Geometric Learning of Latent Parameters with Helmholtz Machines

Dissertation monograph approved by the
Doctoral Degree Committee of
Hamburg University of Technology

in pursuit of the academic degree of

Doktor der Naturwissenschaften (Dr. rer. nat.)

written by

Csongor Huba Várady

from
Baia Mare, Romania

2025

Reviewers:

Prof. Dr. Nihat Ay
Dr. Jens-Peter M. Zemke

Chair of the Board of Examiners:

Prof. Dr. Matthias Schulte

Date of examination:

28.11.2024, Hamburg

ORCID Id:

0000-0003-0823-0786

Abstract

In this thesis, we use concepts from Information Geometry (IG), such as Natural Gradient Descent (NG), to improve the training of a Helmholtz Machine (HM) through the design and implementation of a novel algorithm called the Natural Reweighted Wake-Sleep (NRWS).

First, we prove that for any Directed Acyclic Graph (DAG) the associated Fisher Information Matrix (FIM), which describes the geometry of the statistical manifold, has a fine-grained block-diagonal structure that is efficient to invert. By exploiting the fact that the HM is composed of two DAG networks, we adapt its training algorithm into the NRWS implementing NG.

The NRWS not only achieves better performance in the minimum of the optimization loss compared to other training methods, such as the Reweighted Wake-Sleep (RWS) and Bidirectional Helmholtz Machine but also outperforms them in both epochs and wall-clock time. In particular, we present how the NRWS achieves state-of-the-art performance on standard benchmark datasets (MNIST, FashionMNIST, and Toronto Face Dataset) based on the importance sampling estimation of the log-likelihood of the HM.

By adapting Accelerated Gradients (AG) methods to operate within the geometry defined by the FIM of the HM, we further improve the performance of the NRWS. Using first-order AG methods, such as Momentum and Nesterov Momentum, improves the convergence rate of the NRWS without any computational overhead. Additionally, we develop a regularizer method based on the Maximum Entropy Principle, named the Entropy Regularizer (ER), which we show further improves the NRWS by reaching lower optimization loss and narrowing the generalization gap of the algorithm without extra time penalty, which can also be applied to non-geometric training methods. Conveniently, the NRWS framework is compatible with continuous random variables; hence, we show how the FIM can be derived for normally distributed hidden variables.

Finally, we explore the possibilities of using HMs with Convolutional Neural Networks (CNNs) by computing the FIM for such network topologies and showing that the resulting matrix also has a finely-grained block-diagonal structure. We finish by presenting a hypothesis on the difficulties of using CNNs with HMs and NRWS. We make significant contributions to the field of IG and HM, with numerous findings that could be further explored or reused in other research fields. Our results can represent a starting point for future research on improving training algorithms for neural networks and deep learning models using geometric methods, such as the NG.

Contents

Acknowledgements

"The science of today is the technology of tomorrow." – Edward Teller

Completing this thesis has been a transformative journey, and I am deeply grateful to the many individuals and institutions whose support and guidance have made it possible.

To my fiancée, Bea, thank you for your patience, encouragement, and belief in me throughout this process, and that you stood by my side in this very long and difficult process. To my family, my sister Eme, my mom, and my late dad, who I know would have been proud, your love and support have been my anchor.

I am immensely grateful to my supervisors, Nihat and Luigi, for their invaluable mentorship, to my advisor, Riccardo, for your insightful guidance and thoughtful feedback, and to all three of them for fostering a stimulating and inspiring environment for exploration and growth. A special thanks to my reviewers in the defense process, Prof. Dr. Matthias Schulte and Dr. Jens-Peter M. Zemke, for your detailed evaluation and valuable suggestions.

To my colleagues, Carlotta and Jesse, and everyone from the Data Science Foundations Institute, thank you for your collaboration and support. I am also deeply grateful to everyone from the DeepRiemann research group from the Romanian Institute for Science and Technology for starting me on the journey that resulted in this thesis.

I draw inspiration from the groundbreaking work of Sun'ichi Amari and Geoffrey Hinton, whose contributions to this field have laid the foundation for much of what I have explored.

To my friends from home and around the world, your encouragement has been a source of strength, especially during challenging times. I want to send the message to them, that I finally know the answer to the question: "When will your PhD finally end?". I am also grateful to my psychologist for providing support and perspective throughout this journey.

Finally, this thesis would not have been possible without the institutional backing of the European Union, the Technical University of Hamburg, and the Max Planck Institute for Mathematics in the Sciences, whose resources and funding were vital to this work.

To everyone who has supported me in some way, thank you for being part of this journey. Your contributions, big and small, have made all the difference, as this was the hardest thing I have ever done in my life.

Chapter 1

Introduction

The Natural Gradient (NG), introduced by Shun-ichi Amari, is a non-Euclidean extension of the traditional gradient for optimization over statistical models. The NG follows the steepest descent during training, according to the Fisher-Rao metric defined over the statistical model, and it has been proven both empirically and theoretically to lead to a speed-up of the convergence and to better optimization results.

In the last two decades, there has been a lot of research in the application of the NG to Machine Learning tasks, however, there are only a few instances where such a method reached mainstream use in real-world applications. The reason for this is two-fold, the efficiency in the computation of the NG depends both on the model and on its parameterization, which determines the Fisher Information Matrix (FIM) representing the metric tensor. However, another difficulty is in inverting the FIM, needed for the computation of the NG, an operation that scales more than quadratically with the size of the parameters of the statistical model.

In this thesis, we study the geometry of the statistical model associated with a Helmholtz Machine (HM) and we show that HMs have the property that their associated FIM have a fine-grained block-diagonal structure. Our purpose is to leverage the geometric structure of HMs to facilitate the efficient computation of the FIM and its inverse. This approach enables us to design a novel family of algorithms based on the NG method. Our proposed algorithms outperform modern state-of-the-art methods for the training of HMs in terms of training time and in the quality of the reached optimum after convergence. Additionally, we investigate how regularization, gradient acceleration, and convolutional networks can take advantage of the geometrical properties in the HM, such as the structure of the FIM and the locality of its gradients.

1.1 Context and Motivation

Deep generative models have been successfully employed in unsupervised learning to model complex and high dimensional distributions thanks to their ability to extract higher-order representations of the data and thus generalize

better (Hinton et al., 2006; Bengio, 2009). An approach that proved to be successful and thus is common to several models is based on the use of two separate networks: the recognition network, i.e., the encoder, that provides a compressed latent representation for the input data; and the generative network, i.e., the decoder, able to reconstruct the observation in output. Autoencoders (AEs) (LeCun, 1987; Bourlard and Kamp, 1988; Kramer, 1991; Hinton and Zemel, 1993) are a classical example of this paradigm, where both the encoder and the decoder are commonly implemented as deterministic feed-forward networks.

Variational Autoencoders (VAEs) (Kingma and Welling, 2014; Rezende et al., 2014) introduce an approximate posterior distribution over the latent variables which are then sampled, thus resulting in stochastic models. More modern variants of the VAE such as the VQ-VAE (Van Den Oord et al., 2017; Razavi et al., 2019) are capable of learning and generating fully natural looking images, with highly realistic details, and in some cases, they can even generate realistic fantasy images based on text prompts, as in the case of DALL-E (Ramesh et al., 2021).

Yet another generative model is in the form of Generative Adversarial Networks (GANs)(Goodfellow et al., 2014), where the two networks are set in an adversarial setting, where the one learns to generate samples that are identical to the ones from the dataset, and the other attempts to discriminate the generated samples from the original ones. Alternative and improved versions of it, such as the WassersteinGAN (Arjovsky et al., 2017) reach similarly good, state-of-the-art performance in image generation tasks, as the aforementioned VQ-VAE.

Alternatively, Helmholtz Machines (HMs) (Dayan et al., 1995) are AEs which consist of a recognition and a generative network both modeled as Sigmoid Belief Networks (SBNs) (Neal, 1992), characterized by discrete hidden variables, differently from standard VAEs which commonly adopt continuous Gaussian variables only in the bottleneck layer. HMs did not garner popularity in recent years such as VAEs and GANs, although their network structure has some advantages over the other methods. Namely, from the locality of their gradients, thanks to the SBNs, it follows that their training is not based on back-propagation (Rumelhart et al., 1986) through the layers of the network, but by local layer-wise interactions. In this thesis, we will exploit specifically this property of HMs in order to design a novel algorithm with the use of IG concepts.

The training of stochastic models is a challenging task in deep learning (Glorot and Bengio, 2010a). This extends to generative models based on stochastic models, which are commonly trained by the maximization of the likelihood, or equivalently, by the minimization of a divergence function

between the unknown distribution of the data and the one of the generative model. The challenges for the optimization task are due to the presence of terms that are computationally expensive to estimate, such as the partition function.

A possible solution to this problem resides in the introduction of a family of tractable approximate posterior distributions (Wainwright et al., 2008), parameterized by the encoder network. In the presence of continuous hidden variables, for which the stochastic back-propagation of the gradient is possible, as in VAEs, the encoder and decoder networks can be trained simultaneously, through the definition of a unique loss function that corresponds to a lower-bound for the likelihood, i.e., the ELBO (Kingma and Welling, 2014; Rezende et al., 2014).

In the presence of discrete hidden variables, as for HMs, this approach cannot be directly employed, since it is not possible to back-propagate through stochastic layers. Thus the standard training procedure of HM is the well-known Wake-Sleep (WS) (Hinton et al., 1995) algorithm, in which two optimization steps for the parameters of the recognition and generative networks are alternated. The WS algorithm, as well as more recent advances (Bornschein and Bengio, 2015; Bornschein et al., 2016; Wenliang et al., 2020; Hewitt et al., 2020), relies on the conditional independence between the hidden variables of each layer, which allows a factorization of the loss function as in directed graphical models (Lauritzen, 1996). This leads to a computationally efficient formula for the weights update which does not require the gradients to be back-propagated through the full network. An alternative to WS for HMs is given by the REINFORCE algorithm (Williams, 1992), which is popular in the Reinforcement Learning literature. However, differently from WS, with REINFORCE the variance of the gradient grows linearly with the number of the parameters of the network, an issue addressed in several modern variants, for instance, in the works of Mnih and Gregor (2014), Tucker et al. (2017), Grathwohl et al. (2018) and Kool et al. (2020).

Besides the choice of the specific loss function to be optimized, depending on the nature of the generative model, in the literature, several approaches to speed up the convergence during training have been proposed, through the definition of different optimization algorithms. One line of research, initiated by Amari and co-workers (Amari, 1998, 1997), takes advantage of a geometric framework based on notions of Information Geometry (IG) (Amari and Nagaoka, 2000), leading to the definition and use of the Natural Gradient (NG). Whenever the loss function is defined over a statistical manifold of distributions, whose geometry is given by the Fisher-Rao metric, the NG of the function to be optimized corresponds to the Riemannian gradient of the function itself, computed with respect to the metric of the manifold.

In general, the computation of the NG requires the inversion of the Fisher Information Matrix (FIM), and for this reason, often it cannot be directly applied for the training of large neural network due to its computation cost.

Several approaches have been proposed in the literature to counteract the computational cost (Desjardins et al., 2013, 2015; Grosse and Martens, 2016; Ollivier, 2015; Martens and Grosse, 2015), which are all based on more or less sophisticated approximations of the structure of the FIM. By introducing different independence assumptions between random variables from the network, certain blocks of the FIM are set to zero a priori or, alternatively, they admit specific representations (such as low-rank updates of a diagonal matrix or Kronecker products of matrices) which allow its efficient inversion. A different view is provided by Sun and Nielsen (Sun and Nielsen, 2017), who propose to compute a local version of the Fisher-Rao metric, that they call Relative Fisher Information Matrix, used to analyze the local learning dynamics in a large system. Yet a different approach is introduced by Lin et al. (2021), where they describe a method for the computation of the natural gradient based on the use of local-parameter coordinates, which can be applied to several distributions and algorithms.

NG has been previously applied for the training of generative models. In particular, in the work of Lin et al. (2017) the NG is applied to the large class of Variational Autoencoders with non-conjugate latent graphical models by exploiting the geometry of the variational distribution in the computation of the updates in the mean-parameter space. Along a similar line of research, Zhang et al. (2018a) proposes the use of the NG in connection to variational inference for the training of Bayesian neural networks. Differently from this work, in both cases, the NG exploits a metric defined over the space of the parameters that identify the approximate posterior distribution, instead of the metric associated with the space of the weights associated with the whole neural network.

In this thesis, we follow a different approach for the training of HM based on the computation of the NG without the need to approximate the FIM before its empirical evaluation. Motivated by preliminary results from (Ay, 2002) for the computation of the FIM for an SBN, we observe that the matrix associated with the joint statistical model represented by a HM takes a block-diagonal structure, where the block sizes depend linearly on the size of each hidden layer. This result, which can be seen as a direct consequence of the topology of the directed graphical model associated with the SBN, does not require the introduction of any additional independence assumption between random variables in the FIM, with the purpose of further simplifying the structure of the FIM, before its empirical evaluation. Notice that the level of sparsity for the FIM in SBNs is superior to that associated with the

standard assumption of independence between layers, where the width of the individual blocks of the FIM is given by the product of the sizes of adjacent hidden layers. Indeed, for SBNs we have a finely-grained block-diagonal structure for the FIM, with block widths given by the sizes of the hidden layers. A consequence of these finer-grained blocks is that the number of zero off-diagonal blocks in the FIM grows by a quadratic factor, which allows for a more computationally efficient inversion of the matrix.

Based on these observations we propose an efficient geometric adaptation of the Reweighted Wake-Sleep (RWS) (Bornschein and Bengio, 2015) for the training of HMs, where the gradient is replaced by the corresponding NG. The increased sparsity of the FIM, which is a direct consequence of the topology of the network, limits the impact on the cost per iteration of the use of the NG. This has the advantage of allowing for an exact computation of the NG for a given batch, not requiring further assumptions in terms of the structure of the FIM. This leads to an advantage in the speed-up in convergence during training in terms of epochs, while limiting the computational overhead.

The main contributions of this thesis in order to design a computationally efficient algorithm based on NG for the training of HMs are the following:

1. The design of a novel algorithm called Natural Reweighted Wake-Sleep (NRWS) which exploits a finely-grained block-diagonal structure for the FIM. Such structure for the FIM:

   (a) has never been exploited before in the training of HMs, not even for deterministic networks, indeed previous observations about the structure of the FIM have not been used in training (Ay, 2002, 2020);

   (b) differently from other models, it is exact, i.e., for HM the sparsity structure is not an approximation/assumption but it derives from conditional independence among variables set by the network topology;

   (c) is made of smaller-sized blocks (thus it is more efficient to be computed) than the standard block-diagonal structure used in previous works cf. Sun and Nielsen (2017) and Zhang et al. (2018a).

2. Our results on different datasets show that our algorithm is able not only to converge to a better value for the loss both in training and test compared to RWS and BiHM (which are state-of-the-art in the literature for HM), but also to achieve faster convergence, not only in terms of epochs but also in wall-clock time. This is a strong result, since NG often suffers from large computation complexity which prevents

its use in practice. We present an additional improvement for all WS based algorithms, which can be used without any noticeable drawback on performance.

3. We present how accelerated gradient methods such as Momentum (Polyak, 1964), Nesterov Momentum (Nesterov, 1983) and Adam (Kingma and Ba, 2015) can be used with NRWS to speed up the convergence rate, under some simple assumptions. We show that HMs can benefit from both accelerated and natural gradients, at the same time, reaching state-of-the-art performance on our datasets.

4. We design a regularization method for training algorithms, called the Entropy Regularizer, for HMs that is based on the Maximum Entropy Principle (Jaynes, 1957). Such a regularization method:

   (a) improves on the performance of all WS based algorithms in reaching lower optimization minima on all the evaluated datasets;

   (b) has a faster convergence rate in our experiments;

   (c) improves the performance of the optimization on datasets where traditional regularization methods based on the Euclidean norm of the weights cannot.

5. We show how one can use continuous distributions, i.e., Gaussians, together with HMs in order to represent continuous data, and we show that the structure of the FIM remains finely-grained block-diagonal when using such distributions with the NRWS.

6. We prove that convolutional networks also have finely-grained block-diagonal structures in the FIM and we derive the exact formula for the FIM of convolutional and deconvolutional layers. We present hypotheses about why convolutional layers present limitations when used directly with HMs and give some intuition about the motivation for such behavior.

1.2 Structure of the Thesis

The thesis is organized as follows: The first two chapters introduce topics that are important for the understanding of the thesis. Chapter 2 describes basic concepts about neural networks, an introduction to the HM, and the WS algorithm, as well as improved versions of those, i.e., the RWS and the BiHM. Chapter 3 describes the key concepts of IG that are needed to understand

the work that we did in this thesis. Here we give formal definitions of statistical manifolds, tangent spaces, metrics, divergences, curves, and parallel transports, among others. Most importantly here we define the role of the NG and its relationship to the FIM. In Chapter 4 we derive the FIM which is associated with the statistical model defined by the HM, and we study the intrinsic structure of the matrix. In Chapter 5 we describe an algorithm that we call NRWS, which builds on the previous state-of-the-art method RWSBornschein and Bengio (2015) to train HMs by using the NG. We present how the novel algorithm and a couple of its variants outperform the other methods on common benchmark datasets. In the following two chapters, we present our work on developing the NRWS to be a more versatile algorithm. In Chapter 6 we look at gradient acceleration methods and how they can be employed effectively with NRWS. Furthermore, we study the behavior of well-known regularization methods in conjunction with the NRWS and develop a novel and geometrically sound regularization method. Finally, we present the combined effect of regularization and gradient acceleration on the NRWS to reach improved state-of-the-art performance, compared to the previous chapter. In Chapter 7, we adapt NRWS to work with continuous random variables, by deriving the FIM for HMs with normal distributions. In the second part of the chapter, we prove that an HM using convolutional networks has an associated FIM with a finely-grained block-diagonal structure, similarly to dense layers, and we present the raised implications by it from a practical perspective. Finally, in Chapter 8 we present the conclusions of the thesis and potential future avenues of research.

At the end of the thesis, one can find sections that describe the acronyms and symbols, as well as lists of the figures and tables, used in the thesis. Finally, we finish with supplementary material in the form of appendices: a guide to reproducing the experiments, information about hyperparameter tuning and data augmentation, and an intuitive description of the working mechanism behind convolutional layers.

1.3 Notation

Throughout this thesis, many concepts from probability theory and statistics are used. In different research communities, different notations are used when talking about the same concepts, therefore in this section, we describe briefly the notation choices we made.

We will denote all probability measures and/or distributions with small letters, typically p or q. Furthermore, since most of the time we are dealing with specifically the statistical model associated with the HM, we introduce

some notation that we use for states and random variables. We rarely use a state that is different from 1 or 0 but some specific variable, so instead of using capital letters for random variables, we use small letters, like x for visible variables or h for hidden ones. These same symbols are used to represent also the states of the variables, which usually correspond to the outputs of individual nodes of the neural network, thus h^i corresponds to the state of the i-th layer.

We use small Greek letters such as θ and ϕ, as parameters of the probability distributions. For specific distributions, i.e., normal distribution, we use their canonical parameters μ as the mean and σ as the standard deviation.

Instead of the more commonly used $p(\cdot;\theta)$ notation for underlining that θ is the parameter that controls p, we use the simpler $p_\theta(\cdot)$. Sometimes, we further simplify by writing just $p(\cdot)$, without θ, when the parameterization is obvious or unnecessary in the calculations. In the context of HMs, we always parameterize p by θ and q by ϕ for the generative and recognition networks, respectively, so when a parameter is not explicitly specified for brevity, one should assume such parameterization.

Calculating some quantities in the thesis requires using two separate notations to differentiate between exact and empirically estimated values. When referring to the exact values of some quantities, like the loss function $\mathcal{L}$ and the FIM $\mathcal{F}$, we represent them as uppercase calligraphic letters. When these same quantities are estimated from samples, we represent them as simple uppercase letters L and F with the relationship $\mathcal{L} \simeq L$ and $\mathcal{F} \simeq F$, where $\simeq$ represents similarity by estimation.

1.4 Reproducibility of Experiments

All experiments were programmed in Python and optimized with the help of Tensorflow 1.15 (Abadi et al., 2015) to take advantage of the high levels of parallel computations by running the experiments on Graphical Processing Units (GPU). Experimental details such as hyperparameters, hardware and datasets are explained in the Appendix A. The implementation of the experiments is publicly available at https://github.com/szokejokepu/natural-rws.

Chapter 2

Helmholtz Machines

The Helmholtz Machine (HM), introduced by Dayan et al. (1995), is a generative model based on optimizing the Helmholtz Free Energy (HFE) (Helmholtz, 1882) of the statistical model. In this section we present how the HM is constructed, what is the basis for this structure and how to train it through the Wake-Sleep (WS) algorithm (Hinton et al., 1995).

Later, we describe the Reweighted Wake-Sleep (RWS) (Bornschein and Bengio, 2015) which improves on the performance of the WS with minimal to no drawbacks. We also present the Bidirectional Helmholtz Machine (BiHM) (Bornschein et al., 2016), which changes the learning mechanic of the HM from the two alternating objectives of WS to a single objective. The RWS and BiHM are the state-of-the-art methods on HM which we use as baseline for our experiments later in the thesis.

We conclude the chapter by introducing some common generative models that are used in Machine Learning research in order to show the current state of generative models. We point to key differences, both the advantages and disadvantages, between these and the HM, in order to put into context the current research.

2.1 Helmholtz Machines

Deep generative models have been successfully employed in unsupervised learning to model complex and high dimensional distributions thanks to their ability to extract higher-order representations of the data and thus generalize better (Hinton et al., 2006; Bengio, 2009). Generative models have as main goal approximating some probability distribution of data $p_{\mathcal{D}}(x)$ of which only a subset is known in the form of a set of samples, called a dataset ($\mathcal{D}$). Once the distribution is learned, the model has to be able to **generate** samples from it, which have a high likelihood w.r.t the true data distribution $p_{\mathcal{D}}(x)$.

The HM is a deep generative model constructed from two networks, called the generative and recognition networks. The generative network is the one which is learning the true probability distribution $p_{\mathcal{D}}$, while the recognition

network learns to infer the posterior distribution of the former. [1] This two-network construction is common in deep generative models, which we present in Section 2.3.

The networks of the HM, as classically defined in the work of Dayan et al. (1995), are Sigmoid Belief Networks. However, this is not a constraint, as one could use any probability distribution as we show in Section 7.1.

2.1.1 Sigmoid Belief Networks

Sigmoid Belief Networks (SBNs) (Neal, 1990) are a class of models corresponding to a sequence of stochastic layers, which typically consist of vectors of conditionally independent binary random variables. The activations on each layer of an SBN are sigmoid functions, which generate as output the mean parameters of Bernoulli distributions, one for each hidden random variable. The sigmoid function $s : \mathbb{R} \to (0, 1)$ is defined as

$$s(x) = \frac{1}{1 + e^{-x}} \ .$$

Let $x = (x_1, \ldots, x_{l_0})$ be the vector of visible variables, $h = \{h^1, \ldots, h^M\}$ the set of hidden variables, where each $h^i = (h^{i,1}, \ldots, h^{i,l_i})$ is a vector of hidden variables, corresponding to layer i, M the total number of layers and l_i is the length of the vector of variables of layer i. An SBN can be associated to a joint probability distribution $p_\theta(x, h)$, parameterized by $\theta = (\theta^0, \ldots, \theta^M)$ which factorizes as a directed graphical model (Lauritzen, 1996)

$$p_\theta(x, h) = p_{\theta^0}(x|h^1)p_{\theta^1}(h^1|h^2) \cdots p_{\theta^{M-1}}(h^{M-1}|h^M)p_{\theta^M}(h^M) \ , \tag{2.1}$$

where each vector of random variables in h^i at layer i only depends on the variables h^{i+1} of the previous layer and $(h^{i,1}, \ldots, h^{i,l_i})$ are conditionally independent of each other.

Note 1
Often, to simplify sums and products we implicitly treat x as h^0, therefore we can abbreviate

$$p(x|h^1)p(h^1|h^2) \ldots p(h^M) \ \ to \ \ \left[\prod_{i=0}^{M-1} p(h^i|h^{i+1})\right] p(h^M) \ .$$

[1] The pair of networks is also sometimes referred to as the decoder (generative) and encoder (recognition) networks in the literature.

Typically in SBNs a node j from layer i will be a linear function with the parameters $\theta^{i,j} = \{W^{i,j}, b^{i,j}\}$ followed by a sigmoid activation function

$$\rho^{i,j} = s\left(\left(W^{i,j}\right)^\top \cdot h^{i+1} + b^{i,j}\right) . \tag{2.2}$$

To simplify the notation we consider $b^{i,j}$ as the zeroth element of the vector $W^{i,j}$ ($W^{i,0} = b^i$), and expand h^{i+1} with a 1, to have a more compact inner-product between the weights and the inputs $\left(W^{i,j}\right)^\top \cdot [1 : h^{i+1}]$.

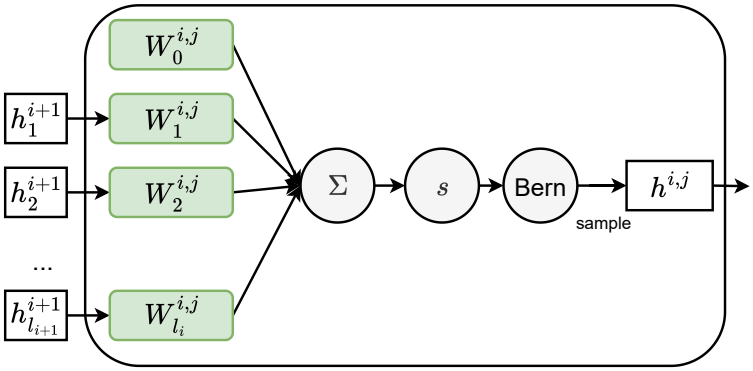


Figure 2.1: Node of a sigmoid network

Finally, the output of the sigmoid is used as the mean parameter of a Bernoulli distribution, from which the state of the next node is sampled

$$\begin{aligned} &Bern\left(h^{i,j}; \rho = s\left(\left(W^{i,j}\right)^\top \cdot [1 : h^{i+1}]\right)\right) \text{ with p.d.f} \\ &p(h^{i,j}|h^{i+1,j}) = \rho^{h^{i,j}}(1-\rho)^{1-h^{i,j}} . \end{aligned} \tag{2.3}$$

The structure of such a node is represented in Figure 2.1.

We can calculate the activations for a whole layer i by treating the vector $W^{i,j}$ as row j of the matrix W^i, and sampling the next layer from l_i Bernoulli distributions with the mean parameters $s\left(W^i \cdot [1 : h^{i+1}]\right)$.

The Helmholtz Machine (HM) (Dayan et al., 1995) is a generative model that consists of a sequence of stochastic layers, one on top of the other, where the layer at the bottom is the visible layer x, while the others are the hidden ones h^i. In an HM, the consecutive layers are connected in opposite directions with two different SBNs, called the recognition and generative networks. The recognition network starts from the input x towards the last hidden layer h^M, and is meant to infer the latent variables of the HM, while the generative network begins with h^M and propagates down to the input x to generate new

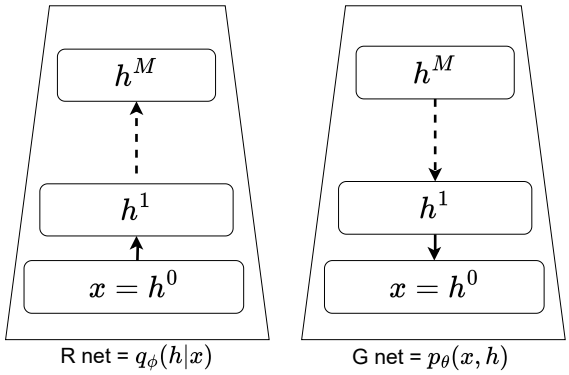


Figure 2.2: Structure of the Helmholtz Machine: (left) recognition network, (right) generative network

data from the latent space. This enables us to define a generative distribution p parameterized by θ, associated with the generative network, as well as a recognition (conditional) distribution q parameterized by ϕ, representing the recognition network. The distributions p and q factorize as follows:

$$p_\theta(x, h) = p(x|h^1)\, p(h^1|h^2) \cdots p(h^{M-1}|h^M)\, p(h^M) \quad \text{and} \tag{2.4}$$

$$q_\phi(h|x) = q(h^M|h^{M-1}) \cdots q(h^2|h^1)\, q(h^1|x)\ . \tag{2.5}$$

Usually, the higher the layer i is in the sequence, it becomes narrower in width l_i, with the top layer h^M being the "bottleneck" layer. The structure of an HM is illustrated in Figure 2.2.

2.1.2 The Objective of Helmholtz Machines

The objective of the training of the HM is to find the parameters θ^* that minimizes the Kullback-Leibler (KL) divergence [2] between the data distribution $p_\mathcal{D}$ and the generative distribution p of the HM:

$$\theta^* = \arg\min \ \ \mathrm{D}_{KL}\left[p_\mathcal{D}(x)||p_\theta(x)\right] \tag{2.6}$$

[2]For more details about the KL divergence see Section 3.4.

where $p_\theta(x)$ corresponds to the Markov Chain shown in Eq. (2.4) marginalized over the hidden variables h. By expanding the KL divergence

$$\mathrm{D}_{KL}\left[p_{\mathcal{D}}(x)||p_\theta(x)\right] = \int p_{\mathcal{D}}(x) \ln \frac{p_{\mathcal{D}}(x)}{p_\theta(x)} dx \tag{2.7}$$

$$= \underbrace{\int p_{\mathcal{D}}(x) \ln p_{\mathcal{D}}(x) dx}_{\text{independent of } \theta} - \int p_{\mathcal{D}}(x) \ln p_\theta(x) dx \tag{2.8}$$

we notice that the first term of the KL is independent of the parameterization, the only term one can minimize is the expectation of the negative log-likelihood of the model

$$\arg\min_\theta \ \mathrm{D}_{KL}\left[p_{\mathcal{D}}(x)||p_\theta(x)\right] = \arg\min_\theta \ \mathbb{E}_{x\sim\mathcal{D}}[-\ln p_\theta(x)] \ . \tag{2.9}$$

This quantity is also called the Helmholtz Free Energy, on which we expand in the next subsection.

2.1.3 Helmholtz Free Energy

In this subsection we describe the Helmholtz Free Energy (HFE) from a historical perspective, and outline why and how it is used to optimize probability measures. The HFE comes from the world of physics, introduced by Hermann von Helmholtz (Helmholtz, 1882), and it is commonly used to describe the behavior of gasses. The HFE is defined as the difference between the internal energy of the system and the entropy of the system multiplied by the temperature,

$$\mathbf{F} \equiv \mathbf{U} - \mathbf{TH} \ , \tag{2.10}$$

where $\mathbf{F}$ is the HFE (SI: joules), $\mathbf{U}$ is the internal energy of the system (SI: joules), $\mathbf{T}$ is the absolute temperature (kelvins) of the surroundings, modelled as a heat bath, $\mathbf{H}$ is the entropy of the system (SI: joules per kelvin). In this definition $\mathbf{U}$ is the expected value of the energy, given by

$$\mathbf{U} = \mathbb{E}_p[\mathbf{E}] = \sum_i p(\mathbf{E}_i)\mathbf{E}_i \ ,$$

where the energy $\mathbf{E}_i$ of each state i is described by a probability distribution p, which is a Boltzmann distribution with temperature $\mathbf{T}$

$$p(\mathbf{E}_i) = \frac{e^{-\frac{\mathbf{E}_i}{\mathbf{T}}}}{\sum_j e^{-\frac{\mathbf{E}_j}{\mathbf{T}}}} \ .\mathbf{E}_i$$

Following the above definition, the HFE can be understood as the remaining energy after the internal potential energy transformed into the entropy. The system is optimal and most efficient when all the energy is transformed by the system, from which follows that the HFE is minimal.

Having in mind the physical definition of HFE, the generative distribution of a Helmholtz Machine can be seen in a new light. Given the generative distribution, we can think of the explanations of the internal variables h as

$$p(h|x) = \frac{p(x,h)}{p(x)} = \frac{p(x,h)}{\sum_{h'} p(x,h')} \ .$$

Using the definition $\mathbf{E}_G[x,h] := -\ln p(x,h)$ the above equation becomes

$$p(h = H_i|x) = \frac{e^{-\mathbf{E}_G[x,h]}}{\sum_{h'} e^{-\mathbf{E}_G[x,h']}} \ .$$

We can clearly see that after the rewrite, our explanations $p(h|x)$ can be understood as the internal energy of a HFE model with temperature $\mathbf{T} = 1$.

We can approach our object of minimization as the HFE of the generative network:

$$\mathbf{F}_G(x) = -\ln p(x)$$

which we can rewrite as

$$\begin{aligned}
\mathbf{F}_G(x) &= -\ln p(x) \\
&= -\ln p(x) \left[\sum_h p(h|x)\right] \\
&= -\sum_h p(h|x) \ln p(x) \\
&= -\sum_h p(h|x) \ln \left[\frac{p(x,h)}{p(h|x)}\right] \\
&= \sum_h p(h|x) \ln p(h|x) - \sum_h p(h|x) \ln p(x,h) \\
&= -\mathbf{H}_G[h|x] + \sum_h p(h|x) \mathbf{E}_G\left[x,h\right] \\
&= \mathbb{E}_{p(h|x)}\left[\mathbf{E}_G\left[x,h\right]\right] - \mathbf{H}_G\left[h|x\right] \ .
\end{aligned}$$

From the above equation $\mathbf{U} = \mathbb{E}_{p(h|x)}\Big[\mathbf{E}_G\left[x,h\right]\Big]$ is the expected energy of the system and $\mathbf{H}_G[h|x]$ is the entropy of the posterior distribution $p(h|x)$. We can also see that $\mathbf{F}_G = -\ln p(x)$ can in fact be written in the form of the

HFE from Eq. (2.10). The name Helmholtz Machine is derived from the fact that its goal is minimizing the HFE of the model $\mathbf{F}_G$.

At this point we could start applying gradient descent to minimize this quantity, but we would quickly run into a problem, since the posterior $p(h|x)$ does not factorize nicely and calculating it is hard to infer in practice. This issue is solved by the widely used training algorithm REINFORCE, which we present shortly in Subsection 2.3.3, however the HM uses the different approach of introducing a second distribution to learn the posterior.

2.1.4 Variational Approach

A common approach to solve the problem of learning the posterior is to use the Variational or Reparameterization trick (Wainwright et al., 2008). Using this trick means that instead of trying to estimate the posterior distribution $p(h|x)$, we introduce a different distribution $q(h|x)$ to approximate it. This is used in practice also for the training of the very common Variational Autoencoder (VAE)s (Kingma and Welling, 2014; Rezende et al., 2014), which we present in Subsection 2.4.1.

We define a recognition network as presented in Eq. (2.5) and shown in Figure 2.2. Sometimes we need the joint probabilities of q, then we consider the real distribution $p_{\mathcal{D}}(x)$ as the visible distribution, i.e.,

$$q(x,h) = q_\phi(h|x)p_{\mathcal{D}}(x) \ .$$

We learn the approximate posterior in such a way that it behaves similarly to the true posterior. We can minimize the KL divergence between the two:

$$\begin{aligned}
\mathrm{D}_{KL}\left[q(h|x)||p(h|x)\right] &= \sum_h q(h|x)\ln\frac{q(h|x)}{p(h|x)} \\
&= \sum_h q(h|x)\ln q(h|x) - \sum_h q(h|x)\ln p(h|x) \\
&= \sum_h q(h|x)\ln q(h|x) - \sum_h q(h|x)\ln\frac{p(x,h)}{p(x)} \\
&= \sum_h q(h|x)\ln q(h|x) - \sum_h q(h|x)\ln p(x,h) + \sum_h q(h|x)\ln p(x) \\
&= -\mathbf{H}_R[h|x] + \sum_h q(h|x)\mathbf{E}_G\left[x,h\right] + \ln p(x)\underbrace{\sum_h q(h|x)}_{=1} \\
&= -\mathbf{H}_R[h|x] + \mathbb{E}_{q(h|x)}\left[\mathbf{E}_G\left[x,h\right]\right] - \mathbf{F}_G(x)
\end{aligned}$$

Reordering the terms we get

$$\mathbf{F}_G(x) + \mathrm{D}_{KL}\left[q(h|x)||p(h|x)\right] = \mathbb{E}_{q(h|x)}[\mathbf{E}_G(h,x)] - \mathbf{H}_R(h|x) \ . \tag{2.11}$$

We can define a new objective as the **variational free energy** $\mathbf{F}_G^R(x)$, which as we can see, still behaves as a HFE

$$\begin{aligned}\mathbf{F}_G^R(x) &:= \mathbf{F}_G(x) + \mathrm{D}_{KL}\left[q(h|x)||p(h|x)\right] \\ &= \mathbb{E}_{q(h|x)}[\mathbf{E}_G(h,x)] - \mathbf{H}_R(h|x)\ .\end{aligned} \tag{2.12}$$

The variational free energy is also sometimes referred to as the Evidence Lower Bound (ELBO), and it is also the objective function optimized in VAEs.

Calculating the gradients w.r.t. parameters θ of the generative network p can now be done

$$\begin{aligned}\nabla_\theta \mathbf{F}_G^R(x) &= \nabla_\theta \mathbb{E}_{q(h|x)}[\mathbf{E}_G(h,x)] - \nabla_\theta \mathbf{H}_R(h|x) \\ &= \nabla_\theta \underbrace{\sum_h q(h|x)\ln q(h|x)}_{\text{const. w.r.t. } G} - \nabla_\theta \sum_h q(h|x)\ln p(x,h) \\ &= -\nabla_\theta \sum_h q(h|x)\ln p(x,h) \\ &= -\sum_h q(h|x)\nabla_\theta \ln p(x,h)\ \text{, expand using Eq. (2.1)} \\ &= -\sum_h q(h|x)\nabla_\theta \ln p_{\theta_1}(x|h^1)p_{\theta_2}(h^1|h^2)\ldots p_{\theta_M}(h^M) \\ &= -\sum_h q(h|x)\sum_{i=0}^{M-1}\nabla_{\theta^i}\ln p_{\theta^i}(h^i|h^{i+1}) \\ &= -\mathbb{E}_{q(h|x)}\left[\sum_{i=0}^{M-1}\nabla_{\theta^i}\ln p_{\theta^i}(h^i|h^{i+1})\right] \\ &= -\mathbb{E}_{q(h|x)}\left[\nabla_\theta \ln p_\theta(x,h)\right]\ .\end{aligned} \tag{2.13}$$

Calculating the gradients w.r.t. θ comes down to layer-wise derivatives, however when we try to optimize with respect of the weights ϕ of the recognition network q the solution does not simplify as nicely.

2.2 The Wake-Sleep Algorithm

In the Wake-Sleep (WS) algorithm Hinton et al. (1995) and Dayan (2000) found a solution to the problem introduced in the previous section. The optimization algorithm consists of two phases which optimize the two networks in an alternating way. The wake phase optimizes the weights of the generative network p as described in Eq. (2.13). By sampling from the distribution $p_{\mathcal{D}}(x)$ and propagating through the recognition network $q(h|x)$ for each layer, the

derivatives can be calculated for the layer i by having the samples h^i and h^{i+1}.

The sleep phase optimizes the weights of the recognition network q, however instead of optimizing the variational free energy $\mathbf{F}_G^R(x)$ it uses an objective with the arguments of the KL divergence switched compared to Eq. (2.12). Thus the objective of the sleep phase is

$$\tilde{\mathbf{F}}_G^R(x) = \mathbf{F}_G(x) + \underbrace{\mathrm{D}_{KL}\left[p(h|x)||q(h|x)\right]}_{\text{args. switched compared to (2.12)}}, \tag{2.14}$$

which we call the **augmented free energy**. By using this minor tweak, the derivation w.r.t. the parameters ϕ of the recognition network q simplifies dramatically, and we get something similar to Eq. (2.13):

$$\begin{aligned}
\nabla_\phi \tilde{\mathbf{F}}_G^R(x) &= \nabla_\phi \underbrace{\mathbf{F}_G(x)}_{\text{const. w.r.t. } \phi} + \nabla_\phi \mathrm{D}_{KL}\left[p(h|x)||q(h|x)\right] \\
&= \nabla_\phi \left[\underbrace{\sum_h p(h|x)\ln p(h|x)}_{\text{const. w.r.t. } \phi} - \sum_h p(h|x)\ln q(h|x) \right] \\
&= -\sum_h p(h|x)\nabla_\phi \ln q(h|x) \\
&= -\mathbb{E}_{p(x,h)}[\nabla_\phi \ln q(h|x)] \,.
\end{aligned} \tag{2.15}$$

Note 2

Note that changing the KL divergence is not symmetric, as discussed in Subsection 3.4, so switching the arguments is not a trivial change and as such it questions the convergence properties of WS. Why does the inverted KL objective from Eq. (2.14) *work?*

Ikeda et al. (1999) studied the convergence properties of the WS with special attention on the switching of the arguments. They show that in fact this change has consequences and introduce a possible solution to circumvent these. In Section 5.4 we present in more detail the convergence properties of the WS through this work and we propose an alternative technique, which we test empirically.

Note 3

In these derivations, we used the language and concepts of the HFE to discuss how the HM works, similarly to the paper of Kirby (2006). However from here on we will simply refer to the variational free energy $\boldsymbol{F}_G^R(x)$ *(Eq.* (2.12)*) as the loss of the wake phase* $\mathcal{L}_p$ *and to the augmented free energy* $\tilde{\boldsymbol{F}}_G^R(x)$

(Eq. (2.14)) as the loss of the sleep phase $\mathcal{L}_q$ as

$$\mathcal{L}_p(x;\theta) = -\ln p_\theta(x) \quad \textit{and} \tag{2.16}$$
$$\mathcal{L}_q(x,h;\phi) = -\ln q_\phi(h|x)\ . \tag{2.17}$$

Thus the derivatives of the two phases are

$$\nabla_\theta \mathcal{L}_p\,(x,h) = -\mathbb{E}_{q(h|x)}\left[\nabla_\theta \ln p(x,h)\right] \quad \textit{and} \tag{2.18}$$
$$\nabla_\phi \mathcal{L}_q\,(x,h) = -\mathbb{E}_{p(h,x)}\left[\nabla_\phi \ln q(h|x)\right]\ . \tag{2.19}$$

This allows us to better compare with other algorithms, which are not necessarily focused around the HFE.

Example: Sigmoid Belief Networks

Using the standard SBN setting for each node of the HM, with sigmoid activation functions and Bernoulli distributions, we can calculate the derivatives of the different objectives. Starting with the wake phase and its objective from Eq. (2.13), we continue to differentiate w.r.t. the parameters θ^i of the network p, which comprises of weights W^i and biases $W^{i,0}$.

$$\begin{aligned}
&\nabla_{\theta^i} \ln p_{\theta^i}(h^i|h^{i+1}) = \\
&\quad = \nabla_{\theta^i} \ln \left(\rho^i\right)^{h^i} (1-\rho^i)^{1-h^i}\ \text{, where } \rho^i = s(W^i \cdot [1 : h^{i+1}]) \\
&\quad = h^i \nabla_{\theta^i} \ln \rho^i + (1-h^i)\nabla_{\theta^i} \ln(1-\rho^i) \\
&\quad = \frac{h^i}{\rho^i}\nabla_{\theta^i}\rho^i + \frac{(1-h^i)}{(1-\rho^i)}\nabla_{\theta^i}(1-\rho^i)\ , \quad \text{knowing } s' = s(1-s) \\
&\quad = \frac{h^i}{\rho^i}\rho^i(1-\rho^i)[1:h^{i+1}] - \frac{(1-h^i)}{(1-\rho^i)}\rho^i(1-\rho^i)[1:h^{i+1}] \\
&\quad = h^i(1-\rho^i)[1:h^{i+1}] - (1-h^i)\rho^i[1:h^{i+1}] \\
&\quad = h^i(1-\rho^i)[1:h^{i+1}] - (1-h^i)\rho^i[1:h^{i+1}] \\
&\quad = (h^i - \rho^i)[1:h^{i+1}]\ ,
\end{aligned} \tag{2.20}$$

where the simplification $W^i[1 : h^{i+1}]$ is a compact form of $W^i h^{i-1} + W^{i,0}$. Similarly we can calculate the gradients of the sleep phase. We continue the derivation from Eq. (2.15), where $\phi^i = \{V^i\}$ are the parameters of the distribution q

$$\begin{aligned}
&\nabla_{\phi^i} \ln q(h^i|h^{i-1}) = \\
&\quad = \nabla_{\phi^i} \ln \left(\psi^i\right)^{h^{i-1}} (1-\psi^i)^{1-h^{i-1}}\ \text{, where } \psi^i = s(V^i[1 : h^{i-1}]) \\
&\quad \dots \\
&\quad = (h^i - \psi^i)[1:h^{i-1}]\ .
\end{aligned} \tag{2.21}$$

Note 4
In these derivations we used $W^i[1 : h^{i+1}]$, however in practice we will simplify this notation even further by writing just $W^i h^{i+1}$ instead and implicitly understand that it includes also the bias term $W^{i,0}$. The same simplification also applies for the weights V^i of the recognition network q.

