gaussian mixture model–universal background model based speaker identification. *Speech communication*, *48*(10), 1273–1282.

Xu, L. (1993). Least mean square error reconstruction principle for self-organizing neural-nets. *Neural networks*, *6*(5), 627–648.

Xu, L., & Jordan, M. I. (1996). On convergence properties of the em algorithm for gaussian mixtures. *Neural computation*, *8*(1), 129–151.

Xu, T., Zhang, Z., Philip, S. Y., & Long, B. (2008). Evolutionary clustering by hierarchical dirichlet process with hidden markov state. In *2008 Eighth IEEE International Conference on Data Mining*, (pp. 658–667). IEEE.

Xu, Y., Kong, Q., Huang, Q., Wang, W., & Plumbley, M. D. (2017). Convolutional gated recurrent neural network incorporating spatial features for audio tagging. In *2017 International Joint Conference on Neural Networks (IJCNN)*, (pp. 3461–3466). IEEE.

Yadav, C., & Bottou, L. (2019). Cold case: The lost mnist digits. In *Advances in Neural Information Processing Systems*, (pp. 13443–13452).

Yadav, S., & Shukla, S. (2016). Analysis of k-fold cross-validation over hold-out validation on colossal datasets for quality classification. In *2016 IEEE 6th International conference on advanced computing (IACC)*, (pp. 78–83). IEEE.

Yamanashi, Y., Umeda, K., & Yoshikawa, N. (2012). Pseudo sigmoid function generator for a superconductive neural network. *IEEE transactions on applied superconductivity*, *23*(3), 1701004–1701004.

Yamano, T. (2019). Some bounds for skewed α-jensen-shannon divergence. *Results in Applied Mathematics*, *3*, 100064.

Yamashita, T., Tanaka, M., Yoshida, E., Yamauchi, Y., & Fujiyoshii, H. (2014). To be bernoulli or to be gaussian, for a restricted boltzmann machine. In *2014 22nd International Conference on Pattern Recognition*, (pp. 1520–1525). IEEE.

Yang, H., & Ni, J. (2005). Dynamic neural network modeling for nonlinear, nonstationary machine tool thermally induced error. *International Journal of Machine Tools and Manufacture*, *45*(4-5), 455–465.

Yang, J., & Ma, J. (2019). Feed-forward neural network training using sparse representation. *Expert Systems with Applications*, *116*, 255–264.

Yang, M.-S., Lai, C.-Y., & Lin, C.-Y. (2012). A robust em clustering algorithm for gaussian mixture models. *Pattern Recognition*, *45*(11), 3950–3961.

Yang, S., Cui, X., & Fang, Z. (2014). Bcrgt: a bayesian cluster regression-based genotyping algorithm for the samples with copy number alterations. *BMC bioinformatics*, *15*(1), 74.

Yang, X. (2017). Understanding the variational lower bound.

Yao, K., Cohn, T., Vylomova, K., Duh, K., & Dyer, C. (2015). Depth-gated recurrent neural networks. *arXiv preprint arXiv:1508.03790*, *9*.

Yarotsky, D. (2018). Universal approximations of invariant maps by neural networks. *arXiv preprint arXiv:1804.10306*.

Yasuda, M. (2018). Learning algorithm of boltzmann machine based on spatial monte carlo integration method. *Algorithms*, *11*(4), 42.

Ye, L., Beskos, A., De Iorio, M., & Hao, J. (2020). Monte carlo co-ordinate ascent variational inference. *Statistics and Computing*, (pp. 1–19).

Yeung, D. S., Ng, W. W., Wang, D., Tsang, E. C., & Wang, X.-Z. (2007). Localized generalization error model and its application to architecture selection for radial basis function neural network. *IEEE Transactions on Neural Networks*, *18*(5), 1294–1305.

Yeung, K. Y., Fraley, C., Murua, A., Raftery, A. E., & Ruzzo, W. L. (2001). Model-based clustering and data transformations for gene expression data. *Bioinformatics*, *17*(10), 977–987.

Yin, M., & Zhou, M. (2018). Semi-implicit variational inference. *arXiv preprint arXiv:1805.11183*.

Yingwei, L., Sundararajan, N., & Saratchandran, P. (1998). Performance evaluation of a sequential minimal radial basis function (rbf) neural network learning algorithm. *IEEE Transactions on neural networks*, *9*(2), 308–318.

