IMPROVING THE LEARNING PERFORMANCE OF THE RESTRICTED BOLTZMANN MACHINE THROUGH OPTIMAL CONNECTIVITY AND NETWORK GRADIENTS

Amanda Camacho Novaes de Oliveira

Dissertação de Mestrado apresentada ao Programa de Pós-graduação em Engenharia de Sistemas e Computação, COPPE, da Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Mestre em Engenharia de Sistemas e Computação.

Orientador: Daniel Ratton Figueiredo

Rio de Janeiro
Março de 2022

IMPROVING THE LEARNING PERFORMANCE OF THE RESTRICTED BOLTZMANN MACHINE THROUGH OPTIMAL CONNECTIVITY AND NETWORK GRADIENTS

Amanda Camacho Novaes de Oliveira

DISSERTAÇÃO SUBMETIDA AO CORPO DOCENTE DO INSTITUTO ALBERTO LUIZ COIMBRA DE PÓS-GRADUAÇÃO E PESQUISA DE ENGENHARIA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

Orientador: Daniel Ratton Figueiredo

Aprovada por: Prof. Daniel Ratton Figueiredo
Prof. Carlos Eduardo Pedreira
Prof.ª Elizabeth Fialho Wanner

RIO DE JANEIRO, RJ – BRASIL
MARÇO DE 2022

Oliveira, Amanda Camacho Novaes de

Improving the Learning Performance of the Restricted Boltzmann Machine through Optimal Connectivity and Network Gradients/Amanda Camacho Novaes de Oliveira. – Rio de Janeiro: UFRJ/COPPE, 2022.

XV, 56 p.: il.; 29, 7cm.

Orientador: Daniel Ratton Figueiredo

Dissertação (mestrado) – UFRJ/COPPE/Programa de Engenharia de Sistemas e Computação, 2022.

Referências Bibliográficas: p. 49 – 53.

1. Neural Networks. 2. Restricted Boltzmann Machine. 3. Network Connectivity. 4. Network Pruning. 5. Neural Architecture Search. 6. Connectivity Optimization. 7. AutoML. I. Figueiredo, Daniel Ratton. II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia de Sistemas e Computação. III. Título.

Continue a nadar

Agradecimentos

Agradeço muito aos meus pais, Izo e Márcia, que sempre me apoiaram muito, e também aos meus irmãos, Iasmin e Tiago. Obrigada por me aturarem nesses dois anos de pandemia, não estaria aqui sem vocês.

A todos os meus familiares, avós, tios, primos, obrigada pelo carinho e ajuda. Sei que aluguei alguns ouvidos de vez em quando, como o Rafa e o Leozinho podem atestar. Aos meus amigos, vocês também foram fundamentais. Em especial, a Amanda Azevedo, o Vinícius Garcia e o Diego Amaro me ajudaram muito durante o mestrado, superamos algumas adversidades juntos. E, claro, não pode faltar aquela menção aos meus amigos da T-18, sempre lá para dar aquela descontraída.

Agradeço também à UFRJ, pelas oportunidades e experiências. Em especial, agradeço também ao meu orientador, Daniel, sem o qual este trabalho não teria nem sequer começado.

Resumo da Dissertação apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (M.Sc.)

APRIMORAMENTO DO DESEMPENHO DO APRENDIZADO DA RESTRICTED BOLTZMANN MACHINE POR MEIO DE OTIMIZAÇÃO DA CONECTIVIDADE E GRADIENTES DA REDE

Amanda Camacho Novaes de Oliveira

Março/2022

Orientador: Daniel Ratton Figueiredo

Programa: Engenharia de Sistemas e Computação

A despeito das técnicas de Busca por Arquitetura Neural (NAS) e de Poda de Redes terem sido recentemente redescobertas como estratégias poderosas para a criação de redes neurais mais eficientes e com menos parâmetros, o foco tem sido em melhorar modelos de redes neurais profundas, com milhões de parâmetros. Entretanto, a conectividade da rede também tem papel fundamental no desempenho do aprendizado de redes rasas, como o modelo da Máquina de Boltzmann Restrita (RBM). Este trabalho apresenta um estudo do espaço de conectividade tal como ele afeta o aprendizado da RBM, além de propor um método para encontrar padrões de conectividade ótimos: o Gradiente de Conectividade da Rede (NCG). O NCG é baseado na ideia de gradientes da rede: ele computa o gradiente de cada conexão em potencial, dada a conectividade atual, e usa esse gradiente para atualizar o parâmetro, contínuo, da força da conexão, que por sua vez é usado para atualizar a conectividade em si. Dessa forma, o aprendizado dos parâmetros tradicionais da RBM e das conexões é realizado concomitantemente, mesmo que com taxas de aprendizados diferentes, e sem alteração na função objetivo do modelo. O método é aplicado aos dados BAS e MNIST, gerando modelos melhores de RBMs para as tarefas de geração de amostras e classificação de dados. Ademais, a rede completamente conectada tem desempenho superado tanto por padrões criados manualmente quanto pelo NCG para ambos os conjuntos de dados, ilustrando a importância de projetarmos padrões de conectividade que levem a modelos de maior acurácia até para redes neurais simples de duas camadas.

Abstract of Dissertation presented to COPPE/UFRJ as a partial fulfillment of the requirements for the degree of Master of Science (M.Sc.)

IMPROVING THE LEARNING PERFORMANCE OF THE RESTRICTED BOLTZMANN MACHINE THROUGH OPTIMAL CONNECTIVITY AND NETWORK GRADIENTS

Amanda Camacho Novaes de Oliveira

March/2022

Advisor: Daniel Ratton Figueiredo

Department: Systems Engineering and Computer Science

While Network Architecture Search (NAS) and Network Pruning have recently re-emerged as powerful techniques to design more effective networks with less parameters, their focus has been on improving deep neural network models with millions of parameters. However, network connectivity also plays a significant role on the learning performance of shallow two-layer networks, such as the classic Restricted Boltzmann Machine (RBM). This work presents a comprehensive study of the connectivity space on the learning performance of RBMs, as well as a method to find optimal connectivity patterns for them: Network Connectivity Gradient (NCG). NCG is based on the idea of network gradients: it computes the gradient of every possible connection, given a specific connection pattern, and uses the gradient to drive a continuous connection strength parameter that in turn is used to determine the connection pattern. Thus, learning RBM parameters and learning network connections is truly jointly performed, albeit using different learning rates, and with no changes to the objective function of the model. The method is employed on the BAS and MNIST datasets showing that better RBM models are found for the benchmark tasks of sample generation and input classification. Moreover, the fully connected network is outperformed both by manually designed connectivity patterns and NCG for the considered datasets, indicating the importance of designing more effective connectivity patterns even for simple two-layer neural networks.

Contents

List of Figures

List of Tables

List of Symbols

mod	Modulo function, p. 21
f	Optimization objective function (in this work the average NLL is used), p. 9
E	Energy of configuration $(\mathbf{x}, \mathbf{h})$ of a RBM, p. 7
F	Free energy of a configuration $(\mathbf{x})$ of a RBM, p. 7
H	Number of hidden units of the RBM, p. 6
X	Number of visible units of the RBM, p. 6
α	Learning rate of RBM training, p. 10
α_A	Connectivity learning rate for NCG training, p. 32
γ	Connectivity learning threshold for NCG training, p. 32
$\hat{\mathbf{h}}$	Auxiliar function in CD training, p. 10
$\mathcal{B}$	Batch of data samples, p. 10
$\mathbb{1}$	Step (or indicator) function, p. 32
σ	Sigmoid function, p. 10
$\mathbf{A}$	Adjacency matrix of the RBM, p. 13
$\mathbf{A}'$	Connectivity strength parameters, p. 32
$\mathbf{C}$	Acting weights of RBM with arbitrary connectivity pattern, p. 14
$\mathbf{W}$	Weights of the RBM, p. 7
$\mathbf{b}$	Biases of the hidden units of the RBM, p. 7
$\mathbf{d}$	Biases of the visible units of the RBM, p. 7

$\mathbf{h}$	Hidden units of the RBM, p. 6
$\mathbf{x}$	Visible units of the RBM, p. 6
θ	Set of RBM parameters, p. 7

List of Abbreviations

BAS	Bars And Stripes, p. 11
CD	Contrastive Divergence, p. 10
CNN	Convolutional Neural Network, p. 1
DBN	Deep Belief Network, p. 8
DL	Deep Learning, p. 6
MC	Monte Carlo (as in Monte Carlo algorithms), p. 14
ML	Machine Learning, p. 9
NAS	Neural Architecture Search, p. 5
NCG	Network Connectivity Gradient, p. 31
NLL	Negative Log-Likelihood, p. 9
RBM	Restricted Boltzmann Machine, p. 6
SGD	Stochastic Gradient Descent, p. 9

Chapter 1

Introduction

With the high demand for intelligent algorithms that can solve increasingly hard problems, deep neural networks have become the standard tool for a variety of problems in areas such as computer vision and natural language processing, for example [1, 2]. In order to excel in specific tasks, different network architectures have been manually designed such as ResNets [3] and BERT [4]. However, designing high performance neural networks is non-trivial, which has prompted the automation of network design in an area known as Neural Architecture Search (NAS) [5].

Despite the advancements, NAS focuses on the interconnection of network layers (modules) and the corresponding activation/aggregation functions. In particular, when two layers are to be connected, a fully connected network is often adopted: all outputs of one layer are connected to all inputs of the next layer. Thus, NAS focuses on macroscopic aspects of the neural network.

However, it is known that the connection pattern between two layers is of fundamental importance for the learning performance of the network. Fukushima & Miyake [6] who also introduced convolutional layers, emphasized decades ago the role of the connectivity pattern within a layer in order to enable the network to recognize visual patterns even when they are deformed or shifted in position within the image. Since their seminal work a wide variety of Convolutional Neural Networks (CNN) have been designed, and deep CNNs are often easier to train and generalize better than their fully connected counterparts [7].

Indeed, while most neural network architectures adopt a fully connected network between units of successive layers, the significance of the connectivity has been long recognized, for it is not only able to reduce the number of parameters but also lead to more accurate models or faster learning [8, 9]. In the last years, these findings have reemerged in the context of deep neural networks, and while classic architectures have millions of parameters that must be learned, recent works indicate that only a small fraction is necessary for the model to attain a similar performance given an equivalent training effort [9].

While both NAS and network pruning have been applied to deep neural networks, the problem of designing more effective neural networks is also relevant on shallow networks. Consider a classic two-layer Restricted Boltzmann Machine (RBM). Although past works have addressed the problem of determining the number of neurons in the hidden layer [10], the connectivity pattern between the input and hidden layers has not been addressed, to the best of our knowledge.

However, the connectivity pattern between layers influences the learning performance of the network, even for shallow networks such as the RBM. Intuitively, too few connections are likely not enough for an effective learning, while too many connections may require a longer training period. Moreover, the connectivity pattern, and not just the number of connections, is likely to play a fundamental role. Finally, the connectivity space is extremely large: there are 2^{n^2} different ways to connect two layers with n neurons each, indicating the complexity of finding connection patterns. In fact, the connectivity of an RBM can be interpreted as a hyperparameter, just as the number of neurons in its hidden layer, which is known to also influence its performance [10, 11].

Therefore, the goal of this dissertation is to explore and design network connectivity patterns for RBMs in order to characterize the learning performance as a function of the network. How sensible are RBMs to connectivity patterns? Are there optimal connectivity networks for a given task? Is it possible to learn good connection patterns from the data? This work addresses these and other questions, in order to answer them in a satisfactory manner.

1.1 Contributions

The contributions of this work are manifold, and they are divided into the sub-topics listed below.

Connectivity search space A comprehensive study of the connectivity search space for the RBM is presented, using the synthetic Bars And Stripes (BAS) model and the MNIST dataset. The main findings are summarized as follows:

- Using different connectivity patterns with a parameter that determines the number of connections in the pattern, results on BAS indicate that learning performance is not monotonic with the number of connections. For each pattern, there is an optimal number of connections.
- Using different connectivity patterns with the same number of connections, results on BAS clearly indicate that the patterns play a fundamental role when the number of connections is small. However, the performance difference

among different patterns diminishes as the number of connections increase. Moreover, more localized connectivity patterns show superior performance for the same number of connections.

- Results on the MNIST dataset indicate that the number of connections plays a role more important than the connectivity pattern, and performance is superior with only 16 connections per unit (as opposed to 784). However, once again the importance of the connectivity pattern increases with the decrease in the number of connections. Furthermore, the results indicate that sparser connectivity patterns often lead not only to better performance, but also to faster learning (achieve peak performance in less epochs).

Network Connectivity Gradient A novel method is proposed for optimizing the connectivity structure in RBMs *during* the learning procedure based on the notion of "network gradients", the Network Connectivity Gradient (NCG). It computes the gradient for every possible network connection for any given connectivity pattern and uses a continuous parameter to represent the strength of each possible connection. The network strengths are updated according to the gradient and then thresholded to yield a discrete connectivity pattern which in turn determines how information (probabilities) and gradients flow on the model during training. The implementation is open source, and available at `https://github.com/AmieOliveira/NCG`.

Furthermore, evaluations on both BAS and MNIST are presented. For MNIST, two orthogonal tasks are used to access RBMs: sample generation (average NLL is the performance metric) and input classification (classification accuracy is the performance metric). In both tasks NCG shows a superior learning curve, both learning faster and learning a more accurate model than a classic fully connected RBM. The evaluation also shows that NCG removes and adds network connections during training, indicating its effectiveness in searching for optimal network patterns.

Generative performance estimation We propose two techniques to approximate and estimate the RBM's normalization constant and, therefore, its learning performance via the classic Negative Log-Likelihood (NLL). A comparison of the proposed techniques with the traditionally used Annealed Importance Sampling (AIS) [12] is presented under different datasets.

1.2 Organization

This work is organized in 5 additional chapters. In Chapter 2 the subject of NAS, Pruning and RBMs are discussed, as well as the datasets used in the experiments.

It presents the necessary background information and related works that this dissertation builds upon.

Chapter 3 starts this work's contributions by discussing the modifications made upon the RBM to allow for the change in the connectivity network between its layers, as well as methods developed to estimate its partition function (normalization constant). This is an important task in the evaluation of the learning performance of RBMs, especially when the model is used for generative purposes.

In order to understand the role of network connectivity on the learning performance of RBMs, Chapter 4 presents a comprehensive study of the connectivity space using the (BAS) model and the MNIST dataset. The findings corroborate the importance of designing connectivity patterns that exhibit effective learning performance even when considering simple two-layer neural networks such as RBMs, which in turns motivates designing a method that is capable of finding the optimal connectivity network.

Therefore, Chapter 5 proposes a novel method tailored to RBMs that is based on the notion of "network gradients", the Network Connectivity Gradient (NCG). Beyond the proposal, the method is evaluated with the BAS model and the MNIST dataset on two tasks: sample generation and input classification. In both tasks NCG shows a superior learning curve, both learning faster and learning a more accurate model than a classic fully connected RBM.

Last, Chapter 6 has the concluding remarks, reviewing this works' key aspects and proposing future work directions.

Chapter 2

Background and Datasets

2.1 Neural Architecture Search

Deep neural networks have become the state-of-the-art paradigm for tackling hard problems in areas such as computer vision and natural language processing [1, 2, 13, 14]. Traditionally, new architectures are designed by human experts, a process that is time consuming and prone to errors. In response to the rising need, the field of Neural Architecture Search (NAS) has been steadily growing in recent years [5].

The recent developments in NAS focus on the design of more effective deep neural networks that are tailored to a specific task. The goal of most approaches is to determine the macro arrangement of network layers as well as the type of operation (aggregation/activation) applied by each layer [15] which in turn determines how information flows through the model. The DARTS algorithm [16], for example, searches for cells with a pre-defined number of layers and afterwards creates a deep architecture by stacking the cells in a bigger network. Most current benchmarks follow this particular cell search trend [17, 18].

Some works broaden their search. For example, Fang et al. [19] proposes a densely connected search space that can generate networks without fixing a priori the number and size of the layers, and thus can generate more diverse networks. Even then, despite the increased flexibility of recent advancements, there is no inclusion of the connectivity pattern between consecutive layers in the search space (other than pre-defined aggregation patterns, such as convolutions).

2.2 Network Pruning

Independently of NAS, the idea of removing (pruning) connections between two adjacent network layers has recently reemerged in the context of deep neural networks [9]. It is a known rule for good generalization, that the simplest model to

fit the data be used [20], and the concept has been much utilized in the context of Deep Learning (DL). Pruning can improve the model's learning curve (learning faster and/or better) and drastically reduce the number of model parameters [8, 9, 21].

Finding the best[1] connectivity pattern for two adjacent layers is not a trivial task. Most approaches start with dense networks and iterate in rounds of training the model parameters and using the parameter values (and the input samples) to prune network connections [22, 23]. However, recent works indicate that it is possible to train sparse networks from the beginning given a favorable weight initialization [24] or identifying more relevant connections (and pruning others) using the data prior to training [25, 26].

A more principled yet less explored approach explicitly includes the network connectivity as a parameter of the model. Thus, the network connectivity becomes part of the optimization problem. However, this often requires increasing the number of parameters and modifying the objective function (in order to induce pruning). A prominent example is Continuous Sparsification [27] that uses continuous parameters and continuous functions to approximate the discrete nature of network connections, and adds a penalization term to the objective function. The discrete network connectivity is determined at the end of training rounds. UGS [28] deploys a similar approach tailored to Graph Neural Networks (GNN). While SR-STE [29] differs by evolving the (discrete) connectivity pattern at each iteration, the method considers $N : M$ sparse neural networks where N and M are hyperparameters of the model (sparsity is predefined, and the connectivity pattern must be learned).

All prior works mentioned above focus on deep neural networks, which is indeed the main focus of Network Pruning. However, network connectivity also plays a fundamental role on simple two-layer networks, such as the Restricted Boltzmann Machine (RBM), as will be shown in this work.

2.3 The Restricted Boltzmann Machine

The Restricted Boltzmann Machine (RBM), first proposed by Smolensky under the name *Harmonium* [30], is an energy-based model for unsupervised learning, a classic network architecture that has been widely explored and applied in literature [11, 31].