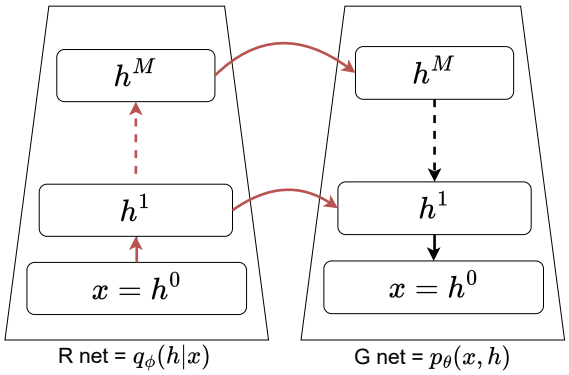


Figure 2.3: Wake phase propagation.

We see that the gradient of the sleep phase simplifies similarly to the gradient of the wake phase. The only thing left is to estimate the expectation from the Eqs. (2.13) and (2.15).

Implementation of Wake-Sleep

A straightforward way to estimate the gradients for the different phases is to do a Monte Carlo sampling on each layer of the HM. We can estimate the result of some expectation by sampling multiple times from the distribution according which the expectation is taken. According to the Law of Large Numbers (Wainwright et al., 2008) the average of the samples will converge to the true result of the expectation. Thus, to have better estimations of the gradients we can sample the distribution S times and average the results.

Another common technique in contemporary Machine Learning research, is the use of Minibatch Gradient Descent (MBGD) (Dekel et al., 2012) which we can use to our advantage. The MBGD method can be understood as a compromise between Stochastic Gradient Descent (SGD) and Batch Gradient Descent (BGD), where instead of using a single sample or the whole dataset for a single training step, we divide the dataset into equal sized so-called minibatches, and average their results per training step. MBGD has some implementation benefits from a parallel computation perspective, but what

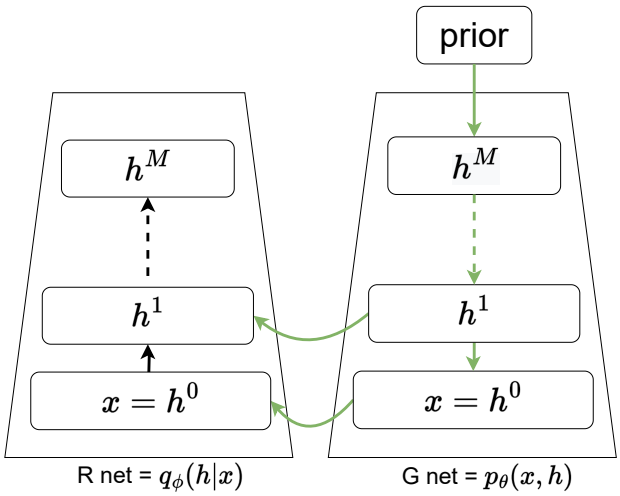


Figure 2.4: Sleep phase propagation.

we are interested in is using B samples of the minibatch to improve the estimation of our expectations.

Combining the Monte Carlo sampling with MBGD results in a better estimation of the expectations for the gradients, because of the increased number of total samples. In the case of the wake phase the sampling is done from the dataset, with $x \sim \mathcal{D}$ which are propagated first upwards through the recognition network q, by sampling h^i from $q(h^i|h^{i-1})$. In the generation network, we take a sample h^i sampled from the recognition network and transfer it to the generation network to sample h^{i-1} from it, as seen in Figure 2.3.[3]

The gradients are calculated layer-wise from the empirical estimation of the loss L_p ($\simeq \mathcal{L}_p$), as in Eq. (2.20), and are averaged over S samples and B number of elements in the mini-batch, i.e.,

$$\sum_{k=1}^{B\cdot S} \nabla_\theta L_{p,(k)} = \sum_{k=1}^{B\cdot S} \frac{\partial \ln p(x, h_{(k)})}{\partial \theta} \quad \text{with} \quad x_{(k)}, h_{(k)} \sim q(x, h) \; . \tag{2.22}$$

In the case of the sleep phase, we take binary samples from a uniform prior distribution, and we input these in the top-most layer h^M. This operation is called a "dream", hence the name, sleep. These samples are then propagated down through the generation network ($h^i \sim p(h^i|h^{i+1})$) and then again layer-wise into the recognition network, as shown in Figure 2.4. The gradients are again calculated layer-wise as described in Eq.(2.21) on the empirical

[3]When transferring a sample from one network to the other, we simply refer to them as layer-wise propagation.

estimation of the loss L_q ($\simeq \mathcal{L}_q$), with

$$\sum_{k=1}^{B\cdot S} \nabla_\phi L_{q,(k)} = \sum_{k=1}^{B\cdot S} \frac{\partial \ln q(h_{(k)}|x_{(k)})}{\partial \phi} \quad \text{with} \quad x_{(k)}, h_{(k)} \sim p(x,h)\ . \tag{2.23}$$

The full WS is described in pseudo-code in Algorithm 1 and visualized in Figure 2.5. At step t, let η be the learning rate, the final update rules will be

$$\begin{aligned} \theta_{t+1} &= \theta_t - \frac{\eta}{B\cdot S} \sum_{k=1}^{B\cdot S} \nabla_\theta L_{p,(k)} \\ \phi_{t+1} &= \phi_t - \frac{\eta}{B\cdot S} \sum_{k=1}^{B\cdot S} \nabla_\phi L_{q,(k)}\ . \end{aligned} \tag{2.24}$$

Algorithm 1: Wake-Sleep

Let x be a sample from the dataset
Let p and q be the joint distributions of the generation and the recognition networks with weights W and V, respectively
Let M be the depth of the HM
`wake phase update`
for *each layer i from q ascending with* $h^0 = x$ **do**
  Sample h^{i+1} from $q(h^{i+1}|h^i)$
  Compute the probability distribution $p(h^i|h^{i+1})$ of a sample h^i from p
  Compute the gradients $\nabla_{\theta^i} L_p$ with respect to W^i as in (2.22)
  Update W^i using η with the $\nabla_{\theta^i} L_p$ as in (2.24)
`sleep phase update`
for *each layer i from p descending with* h^M *sampled from the prior* **do**
  Sample h^{i-1} from $p(h^{i-1}|h^i)$
  Compute the probability distribution $q(h^i|h^{i-1})$ of a sample h^i from q
  Compute the gradients $\nabla_{\phi^i} L_q^s$ with respect to V^i as in (2.23)
  Update V^i using η with $\nabla_{\phi^i} L_q^s$ as in (2.24)

2.3 Advances in Helmholtz Machines

Since the introduction of the HM a number of alternative training algorithms were introduced, which we will present in this section. We use these algorithms

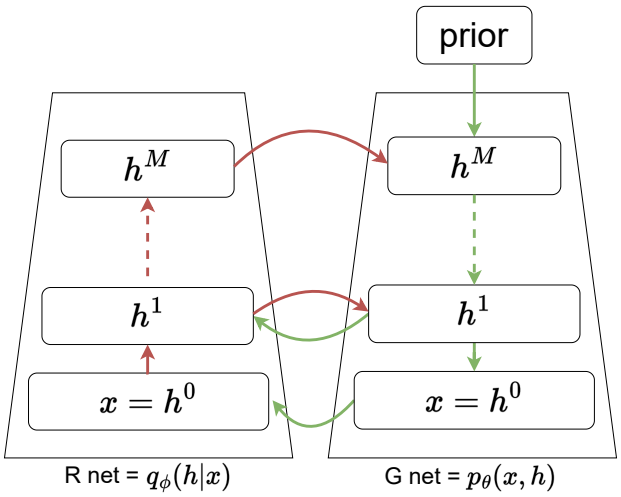


Figure 2.5: The structure of a Helmholtz Machine with M layers and a prior distribution over h^M. The colored arrows indicate the propagation of the samples during the wake (red) and sleep (green) phases. In the wake phase the sampling is done from $p_{\mathcal{D}}(x)$ and samples propagate through the recognition network by $q(h|x)$. In the sleep phase we draw a sample from the prior $p(h^M)$ ("dream") and propagate it through the generative network through $p(x|h)$.

as benchmarks for comparisons for our own algorithm later in the thesis. Furthermore we build on top of some of them, taking the most effective features of them and incorporating them in our own model in Section 5.3 and 5.3.8.

2.3.1 Reweighted Wake-Sleep

The Reweighted Wake-Sleep (RWS) was introduced by Bornschein and Bengio (2015). In their paper the authors revisit the training of HMs and show how the training algorithm WS can be recast in terms of a variational objective. They introduce a third training step and a weighting of the gradients and achieve state-of-the-art performance in reaching the best optimum for HMs. The main idea is motivated by the observation that

$$\ln p(x) \geq \sum_h q(h|x) \ln \frac{p(x,h)}{q(h|x)} \,, \tag{2.25}$$

which is true for any inference network q and generative network p, where this bound becomes tighter as the two distributions approach each other. This method of approximating a probability distribution from below as in the

right-hand side of Eq. (2.25) is also called importance sampling (Srinivasan, 2013). The inequality is an equality if and only if $q(h|x) = p(h|x)$.

In the context of HMs the authors have shown that the derivative of the log-likelihood of x can be approximated with the variational bound as

$$\begin{aligned}\frac{\partial}{\partial\theta}\mathcal{L}_p(x\sim\mathcal{D}) &= \frac{\partial \ln p(x)}{\partial\theta} \\ &= \frac{1}{p(x)}\sum_h \frac{\partial p(x,h)}{\partial\theta} \\ &= \frac{1}{p(x)}\sum_h p(x,h)\frac{\partial \ln p(x,h)}{\partial\theta} \\ &= \frac{1}{p(x)}\mathop{\mathbb{E}}_{h\sim q(h|x)}\left[\frac{p(x,h)}{q(h|x)}\frac{\partial \ln p(x,h)}{\partial\theta}\right] .\end{aligned} \tag{2.26}$$

They call this learning step **p-wake** or just **wake** update, which optimizes the objective $\mathcal{L}_p$ from Eq. (2.16) with samples coming from the dataset $p_{\mathcal{D}}$, as done in Eq. (2.22). These derivatives of $\mathcal{L}_p$ can be empirically estimated by the following formula

$$\sum_{k=1}^{S}\nabla_\theta L_{p,(k)}(x) = \sum_{k=1}^{S}\tilde{\omega}_k\frac{\partial \ln p(x,h_{(k)})}{\partial\theta} \quad \text{with} \quad h_{(k)}\sim q(h|x)\ . \tag{2.27}$$

The last step in Eq. (2.27) involves a Monte Carlo approximation of the expectation value from Eq. (2.26) with $\tilde{\omega}_k$ as importance weights, with

$$\tilde{\omega}_k = \frac{\omega_k}{\sum_{k'}\omega_{k'}}\ ,\ \text{with}\ \omega_k = \frac{p(x,h_{(k)})}{q(h_{(k)}|x)}\ . \tag{2.28}$$

The higher the number of samples S the tighter the approximation is. Notice that the last step of Eq. (2.26) is true because we can rewrite $p(x)$ as

$$p(x) = \sum_h q(h|x)\frac{p(x,h)}{q(h|x)} = \mathop{\mathbb{E}}_{h\sim q(h|x)}\left[\frac{p(x,h)}{q(h|x)}\right] \simeq \mathop{\frac{1}{S}\sum_{k=1}^{S}}_{h_{(k)}\sim q(h|x)}\frac{p(x,h_{(k)})}{q(h_{(k)}|x)}\ . \tag{2.29}$$

Eq. (2.26) can be referred to as the reconstruction likelihood which is optimized in terms of the parameters θ. We can notice that taking $S = 1$ is equivalent to the standard wake phase from Eq. (2.22).

With the help of the same variational bound the authors show how q can be optimized by minimizing the variance of the Monte Carlo estimation in Eq. (2.26), i.e., minimizing the KL divergence with the generative posterior (Bornschein and Bengio, 2015; Le et al., 2020). This can be averaged by

sampling x from the true data distribution $p_{\mathcal{D}}(x)$ with $h_{(k)} \sim q(h|x)$

$$\begin{aligned}\frac{\partial \mathcal{L}_q\left(x \sim p_{\mathcal{D}}(x)\right)}{\partial \phi} &\simeq \sum_{k=1}^{S} \nabla_\phi L^w_{q,(k)}(x) \\ &= \sum_{k=1}^{S} \tilde{\omega}_k \frac{\partial \ln q(h_{(k)}|x)}{\partial \phi} \text{ with } h_{(k)} \sim q(h|x),\end{aligned} \tag{2.30}$$

where $\nabla_\phi L^w_q(x)$ is the empirical estimate of the gradient. The authors call this step of optimization the **q-wake update**.

Another reasonable way to optimize the parameters ϕ is to maximize $\ln q(h|x)$ from the inference network $q(h|x)$, with x, h sampled from the generative model $p(x, h)$, i.e.,

$$\frac{\partial}{\partial \phi} \mathcal{L}^s_q((x,h) \sim p(x,h)) = \frac{\partial}{\partial \phi} \ln q(h|x) \ ,$$

$$\frac{\partial \mathcal{L}_q\left((x,h) \sim p(x,h)\right)}{\partial \phi} \simeq \sum_{k=1}^{S} \nabla_\phi L^s_{q,(k)}(x) = \sum_{k=1}^{S} \frac{\partial \ln q(h_{(k)}|x_{(k)})}{\partial \phi} \ , \tag{2.31}$$

with $\nabla_\phi L^s_q(x)$ as the empirical estimate of the gradient for the **q-sleep update**. This is equivalent to the **sleep** phase in the classic WS from Eq. (2.23).

RWS has been shown to learn better discrete models than the standard VAE (Rezende et al., 2014) and its convergence properties were extensively studied by Le et al. (2020). The full pseudo-code of the algorithm can be seen in Algorithm 2. The final update rules of the RWS are the following

$$\begin{aligned}\theta_{t+1} &= \theta_t - \eta \frac{1}{B} \sum_{r=1}^{B} \sum_{k=1}^{S} \tilde{\omega}_k \nabla_\theta L_{p,(k,r)} \ , \\ \phi_{t+1} &= \phi_t - \frac{\eta}{2} \frac{1}{B} \sum_{r=1}^{B} \sum_{k=1}^{S} \left(\frac{1}{S} \nabla_\phi L^s_{q,(k,r)} + \tilde{\omega}_k \nabla_\phi L^w_{q,(k,r)} \right) \ .\end{aligned} \tag{2.32}$$

The empirical estimates of the loss $L^{k,r}$ are computed with minibatches of size B indexed by r, sampled each S times with indices k.

2.3.2 Bidirectional Helmholtz Machines

In 2016, Bornschein et al. (2016) described a new type of optimization algorithm using a different objective for the minimization, called the Bidirectional Helmholtz Machine (BiHM). Their model is still an HM, with a generation

Algorithm 2: Reweighted Wake-Sleep

Let x be a sample from the dataset
Let p and q be the distributions of the generation and the recognition networks with weights W and V
Let ω be the importance weights from the RWS
Let M be the depth of the HM
`#wake phase update`
for *each layer i from q ascending with $h^0 = x$* **do**
    Sample h^{i+1} from $q(h^{i+1}|h^i)$
    Compute the gradients $\nabla_{\theta^i} L_p$ with respect to W^i as in (2.27)
    Compute the weights $\tilde{\omega}$ from the multiple samples (2.28)
    `#q-wake update`
    Compute the gradients $\nabla_{\phi^i} L_q^w$ with respect to V^i similarily to the `sleep` phase as in (2.30)
    Update W^i and V^i using η with the $\nabla_{\theta^i} L_p$ and $\nabla_{\phi^i} L_q^w$ with the $\tilde{\omega}$-s as in (2.32)
`#sleep phase update`
for *each layer i from p descending with h^M sampled from the prior* **do**
    Sample h^{i-1} from $p(h^{i-1}|h^i)$
    Compute the gradients $\nabla_{\phi^i} L_q^s$ with respect to V^i as in (2.31)
    Update V^i using η with $\nabla_{\phi^i} L_q^s$ as in (2.32)

network p and a recognition (or inference) network q, on which the authors define a new probability distribution as the geometric mean of the two joint distributions p and q as

$$p^*(x,h) = \frac{1}{Z}\sqrt{p(x,h)q(h,x)}\ ,$$

where Z is a normalization constant

$$Z = \sum_{x,h}\sqrt{p(x,h)q(h,x)}\ .$$

The authors define $q(x)$ through the marginalization

$$q(x) = p^*(x) = \sum_h p^*(x,h)$$

and they show that it equals

$$\begin{aligned} q(x) &= \sum_h p^*(x,h) \\ &= \frac{\sqrt{q(x)}}{Z}\sum_h \sqrt{p(x,h)q(h|x)} \\ &= \left(\frac{1}{Z}\sum_h \sqrt{p(x,h)q(h|x)}\right)^2\ . \end{aligned}$$

Now using the Cauchy-Schwarz inequality

$$|\sum_h f(h)g(h)|^2 \leq \sum_h |f(h)|^2 \sum_h |g(h)|^2\ ,$$

we can notice that $Z \leq 1$ and this gives an upper bound on the unnormalized probability $\tilde{p}^*(x)$

$$\tilde{p}^*(x) = \left(\sum_h \sqrt{p(x,h)q(h|x)}\right)^2 = Z^2 p^*(x) \leq p^*(x)\ .$$

From the above equality it also follows

$$\ln p^*(x) \geq \ln \tilde{p}^*(x)$$

and thus the unnormalized likelihood $\ln \tilde{p}^*(x)$ is a lower bound of the likelihood, so it can be used as proxy for the optimization. Furthermore $\tilde{p}^*$ can be

estimated via importance sampling

$$\tilde{p}^*(x) = \left(\sum_h \sqrt{p(x,h)q(h|x)}\right)^2 = \left(\mathop{\mathbb{E}}_{h\sim q(h|x)}\left[\sqrt{\frac{p(x,h)}{q(h|x)}}\right]\right)^2$$
$$\simeq \left(\frac{1}{S}\sum_{k=1}^{S}\sqrt{\frac{p(x,h_{(k)})}{q(h_{(k)}|x)}}\right)^2 \quad \text{with } h_{(k)} \sim q(h|x)\ ,$$

which, by recalling Eq. (2.29) and using Jensen's inequality ($f(\mathrm{E}[x]) \leq \mathrm{E}[f(x)]$ for a convex function f), results in

$$p(x) \simeq \frac{1}{S}\sum_{k=1}^{S}\frac{p(x,h_{(k)})}{q(h_{(k)}|x)} \geq \left(\frac{1}{S}\sum_{k=1}^{S}\sqrt{\frac{p(x,h_{(k)})}{q(h_{(k)}|x)}}\right)^2 \simeq \tilde{p}^*(x)\ .$$

This is a key inequality in the paper of Bornschein et al. (2016) since this means that optimizing their proposed objective $\tilde{p}^*(x)$ is a proxy for the optimization of $p^*(x)$ which is a proxy for $p(x)$.

The training in the paper proceeds with the minimization of $\ln \tilde{p}^*(x)$, estimated analogously to the RWS paper (Bornschein and Bengio, 2015). We can start from calculating the derivatives by ξ which is now a placeholder for both θ and ϕ

$$\frac{\partial \ln \tilde{p}^*(x)}{\partial \xi} = \frac{\partial}{\partial \xi}\ln\left(\sum_h \sqrt{p(x,h)q(h|x)}\right)^2 \tag{2.33}$$
$$= \frac{2\sum_h \sqrt{p(x,h)q(h|x)}\frac{\partial}{\partial \xi}\ln p(x,h)q(h|x)}{\sum_{h'}\sqrt{p(x,h')q(h'|x)}} \tag{2.34}$$
$$\simeq \mathop{\mathbb{E}}_{h\sim\pi(h|x)}\left[\frac{\partial}{\partial \xi}\ln p(x,h)q(h|x)\right]\ , \tag{2.35}$$

where

$$\pi(h|x) = \frac{\sqrt{p(x,h)q(h|x)}}{\sum_{h'}\sqrt{p(x,h')q(h'|x)}}\ .$$

Now using importance sampling estimation

$$\mathbb{E}_{h\sim\pi(h|x)}\left[\frac{\partial}{\partial \xi}\ln p(x,h)q(h|x)\right] \simeq \sum_k \tilde{\omega}_k \frac{\partial}{\partial \xi}\ln p(x,h_{(k)})q(h_{(k)}|x) \tag{2.36}$$
$$\text{with}\quad h_{(k)} \sim q(h|x)\ ,$$

with importance weights

$$\omega_k = \sqrt{\frac{p(x, h_{(k)})}{q(h_{(k)}|x)}}\ ; \quad \tilde{\omega}_k = \frac{\omega_k}{\sum_{k'} \omega_{k'}}\ . \tag{2.37}$$

We can fill in both θ and ϕ in place of ξ in Eq. (2.33) and Eq. (2.36) to get the real and the estimated gradients. The authors use the same loss for both parameter updates, which means that the algorithm does not alternate two phases but updates both sets of parameters simultaneously. One training step for the BiHM instead of two means that there is reduction in execution time, for the same number of epochs, which is a large improvement, compared to WS and RWS. In addition, BiHM is shown to outperform its counterparts by reaching better optimums on the same datasets.

Notice how the weights in Eq. (2.37) compare to the importance weights in Eq. (2.28), we can see that the actual weighted gradients following from (2.36) are the same as the ones of RWS **wake** and **q-wake** phases (Eqs. (2.26) and (2.30)), with slightly different weights (see the extra square root). The full pseudo-code of the algorithm can be seen in Algorithm 3 and the final update rules of the BiHM are

$$\begin{aligned} \theta_{t+1} &= \theta_t - \frac{\eta}{B} \sum_{r=1}^{B} \sum_{k=1}^{S} \tilde{\omega}_k \nabla_\theta L_{p,(k,r)} \quad \text{and} \\ \phi_{t+1} &= \phi_t - \frac{\eta}{B} \sum_{r=1}^{B} \sum_{k=1}^{S} \tilde{\omega}_k \nabla_\phi L^w_{q,(k,r)}\ , \end{aligned} \tag{2.38}$$

with $\nabla_\theta L_p$ and $\nabla_\phi L^w_q$ defined as in Eqs. (2.27) and (2.30) and are computed with minibatches of size B indexed by r, sampled each S times with indices k.

2.3.3 REINFORCE

Originally, a possible solution for learning HMs was the REINFORCE algorithm (Williams, 1992), which already had uses in inference problems. The major difference between the REINFORCE and WS methods is that the former only has one training objective, the ELBO, presented before in Eq. (2.12). While optimizing w.r.t. θ in REINFORCE is equivalent to the wake phase, optimizing w.r.t. ϕ has the well-known side-effect that the variance of the estimation of the gradients grows linearly with the size of the network. For this reason, it is not commonly used to train HMs.

The REINFORCE algorithm is used commonly in conjunction with other algorithms specifically in the field of Reinforcement Learning (RL) as policy

Algorithm 3: Bidirectional Helmholtz Machine

Let x be a sample from the dataset
Let p and q be the distributions of the generation and the recognition networks with weights W and V
Let ω be the importance weights from the BiHM
Let M be the depth of the HM
for *each layer* i *from* q *ascending with* $h^0 = x$ **do**
    Sample h^{i+1} from $q(h^{i+1}|h^i)$
    Compute the weights $\tilde{\omega}$ from the multiple samples
    Compute the gradients $\nabla_{\theta^i} L_p$ with respect to W^i as in (2.27)
    Compute the gradients $\nabla_{\phi^i} L_q^w$ with respect to V^i similarily to `q-wake` as in (2.30)
    Update W^i and V^i using η with the $\nabla_{\theta^i} L_p$ and $\nabla_{\phi^i} L_q^w$ with the $\tilde{\omega}$-s as in (2.38)

learning method (Mnih and Gregor, 2014). The advantage of using it in RL is that it enables the training algorithm to learn the policy together with the value and state functions, instead of learning it separately. However the issue of reducing the variance is addressed in several modern variants (Mnih and Gregor, 2014; Tucker et al., 2017; Grathwohl et al., 2018; Kool et al., 2020).

The exploding variance of the REINFORCE is not a problem in the case of the WS, because of the switched objective in the sleep phase. This is in fact is the reason why REINFORCE is not used in the literature and practice for training HMs. Therefore, in this thesis we will not conduct any experiments with REINFORCE and we will not do any comparison to it.

2.4 Other Generative models

In this section we give a short introduction of alternative generative models which are used in Deep Learning, other than HMs.

2.4.1 Variational Auto-Encoder

An approach which proved to be successful and thus common to several deep generative models is based on the use of two separate networks: the recognition network, i.e., the encoder, which provides a compressed latent representation for the input data, and the generative network, i.e., the decoder, able to reconstruct the observation in output. Autoencoders (AE) (Goodfellow et al., 2016) are classic examples of this paradigm, where both the encoder and the

decoder are commonly implemented as deterministic feed-forward networks. Variational Autoencoders (VAE) (Kingma and Welling, 2014), (Rezende et al., 2014) on the other hand, introduce an approximate posterior distribution over the latent variables which are then sampled, thus resulting in stochastic networks. While HMs (Dayan et al., 1995) consist also of recognition and generative networks, these are both commonly modeled as SBNs (Neal, 1992), characterized by discrete hidden variables. In comparison, standard VAEs commonly adopt continuous Gaussian variables only in the bottleneck layer.

In the presence of continuous hidden variables, for which the stochastic back-propagation of the gradient is possible, as in VAEs, the generative and recognition networks can be trained simultaneously, through the definition of the ELBO, which corresponds to a lower-bound for the likelihood and the use of the reparametrization trick (see Eq. (2.12)). In the presence of discrete hidden variables, as for HMs, this approach **cannot be directly employed**, and thus the need for the WS (Hinton et al., 1995).

Because VAE is one of the more common contemporary techniques, and it is closely related to WS, we include them in some of our comparisons when evaluating experimental results in Chapter 5.

2.4.2 Restricted Boltzmann Machine

The Restricted Boltzmann Machine as a deep generative model is one of the first introduced (Smolensky, 1986), highly successful and highly popular models (Hinton, 2002), (Hinton et al., 2006), (Hinton and Salakhutdinov, 2006) and (Zhang et al., 2018b). Similarly to the models described before, it has an architecture of stacked layers, one on top of other, however with the crucial difference that the edges connecting the nodes of the layers are undirected. In practice this means that for each layer there is a single set of weights which is being applied in both directions in the network, when it is used either as generative or recognition network.

The undirected nature of the edges creates a huge conceptual difference between the RBM and the models which consist of two different networks, like the HM and VAEs. While the goal in the two-network models is to learn some distribution with one network and at the same time train a second network to learn the former, the RBM learns only one set of weights in both directions. The RBM is trained with the help of an algorithm called Contrastive Divergence (Hinton, 2002), which uses Gibbs-sampling to sample alternatingly two adjacent layers. An overview of newer versions of these algorithms is given in the work of Zhang et al. (2018b). Typically, the improvements on RBMs are aimed towards learning faster and more accurate estimations by improving the effectiveness of Contrastive Divergence and

Gibbs-Sampling, which is known to be slow to converge.

The geometric properties of RBM, such as their expressive power, have been studied in depth (Montúfar and Morton, 2015; Montúfar et al., 2015) more recently. Indeed it has been shown that RBMs are universal approximators, with no theoretical drawbacks to feed-forward networks (Montúfar, 2015, 2016).

2.4.3 Generative Adversarial Networks

Generative Adversarial Networks (GAN) were introduced by Goodfellow et al. (2014) and raised to popularity in recent years because of their impressive capabilities (Karras et al., 2019) as generative models. GANs consist of two networks which are called the generator and the discriminator, which are trained in an adversarial setting, where the generator learns to generate samples that look like they come from the target distribution, while the discriminator learns to differentiate between the generated samples and the target ones. While the construction of two networks might remind us of the two networks of HMs and VAEs, the training objective represents a huge paradigm shift in how the algorithm learns.

First, the generator and discriminator functions in principle need not be stochastic networks, virtually any differentiable function can be used as these, which creates already difficulties in comparing GANs with algorithms based on stochastic networks. Secondly, the alternating min-max optimization used to train the two networks of a GAN is conceptually very different from the training of the HM. This makes it very difficult to cast the learning objective of GANs as a variational inference problem, which most other generative models discussed here are.

There are more modern versions of GANs as in the work of Arjovsky et al. (2017), where the authors have introduced some variational inference, stochasticity, and geometrical concepts into their model, by using the Wasserstein I metric (Vaserstein, 1969).

Note 5

In our work, we only compare with training algorithms defined for the HM. Sometimes we report results with VAE given that there is a reference for the benchmark that we are testing.

Chapter 3

Information Geometry

Information Geometry (IG) (Amari, 1985; Amari and Nagaoka, 2000; Amari, 2016; Ay et al., 2017) studies the geometry of statistical models using the language of Riemannian and affine geometry. For a better understanding of the following sections, a baseline understanding of some differential geometry concepts is required. In the present chapter, we will provide a brief introduction of the concepts that pertain to the thesis but we do not provide rigid definitions for all of them. For additional details, we refer the reader to the first two chapters from the book of Amari and Nagaoka (2000) or the works of Nielsen (2020), Kass and Vos (2011) or Murray and Rice (1993).

3.1 Manifolds

A D-dimensional manifold $\mathcal{M}$ is a set of points such that each point has D-dimensional extensions in its neighborhood. Such a neighborhood is locally topologically equivalent to a D-dimensional Euclidean space $\mathbb{R}^D$. Given a manifold $\mathcal{M}$ with D dimensions, we can define a function $\varphi : \mathcal{M} \to \mathbb{R}^D$ as a coordinate system, which maps every point p on $\mathcal{M}$ to D real numbers $\varphi(p) = [\xi_1, \cdots, \xi_D]$, which we call the coordinates of the point[1]. In this work, we are only interested in smooth manifolds, which means that at any point $\mathcal{M}$ is differentiable infinitely many times (the atlas is C^∞).

At each point $p \in \mathcal{M}$, we can define a tangent space $T_p\mathcal{M}$ that locally "linearizes" the manifold. On the tangent space, the inner product is defined by the metric tensor g, which often depends on the specific point p on the manifold. The definition of the inner product space on $T_p\mathcal{M}$ enables the calculation of angles between vectors and vector lengths. In this work, we only discuss and use the Fisher-Rao metric described in Section 3.2.

In addition, we can define another local structure on the tangent bundle $T\mathcal{M}$, the affine connection ∇ which is a differential operator, that makes it

[1]Generally, not all manifolds have a global coordinate system, as we have defined here. Some have local coordinate systems defined on disjoint open subsets of the manifold, in order to cover the whole manifold. In this work we will not consider any such cases, and will always assume a global coordinate system and by extension global coordinates.

possible to define the covariant derivative, parallel transport, and the intrinsic curvature and torsion of the manifold. In this work, we only discuss and use the Levi-Civita connection described in Section 3.5.

For more details about manifolds, we refer the reader to the books of Lee (2010, 2013, 2018) on the subject.

3.1.1 Statistical Models

We assume the reader to be familiar with probability theory, nevertheless, we will give a brief introduction of basic concepts. A probability distribution is a function μ that is defined on $(\mathcal{X}, \Sigma)$, a set $\mathcal{X}$ with σ-algebra Σ, where $\mu(\mathcal{X}) = 1$. Using μ we can define a probability density function p as

$$p(x) \geq 0 \ (\forall x \in \mathcal{X}) \ \text{ and } \int p(x)\mathrm{d}\mu(x) = 1$$

in case $\mathcal{X}$ is a continuous set and

$$p(x) \geq 0 \ (\forall x \in \mathcal{X}) \ \text{ and } \sum_{x \in \mathcal{X}} p(x) = 1$$

in case $\mathcal{X}$ is discrete.

Now consider a set of probability density functions $\mathcal{S}$ which can be parameterized by a set of parameters $[\xi_i]$ from an open subset Ξ of $\mathbb{R}^D$, where D is the dimension of the parameter space. We write such a set as

$$\mathcal{S} = \{ \ p_\xi = p(x;\xi) \mid \xi = [\xi_1, ..., \xi_D] \in \Xi \ \} \ ,$$

or abbreviate it as $\mathcal{S} = \{p_\xi\}$ and we call it a parametric model or a statistical model. Most of the probability density functions that we are familiar with are such parametric models: Bernoulli, Gaussian, Multivariate Gaussian and the whole exponential and mixture families of probability distributions.

3.1.2 Statistical Manifolds

Given a statistical model $\mathcal{S} = \{p_\xi\}$, under certain regularity conditions for ξ (Amari and Nagaoka, 2000, Section 1.1), $\mathcal{S}$ is a statistical manifold and has the parameterization $[\xi]$ as a coordinate system (as there are infinitely many) (Lauritzen, 1987, Chapter 4).

By looking at statistical models as manifolds, Machine Learning tasks (regression, classification or more generally estimation tasks) beget geometric interpretations. An estimation procedure for predictive models, as in the case of HMs, in fact assumes there is some goal probability distribution p^* that

usually needs to be approximated as good as possible by some parametric model p_ξ. This problem is equivalent to finding the point p_{ξ^*} on the manifold that is the closest to the projection of p^* (or more generally, the closest w.r.t. to some cost function to p^*). A visual representation of this can be seen in Figure 3.1. A gradient descent based algorithm moves with small steps on

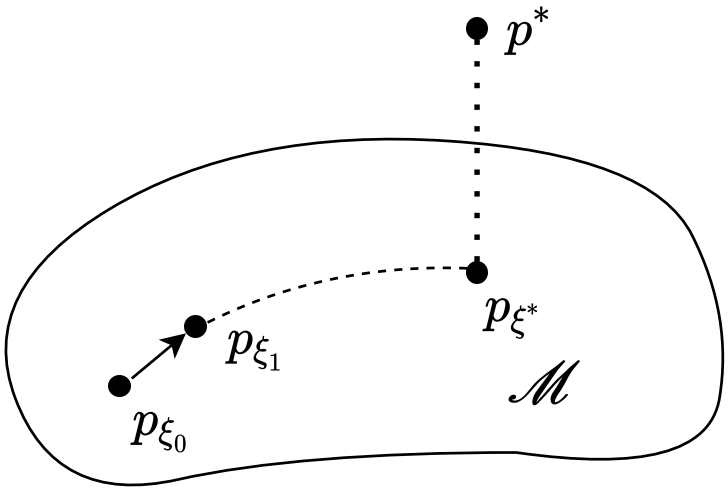


Figure 3.1: A geometric interpretation of learning based on GD. The GD algorithm taking steps from an initial point p_{ξ_0} in the direction of the gradient pointing to the optimal solution p_{ξ^*}, which is the closest to the goal p^*.

the manifold towards the optimal solution p_{ξ^*}. In this setting accounting for the geometry of the manifold is crucial for an effective and efficient learning.

A question arises naturally, if statistical model $\mathcal{S}$ can be interpreted as a manifold, then what is the appropriate metric tensor g and an affine connection ∇ for it? In the next sections we present the Fisher-Rao metric and Levi-Civita connection which are commonly used in Machine Learning applications.

3.2 Fisher-Rao Metric

In IG, statistical manifolds are commonly endowed with the Riemannian Fisher-Rao metric over the tangent bundle. By Chentsov's theorem, the Fisher-Rao metric on statistical models is the **only** Riemannian metric (up to rescaling) that is invariant under sufficient statistics (Čencov, 1978; Amari and Nagaoka, 2000, Chapter 2.4).

The metric is defined by the expected value in point p of the product of two tangent vectors, represented by the derivation of log probabilities, which

are centered random variables, i.e., with zero expected value. Given a basis for the tangent space, derived from the choice of the parametrization, the inner product associated with the Fisher-Rao metric is represented through a quadratic form expressed with the Fisher Information Matrix (FIM) $\mathcal{F}$ with entries g_{ij}. The elements of the FIM of $\mathcal{S}$ is defined at a given point ξ as

$$g_{ij}(\xi) = \mathbb{E}_\xi[\partial_i \ln p_\xi(x) \partial_j \ln p_\xi(x)] = \int p_\xi(x)\ \partial_i \ln p_\xi(x) \partial_j \ln p_\xi(x) dx\ , \quad (3.1)$$

where $\partial_i = \frac{\partial}{\partial \xi_i}$ and $\mathbb{E}_\xi$ is the expectation at point ξ. Another equivalent formulation is given by

$$g_{ij}(\xi) = -\mathbb{E}_\xi[\partial_i \partial_j \ln p_\xi(x)]\ . \quad (3.2)$$

It is easy to see that the FIM $F(\xi)$ is symmetric and it is also easy to show that it is positive-definite since the partial derivatives $\{\partial_1 p_\xi, \ldots, \partial_D p_\xi\}$ are linearly independent for smooth manifolds. The Fisher-Rao metric is an intrinsic geometric quantity and it is invariant to the choice of coordinate systems.

3.3 Natural Gradient

Amari (1997, 1998) in his seminal work introduces the Natural Gradient (NG) and shows how it can be used to speed up the convergence rate of a gradient descent optimization algorithm by taking a step in the direction of the steepest descent w.r.t. the metric of the space. The use of the NG has shown its advantages (but also limitations) in Machine Learning.

In Gradient Descent (GD) we minimize a loss function $\mathcal{L}$ over a parameter space Θ, with parameters $\theta \in \Theta$. The optimization step of GD for the minimization of $\mathcal{L}$ at iteration t is formulated as

$$\theta^{t+1} = \theta^t - \eta \nabla \mathcal{L}(\theta^t)\ , \quad (3.3)$$

where η is the learning rate and ∇ denotes the vector of partial derivatives $\frac{\partial}{\partial \theta_i}$. In this thesis we are dealing with optimization over the parameter space of probabilistic models, so we are going to assume implicitly that Θ are parameters of some probability density function p. Additionally, we suppose $\mathcal{L}$ is a cost function over the statistical model, such as the negative log-likelihood

$$\mathcal{L}(\theta) = -\mathbb{E}\left[\ln p_\theta(x)\right]\ . \quad (3.4)$$

It is well known that, under certain conditions on the choice of η, GD in Eq. (3.3) converges to a local optimum θ^* where all elements of $\nabla \mathcal{L}(\theta^*)$

approach 0. However, if we use a different parametrization $\zeta \in \Theta$ for the function $\mathcal{L}$

$$\zeta^{t+1} = \zeta^t - \eta \nabla \mathcal{L}(\zeta^t) \ ,$$

we arrive at a different optimum ζ^*, depending on η, where $p_{\theta^*} \neq p_{\zeta^*}$. The reason is that GD is not invariant with respect to the parametrization, in particular because it does not take into account the underlying geometry of the parameter space Θ.

The NG can be used to address the previous two issues. If we consider the GD from a geometric perspective, the optimization of $\mathcal{L}$ should be done over the statistical manifold $\mathcal{M}$, as shown in Figure 3.1. In NG the optimization step is computed in the direction of the steepest descent over a statistical manifold $\mathcal{M}$ with associated metric g given by the FIM represented by the Riemannian gradient of $\mathcal{L}$. For infinitesimal steps the NG, by using the Fisher-Rao metric, is independent of the choice of parametrization of θ or ζ.

The vector of partial derivatives $\nabla \mathcal{L}$ represents the coordinates of a covector on the cotangent space, $\frac{\partial}{\partial \theta} \mathcal{L}(\theta) \in \mathrm{T}_p^* \mathcal{M}$. The NG is the vector in $\mathrm{T}_p \mathcal{M}$ associated to $\nabla \mathcal{L}$ through the canonical isomorphism between tangent and cotangent space induced by the metric (Amari, 1998), i.e.,

$$\widetilde{\nabla} \mathcal{L}(\theta) = \mathcal{F}(\theta)^{-1} \nabla \mathcal{L}(\theta) \ , \tag{3.5}$$

where $\mathcal{F}(\theta)$ is the FIM defined as

$$\begin{aligned} \mathcal{F}(\theta) &= \mathbb{E}\left[\frac{\partial}{\partial \theta} \log p(\theta) \left(\frac{\partial}{\partial \theta} \log p(\theta) \right)^\top \right] \\ &= -\mathbb{E}\left[\frac{\partial^2}{\partial \theta \partial \theta} \log p(\theta) \right] \ , \end{aligned} \tag{3.6}$$

adapted from Eq (3.1), where $\frac{\partial}{\partial \theta} \log p(\theta)$ is the vector of partial derivatives, and $\frac{\partial^2}{\partial \theta \partial \theta} \log p(\theta)$ is the Hessian matrix. The natural gradient descent update then takes the form

$$\theta_{t+1} = \mathrm{Exp}_{\theta_t} \left(\eta \widetilde{\nabla} \mathcal{L}(\theta_t) \right) \ , \tag{3.7}$$

where the exponential map operator Exp (explained in Subsection 3.5.3) is a retraction that guarantees that the iterates remain on the manifold (Absil et al., 2009). In practice, often the Exp is avoided and the NG update is used as

$$\theta_{t+1} = \theta_t - \eta \widetilde{\nabla} \mathcal{L}(\theta_t) \ , \tag{3.8}$$

as it is easier to compute and knowing that the domain of the θ is $\mathbb{R}^n$ and there is no risk that θ_{t+1} results in values which are not associated to the distribution (Amari, 1998)[2].

Because the natural gradient $\widetilde{\nabla}\mathcal{L}(\theta_t)$ follows the steepest descent, with respect to the metric of the underlying space, this in practice results in much faster convergence (i.e., fewer steps) compared to regular (Euclidean) GD, as proven by Amari (1998).

The Euclidean gradient is a special case of the Riemannian gradient, where the FIM is the identity matrix, which describes the metric of the Euclidean manifold.

3.4 Kullback–Leibler Divergence

In this work, we are mainly concerned with learning generative models, and more specifically HMs. In these models, we have two distributions representing the generative and recognition networks p and q. The primary goal of HMs, similar to other generative models, is to learn a parametrized distribution $p_\theta(x)$ of some target distribution $p_\mathcal{D}(x)$, i.e., for the two to be as similar as possible $p_\theta(x) \simeq p_\mathcal{D}(x)$. This goal is often obtained by learning a second distribution simultaneously, i.e., to do inference of the parameters by learning the recognition network $q(h|x) \simeq p(h|x)$. In order to compare the similarities between probability distributions, we need a measure that tells us how close those distributions are. This is where the Kullback-Leibler (KL) divergence becomes useful.

Divergences are a common way to measure dissimilarity between probability distributions. A divergence is defined as a function $\mathrm{D} : P(\mathcal{X}) \times P(\mathcal{X}) \rightarrow \mathbb{R}^+$, where $P(\mathcal{X})$ is a set of probability distributions defined on $\mathcal{X}$. The properties of a divergence D are:

1.
$$\mathrm{D}(p||q) \geq 0 \text{ for all } p, q \in P(\mathcal{X}) \; ; \tag{3.9}$$

2.
$$\mathrm{D}(p||q) = 0 \text{ if and only if } p = q. \tag{3.10}$$

Note 6
There are two important properties to take notice of:

[2]To better understand the notion of the retraction, we point the reader to a more detailed explanation in the work Absil et al. (2009, Section 4.1).