Yip, R. K. (2000). A hough transform technique for the detection of reflectional symmetry and skew-symmetry. *Pattern Recognition Letters*, *21*(2), 117–130.

You, Z., & Jain, A. K. (1984). Performance evaluation of shape matching via chord length distribution. *Computer vision, graphics, and image processing*, *28*(2), 185–198.

You, Z., Yan, K., Ye, J., Ma, M., & Wang, P. (2019). Gate decorator: Global filter pruning method for accelerating deep convolutional neural networks. *arXiv preprint arXiv:1909.08174*.

Young, S. S., Scott, P. D., & Nasrabadi, N. M. (1997). Object recognition using multilayer hopfield neural network. *IEEE Transactions on Image Processing*, *6*(3), 357–372.

Yu, J., Chaomurilige, C., & Yang, M.-S. (2018). On convergence and parameter selection of the em and da-em algorithms for gaussian mixtures. *Pattern Recognition*, *77*, 188–203.

Yu, W., & Li, X. (2001). Some new results on system identification with dynamic neural networks. *IEEE Transactions on Neural Networks*, *12*(2), 412–417.

Yu, X., Efe, M. O., & Kaynak, O. (2002). A general backpropagation algorithm for feedforward neural networks learning. *IEEE transactions on neural networks*, *13*(1), 251–254.

Yuan, B. (2016). Efficient hardware architecture of softmax layer in deep neural network. In *2016 29th IEEE International System-on-Chip Conference (SOCC)*, (pp. 323–326). IEEE.

Yuan, H., Xiong, F., & Huai, X. (2003). A method for estimating the number of hidden neurons in feed-forward neural networks based on information entropy. *Computers and Electronics in Agriculture*, *40*(1-3), 57–64.

Yuen, H., Princen, J., Illingworth, J., & Kittler, J. (1990). Comparative study of hough transform methods for circle finding. *Image and vision computing*, *8*(1), 71–77.

Yun, C., Sra, S., & Jadbabaie, A. (2018). Small nonlinearities in activation functions create bad local minima in neural networks. *arXiv preprint arXiv:1802.03487*.

Yunpeng, C., Xiaomin, S., & Peifa, J. (2006). Probabilistic modeling for continuous eda with boltzmann selection and kullback-leibeler divergence. In *Proceedings of the 8th annual conference on Genetic and evolutionary computation*, (pp. 389–396).

Zainuddin, Z., & Pauline, O. (2008). Function approximation using artificial neural networks. *WSEAS Transactions on Mathematics*, *7*(6), 333–338.

Zamanlooy, B., & Mirhassani, M. (2013). Efficient vlsi implementation of neural networks with hyperbolic tangent activation function. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, *22*(1), 39–48.

Zanella, G., et al. (2015). Random partition models and complementary clustering of anglo-saxon place-names. *The Annals of Applied Statistics*, *9*(4), 1792–1822.

Zaremba, W., Sutskever, I., & Vinyals, O. (2014). Recurrent neural network regularization. *arXiv preprint arXiv:1409.2329*.

Zarepour, M., & Al Labadi, L. (2012). On a rapid simulation of the dirichlet process. *Statistics & Probability Letters*, *82*(5), 916–924.

Zeiler, M. D., & Fergus, R. (2014). Visualizing and understanding convolutional networks. In *European conference on computer vision*, (pp. 818–833). Springer.

Zhai, K., Boyd-Graber, J., Asadi, N., & Alkhouja, M. L. (2012). Mr. lda: A flexible large scale topic modeling package using variational inference in mapreduce. In *Proceedings of the 21st International Conference on World Wide Web*, WWW '12, (pp. 879–888). New York, NY, USA: ACM.
URL `http://doi.acm.org/10.1145/2187836.2187955`

Zhang, B., Zhang, C., & Yi, X. (2004). Competitive em algorithm for finite mixture models. *Pattern recognition*, *37*(1), 131–144.

Zhang, D., Jiang, Q., & Li, X. (2007). Application of neural networks in financial data mining. *World Academy of Science, Engineering and Technology, International Journal of Computer, Electrical, Automation, Control and Information Engineering*, *1*(1), 225–228.

Zhang, G., Sun, S., Duvenaud, D., & Grosse, R. (2018a). Noisy natural gradient as variational inference. In *International Conference on Machine Learning*, (pp. 5852–5861). PMLR.