An RBM is a probabilistic model that can be interpreted as a stochastic neural network composed of two layers of binary units: one visible $\mathbf{x}$ of size X, representing the data, and one hidden (or latent) $\mathbf{h}$ of size H, that extracts characteristics and increases learning ability. The two layers are fully connected through undirected weighted connections in a bipartite network. Figure 2.1 shows the example of an RBM network with $X = 4$ and $H = 5$.

[1] One usually defines the best models as the ones that learn faster or more accurately

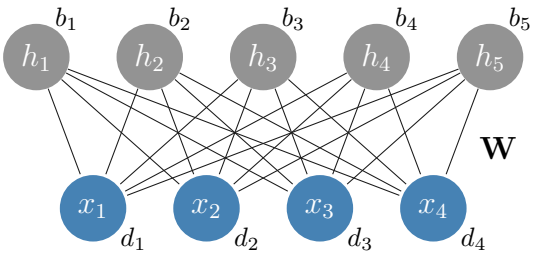


Figure 2.1: RBM network graph for 4 visible and 5 hidden units

Each configuration $(\mathbf{x}, \mathbf{h})$ has an associated energy, defined as:

$$E(\mathbf{x}, \mathbf{h}) = -\mathbf{h}^{\mathrm{T}}\mathbf{W}\mathbf{x} - \mathbf{x}^{\mathrm{T}}\mathbf{d} - \mathbf{h}^{\mathrm{T}}\mathbf{b}, \tag{2.1}$$

for which $\mathbf{W} \in \mathbb{R}^{H,X}$ is the weight matrix of the layers' connections (w_{ij} is the weight between visible unit x_j and hidden unit h_i), $\mathbf{d} \in \mathbb{R}^X$ is the visible units' bias vector (d_j is the bias for x_j) and $\mathbf{b} \in \mathbb{R}^H$ is the hidden units' bias vector (b_i is the bias for h_i). Note that $\mathbf{W}$, $\mathbf{d}$ and $\mathbf{b}$ are the model parameters, subsequently denoted by $\theta = (\mathbf{W}, \mathbf{d}, \mathbf{b})$.

The probability distribution of the RBM is given by the Gibbs distribution [32], and is defined as

$$P_\theta(\mathbf{x}, \mathbf{h}) = Z^{-1} e^{-E(\mathbf{x},\mathbf{h})}, \tag{2.2}$$

with Z being the normalization constant (or partition function), given by $Z = \sum_{\mathbf{x},\mathbf{h}} e^{-E(\mathbf{x},\mathbf{h})}$. Note that this equation is in general not tractable due to the very large number of configurations, which is given by 2^{X+H} since all units are binary.

From this formula, the marginal distribution $P_\theta(\mathbf{x}) = Z^{-1}e^{-F(\mathbf{x})}$ can be derived, as well as the conditional distributions $P_\theta(\mathbf{x}|\mathbf{h}) = \prod_{j=1}^{X} P_\theta(x_j|\mathbf{h})$ and $P_\theta(\mathbf{h}|\mathbf{x}) = \prod_{i=1}^{H} P_\theta(h_i|\mathbf{x})$, which are used for the actual training of the model [11].

2.3.1 Usage

Before attempting to train an RBM, it is important to know how the model is to be used. An RBM is capable of learning the underlying probability distribution of a dataset. Thus, it is a generative model, and allows for sampling from the learned distribution [31, 33]. One can use it to create artificial data or to complete corrupted data, for example [34]. This more traditional use of the RBM can be observed in Figure 2.2.

Another use of the RBM is as a classifier [35, 36]. One can train the model using the joint probability distribution of the data and labels, both represented by the RBM's visible units, without changing the training algorithm. The resulting visible units should be a concatenation of the data sample and its corresponding label. It

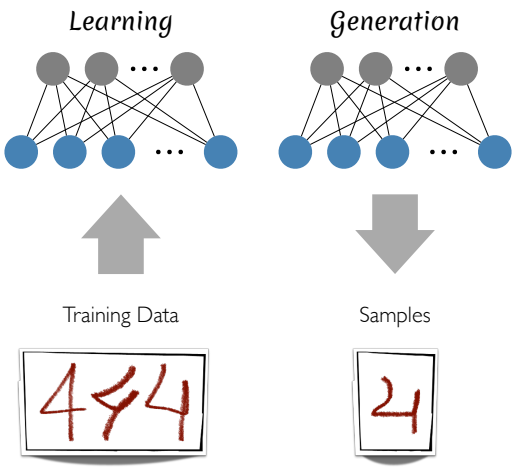


Figure 2.2: Illustration of RBM usage for sample generation. During training, data is given to the RBM, that learns its distribution (left). Once trained, it can be used to generate samples (right).

is usual to represent the label units with one-hot-encoding [36], which means each unit corresponds to a possible label and has value 1 when the data label matches it (or 0 otherwise).

To classify the data, similarly to completing corrupted data, one needs to fix the visible units that correspond to the input data, and sample the units that correspond to the label. Figure 2.3 shows an illustration of a RBM used for classification.

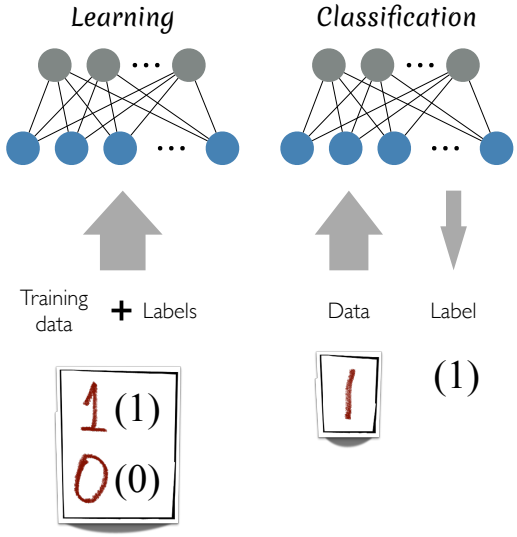


Figure 2.3: Illustration of RBM usage for classification. During training, the aggregated data/label pair is given to the RBM, that learns its distribution (left). Once trained, it can be used to classify samples by fixing the data visible units and sampling the label units (right).

Third, a common use of RBMs is as a pre-processing model. The hidden units of trained RBMs represent relevant features of the data, and thus they can serve as input for other architectures [37]. By stacking RBMs in this way we obtain Deep

Belief Networks (DBN), which have been widely used in the literature [12, 38]. This use case is illustrated in Figure 2.4.

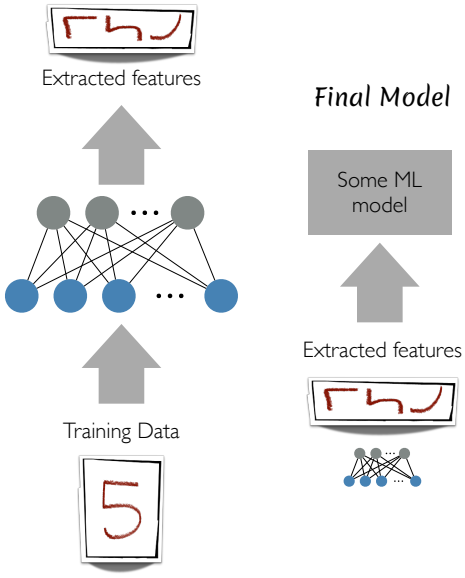


Figure 2.4: Illustration of RBM usage for pre-processing. During training, the data is given to the RBM to learn its distribution, which makes the hidden units capable of extracting features from the data (left). Once trained, other models can be stacked upon it to form a more complex architecture (right).

Despite the different ways one can utilize the RBM, in this work, only the first two approaches are implemented and evaluated.

2.3.2 Training the Model

The RBM is typically trained to minimize the Negative Log-Likelihood (NLL) of the available dataset, which is equivalent to maximizing the Log-Likelihood. In this case, the average NLL is often adopted in order to simplify the learning procedure. Given a dataset $\{x^{(t)}\}_{t=1,\dots,T}$ with T samples, the average NLL of the model, which corresponds to the objective function, is given by

$$f(\theta) = \frac{1}{T} \sum_{t=1}^{T} - \ln P_\theta(\mathbf{x}^{(t)}) \,. \tag{2.3}$$

Note that the probability $P_\theta(\mathbf{x}^{(t)})$ depends on the model parameters, θ.

The RBM is trained by applying Stochastic Gradient Descent (SGD) [39] to its parameters. The gradient of the model is given by:

$$\nabla_\theta f(\theta) = \frac{1}{|\mathcal{B}|} \sum_{t \in \mathcal{B}} \mathbb{E}_{\mathbf{h}} \left[\nabla_\theta E(\mathbf{x}, \mathbf{h}) \left| \mathbf{x} = \mathbf{x}^{(t)} \right. \right] - \mathbb{E}_{\mathbf{x},\mathbf{h}} \left[\nabla_\theta E(\mathbf{x}, \mathbf{h}) \right] \,, \tag{2.4}$$

where $\mathcal{B}$ corresponds to a batch of samples randomly chosen from the dataset, since computing the gradients using batches rather than the full data often leads to better learning curves [40]. However, Equation 2.4 is usually intractable due to the second expectation, which requires computing a sum with 2^{X+H} terms (the space of possible configurations).

In order to overcome this intractability, several methods use gradient approximations. One such method is the Contrastive Divergence algorithm (CD) [41], which substitutes the expectation of the normalization constant with an expectation over the RBM probability distribution. The gradient under CD is given by

$$\nabla_\theta f(\theta) \approx \frac{1}{|\mathcal{B}|} \sum_{t \in \mathcal{B}} \Big(\mathbb{E}_{\mathbf{h}} \left[\nabla_\theta E(\mathbf{x}, \mathbf{h}) \left| \mathbf{x} = \mathbf{x}^{(t)} \right. \right] - \mathbb{E}_{\mathbf{h}} \left[\nabla_\theta E(\mathbf{x}, \mathbf{h}) \left| \mathbf{x} = \tilde{\mathbf{x}}^{(t)} \right. \right] \Big), \quad (2.5)$$

where $\tilde{\mathbf{x}}^{(t)}$ is a random sample of the RBM given its parameters. Note that Equation 2.5 requires generating a random sample from the RBM distribution for each data sample $\mathbf{x}^{(t)}$. In the original CD algorithm, $\tilde{\mathbf{x}}^{(t)}$ is generated applying k steps of Gibbs Sampling on the model, starting from the data sample $\mathbf{x}^{(t)}$.

There are other proposed methods that yield better approximations by changing how one obtains $\tilde{\mathbf{x}}^{(t)}$ [11, 35], but in this work only the traditional CD method was used. Note that it is also possible to change the training objective function, for example, to account for a classification task [36].

Calculating the corresponding expectations for each model parameter w_{ij}, b_i, d_j, the resulting update rules of the SGD are given by:

$$\mathbf{W} \leftarrow \mathbf{W} + \alpha \frac{1}{|\mathcal{B}|} \sum_{t \in \mathcal{B}} \left(\hat{\mathbf{h}}(\mathbf{x}^{(t)}) \mathbf{x}^{(t)^{\mathrm{T}}} - \hat{\mathbf{h}}(\tilde{\mathbf{x}}^{(t)}) \tilde{\mathbf{x}}^{(t)^{\mathrm{T}}} \right) \quad (2.6)$$

$$\mathbf{b} \leftarrow \mathbf{b} + \alpha \frac{1}{|\mathcal{B}|} \sum_{t \in \mathcal{B}} \left(\hat{\mathbf{h}}(\mathbf{x}^{(t)}) - \hat{\mathbf{h}}(\tilde{\mathbf{x}}^{(t)}) \right) \quad (2.7)$$

$$\mathbf{d} \leftarrow \mathbf{d} + \alpha \frac{1}{|\mathcal{B}|} \sum_{t \in \mathcal{B}} \left(\mathbf{x}^{(t)} - \tilde{\mathbf{x}}^{(t)} \right) \ ; \quad (2.8)$$

where $\alpha > 0$ is the learning rate hyperparameter and $\hat{\mathbf{h}}(\mathbf{x}) = \boldsymbol{\sigma}(\mathbf{b} + \mathbf{W}\mathbf{x})$, with $\boldsymbol{\sigma}(\cdot)$ being the element-wise operation of $\sigma(y) = \frac{1}{1+e^{-y}}$.

As would be expected, the RBM has a set of hyperparameters that influence its learning performance. The learning rate α and batch size $|\mathcal{B}|$, for example. The training guide by Hinton [40] presents a qualitative discussion of good hyperparameter choices for the RBM acquired through experience, indicating the inherent difficulty of this task in general. Notwithstanding, some works have proposed automating the search for effective hyperparameters [42] and also the internalizing the hyperparameters [10]. In particular, Côté & Larochelle modify the RBM making the

number of hidden units a parameter of the model which is then determined along with other parameters during training [10].

In this context, the network connectivity pattern of the RBM can also be interpreted as a hyperparameter that must be determined. However, this is only meaningful if the connectivity pattern has a fundamental influence on the learning performance of the model, which is not a given, a priori. The analysis of the influence of the connectivity upon the RBM is addressed in Chapter 4, which has proved favorable. Therefore, Chapter 5 proposes and analyzes the Network Connectivity Gradient (NCG) method, which in turn optimizes the connectivity together with the classical RBM parameters.

2.4 Datasets

In order to evaluate the proposed methods, two datasets were selected: BAS and MNIST. BAS is a toy dataset, used to perform tests upon constrained conditions, in which we can limit the RBM size to perform exact NLL calculations, for example. MNIST, especially, is commonly used in the RBM literature [10, 11, 31, 35, 43], although BAS has also been used before [11], which is the reason they have been selected for this work's analyses.

2.4.1 Bars And Stripes

Bars And Stripes (BAS) [44] is a model of parameterized datasets where elements are a square of l^2 unit-squares with either a set of rows or columns painted black or white (l is the sole parameter). A random sample can be generated by choosing an orientation (horizontal or vertical) with equal probability and then selecting for each row or column a color black or white, also with equal probability. Considering all possible arrangements of black and white rows or columns, the dataset contains $C = 2^{l+1}$ configurations. Five examples for $l = 4$ (BAS 4) can be seen in Figure 2.5.

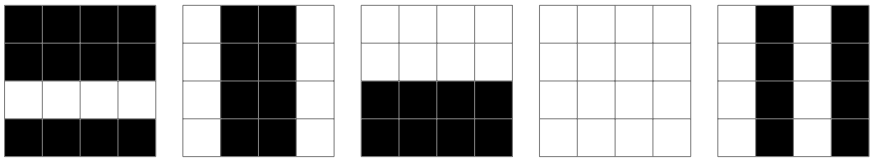


Figure 2.5: Examples of BAS 4 configurations

Since the BAS model has a uniform distribution over its samples, the average NLL for the model can be computed exactly, and is given by

$$\frac{1}{C}\sum_{t=1}^{C} -\ln P(\mathbf{x}^{(t)}) = \ln C\,, \tag{2.9}$$

where C is the number of configurations. Note that this value is the optimal average NLL for a trained RBM.

The BAS provides a controlled and relatively simple environment to assess the influence of network connectivity on the learning performance of the RBM, where results can be interpreted more easily.

2.4.2 MNIST Database of Handwritten Digits

Although good for a initial evaluation, the BAS model is relatively contrived due to its regularity and far from real world data. To overcome this limitation and broaden the experiments, the MNIST dataset[2] was used. This dataset contains images of handwritten digits and is a frequently used benchmark in computer vision, including the RBM literature. Thus, it is an important dataset to consider for this work's studies.

The MNIST dataset consists of gray-scale square images of 28×28 pixels. It has two separate sets of data: a train set, with 60 k samples; and a test set, which has 10 k samples. Only the train set was used to train the RBMs, although some experimental results are also reported on the test set. The images were converted into black and white in order to be directly used as input to the RBM. The conversion was probabilistic such that each pixel was assigned a black color with probability proportional to its darkness (gray-scale) in the original image, a methodology widely adopted [10, 12]. Some examples of the resulting data samples are shown in Figure 2.6.

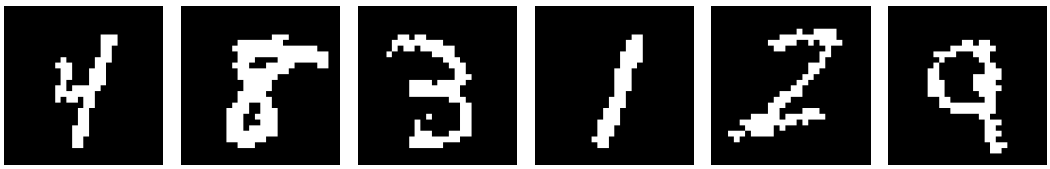


Figure 2.6: Examples of MNIST images after conversion to binary.

This dataset enables for performing two kinds of tasks: sample generation, for which new digit images can be sampled from the trained RBM distribution; and image classification, for which the RBM should predict the digit of a given sample.

[2]Dataset available at `http://yann.lecun.com/exdb/mnist/`.

Chapter 3

Network Connectivity and Performance Estimation

3.1 Network Connectivity

The classic RBM considers a fully connected network between its input and hidden layers. However, this connectivity may not be adequate for training an RBM on a specific task, in the sense that other connectivity patterns could yield better learning curves. In order to explore this possibility, the RBM must be first extended to allow for any connectivity pattern.

Let $\mathbf{A} \in \mathbb{B}^{H,X}$ denote a binary matrix that represents a given connectivity pattern for the RBM, in the sense that $a_{ij} = \mathbf{A}[i,j] = 1$ if hidden unit h_i is connected to input unit x_j, or $a_{ij} = \mathbf{A}[i,j] = 0$ otherwise. Figure 3.1 shows examples of the adjacency matrix (or weight's mask) $\mathbf{A}$ for two connectivity patterns. Note that the size of the space of connectivity networks (all possible matrices $\mathbf{A}$) is given by 2^{HX}, and is generally intractable even to enumerate in the case of small models.