1. *There is no rule for symmetry, although for some divergences* $\mathrm{D}(p||q) = \mathrm{D}(q||p)$ *holds true, in general it is not true.*

2. *Sometimes divergences are erroneously referred to as distances, however, these represent a different geometrical concept. Although Eqs.* (3.9) *and* (3.10) *hold also for the concept of distances, distances also have to be symmetric and obey the triangle inequality* $d(x, z) \leq d(x, y) + d(y, z)$*, which do not necessarily apply to divergences.*

On statistical manifolds, there is a number of divergences that can be defined, such as the family of f-divergences and α-divergences (Amari, 2016, Chapters 3.3 and 4.1). The one that most interests us however is the KL divergence as it is the most commonly used divergence to compare probability distributions and plays a key role in HMs. The KL divergence is defined as

$$\mathrm{D}_{KL}\left[p(x)||q(x)\right] = \int p(x) \ln \frac{p(x)}{q(x)} dx \ .$$

We see that the KL divergence (being part of the f-divergence family) is invariant to the parametrization of the distributions p and q (Rényi, 1961).

There is also a strong connection between the KL and the Fisher-Rao metric (Jeffreys, 1946). We can recover the entries g_{ij} of the FIM (up to a scaling factor) from the KL divergence by looking at how an infinitesimal change $\mathrm{d}\theta$ to the parameters θ affects the parametrization of the distribution given the divergence

$$\begin{aligned} \mathrm{D}_{KL}\left[p_\theta(x)||p(x;\theta+\mathrm{d}\theta)\right] &= \int p_\theta(x) \ln \frac{p(x;\theta+\mathrm{d}\theta)}{p_\theta(x)} \mathrm{d}x \\ &\simeq \frac{1}{2} \sum g_{ij}(\theta) \mathrm{d}\theta_i \mathrm{d}\theta_j \ . \end{aligned}$$

This approximation is done by taking the Taylor expansion of the natural logarithm from within the KL divergence (for a detailed explanation, consult the work of Jeffreys (1946)).

3.5 Levi-Civita Connection and Parallel Transport

The connection or covariate derivative describes how the tangent space of a manifold changes on an infinitesimal scale. In Riemannian geometry, the most commonly used connection is the Riemannian or Levi-Civita connection, which is uniquely defined on every Riemannian manifold (Lee, 2018; Amari,

2016). The two most important properties of this type of connection, by its definition, are that it preserves the metric $\nabla g = 0$ (where ∇ is the covariant derivative operator) and that it is torsion-free[3]. All connections ∇ can be defined on the tangent bundle, with coordinate basis vector fields $\partial_1, \ldots, \partial_D$, where ∇_j stands for ∇_{∂_j}, as

$$\nabla_j \partial_k = \Gamma^l_{jk} \partial_l \ . \tag{3.11}$$

The Γ^l_{jk}-s are called the Christoffel symbols, which describe how each individual basis vector changes in relationship to other basis vectors. For the Levi-Civita connection, these symbols are defined as

$$\Gamma^l_{jk} = \frac{1}{2} g^{lm} \left(\partial_k g_{mj} + \partial_j g_{mk} - \partial_m g_{jk} \right) \ ,$$

where g_{ab} are the elements of the metric described by the FIM in Eq. (3.2), and g^{ab} are the elements of the inverse of the FIM.

3.5.1 Parallel Transport

Connections allow us to introduce the concept of parallel transport, which is the action of transporting vectors along smooth curves on a manifold (Amari, 2016) from one tangent plane to another. The connection describes how we can move a vector v from a point p to $p + \mathrm{d}p$. The parallel transport takes it further, which means that we can take a vector v defined at a point p on the tangent bundle $T_p\mathcal{M}$ of a manifold $\mathcal{M}$ and move it to a different point p' defined on $T_{p'}\mathcal{M}$ along a curve, such that it remains parallel with itself during the movement on the curve. The parallel transport describes how this move is done along some smooth curve γ, where $\gamma(0) = p$ and $\gamma(1) = p'$. We write this transport as

$$v\left(\gamma(1)\right) = \prod_{p=\gamma(0)}^{p'=\gamma(1)} v\left(\gamma\left(0\right)\right) \ ,$$

where at each point on the curve, the direction of the vector is preserved along the curve, i.e., is parallel with itself w.r.t. the connection. This condition can also be formalized as the covariant derivative along the curve is zero, i.e., $\nabla_{\dot{\gamma}} v(t) = 0$.

In general, the parallel transport is not uniquely defined by the start and end-points p and p' but depends also on the curve γ. A consequence of

[3]Torsion-free means that the torsion-tensor is zero, which describes how much a vector rotates (or rolls) around its axis. Thus in the case of the Levi-Civita connection, the vectors do not rotate.

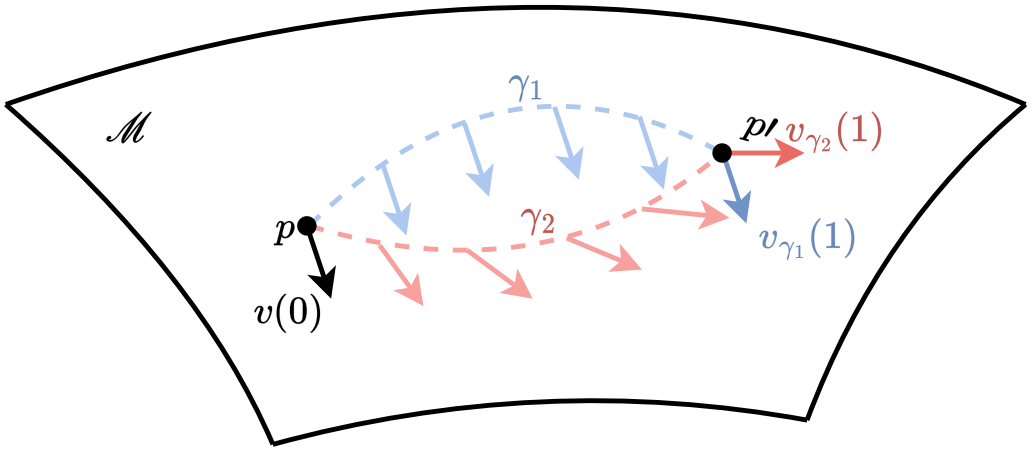


Figure 3.2: Parallel transport of the vector v along the curves γ_1 and γ_2.

this is that transporting a vector v along two different curves with the same beginning and end-points may result in different orientations of the same vector at the target space in p', as presented in Figure 3.2.

Under the Levi-Civita connection, we can define a unique Riemannian geodesic γ which by definition is a smooth curve that minimizes the distance according to the metric g. Using this connection, we also have the following property, for any vector v_1 and v_2

$$\langle v_1, v_2 \rangle_{\gamma(0)} = \left\langle \prod v_1, \prod v_2 \right\rangle_{\gamma(1)} .$$

The property states that the parallel transport along γ preserves the angle between vectors if they have been transported along the same geodesic and it also implies that the vector is unchanged in length after the transport.

Remark 1
We call a manifold flat w.r.t. ∇ if there exists an affine coordinate system θ, such that

$$\nabla_j \partial_i = 0 ,$$

for all $i \neq j$, from which it immediately follows

$$\Gamma_{ij}^k = 0 ,$$

based on Eq. (3.11). This means that for flat manifolds all geodesics w.r.t. the connection ∇ are straight lines ((Amari and Nagaoka, 2000), Section 1.7) and that parallel transport of vectors on flat manifolds works the same as in Euclidean space. This implies that the direction and length of the vectors do not change, i.e., two vectors defined at two different points on the manifold

can be simply added together since the shift from one space to another leaves the vector unchanged. A visual representation can be seen in Figure 3.3.

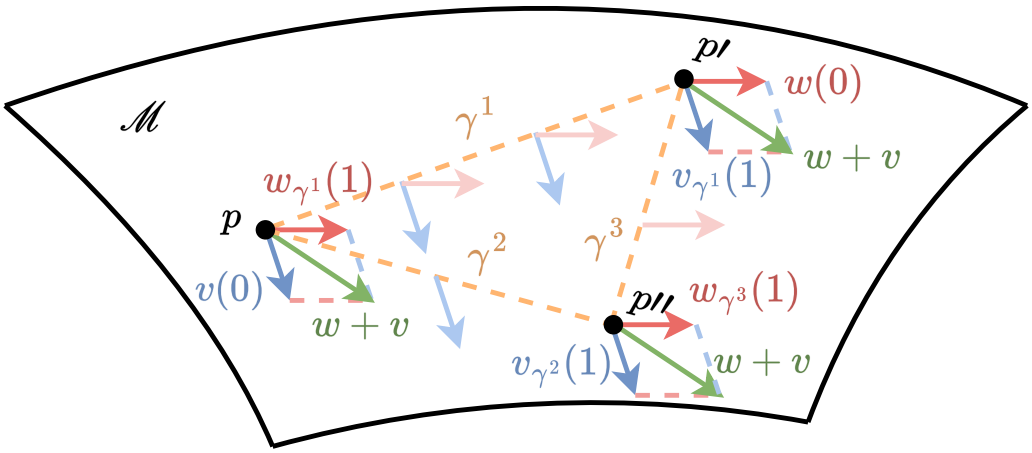


Figure 3.3: Adding two vectors on a flat manifold, is the same at every point of the manifold, as parallel transport does not depend on the curve, only on initial and ending points. With v defined at p and transported to p' added to w which is defined at p' and transported to p results in the same vector with the same orientation and size, at every point. Transporting both vectors v and w to a third point p'' and adding them results in the same vector as in p and p'.

3.5.2 α-Connections and Exponential Families

In addition to the Levi-Civita connection, we can define a whole family of connections, called α-connections (Amari and Nagaoka, 2000). This family of connections is in function of a parameter α, which is a real number, and for specific values of α we have specific connections defined: $\alpha = 0$ is the Levi-Civita connection, $\alpha = 1$ is the Exponential connection and $\alpha = -1$ is defined as the Mixture connection. All these connections are naturally defined and are distinct from one-another (see visual representation in Figure 3.4).

Let us introduce the notion of exponential families, which are statistical models that can be written in the form

$$p_\theta(x) = \exp\left[C(x) + \sum_{i=1}^{D} \theta^i f_i(x) - \psi(\theta)\right] ,$$

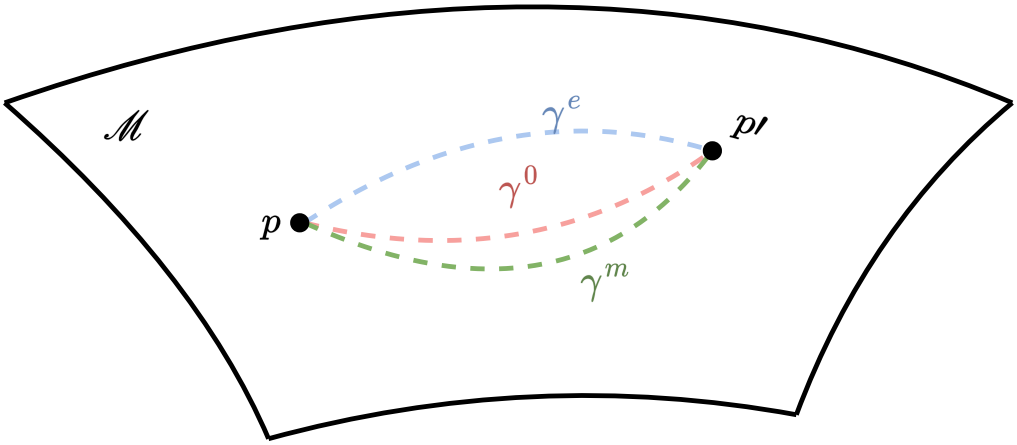


Figure 3.4: Geodesics for different alpha connections: red for $\alpha = 0 \rightarrow \gamma^0$ the Levi-Civita geodesic, blue for $\alpha = 1 \rightarrow \gamma^e$ the exponential geodesic and green for $\alpha = -1 \rightarrow \gamma^m$ the mixture geodesic.

where θ are the parameters of the distribution p; C and f_i are functions defined on input space $\mathcal{X}$ and ψ a function on Θ the parameter space. This family of distributions includes the most common distributions: Bernoulli, Normal, Poisson, Binomial etc. (DasGupta, 2011).

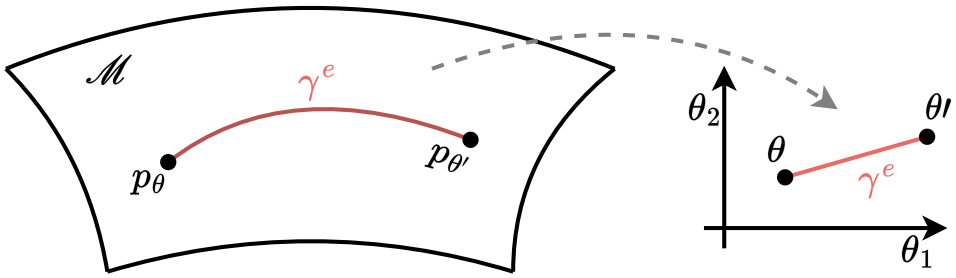


Figure 3.5: The e-flatness of a geodesic for distribution from an exponential family: a curve γ^e on the manifold $\mathcal{M}$ is a straight line in the coordinate system defined by θ_i-s.

Remark 2
An important property of exponential families is that when parametrized by θ they are flat in regards to the exponential connection, which is commonly referred to as e-flat (Amari and Nagaoka, 2000, Theorem 2.4). This flatness

implies that θ is a special coordinate system, in which the e-geodesic on the manifold corresponds to a straight line in the coordinates. Visualized in Figure 3.5.

Thanks to the flatness, transporting vectors along any curves works the same as in Euclidean space in a certain parametrization (based on the observation from Remark 1). From the above property it follows that we can easily move vectors between tangent spaces of the statistical manifold for the exponential family.

3.5.3 Exponential and Logarithmic Maps

When dealing with vectors and geodesics it is important to define two important mappings, the exponential map and the logarithmic map, which is described in more detail in the work of Absil et al. (2009).

Let $v \in T_p\mathcal{M}$ be a tangent vector to the manifold at a point p. For a given connection there is a unique geodesic $\gamma_v(t)$ satisfying $\gamma_v(0) = p$ with initial tangent vector $\gamma_v'(0) = \frac{\mathrm{d}}{\mathrm{d}t}\gamma(t) = v$. The exponential map $\mathrm{Exp}_p : T_p\mathcal{M} \to \mathcal{M}$ is defined as

$$\mathrm{Exp}_p(v) = \gamma_v(1) = p' \ .$$

The exponential map thus gives us the endpoint p' of a geodesic associated to a vector v. The logarithmic map (log-map) $\mathrm{Log}_p : \mathcal{M} \to T_p\mathcal{M}$ is defined as the inverse operator to the exponential map, as it recovers the vector associated to the geodesic for a given point p'

$$\mathrm{Log}_p(p') = v \ .$$

In general, the exponential map is only locally defined, it only applies to a neighborhood p, i.e., it may not be defined for certain $v \in T_p\mathcal{M}$. For flat manifolds, because the geodesics connecting two points are straight lines, the corresponding vector given by the log-map is self-evident, i.e., it is $v = \theta' - \theta$

Note 7
We will use these notions when we talk about the geometrical concepts of Accelerated Gradientss (AGs) in Section 6.1.2. However, since connections and parallel transport only briefly appear in this work, we will not go into more details, and we point the reader to the works of Amari and Nagaoka (2000) and Amari (2016) for more information.

Chapter 4

Geometry of Helmholtz Machines

The HM is a probabilistic model with a specific architecture which can be described as two Directed Acyclic Graphs (DAG) (Ay, 2020; Cowell et al., 2007; Lauritzen, 1996). In this section, we study the geometry of the HM, through the lens of DAGs. Influenced by the previous work of Ay (2002), we exploit a general property of these kinds of models, determined by the underlying structure of DAGs, for the derivation of the FIM for SBNs and HMs. Notably the FIM for SBNs has a fine-grained block-diagonal structure which allows an efficient computation of the NG.

4.1 Gradients of Directed Acyclic Graphs

Consider a Directed Acyclic Graph (DAG), with N nodes divided into two sets, i.e., $N = \{x\} \cup \{h\}$, with x the observed random variables (points in the data set) and h the hidden random variables. Furthermore, we suppose that all variables are discrete and by definition, there are no cycles in the graph. The joint distribution of the nodes in the DAG can be written in the following way, as it is classically done for directed graphical models:

$$p_\theta(x, h) = \prod_{r \in N} K_r(x_r | x_{pa(r)}; \theta_r) \ , \tag{4.1}$$

where for each node $r \in N$, $x_{pa(r)}$ are the random variables corresponding to the set of parent nodes of the random variable x_r. K_r is a kernel function, which in our case is the probability density function (pdf) of x_r given its parents, having parameters θ_r. If $p_{\mathcal{D}}(x)$ is the true distribution underlying the observation in the dataset $\mathcal{D}$, our goal is to minimize $KL(p_{\mathcal{D}}(x)||p_\theta(x))$, w.r.t. the parameters θ, which is equivalent to maximizing $\sum_x p_{\mathcal{D}}(x) \ln p_\theta(x)$. Therefore, the function we are optimizing during the training of the model is $\mathcal{L}(\theta) = \sum_x p_{\mathcal{D}}(x) \ln p_\theta(x)$, where θ is the set of parameters θ_r of the kernels

K_r. Calculating the gradients for a parameter θ_r gives

$$\begin{aligned}
\frac{\partial \mathcal{L}(\theta)}{\partial \theta_r} &= \frac{\partial}{\partial \theta_r} \sum_x p_{\mathcal{D}}(x) \ln p_\theta(x) \\
&= \sum_x p_{\mathcal{D}}(x) \frac{\partial}{\partial \theta_r} \ln \sum_h p_\theta(x,h) \\
&= \sum_x p_{\mathcal{D}}(x) \frac{\partial}{\partial \theta_r} \ln \sum_h \prod_{s \in N} K_s\left(x_s | x_{\mathrm{pa(s)}}; \theta_s\right) \quad \text{(from Eq. (4.1))} \\
&= \sum_x p_{\mathcal{D}}(x) \frac{1}{p_\theta(x)} \sum_h \frac{\partial}{\partial \theta_r} \left(\prod_{s \in N} K_s\left(x_s | x_{\mathrm{pa(s)}}; \theta_s\right) \right) \\
&= \sum_x \frac{p_{\mathcal{D}}(x)}{p_\theta(x)} \sum_h \left(\prod_{s \in N} K_s\left(x_s | x_{\mathrm{pa(s)}}; \theta_s\right) \right) \frac{\partial}{\partial \theta_r} \ln K_r\left(x_r | x_{\mathrm{pa(r)}}; \theta_r\right) \\
&= \sum_{x,h} \frac{p_{\mathcal{D}}(x)}{p_\theta(x)} \prod_{s \in N} K_s\left(x_s | x_{\mathrm{pa(s)}}; \theta_s\right)) \frac{\partial}{\partial \theta_r} \ln K_r\left(x_r | x_{\mathrm{pa(r)}}; \theta_r\right) \\
&= \sum_{x,h} p_{\mathcal{D}}(x)\, p_\theta(h|x) \frac{\partial}{\partial \theta_r} \ln K_r\left(x_r | x_{\mathrm{pa(r)}}; \theta_r\right)
\end{aligned}$$

using $p^\star_\theta(x,h) = p_{\mathcal{D}}(x)\, p_\theta(h|x)$

$$\begin{aligned}
&= \sum_{x,h} p^\star_\theta(x,h) \frac{\partial}{\partial \theta_r} \ln K_r\left(x_r | x_{\mathrm{pa(r)}}; \theta_r\right) \\
&= \mathbb{E}_{p^*_\theta(x,h)} \left[\frac{\partial}{\partial \theta_r} \ln K_r\left(x_r | x_{\mathrm{pa(r)}}; \theta_r\right) \right]
\end{aligned}$$

Since the statement inside the expectation only depends on a specific x_r and its parents $x_{\mathrm{pa(r)}}$, we can simplify the above expectation to

$$\begin{aligned}
\frac{\partial \mathcal{L}(\theta)}{\partial \theta_r} &= \mathbb{E}_{p^*_\theta(x_r, x_{pa(r)})} \left[\frac{\partial}{\partial \theta_r} \ln K_r\left(x_r | x_{\mathrm{pa(r)}}; \theta_r\right) \right] \\
&= \sum_{x_r, x_{\mathrm{pa(r)}}} p^\star_\theta\left(x_r, x_{\mathrm{pa(r)}}\right) \frac{\partial}{\partial \theta_r} \ln K_r\left(x_r | x_{\mathrm{pa(r)}}; \theta_r\right) \ .
\end{aligned} \tag{4.2}$$

Therefore, computing the gradient of $\mathcal{L}(\theta)$ w.r.t. the variable θ_r is independent of the computation of the gradient w.r.t. the other variables θ_s with $s \neq r$. However, this still requires knowing the posterior $p_\theta(h|x)$, which is generally intractable in practice. Gibbs sampling and other Markov Chain Monte-Carlo methods are often computationally expensive in higher dimensions for estimating the posterior distribution. A different approach used in HM consists of introducing a second network to learn the posterior $p_\theta(h|x)$ by approximating it with a distribution $q_\phi(h|x)$.

4.2 The Fisher Information Matrix of Directed Graphical Models

In this section we show how the FIM can be rewritten in a convenient way in the case of SBNs, and in particular for HMs. The FIM for DAG models takes a simplified block-diagonal form thanks to the locality of the connection matrix, given by conditional independence among random variables. Let us formalize this result through the following theorem:

Theorem 1
Let $\mathcal{G}$ be a DAG model, the variables of which are grouped in layers such that each node from the i-th layer has parent nodes from the $(i-1)$-th layer only. The FIM associated to the joint probability distribution p that factorizes as the product of conditional distributions according to $\mathcal{G}$ has a block-diagonal structure, with one block for each hidden unit of size equal to the number of its parent nodes.

The FIM for the distribution $p_\theta(x,h)$ is defined as

$$\mathcal{F}(\theta) = \mathbb{E}_{p_\theta(x,h)}\left[\nabla_\theta \ln p_\theta(x,h)\, \nabla_\theta^\top \ln p_\theta(x,h)\right] ,$$

where $\nabla \ln p_\theta$ is a column vector. Alternatively, it is easy to show that the FIM can also be written as

$$\mathcal{F}(\theta) = -\mathbb{E}_{p_\theta(x,h)}\left[\nabla_\theta^2 \ln p_\theta(x,h)\right] ,$$

where by $\nabla_\theta^2 \ln p_\theta(x,h)$ we understand the Hessian of $\ln p_\theta(x,h)$.

For each $r \in N$, θ_r is the vector of variables denoting the parameters of the distribution of x_r given its parents. From the Eq. (4.2) it follows that

$$\nabla_{\theta_r}\nabla_{\theta_s} \ln p_\theta(x,h) = \delta_{rs}\nabla_{\theta_r}^2 \ln K_r\left(x_r|x_{\mathrm{pa(r)}};\theta_r\right) ,$$

with $\delta_{rs} = 1$, if $r = s$ and 0, otherwise. This shows that the Fisher information matrix has a block-diagonal structure

$$\mathcal{F}(\theta) = -\mathbb{E}_{p_\theta(x,h)}\left[\mathrm{diag}\left(\nabla_{\theta_r}^2 \ln K_r\left(x_r|x_{\mathrm{pa(r)}};\theta_r\right)\right)_{r\in N}\right] , \tag{4.3}$$

and thus, in order to compute the FIM $\mathcal{F}(\theta)$, it is sufficient to compute $\nabla_{\theta_r}^2 \ln K_r\left(x_r|x_{\mathrm{pa(r)}};\theta_r\right)$ for every θ_r.

□

The HM is a DAG and as such the above structure applies to both of its networks. Using our previous notation from Chapter 2, introduced for HMs, with the parameters of p being $\theta_i = W^i$, the formula becomes

$$\mathcal{F}_p^{i,j} = -\mathbb{E}_{p(x,h)}\left[\frac{\partial^2}{\partial W^{i,j}\,(\partial W^{i,j})^\top} \ln p\left(h^{i,j}|h^{i+1}\right)\right] \tag{4.4}$$

which applies similarly to the recognition distribution q as well.

This shows that models such as SBNs, which have a topological structure that can be described as layered DAGs, have an associated FIM with a finer-grained block structure, consisting of **one block per neuron**, with size equal to the cardinality of the previous layer.

Sun and Nielsen (2017) have demonstrated that the structure of the FIM given some architectures of neural networks can be simplified without approximation, compared to a fully dense matrix, to a block-diagonal structure, where the blocks correspond to the sizes of layers. Our work goes beyond their result and provides a crucial refinement of the previous works.

While block-diagonal structures as approximations for FIMs have been used before (Desjardins et al., 2015), (Grosse and Martens, 2016), (Sun and Nielsen, 2017) or (Bahamou et al., 2022), to the best of our knowledge, at the writing of this thesis, this is the first application to algorithm design of such a fine-grained block-diagonal structure for the FIM. Most importantly, the structure is not derived from further approximations of the FIM but instead is a consequence of the topology of the network itself.

The proof of this result is based on a generalization of Theorem 1 from the work of Ay (2020), see also Lemma 1 in Ay (2002), where the locality of NG is studied from a theoretical perspective.

The results presented in this section, which are key results from (Várady et al., 2022), indeed prove that without the need of further approximations the FIM for SBNs is block-diagonal with blocks of smaller size compared to previous results from the literature, typically having one block per layer, e.g., (Desjardins et al., 2015; Sun and Nielsen, 2017). This has significant advantages from a computation perspective. A graphical representation of the difference in the block-sizes between our methods and the ones from the literature can be seen in Figure 4.1.

4.3 The Fisher Information Matrix of Sigmoid Belief Networks

We rewrite Eq. (4.4) for a more compact representation, assuming our model uses a Bernoulli distribution for $K_r\left(x_r|x_{\mathrm{pa(r)}};\theta_r\right)$, as commonly happens in

SBNs and by extension HMs. With mean parameters given by the sigmoid of a linear function of $x_{pa(r)}$, we have that

$$K_r\left(x_r|x_{\mathrm{pa(r)}};\theta_r\right) = s\left(\left(2x_r-1\right)\theta_r^\top x_{\mathrm{pa(r)}}\right).$$

Then the first and second derivatives are

$$\frac{\partial}{\partial\theta_{ri}}\ln K_r\left(x_r|x_{\mathrm{pa(r)}};\theta_r\right) = \left(2x_r-1\right)x_{\mathrm{pa(r);}i}\left[1-s\left(\left(2x_r-1\right)\theta_r^\top x_{\mathrm{pa(r)}}\right)\right]$$

and

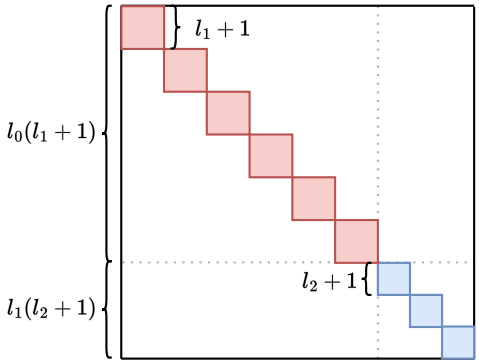
$$\begin{aligned}
&\frac{\partial^2}{\partial\theta_{ri}\partial\theta_{rj}}\ln K_r\left(x_r|x_{\mathrm{pa(r)}};\theta_r\right) = \\
&= \frac{\partial}{\partial\theta_{rj}}\left(2x_r-1\right)x_{\mathrm{pa(r);}i}\left[1-s\left(\left(2x_r-1\right)\theta_r^\top x_{\mathrm{pa(r)}}\right)\right] \\
&= -\left(2x_r-1\right)^2 x_{\mathrm{pa(r);}i}x_{\mathrm{pa(r);}j}s\left(\left(2x_r-1\right)\theta_r^\top x_{\mathrm{pa(r)}}\right)\left[1-s\left(\left(2x_r-1\right)\theta_r^\top x_{\mathrm{pa(r)}}\right)\right] .
\end{aligned}$$

Therefore, the double derivative written using vector notation is

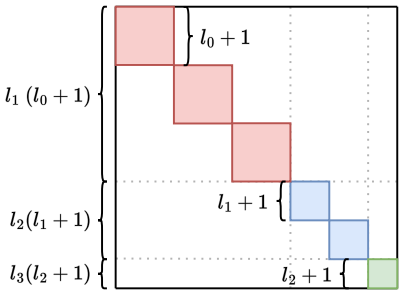
$$\nabla^2_{\theta_r}\ln K_r\left(x_r|x_{\mathrm{pa(r)}};\theta_r\right) = -s'\left(\left(2x_r-1\right)\theta_r^\top x_{\mathrm{pa(r)}}\right)x_{\mathrm{pa(r)}}x_{\mathrm{pa(r)}}^\top , \qquad (4.5)$$

where $s' = s(1-s)$ is the derivative of the sigmoid function s. This leads to a block structure for the FIM $\mathcal{F}$ with respect to the weights θ_r of the distribution p and similarly for ϕ_r of the distribution q.

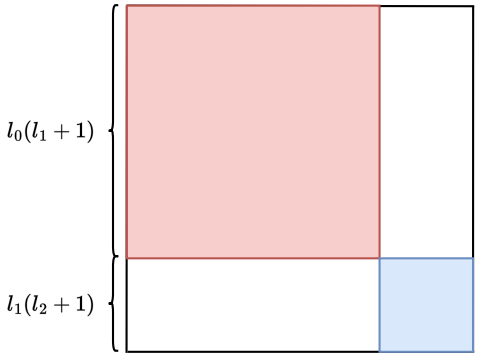
Furthermore, such a fine-grained block-diagonal structure is not only true for binary SBNs, but as we have seen in Eq.(4.2) it generalizes to all DAG models, a result we will exploit in Section 7.1 to learn continuous distributions with HMs.



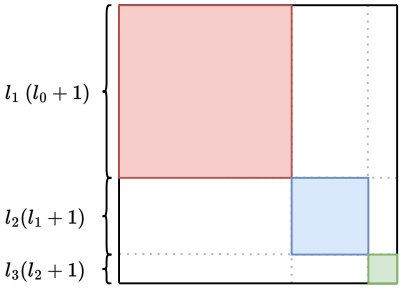
(a) Our structure for the FIM of the Recognition network



(b) Our structure for the FIM of the Generation network



(c) The structure of the Recognition network FIM used in the literature*



(d) The structure of the Generation network FIM used in the literature*

Figure 4.1: Graphical representation of the FIM for a Network with 3 layers with respectfully 6, 3 and 2 nodes. The gray dotted lines identify the blocks associated to the layers of the network. The matrix admits a fine-grained block-diagonal structure with blocks of size equal to the size of the hidden layers. The blocks are ordered in both cases from the bottom layer to the top starting from the upper-left. *The comparison with the literature are typically block-diagonal approximations based on layer sizes from the works of (Desjardins et al., 2015; Sun and Nielsen, 2017).

Chapter 5

Natural Reweighted Wake-Sleep

In the first part of this chapter, we show how one can use Monte-Carlo sampling to estimate the FIM then compute its inverse. By applying results from the previous chapter we derive an efficient formula for the computation of the NG for HM.

We introduce the Natural Reweighted Wake-Sleep (NRWS) algorithm for training HMs. We elaborate on some implementation details and hyperparameters of the model. Finally, we compare experimentally the NRWS with other training algorithms for the HM, and show how we achieve state-of-the-art results for a fixed budget both on training epochs, and real-world time.

In the next part we show how the FIM used for the HM can be used not only for the NRWS but also for other training algorithms, such as the BiHM (Bornschein et al., 2016), which allows us define the Natural Bidirectional Helmholtz Machine (NBiHM). Although there are some theoretical limitations of this new model, we show how the new algorithm can still profit from the FIM of the HM and reach better performance with it.

Finally we analyze the convergence properties of the NRWS through the work of (Ikeda et al., 1999), and we suggest an alternate approximate solution, the Rescaled-sleep (RS), to the convergence problem raised by the authors for which they define the Sleep-well (SW). We compare the two solutions in some experiments and find that both can improve over the standard model.

5.1 The Natural Gradient for Reweighted Wake-Sleep

The definition of the FIM applied to the generation network is

$$\mathcal{F}_p(\theta) = -\mathbb{E}_{p_\theta(x,h)}[\nabla^2_\theta \ln p_\theta(x,h)] \ .$$

If we treat the HM as a DAG model as in Chapter 4, where $N = \{x, h\}$, M is the total number of layers, with l_i the length of each individual layer $i \in [0..M]$ we can write the joint distribution as:

$$p_\theta(x,h) = \prod_{i=0}^{M} \prod_{j=1}^{l_i} p(h^{i,j}|h^{i+1};\theta_{ij}) \ ,$$

and the log probabilities as

$$\ln p_\theta(x,h) = \sum_{i=0}^{M}\sum_{j=1}^{l_i} \ln p(h^{i,j}|h^{i+1};\theta_{ij}) \ .$$

It is easy to see that after applying the Hessian ∇^2, similarly to the derivation in the previous chapter we get

$$\nabla_{\theta_{ij}}\nabla_{\theta_{lk}} \ln p_\theta\,(x,h) = \delta_{il}\delta_{jk}\nabla^2_{\theta_{ij}} \ln p(h^{i,j}|h^{i+1};\theta_{ij}) \ .$$

The FIM is block-diagonal following Theorem 1, with the structure shown in Figure 4.1. From here, by looking at Eq. (4.4) it can easily be calculated

$$\nabla^2_{\theta_j} \ln p_\theta(h^{i,j}|h^{i+1}) = -p_\theta(h^{i,j}|h^{i+1})[1-p_\theta(h^{i,j}|h^{i+1})]h^{i+1}\left(h^{i+1}\right)^\top \ , \quad (5.1)$$

where each block of the FIM is

$$\begin{aligned}\mathcal{F}_p^{i,j} &= -\mathbb{E}_{p(x,h)}\left[\frac{\partial^2}{\partial W^{i,j}\left(\partial W^{i,j}\right)^\top}\ln p(h^{i,j}|h^{i+1})\right] \\ &= \mathbb{E}_{p(x,h)}\left[s'\left(\left(W^{i,j}\right)^\top h^{i+1}\right)h^{i+1}\left(h^{i+1}\right)^\top\right] \ ,\end{aligned} \quad (5.2)$$

where h^{i+1} is a column vector of the previous layers' samples and $W^{i,j}$ is the j-th row vector of the weights of the p network's i-th layer. We stress the fact that the FIM has a different block $\mathcal{F}_p^{i,j}$ for each neuron j from each layer i. Equivalently for the recognition network

$$\begin{aligned}\mathcal{F}_q^{i,j} &= -\mathbb{E}_{q(h|x)}\left[\frac{\partial^2}{\partial V^{i,j}\left(\partial V^{i,j}\right)^\top}\ln q(h^{i,j}|h^{i-1})\right] \\ &= \mathbb{E}_{q(h|x)}\left[s'\left(\left(V^{i,j}\right)^\top h^{i-1}\right)h^{i-1}\left(h^{i-1}\right)^\top\right] \ ,\end{aligned} \quad (5.3)$$

in which case the FIM of q is block-diagonal, with the largest blocks being of the size of the input $x \times x$. This will become more important later, where the complexity of the algorithm is discussed in Subsection 5.3.3.

In the case of binary values $\{\pm 1\}$ instead of $\{0,1\}$, we have

$$p(h^{i,j}|h^{i+1}) = s\left(h^{i,j}\ \left(\left(W^{i,j}\right)^\top h^{i+1}\right)\right) \ , \quad (5.4)$$

while the formula for the the FIM is the same as in Eq. (4.4).

In the light of Chapter 2 we know RWS works very well on HMs. Given the above equations, knowing the structure of the FIMs of both networks, we now have a way to compute a gradient step based on NG for layers of the HM. We can use the formulas from Eqs. (5.2) and (5.3) to apply the NG update to the three phases of the Reweighted Wake-Sleep:

- wake phase: precondition the gradients of the wake phase w.r.t θ, $\nabla_\theta \mathcal{L}_p(x,h)$, with the inverse FIM of the generation distribution p:

$$\widetilde{\nabla}_\theta \mathcal{L}_p(x,h) = \mathcal{F}_p^{-1} \nabla_\theta \mathcal{L}_p(x,h) \ ;$$

- q-wake update: precondition the gradients of the q-wake phase w.r.t ϕ, $\nabla_\phi \mathcal{L}_q^w(x,h)$, with the inverse FIM of the recognition distribution q:

$$\widetilde{\nabla}_\phi \mathcal{L}_q^w(x,h) = \mathcal{F}_q^{-1} \nabla_\phi \mathcal{L}_q^w(x,h) \ ;$$

- sleep phase: precondition the gradients of the sleep phase w.r.t ϕ, $\nabla_\phi \mathcal{L}_q^s(x,h)$, with the inverse FIM of the recognition distribution q:

$$\widetilde{\nabla}_\phi \mathcal{L}_q^s(x,h) = \mathcal{F}_q^{-1} \nabla_\phi \mathcal{L}_q^s(x,h) \ .$$

We will refer to this new algorithm, where we use the NG on the RWS, as Natural Reweighted Wake-Sleep (NRWS). In the following subsections, we will discuss the implementation and algorithmic details of the NRWS that enable it to be an efficient method for the training of HMs.

Remark 3
The FIMs in Eqs. (5.2) *and* (5.3) *only depend on the statistical models associated to the joint distributions $p(x,h)$ and $q(x,h)$, and in particular they are independent from the specific loss function $\mathcal{L}$ defined over the model, as well as from the chosen training algorithm. Hence, since the model of the HM remains unchanged, the same FIMs can be used for different training algorithms, such as WS and RWS.*

Remark 4
There is a good argument to be made (Ay, 2020) that the reference distribution of FIM should be the visible distribution $p(x)$ instead of the joint $p(x,h)$ that we are using. However it has been suggested (Ollivier et al., 2017) that the FIM profits from being computed w.r.t. $p(x,h)$ when it is empirically estimated, as it is numerically more stable and requires fewer examples to estimate, compared to computing with $p(x)$. Derivation of the FIM w.r.t. the visible distribution $p(x)$ however presents additional complications, which do not make its empirical estimation computationally feasible, and is a subject to be explored in future works.

5.2 Estimation of the Fisher Information Matrix

Notice that h^i is a stochastic quantity which can be sampled based on the value of the nodes from the previous layer, because of the conditional independence. We can use this to compute a Monte Carlo estimate of each block of $\mathcal{F}$ with n samples, Eqs. (5.2) and (5.3) can be estimated as

$$\begin{aligned} \mathcal{F}_p^{i,j} &= \mathbb{E}_{p(x,h)}\left[s^{'}\left(\left(W^{i,j}\right)^\top h^{i+1}\right) h^{i+1}\left(h^{i+1}\right)^\top \right] \\ F_p^{i,j} &\simeq \frac{1}{n}\sum s^{'}\left(\left(W^{i,j}\right)^\top h^{i+1}\right) h^{i+1}\left(h^{i+1}\right)^\top \\ &= U^{i+1} Q_p^{i,j}\left(U^{i+1}\right)^\top \text{ with } \; h^{i+1} \sim p(h^{i+1}|h^{i+2}) \end{aligned} \tag{5.5}$$

and

$$\begin{aligned} \mathcal{F}_q^{i,j} &= \mathbb{E}_{q(h|x)p_{\mathcal{D}}(x)}\left[s^{'}\left(\left(V^{i,j}\right)^\top h^{i-1}\right) h^{i-1}\left(h^{i-1}\right)^\top \right] \\ F_q^{i,j} &\simeq \frac{1}{n}\sum s^{'}\left(\left(V^{i,j}\right)^\top h^{i-1}\right) h^{i-1}\left(h^{i-1}\right)^\top \\ &= U^{i-1} Q_q^{i,j}\left(U^{i-1}\right)^\top \text{ with } \; h^{i-1} \sim q(h^{i-1}|h^{i-2}) \; . \end{aligned} \tag{5.6}$$

Eqs. (5.5)-(5.6) represent the blocks $F_p^{i,j}$ and $F_q^{i,j}$of the empirical FIMs, for $W^{i,j}$ and for $V^{i,j}$, respectively, which are obtained by sampling h^{i+1} and h^{i-1} n times from the previous layers. In the last step, we introduced a matrix representation for the empirical estimation F of $\mathcal{F}$, where we obtain the U^i matrices by taking the r-th sample $h^{i,r}$ as column vector r of the matrix U^i, while the diagonal matrices $Q_p^{i,j}$ and $Q_q^{i,j}$ are diagonal matrices based on the evaluation of the activation functions, i.e., $Q_p^{i,j} = \text{diag}\left[s^{'}\left((W^{i,j})^\top h^{i+1}\right)\right]$. A visual representation can be seen in Figure 5.1.

Furthermore, we can reduce the variance of the estimation from Eq. (5.5), by taking inspiration from RWS (Section 2.3.1) and its use of importance sampling Eq. (2.25). By using reweighting for the samples in Eq (5.5), we can get a lower variance estimation using the same amount of samples as for the Monte-Carlo variant. Note that we can reuse the exact same weights as the ones that the RWS uses, with no additional computation. Thus, to obtain a lower variance estimation for the expected value using samples from the distribution $q(h|x)p_{\mathcal{D}}(x)$, we reweigh them by importance sampling with

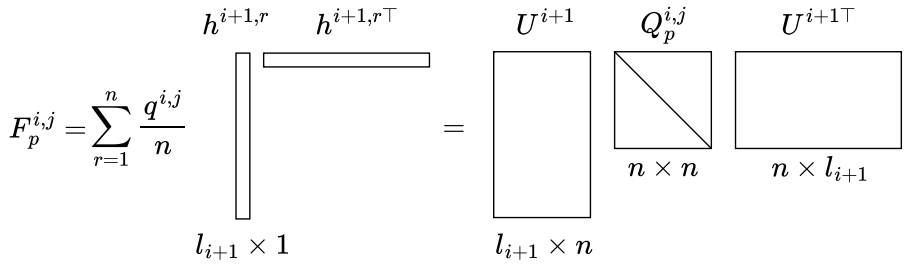


Figure 5.1: FIM represented in matrix notation as a matrix product.

the same weights $\tilde{\omega}_k$ as in Eq. (2.28) and we get

$$\begin{aligned} F_p^{i,j} &= \sum_{k=1}^{n} \tilde{\omega}_k s' \left(\left(W^{i,j} \right)^\top h_{(k)}^{i+1} \right) h_{(k)}^{i+1} \left(h_{(k)}^{i+1} \right)^\top \\ &= U^{i+1} \tilde{Q}_p^{i,j} \left(U^{i+1} \right)^\top \text{with } \; h_{(k)}^{i+1} \sim q(h^{i+1}|h^i) \; , \end{aligned} \tag{5.7}$$

where $\tilde{Q}_p^{i,j} = \text{diag}\left[\tilde{\omega}_k s' \left((W^{i,j})^\top h_{(k)}^{i+1} \right) \right]$.

We only use this approach for the FIM of the generation distribution F_p. The reason for reweighting F_p is because we prefer samples coming from the real-world distribution $p_{\mathcal{D}}$ to ones that come from the HM model since it is the target distribution that we try to approximate. Applying the same reweighting method to the F_q might counterintuitively bias the estimate from samples that come from $p_{\mathcal{D}}$ to samples coming from p which is a probability distribution depending on parameters that are still in the process of being learned.

5.3 Natural Reweighted Wake-Sleep

In this section, we introduce the Natural Reweighted Wake-Sleep (NRWS), a geometric adaptation of the RWS (Bornschein and Bengio, 2015), where the update of the weights is obtained through the computation of the NG of the different loss functions in the wake and sleep phases.

The update rules of NRWS at step t for the weights $\theta = \left(W^0, \ldots, W^M \right)$ and $\phi = \left(V^1, \ldots, V^M \right)$ of the generation and recognition networks are given

by

$$\begin{aligned}\theta_{t+1} &= \theta_t - \eta \widetilde{F}_p^{-1} \frac{1}{B} \sum_{r=1}^{B} \sum_{k=1}^{S} \tilde{\omega}_k \nabla L_{p,(k,r)} \text{ and} \\ \phi_{t+1} &= \phi_t - \frac{\eta}{2} \widetilde{F}_q^{-1} \frac{1}{B} \sum_{r=1}^{B} \sum_{k=1}^{S} \left(\frac{1}{S} \nabla L^s_{q,(k,r)} + \tilde{\omega}_k \nabla L^{w,(k,r)}_q \right) ,\end{aligned} \tag{5.8}$$

where L_p, L^s_q and L^w_q are the empirical estimates of the loss functions $\mathcal{L}_p$, $\mathcal{L}^s_q$ and $\mathcal{L}^w_q$ for the wake, sleep and q-wake phases, respectively. Such empirical estimates $L_{(k,r)}$ are computed with minibatches of size B indexed by r, sampled each S times with indices k. Notice that in accordance with the implementation of the RWS algorithm, the learning rate in the updating rule for ϕ is halved to average the two gradients. Furthermore, the empirical FIMs $\widetilde{F}_p^{-1}$ and $\widetilde{F}_q^{-1}$ are also estimated with B and S, based on Eqs. (5.6) and (5.7), where $n = B * S$.