Zhang, H., Weng, T.-W., Chen, P.-Y., Hsieh, C.-J., & Daniel, L. (2018b). Efficient neural network robustness certification with general activation functions. In *Advances in neural information processing systems*, (pp. 4939–4948).

Zhang, J., Huang, Q., Wu, H., & Liu, Y. (2017). A shallow network with combined pooling for fast traffic sign recognition. *Information*, *8*(2), 45.

Zhang, S., Zhang, C., You, Z., Zheng, R., & Xu, B. (2013). Asynchronous stochastic gradient descent for dnn training. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, (pp. 6660–6663). IEEE.

Zhang, X., Ben, K., & Zeng, J. (2018c). Cross-entropy: A new metric for software defect prediction. In *2018 IEEE International Conference on Software Quality, Reliability and Security (QRS)*, (pp. 111–122). IEEE.

Zhang, Z., & Sabuncu, M. (2018). Generalized cross entropy loss for training deep neural networks with noisy labels. *Advances in neural information processing systems*, *31*, 8778–8788.

Zhao, R., Li, Y., Sun, Y., et al. (2020). Statistical convergence of the em algorithm on gaussian mixture models. *Electronic Journal of Statistics*, *14*(1), 632–660.

Zhen, X., Yu, M., He, X., & Li, S. (2017). Multi-target regression via robust low-rank learning. *IEEE transactions on pattern analysis and machine intelligence*, *40*(2), 497–504.

Zhou, M., Favaro, S., & Walker, S. G. (2017). Frequency of frequencies distributions and size-dependent exchangeable random partitions. *Journal of the American Statistical Association*, *112*(520), 1623–1635.

Zhou, R., & Hansen, E. A. (2006). Breadth-first heuristic search. *Artificial Intelligence*, *170*(4-5), 385–408.

Zhou, S., Wu, Y., Ni, Z., Zhou, X., Wen, H., & Zou, Y. (2016). Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients. *arXiv preprint arXiv:1606.06160*.

Zhou, X.-H., Zhang, M.-X., Xu, Z.-G., Cai, C.-Y., Huang, Y.-J., & Zheng, Y.-J. (2019). Shallow and deep neural network training by water wave optimization. *Swarm and Evolutionary Computation*, *50*, 100561.

Zhu, Q., Wang, X., Keogh, E., & Lee, S.-H. (2009). Augmenting the generalized hough transform to enable the mining of petroglyphs. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, (pp. 1057–1066).

Zhu, X., Sobihani, P., & Guo, H. (2015). Long short-term memory over recursive structures. In *International Conference on Machine Learning*, (pp. 1604–1612).

Zhuang, B., Liu, J., Tan, M., Liu, L., Reid, I., & Shen, C. (2019). Effective training of convolutional neural networks with low-bitwidth weights and activations. *arXiv preprint arXiv:1908.04680*.

Zhuang, B., Shen, C., Tan, M., Liu, L., & Reid, I. (2018). Towards effective low-bitwidth convolutional neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, (pp. 7920–7928).

Židek, M. (2020). Controlled music generation with deep learning.

Zinkevich, M. (2017). Rules of machine learning: Best practices for ml engineering. *URL: https://developers. google. com/machine-learning/guides/rules-of-ml*.

Zinkevich, M., Weimer, M., Li, L., & Smola, A. J. (2010). Parallelized stochastic gradient descent. In *Advances in neural information processing systems*, (pp. 2595–2603).

Zisserman, A., Forsyth, D., Mundy, J., Rothwell, C., Liu, J., & Pillow, N. (1995). 3d object recognition using invariance. *Artificial Intelligence*, *78*(1-2), 239–288.

Zivkovic, Z., & van der Heijden, F. (2004). Recursive unsupervised learning of finite mixture models. *IEEE Transactions on pattern analysis and machine intelligence*, *26*(5), 651–656.

Zorich, V. A. (2016). *Mathematical analysis II*. Springer.