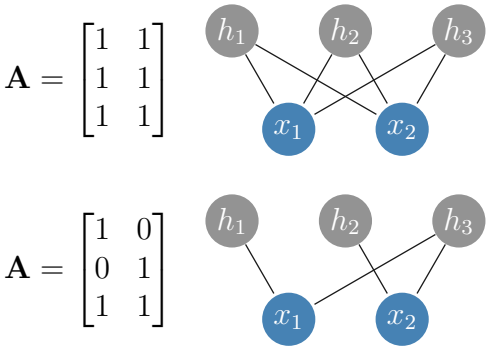


Figure 3.1: Two examples of adjacency matrices $\mathbf{A}$ (left) and the corresponding RBM network (right): the classic fully connected network (top), and an example with two suppressed connections, $x_1 - h_2$ and $x_2 - h_1$ (bottom).

In order to incorporate $\mathbf{A}$ into the model, the weights in matrix $\mathbf{W}$ must be

forced to zero on entries where a connection is not present. Thus, let $\mathbf{C} = \mathbf{W} \odot \mathbf{A}$ denote the acting weights of the model where $\odot$ is the element-wise matrix product such that $c_{ij} = \mathbf{C}[i, j] = w_{ij}a_{ij}$. The resulting energy function, thus, is the same as Equation 2.1, with only weight modifications. Indeed, the acting weights function as merely weight modifiers, not changing the theory behind the energy function. The resulting energy corresponds to

$$E(\mathbf{x}, \mathbf{h}) = -\mathbf{h}^{\mathrm{T}}\mathbf{C}\mathbf{x} - \mathbf{x}^{\mathrm{T}}\mathbf{d} - \mathbf{h}^{\mathrm{T}}\mathbf{b}\,, \tag{3.1}$$

which is the same as the original (Equation 2.1) with a number of weights being forced to zero by $\mathbf{A}$, resulting in the acting weights $\mathbf{C}$.

The classic model parameters ($\mathbf{W}$, $\mathbf{b}$, $\mathbf{d}$) are learned as before, however using matrix $\mathbf{C}$ instead of $\mathbf{W}$ to compute the gradients.

3.2 Generative Performance Estimation

For a generative task, RBMs are trained to minimize the Negative Log-Likelihood (NLL) of the training data. However, this metric is intractable for non trivial models because of the complexity of computing the normalization constant of the model, which makes evaluating the performance of training difficult.

Therefore, there are many techniques to approximate the NLL, and in the case of RBMs the most used is Annealed Importance Sampling (AIS), as proposed by Salakhutdinov & Murray [12]. However, this is a very complex method, which prompts for simpler (and perhaps faster) estimation methods. The two methods proposed in this work, one estimating through Monte Carlo and the other approximating through truncation, have not been able to achieve good results on the MNIST dataset, as the following experiments show in comparison to AIS.

The time results here presented correspond to wall time values obtained on a personal computer running macOS Big Sur with a processor Intel i5, 1.6 GHz.

3.2.1 Monte Carlo

The first alternative algorithm proposed for estimating the partition function is a simple Monte Carlo (MC) implementation [45]. The idea, in this case, is to find an expectation through which one can calculate Z.

The partition function is the constant used to normalize the probability distribution, so that the sum of all probabilities is equivalent to 1. As defined in Section 2.3, the constant is $Z = \sum_{\mathbf{x},\mathbf{h}} e^{-E(\mathbf{x},\mathbf{h})}$. However, sampling over both $\mathbf{x}$ and $\mathbf{h}$ is more complex than sampling only over $\mathbf{x}$. For that reason, we simplify the formula using

the RBM's visible units distribution $P_\theta(\mathbf{x}) = Z^{-1}e^{-F(\mathbf{x})}$ with the resulting equation being:

$$Z = \sum_x e^{-F(\mathbf{x})} . \tag{3.2}$$

Therefore, a simple expectation can be used to find Z, that is the expected value of $e^{F(x)}$:

$$\mathbb{E}_{P_\theta(\mathbf{x})}\left[e^{F(\mathbf{x})}\right] = \sum_x e^{F(\mathbf{x})} P_\theta(\mathbf{x}) = \sum_x e^{F(\mathbf{x})} \frac{e^{-F(\mathbf{x})}}{Z} = \frac{1}{Z}\sum_{\mathbf{x}} 1 = \frac{1}{Z} 2^X , \tag{3.3}$$

where X is the number of input units. Accordingly, if the expected value is known, it is trivial to calculate the normalization constant:

$$Z = \frac{2^X}{\mathbb{E}_{P_\theta(\mathbf{x})}\left[e^{F(\mathbf{x})}\right]} . \tag{3.4}$$

Now, one can estimate the expectation of $e^{F(\mathbf{x})}$ through sampling, as per Monte Carlo (MC) algorithms. If we sample enough $\mathbf{x}^i$ values according to the $P_\theta(\mathbf{x})$ distribution (e.g. the RBM's visible units distribution), then the sample mean of $e^{F(\mathbf{x}^i)}$ should converge to the expectation $\mathbb{E}_{P_\theta(\mathbf{x})}\left[e^{F(\mathbf{x})}\right]$, by the law of large numbers:

$$\lim_{n\to\infty} \frac{1}{n}\sum_{i=1}^{n} e^{F(\mathbf{x}^i)} = \mathbb{E}_{P_\theta(\mathbf{x})}\left[e^{F(\mathbf{x})}\right] . \tag{3.5}$$

Therefore, this simple method only needs to determine n, the number of samples to perform so that a good estimation of Z is obtained. However, it is tricky to obtain $\mathbf{x}^i$ independent samples from the RBM distribution $P_\theta(\mathbf{x})$, since many Gibbs Sampling steps may be required, making the algorithm too costly. Furthermore, it is not easy to ascertain how many steps are needed to guarantee independence between the samples (and therefore an unbiased estimator).

3.2.2 Truncation

Since the MC estimator might severely increase the computational load with larger RBMs, a second estimator is proposed, which truncates the partition function.

The idea of truncation is that, since one has a summation with too many parcels, one clips (or truncates) the sum to a smaller number of parcels, so that it becomes feasible to compute it in a reasonable time. The truncated estimate can be defined as

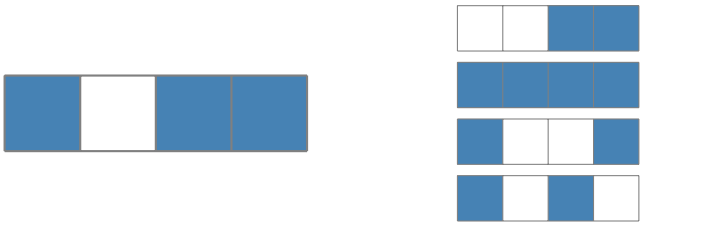
$$\hat{Z} = \sum_{x\in\mathcal{X}'} e^{-F(\mathbf{x})} , \tag{3.6}$$

where $\mathcal{X}' \subseteq \mathbb{B}^X$. Note that by definition, if no sample $\mathbf{x}$ is added twice, the estimator

is a lower bound on the constant: $\hat{Z} \leq Z$. In order to be a good estimator, it is necessary that all the parcels with larger values (greater participation in the final sum) be used for the truncated summation.

In this work, we propose to select, based on data, the configurations of visible units which should be the most important to the calculation of Z. After all, the RBM is trained to match the data distribution and, consequently, as training progresses, the configurations found on the dataset should become more probable and such an estimator should only get better.

Therefore, we define the subset $\mathcal{X}'$ as the set of all the samples in the dataset, plus all samples that differ from the dataset ones by a single unit (which, being similar enough to the correct samples, could still have high probabilities). To illustrate how such a dominion is formed, consider Figure 3.2: if the dataset has the sample on the left (a), the truncation method will sum not only the parcel relative to it, but also the four other displayed on the right (b), which are all configurations that differ from (a) by a single unit.



(a) Original sample (b) Extra samples generated

Figure 3.2: An example of how the dominion $\mathcal{X}'$ is formed for the estimation of the partition function through Truncation. For every sample (a) in the dataset, X extra samples are generated (b) by changing a single unit from the original sample.

Note that, by the $\mathcal{X}'$ definition, this truncation proposal corresponds to a deterministic method, always producing the same estimate for a given RBM and dataset. Furthermore, truncating the normalization constant should allow for a faster code than the MC estimation, for it does not need to sample from the RBM. It may, however, become slow with the increase in the dataset size.

3.2.3 Comparison

Before comparing both the Monte Carlo and the Truncation methods, they need to be validated to certify that they work as expected. That was done using the BAS 5 dataset, with a minor modification to suit Truncation better: the repeated samples were removed. In BAS, by its law of creation, there are two configurations that are duplicated: the ones with all squares of the same color. That happens because there are two ways to get such configurations: by having all rows in the same color, or by

having all columns on the same color. With their removal, the assumption that the truncated estimate $\hat{Z}$ must be smaller or equal to the actual normalization constant Z holds true.

Table 3.1 shows the results for both MC and Truncation for two different RBMs: one that was trained for only 100 epochs (dubbed the undertrained model) and another trained for 1.5 k epochs (dubbed the trained model). Training was achieved using CD with 10 steps of Gibbs Sampling (CD-10), $\alpha = 0.1$ and batch size of 5 random samples. The RBM was initialized with null biases and weights uniformly sampled between -1 and 1. The exact NLL for these networks are 13.022 and 7.186 respectively, with the calculus taking approximately 35 ms to determine it. MC used 10 k samples, giving $\texttt{int}\,(\ln X) = 3$ Gibbs Sampling steps between samples. 5 repetitions were performed to acquire the mean and standard deviation values. Note that truncation has no standard deviation because it is deterministic by nature.

Table 3.1: Performance of NLL estimation using MC (a) and Truncation (b). For each method, the estimate, its error and the time needed to perform it are presented for two RBMs trained with the BAS 5 dataset: one undertrained model, for which 100 epochs are performed; and one fully trained mode, with 1.5 k epochs of training. A negative error indicates that the resulting estimate is bigger than the real value of the NLL.

(a) MC Estimation

	Undertrained Model	**Trained Model**
NLL Estimate	13.016 ± 0.028	8.160 ± 3.123
Mean Relative Error	0.5%	-13.6%
Time (ms)	77 ± 3	75 ± 2

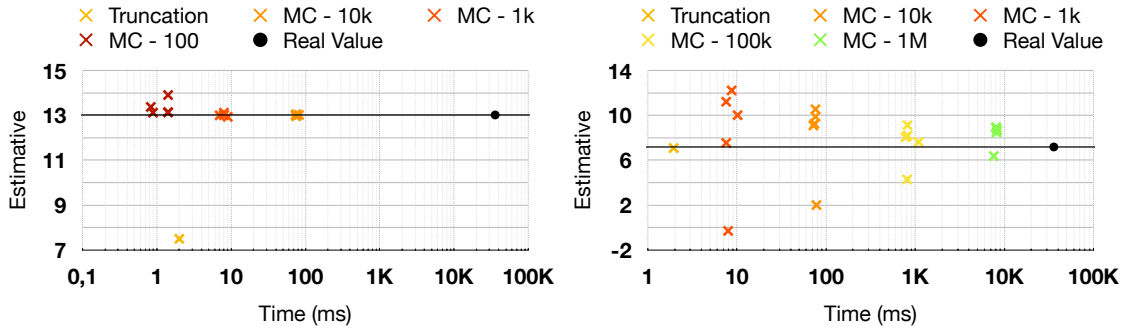
(b) Truncation Estimation

	Undertrained Model	**Trained Model**
NLL Estimate	7.494	7.089
Relative Error	42.5%	1.3%
Time (ms)	2	2

One can see that there is an inversion between the estimates: MC begins as the better estimate, when the RBM distribution is still very random, and worsens with training; Truncation, on the other hand, begins as a very poor estimate, but grows closer to the correct value as the RBM distribution grows closes to the data distribution.

This trend is more notable with a visual inspection, provided by Figure 3.3. Alternate versions of the MC method were added, using different numbers of samples n to calculate the sample mean ($n \in \{100, 1\,\text{k}, 10\,\text{k}, 100\,\text{k}, 1\,\text{M}\}$). In the figure, each single estimation is marked by an "$\times$", which is colored according to the estimator used. Since the truncation estimate is deterministic, it was performed only once, while MC estimated were repeated 5 times for each configuration. The exact value

of the NLL is marked by the black line, with the black circle indicating the time it took to calculate it.



(a) Undertrained Model

(b) Trained Model

Figure 3.3: Scatter plot of NLL estimation performance (wall time × value) using MC and Truncation for two RBMs on BAS 5: one trained for 100 epochs (a) and one for 1.5 k epochs (b). MC was applied with different numbers of samples (varying between $n = 100$ and $n = 1\,\mathrm{M}$), each experiment repeated 5 times. The black line corresponds to the exact NLL, and the time taken to calculate it is marked with a circle.

With the increase in the number of samples n the MC estimator gets better, diminishing the variance observed, as would be expected. Also, there seems to be no bias in the estimation, which suggests that the method functions as it should.

The results with $n = 100$ have comparable wall time as Truncation, and indeed are a better estimate for a poorly trained network. However, with more training even $n = 10^6$ still presents a high variance, despite it already being a costly algorithm, when you compare with the time it took to calculate the exact value. Truncation, on the other hand, presents itself as a good estimator, for it gets a very good estimate in a very short time for the trained model (1.5 k epochs).

Having validated the proposed estimators on BAS, the next step is to compare them under a more strenuous setting. Therefore we use an RBM trained on MNIST. Training was achieved using CD-10 for 6 k epochs, having $\alpha = 0.01$, with batch size of 50 randomly drawn samples. The weights were initialized with small random values, uniformly distributed in the $[-1, 1]$ interval, and the biases were initialized at zero.

This network uses only 16 hidden units ($H = 16$), which enables us to calculate the exact NLL, despite the great number of visible units ($X = 784$). Besides the exact calculation, MC was calculated using 10 samples to compose the sample average ($n = 10$), applying approximately 3 million Gibbs Sampling steps between samples. [1] Lastly, the AIS estimator was added for comparison, for it is the tech-

[1] We used 2 953 133 steps between samples, which is equivalent to $10\left[\ln\left(2^X\right)\right]^2$.

nique used in the RBM literature. AIS was calculated with 100 runs and 5 500 intermediate distributions.

The NLL results are presented at Table 3.2, including the exact value. It is clear that the best estimation is provided by AIS, which is made even clearer by the analysis of Figure 3.4. In the figure, each single estimation is marked by an "×", which is colored according to the estimator used. Since Truncation is deterministic, it was performed once only, while MC and AIS were repeated 5 times. The exact value of the NLL, which is the goal of the estimations, is displayed as a black lines.

Table 3.2: Comparison of NLL estimations for a classical RBM used with the MNIST dataset. For each method, the estimated value and the time taken to acquire it are reported. Average and standard deviation reported for MC and AIS, calculated for 5 estimates.

	Real Value	**Truncation**	**MC**	**AIS**
NLL Calculated	153.85	98.94	467.63 ± 7.78	153.62 ± 0.13
Time (s)		54	1007 ± 77	66 ± 6

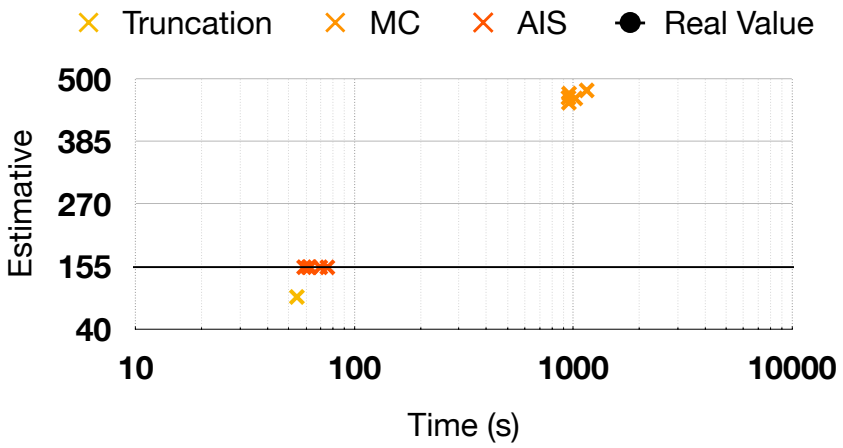


Figure 3.4: Scatter plot of NLL estimation performance (wall time × value) for a RBM used with the MNIST dataset. MC and AIS estimations were repeated 5 times, and the black line corresponds to the exact NLL.

It is clear that the MC did not perform well and, indeed, has a large bias in the output, which suggests that despite the already huge computational load, there are still not enough Gibbs Sampling steps to guarantee that the acquired samples are independent. This makes unfeasible the use of this estimator, since it is already so much more costly than the others. As for the truncation method, it does not have such a good estimate yet, even after 6 000 epochs, and it takes almost as much time to calculate as the AIS method does, which has a much better accuracy, and excellent precision to boot.

These findings show that there is no gain in applying those methods, as they are to the RBM performance estimation problem. The standing AIS method is much

more suited to the task, and it is the one that will be used throughout the remainder of this work.

Chapter 4

Connectivity Search Space Analysis

The connectivity pattern between layers influences the learning performance of the network, in particular for shallow networks such as the classic RBM. While recent works have addressed the problem of determining the number of neurons in the hidden layer [10], the connectivity pattern between the input and hidden layers have not been addressed, to the best of our knowledge.

In order to understand the role of network connectivity on the learning performance of RBMs, this chapter presents a comprehensive study of the connectivity space using the synthetic BAS model and the MNIST dataset. Different connectivity structures are proposed, and the effect of training RBMs with those is evaluated.

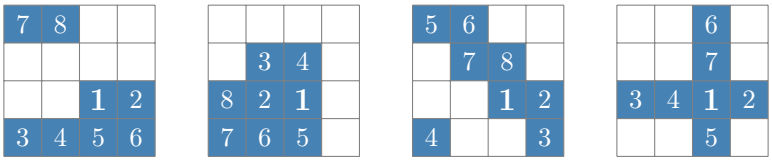
4.1 Connectivity Patterns

In order to characterize the effect of network connectivity, four connectivity patterns are considered in this work: line, spiral, stairs and cross.

Figure 4.1 shows an illustration of the patterns. It portrays the connections with regards to the hidden unit h_{11} for BAS 4 with 8 neighbors ($v = 8$) for the two parameterized patterns. For other units h_i, the connections should merely shift the start point (marked 1 in the figures) to the corresponding unit. The numbers in the illustrations give an idea of the pattern obtained for other values of v, in the case of the line and spiral patterns.