The pseudo-code for NRWS is presented in Alg. 4. In the following subsections, we address some of the implementation details of the NRWS algorithm.

5.3.1 Solving systems with the Fisher Information Matrix

In this subsection, for simplicity, we will only discuss the case of the generation network since everything is analogously true for the recognition network as well.

The computation of the NG update requires the inversion of the FIM, which is not a trivial operation from a computational complexity perspective[1]. It is well known that the inversion of an $m \times m$ matrix takes $\mathcal{O}(m^{\omega+o(1)})$, with $2 \leq \omega \leq 3$, by using some variant of the Cayley–Hamilton theorem (Decell, 1965; Williams, 2012)[2].

The FIM has the size $D \times D$, where D is the total size of all the parameters of HM. The size of the parameters in our case is the sizes of the weights and biases in each dense layer in the HM: $D = \sum_{i=1}^{M} D_i = \sum_{i=1}^{M} (l_i + 1) l_{i-1}$ for the generation network, where D_i. This sum can easily be in the millions, even

[1]Where we refer here as the inversion of a matrix, in the language of mathematics, in practice in computational science, we are actually using the method of solving a linear system, as there is no need to compute a full inverse of the FIM. Since the inverse is multiplied with the gradient of the loss, which is a vector, $\mathcal{F}^{-1} \cdot \nabla \mathcal{L}$, their product can be viewed as the solution to a linear system $\mathcal{F} \cdot X = \nabla \mathcal{L}$. From a mathematical perspective, all the following derivations are valid and useful, this comment only serves to clarify that we are using the computationally more efficient formulation in practice.

[2]Currently the smallest know value is $\omega \simeq 2.371552$ (Williams et al., 2024).

Algorithm 4: Natural Reweighted Wake-Sleep

Let x be a minibatch of samples from the dataset
Let p and q be the distributions of the generation and the recognition networks with weights W and V
Let ω be the importance weights from the RWS
Let M be the depth of the HM
`#wake phase update`
for *each layer i from q ascending with $h^0 = x$* **do**
    Compute the gradients $\nabla_{\theta^i} L_p$ with respect to W^i as in (2.27)
    Compute the weights $\tilde{\omega}$ from the multiple samples (2.28)
    Compute the matrices for $\widetilde{F}_p^i$ for the sub-blocks in i with h^{i+1} and $p(h^i|h^{i+1})$ as in (5.5)
    Compute $\widetilde{\nabla}_p^i L_p = (\widetilde{F}_p^i)^{-1} \nabla_p^i L_p$
    `#q-wake update`
    Compute the gradients $\nabla_{\phi^i} L_q^w$ with respect to V^i similarly to the `sleep` phase as in (2.30)
    Compute the matrices $\widetilde{F}_q^i$ for the sub-blocks in i with h^{i-1} and $q(h^i|h^{i-1})$ as in (5.6)
    Compute $\widetilde{\nabla}_q^i L_q^w = (\widetilde{F}_q^i)^{-1} \nabla_q^i L_q^w$
    Update W^i and V^i using η with the $\widetilde{\nabla}_p^i L_p$ and $\widetilde{\nabla}_q^i L_q^w$ with weights $\tilde{\omega}$ as in (5.8)
`#sleep phase update`
for *each layer i from p descending with h^M sampled from the prior* **do**
    Compute the gradients $\nabla_{\phi^i} L_q^s$ with respect to V^i as in (2.31)
    Reuse the matrix $\widetilde{F}_q^i$ calculated in the `q-wake` phase
    Compute $\widetilde{\nabla}_q^i L_q^s = (\widetilde{F}_q^i)^{-1} \nabla_q^i L_q^s$
    Update V^i using η with $\widetilde{\nabla}_q^i L_q^s$ as in (5.8)

for relatively small networks, which prevents the calculation of the inverse due to the computation complexity.

This is where the usefulness of a block-diagonal structure comes in, because the computational complexity of inverting a block-diagonal matrix equals that of inverting its largest block since all blocks can be inverted independently of the others and in parallel. In our case, the dimension of the largest block is the product of the last two layers of the network, which already brings down the complexity from $\mathcal{O}(D^\omega)$ to $\mathcal{O}\left((l_0 l_1)^\omega\right)$.

However, we can do even better by using the fact that the empirical FIM is a low-rank estimation of the true one. Recall the matrix product formulation of the FIM from the previous section and in Figure 5.1. If the number of samples $n = S \cdot B$ used for the estimation is less than D, which in practice is almost always true, then F is a low-rank estimation of the true FIM. This implies also that the empirical FIM is not even invertible, because F is singular, and thus a perturbation of it is needed to make it invertible.

In our case, however, because of the block-diagonal nature of the FIM, the condition is that the number of samples has to be smaller than the largest block of the largest layer $n < D_i$, typically $l_0 l_1$. This is still sometimes the case, as the largest layer usually grows with the size of the data. While we can manipulate the size of n, we have some limitations as smaller B-s leads to better fitting of the data, using the MBGD, and helps avoid overfitting, and larger S-s lead to diminishing returns in estimation accuracy, thus the FIM remains singular. Our only remaining option is the perturbation of the singular matrix $\mathcal{F}$ using Tikhonov Regularization.

5.3.2 Renormalized Tikhonov Regularization

In order to approximate F by an invertible matrix, it is a common technique to add a Tikhonov regularization term in the form of a damping factor αI_n, where $\alpha \in [0, 1]$ and I_n is the identity matrix of size n. Our regularized estimate of the inverse FIM for a network with a single layer becomes

$$\begin{aligned} \widetilde{F}^{-1} &= (\alpha I_n + F)^{-1} \\ &= (\alpha I_n + UQU^\top)^{-1} \, . \end{aligned} \tag{5.9}$$

We would prefer, that when $\alpha \to \infty$ then $\widetilde{F}^{-1} \to I_n$ and when $\alpha \to 0$ that $\widetilde{F}^{-1} \to F^{-1}$. However

$$\begin{aligned} \lim_{\alpha \to \infty} \widetilde{F}^{-1} &= \lim_{\alpha \to \infty} (\alpha I_n + UQU^\top)^{-1} = 0 \, , \\ \lim_{\alpha \to 0} \widetilde{F}^{-1} &= \lim_{\alpha \to 0} (\alpha I_n + UQU^\top)^{-1} = F^{-1} \, . \end{aligned}$$

To solve this problem, we can use the term $1/(1+\alpha)$ to renormalize the inversion

$$\widetilde{F} = \frac{\alpha I_n + F}{1+\alpha} = \frac{\alpha I_n + UQU^\top}{1+\alpha} \; . \tag{5.10}$$

This renormalization term gives us the right limits for the inversion:

$$\lim_{\alpha \to \infty} \widetilde{F}^{-1} = \lim_{\alpha \to \infty} (1+\alpha)(\alpha I_n + UQU^\top)^{-1} = I_n \; ,$$

and

$$\lim_{\alpha \to 0} \widetilde{F}^{-1} = \lim_{\alpha \to 0} (1+\alpha)(\alpha I_n + UQU^\top)^{-1} = F^{-1} \; .$$

5.3.3 Low-rank Inversion of the Fisher Information Matrix

Since $\widetilde{F}$ is a symmetric and positive-definite matrix, for large enough α, we can use the Sherman-Morisson-Woodbury formula (Woodbury, 1950), which is the most efficient way to do a low-rank matrix inversion, or more precisely: a low-rank update of an invertible matrix. While there are multiple equally valid versions of this formula, which all have different advantages, we settled on using the original Woodbury formulation.

The formula introduced by Woodbury states that for matrices Z is $n \times n$, C is $k \times k$, U is $n \times k$, and V is $k \times n$, with assumed $n > k$, the following holds true:

$$(Z + UCV)^{-1} = Z^{-1} - Z^{-1}U(C^{-1} + VZ^{-1}U)^{-1}VZ^{-1} \; .$$

If we take $Z = \alpha I_n$, $U = U$, $C = Q$ and $V = U^\top$, we have:

$$(\alpha I_n + UQU^\top)^{-1} = \frac{1}{\alpha} I_n - \frac{1}{\alpha^2} U(Q^{-1} + \frac{1}{\alpha} U^\top U)^{-1} U^\top \; .$$

We took two main features into consideration when choosing the method: the overall complexity of the operations needed to execute the formulation, with our matrices, and how much memory is needed to keep in memory the partial matrices. This last condition/requirement will become evident in Section 5.3.4.

There are two advantages to this version: first, inverting Q is in linear time because it is a diagonal matrix, so inverting it is just taking the reciprocal of the diagonal. Second, we only have to keep in memory two matrices after the first use of the formula U, which is $n \times k$, and the inverse of the inner parenthesis matrix $k \times k$ (more details about this in Section 5.3.4).

Note 8
Using $1/\alpha^2$ computationally might lead to numerical instability for $\alpha \ll 1$, therefore we reformulate the equations.

For the Sherman-Morrison-Woodbury formula, it is possible to bring $1/\alpha$ out of the inner inversion, to get rid of $1/\alpha^2$.

$$\begin{aligned}
(\alpha I_n + UQU^\top)^{-1} &= \frac{1}{\alpha} I_n - \frac{1}{\alpha^2} U(Q^{-1} + \frac{1}{\alpha} U^\top U)^{-1} U^\top \\
&= \frac{1}{\alpha} I_n - \frac{1}{\alpha^2} U(\frac{\alpha}{\alpha} Q^{-1} + \frac{1}{\alpha} U^\top U)^{-1} U^\top \\
&= \frac{1}{\alpha} I_n - \frac{1}{\alpha^2} U \frac{1}{\alpha}^{-1} (\alpha Q^{-1} + U^\top U)^{-1} U^\top \\
&= \frac{1}{\alpha} I_n - \frac{1}{\alpha} U(\alpha Q^{-1} + U^\top U)^{-1} U^\top \\
&= \frac{1}{\alpha} \Big[I_n - U(\alpha Q^{-1} + U^\top U)^{-1} U^\top \Big] .
\end{aligned}$$

5.3.4 K-step Update

By moving with a small step in the direction of the gradient descent, we can make the assumption that the metric is changing slowly during a few training steps. Under this assumption, we can choose to reuse the FIM for a certain amount of steps K before recalculating it from the mini-batch. We will call this technique the K-step update (a similar approach was used in (Martens and Grosse, 2015)).

Notice that it is not very efficient to save in memory the whole inverse $\left(F_p^{i,j}\right)^{-1}$ of each block of the FIM, as it would take a memory requirement of $\mathcal{O}\left(l_i^2\right)$. Indeed, even when using the low-rank inversion of the diagonal matrices for which we used the Sherman-Morisson-Woodbury technique, the NG update step would have a computational complexity of $\mathcal{O}\left(l_i^2\right)$.

We can do better than that in both memory requirement and computational complexity, when using the low-rank technique, by saving only the U^{i+1} and the $\left(A_p^{i,j}\right)^{-1}$ partial matrices of the formula

$$\begin{aligned}
\left(\widetilde{F}_p^{i,j}\right)^{-1} &= \left(\alpha I_n + U^{i+1} Q_p^{i,j} \left(U^{i+1}\right)^\top\right)^{-1} \\
&= \frac{1}{\alpha}\left[I_n - U^{i+1} \Big(\underbrace{\alpha \left(Q_p^{i,j}\right)^{-1} + \left(U^{i+1}\right)^\top U^{i+1}}_{A_p^{i,j}} \Big)^{-1} \left(U^{i+1}\right)^\top \right] ,
\end{aligned}$$

which we can use in the computation of $\widetilde{F}^{-1} \cdot \nabla L$ when the NG update is computed.

Saving the matrices U^{i+1} and the inverse of the inner matrix $A_p^{i,j}$, we have a memory requirement of $\mathcal{O}(l_i n)$ and $\mathcal{O}(n^2)$ respectively, which is much smaller in comparison to keeping the whole inverse $\mathcal{O}(l_i^2)$.

When reusing the previously computed FIM blocks, the computational complexity for each update is dominated by the multiplication of the inverse FIM with the vanilla gradient, and it becomes $\mathcal{O}(n^\omega + l_i n)$ every K-th step when the inverse is calculated and $\mathcal{O}(n^2 + l_i n)$ every in-between step.

The result of using the K-step update is that we obtain a speed up every time we are reusing the partial matrices, by trading in memory space, since we need to save U^{i+1} and $\left(A_p^{i,j}\right)^{-1}$ among iterations.

5.3.5 Implementation Details

A final aspect related to the implementation is the technology we are using to program/run the algorithm, and the side-effects arising from that choice. Our decision fell on CUDA-optimized Tensorflow (Abadi et al., 2015)[3] since we can take advantage of the large level of parallelization built into the framework. This means that we can parallelize the calculation of the inversion of the FIM over the nodes of a layer, so we can calculate F_p^i by calculating all $F_p^{i,j}$ in parallel, and by extension the complexity of the algorithm $\mathcal{O}(l_0(l_1 n + n^2))$ becomes in practice close to $\mathcal{O}(l_1 n + n^2)$ (if we suppose that complexity is dominated by the sizes of the bottom two layers). In doing so we use 3 dimensional tensors for the FIM matrices F, which is visualized in Figure 5.2.

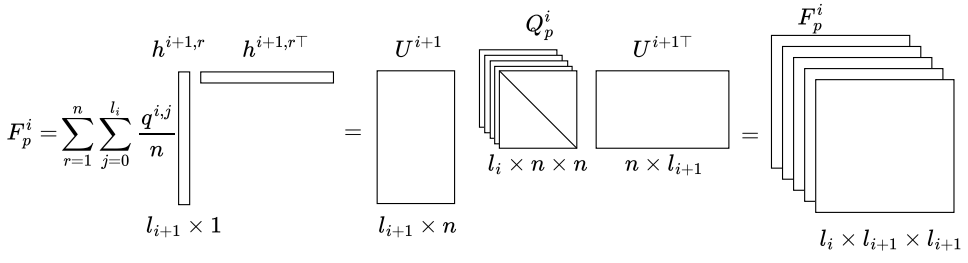


Figure 5.2: FIM defined as tensor products, notice that Q^i is a made up of diagonal matrices.

[3]More details about the technology in the Appendix A.2

The final form of calculating the inverse of a series of blocks corresponding to layer i of the FIM thus is given by

$$\left(\tilde{F}_p^i\right)^{-1} = \left(\frac{\alpha I_{l_i} + U^{i+1} Q_p^i \left(U^{i+1}\right)^\top}{1+\alpha}\right)^{-1} \tag{5.11}$$

$$= \frac{1+\alpha}{\alpha}\left[I_{l_i} - U^{i+1}\left(\alpha\left(Q_p^i\right)^{-1} + \left(U^{i+1}\right)^\top U^{i+1}\right)^{-1}\left(U^{i+1}\right)^\top\right]. \tag{5.12}$$

By saving the FIM blocks the memory usage for each layer l_i increases to $\mathcal{O}\left(l_i n^2 + l_{i+1} n\right)$ when using the low-rank inversion and saving U^{i+1} and $\left(A^i\right)^{-1}$ matrices, as visualized in Figure 5.3, while this complexity is $\mathcal{O}\left(l_{i+1} l_i^2\right)$ using the straightforward inverse for weights W^i of size $l_i \times l_{i+1}$, and analogously for V^i.

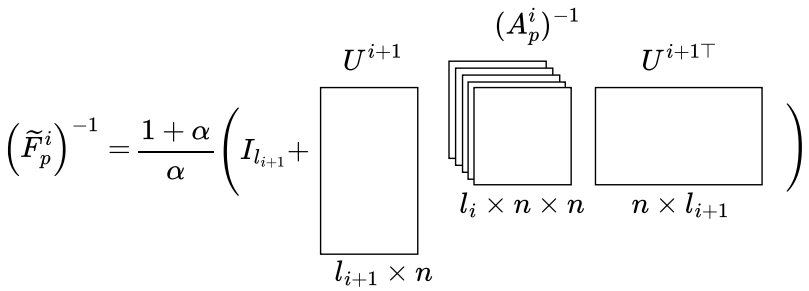


Figure 5.3: Inverse of the FIM defined as tensor products.

Using the Woodbury formula thus reduces the theoretical complexity of the method to $\mathcal{O}(l_0(l_1 n + n^\omega))$ every K-th step and $\mathcal{O}(l_0(l_1 n + n^2))$ on all the other steps. In the layers i where $n > l_i$, which is true for the narrower layers towards the top of the HM, there is no need to apply the Woodbury formula, because calculating and storing the inverse of the block is actually more efficient as in Eq. (5.11).

Remark 5
All calculations for computational complexities assume that the matrix products are calculated as efficiently as possible. In our case this means multiplying from the right, starting with the gradient vector and the last partial matrix. A short explanation about the nuances of matrix multiplications for the FIM is given in the Appendix B.

Besides the number of samples S, the minibatch size B and learning rate η, two other hyperparameters have been introduced for the NRWS: the damping

factor α, needed to invert the estimation of the FIM computed from the samples when it is not full rank, and the number of steps K during which the FIM is frozen, i.e., it is not updated with respect to the new minibatch, for computational efficiency. Hyperparameter tuning for the learning rate η, the damping factor α, and the value for K are presented in the Appendix C.2.

Our hyperparameter tuning experiments show that appropriate values for α are in the range of 0.01 to 0.2, depending on the network topology. We also found that K can be kept relatively high with values between 100 and $1,000$ with almost no loss in performance but with a significant gain in time.

This result shows that during training it is possible to avoid continuously re-estimating the geometry of the manifold of probability distributions, through the estimation of the FIM at each iteration, and that instead, a local approximation is sufficient to speed up the convergence when using the NG. A plausible explanation for this behavior is given by the use of the Tikhonov regularization which allows us to obtain more robust estimations for the FIM.

5.3.6 Diagonal Natural Reweighted Wake-Sleep

As part of our study of the NRWS, we verified whether it is possible to further constrain the structure of the FIMs and by extension make the algorithm faster, but without loss of accuracy in training. A rough way of approximating covariance matrices is to use only its diagonal components. The Diagonal Natural Reweighted Wake-Sleep (DNRWS) is a version of the NRWS where we approximate the FIM by taking only its diagonal elements. In the case of the HM, it is easy to calculate them by

$$F_p^{i,j} = \frac{1}{n} \sum s' \left(\left(W^{i,j} \right)^\top h^{i+1} \right) (h^{i+1})^2 \quad \text{and} \tag{5.13}$$

$$F_q^{i,j} = \frac{1}{n} \sum s' \left(\left(V^{i,j} \right)^\top h^{i-1} \right) (h^{i-1})^2 \, . \tag{5.14}$$

Inverting the matrices becomes trivial since they are diagonal in this setting. Because it is much faster, we can calculate the FIM approximation in every gradient step, with no need to save it for K steps.

Analyzing the change in the hyperparameters of the learning rate and damping factor α in Figure 5.4 reveals that usually, the best combination is similar to the one used for NRWS. Taking a smaller α leads to quicker convergence, but a worse minimum, with some instability when closer to convergence. Larger damping leads to a more stable convergence, but a slower one, compensating by speeding up with a larger learning rate leads to premature convergence. The DNRWS preserves the property of the damping factor, that for $\alpha \to \infty$ we recover the Euclidean gradient.

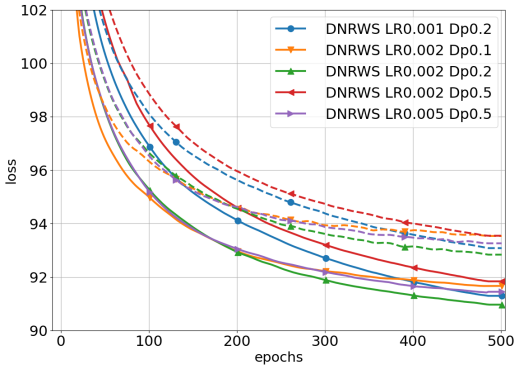


Figure 5.4: Loss of training (continuous line) and validation (dashed line) on MNIST with different Learning Rates and Damping Factors for the DNRWS [LR=learning rate η, Dp=Damping factor α].

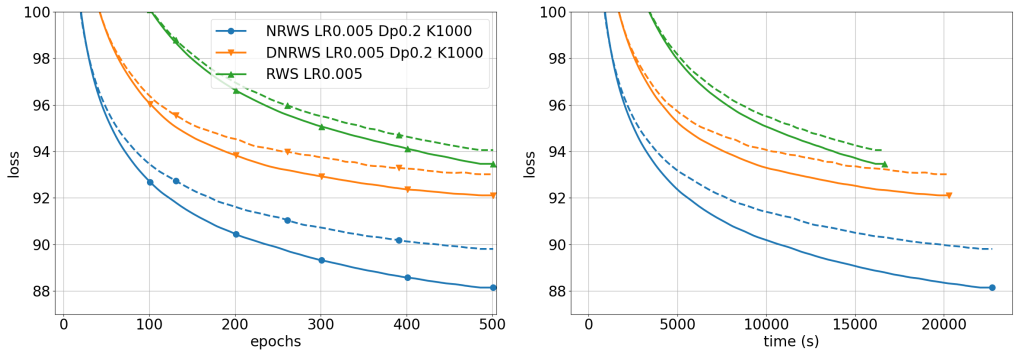


Figure 5.5: Loss of training (continuous line) and validation (dashed line) of RWS, DNRWS and NRWS on MNIST; (right) epochs (left) seconds of 500 epochs [LR=learning rate η, Dp=Damping factor α, K=K-step].

In Figure 5.5 we compare DNRWS to NRWS and RWS for a shorter period of 500 epochs where S and B was kept the same for all algorithms. We see a speedup of DNRWS compared to the RWS, but the achieved minimum is worse than that of NRWS. This observation is in line with what we were expecting, as the diagonal approximation of the FIM leads to worse results than the estimation of the actual structure. In wall-clock time the DNRWS is a bit more promising, as it manages to outperform RWS for a longer period. A deeper analysis with a full comparison of training algorithms till convergence can be found in the following section, where we find that our observations from this small-scale experiments remain true as the DNRWS fails to improve on either RWS or NRWS.

5.3.7 Experiments with the Natural Reweighted Wake-Sleep

For the performance evaluation of NRWS we use the binarized version of the MNIST dataset of handwritten digits (Deng, 2012) as a standard benchmark. In addition to MNIST, we show the efficiency of the NRWS on the FashionMNIST dataset and a downsampled version of the Toronto Face Dataset (TFD). In addition, we used the miniMNIST dataset for a brief explorative analysis of the hyperparameters shown in Appendix C, to determine good values for the learning rate, damping factor, and K-step parameters[4].

The functions optimized in training differ depending on the algorithm updating phases, see Section 2.3.1. To favor comparisons, in our plots, we report as loss function the Negative Log-Likelihood (NLL) (see Eq. (2.9)) averaged over minibatches and samples for all algorithms, since the NLL plays a fundamental role in the training of HMs.

In addition, we compared NRWS also to a version of the algorithm noted as DNRWS in the experiments, where only the diagonal elements of the FIM are computed and used in the evaluation of the NG. DNRWS employs a rough approximation of the FIM which is much faster to invert. We added this algorithm to our experiments to assess whether or not this is a good trade-off. We give some additional details about DNRWS and its performance in Section 5.3.6.

Preliminary analysis on the miniMNIST showed a very small standard deviation for the NLL over multiple runs of the same experiment, with different seeds. We tested the miniMNIST dataset with 24 different seeds and the best hyperparameters (LR 0.002 and Dp 0.05 as in Table D.1 and Figure D.2 in Appendix D). After 100 epochs we obtain an average log likelihood of -28.39 with std 0.04, while after 200 epochs an average of -28.20 and std 0.03. This shows that the variance is relatively small for different seeds and gets smaller over time. We repeated the experiments with multiple seeds on the TFD dataset as well, with the best hyperparameters, with 10 seeds. After $1,000$ epochs, the resulting LL on the test set had a mean of -370.0 and std 0.18. In light of these results, we could conclude that the algorithm is robust against randomness and that there is no growth of the variance (usually associated with the REINFORCE algorithm and its variants, see Section 2.3.3). Based on these observations we only present a single run per experiment with the confidence that they behave closely to an average run.

In all the experiments we worked with an epoch budget and a time budget

[4]More details about the datasets used for all experiments in this thesis are described in Appendix A.1

for the NRWS, or until the algorithm has converged, on the same hardware. For Figures 5.6, 5.7 and 5.8 we used an epoch budget of $2,000$ and a time budget of $70,000$ seconds which corresponds to roughly 20 hours. For each training algorithm, in the plot comparisons, we present the results associated with the best choice of the parameters (learning rate η, K, and damping factor α), optimized for $2,000$ epochs[5].

MNIST

The training is performed without data augmentation, with binary variables in $\{-1, 1\}$. In Figure 5.6 and Table 5.1 we report the results of experiments on the MNIST dataset with hyperparameters tuned for each individual algorithm. The experiments are performed with a binarized dataset, equivalently to other benchmarks in the literature (Bornschein and Bengio, 2015; Bornschein et al., 2016).

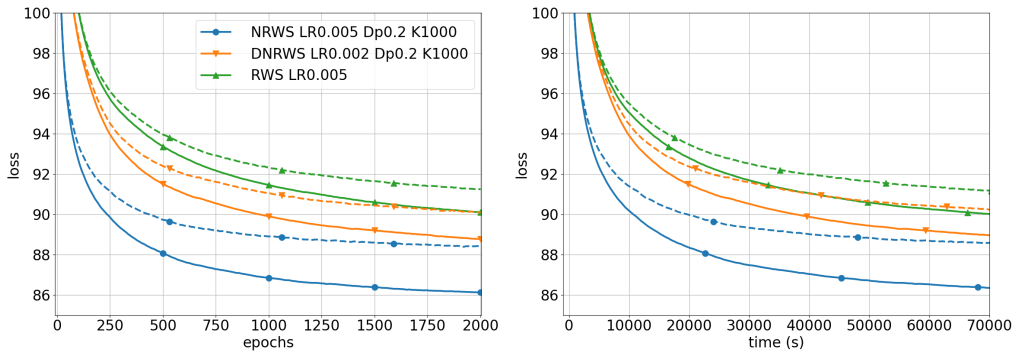


Figure 5.6: Training curves for MNIST with MBGD, continuous lines represent the quantities on the train set, and dashed lines the ones on validation; Left: loss of algorithms in epochs; Right: loss of algorithms in wall-clock time (s) [LR=learning rate η, Dp=Damping factor α, K=K-step].

In Figure 5.6 we present the loss curves during training, for the training and validation sets. The advantage of NRWS over RWS in these experiments comes in the form of convergence to a better minimum. NRWS converges faster than its non-geometric counterpart in epochs. In time (right panel), NRWS is faster than vanilla RWS, even if the time for each epoch is roughly 25% more (see Table 5.1).

[5]The complete details about the settings used in the experiments in this thesis, including software and hardware details, are described in Appendix A.2

The NG by pointing to the steepest direction with respect to the Fisher-Rao metric, allows for higher rates of convergence, however at the same time it might incur in premature convergence and thus reduce generalization properties. This phenomenon is known in the literature and has been already reported by other authors in different contexts (Glasmachers et al., 2010; Martens and Grosse, 2015; Pajarinen et al., 2019). In our experiments, we found that tuning the damping factor was sufficient to regularize the experiments.

We compare our MBGDs implementations of WS, RWS and NRWS with state-of-the-art algorithms (Bornschein and Bengio, 2015; Bornschein et al., 2016), see Table 5.1. We found, based on the final convergence values of the experiments, that our implementation of RWS and BiHM performs as well as the ones from the literature. Implementations from the literature also take advantage of accelerated gradient methods (ADAM(Kingma and Ba, 2015)), learning rate decay (from 10^{-3} to 3×10^{-4}), L^1 and L^2 regularizers, and an increased number of samples towards the end of the training (from 10 to 100), in order to achieve better results. While using variable learning rates, regularizers and variable number of samples (Bornschein et al., 2016) could be successfully employed to improve our reported results, this was not the scope of this work. Even with a simple training procedure (fixed learning rate, no regularization and fixed number of samples $S = 10$), we notice how the IS Likelihood on 10000 samples is better than RWS as reported from the literature (Bornschein and Bengio, 2015) and even slightly better than BiHM (Bornschein et al., 2016). The impact of variable learning rates and increased number of samples at convergence provides a substantial advantage for BiHM in (Bornschein et al., 2016), as it can be seen from the results obtained with our implementation discussed in Section 5.3.8, where NRWS compares favorably to BiHM using the same settings in training, up to hyper-parameter tuning. In particular, we expect our results for NRWS to improve further with the use of variable learning rates, additional regularizers[6], and an increased number of samples once the algorithm has reached convergence.

Additionally, notice that when training until convergence, the difference between DNRWS and NRWS becomes more significant, we conjecture that the rough approximation of the DNRWS is not able to capture information useful to reach a better optimum.

[6]See experiments with NRWS using L^1 and L^2 regularizers in Section 6.2.

ALG	**S**	$\boldsymbol{\eta}$	$\boldsymbol{\alpha}$	**K**	**LL**	**T/E**
WS	10	0.002	-	-	-90.56	30s
RWS	10	0.002	-	-	-87.36	34s
DNRWS	10	0.002	0.2	-	-86.88	39s
NRWS	10	0.002	0.2	1000	**-84.91**	43s
VAE	-	-	-	-	$\approx$ -89.5	-
RWS	10-100	0.001-0.0003	-	-	$\approx$ -86.0	-
BiHM	10-100	0.001-0.0003	-	-	$\approx$ -85.0	-

Table 5.1: Importance Sampling estimation of the log-likelihood (**LL**) on the test set with $10,000$ samples for different algorithms after training till convergence with MBGD. **T/E** is the average time per epoch, **S** is the number of samples in training, $\boldsymbol{\eta}$ is the learning rate, $\boldsymbol{\alpha}$ is damping factor and **K** from $\boldsymbol{K}$-step. The values for VAE (Kingma and Welling, 2014), RWS (Bornschein and Bengio, 2015), and BiHM are reported from (Bornschein et al., 2016) however the **T/E** are not comparable because of different hardware used in the experiments.

Toronto Face Dataset and FashionMNIST

We tested NRWS on a downsampled version of the Toronto Face Dataset (TFD) (Susskind et al., 2010) and the FashionMNIST dataset (Xiao et al., 2017). We used a 24×24 resized version for the TFD to be able to use only dense layers in the neural networks of the HM. Given the absence in the literature of experiments on HM with RWS on those datasets, the comparisons were performed with our implementation of RWS and NRWS.

We used a similar setting for experiments for each of the datasets, as for the MNIST. The same sample and batch size and architecture were used for the RWS and NRWS, but the learning rate and damping factor were individually tuned for each algorithm. For FashionMNIST we used the same network as for the MNIST experiments and for TFD we used $300, 200, 100, 75, 50, 35, 30, 25, 20, 20$ nodes for each layer, which is very similar, but wider at the last layer.

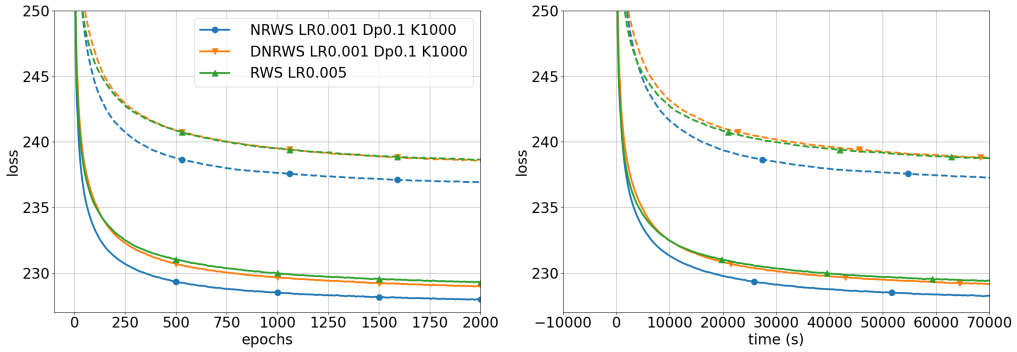


Figure 5.7: Training curves for FashionMNIST with Gradient Descent, continuous lines represent the quantities on the train set, and dashed lines the ones on test; Left: loss of algorithms in epochs; Right: loss of algorithms in wall-clock time (s) [LR=learning rate η, Dp=Damping factor α, K=K-step].

DS	ALG	S	η	α	K	LL	T/E
FashionMNIST	RWS	10	0.004	-	-	-236.96	38s
	NRWS	10	0.002	0.1	1000	**-235.65**	51s
TFD	RWS	10	0.002	-	-	-372.73	30s
	NRWS	10	0.002	0.2	1000	**-370.05**	39s

Table 5.2: Importance Sampling estimation of the log-likelihood (**LL**) on the test set with 10,000 samples for different algorithms after training till convergence with MBGD. **T/E** is the average time per epoch, **S** is the number of samples in training, $\boldsymbol{\eta}$ is the learning rate, $\boldsymbol{\alpha}$ is damping factor and **K** is from $\boldsymbol{K}$-step. The values for RWS are from our own implementation.

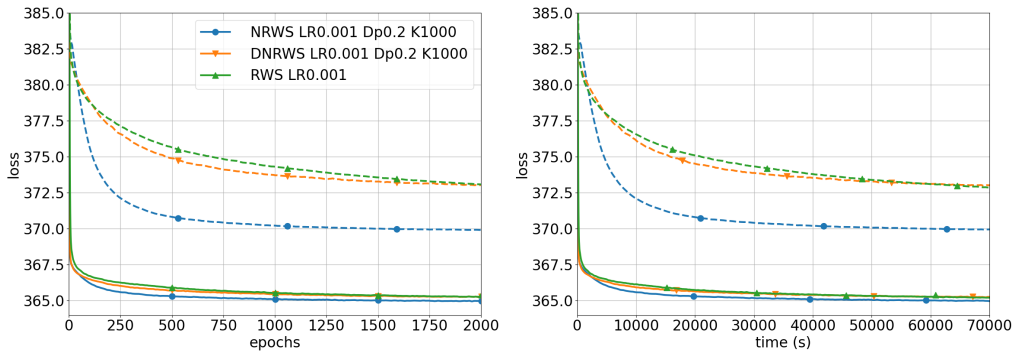
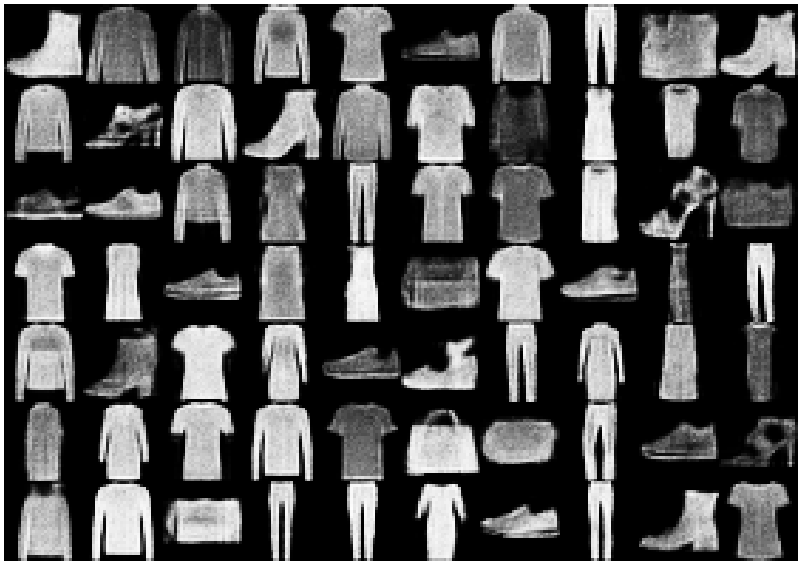


Figure 5.8: Training curves on TFD with Gradient Descent, continuous lines represent the quantities on the train set, and dashed lines the ones on test; Left: loss of algorithms in epochs; Right: loss of algorithms in wall-clock time (s) [LR=learning rate η, Dp=Damping factor α, K=K-step].



(a) Generated images FashionMNIST

(b) Generated images TFD

Figure 5.9: Example images generated with NRWS after 1000 epochs (a) FashionMNIST (b) TFD

In the results in the Figures 5.7 and 5.8 we can observe similar curves to what we saw in the case of MNIST. Even the best learning rate for the RWS cannot catch up with the NRWS neither in epochs nor in real-world time on both datasets. In the case of the TFD in particular we see a pretty big difference in the algorithms' ability to generalize. The test curves of

the NRWS perform much better than the RWS on both datasets, which the results from Table 5.2 corroborate. In Figures 5.9a and 5.9b we can see some generated images from an HM after being trained with NRWS on FashionMNIST and TFD respectively.

In the case of DNRWS, the limitations given by the use of a diagonal estimation of the FIM becomes more apparent, as for both datasets the algorithm behaves similarly to or marginally worse than RWS. From these experiments, it is obvious that the DNRWS uses a much too crude an approximation to the FIM to be useful in applications.

5.3.8 Natural Bidirectional Helmholtz Machine

The Bidirectional Helmholtz Machine (BiHM) (Bornschein et al., 2016), has been shown to obtain better performances compared to WS and RWS. However, differently from the former methods, BiHM optimizes a lower bound of the log-likelihood with respect to the probability distribution

$$p^*(x) = \left(\frac{1}{Z}\sqrt{p(x,h)q(x,h)}\right)^2 , \tag{5.15}$$

where Z is the normalization constant[7]. The advantage of this method is that both the p and q distributions are learned simultaneously without the need for alternating phases. On the other hand, the update rules for BiHM in practice are the same as the **wake** and **q-wake** phases from RWS, see Eq. (2.26) and (2.30), only with different weights $\tilde{\omega}_k$.

Unfortunately, the computation of the FIM for BiHM does not lead to a block-diagonal structure, due to how p^* is defined. However, with the network topology being the same in the two cases and due to the relationship of the updating rules of BiHM with those of RWS, as well as the fact that the underlying structure of the BiHM is an HM by construction, we propose, as a possible workaround, to employ as FIM a block diagonal matrix with the blocks $\mathcal{F}_p$ and $\mathcal{F}_q$ from Eqs. (5.6) and (5.7).

We are aware that this is not the true FIM for the underlying probability model employed by BiHM, however motivated by the use of the same updating rules for both algorithms and by promising experimental results presented here, we decided to study such a variant and call it Natural Bidirectional

[7]See Section 2.3.2 for more details.

Helmholtz Machine (NBiHM). The update rules of the NBiHM are

$$\begin{aligned}\theta_{t+1} &= \theta_t - \eta \widetilde{F}_p^{-1} \frac{1}{B} \sum_{r=1}^{B} \sum_{k=1}^{S} \tilde{\omega}_k \nabla L_{p,(k,r)} \quad \text{and} \\ \phi_{t+1} &= \phi_t - \eta \widetilde{F}_q^{-1} \frac{1}{B} \sum_{r=1}^{B} \sum_{k=1}^{S} \tilde{\omega}_k \nabla L^w_{q,(k,r)} \ .\end{aligned} \tag{5.16}$$

The algorithm is presented in Algorithm 5.

Algorithm 5: Natural Bidirectional Helmholtz Machine

Let x be a minibatch of samples from the dataset
Let p and q be the distributions of the generation and the recognition networks with weights W and V
Let ω be the importance weights from the BiHM
Let M be the depth of the HM
for *each layer* i *from* q *ascending with* $h^0 = x$ **do**
    Sample h^{i+1} from $q(h^{i+1}|h^i)$
    Compute the weights $\tilde{\omega}$ from the multiple samples (2.37)
    Compute the gradients $\nabla_{\theta^i} L_p$ with respect to W^i as in (2.27)
    Compute the matrices for $(\widetilde{F}_p^i)^{-1}$ for the sub-blocks in i with h^{i+1} and $p(h^i|h^{i+1})$ as in (5.5)
    Compute $\widetilde{\nabla}_p^i L_p = (\widetilde{F}_p^i)^{-1} \nabla_p^i L_p$
    Compute the gradients $\nabla_{\phi^i} L_q^w$ with respect to V^i similarly to `q-wake` as in (2.30)
    Compute the matrices $(\widetilde{F}_q^i)^{-1}$ for the sub-blocks in i with h^{i-1} and $q(h^i|h^{i-1})$ as in (5.6)
    Compute $\widetilde{\nabla}_q^i L_q^w = (\widetilde{F}_q^i)^{-1} \nabla_q^i L_q^w$
    Update W^i and V^i using η with the $\widetilde{\nabla}_p^i L_p$ and $\widetilde{\nabla}_q^i L_q^w$ with the $\tilde{\omega}$-s as in (5.16)

Experiments with the Natural Bidirectional Helmholtz Machine

For the experiments with NBiHM and BiHM we use the exact same model architecture, hyperparameters (mini-batch size and sample size) and data-augmentation as we have previously described[8]. The exact values for the

[8]See Appendix D for more details.

hyperparameters that are set on a per-experiment basis (learning rate, damping factor and K-step) are specified in the following and they have been chosen always to favor each algorithm for the given experimental setting.

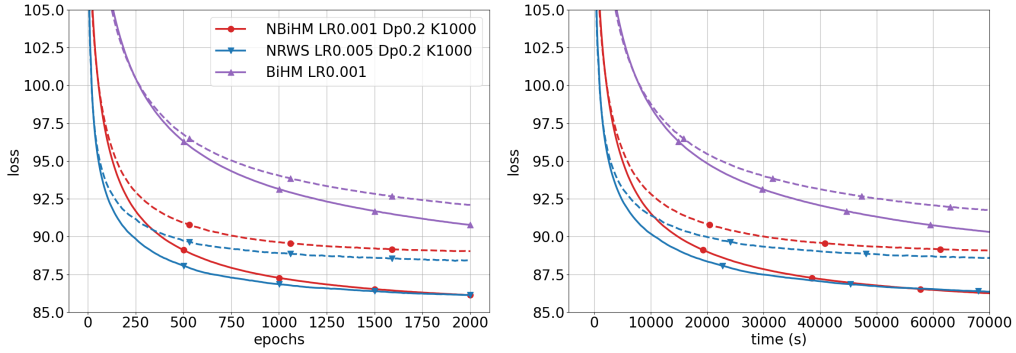


Figure 5.10: Training curves for MNIST for the BiHM and NBiHM algorithms, continuous lines represent the quantities on the train set, and dashed lines the ones on validation. Left: loss of algorithms over epochs; Right: loss of algorithms over wall-clock time (s) [LR=learning rate η, Dp=Damping factor α, K=K-step].

MNIST

We trained BiHM on MNIST as in the original paper (Bornschein et al., 2016), with MBGD with fixed learning rate and sample size, and we compared it to our NBiHM implementation, as in Section 2.3.2, to evaluate the impact of the NG for BiHM based on the FIM computed in NRWS. Similarly to our previous experiments, we performed them without gradient acceleration, no regularization, and no adaptive sample size.

In Figure 5.10, we notice that NBiHM compared to BiHM benefits from the use of the NG, both in convergence rate and for the value of the minimum obtained at convergence, even though the FIM used in the computation of the NG is not the proper one, but instead it is the one inherited from RWS. These results are somewhat unexpected as we know that we are not computing the gradient the correct FIM associated with the p^* from Eq. (5.15). The same finding can be observed in Table 5.3.

A further observation is that a big advantage of NBiHM versus NRWS is its running time. Since NBiHM is doing a single update for the weights of both networks, which corresponds to the wake and q-wake updates up to different reweighting factors, while the NRWS has a total of 3 phases,

ALG	S	η	α	K	LL p	LL p^*	T/E
BiHM	10	0.001	-	-	-87.6	-90.745	29s
NBiHM	10	0.001	0.1	1000	**-86.18**	**-89.21**	38s
NRWS	10	0.002	0.2	1000	**-84.91**	-	43s

Table 5.3: Importance Sampling estimation of the log-likelihood (**LL**) for both p and p^* on the test set for MNIST with 10,000 samples for different algorithms after training till convergence with SGD. **T/E** is the average time per epoch, **S** is the number of samples in training, $\boldsymbol{\eta}$ is the learning rate, $\boldsymbol{\alpha}$ is damping factor and **K** is from $\boldsymbol{K}$-step. The values for BiHM and NBiHM are from our own implementation.