Appendix A

Other Experiments Results with the Shallow Gibbs Models

Table A.1: RRMSE (%) and MSE Results with the shallow Gibbs Networks: B-sparse-Gibbs

	Shallow Gibbs settings								Measure of fitness S-Gibbs			Time (sec.)
Data sets	NH	ET	LR	NE	BS	SWbP	PSY	SPP	RMSE	RMSE	RRMSE	Time
									Train	Test	-	
Slump	1	1	$10e(-3)$	10	0.8	5	5	5	1542.67	**422.17**	399.11	2633.44
Slump	1	1	$10e(-6)$	10	0.8	5	5	5	1351.71	**527.27**	104.17	2622.03
Slump	1	1	$10e(-6)$	10	0.8	5	5	5	1815.48	**778.58**	117.12	2616.32
Slump	1	1	$10e(-6)$	10	0.8	5	5	5	3457.77	**1479.01**	175.50	716.58
EDM	1	1	$10e(-4)$	5	0.8	5	5	5	7.24	**1.27**	87.93	1160.06
EDM	1	1	$10e(-3)$	5	0.8	5	5	5	6.82	**1.55**	94.77	1143.09
EDM	1	1	$10e(-3)$	5	0.8	5	5	5	5.33	**3.37**	129.76	2414.18
EDM	1	1	$10e(-5)$	5	0.8	5	5	5	5.55	**2.50**	112.12	1155.88
EDM	2	1	$10e(-5)$	5	0.8	5	5	5	10.23	**3.07**	87.36	1549.87
EDM	10	1	$10e(-5)$	2	0.8	5	5	5	24.17	**8.92**	192.35	1299.76
Jura	1	10	$10e(-3)$	10	0.8	2	2	2	476.95	**174.69**	650.35	9833.41
SCPF	1	1	$10e(-4)$	5	0.8	5	5	5	470.14	**147.33**	**48.07**	2101.72
SCPF	1	1	$10e(-3)$	5	0.8	5	5	5	324.71	**116.63**	120.83	2083.83
SCPF	1	1	$10e(-4)$	5	0.8	5	5	5	312.39	**130.77**	91.36	4432.99
SCPF	1	2	$10e(-3)$	5	0.8	5	5	5	380.89	**137.21**	60.00	4560.25

**We have generated 5 Potts partitions with a shrinkage constraint of 5, RMSE=Root Mean Squared Error, RRMSE=Relative Root Mean Squared Error; all without the DBS optimizer and the smoother δ*

Table A.2: RRMSE (%) and MSE Results with the shallow Gibbs Networks: B-sparse-Gibbs, with different smoother δ applied

	Shallow Gibbs settings								Measure of fitness S-Gibbs			Time (sec.)
Data sets	NH	ET	LR	NE	BS	SWbP	PSY	SPP	RMSE	RMSE	RRMSE	Time
									Train	Test	-	
Slump	1	1	$10e(-3)$	10	0.8	5	5	5	82.47	**50.24**	83.74	493.65
Slump	1	1	$10e(-6)$	10	0.8	5	5	5	66.09	**67.86**	100.63	497.35
Slump	1	1	$10e(-6)$	10	0.8	5	5	5	66.87	**68.13**	100.00	491.90
Jura	1	1	$10e(-3)$	10	0.8	2	2	2	148.55	**88.09**	90.24	9312.22
Jura	1	1	$10e(-2)$	5	0.8	2	2	2	64.16	**52.02**	520.49	4305.73
Jura	10	1	$10e(-2)$	5	0.8	5	5	5	48.97	**41.33**	284.72	14395.23
SCPF	1	1	$10e(-3)$	5	0.8	5	5	5	142.82	40.06	**87.93**	1997.34
SCPF	1	1	$10e(-4)$	5	0.8	5	5	5	111.84	41.15	**97.97**	1980.34
SCPF	1	2	$10e(-3)$	5	0.8	5	5	5	380.89	137.21	**60.00**	4560.25

**We have generated 5 Potts partitions with a shrinkage constraint of 5, RMSE=Root Mean Squared Error, RRMSE=Relative Root Mean Squared Error; with different smoother δ, without the DBS optimizer*