The line pattern is the most simple: each hidden unit h_i is connected to the v adjacent visible units $\{x_j | i \leq j \leq [(i + v) \mod X] + 1\}$, where $r \mod q$ is the remainder of the division r/q. Note that v is a parameter of the pattern that represents the number of connections of each hidden unit (and consequently visible unit). This is obviously a very important parameter, and analyses of the impact of changing this parameter are presented throughout the chapter.

The spiral pattern was inspired by the convolutional layers from neural networks applied to computer vision: considering the organization of the visible units arranged



(a) Line (b) Spiral (c) Stairs (d) Cross

Figure 4.1: Connections of unit h_{11} in BAS 4. The line (a) and spiral (b) patterns use $v = 8$, while the stairs (c) and cross (d) patterns have a fixed number of 8 connections and 7 per unit, respectively. The grid represents the 16 visible units and the blue squares are the visible units connected to h_{11} (the 11th visible unit x_{11} corresponds to the square in the 3rd row and 3rd column) for each pattern. The numbers indicate the order in which the units are added, to illustrate how the pattern is generated as a function of v, the number of connections.

into rows and columns, each hidden unit h_i is connected to its respective visible unit x_i and to other $v - 1$ units in an spiral pattern beginning in x_i. The spiral is constructed clockwise starting to the left of the initial unit, forming squares. Each new square begins with a downward step from x_i. The spiral assumes that the space of rows and columns wraps around itself.

The stairs pattern was designed specifically for the BAS model (see Section 2.4.1). Considering that this dataset is formed of painted rows or columns, each hidden unit needs only to know the state of two neurons in each row and in each column in order to be able to ascertain the whole state of the configuration. Organizing the connections in the form of stairs generates a pattern with the minimum amount of connections that still guarantees that all rows and all columns in the hidden layer are connected to two different units. For BAS 4, this corresponds to 8 connections for each unit. Note that the stairs pattern has no parameter.

Finally, the last pattern is the cross which was also designed for the BAS model. It connects each hidden unit to the row and column of its respective visible unit, so that the resulting pattern forms a cross centered on that visible unit. The intuition is that, if the input values for both the row and column are known, the color of the center of the cross should be trivially determined in a valid configuration.

4.2 Analysis on BAS

For the first set of experiments, the BAS model (Section 2.4.1) is utilized. It generates simple artificial datasets, and its clear formation rule together with the small sizes considered makes it a prime candidate for initial testing.

Most of the following experiments use BAS 4 which means the RBM has 16 visible units ($X = 16$) and there are $C = 32$ configurations in total. The number of hidden units is the same as the visible units ($H = X = 16$). Training was achieved

using CD with 10 steps of Gibbs Sampling (CD-10), with the exception of Section 4.2.3. The learning rate was $\alpha = 0.01$ and the batch size was 5 randomly chosen samples.

The model's parameters are updated after every batch and an epoch corresponds to an iteration of updates through the entire dataset. No momentum or weight decay was used. The RBM was initialized with null biases and random weights uniformly distributed in $[-1, 1]$. Each experiment was independently repeated 25 times to account for statistical fluctuation at initialization. The lines in the plots correspond to the sample average and the shades, when available, correspond to the distribution quartiles (50% of the results fall within the shaded area). Note that not all plots contain the quartiles' information to avoid visual pollution. The missing quartile figures can be found in Appendix A.

In this scenario (BAS 4 and BAS 5), the normalization constant, as well as the average NLL, can be computed in an exact manner, and therefore the learning curve of the RBM can be precisely evaluated (at every epoch for BAS 4 and at every 5 epochs for BAS 5). For BAS 8, it is intractable to compute the normalization constant exactly, thus it was approximated using Annealed Importance Sampling [12] every 5 epochs. The parameters for the AIS was 100 runs and 14.5k intermediate distributions.

Although it is known that fine-tuning its hyperparameters improves the RBM training, the purpose here is to ascertain the impact of different connectivity networks and therefore it focuses on comparative results instead of trying to obtain the best possible RBM.

4.2.1 Connectivity Structure

For the first set of experiments, a comparison of the connectivity patterns presented in section 4.1 was made. For fairness of comparison, the line and spiral patterns use $v = 8$, so that the networks mostly have the same number of connections.

Figure 4.2 shows the evolution of the average NLL as a function of the training epochs for the four connectivity patterns, where all units have 8 neighbors ($v = 8$), and the cross pattern 7 neighbors. The curves indicate a clear difference in the average NLL among the patterns, and the relatively small uncertainties (shaded area) show that the patterns have consistent results regardless of the initialization.

Although the three first patterns have the same number of connections, it is clear that the structure itself is of fundamental importance, for they have vastly different learning curves. It is also of notice that despite having one less connection than the others, the cross pattern presents a superior performance for the first 4000 epochs, with the fastest initial decrease in the average NLL. At around 6 k epochs,

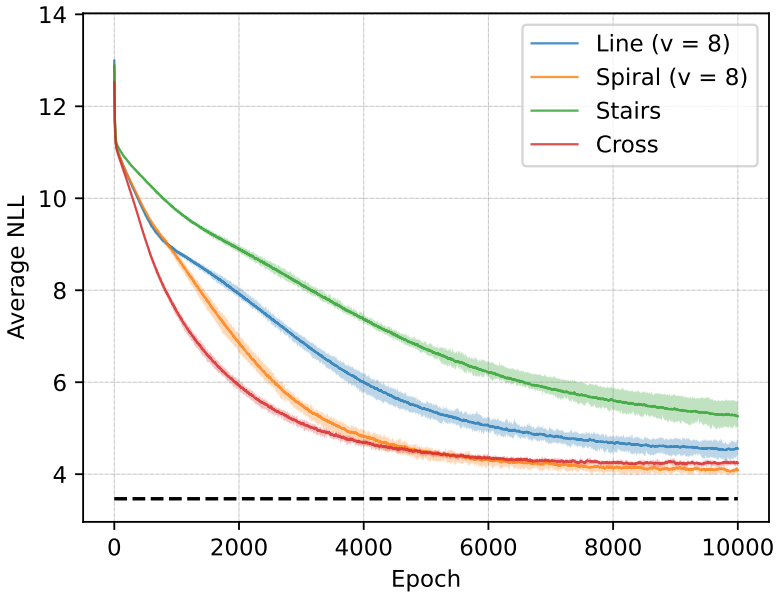


Figure 4.2: Evolution of the average NLL over training epochs for four different connectivity networks: stairs, spiral and line, that have 8 connections per each unit, and cross, that has 7 connections per unit. The black dashed line corresponds to the data (optimal) average NLL.

the spiral learning curve overtakes the cross, and the mean results suggest that it has the better learning performance by the end of 10 k epochs (smallest average NLL mean).

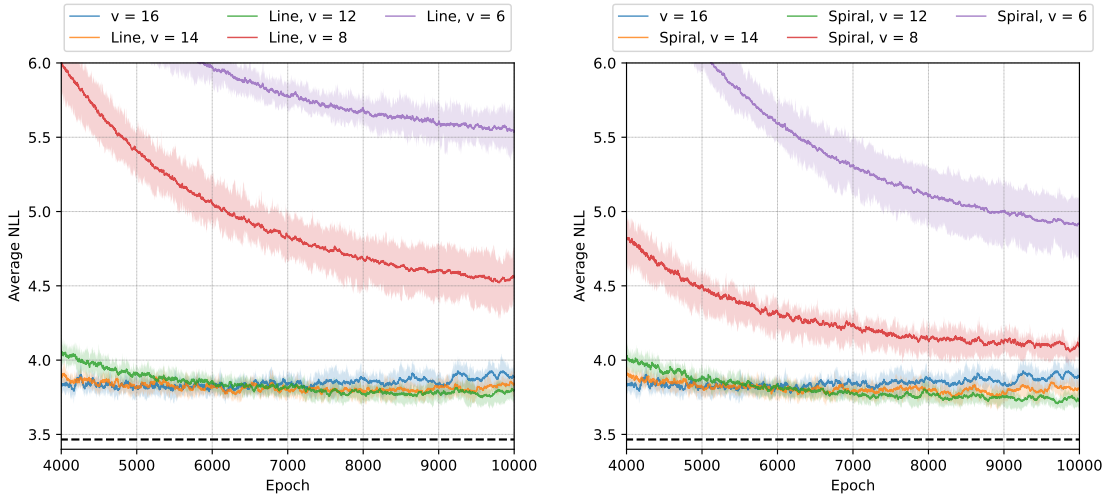
Interestingly, the stairs pattern showed the worst learning curves for all epochs, despite being intuitively designed for the BAS model. Thus, intuition did not work for this pattern. Finally, line pattern showed comparable performance to the spiral during the first thousand epochs but then distanced itself from the decay of the spiral.

The results indicate that hidden units of the RBM prefer to have more local than scattered information from the input. Note that the spiral pattern has the most local information: if you take into account the square numbered 2 in Figure 4.1(b), the hidden unit is connected to seven of the eight possible neighboring squares, and to the four cardinal neighbors (north, south, east and west squares). In comparison, the square numbered 1 in the cross pattern has four of the eight possible neighboring square, all cardinals. The square numbered 5 in line pattern is connected to 5 neighboring squares, of which three are cardinals, and in the stairs pattern units are connected to two cardinal neighbors and two diagonal neighbors. Last, the spiral pattern is identical to a convolution connection when v is a square number, and corroborates the vast success of convolutional patterns in image recognition tasks.

4.2.2 Number of Connections

Intuitively, the learning performance of the RBM depends not only on the connectivity pattern but also on the number of connections in the pattern. But it begs the questions of whether this dependence is monotonic. In other words, whether having more connections for a given pattern yields a strictly better learning curve.

Figure 4.3 shows the learning curves for networks with different number of connections for two patterns (line and spiral). For both patterns, note that as the number of connections increases from 6 to 8 to 12, the learning curve improves monotonically. However, as the number of connections continues to increase from 12 to 14 to 16, the learning curves do not strictly improve across all epochs. In particular, the learning curves cross one another and for a large enough epoch the best apparent learning performance is attained with $v = 12$ connections. Interestingly, this observation holds for both line and spiral patterns. Last, while the differences in the learning curves for $v \in \{12, 14, 16\}$ are rather small for BAS 4, Section 4.2.4 shows that such differences are very significant as the model becomes larger (e.g., BAS 8).



(a) Line (b) Spiral

Figure 4.3: Evolution of the average NLL over training epochs for different numbers of connections for the line (left plot) and spiral patterns (right pattern). The network with $v = 16$ is the same for both patterns and corresponds to the fully connected traditional RBM. The black dashed line corresponds to the data (optimal) average NLL.

As the number of connections increases, the differences between the patterns diminish. For example, for $v = 12$ both line and spiral patterns are relatively similar (since there are at most 16 connections). Will this observation reflect on the learning curves?

Figure 4.4 shows the learning curves for both the line and spiral patterns for

different values of v. The cross pattern is also added for comparison, but its performance seems not to be comparable to the best line and spiral patterns. Note that for the same value for v the spiral pattern has a superior learning performance which is more pronounced when v is small (e.g., $v = 4$ and $v = 6$). However, this superiority diminishes as v increases, such that for $v = 12$ and larger both patterns show comparable performance. Indeed, at $v = 16$, which corresponds to a fully connected network (the traditional RBM architecture), both patterns become identical.

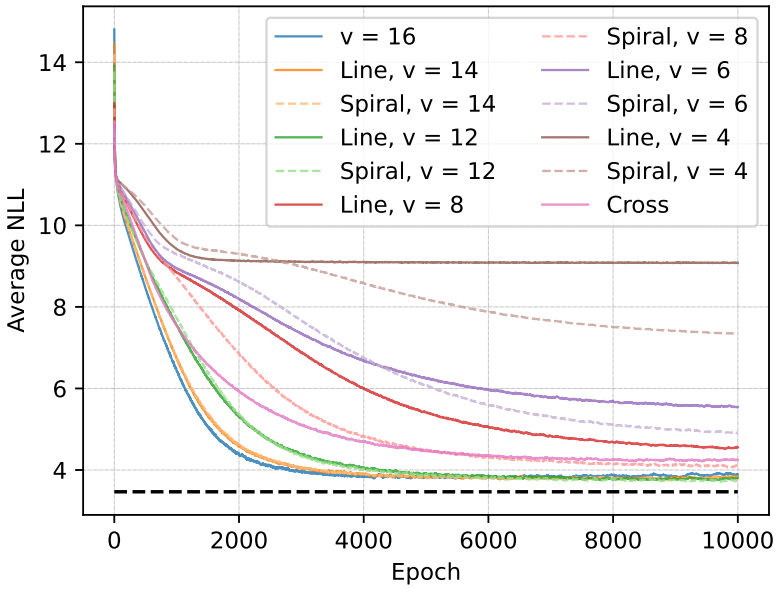


Figure 4.4: Evolution of the average NLL over training epochs for different numbers of connections for the line (full curves) and spiral patterns (dashed curves). The cross pattern is also added. RBMs with the same number of connections are shown with different shades of the same color. The network with $v = 16$ is the same for both patterns and corresponds to the fully connected traditional RBM. The black dashed line corresponds to the data (optimal) average NLL.

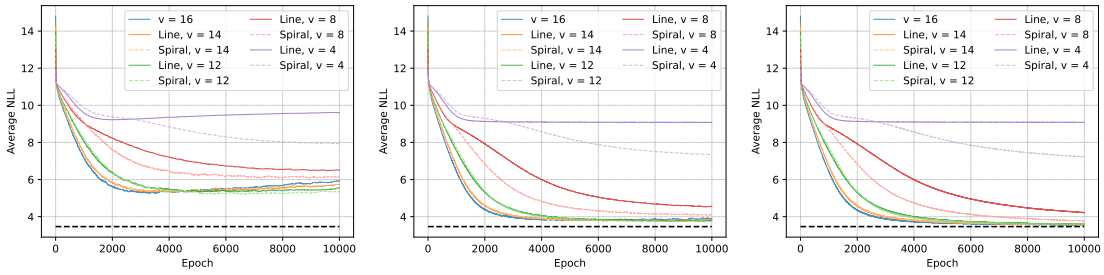
Interestingly, when v is small the patterns are more different and thus their performance depends more on the pattern, in which case the spiral is superior. Moreover, if v is too small (e.g., $v = 4$) the RBM cannot be trained adequately and the average NLL does not decay (line pattern) or decays very slowly (spiral pattern). Again, RBM seems to prefer connections that are more localized (spiral) even if the image is a regular pattern, as the BAS model.

4.2.3 Contrastive Divergence Approximation

The previous experiments have been training the models using 10 steps of Gibbs Sampling (CD-10). However, it is well known that the number of steps has great influence over training, which leads to the question of how this influence impacts different connectivity networks and whether it changes the conclusions drawn from

the previous experiments.

Figure 4.5 shows the comparison of connection structures with different v for three different CD approximations: with 1, 10 and 100 sampling steps. It is clear from the results that the increase in the number of steps is beneficial to training, from the NLL point of view. In fact, CD-100 manages results very close to the NLL lower bound for both the traditional training and patterns with $v = 12$.



(a) CD-1 (b) CD-10 (c) CD-100

Figure 4.5: Evolution of the average NLL over training epochs for different numbers of connections and network structures. Each subplot corresponds to a different CD approximation: the left plot was trained with CD-1, in which only 1 sampling step is applied, the center plot uses 10 sampling steps and the right plot 100 steps. RBMs with the same number of connections are shown with different shades of the same color. The network with $v = 16$ is the same for both patterns and corresponds to the fully connected traditional RBM. The black dashed line corresponds to the data (optimal) average NLL.

Although the plots are different from each other, the overall pattern remains and the conclusions drawn from the results are unchanged. It is of note, however, that with worse gradient approximations (less sampling steps) the connectivity bears higher impact upon the results and there is greater advantage in using them in such cases. With a higher number of steps, the final results between the fully connected network and spiral/line patterns with $v = 12$ get more similar, to the point where in CD-100 we see no apparent advantage in using a connectivity different from the complete graph, which is already very close to the desired data NLL.

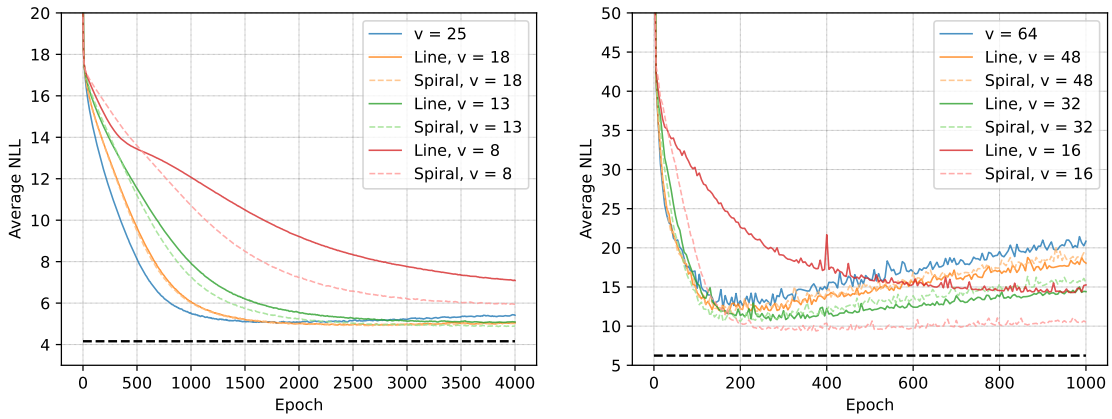
Even then, these findings still point out towards there being advantages in changing the model's connectivity network, for it only became moot when the traditional network is already capable itself of achieving near optimal results, which is unlikely to occur in most datasets, even with CD-100 approximation.

4.2.4 Larger Models

All prior experiments used BAS 4 which albeit efficient to train is a rather small model ($C = 32$). However, the previous findings concerning patterns and number of connections are not an artifact of BAS 4. On the contrary, the differences are

magnified as the model increases complexity.

Figure 4.6 shows the learning curves for the line and stairs patterns, using different numbers of connections, for BAS 5 (top plot) and BAS 8 (bottom plot). Note that the number of hidden units is the same as the number of visible units, and therefore BAS 5 has $H = 25$ and BAS 8 has $H = 64$.