NBiHM takes significantly less time for an epoch compared to NRWS (see Table 5.3). However, in spite of this, even if NBiHM outperforms BiHM, we could not reach the same accuracy and convergence rate obtained NRWS, which surpasses both methods. We hypothesize that this could be the side-effect of not using the proper FIM for the algorithm.

Toronto Face Dataset and FashionMNIST

DS	ALG	S	η	α	K	LL p	LL p^*	T/E
F-MNIST	BiHM	10	0.002	-	-	-237.99	-239.41	31s
	NBiHM	10	0.002	0.1	1000	**-235.95**	**-237.15**	38s
	NRWS	10	0.002	0.1	1000	**-235.65**	-	48s
TFD	BiHM	10	0.002	-	-	-375.44	-375.54	27s
	NBiHM	10	0.002	0.2	1000	**-370.24**	**-370.39**	30s
	NRWS	10	0.002	0.2	1000	**-370.05**	-	37s

Table 5.4: Importance Sampling estimation of the log-likelihood (**LL**) for both p and p^* on the test set with $10,000$ samples for different algorithms after training till convergence with SGD. **T/E** is the average time per epoch, **S** is the number of samples in training, $\boldsymbol{\eta}$ is the learning rate, $\boldsymbol{\alpha}$ is damping factor and **K** is from $\boldsymbol{K}$-step.

We compare BiHM and NBiHM to NRWS on the FashionMNIST and TFD datasets as well, with the results visible in Figures 5.11 and 5.12, respectively. The curves on the TFD seem to confirm that NRWS outperforms both NBiHM and BiHM, as previously noticed on the MNIST dataset. On the FashionMNIST dataset instead, we observe a different trend, NBiHM is the best overall method, and BiHM keeps a lead on NRWS for half the running time when looking at the wall-clock time.

The trend previously seen in the training curves is corroborated by the final convergence minima in Table 5.4, where NBiHM shows a large improvement over its non-geometric counterpart, for both log-likelihoods p and p^*. However, at convergence, in Table 5.4, the NRWS and NBiHM eventually catch up and achieve for both datasets very close final minima. Hence, while in the initial phases of training on the FashionMNIST, NBiHM prevails, as seen in the training curves, the final values at convergence are approximately within the standard deviation of the results (estimated to be approximately 0.18 on TFD, as shown in the beginning of the present section).

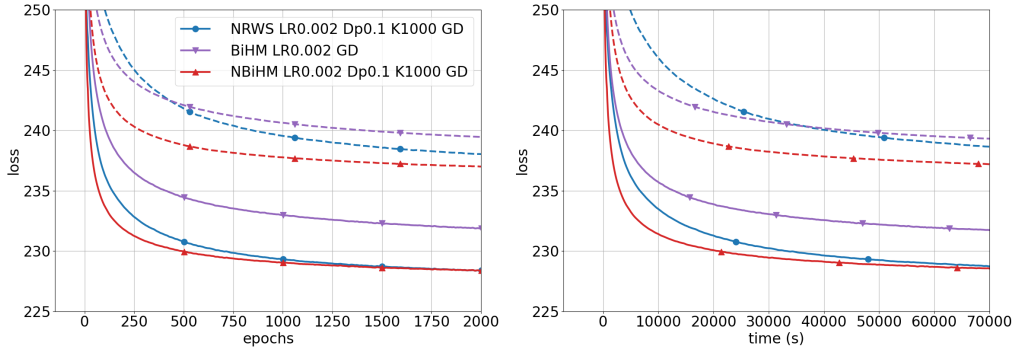


Figure 5.11: Training curves for FashionMNIST for the BiHM and NBiHM algorithms, continuous lines represent the quantities on the train set, and dashed lines the ones on validation. Left: loss of algorithms over epochs; Right: loss of algorithms over wall-clock time (s) [LR=learning rate η, Dp=Damping factor α, K=K-step].

5.4 Convergence Properties of the Wake-Sleep Algorithm

As introduced in Section 2.2 the WS algorithm switches parameters in the KL divergence for the sleep step. The KL divergence is not symmetric (see Section 3.4), so the switch in the order of the parameters breaks the basic converge guarantee of optimizing a single objective.

Ikeda et al. (1999) shows through a factor analysis model that one can understand each of the optimization steps as a geometric projection from one data manifold to the other. Consequently, if you only had a single KL function in both optimization steps (similarly to VAE Section 2.4.1 or REINFORCE Section 2.3.3), the algorithm would be a version of the geometric *em* algorithm

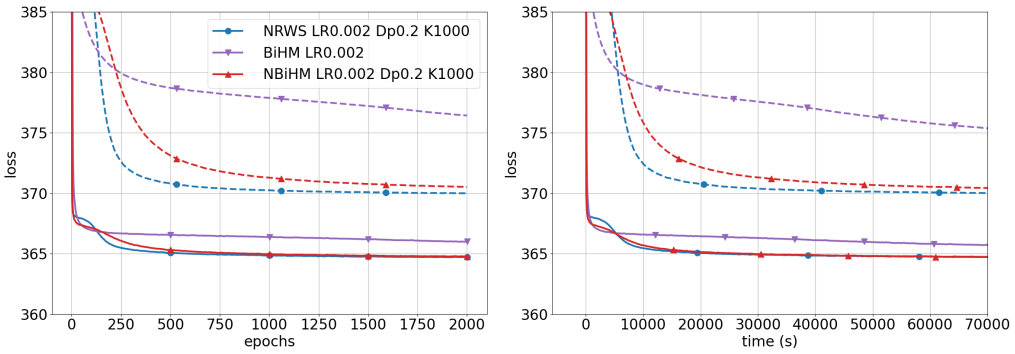


Figure 5.12: Training curves for TFD for the BiHM and NBiHM algorithms, continuous lines represent the quantities on the train set, and dashed lines the ones on validation. Left: loss of algorithms over epochs; Right: loss of algorithms over wall-clock time (s) [LR=learning rate η, Dp=Damping factor α, K=K-step].

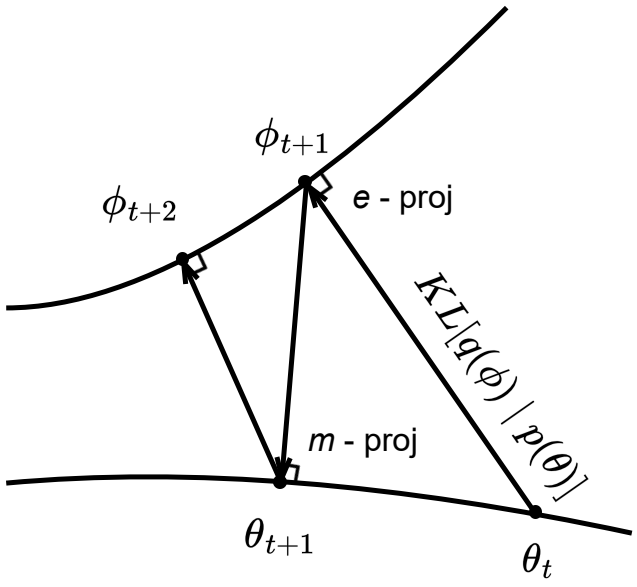


Figure 5.13: Example of optimization with e and m projections for the KL divergence; θ_t are the parameters of the generation network p and ϕ_t are the parameters of the recognition network q at step t. Figure readapted from (Ikeda et al., 1999)

as seen in Figure 5.13. The convergence of the *em* and their relationship to the Expectation-Maximization (EM) optimization process is known in literature and in particular has been studied by Fujiwara and Amari (1995)

and Amari (1995).

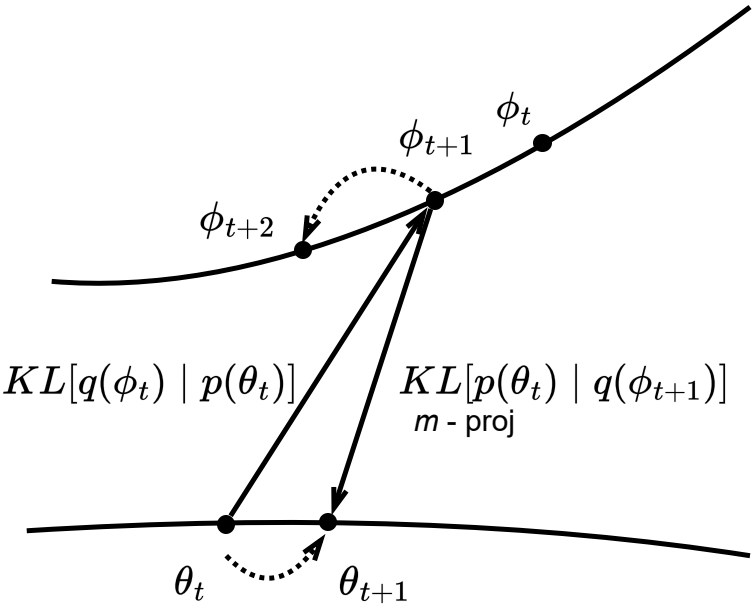


Figure 5.14: Optimization with e and m projections of the wake and sleep phases. Figure readapted from (Ikeda et al., 1999).

In the case of WS because of the different objectives, while the wake phase does behave like an m projection, the sleep phase does not conform to an e projection. Therefore, there is no straightforward way to prove that WS converges to an optimum for the primary objective. In Figure 5.14 we show a possible oscillating behavior of the projections.

5.4.1 The Sleep-well and Rescaled-sleep Variants of NRWS

Ikeda et al. describe a method, which they call Sleep-well (SW), where between each consecutive wake phase, they perform as many sleep phase gradient updates as needed until convergence. They prove that with this modification the SW phase is in fact equivalent to a gradient flow in the e step. Thus, the Wake-Sleep-well variant of the algorithm is a version of the *em* algorithm, and thus it converges to the Maximum Likelihood Estimation (MLE), but only when the model p is realizable by q, and can also be viewed as a Generalized Expectation-Maximization (GEM) algorithm (McLachlan and Krishnan, 2007). We illustrate in Figure 5.15 how the SW behaves geometrically.

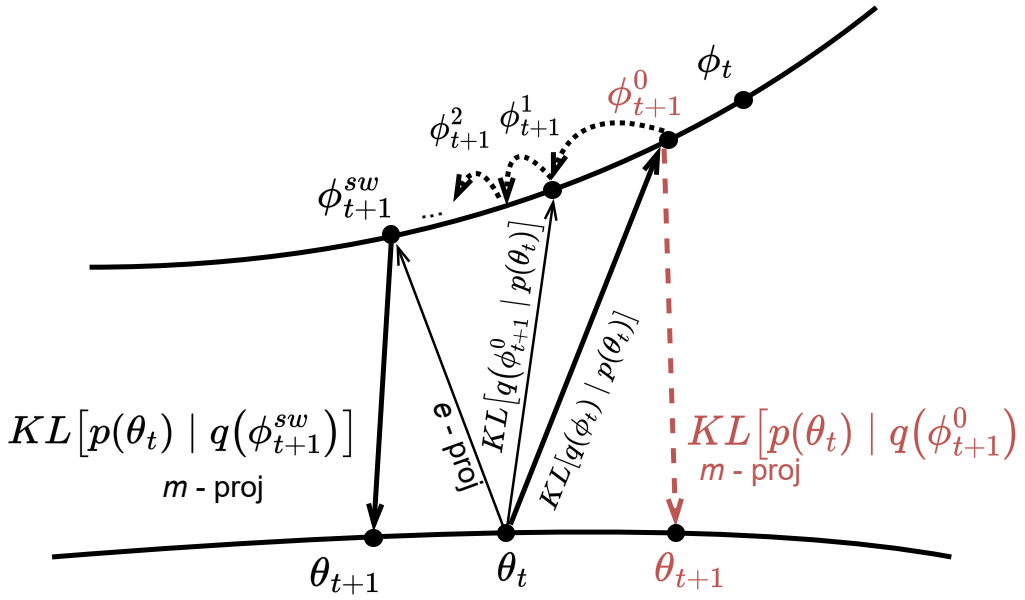


Figure 5.15: Optimization with e and m projections of the wake and sleep phases for the Sleep-well variant of the Wake-Sleep, for comparison the WS projection in dark red. ϕ^i_{t+1} are the parameters of the recognition network q at the i-th step sleep step within the $t+1$ update step. ϕ^{sw}_{t+1} is the final parametrization of the Sleep-well when it converges.

Notice that the algorithm by Ikeda et al. uses the exact FIM, while in the present work, we are employing an estimation of the gradients and of the FIM based on the minibatch. In the training of the model and in the estimation of the FIM RWS and NRWS are using weighted samples for each point in the batch to improve the quality of the estimation, this does not impact on the convergence properties derived for the WS, as the multiple samples only serve to better estimate the exact FIM and gradient. The only change RWS and NRWS introduce, is the use of the q-wake step, which is analogous to optimizing the same KL as the wake phase, which supports the original *em* convergence. Further studies on the convergence properties of RWS and NRWS in relation to the number of samples used in training represents an interesting research direction and will be the object of future work.

To the best of our knowledge, the SW variant of the WS is not used in practice in experiments in the literature. We show in a few experiments in Subsection 5.4.2 that while there is a noticeable difference between the WS and the SW algorithms from the perspective of the final convergence minimum, it is more convenient to use the former because of the increased

running time of the second.

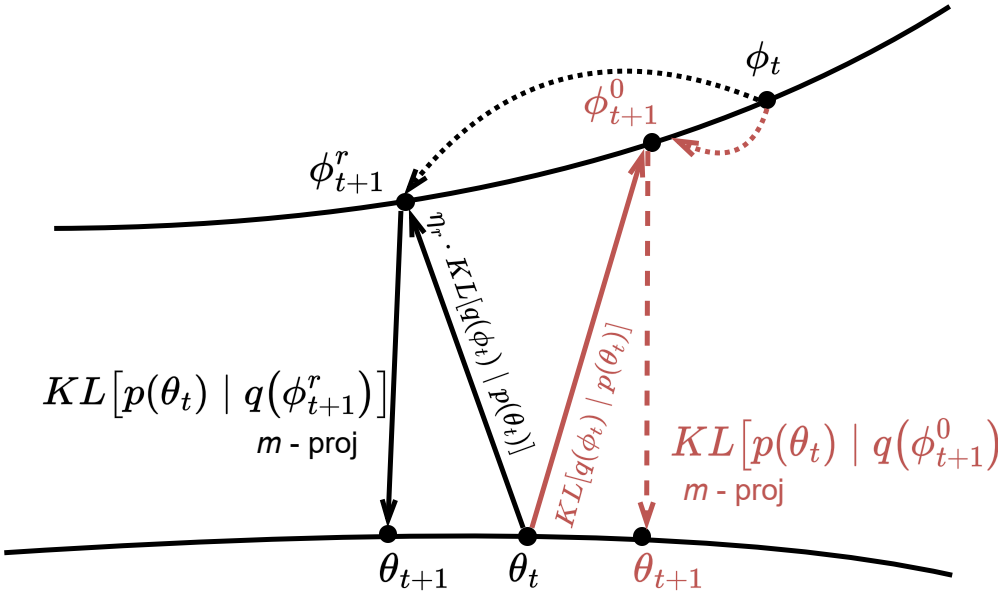


Figure 5.16: Optimization with e and m projections of the wake and sleep phases for the RS step variant of the Wake-Sleep. For comparison, we show the WS projection in dark red.

However, what if we could have the best of both techniques? We could approximate multiple sleep steps, by doing a larger step in the sleep step, which does not carry any time penalty with it. By rescaling the learning rate with some factor η_r, only for the sleep phase, could potentially carry most of the benefits of the SW, with none of the drawbacks. We call this variant of the WS Rescaled-sleep (RS) and we present in Figure 5.16 an illustration of such a rescaled sleep step during training.

Whether the RS step is equivalent geometrically to the multiple sleep steps, we can confirm that it is not. As in any GD based method, the consecutive gradient steps can change direction based on the underlying geometry of the manifold and are not equivalent to rescaling just one step. However, whether the rescaled method is "close enough" in approximating the SW to have its benefits, we show experimentally in the next section.

5.4.2 Experiments with Sleep-well and Rescaled-sleep Variants of NRWS

As preliminary exploratory analysis, we studied briefly the SW variant of Wake-Sleep. As mentioned in Section 5.4, WS only has theoretical convergence

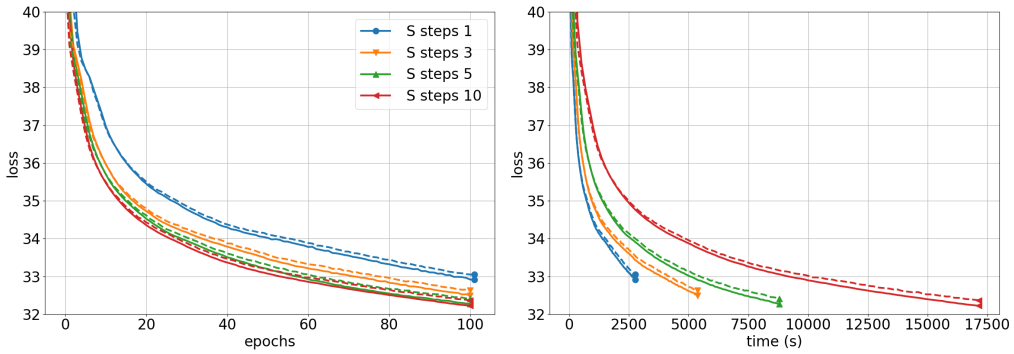


Figure 5.17: Loss on the train and validation sets of the miniMNIST with different 1, 3, 5, 10 sleep steps for every wake step, with learning rate $\eta = 0.001$. Left: loss of algorithms over epochs; Right: loss of algorithms over wall-clock time (s).

guarantees for the variant where the sleep phase is repeated until convergence after every wake phase. To the best of our knowledge, there are no studies in the literature using this variant. It is usually commonly accepted to use WS as a simple alternating algorithm, with one step of each phase instead.

In Figure 5.17 we compare 4 variants of the WS on miniMNIST[9], with the same hyperparameters, where we only changed the number of sleep steps: 1, 3, 5, 10 per one wake step. We see that there is a noticeable difference between the variants, with more sleep steps resulting in better convergence in epochs. However, looking at the real-time comparisons of the experiments, the conclusion takes a different perspective, as the time penalty for the multiple sleep phases ends up slowing down the algorithm significantly, with an amount that scales linearly with the number of steps.

Next, in Figure 5.18 we studied how the SW affects the NRWS on a longer training period. We noticed that the NRWS also benefits from extra steps in the sleep cycle, however, seemingly to a lesser degree than the WS. We also tested the RS variant, because taking a larger step could benefit the algorithm in a similar way as taking more but smaller steps. We found that slightly increasing the length of the sleep step with a factor of 3 does benefit both WS and NRWS. The benefit of increasing the size rather than the number of steps is that it does not come with any extra time penalty, however taking more steps achieves consistently better minima. Doing both larger and more steps, however, can impact negatively the algorithm (see red line in Figure 5.18).

[9]More details about the datasets and settings used in the experiments can be found in the Appendix Sections A.1 and A.2.

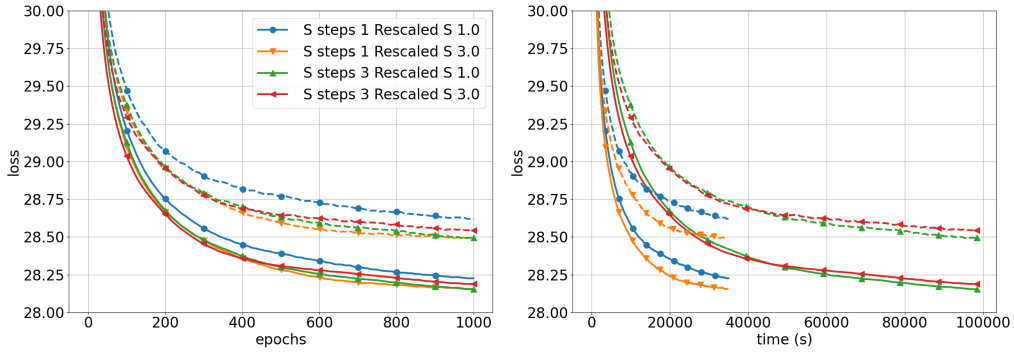


Figure 5.18: Loss on the train and validation sets of the miniMNIST with different 1, 3 sleep steps and 1 or 3 times RS steps for NRWS, with learning rate $\eta = 0.001$ and damping factor $\alpha = 0.2$. Left: loss of algorithms over epochs; Right: loss of algorithms over wall-clock time (s).

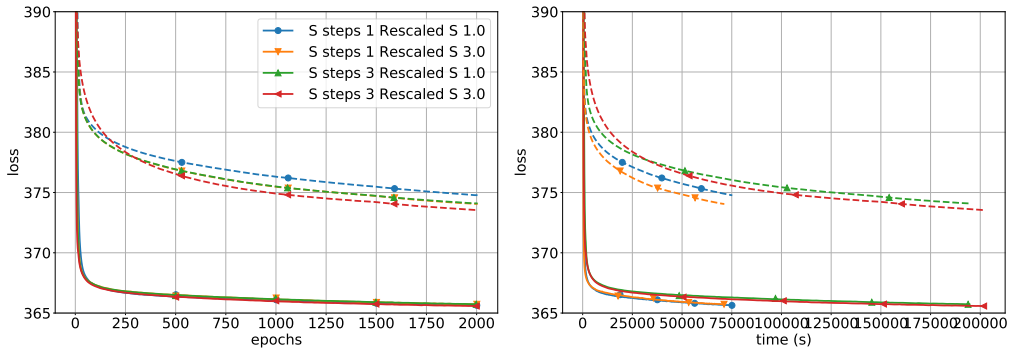


Figure 5.19: Loss on the train and validation sets of the TFD with different 1, 3 sleep steps and 1 or 3 times RS steps for NRWS, with learning rate $\eta = 0.0005$ and damping factor $\alpha = 0.2$. Note that the orange and green curves are almost completely overlapping in the left plot in epochs. Left: loss of algorithms over epochs; Right: loss of algorithms over wall-clock time (s).

To confirm these findings we repeated the same experiment but for the TFD dataset, and for a longer period. As we can see in Figure 5.19 the results are similar to the ones in the previous experiment, as 3 sleep steps and the RS step by a factor of 3 are almost completely overlapping. Contrary to the previous example, however, we see that in the case of TFD the combined Rescaled-Sleep-well performs the best, but again, we conclude that it is not worth doing multiple sleep steps, as it is too costly.

Remark 6
Given the relatively small loss improvement of SW and RS over WS and NRWS in Figures 5.17, 5.18 and 5.19, we decided to opt for the original variant of the WS with 1 *sleep step and with the same size as the wake phase for the rest of our experiments. The original variant is the fastest time-wise and it is in line with all other works from the literature, that we compare to, thus we used the standard single sleep step through all of our next experiments. However, based on our analysis in this section, we recommend the use of the RS for all variants of the WS for any future research, as it is easy to implement, and it has virtually no drawbacks when combined with an appropriate learning rate. In particular, RS does not lead to premature convergence, or overshoot the minimum in the sleep phase.*

Chapter 6

Acceleration and Regularization for Natural Reweighted Wake-Sleep

In this chapter, we expand on our previous work on the NRWS by applying techniques that are known to improve the performance of GD based algorithms, in terms of faster convergence or generalization capabilities, gradient acceleration and regularization.

We start by showing how we could adopt a class of methods to speed up the convergence of NG that we will refer to as Accelerated Gradients (AG). This class includes: Momentum (Polyak, 1964; Jacobs, 1988), Nesterov Momentum (Nesterov) (Nesterov, 1983) and Adam (Kingma and Ba, 2015). We discuss a formal framework for the definition of AG for training algorithms and we show how we can use such methods with the NRWS to speed up the convergence, with minimal drawbacks to the generalization capabilities.

Furthermore, we investigate the use of classical L^1 and L^2 regularization methods with the NRWS and propose a parametrization invariant regularization method, that can regularize independently from the choice of parameters. We show how such a Geometric Regularizer (GR) can decrease the generalization error in both NRWS and its non-geometric counterpart.

6.1 Accelerated Natural Gradients

Accelerated Gradients (AG) methods (see for a review Goodfellow et al. (2016, Chapter 8)) are very common in Deep Learning and have gained popularity in recent years (Kingma and Ba, 2015; Dozat, 2016; Keskar and Socher, 2017; Wilson et al., 2017; Duchi et al., 2011; Zeiler, 2012), as they can speed up the learning process of gradient descent-based optimization, with negligible increase in computation cost per iteration, compared to non-accelerated gradients. Intuitively, some of these methods, such as Momentum (Polyak, 1964), Nesterov and Adam, work on the principle of modifying the gradient vector based on how much it is aligned to the gradient vectors of previous

steps. More formally acceleration is often obtained as the discretization of second-order dynamics (Kingma and Ba, 2015). Additionally, for some AG methods there is an additional term in the gradient update step, a correction factor, which helps the GD based optimization to retain the correct direction, to lower the probability of overshooting the optimization minimum (Nesterov, 1983).

Using AG methods with the Euclidean gradients for the optimization of HMs does not present difficulties, and it has been applied in previous works by Bornschein and Bengio (2015) for RWS and Bornschein et al. (2016) for BiHM. In the following, we illustrate more in detail how AG methods are applied to the GD, through the example of the Momentum method (Jacobs, 1988) and the Nesterov Momentum (Nesterov) with the correction factor (Nesterov, 1983).

Momentum

We describe the gradient descent step as in (3.3)

$$\theta_{t+1} = \theta_t - \eta \nabla \mathcal{L}(\theta_t) \ ,$$

where θ are the parameters of the model, that are optimized, and $\nabla \mathcal{L}(\theta_t)$ are the gradients of the loss at step t and η is the learning rate. Compared to the simple GD the Momentum method adds a v_{t-1}^m velocity component to the parameter update rule, which is based on the previous update step at time $t-1$

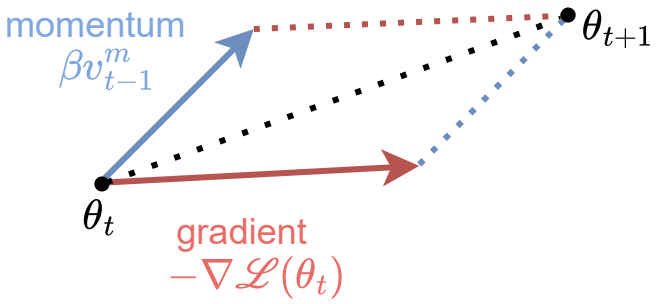
$$\begin{aligned} v_t^m &= \beta v_{t-1}^m - \nabla \mathcal{L}(\theta_t) \\ \theta_{t+1} &= \theta_t + \eta v_t^m \end{aligned} \tag{6.1}$$

where β is a parameter that controls the strength of the momentum component.

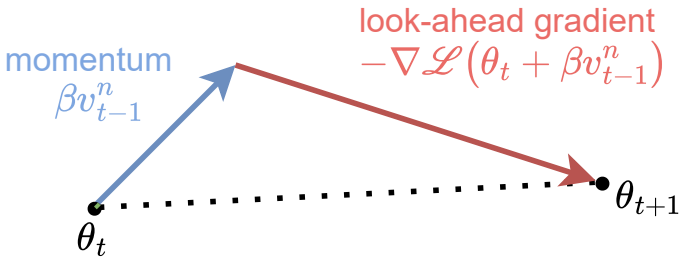
The inclusion of the velocity component results in much faster convergence (in training steps), especially at the beginning of the optimization.

Nesterov Momentum

The intuition behind the Nesterov Momentum (Nesterov) is that the update of the velocity vector and gradient vector is executed in reverse order, compared to the standard Momentum. In the Nesterov case, first we take a step in the direction of the velocity vector, which results in an intermediary point $\theta_{t+1/2}$, and from that point we calculate the gradient update. This second step can be understood as a "correction" to the potential overshooting of the



(a) Momentum step



(b) Nesterov Momentum step

Figure 6.1: The (a) Momentum and (b) Nesterov Momentum steps are illustrated as vector updates.

velocity term, therefore, it is also called the look-ahead gradient term or the correction factor (Goodfellow et al., 2016). The Nesterov parameter updates at step t are formulated as

$$\begin{aligned} \theta_{t+1/2} &= \theta_t + \beta v^n_{t-1} \\ v^n_t &= \beta v^n_{t-1} - \nabla \mathcal{L}(\theta_{t+1/2}) \\ \theta_{t+1} &= \theta_t + \eta v^n_t \ . \end{aligned} \tag{6.2}$$

The Momentum and Nesterov steps are illustrated in Figure 6.1.

Adam

While Momentum and Nesterov incorporate only the first-order dynamics of the gradient to adapt the learning rate, Adam (Kingma and Ba, 2015) also

adapts the second moment to accelerate the gradients. An update step of Adam is formulated as

$$\begin{aligned} v_t^a &= \beta_1 v_{t-1}^a - (1-\beta_1)\nabla\mathcal{L}(\theta_t) \\ m_t &= \beta_2 m_{t-1} - (1-\beta_2)\left(\nabla\mathcal{L}(\theta_t)\right)^2 \\ \theta_{t+1} &= \theta_t + \eta\frac{v_t^a}{\sqrt{m_t}+\epsilon}\ , \end{aligned} \tag{6.3}$$

where m_t is the discretization of the second moment, β_1 and β_2 are parameters setting the influence of v^a and m, and ϵ is used to control numerical tolerance.

In the remainder of this chapter, we will approach the AG methods from a geometric perspective. In the context of Euclidean geometry, the parameters are defined in $\theta \in \mathbb{R}^D$, where $\mathbb{R}^D$ could be understood as the Euclidean manifold $\mathcal{M}_E$. Then, the vectors v_t which are defined at θ_t are defined in the tangent space associated to the manifold $\mathcal{M}_E$ at point θ_t, $v_t \in T_{\theta_t}\mathcal{M}_E$. Let us take as an example the equations defining Nesterov, from a geometrical perspective we can notice how the vectors defined above belong to different tangent spaces

$$\begin{aligned} \theta_t, -\eta\nabla\mathcal{L}(\theta_t) &\in T_{\theta_t}\mathcal{M}_E \\ \beta v_{t-1}^n &\in T_{\theta_{t-1}}\mathcal{M}_E \\ -\eta\nabla\mathcal{L}(\theta_t + \beta v_{t-1}^n) &\in T_{\theta_{t+1/2}}\mathcal{M}_E\ . \end{aligned} \tag{6.4}$$

In most applications making use of AG methods for the training of Neural Networks, an underlying Euclidean geometry is implicitly assumed for the parameters to be optimized. This does not pose any issue related to comparing gradients belonging to different tangent spaces, since all the tangent spaces are canonically isomorphic, thus they can be naturally identified (Lee, 2013, p. 42), i.e.

$$T_{\theta_t}\mathcal{M}_E \cong T_{\theta_{t-1}}\mathcal{M}_E \cong T_{\theta_{t+1/2}}\mathcal{M}_E \cong T\mathcal{M}_E\ .$$

Therefore, the addition and subtraction of the vectors are well-defined.

However, AG methods in their basic formulation do not take into account the geometry of the statistical model defined by the HM, therefore we cannot apply them directly to NRWS.

6.1.1 Accelerated Gradients in Riemannian Geometry

AG methods have been studied before in the context of IG and Riemannian Optimization in the works of Alimisis et al. (2021, 2020); Chirco et al. (2022); Bécigneul and Ganea (2018); Liu et al. (2017). Furthermore, Martens (2020a) discusses how popular AG training algorithms such as AdaGrad (Duchi et al.,

2011), AdaDelta (Zeiler, 2012), and Adam (Kingma and Ba, 2015) are related to the NG, since they can be interpreted as providing diagonal approximations for the FIM.

On a Riemannian manifold $\mathcal{M}$, as in our case for the statistical model associated with the HM, the different vectors from Eq. (6.4) belong to different tangent spaces

$$T_{\theta_t}\mathcal{M} \neq T_{\theta_{t-1}}\mathcal{M} \neq T_{\theta_{t+1/2}}\mathcal{M} \tag{6.5}$$

hence they cannot be summed directly.

Note 9

Recall from Chapter 3 that we are using as retraction the subtraction operation, instead of an exponential map, in the case of vector operations on a Riemannian manifold. For more details about retraction and gradient updates, see Section 3.3 and the work of Absil et al. (2009).

In the following, we will revisit the concepts of connection, metric and parallel transform. For a brief explanation of these concepts see Sections 3.5.

On a Riemannian manifold, given a connection ∇, a special mapping can be defined among tangent spaces as the parallel transport, providing a solution to transport vectors from one tangent space to another (Absil et al., 2009). In the AG setting, the velocity vector v_{t-1}^m can be parallelly transported to the tangent space $T_{\theta_t}\mathcal{M}$ of the current point p_{θ_t}, in the case of the Momentum or the look-ahead gradient in the case of Nesterov. We illustrate visually the parallel transport before the sum of the gradient and velocity vectors needed for the Nesterov in Figure 6.2.

To compute the parallel transport of a vector X on the tangent space defined on the statistical manifold $\mathcal{M}$, one needs to solve the following differential equation

$$\nabla_{\dot{\gamma}} X = 0 \ , \tag{6.6}$$

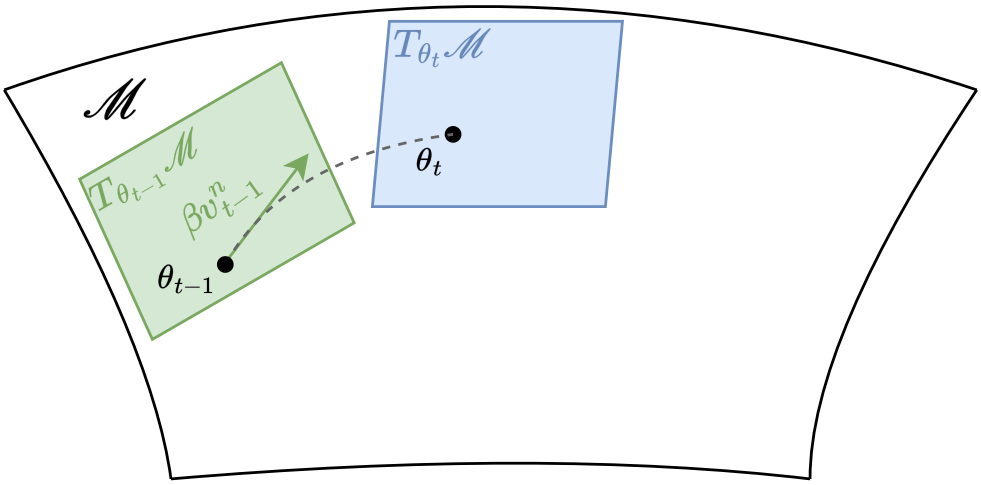
where γ is a curve connecting the source and destination points and ∇ is the connection (see 3.5). The expression can be written as

$$\nabla_{\gamma} X = \sum_{i,j,k} \gamma^j \left(X^k \Gamma_{jk}^i + \partial_j X^i \right) E_i \ , \tag{6.7}$$

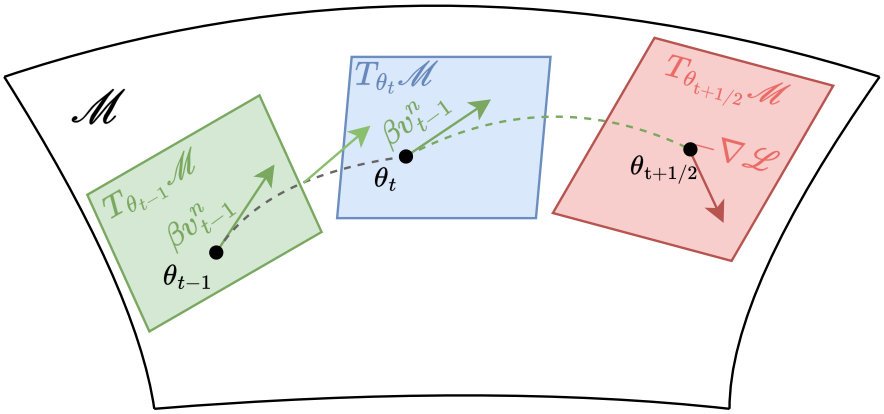
where Γ_{jk}^i are the Christoffel symbols, ∂_j are the partial derivatives with respect to the basis vector e_j and E_i is the coordinate vector field corresponding to the e_i basis vector (Absil et al., 2009).

One option for the choice of parallel transport would be to use the Levi-Civita connection, in which case the Christoffel symbols are defined as

$$\Gamma_{jk}^l = \frac{1}{2} g^{lm} \left(\partial_k g_{mj} + \partial_j g_{mk} - \partial_m g_{jk} \right) \ , \tag{6.8}$$



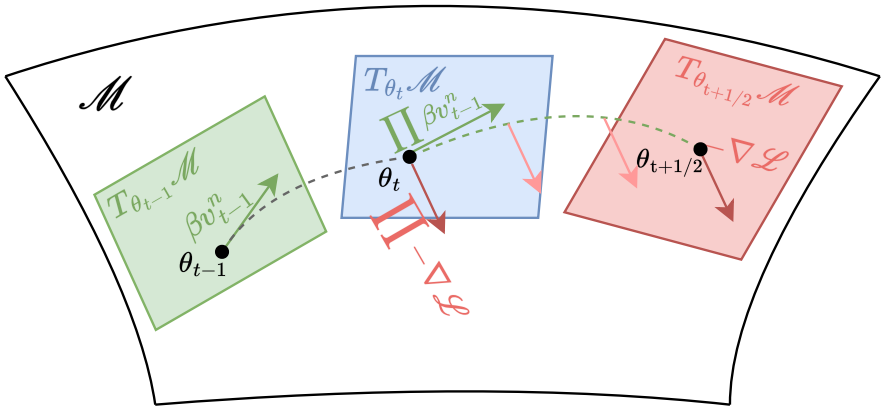
(a) The velocity vector v_{t-1} defined on the tangent plane $T_{\theta_{t-1}}\mathcal{M}$ associated to θ_{t-1}, while θ_t defines $T_{\theta_t}\mathcal{M}$.



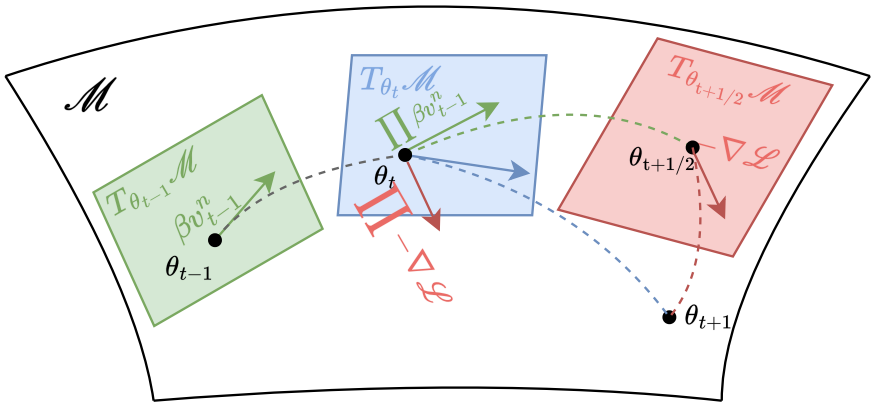
(b) Parallel transport of the velocity vector v_{t-1} from $T_{\theta_{t-1}}\mathcal{M}$ to $T_{\theta_t}\mathcal{M}$ and the look-ahead gradient vector $\nabla\mathcal{L}$ on $T_{\theta_{t+1/2}}\mathcal{M}$.

where g_{ab} are the elements of the metric described by the FIM, and g^{ab} are the elements of the inverse of the FIM. Computing Eq. 6.8, and by extension solving Eq. (6.6), is in general computationally very expensive on non-flat manifolds ($\Gamma_{jk}^{l} \neq 0$).

Another option would be to use the exponential connection, which would simplify the parallel transport in the general case, as for models in the exponential family where the connection is flat, i.e., we could transport the vectors as on the Euclidean manifold when models are parameterized in the



(c) Parallel transport of the look-ahead gradient vector $\nabla\mathcal{L}$ from $T_{\theta_{t+1/2}}\mathcal{M}$ to $T_{\theta_t}\mathcal{M}$.



(d) The sum of velocity v_{t-1} and look-ahead gradient vectors $\nabla\mathcal{L}$ with θ_t after the parallel transport to get θ_{t+1}.

Figure 6.2: The sum of the velocity and look-ahead gradient vectors using the parallel transport for a Nesterov Momentum update.

natural parametrization[1]. The HM, however, is by construction not part of the exponential family. A requirement for an SBN (both networks of the HM) to be part of the exponential family is for each node of the network, to have parent nodes which are either only one, or married, i.e., to have a connection between them (Lauritzen, 1987). This condition only applies for a HM if it is a chain of one node per layer since the children of the nodes in one network

[1]See Section 3.5.3 for a detailed explanation of exponential families and the exponential connection.

become the parents in the other network.

In order to obtain a computationally feasible approach to parallel transport, we relax these conditions, and we compute the parallel transport as if we had an exponential family at hand. In other words, we behave as in a Euclidean space with respect to the natural parameters and we transport vectors accordingly. Our main argument, to support this choice is the following.

Based on our previous experiments with the NRWS we noticed that the FIM describing the metric at a point θ_t is changing very slowly (see 5.3.4). If we assume the slowly changing metric to be constant for large portions of the statistical manifold, from equation (3.11) it follows that the Christoffel symbols Γ^i_{jk} are 0, since the derivative of a constant metric is 0 (Zuoqin, 2016, Chapter 10). Furthermore, it also follows according to Eq. (6.7) that the parallel transport corresponds to the identity. Therefore we could assume, that the tangent planes for the points θ_t, θ_{t-1} and $\theta_{t+1/2}$ are locally equivalent, and we can transport vectors between them as in Euclidean spaces.

Therefore, to use AG methods with NRWS the only modification we need to do to the methods is to exchange the gradients ∇ for the NG $\tilde{\nabla}$. These methods are then formulated as:

- Momentum

$$\begin{aligned} v_t^m &= \beta v_{t-1}^m - \tilde{\nabla}\mathcal{L}(\theta_t) \\ \theta_{t+1} &= \theta_t + \eta v_t^m \end{aligned} \tag{6.9}$$

- Nesterov

$$\begin{aligned} \theta_{t+1/2} &= \theta_t + \beta v_{t-1}^n \\ v_t^n &= \beta v_{t-1}^n - \tilde{\nabla}\mathcal{L}(\theta_{t+1/2}) \\ \theta_{t+1} &= \theta_t + \eta v_t^n \ . \end{aligned} \tag{6.10}$$

- Adam

$$\begin{aligned} v_t^a &= \beta_1 v_{t-1}^a - (1-\beta_1)\tilde{\nabla}\mathcal{L}(\theta_t) \\ m_t &= \beta_2 m_{t-1} - (1-\beta_2)\left(\tilde{\nabla}\mathcal{L}(\theta_t)\right)^2 \\ \theta_{t+1} &= \theta_t + \eta \frac{v_t^a}{\sqrt{m_t} + \epsilon} \ , \end{aligned} \tag{6.11}$$

6.1.2 Experiments for NRWS with Accelerated Gradients

In this section, we approach the training of the HM with AG from a practical standpoint as we test through experiments the effect they have on RWS and

NRWS. In these experiments, we do not use the mathematically correct (from a Riemannian perspective) AG methods for the NRWS, which would have to use parallel transport over statistical models, because, as we have discussed in the previous section, it would be computationally too expensive to do so in practice. Instead, we use the default version of the acceleration methods, as they might still be helpful from an optimization perspective. For each AG method we used the standard recommended parameters, which are the following:

- Momentum (Jacobs, 1988): $\beta = 0.95$;
- Nesterov Momentum (Nesterov, 1983): $\beta = 0.95$;
- Adam (Kingma and Ba, 2015): $\beta_1 = 0.9$ and $\beta_2 = 0.999$.

Since we use the NG as a drop-in replacement for the gradient, we do not fine-tune the parameters further for the individual algorithms, but take them as given, to see how a NRWS improves on the recommended setting.