Table A.3: RRMSE (%) and MSE Results with the shallow Gibbs Networks: Full-Gibbs

	Shallow Gibbs settings								Measure of fitness S-Gibbs			Time (sec.)
Data sets	NH	ET	LR	NE	BS	SWbP	PSY	SPP	RMSE	RMSE	RRMSE	Time
									Train	Test	-	
Slump	1	1	$10e(-3)$	5	0.8	5	5	5	1433.99	482.73	**45.64**	1782.12
Slump	10	1	$10e(-3)$	5	0.8	5	5	5	3178.67	**1330.54**	934.75	2122.28
EDM	1	1	$10e(-4)$	5	0.8	5	5	5	7.24	**1.27**	87.93	1160.06
EDM	1	1	$10e(-3)$	5	0.8	5	5	5	6.82	**1.55**	94.77	1143.09
EDM	1	1	$10e(-3)$	5	0.8	5	5	5	5.33	**3.37**	129.76	2414.18
EDM	1	1	$10e(-5)$	5	0.8	5	5	5	5.55	**2.50**	112.12	1155.88
EDM	2	1	$10e(-5)$	5	0.8	5	5	5	10.23	**3.07**	87.36	1549.87
EDM	10	1	$10e(-5)$	2	0.8	5	5	5	24.17	**8.92**	192.35	1299.76
Jura	1	3	$10e(-3)$	2	0.8	5	5	5	75.12	**50.64**	102.82	5360.25
Jura	1	3	$10e(-3)$	2	0.3	5	5	5	32.94	**35.99**	100.0	1385.67
SCPF	1	1	$10e(-4)$	5	0.8	5	5	5	470.14	147.33	**48.07**	

**We have generated 5 Potts partitions with a shrinkage constraint of 5, RMSE=Root Mean Squared Error, RRMSE=Relative Root Mean Squared Error; all without the DBS optimizer and the smoother δ*

Table A.4: RRMSE (%) and MSE Results with the shallow Gibbs Networks: Sparse-Gibbs

	Shallow Gibbs settings								Measure of fitness S-Gibbs			Time (sec.)
Data sets	NH	ET	LR	NE	BS	SWbP	PSY	SPP	RMSE	RMSE	RRMSE	Time
									Train	Test	-	
Slump												
EDM	1	1	$10e(-4)$	5	0.8	5	5	5	7.24	**1.27**	87.93	1160.06
EDM	1	1	$10e(-3)$	5	0.8	5	5	5	6.82	**1.55**	94.77	1143.09
EDM	1	1	$10e(-3)$	5	0.8	5	5	5	5.33	**3.37**	129.76	2414.18
EDM	1	1	$10e(-5)$	5	0.8	5	5	5	5.55	**2.50**	112.12	1155.88
EDM	2	1	$10e(-5)$	5	0.8	5	5	5	10.23	**3.07**	87.36	1549.87
EDM	10	1	$10e(-5)$	2	0.8	5	5	5	24.17	**8.92**	192.35	1299.76
SCPF	1	1	$10e(-4)$	5	0.8	5	5	5	470.14	147.33	**48.07**	

**We have generated 5 Potts partitions with a shrinkage constraint of 5, RMSE=Root Mean Squared Error, RRMSE=Relative Root Mean Squared Error; all without the DBS optimizer and the smoother δ*

Appendix B

Some Statistical Tables

Table of Common Distributions

taken from *Statistical Inference* by Casella and Berger

Discrete Distrbutions

distribution	pmf	mean	variance	mgf/moment
Bernoulli(p)	$p^x(1-p)^{1-x};\ x=0,1;\ p\in(0,1)$	p	$p(1-p)$	$(1-p)+pe^t$
Beta-binomial(n,α,β)	$\binom{n}{x}\frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)}\frac{\Gamma(x+\alpha)\Gamma(n-x+\beta)}{\Gamma(\alpha+\beta+n)}$	$\frac{n\alpha}{\alpha+\beta}$	$\frac{n\alpha\beta}{(\alpha+\beta)^2}$	
Notes: If $X\|P$ is binomial (n,P) and P is beta(α,β), then X is beta-binomial(n,α,β).				
Binomial(n,p)	$\binom{n}{x}p^x(1-p)^{n-x};\ x=1,\dots,n$	np	$np(1-p)$	$[(1-p)+pe^t]^n$
Discrete Uniform(N)	$\frac{1}{N};\ x=1,\dots,N$	$\frac{N+1}{2}$	$\frac{(N+1)(N-1)}{12}$	$\frac{1}{N}\sum_{i=1}^{N}e^{it}$
Geometric(p)	$p(1-p)^{x-1};\ p\in(0,1)$	$\frac{1}{p}$	$\frac{1-p}{p^2}$	$\frac{pe^t}{1-(1-p)e^t}$
Note: $Y=X-1$ is negative binomial$(1,p)$. The distribution is *memoryless*: $P(X>s\|X>t)=P(X>s-t)$.				
Hypergeometric(N,M,K)	$\frac{\binom{M}{x}\binom{N-M}{K-x}}{\binom{N}{K}};\ x=1,\dots,K$	$\frac{KM}{N}$	$\frac{KM}{N}\frac{(N-M)(N-k)}{N(N-1)}$	?
	$M-(N-K)\le x\le M;\ N,M,K>0$			
Negative Binomial(r,p)	$\binom{r+x-1}{x}p^r(1-p)^x;\ p\in(0,1)$	$\frac{r(1-p)}{p}$	$\frac{r(1-p)}{p^2}$	$\left(\frac{p}{1-(1-p)e^t}\right)^r$
	$\binom{y-1}{r-1}p^r(1-p)^{y-r};\ Y=X+r$			
Poisson(λ)	$\frac{e^{-\lambda}\lambda^x}{x!};\ \lambda\ge 0$	λ	λ	$e^{\lambda(e^t-1)}$
Notes: If Y is gamma(α,β), X is Poisson$(\frac{x}{\beta})$, and α is an integer, then $P(X\ge\alpha)=P(Y\le y)$.				