(a) BAS 5 (b) BAS 8

Figure 4.6: Average NLL through training epochs for different numbers of connections and BAS models: BAS 5 with $X = 25$ (a) and BAS 8 with $X = 64$ (b). For each plot, RBMs with the same number of connections are shown with different shades of the same color. The fully connected RBM is the same for both patterns (blue curves). The black dashed line corresponds to the data (optimal) average NLL.

It is of note, that in the right plot (BAS 8) there is a clear increase in the average NLL after 200 training epochs for most of the learning curves. That is a common behavior in RBMs: since training is achieved by approximating the gradient, if the model is trained for too many epochs the performance starts to deteriorate instead of improve. These phenomena can be mitigated by increasing the number of Gibbs Sampling steps performed, which improves the quality of the gradient approximation.

For BAS 5, the results are qualitatively similar to BAS 4, where the two patterns have different performance for small enough v (spiral is superior to the line) and identical performance for large enough v, and that learning curves are not monotonic for large enough epochs.

However, the learning performance for BAS 8 is quantitative and qualitatively different, in particular with respect to the non-monotonicity of the learning curves, which cross over quite early in the training. The plot shows the spiral and line patterns with 16, 32 and 48 connections per unit, and all of them eventually surpassed the performance of the fully connected network. Moreover, even early in training the fully connected network was not superior, showing performance comparable to

$v = 48$ for both patterns. Interestingly, the best learning curve is the spiral pattern with $v = 16$ which has only $1/4$ of the connections of the fully connected network, but its performance is not matched by the line pattern with $v = 16$. Note that for $v = 16$ the spiral corresponds to a perfect 4×4 convolution.

As illustrated by BAS 8, it is clear that both the connectivity pattern and number of connections play a fundamental role in the learning performance. Intuitively, a model with higher complexity has more parameters to train, and thus offer more opportunities for sparse connectivity patterns to have a learning performance that is superior to the fully connected network.

4.3 Analysis on MNIST

Although the previous experiments indicate the importance of the connectivity pattern and number of connections even for small scale networks, the BAS model is relatively contrived due to its regularity and far from real world data. To overcome this limitation and broaden the findings, the MNIST dataset (see Section 2.4.2) is used to characterize the learning performance of the RBM under different connectivity patterns.

The RBMs were trained with 784 hidden units which is the same number of visible units for the dataset. The learning rate used was 0.01, with batch size of 50 randomly chosen samples. Runs used CD-10 and each configuration was repeated 10 times. The sample average is reported, and the distribution quartiles are added in Appendix A. RBM parameters were initialized as in the previous experiments: null biases and small random weights uniformly distributed between -1 and 1. As with the previous experiments, further fine-tuning of these parameters was not attempted. Also, as with BAS 8, the normalization constant cannot be computed exactly for this model and was approximated using AIS [12].

Figure 4.7 shows the evolution of the learning curve for the line and spiral patterns varying the number of connections. Note that $v = 784$ corresponds to the fully connected network. The two patterns were evaluated with $\frac{3}{4}(v = 588)$, $\frac{1}{2}(v = 392)$, $\frac{1}{4}(v = 196)$ and $\frac{1}{8}(v = 98)$ of the maximum number of connections, as well as $v = 16$ connections per unit, which forces a significant difference between line and spiral patterns (perfect 4×4 convolution for spiral and single line for line pattern).

Interestingly, the the fully connected network showed the worst performance. As the number of connections decrease, the learning performance of both alternative patterns improve in general. Clearly, a more sparse network can be trained more effectively, improving the average NLL by a factor of four at epoch 60 (if compared to the fully connected network). However, the learning curves of the model with less connections ($v = 196$, $v = 98$ and $v = 16$) did not show a decreasing trend over the

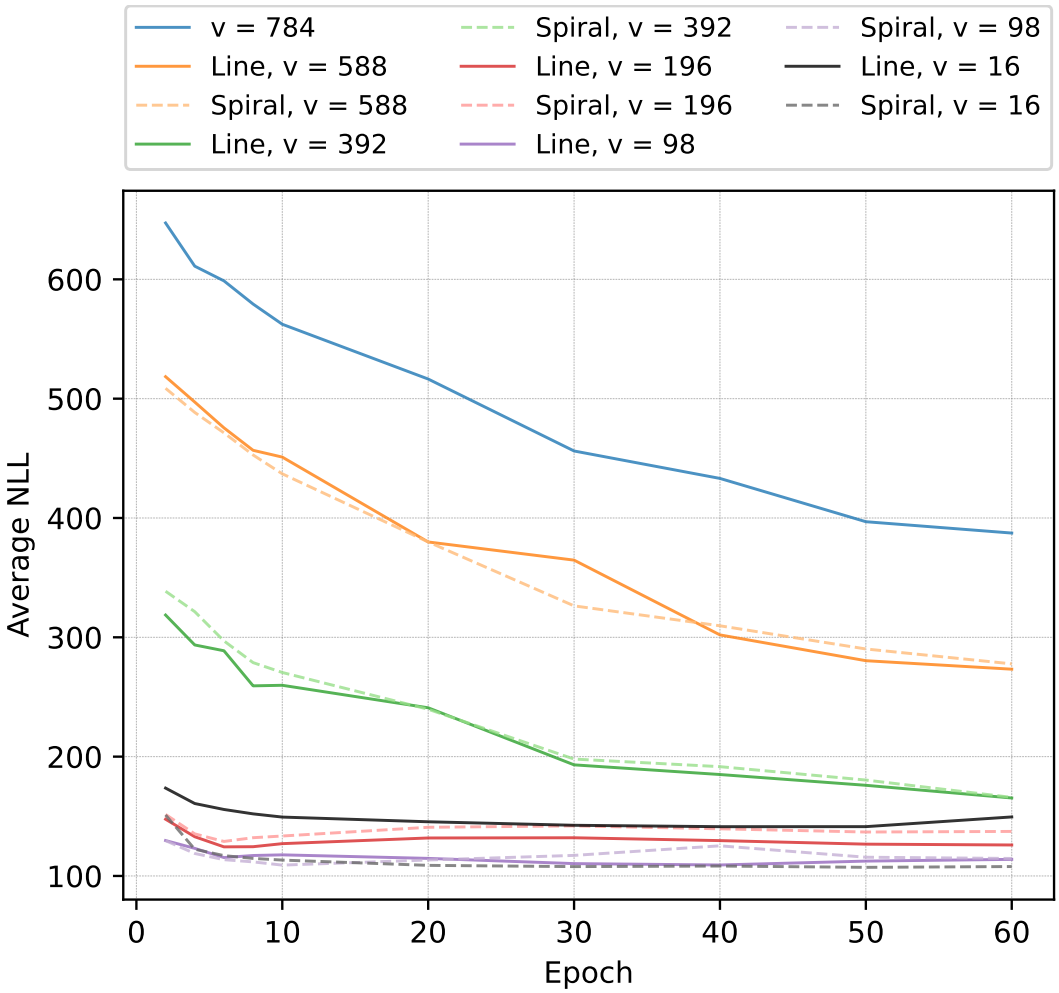


Figure 4.7: Average NLL through training epochs for different numbers of connections and connectivity patterns. RBMs with the same number of connections are shown with different shades of the same color. $v = 784$ corresponds to the fully connected RBM.

epochs, suggesting that their best performance (minimum NLL) was reached early in training (before 10 epochs), which is also a positive observation (after 10 epochs, the average NLL of the fully connected model is 5 times larger).

It is also of note that no significant difference between the line and spiral pattern was observed in the learning curves, with the exception of $v = 16$. In this case, the spiral pattern shows the best performance, together with $v = 98$ results, while the line pattern presents higher NLL values than $v = 196$ curves. Intuitively, with such a small number of connections the differences between the connection patters is greater, and the chosen pattern has greater impact upon the results. Still, the number of connections plays a more important role in the overall training performance, for these patterns and this dataset.

Chapter 5

Network Connectivity Gradient

The previous chapter presented an analysis of the connectivity effect upon the the RBM training, attesting to the importance of choosing a good connectivity pattern in order to enhance learning. Of course, finding the best network connectivity for a given task is not a trivial problem, even in the context of RBMs. It is dependent on the input (training data), the discrete nature of the connections, and the exponentially large space of possible connection patterns (there are 2^{n^2} different networks between two layers with n units each).

In this chapter, a novel method tailored to RBMs is proposed, which is based on the notion of "network gradients". The Network Connectivity Gradient (NCG) method computes the gradient for every possible network connection for any given connectivity pattern. Note that the gradient can be non-zero even when a connection is not present in the pattern. Moreover, NCG uses a continuous parameter to represent the strength of every possible network connection which is updated according to the gradient. Finally, the network strength is thresholded to yield a discrete connectivity pattern *during* optimization (i.e., at each iteration) which in turn determines how information (probabilities) and gradients flow on the model during training.

5.1 Method

The novelty of the proposed method lies on computing a gradient for the connectivity pattern, $\mathbf{A}$. This gradient (Equation 2.5) can be analytically derived in the same manner the other RBM parameters' gradients. The optimization is performed over θ, which in NCG also includes $\mathbf{A}$. In particular, the expectation over the energy gradient is given by

$$
\begin{aligned}
\mathbb{E}_{\mathbf{h}}\left[\nabla_{a_{ij}} E(\mathbf{x},\mathbf{h})\,|\mathbf{x}\right] &= \mathbb{E}_{\mathbf{h}}\left[\nabla_{a_{ij}}(-h_i a_{ij} w_{ij} x_j)\big|\,\mathbf{x}\right] \\
&= -P_\theta(h_i = 1|\mathbf{x}) w_{ij} x_j \\
&= -\sigma(\mathbf{C}_{i\cdot}\mathbf{x} + b_i) w_{ij} x_j \\
&= -\hat{h}_i(\mathbf{x}) w_{ij} x_j \ ;
\end{aligned}
\tag{5.1}
$$

where $\mathbf{C}_{i\cdot}$ is the i-th row in matrix $\mathbf{C}$. This expectation is used to calculate the gradient, given by Equation 2.5. Note that the gradient for a connection (i, j) can be non-zero even when $a_{ij} = 0$. This is a key aspect in the methodology here proposed, since it provides a gradient for absent connections and consequently the possibility for them to be enabled.

However a_{ij} is binary, and thus the usual continuous optimization framework that leverages the gradient to update its value does not apply. To circumvent this limitation, a continuous parameter denoting the connectivity strength is introduced in the model, and represented by $\mathbf{A}' \in [0,1]^{H,X}$ such that $0 \leq a'_{ij} = \mathbf{A}'[i,j] \leq 1$. Thus, the connection strength can be updated using the corresponding gradient (but saturating at 0 or 1). Moreover, the binary connection is a function of the connection strength. In particular, a simple threshold (step) function is used to determine the presence or absence of a connection. This idea leads to the following two-step update rule for the connection parameters in the SGD framework:

$$
\begin{aligned}
a'_{ij} &\leftarrow a'_{ij} + \frac{\alpha_A}{|\mathcal{B}|} \sum_{t \in \mathcal{B}} \left[\hat{h}_i(\mathbf{x}^{(t)}) w_{ij} x_j^{(t)} - \hat{h}_i(\tilde{\mathbf{x}}^{(t)}) w_{ij} \tilde{x}_j^{(t)}\right] \\
a_{ij} &\leftarrow \mathbb{1}\left[a'_{ij} \geq \gamma\right] \ ;
\end{aligned}
\tag{5.2}
$$

where γ is the hyperparameter that denotes the threshold for enabling/disabling a connection based on the connection's strength, $\mathbb{1}[\cdot]$ corresponds to the indicator/step function, and α_A the connectivity learning rate. We have named the method as Network Connectivity Gradient (NCG), and it jointly learns the connectivity pattern and classic model parameters for the RBM. Note that α_A allows to decouple the learning rate of model parameters from the connectivity, which has empirically been shown to bring advantages to the learning curves (discussed in Section 5.5.2).

5.2 Initialization

A fundamental aspect in continuous optimization frameworks such as SGD is the initialization of the parameters that must be optimized. Being parameters, the connectivity pattern and connection strengths must also be initialized. While the fully connected network is a possible initialization, intuitively it may not be the best pattern to start the optimization since it may take too many iterations to remove

connections. A common initialization in the context of the RBM (and other models) is choosing random and small values for the parameters. This approach is also taken for initializing the connection pattern and connection strengths, as follows.

Each connection is randomly initialized according to a probability parameter p. In particular, $a_{ij} = 1$ with probability p and $a_{ij} = 0$ with probability $1 - p$, independently from any other connection. Note that the initial connection pattern is a random bipartite graph where p determines the (expected) edge density of the network. Intuitively, p will influence the learning performance of the RBM, since large/small p can lead to dense/sparse networks that may require many iterations to evolve. Thus, p is a hyperparameter of the initialization procedure.

Once the initial connection pattern has been determined, the connection strengths must also be defined. While initializing $a'_{ij} = a_{ij}$ is a possible initialization, this leads to connection strengths that are either 0 or 1 which may require too many iterations in order to cross the threshold to enable or disable the connection, respectively. A more effective initialization should also to resort to randomness. Thus, connection strengths are randomly initialized given the corresponding connection as follows:

$$a'_{ij} = \mathrm{U}(0, \gamma)\mathbb{1}\left[a_{ij} = 0\right] + \mathrm{U}(\gamma, 1)\mathbb{1}\left[a_{ij} = 1\right] , \tag{5.3}$$

where $\mathrm{U}(a, b)$ is the continuous uniform random value in the interval $[a, b]$. Note that the random value of the connection strength depends on the threshold γ for enabling/disabling the connection. Intuitively, a random value is chosen in the segment corresponding to the connection being absent (range $[0, \gamma]$) or present (range $[\gamma, 1]$). The idea behind this initialization is also to avoid the cold start for the connection strengths while following the (random) initialization of the corresponding connection.

5.3 Implementation

In order to test the proposed method, the RBM model and its training (with or without changing the connectivity network) were implemented on the Programming Language `C++`. The full project code is available at `https://github.com/AmieOliveira/NCG`, and it was tested for MacOS BigSur and Ubuntu 18 operating systems.

Programming in `C++` allows for very efficient coding in comparison to other higher level languages, which is the main reason it was chosen for this project. Since the RBM training is mostly based in matrix operations, the linear algebra library `Eigen` was used as a base, which allows for efficient calculation and dynamic matrix allocation.

The program was implemented with the Object Orientation paradigm, with a class `Data` for the datasets used and a class `RBM` to store the model, with its parameters and training algorithms. The following snippet illustrates the creation of `Data` and `RBM` objects, that are in turn used to train the RBM model.

```
#include "RBM.h"
#include <stdlib.h>
#include <fstream>

using namespace std;

// Creating BAS 4 dataset
int size = 4;
Data bas(DataDistribution::BAS, size);

// Creating RBM with H=X=16 (for the BAS 4 dataset 'bas')
int X = bas.get_sample_size();
int H = X;
RBM model(X, H);

// Training the classical RBM for 10 epochs with CD-10
unsigned seed = 728356;
int k = 10;
int iter = 10;
int b_size = 5;
double l_rate = 0.01;
bool calculate_NLL = true;

model.setRandomSeed(seed);
model.trainSetup(SampleType::CD, k, iter, b_size, l_rate,
    calculate_NLL);
model.fit(bas);

// RBM can be saved for later examining
model.save("myClassicalRBM.rbm");

// Same holds true for training history (NLL through epochs)
vector<double> h = model.getTrainingHistory();
ofstream f;
f.open("nll_history.csv");
f << "NLL" << endl;
for (auto s: h) {
    f << s << endl;
}
f.close();

// Alternatively, you can train with NCG (p = 0.5)
double p = 0.5;
model.optSetup(Heuristic::SGD, p);
model.fit_connectivity(bas);

// And later print the resulting RBM paramters
model.printVariables();
```

The RBM (object `model`) is first trained in the traditional way, with no connectivity change whatsoever, for 10 epochs, and later with NCG for 10 epochs more.

Besides specifying the training parameters (CD-10, batch of size 5, $\alpha = 0.01$), it is important to set the random seed to be used (via method `RBM::setRandomSeed`). Keeping track of the seeds used enables results reproducibility.

Some useful tools are also presented, such as the `RBM::save` method, which allows for saving the RBM weights and biases in a text file, to be further analyzed at a later date or imported in other programs. If you simply wish to dump the RBM parameters you can use the `RBM::printVariables` instead. By using `RBM::getTrainingHistory`, after the model has been trained, you get a `vector` with the NLL values through the training epochs. For large RBMs the value is an AIS estimate.

5.4 Experiments on BAS

In order to validate the method, it was firstly tested upon the BAS model (Section 2.4.1). The dataset BAS 5 with 25 hidden units was used ($H = X = 25$), for which we are still able to calculate the exact NLL. Training was performed using CD with 10 steps of Gibbs Sampling (CD-10). As with previous BAS experiments, the learning rate was set to $\alpha = 0.01$ and the mini-batches use 5 random samples for each update, with no momentum or weight decay (see Section 4.2). The decoupled connectivity learning rate was set to 5 times the network's connectivity, obtaining $\alpha_A = 0.05$, and the training threshold was set to a neutral $\gamma = 0.5$ (it does not favor either sparser or denser networks). A training epoch iterates batches until all the dataset is used to update the model. The NLL was calculated at every 5 epochs.

The RBM is initialized with null biases and small random weights uniformly distributed between -1 and $+1$. Each experiment was repeated 25 times to account for statistical fluctuation, and therefore the plots show the sample average as the lines for a set of experiments, and the distribution quartiles as the shades (as per the quartile definition, 50% of the results fall within the shaded area).

Although better hyperparameter fine-tuning could improve the overall RBM performance, this test aims at validating the NCG method behavior and comparing its performance to the traditional, unmodified model, and therefore one does not need the best set of hyperparameters.

Figure 5.1 shows the evolution of the NLL through the training epochs for the fully connected RBM and three NCG initializations. This result is qualitatively similar to the ones obtained in Section 4.2.4, for the traditional network has a steeper initial NLL decrease, but after 2 k epochs the learning curves invert themselves and NCG trainings with $p = 0.5$ and $p = 1$ end with smaller NLL means, and appear to have better performance.