For each experiment, we report the results for $2,000$ epochs of training for the NRWS and RWS. In Figures 6.4 and 6.6 we additionally compare the algorithms with LR in wall-clock time for a period of $70,000$ seconds which corresponds to roughly 20 hours. Finally, we train the RWS for $5,000$ epochs to compare the final convergence minima of the algorithms, to compensate for the larger per epoch run-time of the NRWS (see Table 5.1)[2].

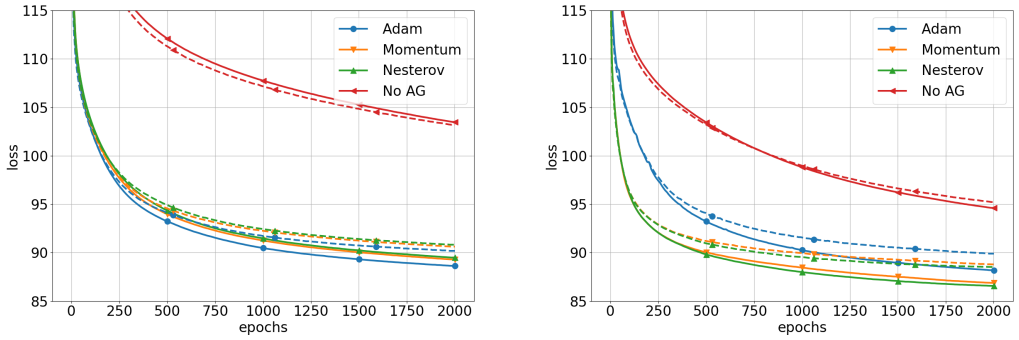
MNIST

We tested each of the AG methods discussed above with the RWS and NRWS variants of WS. In the experiments, we used learning rates of $\eta = 10^{-4}$, the damping factor $\alpha = 0.2$, $K = 1,000$, and a network structure of 300, 200, 100, 75, 50, 35, 30, 25, 20, 10 dense layers[3]. The reason we used a lower η than in the experiments in Chapter 5 is that we noticed a tendency for the AG methods to overfit.

In Fig. 6.3 we see that both RWS and NRWS benefit from the AG methods, outperforming in epochs the non-accelerated method by a visibly large margin, reaching better minima in a fraction of the time. In the case of RWS, we see that Momentum and Nesterov behave very similarly during the training,

[2]Based on the experiments in Section 5.3.7, the run-time of NRWS is only ~ 1.4 times slower than the one of RWS, which means that $\sim 3,000$ epochs would be enough to compensate for the time difference. However, the latter did not converge in that period, so we ran it till $5,000$ to compare final convergence values.

[3]The values for learning rate, damping factor and further hyper-parameters are described and motivated in Appendix A.



(a) AG on MNIST for RWS

(b) AG on MNIST for NRWS

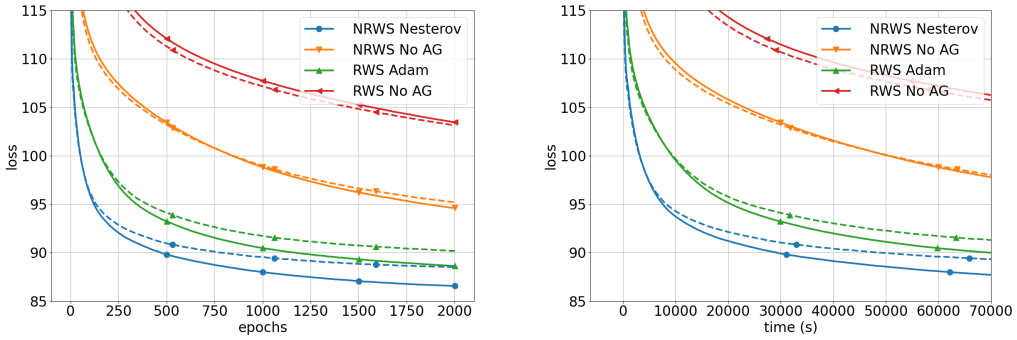
Figure 6.3: Comparison of Adaptive Gradient methods on MNIST for RWS and NRWS, continuous lines represent the loss on the train set, and dashed lines are the ones on validation.

with almost overlapping curves. At the same time, Adam outperforms them both for the best results in convergence rate and minimum reached. These results are in line with what we would expect from the literature about these methods (Goodfellow et al., 2016). In the case of the NRWS, we see a slightly different behavior, as both Momentum and Nesterov outperform Adam. This behavior is somewhat surprising, as we would expect better performance from Adam, based on what we have seen in the case of RWS. We speculate about the reasons behind this behavior in the following section.

In Figure 6.4, we compare RWS and NRWS with their respective best performing AG method, Adam for RWS and Nesterov for NRWS. In these plots, we see that both in epochs and wall-clock time, NRWS using Nesterov outperforms RWS using Adam, from the very beginning of the training. The IS Likelihood estimations corroborate the results from the plots in Table 6.1. For the RWS, we see that Adam presents the best results, while for NRWS reaches the best log-likelihood value for Nesterov.

TFD and FashionMNIST

We begin the experiments by comparing the Momentum and Nesterov methods on the TFD and FashionMNIST datasets and then Adam (Kingma and Ba, 2015). We use the same structure for the underlying HM as in the previous experiments, for the algorithms RWS and NRWS we use the same hyperparameters as in Subsection 5.3.7, with a fixed learning rate of $\eta = 0.0001$ and the other hyperparameters set to the standard values as described in Appendix A. Our aim in these experiments is to see whether any of the



(a) AG on MNIST for RWS and NRWS in epochs

(b) AG on MNIST for RWS and NRWS in time

Figure 6.4: Comparison of best AG methods on MNIST for RWS and NRWS, continuous lines represent the loss on the train set, and dashed lines are the ones on validation. Left: loss of algorithms over epochs; Right: loss of algorithms over wall-clock time (s).

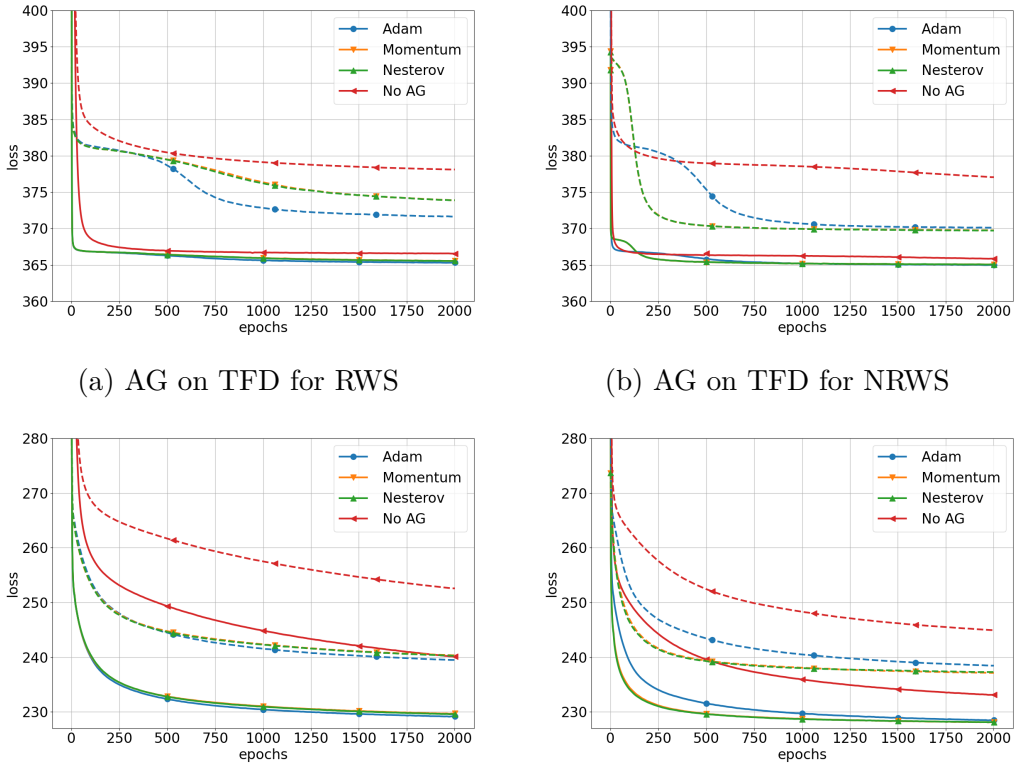
ALG	Epochs	AG method	LL
RWS	2000	-	-97.58
		Momentum	-87.14
		Nesterov	-87.25
		Adam	**-86.75**
	5000	-	-92.83
		Momentum	-86
		Nesterov	-86.07
		Adam	**-85.65**
NRWS	2000	-	-90.79
		Momentum	-85.7
		Nesterov	**-85.47**
		Adam	-86.59

Table 6.1: Importance Sampling estimation of the log-likelihood (**LL**) on the test set with 10,000 samples for different algorithms after training till convergence.

Momentum based AG methods can improve on the performance of the NRWS.

In Figure 6.5 we show how the curves of the loss evolve during the training on the datasets, for the RWS and the NRWS. In each plot, we compare the 3 aforementioned AG methods to the simple non-AG variant of the algorithms.

We notice that the curves for Momentum and Nesterov almost completely



(a) AG on TFD for RWS

(b) AG on TFD for NRWS

(c) AG on FashionMNIST for RWS

(d) AG on FashionMNIST for NRWS

Figure 6.5: Comparison of Adaptive Gradient methods on TFD and FashionMNIST for RWS and NRWS, continuous lines represent the loss on the train set, and dashed lines the ones on validation.

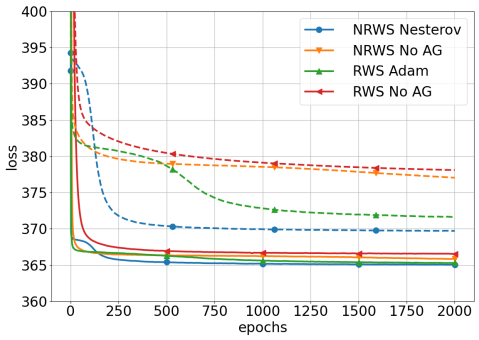
overlap. Even when they do not overlap, their behavior is very similar, with approximately the same inflection points and minima, even though all weights in the experiments are initialized with some random noise added. Therefore we will mostly discuss the AG methods in pairs as the first-degree methods (Momentum and Nesterov) and the second-degree method (Adam)[4].

Looking at the experiments done with RWS (Figures 6.5a and 6.5c), they reflect our expectations based on the literature, for both datasets. The algorithms employing first-order AG methods outperform the plain gradient variant, both in convergence rate and reached minimum loss. Adam further improves on the performance over the first-order methods, in both speed and

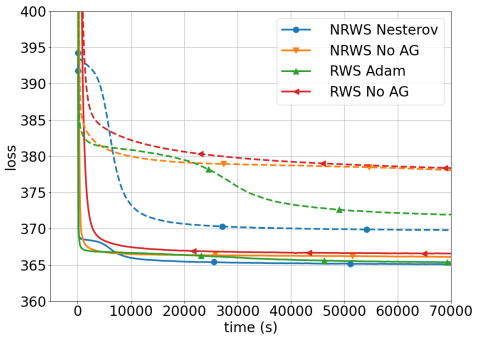
[4]By first-degree and second-degree methods we mean that the former use the discretization of the first derivative of the moment and the latter use the discretization of the first- and second derivatives (Kingma and Ba, 2015), as described in the previous Section.

convergence minima, more in the case of TFD than for FashionMNIST, but also in the latter to a noticeable degree.

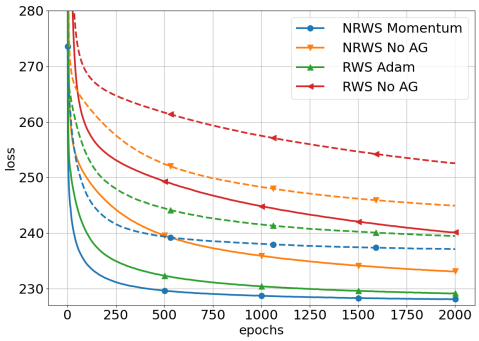
When analyzing the experiments done with NRWS (Figures 6.5b and 6.5d), we see a slightly different behavior. While all AG methods seem to improve on the results in convergence rate and loss, we notice a switch in the order of the methods from a performance perspective, as the first-order methods overtake the Adam very early in the training (from the beginning in the case of FashionMNIST and at around 100 epochs for TFD). Not only do the first-order methods have a faster convergence rate, but they also reach better minimums for both datasets.



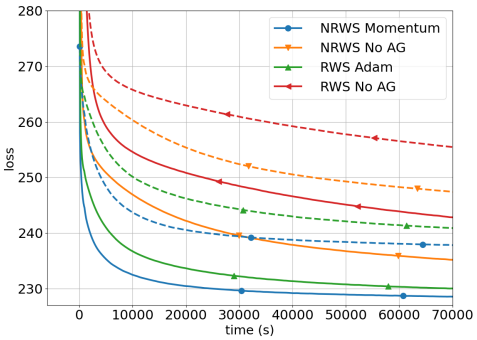
(a) AG on TFD comparison in epochs



(b) AG on TFD comparison in time



(c) AG on FashionMNIST comparison in epochs



(d) AG on FashionMNIST comparison in time

Figure 6.6: Comparison of the best Adaptive Gradient methods on TFD and FashionMNIST for RWS and NRWS, continuous lines represent the loss on the train set, and dashed lines the ones on validation. Left: loss of algorithms over epochs; Right: loss of algorithms over wall-clock time (s).

In Figure 6.6, we compare RWS and NRWS on FashionMNIST and

TFD with their respective best performing AG method, Adam for RWS and Nesterov or Momentum for NRWS. In these plots, we see that both in epochs and wall-clock time, for both datasets, NRWS outperforms RWS both using AG, from the very beginning of the training.

The results are summarized in Table 6.2, where we present the losses of the algorithms using Importance Sampling with $10,000$ samples after $2,000$ epochs for NRWS and $5,000$ epochs (till convergence) for RWS. We see the results from Figure 6.6 reflected in the values of the table, namely, that in the case of RWS, Adam dominates the other AG methods, while for NRWS Momentum and Nesterov seem to be the best choices. It is worth mentioning that in almost all cases the NRWS outperforms the RWS when comparing them with the same AG, with the notable exception of Adam for FashionMNIST.

Our hypothesis about the reason why NRWS prefers first-order methods over Adam lies in the assumptions we take when we use the NG with AG methods. When applying the first-order methods to an NG optimization without taking into account the geometry, we are implicitly assuming a locally flat manifold for the associated statistical model, where one can parallelly transport vectors, as in the Euclidean space. Probably, if the manifold is close to being flat in the neighborhood of a point, this action can be done, without a strong impact on the optimization process, and it leads to good results, as in our case for the Momentum and the Nesterov which both improve on the non-AG variant.

When using second-degree methods, the implicit assumption of almost flatness in a neighborhood of a point appears to be stronger compared to first-order methods, since the covariant derivative (which we assume to vanish) is used not only to compute parallel transports but also to compute accelerations. While we see that the second-degree methods also improve on the NRWS, not to the same degree as the first-degree ones. One could argue that this is because of too strong of an assumption, however, we have seen in previous experiments (Section 5.3.4 and Appendix C.2), that the FIM of the manifold is very slowly changing allowing us to avoid the recalculation of it for a K amount of steps. Seeing how Christoffel symbols for the Levi-Civita connection depend on the gradient of the metric, small changes in the FIM result in almost negligible changes in the covariant derivative on the manifold with respect to the directional derivative.

Notice how Martens (2020b, Section 11) discusses how second-order methods, such as Adam can be understood as using empirical estimations of the diagonal components of the FIM and by extension, they can be viewed as NG variants. Therefore we could understand using Adam and NG at the same time for HM as overlapping methods, where Adam adopts a rough

approximation of the FIM in the computation of the gradient and NRWS uses an estimation of the true FIM. In the light of this, we speculate that in this context applying Adam to NG, appears to be incompatible, indeed, in our experiments we found that using Adam with NRWS leads to sub-optimal performance.

DS	ALG	Epochs	AG method	LL
FashionMNIST	RWS	2,000	-	-249.44
			Momentum	-239.05
			Nesterov	-238.99
			Adam	**-238.28**
		5,000	-	-244.85
			Momentum	-237.44
			Nesterov	-237.39
			Adam	**-236.66**
	NRWS	2,000	-	-243.1
			Momentum	**-236.16**
			Nesterov	-236.29
			Adam	-237.33
TFD	RWS	2,000	-	-377.77
			Momentum	-373.76
			Nesterov	-373.72
			Adam	**-371.51**
		5,000	-	-377.24
			Momentum	-372.57
			Nesterov	-372.42
			Adam	**-370.89**
	NRWS	2,000	-	-376.74
			Momentum	-369.7
			Nesterov	**-369.68**
			Adam	-370.01

Table 6.2: Importance Sampling estimation of the log-likelihood (**LL**) on the test set with 10,000 samples for different algorithms after training till convergence.

While our results using AG methods with NRWS are promising, the adaptive steps of Momentum and Nesterov and the accumulated momentum of Adam are still implicitly assuming a Euclidean manifold, which from a geometrical perspective is not the correct approach, since it is not considering the Fisher-Rao geometry of statistical models. This motivates the exploration

of Adaptive Riemannian Gradient methods for HMs, in potential future works, which account for the geometry of the statistical models.

6.2 Regularization of Weights in Helmholtz Machines

A recurring problem in all Machine Learning tasks is the lack of generalization. The most common way to quantify this issue is to measure the generalization error, which measures how well a model can behave on yet unseen data, compared to the performance on the training set. Commonly the generalization error is measured by studying the generalization gap, which is the gap between the losses measured on the train set and the validation set during training. While we expand on our previous work, we can develop better methods to improve the generalization gap, by regularizing the parameter space.

Regularization[5] is a technique in Machine Learning that is commonly used to obtain better generalization properties, a typical approach is by forcing the weights of a model to be small, and implicitly to keep the model more simple (Occam's razor principle) (Goodfellow et al., 2016). The reason for this is usually to prevent overfitting when a model learns the training data too well and does not generalize on new unseen data. One method commonly used is forcing a simplified structure on the model, so it does not get as "overconfident" in its goal by learning noise in the training set, and thus as a consequence, by reducing the number of weights it reduces the generalization gap (or generalization error). We measure the generalization gap as the distance between the performance of the algorithm on the training set and the validation set (Bishop, 2006). This measure is not an absolute measure, however, but a relative one. When comparing two different training methods on the same dataset which have the same performance on the train set, we prefer the one where the generalization gap is smaller, i.e. the performance of the algorithm on the validation set is better or equal, as it usually generalizes better on yet unseen data.

6.2.1 L^1 and L^2 Regularization

A common way of regularizing a training algorithm is to add to the loss function a term that penalizes some norm of the weights, i.e. to force the values to be small, as illustrated in Figure 6.7.

[5] We refer to Goodfellow et al. (2016, Chapter 7) for a review of common regularization techniques in Machine Learning.

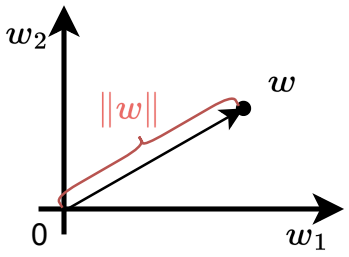


Figure 6.7: L^2-norm of a weight vector $w = [w_1, w_2]$.

The most common regularizers, which we will be referring to as $\mathsf{L}^1/\mathsf{L}^2$ Regularizers (LR), are L^1 and L^2 regularizers, based on the L^1 and L^2 norms, which are defined as

$$\mathsf{L}^1(x) = \|x\|_1 = \sum_i |x_i| \quad ;and \tag{6.12}$$

$$\mathsf{L}^2(x) = \|x\|_2 = \sqrt{\sum_i x_i^2}\,, \tag{6.13}$$

which correspond to the first- and second-order *p-norms*

$$\mathsf{L}^p(x) = \|x\|_p = \sqrt[p]{\sum_i x_i^p}\,.$$

Commonly, the norms are raised to the power of p so that the root functions disappear, which simplifies the calculation of the gradients with respect to them.

These common regularization methods, however, do not account for the geometry of statistical models. In the case of statistical models that have a Riemannian geometry, as for HMs, these regularization methods are not compatible with the geometry, for instance, because they are not invariant to the choice of parameters. This, in turn, means that LR for statistical models might not lead to the desired effect of favoring models with better generalization properties.

To evaluate the effectiveness of LR methods on HM, we add a regularization term to the empirical estimate of the loss, with a coefficient β, to control the strength of regularization. Thus the new update rules for the NRWS will be

$$\begin{aligned}
\theta_{t+1} &= \theta_t - \eta \widetilde{F}_p^{-1} \frac{1}{B} \sum_{r=1}^{B} \sum_{k=1}^{S} \left(\tilde{\omega}_k \nabla L_{p,(k,r)} \right) + \beta \nabla \|\theta_t\|, \\
\phi_{t+1} &= \phi_t - \eta \widetilde{F}_q^{-1} \frac{1}{B} \sum_{r=1}^{B} \sum_{k=1}^{S} \left(\frac{1}{2S} \nabla L^s_{q,(k,r)} + \frac{\tilde{\omega}_k}{2} \nabla L^w_{q,(k,r)} \right) + \beta \nabla \|\phi_t\| \,,
\end{aligned} \tag{6.14}$$

where $\|\cdot\|$ is a placeholder for both L^1 and L^2 norms.

6.2.2 Experiments with L^1 and L^2 Regularization

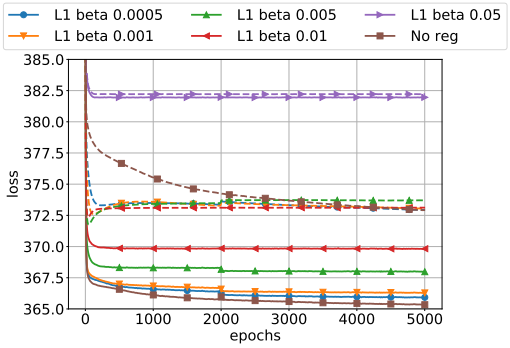
We conducted some experiments with the LR on HMs, in order to evaluate what effect they have on the training and generalization properties of the algorithms. We tested the L^1 and L^2 regularizers on both FashionMNIST and TFD in a similar experimental setting, as in Subsections 5.3.7 and Appendix A. We chose the best-performing learning rates, damping factors, and K-step for each algorithm from the previous experiments. For both FashionMNIST and TFD datasets we used the same network as in all previous experiments Subsection 5.3.7. The added regularization terms to the loss are equal in both cases (RWS and NRWS) and the added time penalty is negligible, compared to the cost of the other operations (dominated by the computation of the FIM). We present the behaviors of the LR variants on RWS and NRWS during training in Figures 6.8 and 6.9, where we used a budget of $2,000$ epochs. In Table 6.3, we present the final convergence minima of the best-behaving algorithms, where we let the RWS train till $5,000$ epochs, to compensate for the slower per-epoch run-time of NRWS (see Table 5.1)[6].

We tested different values for β for RWS and NRWS, for both datasets, to see the effect that each regularization method has on the respective algorithm. Through these experiments, we also want to find a candidate value for β, which minimizes the generalization gap and provides good performance on the validation set, for each training method and dataset.

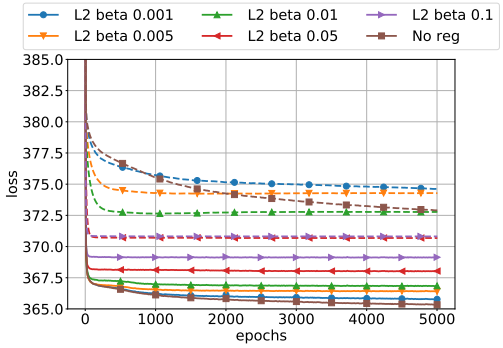
We present in Figure 6.9 the effects of the L^1 and L^2 regularizers on both RWS and NRWS for FashionMNIST and in Figure 6.8 for TFD.

In the case of the TFD in Figure 6.8, we see different behaviors for L^1 and L^2 regularizers, while both improve on RWS and NRWS in some capacity. Starting with the L^1 case, notice that for RWS using most of the tested values for β in the range of $0.0005 - 0.01$ leads to almost the same performance on the validation set, with an early convergence, compared to the non-regularized version. We can see the same early convergence behavior in the case of the NRWS for all values of β, however, it is immediately apparent that all regularized models end up at a worse convergence minimum than the non-regularized algorithm. In this sense, L^1 regularization makes the learning worse, than if we would not regularize at all.

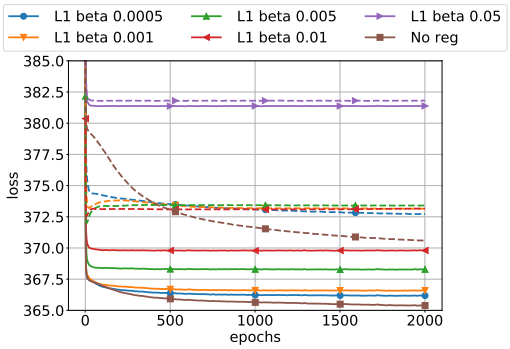
[6]Based on the experiments in Section 5.3.7, the run-time of NRWS is only ~ 1.4 times slower than the one of RWS, which means that $\sim 3,000$ epochs would be enough to compensate for the time difference. However, the latter did not converge in that period, so we ran it till $5,000$ to compare final convergence values.



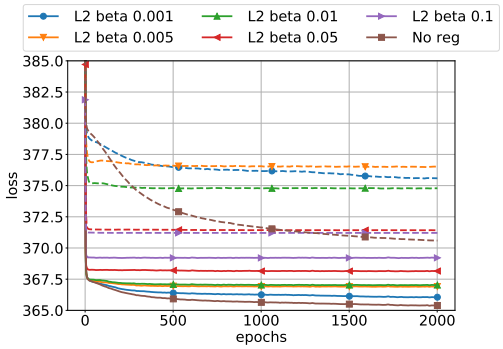
(a) L^1 on TFD with RWS.



(b) L^2 on TFD with RWS.



(c) L^1 on TFD with NRWS.



(d) L^2 on TFD with NRWS.

Figure 6.8: Training curves representing the loss using LR on TFD with RWS and NRWS, for different values of β. Continuous lines represent the quantities on the train set and dashed lines the ones on validation.

L^2 regularization behaves as we expect on RWS, where we see that the regularization improves the performance of the algorithm, with stronger regularization being better up to a given value ($\beta = 0.05$). However for NRWS we see a different behavior, compared to the non-geometric counterpart. We see that a small amount of L^2 regularization ($\beta = 0.0005, 0.001$) slowly derails the algorithm from a better optimum to a worse convergence minimum, while for larger amounts of regularization ($\beta \geq 0.05$), the algorithm converges almost instantaneously to a sub-optimal convergence value.

Therefore, in the case of the TFD, both LR methods fail to improve the performance of the NRWS.

Studying the behavior of LR methods on FashionMNIST in Figure 6.9, we notice that neither regularization method, for any tested value for β, improves

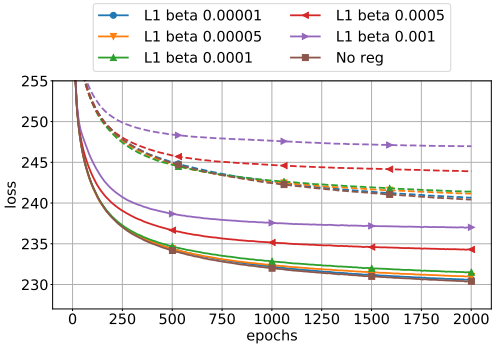
on either RWS or NRWS. We see a clear trend for both L^1 and L^2 that the larger β we take, the more the training underperforms to the non-regularized version.

We compared the final minima after the convergence of the algorithms, with the best β-s for each regularization type and without any regularization and we present these results in Table 6.3. We used the best values for the learning rate and damping factor as in the previous experiments. The findings in the table corroborate the results from the previous figures, as we find again that the tested L^p regularizers do not work for the geometric algorithm NRWS as the non-regularized versions of the algorithms come out on top for both datasets. In addition, for the RWS even if regularizers improve on the results, as in the case of the TFD, they do so only marginally.

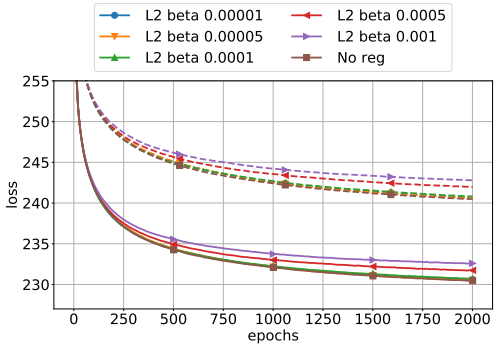
DS	**ALG**	**Epochs**	**Regularizer**	β	**LL**
FashionMNIST	RWS	2000	-	-	**-239.39**
			L^1	0.00001	-239.29
			L^2	0.00001	-239.34
		5000	-	-	**-237.47**
			L^1	0.00001	-237.61
			L^2	0.00001	-237.63
	NRWS	2000	-	-	**-236.49**
			L^1	0.00001	-236.65
			L^2	0.00001	-236.51
TFD	RWS	2000	-	-	-373.87
			L^1	0.01	-372.89
			L^2	0.05	**-370.39**
		5000	-	-	-372.63
			L^1	0.01	-372.9
			L^2	0.05	**-370.3**
	NRWS	2000	-	-	**-370.32**
			L^1	0.01	-372.5
			L^2	0.1	-370.9

Table 6.3: Importance Sampling estimation of the log-likelihood on the test set with 10,000 samples for different algorithms after training till convergence ($5,000$ epochs for RWS and $2,000$ epochs for NRWS) [β - the rate of regularization; LL - log-likelihood].

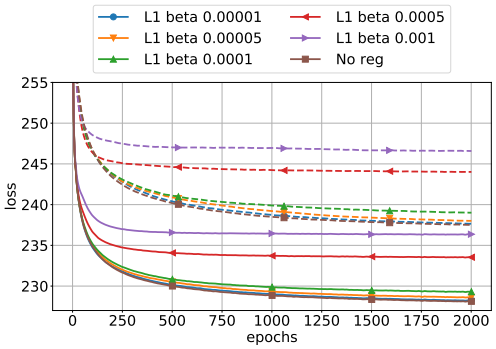
Based on the previous results, the following questions arise: Why is the performance of the LR less effective for the NRWS in the case of TFD and why do they only affect the learning on FashionMNIST in a negative way?



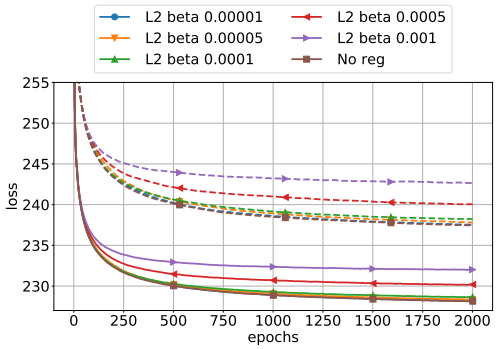
(a) L^1 on FashionMNIST with RWS.



(b) L^2 on FashionMNIST with RWS.



(c) L^1 on FashionMNIST with NRWS.



(d) L^2 on FashionMNIST with NRWS.

Figure 6.9: Training curves representing the loss using LR on FashionMNIST with RWS and NRWS, for different values of β. Continuous lines represent the quantities on the train set and dashed lines the ones on validation.

Our hypothesis lies in how the regularizers work in the context of an HM and what specifics the data of FashionMNIST has. As we have mentioned previously, LR works by enforcing a constraint on the magnitude of weights of the network, ensuring that the models do not learn extreme values for their parameters.

The fact that regularized algorithms are discouraged from learning extreme values for the parameters explains why regularizers have a detrimental effect on the performance of the training algorithms on FashionMNIST. If we look at any sample from the dataset (samples shown in Figure 6.10 and the Appendix A, Figure A.1), we notice that every image has a black background. A black value is represented as 0 (or analogously -1 for a tanh activation), to

sample a 0 from a Bernoulli distribution one needs the output of the sigmoid to be close to 0, which means that the input of the sigmoid has to approach $-\infty$ as

$$\lim_{x \to -\infty} s(x) = 0 \,,$$

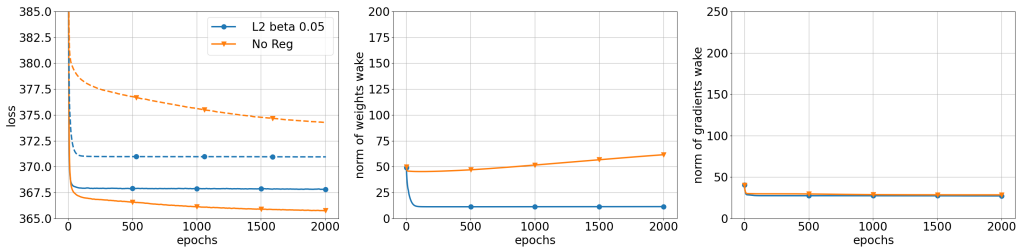
therefore, the model is required to learn extreme values for its parameters to generate constant black backgrounds. Hence, regularizing the model with L^1 or L^2 for FashionMNIST is not beneficial to the learning, as it is discouraged to learn the extreme values associated with the background of the images.

Figure 6.10: Sample images from the FashionMNIST dataset.

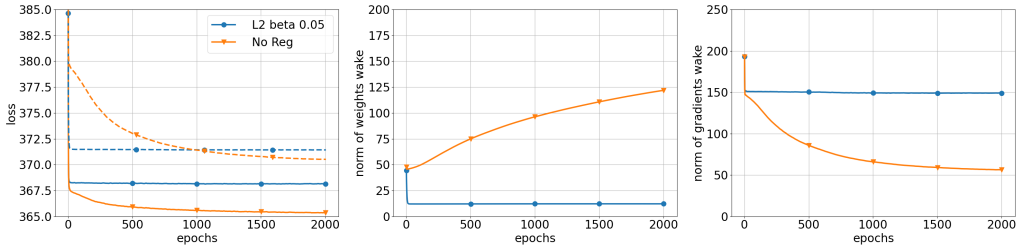
The success of NRWS over RWS is partially based on the fact that thanks to the NG, the size of the gradients is modified depending on the metric, and it can grow in magnitude, depending on their direction, compared to the gradients of RWS (Amari, 1998). With larger values for gradients, the size of the weights can grow faster, and as such the algorithm can explore a much larger area in the parameter space, for a fixed amount of steps, so that it can converge faster to better local minima of the optimization. The LR methods, however, set constraints on the size of the weights, by definition, which limits the area in the parameter space where the training algorithm explores, which area consists of less extreme values. In this sense, we hypothesize that the LRs for HMs actively work against the benefits of the NRWS, as it limits how large the weights of the network can grow. Indeed, we can see this behavior in the case of the black values of the FashionMNIST data, where the NRWS is much better at learning the extreme values for the weights, representing the black values, and achieves smaller losses than using LR methods.

We illustrate how the norms of the weights and gradients change throughout the training of the wake phase[7] with a regularized and a non-regularized

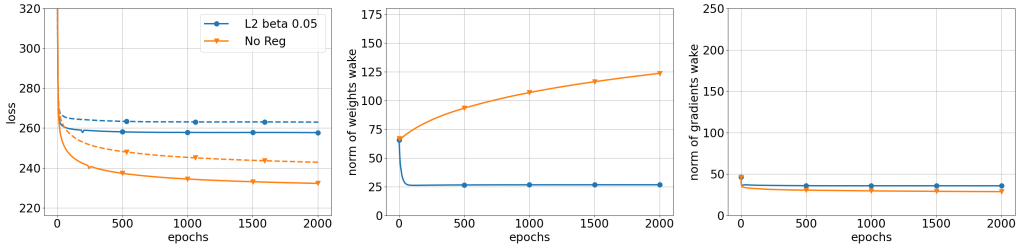
[7]We have verified that the curves of the loss and norms of the weights and gradients



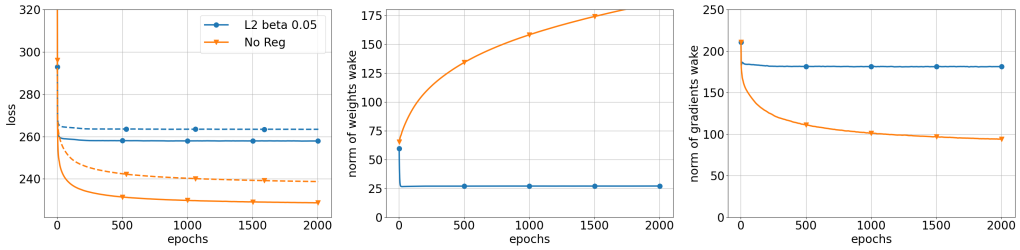
(a) Loss, weights and gradients of RWS with LR for TFD.



(b) Loss, weights and gradients of NRWS with LR for TFD.



(c) Loss, weights and gradients of RWS with LR for FashionMNIST.



(d) Loss, weights and gradients of NRWS with LR for FashionMNIST.

Figure 6.11: The evolution of the norms of the weights and gradients on TFD and FashionMNIST with RWS and NRWS: (left) evolution of the loss during training, validation set with dashed line, train with solid line; (center) norm of weights; (right) norm of gradients; in epochs.

algorithm for both RWS and NRWS in Figure 6.11. We see that the losses for both methods and both datasets, the regularized versions in the first few epochs get reduced to a minimal value, and stay there until the end of the experiment. In all cases, except for the RWS on TFD, this minimal convergence value on the validation set is larger, than the non-regularized version. Simultaneously, the gradient becomes near constant but not 0 for the entire experiment length. However, we see that for the non-regularized NRWS, which outperforms the other methods by achieving a smaller value for the loss, the norm of the weights takes a much higher value. We can also notice that the norm of the gradients in this case decreases and eventually, it will level out. The non-regularized RWS behaves similarly to the non-regularized NRWS in loss and norms of gradient and weights, only it is slower, as we know from previous experiments (see Section 5.3.7) to reach the same values for the measured quantities. Furthermore, the behavior of the norms of the weights shown in the plots confirms our hypothesis about FashionMNIST. We can observe that the norms of the weights of the non-regularized algorithm in Figures 6.11c and 6.11d reach much higher values than for the regularized version, implying that the non-regularized algorithms learn larger weights.

We have seen that using the LR does not work for all datasets, as well as, LR does not help the convergence of the NRWS. However, this does not mean that we cannot regularize geometric algorithms on HMs. In the following section, we explore some possibilities of regularizing statistical models using IG concepts.

6.3 Geometrical Regularization over Statistical Models

In the case of a statistical model p_θ, we need a different measure of complexity, than the norm of the parameters θ to differentiate between simple and complex models (Occam's Razor). A good geometric principle is to find a dissimilarity measure that is independent of the parametrization θ. One such measure of complexity could be the entropy of p_θ, as it is most commonly associated with a state of disorder, randomness, or uncertainty.

Another approach to measuring the complexity of a model p_θ is to compare it to some fixed reference distribution p_0, one which is a standard or a non-informative distribution, and preferably independent of the parametrization θ. In the case of SBNs, using Bernoulli distributions for the random variables, p_0 could be chosen to be the uniform distribution, for which the weights of

during the sleep phase behave analogously to the curves of the wake phase.

the parametrization are $\theta_i = 0$. As a reference distribution p_0 is independent of the training data and it is always returning probability $1/2$ independently from input.

Given a reference distribution p_0, there are multiple ways to regularize our distribution p_θ in relation to it, which usually requires that we have to define some form of dissimilarity measure, a distance or more generally a divergence, between the two distributions, where a smaller value for such a measure relates to a higher similarity between the two. Regularization then is understood as encouraging similarity between the two distributions. Some of these measures we could use and minimize are the following:

1. The KL divergence[8] between the distribution p and the reference p_0: The KL divergence has the advantage that it is independent of the parametrization of p_θ and p_0. Indeed, we can show that using the KL divergence as a measure equals maximizing the entropy of the statistical model p_θ if p_0 is the uniform distribution over binary variables (see Subsection 6.3.1 for more details).

2. The length of a geodesic defined on the probability manifold $\mathcal{M}$ connecting the distributions p_θ and p_0: Minimizing the length of geodesic connecting p_0 and p_θ is not a trivial task, since calculating the geodesic itself is generally computationally expensive (see Section 6.1.1 for more details). Equivalently, one could minimize the norm of the velocity vector associated with the geodesic, however, this requires calculating the logarithmic map $Log_{p_0}(p_\theta)$, which is also not easy to compute in the general case. Such a method could be possible for proper choices of the geodesic (not necessarily using the Levi-Civita connection) for some statistical models, such as the ones in the Exponential family, but as the HM is not part of any of those. Applying this type of regularization is not trivial, and the exploration of such techniques could be explored in future works[9].

6.3.1 Entropy Regularization of a Helmholtz Machine

In this section, we calculate the KL divergence between the generative distribution p_θ and a uniform one p_0 with fixed parameters. Notice that the uniform distribution p_0 has maximum entropy, and is defined for finite sample spaces. If we take the binary case where our p_θ are Bernoulli distributions

[8] For more information about the KL divergence, see Section 3.4.

[9] Geodesics, velocity vectors, the exponential family and logarithmic maps are explained in Chapter 3.

and p_0 is the uniform distribution, we get

$$\begin{aligned}
\arg\min_\theta \ \mathrm{D}_{KL}\left[p_\theta(x,h)||p_0(x,h)\right] &= \arg\min_\theta \ \int p_\theta(x,h)\ln\frac{p_\theta(x,h)}{p_0(x,h)}\mathrm{d}x\mathrm{d}h \\
&= \arg\min_\theta \ \int p_\theta(x,h)\ln p_\theta(x,h)\mathrm{d}x\mathrm{d}h \\
&\qquad - \arg\min_\theta \ \underbrace{\int p_\theta(x,h)\ln p_0(x,h)\mathrm{d}x\mathrm{d}h}_{\text{const. w.r.t. }\theta} \\
&= \arg\min_\theta \ \int p_\theta(x,h)\ln p_\theta(x,h)\mathrm{d}x\mathrm{d}h \\
&= -\arg\min_\theta \ \mathcal{H}_p[x,h] = \arg\max_\theta \ \mathcal{H}_p[x,h] \ .
\end{aligned} \tag{6.15}$$

We see that by maximizing the entropy $\mathcal{H}$ of the distribution p_θ we minimize its divergence from p_0.

Methods based on the maximization of the entropy are well known in the literature, they have been introduced by Jaynes (1957) in the form of the Maximum Entropy Principle and have had a rich research history since (Shore and Johnson, 1980; Guiasu and Shenitzer, 1985; Fornalski et al., 2010). The Principle of Maximum Entropy is based on the premise that when estimating the probability distribution, one should select the distribution that leaves the largest remaining uncertainty (i.e., the maximum entropy) consistent with your constraints. This choice maximizes the probability that we do not introduce any additional assumptions or biases into the estimation (Lloyd and Penfield, 2003). In simpler terms, the principle states that when faced with multiple probability distributions that all represent the same data points from a dataset similarly well, the best choice is the distribution that has the maximum entropy, as that one is the most likely to generalize well on new/unseen data (Ma, 2021). Regularizing the entropy is very popular in Reinforcement Learning (Audiffren et al., 2015; Haarnoja, 2018), in particular, the method known as Soft Actor-Critic (Haarnoja et al., 2018) uses the concept of maximizing the entropy in training to balance the exploration-exploitation trade-off in continuous exploration spaces.