Continuous Distributions

distribution	pdf	mean	variance	mgf/moment
Beta(α,β)	$\frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)}x^{\alpha-1}(1-x)^{\beta-1};\ x\in(0,1),\ \alpha,\beta>0$	$\frac{\alpha}{\alpha+\beta}$	$\frac{\alpha\beta}{(\alpha+\beta)^2(\alpha+\beta+1)}$	$1+\sum_{k=1}^{\infty}\left(\prod_{r=0}^{k-1}\frac{\alpha+r}{\alpha+\beta+r}\right)\frac{t^k}{k!}$
Cauchy(θ,σ)	$\frac{1}{\pi\sigma}\frac{1}{1+(\frac{x-\theta}{\sigma})^2};\ \sigma>0$	does not exist	does not exist	does not exist
Notes: Special case of Students's t with 1 degree of freedom. Also, if X,Y are iid $N(0,1)$, $\frac{X}{Y}$ is Cauchy				
χ^2_p	$\frac{1}{\Gamma(\frac{p}{2})2^{\frac{p}{2}}}x^{\frac{p}{2}-1}e^{-\frac{x}{2}};\ x>0,\ p\in N$	p	$2p$	$\left(\frac{1}{1-2t}\right)^{\frac{p}{2}},\ t<\frac{1}{2}$
Notes: Gamma$(\frac{p}{2},2)$.				
Double Exponential(μ,σ)	$\frac{1}{2\sigma}e^{-\frac{\lvert x-\mu\rvert}{\sigma}};\ \sigma>0$	μ	$2\sigma^2$	$\frac{e^{\mu t}}{1-(\sigma t)^2}$
Exponential(θ)	$\frac{1}{\theta}e^{-\frac{x}{\theta}};\ x\geq 0,\ \theta>0$	θ	θ^2	$\frac{1}{1-\theta t},\ t<\frac{1}{\theta}$
Notes: Gamma$(1,\theta)$. Memoryless. $Y=X^{\frac{1}{\gamma}}$ is *Weibull*. $Y=\sqrt{\frac{2X}{\beta}}$ is *Rayleigh*. $Y=\alpha-\gamma\log\frac{X}{\beta}$ is *Gumbel*.				
F_{ν_1,ν_2}	$\frac{\Gamma\left(\frac{\nu_1+\nu_2}{2}\right)}{\Gamma(\frac{\nu_1}{2})\Gamma(\frac{\nu_2}{2})}\left(\frac{\nu_1}{\nu_2}\right)^{\frac{\nu_1}{2}}\frac{x^{\frac{\nu_1-2}{2}}}{\left(1+(\frac{\nu_1}{\nu_2})x\right)^{\frac{\nu_1+\nu_2}{2}}};\ x>0$	$\frac{\nu_2}{\nu_2-2},\ \nu_2>2$	$2(\frac{\nu_2}{\nu_2-2})^2\frac{\nu_1+\nu_2-2}{\nu_1(\nu_2-4)},\ \nu_2>4$	$EX^n=\frac{\Gamma(\frac{\nu_1+2n}{2})\Gamma(\frac{\nu_2-2n}{2})}{\Gamma(\frac{\nu_1}{2})\Gamma(\frac{\nu_2}{2})}\left(\frac{\nu_2}{\nu_1}\right)^n,\ n<\frac{\nu_2}{2}$
Notes: $F_{\nu_1,\nu_2}=\frac{\chi^2_{\nu_1}/\nu_1}{\chi^2_{\nu_2}/\nu_2}$, where the χ^2s are independent. $F_{1,\nu}=t^2_\nu$.				