While the NCG method is able to eventually surpass the fully connected model,

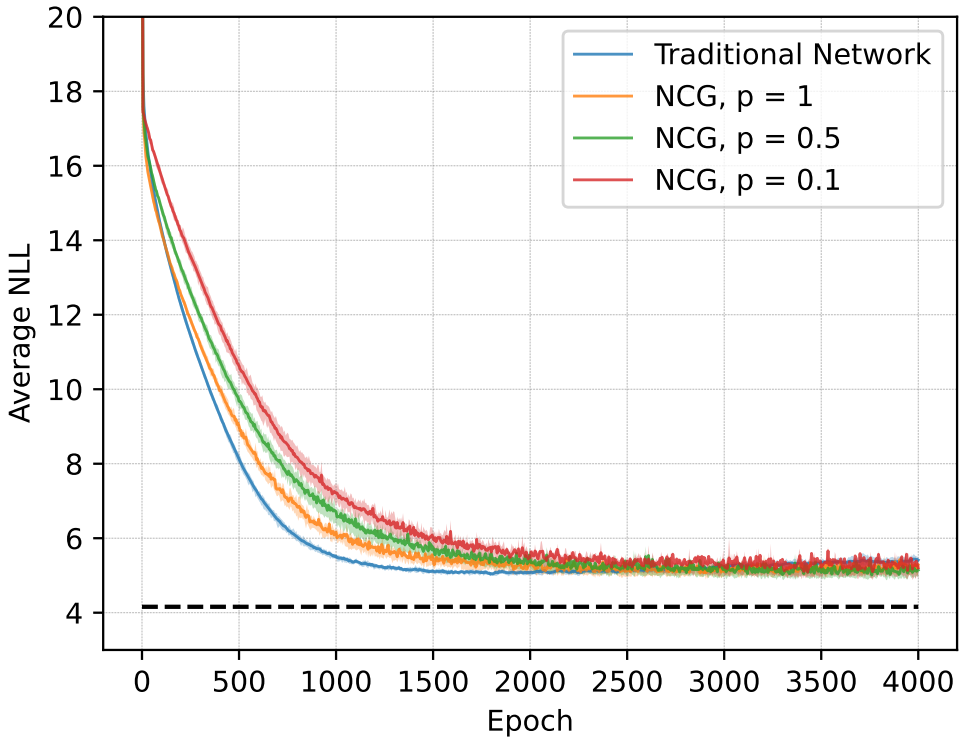


Figure 5.1: Average NLL over the training epochs for different NCG initializations. Training performed over BAS 5 dataset. The black dashed line corresponds to the data (optimal) average NLL.

these results still do not show much advantage in utilizing it. The increase in performance is minimal, and comes coupled not only with a higher computational load, but also with an increase in the noise of the learning curves.

However, this is a very small model, with a fairly simple dataset. This is not the ideal condition to see the potential benefits of changing the connectivity pattern. Section 4.3 shows that on the MNIST dataset the performance gain is significantly higher than on BAS. The next section aims to evaluate this method on MNIST, for which it is expected that there is much more to gain from introducing the connectivity optimization.

5.5 Experiments on MNIST

Since the BAS dataset is artificial and very different from what one is likely to find in real world scenarios, Sections 5.5.1 and 5.5.2 present experiments performed upon the MNIST dataset (Section 2.4.2), considering the sample generation task (average NLL is the performance metric) and the image classification task (accuracy is the performance metric), respectively.

In the sample generation task, a classic generative RBM is trained as to generate random samples similar to the input examples. The performance is assessed using

the average NLL across the training set. However, since the exact average NLL cannot be computed due to intractability of the normalization constant, the Annealed Importance Sampling (AIS) method is used to derive an approximation [12].

In the classification task, the RBM is trained to classify the digit in the input image. The RBM is expanded to have 10 additional visible (input) units in order to encode the label of the image during training. Note that exactly one of the additional visible units is activated for each input sample, the one corresponding to the digit (from 0 to 9) in the image. The objective function used in this task was the same as in the previous task, and is given by the Contrastive Divergence equation (see Eq. 2.5).

Each image in the dataset has 784 pixels, each of which corresponds to a visible unit of the RBM. All experiments use 500 hidden units, and training was achieved using CD with 10 steps of Gibbs sampling (CD-10), or with CD-1, when thus specified. The learning rate for the model parameters was set to $\alpha = 0.1$ and mini batches to size 50. The connectivity learning rate was set to $\alpha_A = 0.5$, with the exception of one of the experiments in Section 5.5.2, for which $\alpha_A = \alpha = 0.1$ was used. No momentum or weight decay were used.

The RBMs weight parameters were initialized with null biases and small random weights, uniformly distributed between $[-1, 1]$. For the connection threshold in NCG, $\gamma = 0.5$ was adopted as this is the midpoint value in the possible range for the connection strengths, not favoring either a more sparse ($\gamma > 0.5$) or dense network ($\gamma < 0.5$). The connections in NCG were initialized using three values for $p \in \{1.0, 0.5, 0.1\}$, which includes initializing with a fully connected network, when $p = 1$. Note that in the classification task, the 10 visible units corresponding to the label of the image are always connected to all hidden units, and these connections are not subject to optimization.

During training, one epoch corresponds to one iteration over the entire training dataset with the model's parameters being updated at every batch. Since the batch size was 50 data samples, an epoch corresponds to 1200 parameter updates necessary to iterate over the 60 k samples in the dataset. Batch elements are randomly determined for every epoch. The normalization constant used to compute the average NLL was approximated using AIS with 100 runs and 14.5 k intermediate distributions.

Each generative experiment was repeated 10 times and each classification experiment was repeated 25 times. The sample mean performance (lines) along with the sample distribution quartiles (shades) are reported (50% of the results are within the shaded area).

Once again, although fine-tuning these parameters could potentially improve the learning performance of the RBM, the goal here is to compare NCG with the fully

connected RBM, and not necessarily obtain the best model.

5.5.1 Generative Results on MNIST

The first set of experiments performed upon the MNIST dataset are similar to the ones' observed in Section 4.3, in which the NLL that the RBM achieves throughout training is analyzed, and the performance obtained by NCG is compared with the fully connected, traditional RBM.

Figure 5.2 shows the learning curve (evolution of the average NLL over the epochs) for the classic fully connected RBM and three initializations of the NCG method. Clearly, the fully connected network exhibits a significantly worse learning curve, both in terms of sample mean and variance. Interestingly, the three different initializations for NCG exhibit a very similar performance (with the exception of apparent outliers observed for the case $p = 1$).[1] While the mean performance for $p = 0.1$ could be said to be slightly better, the overlapping quartiles show that the sparsity of the random initialization is not particularly important in this scenario. Indeed, the similar learning curves for very different initial networks indicates that NCG can find effective networks independent of the (random) initial connectivity pattern.

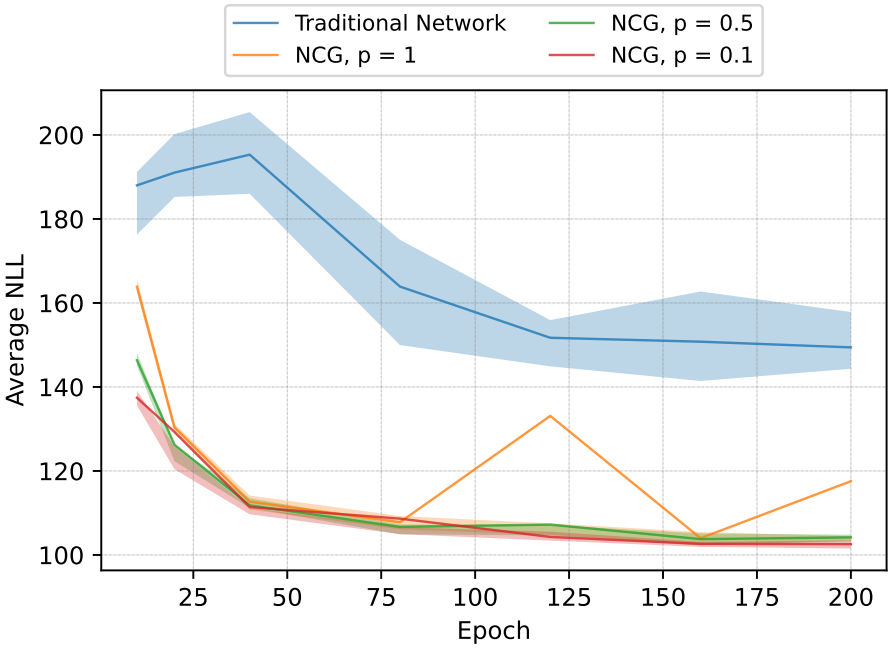
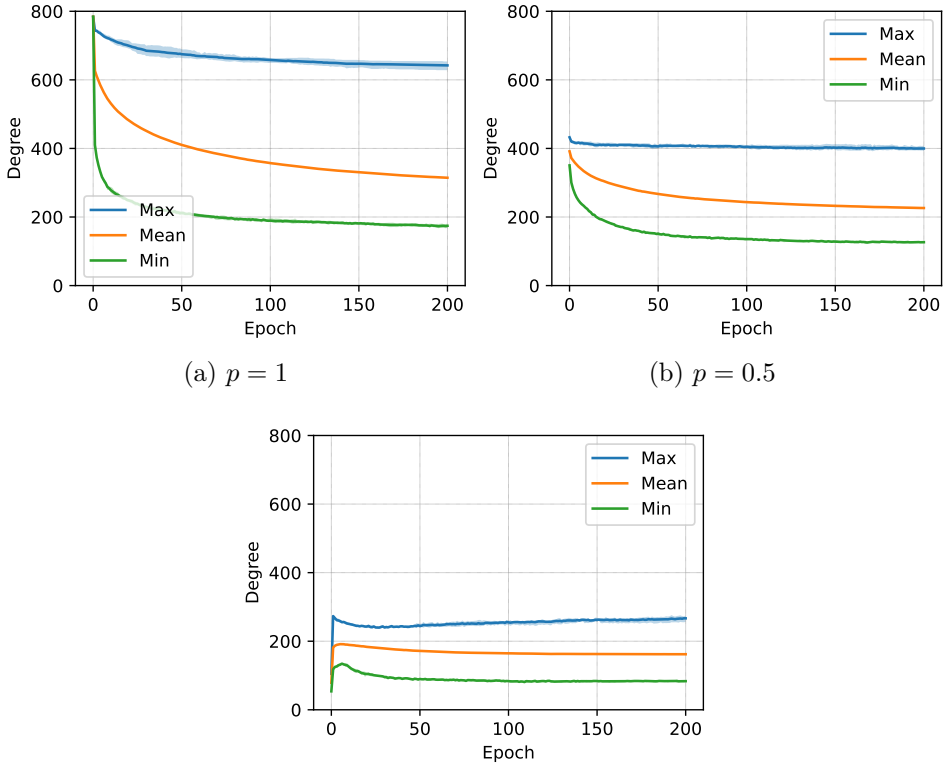


Figure 5.2: Average NLL over the training epochs for different NCG initializations. Training performed over MNIST dataset with CD-10.

Despite the similar learning curves, the evolution of the network degrees is very dissimilar across the different initializations. Figure 5.3 shows the evolution of the

[1] NCG training with $p = 1$ initialization showed 2 of the 10 runs with much higher than average NLL at epoch 120 and 1 of the 10 runs at epoch 200.



(a) $p = 1$ (b) $p = 0.5$

(c) $p = 0.1$

Figure 5.3: Degree statistics (minimum, average, maximum) of the hidden units over the training epochs for three different NCG initializations. Generative results on MNIST, trained using CD-10.

maximum, minimum and average degree of the hidden units for the three initializations. For $p = 1$ a sharp decrease is observed in all three statistics in the first 10 epochs, with the curves indicating a slight decay even aver 200 epochs (in particular the average degree). For $p = 0.5$, the initial decrease is not as strong and the curves indicate convergence after 200 epochs. Interestingly, the case $p = 0.1$ shows an increase in all three statistics in the first 10 epochs and convergence after 200 epochs. This indicates that NCG can not only prune connections but also *add* connections when the network is too sparse. However, the average degree of the network after 200 epochs depends on the initialization, and is proportional to the density of the initial network.

The similar learning curves but different network patterns indicate that the joint optimization of model parameters and network connectivity can compensate for one another, leading to similar performance even when the connectivity pattern is different. Indeed, recent results in network pruning suggest that different network patterns can often achieve similar performance [9]. Moreover, the fast initial change in the connectivity pattern is related to the relatively large connectivity learning rate, α_A, in comparison to the learning rate of other model parameters (i.e., 5 times

larger). The next section explores this hyperparameter.

Contrastive Divergence Approximation

The previous experiments all used the CD-10 approximation, but it is well known that the number of steps of Gibbs Sampling performed has fundamental influence over training performance. Therefore, this next set of experiments attempts to evaluate what is the impact of using CD-1 training (1 step of Gibbs sampling) upon the NCG method.

Figure 5.4 portrays the learning curves of the traditional and NCG RBMs, and 5.5 the corresponding degree statistics. It is clear that the change to CD-1 results in a major performance drop for all the considered models: by the end of training the fully connected RBM has the average NLL around 290 in comparison to the 150 previously achieved, and the models trained with the NCG method reach at most 150, when before the worse average did not surpass 120. The degree statistics for $p = 1$ and $p = 0.5$ appear to show less change than what was observed in CD-10, but not dramatically so.

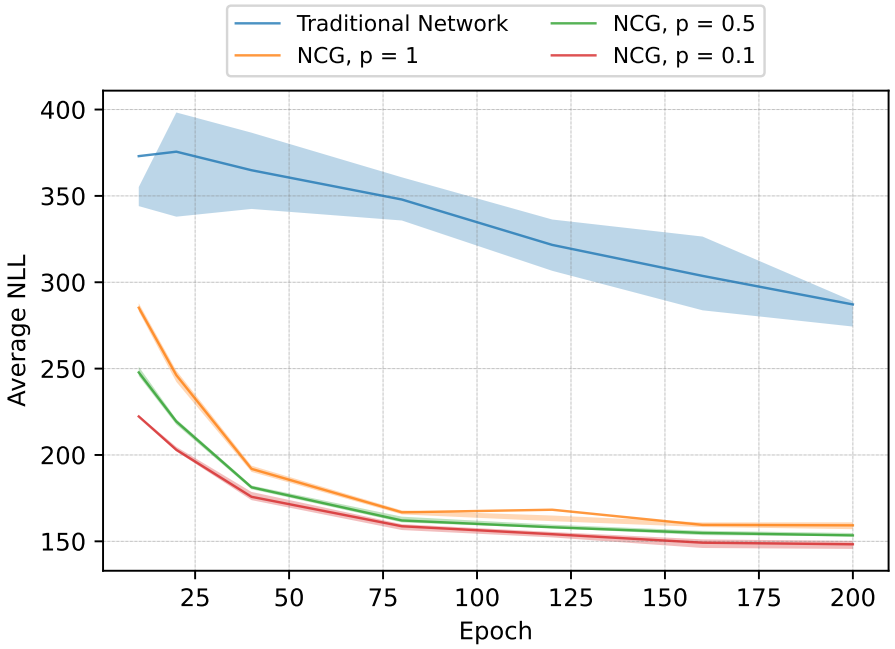
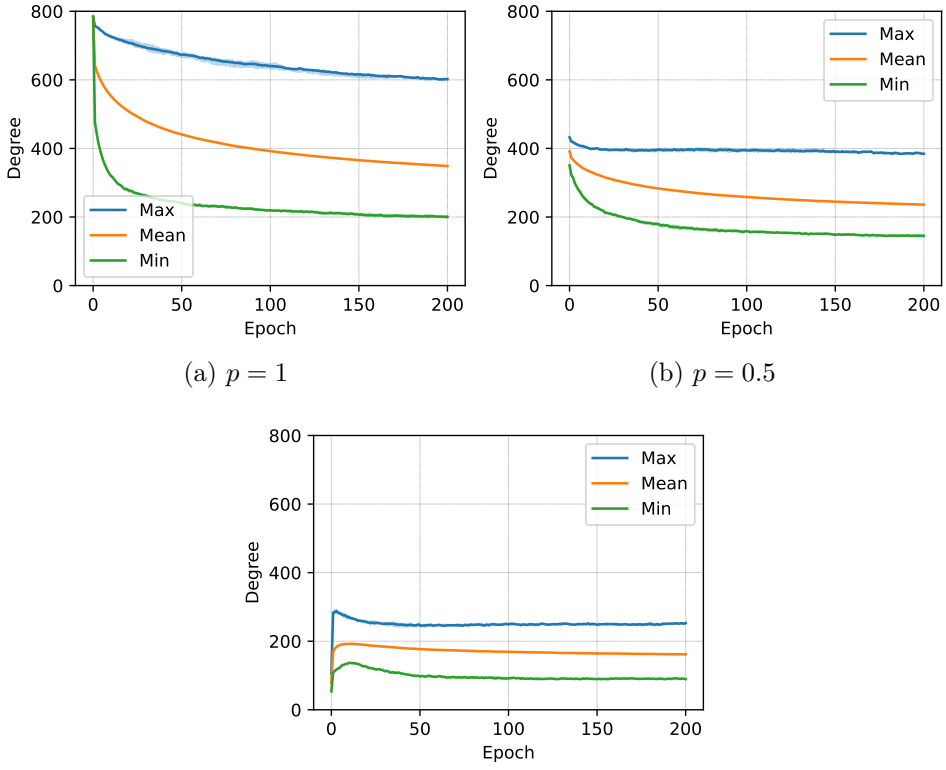


Figure 5.4: Average NLL over the training epochs for different NCG initializations. Training performed over MNIST dataset with CD-1.

Interestingly, the NCG models' NLL values suffer less increase than the classical RBM, for which the end result is double the CD-10 value, and the relative increase in performance derived from optimizing the connectivity in this instance increases.



(a) $p = 1$ (b) $p = 0.5$

(c) $p = 0.1$

Figure 5.5: Degree statistics (minimum, average, maximum) of the hidden units over the training epochs for three different NCG initializations. Generative results on MNIST, trained using CD-1.