In the case of the HM, we can use entropy maximization to increase the generalization capability of the learned model. By choosing as a reference distribution the uniform distribution, in the light of Eq. (6.15), we can estimate the entropy for both the generative and recognition distributions of the HM and add it as a regularization term to the loss during training. These extra terms force the training to converge towards minima with higher entropy. Therefore, following the Maximum Entropy Principle, the model

should be able to generalize better. The entropies for the HMs are defined as

$$\begin{aligned}\mathcal{H}_p &= E_{p_\theta(x,h)}\left[\ln p_\theta\left(x,h\right)\right] \text{ and}\\ \mathcal{H}_q &= E_{q_\phi(h|x)}\left[\ln q_\phi\left(h|x\right)\right] \ .\end{aligned}$$

In the case of the generative distribution p, we can rewrite the KL divergence from Eq. (6.15) as a sum of KL divergences following the chain rule (Braverman, 2011)

$$\begin{aligned}&\mathrm{D}_{KL}\left[p(x,h)||p_0(x,h)\right] = \\ &\quad = \mathrm{D}_{KL}\left[p(x|h^1)||p_0(x|h^{1)}\right] + \cdots + \mathrm{D}_{KL}\left[p(h^M)||p_0(h^M)\right] \ .\end{aligned} \tag{6.16}$$

Similarly, following (6.15), the entropy of p can be rewritten as a sum of layer-wise entropies

$$\mathcal{H}_p[x,h] = \mathcal{H}_p[x|h^1] + \cdots + \mathcal{H}_p[h^M] \ .$$

Because of this decomposition, by introducing regularization to the loss function, we still conserve the advantage of WS based methods, that the gradients can be computed and updated per-layer, a very important property for the efficient use of HMs[10]. Hence, we can write the conditional entropies for a layer i as

$$\begin{aligned}\mathcal{H}_p\left[h^i|h^{i+1}\right] &= E_{p_\theta(h^i|h^{i+1})}\left[\ln p_\theta\left(h^i|h^{i+1}\right)\right] \text{ and}\\ \mathcal{H}_q\left[h^i|h^{i-1}\right] &= E_{q_\phi(h^i|h^{i-1})}\left[\ln q_\phi\left(h^i|h^{i-1}\right)\right] \ .\end{aligned} \tag{6.17}$$

In practice, the exact quantities are replaced by empirical estimations, which allow the computation of the gradients by Monte Carlo sampling. We estimate the gradients w.r.t. the expectations in Eq.(6.17) of H_p and H_q by sampling from the parent layers

$$\begin{aligned}\nabla_\theta H_p &= \frac{1}{B\cdot S}\sum_{k=1}^{S\cdot B}\frac{\partial \ln p_\theta(x_{(k)},h_{(k)})}{\partial\theta} \ \text{ with } \ x_{(k)},h_{(k)}\sim p_\theta(x,h) \text{ and}\\ \nabla_\phi H_q &= \frac{1}{B\cdot S}\sum_{k=1}^{S\cdot B}\frac{\partial \ln q_\phi(h_{(k)}|x_{(k)})}{\partial\phi} \ \text{ with } \ x_{(k)},h_{(k)}\sim q_\phi(x,h) \ .\end{aligned}$$

Notice that the computations of the gradients of H are almost identical to the computations we use for the gradients of the loss L in Eqs. (2.22) and (2.23) only with samples coming from the opposite distributions. While the

[10] Although we only showed here the derivation of the entropy of $\mathcal{H}_p$, the derivations are analogous for the $\mathcal{H}_q$.

samples for calculating the losses L per layer come from either the data distribution p^* or the opposite distribution (p for the recognition and q for the generative), the sampling needed for computing H-s layerwise come from the same distributions, therefore the entropies are independent of the data.

By adding the gradients of the entropies to the update rule of the training algorithm, we can regularize the statistical model to prefer higher entropy distributions while learning p and q. We will refer to this method of regularization as Entropy Regularizer (ER). The final update rules for the parameters of the HM for the NRWS with ER can then be formulated as

$$\begin{aligned}\theta_{t+1} &= \theta_t - \eta \widetilde{F}_p^{-1} \left[\beta \nabla_\theta H_p + \frac{1}{B} \sum_{r=1}^{B} \sum_{k=1}^{S} \tilde{\omega}_k \nabla_\theta L_{p,(k,r)} \right] \text{ and} \\ \phi_{t+1} &= \phi_t - \eta \widetilde{F}_q^{-1} \left[\beta \nabla_\phi H_q + \frac{1}{2B} \sum_{r=1}^{B} \sum_{k=1}^{S} \frac{1}{S} \nabla_\phi L^s_{q,(k,r)} + \tilde{\omega}_k \nabla_\phi L^w_{q,(k,r)} \right] ,\end{aligned} \tag{6.18}$$

where we can control the strength of its effect with a parameter β.

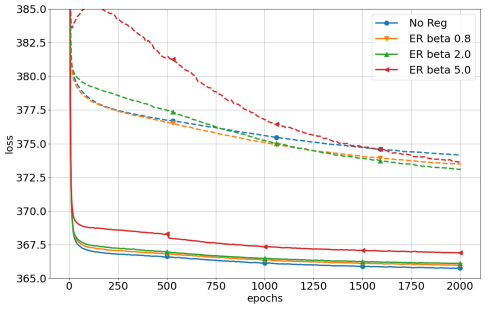
6.3.2 Experiments with Entropy Regularization

We tested ER on both FashionMNIST and TFD in a similar experimental setting, as in Subsections 5.3.7 and 6.2.2. We present the results in the plots with algorithms running $2,000$ epochs for training both for RWS and NRWS. For the final importance sampling values which we present in the tables, we also include RWS ran for $5,000$ epochs, to compensate for the longer per-epoch run-time of NRWS[11]. In all our experiments with ER we use a warm-up period of 50 epochs, which we have shown to be appropriate in the Appendix C.3. Similarly to the LR methods, in the case of the ER the added time from the computation of the gradient of the entropy is negligible, as it has the same complexity as the computation of the gradient of the loss, therefore we report the experiments only in training epochs.

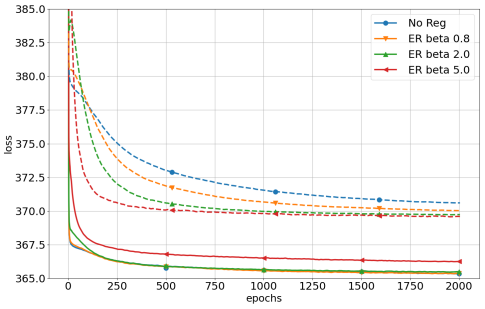
In Figure 6.12 we present the change in the loss of HM models trained with RWS and NRWS on TFD and FashionMNIST for different values of β for the ER (0, 0.8, 2.0, 5.0).

By analyzing the plots, we immediately notice that the ER for certain values of β improves on the performance of both RWS and NRWS in convergence rate and final minimum, compared to the non-regularized version,

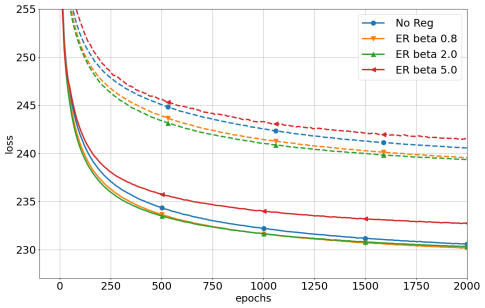
[11] Based on the experiments in Section 5.3.7, the run-time of NRWS is only 1.4 times slower than the one of RWS for all datasets, which means that $\sim 3,000$ epochs would be enough to compensate for the time difference. However, the latter did not converge in that period, so we ran it till $5,000$ to compare the final convergence values.



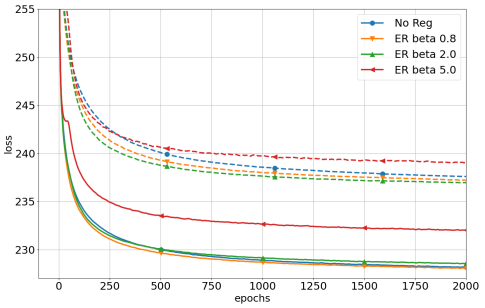
(a) ER on TFD for RWS.



(b) ER on TFD for NRWS.



(c) ER on FashionMNIST for RWS.



(d) ER on FashionMNIST for NRWS.

Figure 6.12: Loss of the training algorithms using ER on TFD and FashionMNIST for RWS and NRWS, for $\beta \in \{0, 0.8, 2.0, 5.0\}$. Continuous lines represent the quantities on the train set and dashed lines the ones on validation.

commonly on both the validation and train set. As we see in Figures 6.12c and 6.12d, the ER also improves on the FashionMNIST dataset, on which the LR methods failed to improve (see Section 6.2.2). Looking at the effect of the different β-s on the loss, we conclude that the most favorable value is $\beta = 2.0$ among those we tested, as smaller values lead to less performance, and larger values can lead to sub-optimal behavior (Figure 6.12a) or worse performance than the non-regularized variant (Figure 6.12d). In the case of the NRWS the ER seems to have immediate benefits on the training, while for RWS we see less improvement at the beginning of the training for TFD in the convergence rate, but ultimately the curves of the regularized algorithms overtake the non-regularized one.

An important difference to mention, when comparing the results of the ER with the ones done with L^1 and L^2 regularizers, is that the former does not seem to improve just on the generalization gap between the train and validation

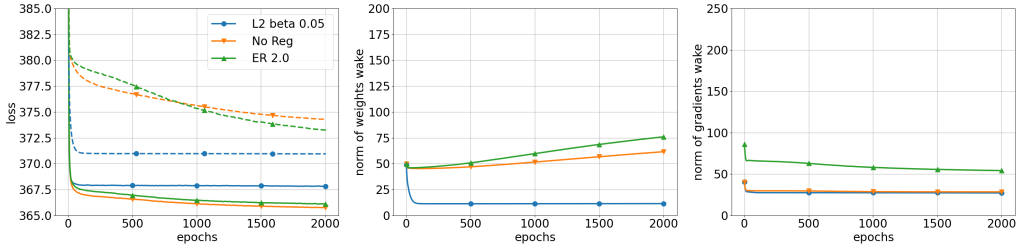
curves of the experiments, but they improve on the overall performance of the algorithms, by achieving better performance on the train set as well, not just the validation.

We analyzed the effect the ER has on the weights and gradients during training for the TFD and FashionMNIST when using it with RWS and NRWS in Figure 6.13. We see that, in fact, ER has the opposite effect on the weights and gradients of the model than the L^2 regularizer, while also outperforming the latter in reached minimum loss. We observe that the ER encourages larger norms for the weights, which, as we speculated before in Section 6.2.2 is likely to result in more extreme values, which leads to better minima of the loss. We can also notice for both datasets and both RWS and NRWS when using ER, that norms of the gradients converge to some minimum, after which they stay constant, similarly to the non-regularized version, but with a higher norm, which likely contributes to the higher norms for the weights.

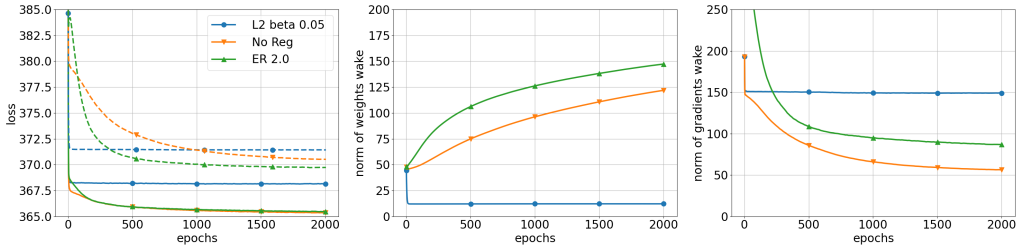
DS	**ALG**	**Epochs**	**Regularizer**	β	**LL**
FashionMNIST	RWS	2000	-	-	-239.17
			ER	2.0	**-238.71**
		5000	-	-	–237.52
			ER	2.0	**-237.04**
	NRWS	2000	-	-	-236.49
			ER	2.0	**-236.05**
TFD	RWS	2000	-	-	-373.9
			ER	2.0	**-373.18**
		5000	-	-	-372.56
			ER	2.0	**-371.31**
	NRWS	2000	-	-	-370.32
			ER	2.0	**-369.61**

Table 6.4: Importance Sampling estimation of the log-likelihood (**LL**) on the test set with 10,000 samples for different algorithms after training till convergence with SGD, where β is the regularization parameter.

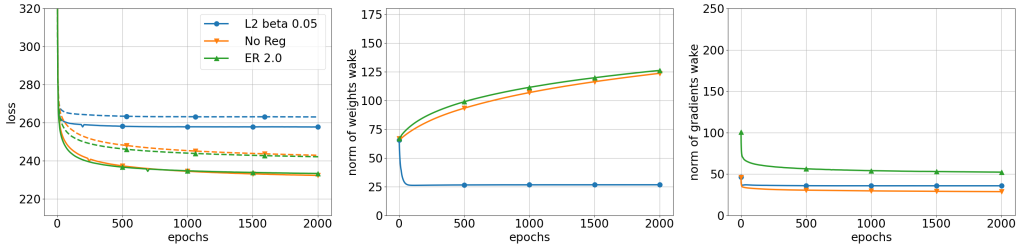
Finally, in Table 6.4 we present the best performance of the algorithms measured in the loss with and without ER, where we see the trends seen in the plots confirmed by Importance Sampling estimation of the loss, with $10,000$ samples. We see that indeed, for both datasets and with both training algorithms RWS and NRWS, using ER improves on the final optimization minimum. We notice in fact even when we run the RWS for $5,000$ epochs, the final log-likelihoods are lower than the ones for the NRWS. These results are strengthened by what we learned from the curves in Figure 6.12, that the



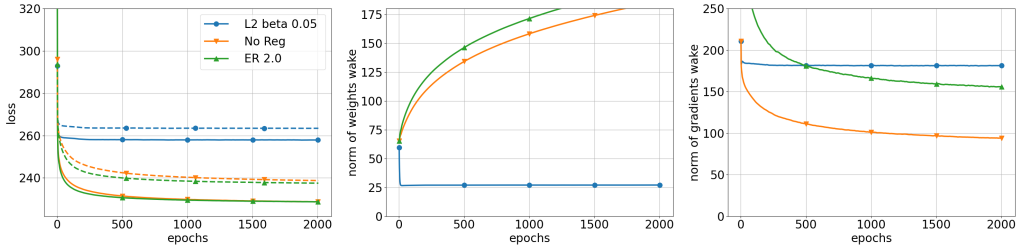
(a) Loss, weights and gradients of RWS with ER for TFD.



(b) Loss, weights and gradients of NRWS with ER for TFD.



(c) Loss, weights and gradients of RWS with ER for FashionMNIST.



(d) Loss, weights and gradients of NRWS with ER for FashionMNIST.

Figure 6.13: The evolution of weights and gradients in epochs on TFD and FashionMNIST with RWS and NRWS: (left) evolution of the loss, validation set is represented with a dashed line, train with a solid line; (center) norm of weights; (right) norm of gradients.

DS	ALG	Regularizer and AG	LL
MNIST	RWS	-	-87.36
	NRWS	-	-84.91
		ER $\beta = 0.5$, Nesterov	**-84.88**
FashionMNIST	RWS	-	-237.52
	NRWS	-	-235.65
		ER $\beta = 0.5$, Nesterov	**-235.43**
TFD	RWS	-	-372.56
	NRWS	-	-370.32
		ER $\beta = 2.0$, Momentum	**-369.61**

Table 6.5: Importance Sampling estimation of the log-likelihood (**LL**) on the test set with 10,000 samples for different algorithms after training till convergence, where β is the regularization parameter. The values for RWS and the plainNRWS are taken from the tables of Chapter 5.

ER improves on the non-regularized algorithms in both convergence rate and minima of the loss reached.

6.4 Regularized Natural Reweighted Wake-Sleep

We have seen in Section 6.1 that we can improve the convergence rate of NRWS by using AG methods. However, we noticed that in some cases there is a need for regularizing the model. In Section 6.2 we showed that regularizing the HM with LR techniques leads to some advantages in certain cases (TFD) but in general, it cannot be applied to learning all datasets. Furthermore, we discussed how LR methods are geometrically not sound for using them in concordance with NRWS. Therefore, in Subsection 6.3.1 we showed how the ER can be used as a geometrically suitable regularization method, to improve the performance of NRWS on all datasets. In this section, we conclude the chapter by combining all the improvements mentioned above (AG and ER) to compare to the state-of-the-art results of NRWS presented in the previous chapter.

In Table 6.5 we present the results on the MNIST, FashionMNIST and TFD datasets using the best settings for gradient acceleration and regularization, for each dataset. Notice that for all datasets, the version using ER and AG outperforms the plain NRWS. For the MNIST and FashionMNIST datasets, we see only marginal improvements, however, if we consider that

both improvement methods have been shown in the previous sections to improve the convergence rate of the algorithms, with no noticeable computational time-penalty, we think they are improvements nonetheless as they converge faster to better results, even if we do not train until convergence. In the case of TFD, we see a larger improvement, compared to the other datasets, which also confirms our hypotheses about the difference between black-background images and more neutral images from the previous Subsection 6.2.2.

We conclude that using both AG and ER methods is almost always beneficial to training. Since both methods can improve on both RWS and NRWS methods, with no noticeable drawbacks it is an obvious choice for any training method on an HM.

Chapter 7

Applications to Natural Images

In this chapter, we focus on the flexibility and scalability of Natural Reweighted Wake-Sleep, in particular, we focus on types of random variables that can be modeled with Helmholtz Machines, and on the size of the data in terms of image resolution that the model can process.

So far we modeled image data with binary variables, when continuous variables would be a better representation, therefore, we adapt the NRWS to represent non-binary data. We compute the FIM associated with Gaussian random variables for the HM, which we show has a fine-grained block-diagonal structure comparable to the one used for binary variables.

Later, we present how convolutional networks can also be used together with the NRWS to learn datasets with higher resolution images, which would be computationally demanding when using only dense layers, because of the increased size of the blocks of the FIM.

7.1 The Normal Distribution in Helmholtz Machines

In their work, Dayan and Hinton (1996) proposed multiple variants of the HM, in addition to what we have used thus far, such as using different network architectures, sampling methods, activation functions or distributions with the model. One of the proposed options is to use Normal distributions to represent continuous data, instead of the Bernoulli we have used up until now, which enables us to use continuous random variables for the HM, which is particularly useful for learning and representing natural images.

Natural images are typically represented as 256 discrete gray or color levels, in machine learning applications we usually rescale the discrete values equally spaced onto the $[0, 1]$ interval. These discrete values are typically then treated as continuous values, and generally, we use a continuous distribution, e.g., Gaussian, to model them, up to truncation for values outside the interval of interest. While previously data augmentation techniques and sampling were needed to learn continuous data represented through the mean parameters of

Bernoulli distributions[1], with a Normal distribution we can learn the values directly.

When we use a Normal distribution in a HM the definition of the loss function of the NRWS (log-likelihood of wake, q-wake and sleep) do not change, however, the per-layer gradients for the parameters of the distribution do. Recall the loss of a single layer of the HM as a conditional log-likelihood $\ln p(h^i|h^{i+1};\theta)$, we can substitute p with the pdf of a conditional Normal distribution $\mathcal{N}(\mu,\sigma)$ having mean μ and standard deviation σ. More precisely[2] we consider the parameters of node j as $\theta^j = (W_\mu^{i,j}, W_\sigma^{i,j})$ with $\mu^{i,j} = \left(W_\mu^{i,j}\right)^\top h^{i+1}$ and $\sigma^{i,j} = \exp\left((W_\sigma^{i,j})^\top h^{i+1}\right)$, i.e.

$$\begin{aligned} p_\theta\left(h^{i,j}|h^{i+1}\right) &= \mathcal{N}(h^{i,j}|\mu^{i,j},\sigma^{i,j}) \\ &= \frac{1}{\sqrt{2\pi\left(\sigma^{i,j}\right)^2}} \exp\left(-\frac{\left(h^{i,j}-\mu^{i,j}\right)^2}{2\left(\sigma^{i,j}\right)^2}\right) \\ &= \frac{1}{\exp\left(\left(W_\sigma^{i,j}\right)^\top h^{i+1}\right)\sqrt{2\pi}} \exp\left(-\frac{\left(h^{i,j}-\left(W_\mu^{i,j}\right)^\top h^{i+1}\right)^2}{2\exp\left(\left(W_\sigma^{i,j}\right)^\top h^{i+1}\right)^2}\right), \end{aligned} \tag{7.1}$$

where exp is the exponential function $\exp(x) = e^x$. The exp function in the standard deviation ensures that σ has only non-negative values, which is needed by definition[3].

Remark 7
In this section, we will do all derivations for Gaussian random variables only for the generation network p of the HM for simplicity, however, every derivation applies analogously to the recognition network q as well, which we skip for brevity.

We can calculate the gradients of the log-likelihood, first with respect to

[1]For more details about data augmentation for binary data see Appendix D.

[2]We make the same assumption as we made in Note 4, that there is an implicit bias term as in $W \cdot x + b$, but we shorten it for brevity.

[3]When putting in practice the above derivations, we add a constant ϵ to the σ, where $0 < \epsilon \ll 1$, which is a common technique in machine learning, to ensure the σ is never 0 to avoid numerical errors (i.e., the denominator of the first term in Eq. 7.1), however, we ignore this term in the formulae.

$W_{\mu}^{i,j}$ as

$$\begin{aligned}
\frac{\partial}{\partial W_{\mu}^{i,j}} \ln p\left(h^{i,j}|h^{i+1}\right) &= \frac{\partial}{\partial W_{\mu}^{i,j}} \ln \frac{1}{\sigma^{i,j}\sqrt{2\pi}} \exp\left(-\frac{\left(h^{i,j}-\left(W_{\mu}^{i,j}\right)^{\top}h^{i+1}\right)^{2}}{2\left(\sigma^{i,j}\right)^{2}}\right) \\
&= \underbrace{\frac{\partial}{\partial W_{\mu}^{i,j}} \ln \frac{1}{\sigma^{i,j}\sqrt{2\pi}}}_{0} - \frac{\partial}{\partial W_{\mu}^{i,j}} \frac{\left(h^{i,j}-\left(W_{\mu}^{i,j}\right)^{\top}h^{i+1}\right)^{2}}{2\left(\sigma^{i,j}\right)^{2}} \\
&= -\frac{1}{2\left(\sigma^{i,j}\right)^{2}} \frac{\partial}{\partial W_{\mu}^{i,j}} \left(h^{i,j}-\left(W_{\mu}^{i,j}\right)^{\top}h^{i+1}\right)^{2} \\
&= -\frac{\left(h^{i,j}-\left(W_{\mu}^{i,j}\right)^{\top}h^{i+1}\right)}{\left(\sigma^{i,j}\right)^{2}} \frac{\partial}{\partial W_{\mu}^{i,j}} \left(h^{i,j}-\left(W_{\mu}^{i,j}\right)^{\top}h^{i+1}\right) \\
&= \frac{\left(h^{i,j}-\mu^{i,j}\right)}{\left(\sigma^{i,j}\right)^{2}} h^{i+1} \; ;
\end{aligned} \tag{7.2}$$

then with respect to $W_{\sigma}^{i,j}$ as

$$\begin{aligned}
&\frac{\partial}{\partial W_{\sigma}^{i,j}} \ln p\left(h^{i,j}|h^{i+1}\right) = \\
&= \frac{\partial}{\partial W_{\sigma}^{i,j}} \ln \frac{1}{\exp\left(\left(W_{\sigma}^{i,j}\right)^{\top}h^{i+1}\right)\sqrt{2\pi}} \exp\left(-\frac{\left(h^{i,j}-\mu^{i,j}\right)^{2}}{2\exp\left(\left(W_{\sigma}^{i,j}\right)^{\top}h^{i+1}\right)^{2}}\right) \\
&= -\underbrace{\frac{\partial}{\partial W_{\sigma}^{i,j}} \ln\sqrt{2\pi}}_{0} - \frac{\partial}{\partial W_{\sigma}^{i,j}} \ln\exp\left(\left(W_{\sigma}^{i,j}\right)^{\top}h^{i+1}\right) - \\
&\qquad - \frac{\partial}{\partial W_{\sigma}^{i,j}} \frac{\left(h^{i,j}-\mu^{i,j}\right)^{2}}{2\exp\left(\left(W_{\sigma}^{i,j}\right)^{\top}h^{i+1}\right)^{2}} \\
&= -h^{i+1} - \frac{\left(h^{i,j}-\mu^{i,j}\right)^{2}}{2} \frac{\partial}{\partial W_{\sigma}^{i,j}} \frac{1}{\exp\left(\left(W_{\sigma}^{i,j}\right)^{\top}h^{i+1}\right)^{2}} \\
&= \left(\frac{\left(h^{i,j}-\mu^{i,j}\right)^{2}}{\left(\sigma^{i,j}\right)^{2}} - 1\right) h^{i+1} \; .
\end{aligned} \tag{7.3}$$

The computation of the above gradients enables us to train the HM with Gaussian distributions using the non-geometrical algorithms, such as: WS,

RWS and BiHM. However, we are interested in training with the NRWS, which is based on the NG and requires the computation of the FIM and its inverse. We recall from Chapter 4 in Eq.(4.3) our previous result that the FIM for the HM can be written as a block-diagonal matrix

$$\mathcal{F}(\theta) = -\mathbb{E}_{p_\theta(x,h)}\left[\text{block-diag}\left(\nabla^2_\theta \ln p\left(h^{i,j}|h^{i+1};\theta^{i,j}\right)\right)_{i\in[0,M-1],j\in[0,l_i]}\right] , \quad (7.4)$$

where $\theta^{i,j}$ are the weights of network parameterizing the random variable $h^{i,j}$ given its parents h^{i+1}.

Theorem 2
The blocks of the FIM for the HM using Normal distributions for the random variable $\mu^{i,j}$, the mean of a node j in layer i, parameterized by $W^{i,j}_\mu$ are

$$\mathcal{F}^{i,j}_{p,\mu} = \mathbb{E}_{p(x,h)}\left[\frac{1}{\left(\sigma^{i,j}\right)^2} h^{i+1}\left(h^{i+1}\right)^\top\right] . \quad (7.5)$$

For the random variable $\sigma^{i,j}$, the standard deviation of node j in layer i, parameterized by $W^{i,j}_\sigma$, the blocks of the FIM can be defined as

$$\mathcal{F}^{i,j}_{p,\sigma} = \mathbb{E}_{p(x,h)}\left[\frac{2\left(h^{i,j}-\mu^{i,j}\right)^2}{\left(\sigma^{i,j}\right)^2} h^{i+1}\left(h^{i+1}\right)^\top\right] . \quad (7.6)$$

The off-diagonal blocks of the FIM, using the derivatives of both μ and σ, are

$$\begin{aligned}\mathcal{F}^{i,j}_{p,\mu-\sigma} &= \mathbb{E}_{p(x,h)}\left[\frac{\partial^2}{\partial W^{i,j}_\mu \partial\left(W^{i,j}_\sigma\right)^\top} \ln p\left(h^{i,j}|h^{i+1}\right)\right] \\ &= \mathbb{E}_{p(x,h)}\left[-\frac{2\left(h^{i,j}-\mu^{i,j}\right)}{\left(\sigma^{i,j}\right)^2} h^{i+1}\left(h^{i+1}\right)^\top\right] = 0 .\end{aligned} \quad (7.7)$$

From the above theorem, it follows that the FIM using Gaussian random variables has a similar fine-grained block diagonal structure as we had with Bernoulli distributions, which is one block per neuron j. In this case, the blocks are twice as large because the distribution has two parameters (μ and σ). However, following Eq. (7.7) the block of the neuron j also has two diagonal blocks, one for μ and one for σ, resulting in double the number of weights and blocks for a fixed network topology, compared to the Binary case. A visual representation of a FIM for a single layer of the HM is shown in Figure 7.1.

In the following two subsections, 7.1.1 and 7.1.2, we will prove this theorem.

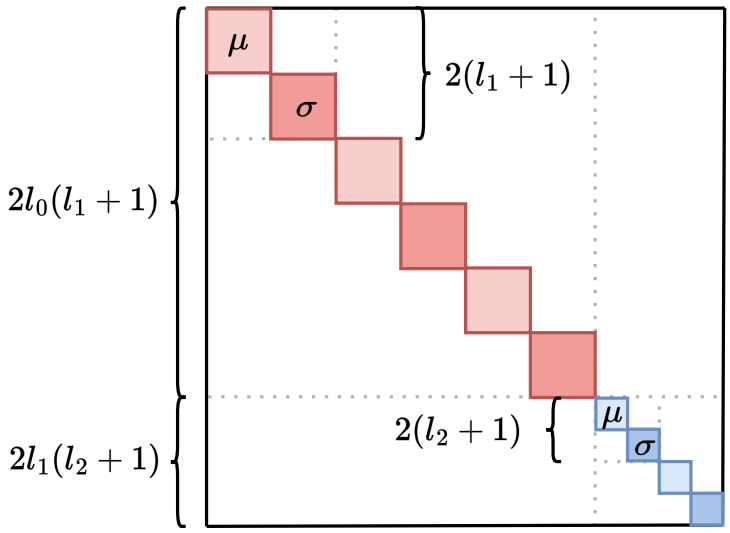


Figure 7.1: Graphical representation of the Fisher information matrix for a Network with 3-2 nodes and the prior using Gaussian random variables. The gray lines identify the blocks associated with the layers of the network and a single node of the network. The matrix admits a fine-grained block-diagonal structure with two blocks for each node. The blocks are ordered from the bottom layer to the top starting from the upper left.

7.1.1 The Hessian of the Normal Distribution

To prove Theorem 2, we have to first calculate the Hessian of the joint log-likelihood defined in Eq. (7.1). Let us calculate the derivatives w.r.t.

$$\frac{\partial^2}{\partial W_\mu^{i,j} \partial \left(W_\mu^{i,j}\right)^\top}, \quad \frac{\partial^2}{\partial W_\sigma^{i,j} \partial \left(W_\sigma^{i,j}\right)^\top} \text{ and } \quad \frac{\partial^2}{\partial W_\mu^{i,j} \partial \left(W_\sigma^{i,j}\right)^\top} .$$

Since we already calculated the first derivatives w.r.t. $W_\mu^{i,j}$ in Eq. 7.2 and w.r.t. $W_\sigma^{i,j}$ in Eq. 7.3, let us start by calculating the second derivative of the first term $W_\mu^{i,j}$

$$\begin{aligned}\frac{\partial^2}{\partial W_\mu^{i,j}\partial\left(W_\mu^{i,j}\right)^\top}\ln p\left(h^{i,j}|h^{i+1}\right) &= \frac{\partial}{\partial\left(W_\mu^{i,j}\right)^\top}\frac{\left(h^{i,j}-\left(W_\mu^{i,j}\right)^\top h^{i+1}\right)h^{i+1}}{\left(\sigma^{i,j}\right)^2}\\ &= \frac{h^{i+1}}{\left(\sigma^{i,j}\right)^2}\frac{\partial}{\partial\left(W_\mu^{i,j}\right)^\top}\left(h^{i,j}-\left(W_\mu^{i,j}\right)^\top h^{i+1}\right)\\ &= -\frac{h^{i+1}(h^{i+1})^\top}{\left(\sigma^{i,j}\right)^2}\ .\end{aligned}$$

To calculate the derivative of the loss w.r.t. $\frac{\partial^2}{\partial W_\sigma^{i,j}\partial\left(W_\sigma^{i,j}\right)^\top}$ we can start from the first derivative $\frac{\partial}{\partial W_\sigma^{i,j}}$ we calculated in Eq. (7.3):

$$\begin{aligned}&\frac{\partial^2}{\partial W_\sigma^{i,j}\partial\left(W_\sigma^{i,j}\right)^\top}\ln p\left(h^{i,j}|h^{i+1}\right) =\\ &= \frac{\partial}{\partial\left(W_\sigma^{i,j}\right)^\top}\frac{\left(h^{i,j}-\mu^{i,j}\right)^2}{\exp\left(\left(W_\sigma^{i,j}\right)^\top h^{i+1}\right)^2}h^{i+1}\\ &= -\frac{2\left(h^{i,j}-\mu^{i,j}\right)^2}{\left(\sigma^{i,j}\right)^2}h^{i+1}\left(h^{i+1}\right)^\top .\end{aligned}$$

Finally, we can compute the off-(block)diagonal elements of the FIM, by taking the first derivative with respect to $W_\mu^{i,j}$ from Eq. (7.2) and the second derivative, with respect to $W_\sigma^{i,j}$

$$\begin{aligned}\frac{\partial^2}{\partial W_\mu^{i,j}\partial\left(W_\sigma^{i,j}\right)^\top}\ln p\left(h^{i,j}|h^{i+1}\right) &= \frac{\partial}{\partial\left(W_\sigma^{i,j}\right)^\top}\frac{\left(h^{i,j}-\mu^{i,j}\right)}{\exp\left(\left(W_\sigma^{i,j}\right)^\top h^{i+1}\right)^2}h^{i+1}\\ &= \left(h^{i,j}-\mu^{i,j}\right)h^{i+1}\frac{\partial}{\partial\left(W_{\sigma,k}^{i,j}\right)^\top}\frac{1}{\exp\left(\left(W_\sigma^{i,j}\right)^\top h^{i+1}\right)^2}\\ &= -\frac{2\left(h^{i,j}-\mu^{i,j}\right)}{\left(\sigma^{i,j}\right)^2}h^{i+1}\left(h^{i+1}\right)^\top .\end{aligned}$$

To conclude, the Hessian of the loss of a layer i contains the following

derivatives

$$\frac{\partial^2}{\partial W_\mu^{i,j} \partial \left(W_\mu^{i,j}\right)^\top} \ln p\left(h^{i,j} | h^{i+1}\right) = -\frac{1}{\left(\sigma^{i,j}\right)^2} h^{i+1} \left(h^{i+1}\right)^\top , \tag{7.8}$$

$$\frac{\partial^2}{\partial W_\sigma^{i,j} \partial \left(W_\sigma^{i,j}\right)^\top} \ln p\left(h^{i,j} | h^{i+1}\right) = -\frac{2\left(h^{i,j} - \mu^{i,j}\right)^2}{\left(\sigma^{i,j}\right)^2} h^{i+1} \left(h^{i+1}\right)^\top \tag{7.9}$$

and

$$\frac{\partial^2}{\partial W_\mu^{i,j} \partial \left(W_\sigma^{i,j}\right)^\top} \ln p\left(h^{i,j} | h^{i+1}\right) = -\frac{2\left(h^{i,j} - \mu^{i,j}\right)}{\left(\sigma^{i,j}\right)^2} h^{i+1} \left(h^{i+1}\right)^\top . \tag{7.10}$$

7.1.2 The Fisher information matrix of a Normal distribution

In order to derive the FIM for a statistical model associated with the HM with conditional Gaussian random variables, we need to compute the expectation for each Hessian from Eqs. (7.8), (7.9) and (7.10).

Starting with the FIM for the parameters of the random variable $\mu^{i,j}$, we get

$$\begin{aligned} \mathcal{F}_{p,\mu}^{i,j} &= -\mathbb{E}_{p(x,h)} \left[\frac{\partial^2 \ln p\left(h^{i,j} | h^{i+1}\right)}{\partial W_\mu^{i,j} \partial \left(W_\mu^{i,j}\right)^\top} \right] \\ &= \mathbb{E}_{p(x,h)} \left[\frac{1}{\left(\sigma^{i,j}\right)^2} h^{i+1} \left(h^{i+1}\right)^\top \right] . \end{aligned} \tag{7.11}$$

Since the structure of the probability distribution p is a Markov chain

$$p(x,h) = p(x|h^1) p(h^1|h^2) ... p(h^M) ,$$

we know that for any function $f(h^i)$ the following equivalence holds true

$$\mathbb{E}_{p(x,h)}[f(h^i)] = \mathbb{E}_{p(h^M)} \left[\dots \mathbb{E}_{p(h^{i+1}|h^{i+2})} \left[\mathbb{E}_{p(h^i|h^{i+1})} \left[f(h^i) \right] \right] \right] . \tag{7.12}$$

The expectation terms from $p(h^{i-1}|h^i)$ to $p(x|h^1)$ can be ignored, since from their perspective $f(h^i)$ is a constant, which can be extracted, and then they all resolve to 1. Then the expectation of $f(h^i)$ can be estimated by Monte Carlo sampling through the previous layers:

$$\mathbb{E}_{p(x,h)}[f(h^i)] \sim \frac{1}{n} \sum f(h^i), \text{ with } \; h^{i+1} \sim p(h^{i+1} | \dots h^M) . \tag{7.13}$$

In the case of the HM we can sample from $q(h^{i+1}|h^i)$ instead, which is much simpler.

Since the expectation (7.11) is only dependent on the random variable h^{i+1} ($\sigma^{i,j}$ is also a function of h^{i+1}) we can compute the empirical estimation of the block of the FIM by Monte-Carlo sampling h^{i+1} n times from the approximate distribution[4] as

$$F_{p,\mu}^{i,j} = \frac{1}{n} \sum \frac{1}{\left(\sigma^{i,j}\right)^2} h^{i+1} \left(h^{i+1}\right)^\top \text{ , with } h^{i+1} \sim q(h^{i+1}|h^i) \,. \tag{7.14}$$

In our case we use $n = B \cdot S$, where B is the size of the minibatch and S is the sample size, similarly to how we did it in Eqs. (5.5) and (5.6) from Section 5.2. In the case of the FIM for the parameters of the random variable $\sigma^{i,j}$, we have

$$\begin{aligned} \mathcal{F}_{p,\sigma}^{i,j} &= -\mathbb{E}_{p(x,h)} \left[\frac{\partial^2 \ln p\left(h^{i,j}|h^{i+1}\right)}{\partial W_\sigma^{i,j} \partial {W_\sigma^{i,j}}^\top} \right] \\ &= \mathbb{E}_{p(x,h)} \left[\frac{2\left(h^{i,j} - \mu^{i,j}\right)^2}{\left(\sigma^{i,j}\right)^2} h^{i+1} \left(h^{i+1}\right)^\top \right] \,. \end{aligned}$$

Notice that, compared to Eq. (7.11), there are two random variables (h^i and h^{i+1}) in the above expectation, which we can solve by splitting it over the two separate variables. The innermost expectation concerning h^i considers h^{i+1} and thus the parameters $\mu^{i,j}$ and $\sigma^{i,j}$ as constants, hence we can pull them out of the expectation

$$\begin{aligned} &\mathbb{E}_{p(h^i|h^{i+1})} \left[\frac{2\left(h^{i,j} - \mu^{i,j}\right)^2}{\left(\sigma^{i,j}\right)^2} h^{i+1} \left(h^{i+1}\right)^\top \right] = \\ &\qquad = \frac{2h^{i+1}\left(h^{i+1}\right)^\top}{\left(\sigma^{i,j}\right)^2} \mathbb{E}_{p(h^i|h^{i+1})} \left[\left(h^{i,j} - \mu^{i,j}\right)^2 \right] \,. \end{aligned} \tag{7.15}$$

Knowing that for a random variable x sampled from a Normal distribution $\mathcal{N}(\mu, \sigma)$, the second central moment is $\mathbb{E}\left[(x-\mu)^2\right] = \sigma^2$, we can simplify the above expectation as

$$\begin{aligned} \frac{2h^{i+1}\left(h^{i+1}\right)^\top}{\left(\sigma^{i,j}\right)^2} \mathbb{E}_{p(h^i|h^{i+1})} \left[\left(h^{i,j} - \mu^{i,j}\right)^2 \right] &= \frac{2\left(\sigma^{i,j}\right)^2}{\left(\sigma^{i,j}\right)^2} h^{i+1} \left(h^{i+1}\right)^\top \\ &= 2h^{i+1} \left(h^{i+1}\right)^\top \,. \end{aligned}$$

[4]This estimation is explained in detail in Section 5.2.

Notice that the geometry of σ is constant over the individual nodes j and only depends on the layer $i+1$. Furthermore, if $p\left(h^{i+1}|h^{i+2}\right)$ is also a Gaussian distribution, and knowing that for a normally distributed random variable x the non-central moment is $\mathbb{E}\left[x^2\right] = \mu^2 + \sigma^2$, we can simplify the expectation from Eq. 7.12) as

$$\mathbb{E}_{p(h^{i+1}|h^{i+2})}\left[2h^{i+1}\left(h^{i+1}\right)^\top\right] = 2\left(\mu^{i+1}\left(\mu^{i+1}\right)^\top + \sigma^{i+1}\left(\sigma^{i+1}\right)^\top\right) .$$

This result is satisfying since the solution to the expectation can be estimated more simply and can be shared for all blocks of the layer i. The empirical estimation of $\mathcal{F}_{p,\sigma}^{i,j}$ can be obtained by

$$F_{p,\sigma}^{i,j} = \frac{2}{n}\sum h^{i+1}\left(h^{i+1}\right)^\top \text{, with } \; h^{i+1} \sim q(h^{i+1}|h^i) \; , \tag{7.16}$$

which holds for any distribution of the random variable h^{i+1}. If $p(h^{i+1}|h^{i+2})$ is also a Gaussian distribution, we can estimate the block of the FIM as

$$\begin{aligned} F_{p,\sigma}^{i,j} &= \frac{2}{n}\sum\left(\mu^{i+1}\left(\mu^{i+1}\right)^\top + \sigma^{i+1}\left(\sigma^{i+1}\right)^\top\right) \\ &= \frac{2}{n}\sum W_\mu^{i+1}h^{i+2}\left(W_\mu^{i+1}h^{i+2}\right)^\top + \exp\left(W_\sigma^{i+1}h^{i+2}\right)\left(\exp\left(W_\sigma^{i+1}h^{i+2}\right)\right)^\top , \\ &\quad \text{with } \; h^{i+2} \sim q(h^{i+2}|h^{i+1}) \; . \end{aligned} \tag{7.17}$$

For the off-diagonal elements of the FIM for node j in layer i involving both μ and σ, we have

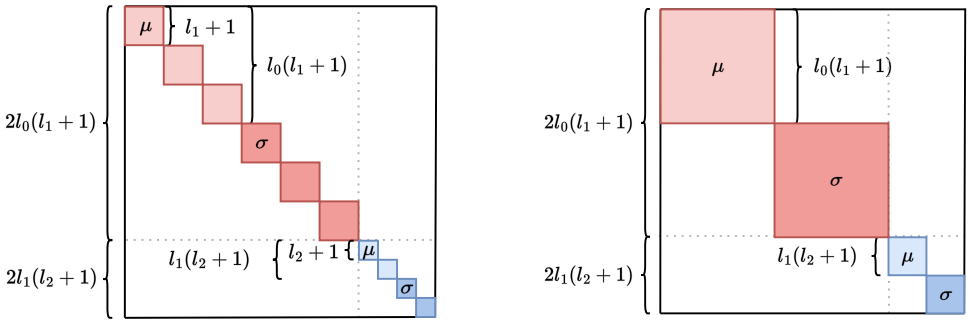
$$\begin{aligned} &\mathbb{E}_{p(x,h)}\left[\frac{\partial^2}{\partial W_\mu^{i,j}\partial W_\sigma^{i,j}}\ln p\left(h^{i,j}|h^{i+1}\right)\right] = \\ &\qquad = \mathbb{E}_{p(x,h)}\left[\frac{2\left(h^{i,j}-\mu^{i,j}\right)}{\left(\sigma^{i,j}\right)^2}h^{i+1}\left(h^{i+1}\right)^\top\right] \; . \end{aligned}$$

Refactoring the expectations with respect to $h^{i,j}$ as we did in Eqs. (7.12) and (7.15), and knowing that the first central moment of a random variable x sampled from a Normal distribution $\mathcal{N}(\mu,\sigma)$ is $\mathbb{E}\left[x-\mu\right] = 0$, we have

$$\begin{aligned} &\mathbb{E}_{p(h^i|h^{i+1})}\left[-\frac{2\left(h^{i,j}-\mu^{i,j}\right)}{\left(\sigma^{i,j}\right)^2}h^{i+1}\left(h^{i+1}\right)^\top\right] \\ &\qquad = -\frac{2h^{i+1}\left(h^{i+1}\right)^\top}{\left(\sigma^{i,j}\right)^2}\mathbb{E}_{p(h^i|h^{i+1})}\left[h^{i,j}-\mu^{i,j}\right] \\ &\qquad = 0 \; . \end{aligned} \tag{7.18}$$

□

Since the expectation with respect to h^i given h^{i+1} is 0, it follows that we have a similarly fine-grained structure for the FIM to the one of the Bernoulli case (see Section 4.3), which has smaller blocks than what is commonly used in the literature. We visualize the structure of a FIM in Figure 7.2 with the blocks reordered compared to Figure 7.1 to illustrate the difference to the usual assumption from the literature (Desjardins et al., 2015; Grosse and Salakhudinov, 2015; Sun and Nielsen, 2017).



(a) FIM structure of our model.

(b) FIM structure of other's model.