Gamma(α,β)	$\frac{1}{\Gamma(\alpha)\beta^\alpha}x^{\alpha-1}e^{-\frac{x}{\beta}};\ x>0,\ \alpha,\beta>0$	$\alpha\beta$	$\alpha\beta^2$	$\left(\frac{1}{1-\beta t}\right)^\alpha,\ t<\frac{1}{\beta}$
Notes: Some special cases are exponential ($\alpha=1$) and χ^2 ($\alpha=\frac{p}{2},\beta=2$). If $\alpha=\frac{2}{3},\ Y=\sqrt{\frac{X}{\beta}}$ is *Maxwell*. $Y=\frac{1}{X}$ is *inverted gamma*.				
Logistic(μ,β)	$\frac{1}{\beta}\frac{e^{-\frac{x-\mu}{\beta}}}{\left[1+e^{-\frac{x-\mu}{\beta}}\right]^2};\ \beta>0$	μ	$\frac{\pi^2\beta^2}{3}$	$e^{\mu t}\Gamma(1+\beta t),\ \lvert t\rvert<\frac{1}{\beta}$
Notes: The cdf is $F(x\lvert\mu,\beta)=\frac{1}{1+e^{-\frac{x-\mu}{\beta}}}$.				
Lognormal(μ,σ^2)	$\frac{1}{\sqrt{2\pi}\sigma}\frac{1}{x}e^{-\frac{(\log x-\mu)^2}{2\sigma^2}};\ x>0,\sigma>0$	$e^{\mu+\frac{\sigma^2}{2}}$	$e^{2(\mu+\sigma^2)}-e^{2\mu+\sigma^2}$	$EX^n=e^{n\mu+\frac{n^2\sigma^2}{2}}$
Normal(μ,σ^2)	$\frac{1}{\sqrt{2\pi}\sigma}e^{-\frac{(x-\mu)^2}{2\sigma^2}};\ \sigma>0$	μ	σ^2	$e^{\mu t+\frac{\sigma^2t^2}{2}}$
Pareto(α,β)	$\frac{\beta\alpha^\beta}{x^{\beta+1}};\ x>\alpha,\ \alpha,\beta>0$	$\frac{\beta\alpha}{\beta-1},\ \beta>1$	$\frac{\beta\alpha^2}{(\beta-1)^2(\beta-2)},\ \beta>2$	does not exist
t_ν	$\frac{\Gamma(\frac{\nu+1}{2})}{\Gamma(\frac{\nu}{2})}\frac{1}{\sqrt{\nu\pi}}\frac{1}{(1+\frac{x^2}{\nu})^{\frac{\nu+1}{2}}}$	$0,\ \nu>1$	$\frac{\nu}{\nu-2},\nu>2$	$EX^n=\frac{\Gamma(\frac{\nu+1}{2})\Gamma(\nu-\frac{n}{2})}{\sqrt{\pi}\Gamma(\frac{\nu}{2})}\nu^{\frac{n}{2}},\ n$ even
Notes: $t^2_\nu=F_{1,\nu}$.				
Uniform(a,b)	$\frac{1}{b-a},\ a\leq x\leq b$	$\frac{b+a}{2}$	$\frac{(b-a)^2}{12}$	$\frac{e^{bt}-e^{at}}{(b-a)t}$
Notes: If $a=0,\ b=1$, this is special case of beta ($\alpha=\beta=1$).				
Weibull(γ,β)	$\frac{\gamma}{\beta}x^{\gamma-1}e^{-\frac{x^\gamma}{\beta}};\ x>0,\ \gamma,\beta>0$	$\beta^{\frac{1}{\gamma}}\Gamma(1+\frac{1}{\gamma})$	$\beta^{\frac{2}{\gamma}}\left[\Gamma(1+\frac{2}{\gamma})-\Gamma^2(1+\frac{1}{\gamma})\right]$	$EX^n=\beta^{\frac{n}{\gamma}}\Gamma(1+\frac{n}{\gamma})$
Notes: The mgf only exists for $\gamma\geq 1$.				