5.5.2 Classification Results on MNIST

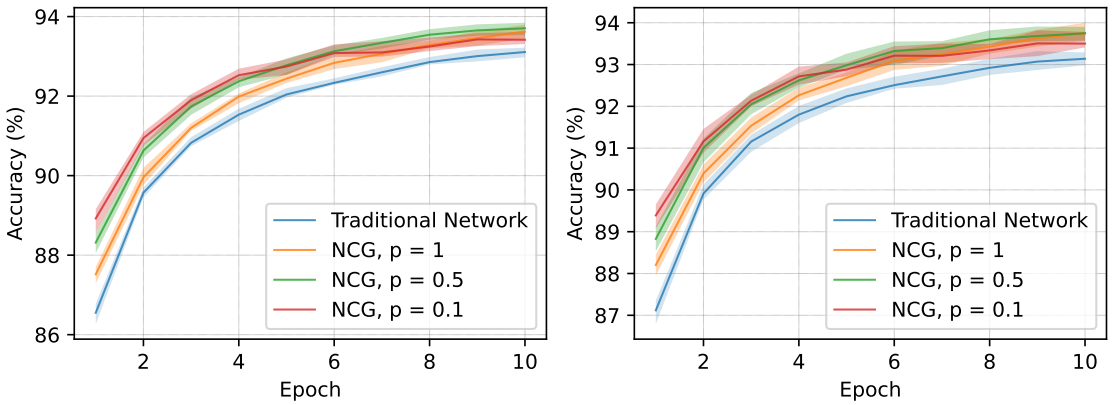
While the generative task relied on the approximate average NLL to measure the learning performance of the RBM, the classification task uses a direct and easy to compute performance metric: the classification accuracy. Thus, the model is trained to classify the digit in the input image, and the accuracy of the model is simply the fraction of images correctly classified.

Recall that for the classification task, 10 additional visible units are added to the input, each corresponding to a digit (from 0 to 9). The connections between these visible units and all hidden units are not subject to optimization, as they are crucial for the classification task. In particular, the RBM is trained using Contrastive Divergence as in the generative task, and is not a priori aware of the classification task.

Classification is performed by presenting the image to the RBM, setting each label units to 0.5, calculating the probabilities of each hidden unit being activated, and finally selecting the label unit (digit) with the higher probability of being activated. This digit is the predicted label for the image.

Figure 5.6 shows the evolution of the classification accuracy over the epochs for

different models for the training and test sets (the test set accuracy is obtained using the model trained up to the corresponding epoch). Note that all three NCG initializations generate models that appear consistently better than the fully connected RBM, for both the training and test sets, with higher accuracy means and no overlap of the uncertainty's areas at the end of 10 epochs. Moreover, the performance in the training and test set are qualitative and quantitatively similar, indicating there is likely no overfitting occurring.

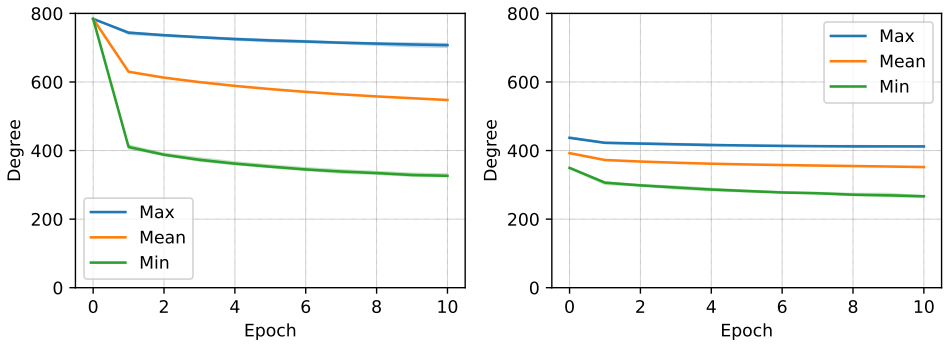


(a) Train Set (b) Test Set

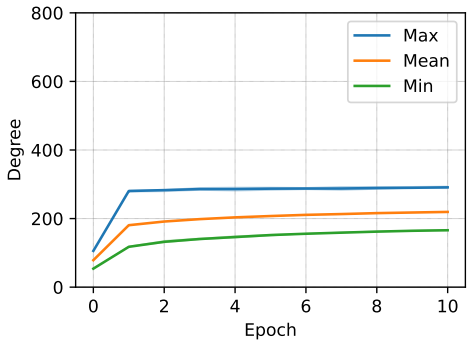
Figure 5.6: Classification accuracy over the training epochs for different NCG initializations. Results shown for the train (a) and test (b) sets. Trained with CD-10 and $\alpha_A = 0.5$.

Interestingly, results show that accuracy is inversely proportional to the initial density during the first epochs of training: initializing the network with fewer connections yields superior accuracy in early stages of training. However, as the number of epochs increase, the accuracy between the NCG models becomes more similar. In fact, for 10 epochs the model initialized with $p = 0.5$ has slightly superior performance. This indicates that NCG is capable of overcoming a poorly initialized connectivity pattern by adjusting the connections and model weights.

Figure 5.7 portrays the degree statistics (minimum, average, and maximum) of the network's hidden units over the epochs for three initializations. Note that for $p = 1$ all degrees are 784 at time zero, and NCG significantly reduces the degrees of the network; the average degree is reduced by 30% after 10 epochs. On the other hand, for $p = 0.1$, NCG significantly increases the degrees of the network; the average degree is 2.5 times larger after 10 epochs. Finally, for $p = 0.5$ NCG shows a relatively small change in the degrees. Moreover, while the degrees change and converge over the epochs, the initialization density has a strong influence: the average degree of the three models after 10 epochs reflects their initial density.



(a) $p = 1$ (b) $p = 0.5$



(c) $p = 0.1$

Figure 5.7: Degree statistics (minimum, average, maximum) of the hidden units over the training epochs for three different NCG initializations. Classification results trained with CD-10 and $\alpha_A = 0.5$.

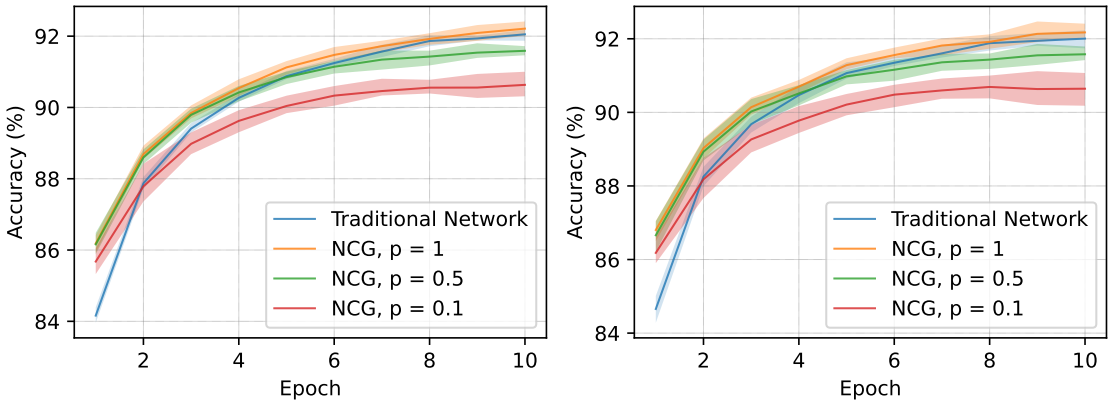
Contrastive Divergence Approximation

As in the generative experiments, we wish to know how the Contrastive Divergence approximation, as given by the number of Gibbs Sampling steps performed, affects the NCG method for this alternative task.

In the classification task, NCG is already at a disadvantage, because its goal, maximize the accuracy, is not the objective function used during training with CD, which aims to minimize the NLL, and is an approximation to boot. Therefore NCG trains the connectivity network for a slightly inaccurate objective, as well as all the other parameters that the traditional network trains. It stands to reason, therefore, that in worsening the approximation, its performance might suffer.

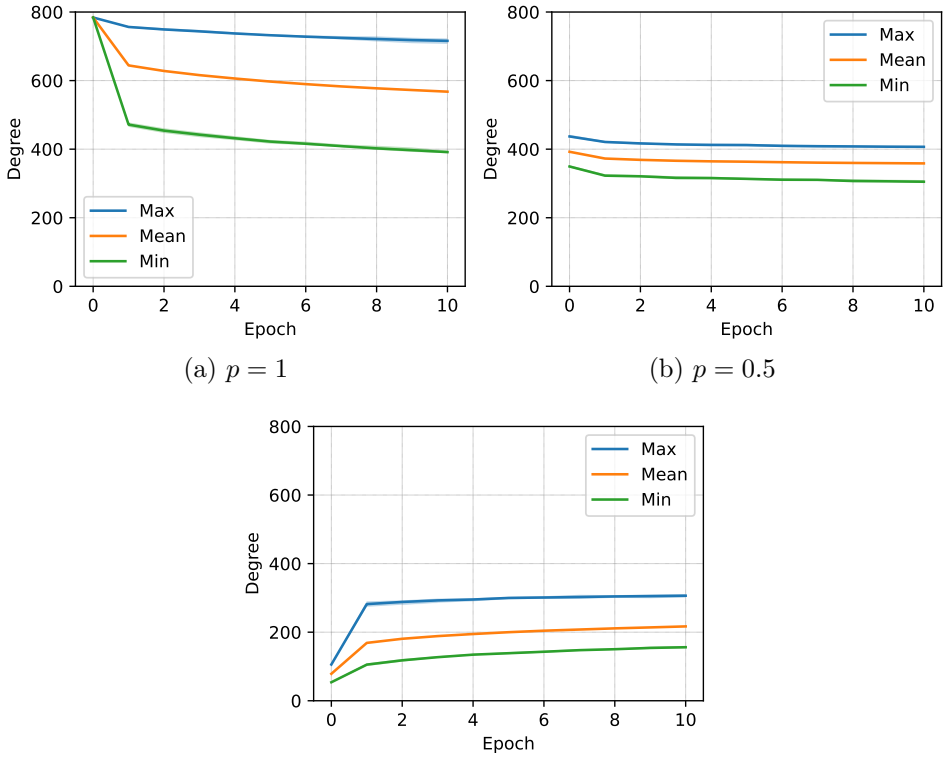
One can see in Figure 5.8 the evolution of the accuracy over epochs for the traditional RBM as well as three NCG experiments with initializations, for both the train and test sets. Figure 5.9 presents the corresponding degree statistics evolution, giving an idea of how the connectivity changes with training. Once again, the degree statistics do not show much difference from their CD-10 counterparts, except that they suffer less change through the training.

Although all RBMs have a worse accuracy performance for this training, it is



(a) Train Set (b) Test Set

Figure 5.8: Classification accuracy over the training epochs for different NCG initializations. Results shown for the train (a) and test (b) sets. Trained with CD-1 and $\alpha_A = 0.5$.



(a) $p = 1$ (b) $p = 0.5$

(c) $p = 0.1$

Figure 5.9: Degree statistics (minimum, average, maximum) of the hidden units over the training epochs for three different NCG initializations. Classification results trained with CD-1 and $\alpha_A = 0.5$.

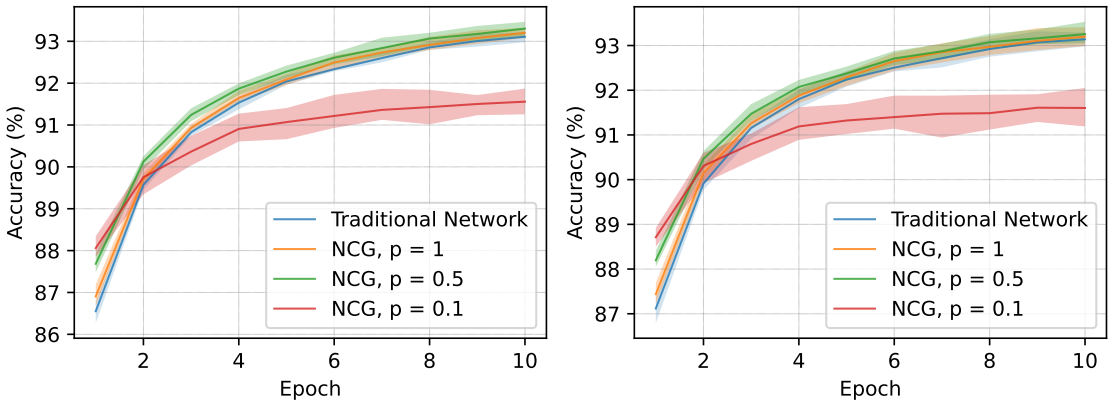
clear from the images that the NCG method loses also relative performance with respect to the fully connected model. In these circumstances, only NCG initializing with all connections activated ($p = 1$) manages to surpass the traditional RBM, and even then they have very close results. It is not clear that the difference is

statistically significant. The $p = 0.1$ training seems to suffer the most, not managing a better accuracy even in the first epoch of training. Overall, we see an altogether different situation as the one observed in the generative task, for which the addition of connectivity optimization resulted only in positive results, regardless of the CD approximation used.

Connectivity Learning Rate

The relatively higher connectivity learning rate $\alpha_A = 0.5$ plays an important role in allowing the network to evolve fast in the early stages of training. Intuitively, this allows NCG to quickly adjust for poor initial network patterns before other model parameters start to converge.

Figure 5.10 shows the accuracy when using a connectivity learning rate of $\alpha_A = 0.1$, which is equal to the learning rate of other model parameters. Note the decrease in the accuracy for all three initializations for all 10 epochs (in comparison to Figure 5.6).

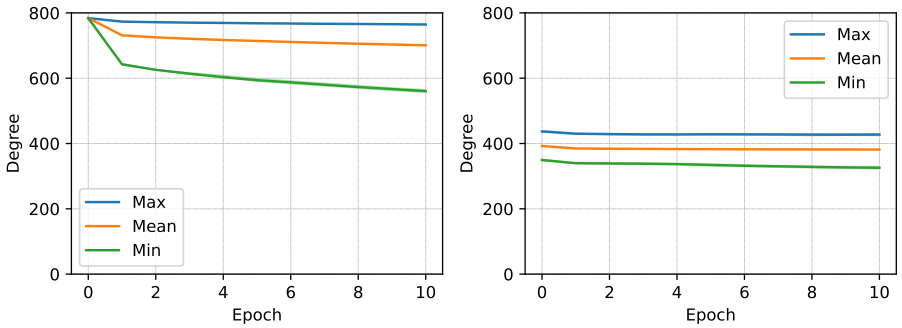


(a) Train Set (b) Test Set

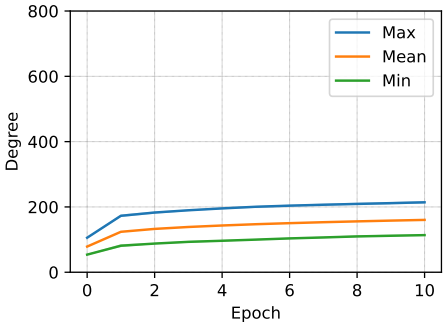
Figure 5.10: Classification accuracy over the training epochs for different NCG intializations. Results shown in the train (a) and test (b) sets. Trained with CD-10 and $\alpha_A = 0.1$.

Interestingly, while the performance for $p = 0.1$ is superior after 1 epoch of training (as with $\alpha_A = 0.5$), the model fails to improve its accuracy as in the previous experiment and falls behind the other models, including the fully connected network. Intuitively, the model cannot adjust its connection pattern fast enough and the connectivity gradient becomes subdued by other model parameters. This is corroborated by the results observed in Figure 5.11, that has the degree statistics evolution through training. It is clear, when compared with Figure 5.7, that there is a much slower connectivity update.

This example highlights the importance of decoupling the learning rates when jointly optimizing network connectivity and other model parameters.



(a) $p = 1$ (b) $p = 0.5$



(c) $p = 0.1$

Figure 5.11: Degree statistics (minimum, average, maximum) of the hidden units over the training epochs for three different NCG initializations. Classification results trained with CD-10 and $\alpha_A = 0.1$.

Chapter 6

Conclusions

This work presented a broad analysis of the Connectivity Network of the Restricted Boltzmann Machine (RBM), first verifying that this hyperparameter plays a fundamental role in its learning performance, and then by proposing and perusing the Network Connectivity Gradient (NCG) method, designed to learn the optimal connectivity jointly with other model parameters (weights and biases).

Experiments with different connectivity patterns and different number of connections (per unit) revealed that the average NLL for both the synthetic BAS model and the MNIST dataset strongly depends on the network connectivity. In particular, results demonstrate that connectivity pattern and number of connections play independent roles: (1) two patterns with the same number of connections can have very different performance; (2) a single pattern using two different number of connections can have very different performance. Interestingly, for a given pattern the learning performance is not monotonic on the number of connections, in the sense that having less connections than the fully connected network can exhibit increased performance, allowing for faster and/or better learning. Moreover, for larger models (BAS 8 and MNIST) the learning performance of the RBM depends even more on network connectivity given that differences in patterns and number of connections are magnified.

NCG computes gradients for each possible network connection given a connectivity pattern. The gradients are used to drive the continuous connectivity strength parameter that in turn determines to maintain, add or remove the connection. NCG requires no change in RBM's objective function nor its classic optimization framework. Despite not showing much improvement on the BAS 5 dataset, evaluation of NCG on a generative and classification task using MNIST data demonstrated its effectiveness in learning better models (learning faster and better), and also robustness with respect to initialization.

6.1 Future Work

There are many more analyses that can still be made upon the NCG method. For example, we wish to evaluate what is its performance using other types of initialization, in particular deterministic initializations. It would be interesting to observe what is the method's performance if the RBM already starts with a "good" connectivity pattern. Will it surpass the training without connectivity optimization or perhaps fall short? This analysis could include both the neighbors patterns proposed in Chapter 4 and others obtained through the analysis of the data, such as techniques used to prune neural networks at the initialization phase [25, 26, 46]. We believe that these changes can greatly improve NCG's performance.

Furthermore, we wish to further explore both the connectivity and the default training learning rates. These are key paramters, and can greatly improve the RBM learning. In this work only static learning rates have been used, but it is known that better results are yielded by using dynamic rates, which start with high values, providing a fast initial improvement, and decrease with learning time, so that a better parameter fine-tuning can be achieved.

It is also important to address the issue of determining which connectivity learning threshold should be used. We have utilized the intuitive value of 0.5, but it is not clear that this is the best for training. It is important that the variation of the threshold be explored. Favoring of a more sparse network via the use of a higher threshold could be beneficial to training, as the works on network pruning suggest. Moreover, the optimal threshold might not be symmetrical: hysteresis could help avoid oscillations and make the connectivity optimization more dynamic.

References

[1] DAI, Z., LIU, H., LE, Q. V., et al. "CoAtNet: Marrying Convolution and Attention for All Data Sizes". In: *Advances in Neural Information Processing Systems (NIPS)*, 2021.

[2] BROWN, T., MANN, B., RYDER, N., et al. "Language Models are Few-Shot Learners". In: *Advances in Neural Information Processing Systems (NIPS)*, pp. 1877–1901, 2020.

[3] HE, K., ZHANG, X., REN, S., et al. "Deep Residual Learning for Image Recognition". In: *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, 2016.

[4] DEVLIN, J., CHANG, M.-W., LEE, K., et al. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". In: *Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*, pp. 4171–4186, 2019.

[5] LINDAUER, M., HUTTER, F. "Best Practices for Scientific Research on Neural Architecture Search", *Journal of Machine Learning Research*, v. 21, n. 243, pp. 1–18, 2020.

[6] FUKUSHIMA, K., MIYAKE, S. "Neocognitron: A Self-Organizing Neural Network Model for a Mechanism of Visual Pattern Recognition". In: *Competition and Cooperation in Neural Nets*, pp. 267–285, 1982.

[7] LECUN, Y., BENGIO, Y., HINTON, G. "Deep learning", *Nature*, v. 521, n. 7553, pp. 436–444, 2015.

[8] REED, R. "Pruning Algorithms–A Survey", *IEEE Transactions on Neural Networks*, v. 4, n. 5, pp. 740–747, 1993.

[9] BLALOCK, D., GONZALEZ ORTIZ, J. J., FRANKLE, J., et al. "What is the State of Neural Network Pruning?" In: *Machine Learning and Systems (MLSys)*, pp. 129–146, 2020.

[10] CÔTÉ, M.-A., LAROCHELLE, H. "An Infinite Restricted Boltzmann Machine", *Neural computation*, v. 28, n. 7, pp. 1265–1288, 2016.

[11] FISCHER, A., IGEL, C. "Training Restricted Boltzmann Machines: An introduction", *Pattern Recognition*, v. 47, n. 1, pp. 25–39, 2014.

[12] SALAKHUTDINOV, R., MURRAY, I. "On the Quantitative Analysis of Deep Belief Networks". In: *International Conference on Machine Learning (ICML)*, pp. 872–879, 2008.

[13] GRCIĆ, M., GRUBIŠIĆ, I., ŠEGVIĆ, S. "Densely connected normalizing flows". In: *Advances in Neural Information Processing Systems (NIPS)*, 2021.

[14] XIE, Q., DAI, Z., HOVY, E., et al. "Unsupervised Data Augmentation for Consistency Training". In: *Advances in Neural Information Processing Systems (NIPS)*, pp. 6256–6268, 2020.

[15] ELSKEN, T., METZEN, J. H., HUTTER, F. "Neural Architecture Search: A Survey." *Journal of Machine Learning Research*, v. 20, n. 55, pp. 1–21, 2019.

[16] LIU, H., SIMONYAN, K., YANG, Y. "DARTS: Differentiable Architecture Search". In: *International Conference on Learning Representations (ICLR)*, 2019.

[17] YING, C., KLEIN, A., CHRISTIANSEN, E., et al. "NAS-Bench-101: Towards Reproducible Neural Architecture Search". In: *International Conference on Machine Learning (ICML)*, pp. 7105–7114, 2019.

[18] DONG, X., LIU, L., MUSIAL, K., et al. "NATS-Bench: Benchmarking NAS Algorithms for Architecture Topology and Size", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, v. PP, pp. 1–1, 2021.

[19] FANG, J., SUN, Y., ZHANG, Q., et al. "Densely Connected Search Space for More Flexible Neural Architecture Search". In: *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 10628–10637, 2020.

[20] ABU-MOSTAFA, Y. S., MAGDON-ISMAIL, M., LIN, H.-T. *Learning from data*, v. 4. 2012.

[21] LIANG, T., GLOSSNER, J., WANG, L., et al. "Pruning and quantization for deep neural network acceleration: A survey", *Neurocomputing*, v. 461, pp. 370–403, 2021.

[22] LECUN, Y., DENKER, J. S., SOLLA, S. A. "Optimal Brain Damage". In: *Advances in Neural Information Processing Systems (NIPS)*, pp. 598–605, 1990.

[23] HAN, S., POOL, J., TRAN, J., et al. "Learning Both Weights and Connections for Efficient Neural Networks". In: *Advances in Neural Information Processing Systems (NIPS)*, p. 1135–1143, 2015.

[24] FRANKLE, J., CARBIN, M. "The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks". In: *International Conference on Learning Representations (ICLR)*, 2019.

[25] LEE, N., AJANTHAN, T., TORR, P. "SNIP: Single-shot Network Pruning based on Connection Sensitivity". In: *International Conference on Learning Representations (ICLR)*, 2019.

[26] DE JORGE, P., SANYAL, A., BEHL, H. S., et al. "Progressive skeletonization: Trimming more fat from a network at initialization". In: *International Conference on Learning Representations (ICLR)*, 2021.

[27] SAVARESE, P., SILVA, H., MAIRE, M. "Winning the Lottery with Continuous Sparsification". In: *Advances in Neural Information Processing Systems (NIPS)*, pp. 11380–11390, 2020.

[28] CHEN, T., SUI, Y., CHEN, X., et al. "A Unified Lottery Ticket Hypothesis for Graph Neural Networks". In: *International Conference on Machine Learning (ICML)*, pp. 1695–1706, 2021.

[29] ZHOU, A., MA, Y., ZHU, J., et al. "Learning N:M Fine-grained Structured Sparse Neural Networks From Scratch". In: *International Conference on Learning Representations (ICLR)*, 2021.

[30] SMOLENSKY, P. "Information Processing in Dynamical Systems: Foundations of Harmony Theory". In: *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 1: Foundations*, p. 194–281, 1986.

[31] DECELLE, A., FURTLEHNER, C. "Restricted Boltzmann Machine: Recent advances and mean-field theory", *Chinese Physics B*, v. 30, n. 4, pp. 040202, 2021.

[32] LANDAU, L. D., LIFSHITZ, E. M. "Chapter III - The Gibbs Distribution". In: *Statistical Physics*, pp. 79–110, 1980.

[33] ROUX, N. L., HEESS, N., SHOTTON, J., et al. "Learning a generative model of images by factoring appearance and shape", *Neural Computation*, v. 23, n. 3, pp. 593–650, 2011.

[34] TANG, Y., SALAKHUTDINOV, R., HINTON, G. "Robust Boltzmann Machines for recognition and denoising". In: *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2264–2271, 2012.

[35] TIELEMAN, T. "Training Restricted Boltzmann Machines Using Approximations to the Likelihood Gradient". In: *International Conference on Machine Learning (ICML)*, p. 1064–1071, 2008.

[36] LAROCHELLE, H., MANDEL, M., PASCANU, R., et al. "Learning Algorithms for the Classification Restricted Boltzmann Machine", *Journal of Machine Learning Research*, v. 13, n. 1, pp. 643–669, 2012.

[37] MIDHUN, M. E., NAIR, S. R., PRABHAKAR, V. T. N., et al. "Deep Model for Classification of Hyperspectral Image Using Restricted Boltzmann Machine". In: *International Conference on Interdisciplinary Advances in Applied Computing (ICONIAAC)*, pp. 1–7, 2014.

[38] QIANG, N., DONG, Q., ZHANG, W., et al. "Modeling task-based fMRI data via deep belief network with neural architecture search", *Computerized Medical Imaging and Graphics*, v. 83, pp. 101747, 2020.

[39] BOTTOU, L. "Large-Scale Machine Learning with Stochastic Gradient Descent". In: *International Conference on Computational Statistics (COMPSTAT)*, pp. 177–186, 2010.

[40] HINTON, G. E. "A Practical Guide to Training Restricted Boltzmann Machines". In: *Neural networks: Tricks of the trade*, pp. 599–619, 2012.

[41] HINTON, G. E. "Training Products of Experts by Minimizing Contrastive Divergence", *Neural computation*, v. 14, n. 8, pp. 1771–1800, 2002.

[42] PAPA, J. P., ROSA, G. H., COSTA, K. A., et al. "On the Model Selection of Bernoulli Restricted Boltzmann Machines Through Harmony Search". In: *Conference on Genetic and Evolutionary Computation (GECCO)*, p. 1449–1450, 2015.

[43] SAVARESE, P. H. P., KAKODKAR, M., RIBEIRO, B. "From Monte Carlo to Las Vegas: Improving Restricted Boltzmann Machine Training through Stopping Sets". In: *AAAI Conference on Artificial Intelligence*, pp. 4016–4025, 2018.

[44] MACKAY, D. J. C. *Information Theory, Inference and Learning Algorithms*. 2003.

[45] METROPOLIS, N. "The Beginning of the Monte Carlo Method". In: *Los Alamos Science Special Issue*, v. 15, pp. 125–130, 1987.

[46] WANG, C., ZHANG, G., GROSSE, R. "Picking winning tickets before training by preserving gradient flow". In: *International Conference on Learning Representations (ICLR)*, 2020.

Appendix A

Connectivity Analysis Quartile Figures

This appendix contains the missing quartile figures from Chapter 4, organized by the sections on which the original plots are displayed.

A.1 Analysis on BAS

A.1.1 Number of Connections

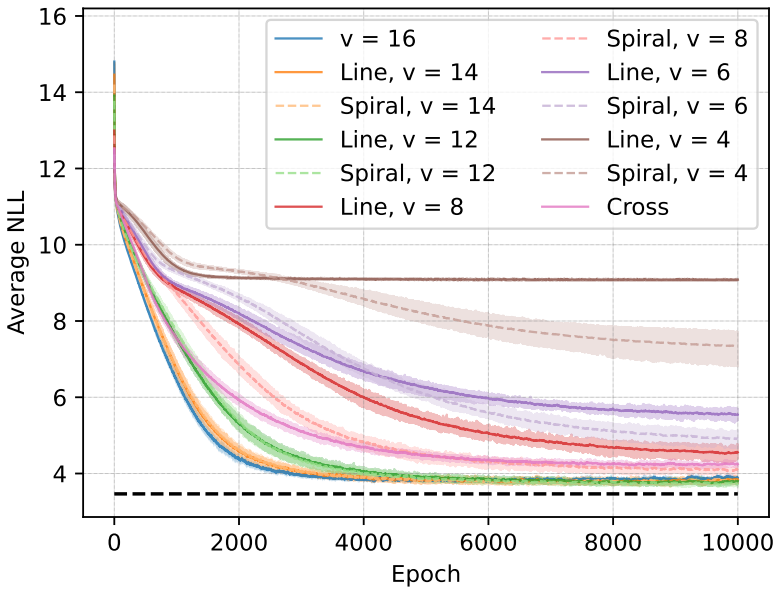
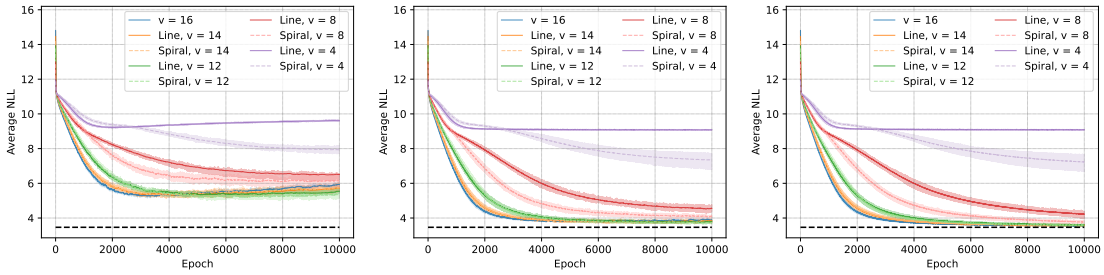


Figure A.1: (Regarding Figure 4.4) Evolution of the average NLL over training epochs for different numbers of connections for the line (full curves) and spiral patterns (dashed curves). The cross pattern is also added. RBMs with the same number of connections are shown with different shades of the same color. The network with $v = 16$ is the same for both patterns and corresponds to the fully connected traditional RBM. The black dashed line corresponds to the data (optimal) average NLL.

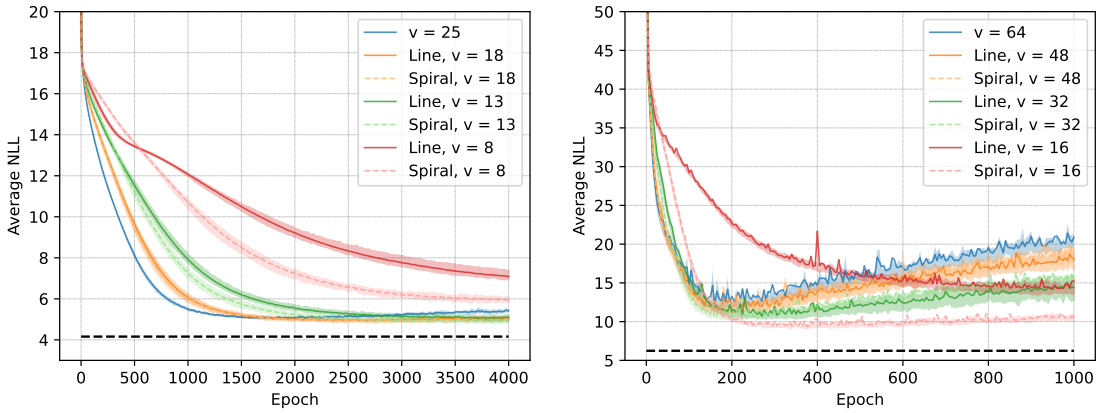
A.1.2 Contrastive Divergence Approximation



(a) CD-1 (b) CD-10 (c) CD-100

Figure A.2: (Regarding Figure 4.5) Evolution of the average NLL over training epochs for different numbers of connections and network structures. Each subplot corresponds to a different CD approximation: the left plot was trained with CD-1, in which only 1 sampling step is applied, the center plot uses 10 sampling steps and the right plot 100 steps. RBMs with the same number of connections are shown with different shades of the same color. The network with $v = 16$ is the same for both patterns and corresponds to the fully connected traditional RBM. The black dashed line corresponds to the data (optimal) average NLL.

A.1.3 Larger Models



(a) BAS 5 (b) BAS 8

Figure A.3: (Regarding Figure 4.6) Average NLL through training epochs for different numbers of connections and BAS models: BAS 5 with $X = 25$ (a) and BAS 8 with $X = 64$ (b). For each plot, RBMs with the same number of connections are shown with different shades of the same color. The fully connected RBM is the same for both patterns (blue curves). The black dashed line corresponds to the data (optimal) average NLL.

A.2 Analysis on MNIST

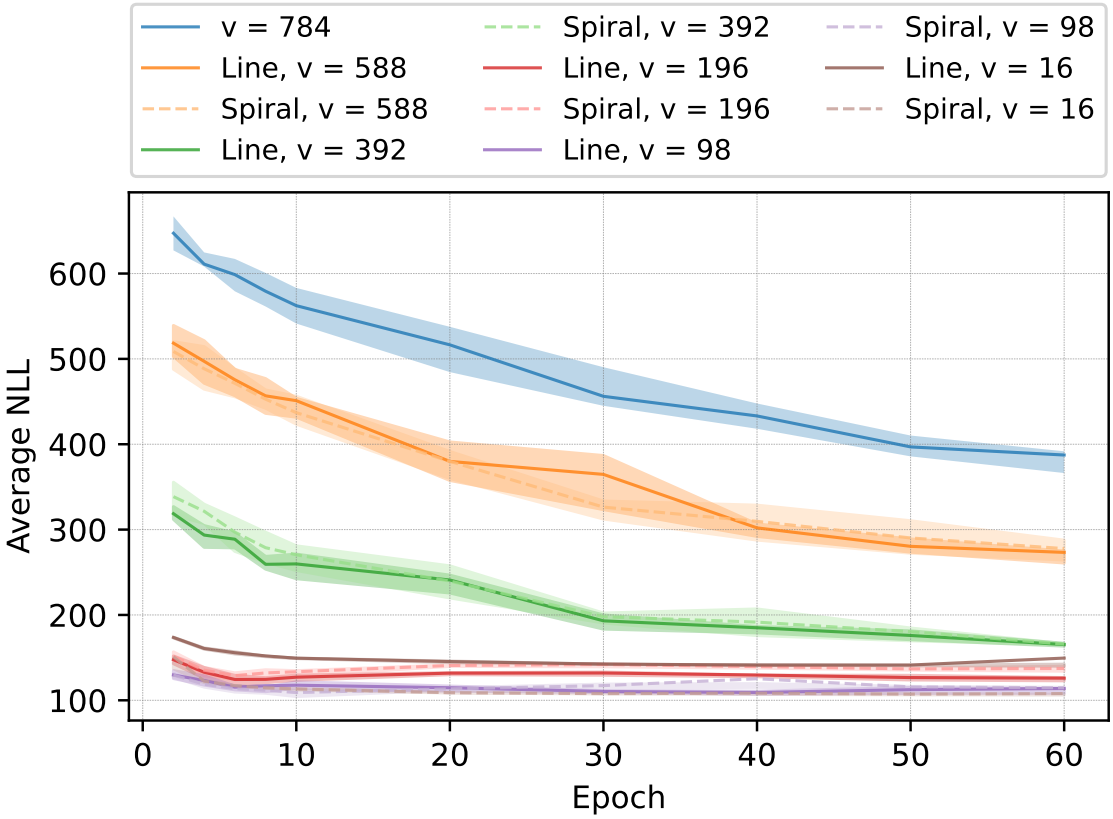


Figure A.4: (Regarding Figure 4.7) Average NLL through training epochs for different numbers of connections and connectivity patterns. RBMs with the same number of connections are shown with different shades of the same color. $v = 784$ corresponds to the fully connected RBM.