Figure 7.2: Graphical representation of the FIM for a network using Gaussian variables with 3-2 nodes and the prior. The gray lines correspond to the blocks associated with the network layers. The FIM admits a fine-grained block-diagonal structure with blocks of size equal to the size of the hidden layers. The blocks are ordered from the bottom layer to the top starting from the upper left. The literature comparisons are from the works of Desjardins et al. (2015),Grosse and Salakhudinov (2015) and Sun and Nielsen (2017).

7.1.3 Natural Reweighted Wake Sleep with Normal Distributions

In order to use the NRWS with Gaussian distributions, we need to invert the FIMs from the previous section.

Remark 8
Notice that the FIM in Eqs. (7.14) *and* (7.16) *can be written as a product of matrices in the form* $UQU^\top$ *as we have seen in Section 5.2, with different values for* Q *compared to the binary case. It follows that Eq.* (5.12)*, which we introduced in Section 5.3.5, is still valid to invert and regularize the FIM in the NRWS.*

Thus, the inverse of a series of blocks corresponding to layer i of the FIM is given by

$$\begin{aligned}\left(\tilde{F}_p^i\right)^{-1} &= \left(\frac{\alpha I_{l_i} + U^{i+1} Q^i \left(U^{i+1}\right)^\top}{1+\alpha}\right)^{-1} \\ &= \frac{1+\alpha}{\alpha}\left[I_{l_i} - U^{i+1}\left(\alpha\left(Q^i\right)^{-1} + \left(U^{i+1}\right)^\top U^{i+1}\right)^{-1}\left(U^{i+1}\right)^\top\right],\end{aligned} \tag{7.19}$$

where α is the damping factor, U^{i+1} has as columns the samples h^{i+1} and Q^i is a diagonal matrix with entries $1/\sigma^{i,j}$ in the case of the $F_{p,\mu}^{i,j}$, and the identity I_{l_i} for $F_{p,\sigma}^{i,j}$.

When all layers of the HM are normally distributed we can directly estimate the FIM from Eq. (7.17). In this case, we do not need an elaborate inversion as in the Eq. (7.19), but we just calculate the inverse after the summation of the terms, as it has the same computational cost as computing the inner parenthesis in the former equation $(\alpha\left(Q^i\right)^{-1} + \left(U^{i+1}\right)^\top U^{i+1})$.

As a consequence of the remark above, the complexity analysis we have done in Chapter 5 also applies here, i.e., the complexity of the NRWS is dominated by the inversion the FIMs of $F_{p,\mu}^{i,j}$ and $F_{p,\sigma}^{i,j}$ (only in the general case as in Eq. 7.16) using the Woodbury formula. Since usually, the bottom two layers of the HM are the largest, in the general case for σ we have a computational complexity of $\mathcal{O}(l_0(l_1 n + n^\omega))$ every K-th step and $\mathcal{O}(l_0(l_1 n + n^2))$ on all the other steps for both matrices. On the other side, when all layers have a Gaussian distribution, only the inversion of F^μ has the complexity presented above, which still dominates the computation.

To illustrate the capabilities of our algorithm we used it on the TFD (Susskind et al., 2010), with the same hyperparameters as we used in our previous work (damping factor α, K-step, batch size B and sample size S), except the network architecture, which has the same sizes with Normal distributions on every layer and the learning rate η. The learning rate has to be lowered compared to the binary case because the continuous random variables are faster to learn, as a consequence, we chose the value of 10^{-4}.

In Figure 7.3, we show generated images using binary and continuous distributions. First, notice the difference between the number of iterations till convergence of the methods, as in the continuous case just after 200 epochs the faces look comparable to the ones of the binary case after $5,000$, thanks to the faster convergence of continuous variables compared to binary ones. In the binary case, we see faces with much stronger facial features, i.e., shadows, orientation, mustaches or emotions, compared to the continuous case, but also more salt-and-pepper noise and blurriness, sometimes to the point that the

(a) Images generated as means of binary variables after $5,000$ epochs.

(b) Images generated as means of continuous variables after 200 epochs.

Figure 7.3: Images generated from an HM trained with NRWS with (top) Bernoulli and (bottom) Normal distributions.

face is unrecognizable. While faces in the continuous case are more smooth, but with less recognizable features (sometimes we cannot differentiate between lips and teeth, and almost all faces are oriented directly towards the camera),

however, there are no unrecognizable faces. Some of these problems may not appear at different scales, as for larger images the small, pixel-scale mistakes would be less noticeable.

7.2 Convolutional Neural Network

So far we have analyzed dense network architectures, where every neuron is connected to every other neuron from the adjacent layers. However, we would benefit from the flexibility of using other types of layer structures to introduce sparsity in the weights, to be able to scale up HMs to support e.g. larger images. Less dense structures, such as convolutional layers (LeCun et al., 1998), through weight sharing and by exploiting spacial information, have been shown to greatly reduce the number of parameters used for learning datasets with larger data.

We aim at expanding the capabilities of the NRWS by being able to apply it to convolutional networks. To achieve this, we first need to investigate the geometry of convolutional layers through its FIM to enable us to use it in the context of the NG update.

7.2.1 Convolutional Layer

Convolutional layers and Convolutional Neural Networks (CNNs) gained considerable attention after the pioneering work of LeCun et al. (1998) where they first used it for classification tasks on images. Typically such a convolutional layer applied to images can be represented as a function, where the inputs x and the outputs y are 3-dimensional tensors, where the weights V is a 4-dimensional tensor[5] and the bias d is a vector. Let us define these quantities with sizes as:

- $x : \mathtt{H} \times \mathtt{W} \times \mathtt{C}$, where $\mathtt{H}$ is the height, $\mathtt{W}$ is the width and $\mathtt{C}$ is the number of channels of the input x;
- $y : \mathtt{H}' \times \mathtt{W}' \times \mathtt{F}$, where $\mathtt{H}'$ is the height, $\mathtt{W}'$ is the width and F is the number of channels of the output y;
- $V : \mathtt{K} \times \mathtt{L} \times \mathtt{C} \times \mathtt{F}$, where $\mathtt{K}$ is the height and $\mathtt{L}$ is the width of the weights of the filter V;
- $d : \mathtt{F}$, which is the bias;

[5] Also usually referred to as filters in the context of CNNs

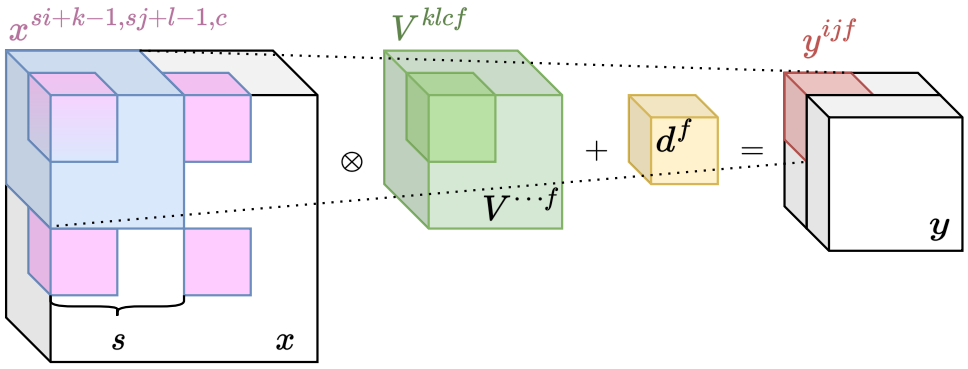


Figure 7.4: Convolutional mapping of the input layer x to an example output element from y. In this example $\mathtt{H} = 4$, $\mathtt{W} = 4$, $\mathtt{C} = 2$, $\mathtt{H}' = 2$, $\mathtt{W}' = 2$, $\mathtt{F} = 2$, $\mathtt{K} = 2$, $\mathtt{L} = 2$, with $s = 2$. The blue area shows which area of x is multiplied with the weights $V^{\cdots\mathtt{f}}$ to define $y^{ij\mathtt{f}}$ in the case where $i = 1$, $j = 1$ and $\mathtt{f} = 1$. The bias $d^{\mathtt{f}}$, represented as a yellow cube, is added output to the previous multiplication which results in $y^{ij\mathtt{f}}$, red cube. The dark green element $V_{\mathtt{klcf}}$ is multiplied with the pink elements $x^{\mathtt{s}i+\mathtt{k}-1,\mathtt{s}j+\mathtt{l}-1,\mathtt{c}}$, thanks to the shift given by the stride $\mathtt{s}$.

- $\mathtt{s}$: scalar, is the stride.

The weights V of the layer usually do not span the whole height and width of the input layer x ($\mathtt{K} < \mathtt{H}$ and $\mathtt{L} < \mathtt{W}$). The convolutional operation works by applying the same filter V as a sliding window over the input x. The weights V are shifted by $\mathtt{s}$ number of elements vertically and horizontally until they cover the whole input. We can define the function implemented by the convolutional layer for an element of the output $y^{ij\mathtt{f}}$ as

$$y^{ij\mathtt{f}} = \sum_{\mathtt{k}=1}^{\mathtt{K}} \sum_{\mathtt{l}=1}^{\mathtt{L}} \sum_{\mathtt{c}=1}^{\mathtt{C}} V^{\mathtt{klcf}} \, x^{\mathtt{s}i+\mathtt{k}-1,\mathtt{s}j+\mathtt{l}-1,\mathtt{c}} + d^{\mathtt{f}} \; . \tag{7.20}$$

The result is that the outputs $y^{ij\mathtt{f}}$ capture only local information about the input x, depending on the indices i, j, and the width and height of the filter $\mathtt{K} \times \mathtt{L}$. In Figure 7.4 we show an example image[6], of how such a convolutional function applies its weights V to an input x. Additionally, activation functions and probability distributions can be applied to the end of such layers, similarly to the previously used dense layers, to transform them into stochastic layers.

[6]In Figure 7.4 the notation $\cdot$ means all the indices along the axis, therefore $V^{\cdots\mathtt{f}}$ represents the $\mathtt{f}$-th $3D$ sub-tensor of the 4-dimensional tensor V.

Note 10
For an intuitive explanation of how we get to this many indices, and how the convolutional formulae work, we refer to Appendix E, where we build up from the simplest two-dimensional case to the final, complex version we see in Eq. (7.20)*, with multiple input and output channels, padding, and stride.*

Note, that the fact that in a convolutional operation, we shift the filter V over the input x, can be understood as sharing the said weights per layer. This is an important property of convolutional layers since the sharing of weights reduces the number of weights per layer.

7.2.2 Deconvolutional Layer

The deconvolutional operation can also be used to define a deconvolutional layer where the sizes of the parameters are equivalent to the convolutional ones, only now we treat y as the input and x as the output

- $x : \mathtt{H} \times \mathtt{W} \times \mathtt{C}$, where $\mathtt{H}$ is the height, $\mathtt{W}$ is the width, and $\mathtt{C}$ is the number of channels of the output x;
- $y : \mathtt{H}' \times \mathtt{W}' \times \mathtt{F}$, where $\mathtt{H}'$ is the height, $\mathtt{W}'$ is the width, and $\mathtt{F}$ is the number of channels of the input y;
- $W : \mathtt{K} \times \mathtt{L} \times \mathtt{F} \times \mathtt{C}$, where $\mathtt{K}$ is the height, and $\mathtt{L}$ is the width of the weights of the filter W;
- b: $\mathtt{C}$, which is the bias;
- $\mathtt{s}$: scalar, is the stride.

Note 11
For simplicity of notation, we assumed the same sizes $\mathtt{H}, \mathtt{W}, \mathtt{C}, \mathtt{F}, \mathtt{K}, \mathtt{L}$ *for heights, widths, and depths of weight tensor, input and output tensors for the convolutional and the deconvolutional case, which would mean that they are mirror images of each other. In practice, that is not always the case, as by modifying the sizes of the filter* W *and stride* $\mathtt{s}$ *we could end up with the same size for the input and output, therefore sizes of the filters of the convolutional and deconvolution layers may not be equal. We made this assumption only to simplify the text and to not introduce any further constants.*

The deconvolution operation for a single element $x^{ij\mathtt{c}}$ can be defined as

$$\begin{aligned} x^{ij\mathtt{c}} &= \sum_{\mathtt{k}=1}^{\mathtt{K}} \sum_{\mathtt{l}=1}^{\mathtt{L}} \sum_{\mathtt{f}=1}^{\mathtt{F}} W^{\mathtt{klfc}}\, y^{(i-\mathtt{k})/\mathtt{s}+1,(j-\mathtt{l})/\mathtt{s}+1,\mathtt{f}} + b^{\mathtt{c}} \text{ , where} \\ y^{(i-\mathtt{k})/\mathtt{s}+1,(j-\mathtt{l})/\mathtt{s}+1,\mathtt{f}} &= 0 \quad \text{if} \quad \frac{(i-\mathtt{k})}{\mathtt{s}} + 1, \frac{(j-\mathtt{l})}{\mathtt{s}} + 1 \notin \mathbb{N} \,. \end{aligned} \tag{7.21}$$

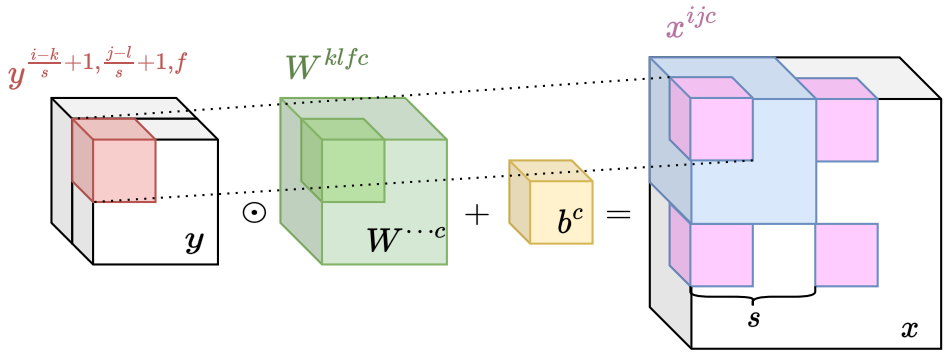


Figure 7.5: Deconvolutional mapping of the input layer y to an example output element from x. In this example $\mathtt{H} = 4$, $\mathtt{W} = 4$, $\mathtt{C} = 2$, $\mathtt{H}' = 2$, $\mathtt{W}' = 2$, $\mathtt{F} = 2$, $\mathtt{K} = 2$, $\mathtt{L} = 2$, with $s = 2$. The blue area shows which area of x is the result of the deconvolution of $y^{ij\mathtt{f}}$, red cube, with the weights $W^{\cdots\mathtt{c}}$, green cube, summed with the bias $b^{\mathtt{c}}$, yellow cube, in the case where $i = 1$ $j = 1$, and $\mathtt{c} = 1$, summed. The pink blocks of x are the result of the deconvolution of y by the elements $W_{\mathtt{klfc}}$, dark green, thanks to the shift given by the stride $\mathtt{s}$.

Similarly to the convolutional layer, we can view the deconvolutional operation as a sliding window of shared weights over the input. In Fig 7.5 we present an illustration of a deconvolution operation.

Remark 9
Note that a deconvolutional layer is not an inverse operator of a convolutional one, as there is no direct, one-to-one correspondence between the weights W and V.

7.3 The Geometry of a Convolutional Neural Network

Recall our derivation of the gradients of a DAG from Theorem 1, where we have shown that the FIM of a DAG is block-diagonal. It is easy to see that CNNs are still DAGs, which means that the theorem holds for these types of layers as well. However, CNNs introduce an extra property by weight sharing, which means we have to treat them differently.

Theorem 3
Let $\mathcal{G}$ be a directed acyclic graphical model, whose variables are grouped in

layers such that each node from the i-th layer has parent nodes from the $(i-1)$-th layer. The variables are grouped per layer in disjoint sets (partition) $\bigcup S_i = N$, where $S_i \bigcap S_j = \emptyset$, $\forall S_i, S_j \subset N$, where the same variables are applied to multiple pairs of parent-child nodes within one set. The FIM associated with the joint probability distribution p that factorizes as the product of conditional distributions according to $\mathcal{G}$ has a block-diagonal structure, with one block for each subset S_i per layer, of size equal to the number of parent nodes.

In the following section, we prove Theorem 3, and we see how it applies to CNNs in HMs.

7.3.1 The Directed Acyclic Graph View of a Convolutional Network

Let N be a directed graphical model, with the nodes divided into two sets, $N = \{x, h\}$, with x the set of observed random variables (points in the data set) and h the set of hidden random variables, generating the observations. We suppose in our calculations that x and h are discrete. The joint distribution of the nodes in N can be written in the following way, as it is classically done in directed graphical models:

$$p_\theta(x, h) = \prod_{r \in N} K_r(x_r | x_{pa(r)}; \theta_r) \tag{7.22}$$

where for each node $r \in N$, $x_{pa(r)}$ are the random variables corresponding to the set of parent nodes of the random variable x_r, and K_r is the pdf (or pmf) of x_r given its parents, having parameters θ_r.

However, since our weights θ_i are shared within subsets $S_i \subset N$, where $S_i \bigcup S_j = \emptyset$, $\forall S_i, S_j \subset N$ and $i \neq j$, with a total of R subsets, we can rewrite Eq. (7.22) as

$$p_\theta(x, h) = \prod_{r}^{R} \prod_{s \in S_r} K_r(x_s | x_{pa(s)}; \theta_r) \ . \tag{7.23}$$

We illustrate an example DAG with shared weights in Figure 7.6. In this example, we have two kernels $K_1(\cdot; \theta_1)$ parameterized by $\theta_1 = \{\alpha\}$, and $K_2(\cdot; \theta_2)$ with $\theta_2 = \{\beta, \gamma\}$, where K_1 is applied to the set of nodes $S_1 = \{x_5, x_6, x_7, x_8\}$, as $K_1(x_5|x_1; \theta_1)$, $K_1(x_6|x_2; \theta_1)$, etc., and K_2 is applied to the set of nodes $S_2 = \{x_9, x_{10}, x_{11}\}$, as $K_2(x_9|x_5, x_6; \theta_2)$, $K_2(x_{10}|x_6, x_7; \theta_2)$, etc. Note, that in the general case, the nodes of S_i do not need to be distributed per layer, as shown in the current example, but can be arbitrarily arranged throughout the DAG. We chose a layered distribution of the nodes

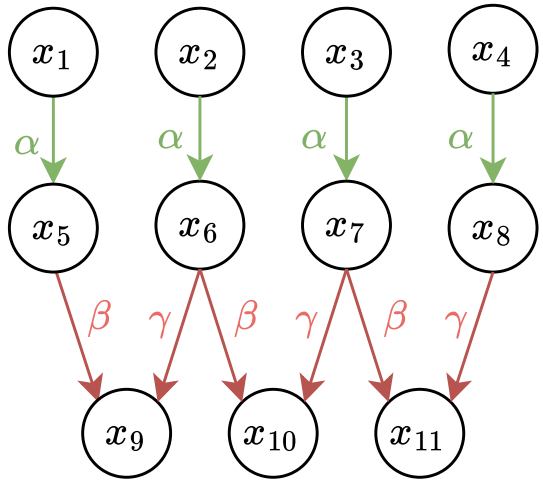


Figure 7.6: Example DAG with shared weights $\theta_1 = \{\alpha\}$ (green arrows) for the set $S_1 = \{x_5, x_6, x_7, x_8\}$, and $\theta_2 = \{\beta, \gamma\}$ (red arrows) for the set $S_2 = \{x_9, x_{10}, x_{11}\}$.

to more closely resemble an SBN's structure, which forms the networks of an HM.

If $p_{\mathcal{D}}(x)$ is the true distribution of the dataset, our goal is to minimize $\mathrm{D}_{KL}[p_{\mathcal{D}}(x)||p_\theta(x)]$, w.r.t. the parameters θ, which is in the discrete case is equivalent to maximizing $\sum_x p_{\mathcal{D}}(x) \ln p_\theta(x)$ as shown in Chapter 2. The loss function we are optimizing is $\mathcal{L}(\theta) = \sum_x p(x) \ln p_\theta(x)$, where θ is the set of parameters θ_r which define the kernels K_r. The derivatives of the loss can be

computed as

$$\begin{aligned}
\frac{\partial \mathcal{L}(\theta)}{\partial \theta_r} &= \frac{\partial}{\partial \theta_r} \sum_x p_{\mathcal{D}}(x) \ln p_\theta(x) \\
&= \sum_x p_{\mathcal{D}}(x) \frac{\partial}{\partial \theta_r} \ln \sum_h p_\theta(x,h) \\
&= \sum_x p_{\mathcal{D}}(x) \frac{\partial}{\partial \theta_r} \ln \sum_h \prod_i^R \prod_{s \in S_i} K_i\left(x_s | x_{\mathrm{pa(s)}}; \theta_i\right) \quad \text{(from Eq. (7.23))} \\
&= \sum_x p_{\mathcal{D}}(x) \frac{1}{p_\theta(x)} \sum_h \frac{\partial}{\partial \theta_r} \left(\prod_i^R \prod_{s \in S_i} K_i\left(x_s | x_{\mathrm{pa(s)}}; \theta_i\right) \right) \\
&= \sum_x \frac{p_{\mathcal{D}}(x)}{p_\theta(x)} \sum_h \left(\prod_j^R \prod_{t \in S_j} K_j\left(x_t | x_{\mathrm{pa(t)}}\right) \right) \frac{\partial}{\partial \theta_r} \ln \prod_i^R \prod_{s \in S_i} K_i\left(x_s | x_{\mathrm{pa(s)}}; \theta_i\right) \\
&= \sum_{x,h} \frac{p_{\mathcal{D}}(x)}{p_\theta(x)} p_\theta(x,h) \frac{\partial}{\partial \theta_r} \sum_{S_i \subset N} \sum_{s \in S_i} \ln K_i\left(x_s | x_{\mathrm{pa(s)}}; \theta_i\right) \\
&= \sum_{x,h} p_{\mathcal{D}}(x) p_\theta(h|x) \sum_{s \in S_r} \frac{\partial}{\partial \theta_r} \ln K_r\left(x_s | x_{\mathrm{pa(s)}}; \theta_r\right) \ ,
\end{aligned}$$

using $p^\star_\theta(x,h) = p_{\mathcal{D}}(x) p_\theta(h|x)$

$$\begin{aligned}
&= \sum_{x,h} p^\star_\theta(x,h) \sum_{s \in S_r} \frac{\partial}{\partial \theta_r} \ln K_r\left(x_s | x_{\mathrm{pa(s)}}; \theta_r\right) \\
&= \mathbb{E}_{p^\star_\theta(x,h)} \left[\sum_{s \in S_r} \frac{\partial}{\partial \theta_r} \ln K_r\left(x_s | x_{\mathrm{pa(s)}}; \theta_r\right) \right]
\end{aligned}$$

Since the statement inside the expectation only depends on x_s and its parents $x_{\mathrm{pa(s)}}$, where $s \in S_r$, we can simplify the above expectation to

$$\frac{\partial \mathcal{L}(\theta)}{\partial \theta_r} = \mathbb{E}_{x_s, x_{\mathrm{pa(s)}}; s \in S_r} \left[\sum_{s \in S_r} \frac{\partial}{\partial \theta_r} \ln K_r\left(x_s | x_{\mathrm{pa(s)}}; \theta_r\right) \right] \tag{7.24}$$

Therefore, computing the gradient w.r.t. the variable θ_r is independent of the computation of the gradient w.r.t. the other variables. Notice that the gradient w.r.t. a variable θ_r is dependent on the sum of all gradients of the kernels in which it is involved.

In order to compute the FIM for a DAG with a convolutional structure, we have to calculate the Hessian for the loss function above. From the Eq. (7.24) it follows that

$$\nabla_{\theta_r} \nabla_{\theta_t} \ln p_\theta(x,h) = \delta_{rt} \sum_{s \in S_r} \nabla^2_{\theta_r} \ln K_r\left(x_s | x_{\mathrm{pa(s)}}; \theta_r\right) \ ,$$

with $\delta_{rt} = 1$, if $r = t$ and 0, otherwise, since the partitions S_r and S_t are by definition disjoint if $r \neq t$. This shows that the FIM has a block-diagonal structure, where the blocks are sums over the individual applications of the shared weights

$$\mathcal{F}(\theta) = -\mathbb{E}_{p_\theta(x,h)} \left[\text{block-diag} \left(\sum_{s \in S_r} \nabla^2_{\theta_r} \ln K_r \left(x_s | x_{\text{pa(s)}}; \theta_r \right) \right)_{r \in N} \right] . \quad (7.25)$$

□

7.3.2 Helmholtz Machines based on Convolutional Networks

In this section, we present how to apply Theorem 3 to a HM using convolutional or deconvolutional layers. Since the HM is by construction a DAG, based on Theorem 3 we have that both convolutional and deconvolutional layers will have a FIM that is block-diagonal, with a number of blocks equal to the number of output channels, and with size equal to the product of the layers' weights' first three dimensions. To understand these blocks, we have to derive the formula for the FIM incrementally by calculating the first-order derivative of the loss and then the FIM.

We start from the representation of the loss as the log-likelihood of the model[7]:

$$\mathcal{L}(\theta) = \mathbb{E}[\ln p_\theta (x, h; \theta)]$$

We can split the nodes of the model θ into sets $S^{l,r}$ by M layers and by R^l filters that share the same parameters, where l is the index of the layer, and r is the index of the shared set within layer l. We can describe the parameters that are shared by the nodes of the set $S^{l,r}$ as $\{\theta^{l,r}\} \in \theta$. Calculating the gradient w.r.t. a specific $\theta^{l,r}$ for a network containing M convolutional or deconvolutional layers we get

$$\begin{aligned} \frac{\partial \mathcal{L}(\theta)}{\partial \theta^{l,r}} &= \frac{\partial}{\partial \theta^{l,r}} \ln p_\theta (x, h; \theta) \\ &= \frac{\partial}{\partial \theta^{l,r}} \sum_{i=0}^{M-1} \sum_{t}^{R^l} \sum_{s \in S^{l,t}} \ln p \left(h^{i,s} | h^{i+1}, \theta^{i,t} \right) \\ &= \sum_{s \in S^{l,r}} \frac{\partial}{\partial \theta^{l,r}} \ln p \left(h^{l,s} | h^{l+1}; \theta^{l,r} \right) , \end{aligned}$$

[7]For a more detailed explanation of the basics of HMs, see Chapter 2

We know that the weights of the convolution and deconvolution are shared along the horizontal and vertical axes of the output layer, the set $S^{l,r}$ corresponds to the set of indices on those axes. Since both convolution and deconvolution layers are independent over their respective output channels[8], `C` or `F`, these correspond to the number of unique blocks R. Therefore, the gradient w.r.t. the loss, for both types of layers is

$$\frac{\partial \mathcal{L}(\theta)}{\partial \theta^{l,r}} = \sum_{i=1}^{\mathtt{H}'_l} \sum_{j=1}^{\mathtt{W}'_l} \frac{\partial}{\partial \theta^{l,r}} \ln p\left(h^{l,ijr} | h^{l+1}; \theta^{l,r}\right) \ .$$

Consequently, according to Eq. (7.25) we can define the FIM of a belief network containing convolutional or deconvolutional layers as

$$\mathcal{F}(\theta) = -\mathbb{E}_{p_\theta(x,h)} \left[\underset{l \in [0, M-1]; r \in [1, R^l]}{\text{block-diag}} \left(\sum_{i=1}^{\mathtt{H}'_l} \sum_{j=1}^{\mathtt{W}'_l} \nabla^2_{\theta^{l,r}} \ln p\left(h^{l,ijr} | h^{l+1}; \theta^{l,r}\right) \right) \right] , \tag{7.26}$$

with a single block for layer h^l with output channel r will have the form

$$\mathcal{F}^{l,r}_p = -\mathbb{E}_{p(x,h)} \left[\sum_{i=1}^{\mathtt{H}'_l} \sum_{j=1}^{\mathtt{W}'_l} \nabla^2_{\theta^{l,r}} \ln p\left(h^{l,ijr} | h^{l+1}; \theta^{l,r}\right) \right] . \tag{7.27}$$

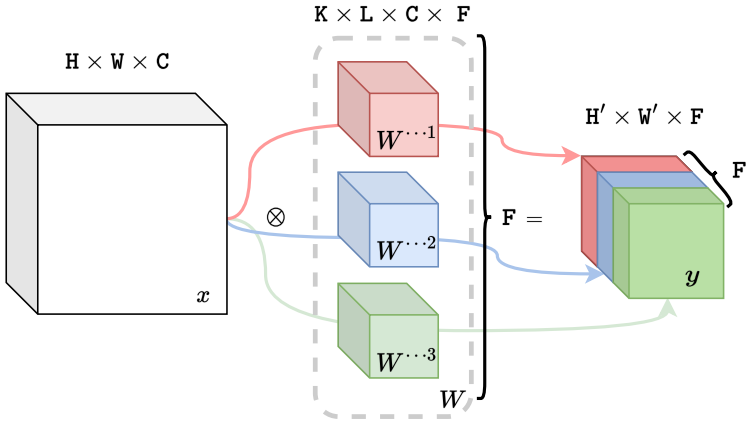
Let us focus on a single layer l of the belief network to understand better the derivations of the FIM, as illustrated in Figure 7.7. If we view HM as an SBN, we have the layered formulation of the gradients, which holds true when we use convolutional layers as well. Consider a single layer with the distribution $p\left(y|x;\theta\right)$, parametrized by θ, with input x and output y of the layer. In the following, the distribution p can have an underlying convolutional or deconvolutional layer, the distinction is not important at this point.

Knowing that the loss of the HM is the log-likelihood of the network, we can calculate the gradients of the layer with respect to the loss as

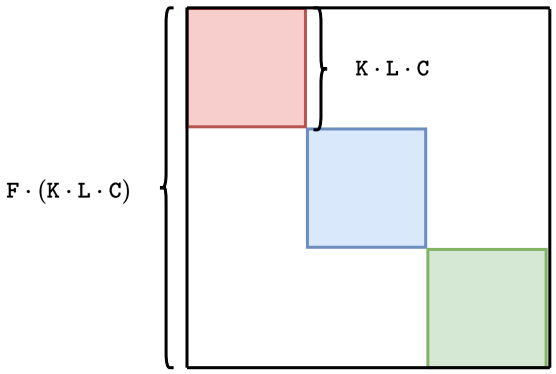
$$\frac{\partial}{\partial \theta} \ln p\left(y|x;\theta\right) = \sum_{i=1}^{\mathtt{H}'} \sum_{j=1}^{\mathtt{W}'} \sum_{r=1}^{R} \frac{\partial}{\partial \theta^r} \ln p\left(y^{ijr} | x; \theta^r\right) \ , \tag{7.28}$$

where the sums over $\mathtt{H}'$ and $\mathtt{W}'$ represent the sliding window over the height and width of the output, and R represents the output channel, which corresponds to the `C` in case of the convolutional layer and `F` for a deconvolutional layer.

[8] All the constants and indices that are written `Monospace` fonts are used as they have been defined in Sections 7.2.1 and 7.2.2.



(a) Convolutional layer with $F = 3$ output layers.



(b) FIM of a convolutional layer with $F = 3$ output layers.

Figure 7.7: (a) Graphic representation of a convolutional layer with input x of size $\mathtt{H} \times \mathtt{W} \times \mathtt{C}$ and output y of size $\mathtt{H}' \times \mathtt{W}' \times \mathtt{F}$ with weights W of size $\mathtt{K} \times \mathtt{L} \times \mathtt{C} \times \mathtt{F}$, $W^{\cdots \mathtt{f}}$ represents the $\mathtt{f}$-th tensor of size $\mathtt{K} \times \mathtt{L} \times \mathtt{C}$; each color represents a different output channel in both the weights W and output y; (b) the structure of the FIM associated to the weights W of such a convolutional layer, with the different colors aligned with the blocks of the FIM.

We consider θ^r to be a linear function of x with the weights W. Taking the derivative w.r.t. a specific weight $W^{\mathtt{kl}tr}$ using the chain rule

$$\frac{\partial}{\partial W^{\mathtt{kl}tr}} \ln p\left(y^{ijr} | x; \theta^r\right) = \underbrace{\frac{\partial}{\partial \theta^r} \ln p\left(y^{ijr} | x; \theta^r\right)}_{①} \underbrace{\frac{\partial}{\partial W^{\mathtt{kl}tr}} \theta^r\left(W, i, j\right)}_{②} . \tag{7.29}$$

In this formulation, we assume that the activation function is part of the first term ①, e.g., the sigmoid function for a Bernoulli distribution or the

exponential function for the σ of the Normal distribution, and we consider the second term ② to be a derivative of a linear function of W.

We know that ② will result in a variable of the input layer x (since $\theta^r(W,i,j)$ is a linear function), depending on the indices of the output ijr and the indices of the weights $\mathtt{klt}r$. We can define a function Ω that recovers the input variable from x with the right indices, based on the ijr and $\mathtt{klt}r$ to replace our derivative as

$$\Omega^{ijr}_{\mathtt{klt}r} = \frac{\partial}{\partial W^{\mathtt{klt}r}} \theta^r(W,i,j) \ . \tag{7.30}$$

This function is defined for convolutional layers according to Eq. (7.20) as

$$\begin{aligned} \Omega^{ijr}_{\mathtt{klt}r} &= \frac{\partial}{\partial W^{\mathtt{klt}r}} \sum_{\mathtt{k}=1}^{\mathtt{K}} \sum_{\mathtt{l}=1}^{\mathtt{L}} \sum_{\mathtt{c}=1}^{\mathtt{C}} W^{\mathtt{klcf}} \, x^{\mathtt{s}i+\mathtt{k}-1,\mathtt{s}j+\mathtt{l}-1,\mathtt{c}} \\ &= x^{\mathtt{s}i+\mathtt{k}-1,\mathtt{s}j+\mathtt{l}-1,t} \end{aligned} \tag{7.31}$$

and for deconvolutional layers according to Eq. (7.21) as

$$\begin{aligned} \Omega^{ijr}_{\mathtt{klt}r} &= \frac{\partial}{\partial W^{\mathtt{klt}r}} \sum_{\mathtt{k}=1}^{\mathtt{K}} \sum_{\mathtt{l}=1}^{\mathtt{L}} \sum_{\mathtt{f}=1}^{\mathtt{F}} W^{\mathtt{klfc}} \, x^{(i-\mathtt{k})/\mathtt{s}+1,(j-\mathtt{l})/\mathtt{s}+1,\mathtt{f}} \\ &= x^{(i-\mathtt{k})/\mathtt{s}+1,(j-\mathtt{l})/\mathtt{s}+1,t} \ , \end{aligned} \tag{7.32}$$

where $x^{(i-\mathtt{k})/\mathtt{s}+1,(j-\mathtt{l})/\mathtt{s}+1,t}$ is defined only for $\frac{(i-\mathtt{k})}{\mathtt{s}}+1, \frac{(j-\mathtt{l})}{\mathtt{s}}+1 \in \mathbb{N}$.

Notice, that in both cases in the definition of Ω, the index r does not appear on the right side of the equation. This is not surprising, since in the convolutional case the weights are shared per output channel $\mathtt{f}$, and in the deconvolutional case the same for $\mathtt{c}$, therefore, we can drop the respective indices from Ω. We can rewrite Eq. 7.29 using Ω as

$$\frac{\partial}{\partial W^{\mathtt{klt}r}} \ln p\left(y^{ijr}|x;\theta^r\right) = \left(\frac{\partial}{\partial \theta^r} \ln p\left(y^{ijr}|x;\theta^r\right)\right) \Omega^{ij}_{\mathtt{klt}} \ . \tag{7.33}$$

Furthermore, we can vectorize the operation above by noticing that the second term $\Omega^{ij}_{\mathtt{klt}}$, which is a derivative of θ^r, is shared for all indices $\mathtt{klt}$, and therefore we can flatten the Ω-s to a vector as[9]

$$\frac{\partial}{\partial \mathrm{vect}\left(W^{\cdots r}\right)} \ln p\left(y^{ijr}|x;\theta^r\right) = \frac{\partial}{\partial \theta^r} \ln p\left(y^{ijr}|x;\theta^r\right) \mathrm{vect}\left(\Omega^{ij}\right) \ , \tag{7.34}$$

[9]The notation $\cdot$ means all the indices along the axis, therefore $W^{\cdots r}$ represents the r-th $3D$ sub-tensor of the 4-dimensional tensor W.

where Ω^{ij} is the tensor made out of $\Omega^{ij}_{\mathtt{kl}t}$ for all indices $\mathtt{kl}t$. In order to derive the final form of the FIM according to Eq. (7.27) we need to compute the Hessian of the loss, which results in

$$\begin{aligned}\frac{\partial^2}{\partial \text{vect}\left(W^{\cdots r}\right)\partial \text{vect}\left(W^{\cdots r}\right)^\top} \ln p\left(y^{ijr}|x;\theta^r\right) = \\ = \frac{\partial^2}{\partial\theta^r\partial(\theta^r)^\top} \ln p\left(y^{ijr}|x;\theta^r\right) \text{vect}\left(\Omega^{ij}\right)\text{vect}\left(\Omega^{ij}\right)^\top \quad .\end{aligned} \tag{7.35}$$

Based on Theorem 3 all other blocks of the Hessian for $\text{vect}\left(W^{\cdots r}\right)$ and $\text{vect}\left(W^{\cdots t}\right)$, where $r \neq t$ are 0.

Remark 10
Notice, that the formulation of the Hessian of the loss in Eq. (7.35) *applies to both types of distributions which we have studied thus far, i.e., Bernoulli and Gaussian.*

Finally, to define the FIM of either the convolutional or deconvolutional layers for binary and continuous variables, we only have to replace ①

$$\frac{\partial^2}{\partial\theta^r\partial(\theta^r)^\top} \ln p\left(y^{ijr}|x;\theta^r\right)$$

with the appropriate derivatives for the respective distributions. In the Bernoulli case where $y^{ijt} \sim \text{Bern}(\rho^{ijt})$, for $\theta^r = \rho^{ijt}$ we have

$$\frac{\partial^2}{\partial\rho^{ijt}\partial(\rho^{ijt})^\top} \ln p\left(y^{ijt}|x;\rho^{ijt}\right) = -\rho^{ijt}\left(1-\rho^{ijt}\right)$$

according to Eq. (5.1). In the Gaussian case, where $y^{ijt} \sim \mathcal{N}(\mu^{ijt}, \sigma^{ijt})$, for $\theta^r = \mu^{ijt}$ we have

$$\frac{\partial^2}{\partial\mu^{ijt}\partial(\mu^{ijt})^\top} \ln p\left(y^{ijt}|x;\mu^{ijt}\right) = -\frac{1}{\left(\sigma^{ijt}\right)^2}$$

as in Eq. (7.8) and for $\theta^r = \sigma^{ijt}$ we have

$$\frac{\partial^2}{\partial\sigma^{ijt}\partial(\sigma^{ijt})^\top} \ln p\left(y^{ijt}|x;\sigma^{ijt}\right) = -\frac{2\left(y^{ijt}-\mu^{ijt}\right)^2}{\left(\sigma^{ijt}\right)^2}$$

as in Eq. (7.9).

Having all the ingredients we can define a block of the FIM for the parameters of $W^{\cdots r}$ as

$$\begin{aligned}\mathcal{F}_p^{l,r} &= -\mathbb{E}_{p(x,h)}\left[\sum_{i=1}^{\mathtt{H}'_l}\sum_{j=1}^{\mathtt{W}'_l}\nabla^2_{\theta^{l,r}}\ln p\left(h^{l,ijr}|h^{l+1};\theta^{l,r}\right)\right]\\ &= -\mathbb{E}_{p(x,h)}\left[\sum_{i=1}^{\mathtt{H}'_l}\sum_{j=1}^{\mathtt{W}'_l}\frac{\partial^2\ln p\left(h^{l,ijr}|x;\theta^r\right)}{\partial\theta^r\partial(\theta^r)^\top}\mathrm{vect}\left(\Omega^{l+1,ij}\right)\mathrm{vect}\left(\Omega^{l+1,ij}\right)^\top\right]\\ &= -\sum_{i=1}^{\mathtt{H}'_l}\sum_{j=1}^{\mathtt{W}'_l}\mathbb{E}_{p(x,h)}\left[\frac{\partial^2\ln p\left(h^{l,ijr}|x;\theta^r\right)}{\partial\theta^r\partial(\theta^r)^\top}\mathrm{vect}\left(\Omega^{l+1,ij}\right)\mathrm{vect}\left(\Omega^{l+1,ij}\right)^\top\right],\end{aligned} \tag{7.36}$$

where $\Omega^{l+1,ij}$ is the Ω^{ij} function for the $l+1$-th layer.

Remark 11
From Eq. (7.36) *it follows that all versions of the convolutional and deconvolutional FIMs, whether they are Bernoulli or Normal distributions can be written as a product of matrices in the form of* $UQU^\top$.

It is clear to see that for the $UQU^\top$ representation of layers using Bernoulli and Normal distributions only differ in Q matrices (see Remark 8). Notice that we can use the same sampling technique from Section 5.2 to estimate the block of the FIM as

$$\begin{aligned}F_p^{l,r} &= \frac{1}{n}\sum_{i=1}^{n}\sum_{i=1}^{\mathtt{H}'_l}\sum_{j=1}^{\mathtt{W}'_l}\frac{-\partial^2\ln p\left(h^{l,ijr}|x;\theta^r\right)}{\partial\theta^r\partial(\theta^r)^\top}\mathrm{vect}\left(\Omega^{l+1,ij}\right)\mathrm{vect}\left(\Omega^{l+1,ij}\right)^\top\\ &\text{with}\ \ \Omega^{l+1,ij}\subseteq h^{l+1}\sim q(h^{l+1}|h^l)\ ,\end{aligned}$$

where $n = B\cdot S$, B is the size of the minibatch and S is the sample size.

The only difference between the $UQU^\top$ representation of convolutional/ deconvolutional and dense layers is that U matrices grow larger because the former take more samples per parameters θ^r thanks to the shared weights over height $\mathtt{H}'$ and width $\mathtt{W}'$ of the output. While for dense layers U is of size $l_i \times n$, where $n = B\cdot S$, for convolutional layers n_c becomes $n_c = B\cdot S\cdot \mathtt{W}_l\cdot \mathtt{H}_l$ and for deconvolutional layers $n_d = B\cdot S\cdot \mathtt{W}_{l+1}\cdot \mathtt{H}_{l+1}$. Thanks to the higher number of samples for convolutional and deconvolutional layers the estimation of their associated FIMs become more accurate, compared to a dense network with the same number of input/output variables.

We notice that in both the convolutional and deconvolutional case the FIM is block-diagonal along the output layers' channels according to Eq. (7.26), which is axis `C` for a convolutional layer and axis `F` for a deconvolutional one. This means that for a convolutional layer instead of having blocks the

size of $(\mathtt{K} \cdot \mathtt{L} \cdot \mathtt{F} \cdot \mathtt{C})^2$ we get $\mathtt{C}$ blocks of the size $(\mathtt{K} \cdot \mathtt{L} \cdot \mathtt{F})^2$ in the case of the deconvolutional layer and $\mathtt{F}$ blocks of the size of $(\mathtt{K} \cdot \mathtt{L} \cdot \mathtt{C})^2$.

Another consequence of Remark 11 is that we can reuse all derivations from Section 5.3 (Tikhonov Regularization, Shermann-Morisson-Woodbury method, and K-step) to invert and regularize the FIM of convolutional and deconvolutional layers in an HM. Hence we can use the NRWS as defined in Algorithm 4 using these layers without any modifications.

7.4 Convolutional Helmholtz Machines

We start by performing experiments where we compare RWS and NRWS using CNNs. We use continuous variables in all layers of the HM as shown in Section 7.1, with the same hyperparameters as we used in our previous experiments (damping factor $\alpha = 0.2$, $K = 1,000$, batch size $B = 32$ and sample size $S = 10$), except learning rate η. The learning rate has to be smaller compared to the binary case and compared to the continuous case without CNNs because the sharing of the weights in the convolutional and deconvolutional layers speeds up the convergence of the respective layers by summing up the gradients. We chose the value of $\eta = 10^{-5}$.

One of the advantages of using CNNs is that the total number of weights gets reduced, compared to a dense layer, therefore we can use a higher resolution version of the TFD dataset, where the images are 48×48, compared to the 24×24 pixel images we have used so far. Using larger images generally benefits convolutional layers (depending on the filter sizes $\mathtt{K}$ and $\mathtt{L}$), as their "field of view" (convolutional window) is focused on a smaller area of the image, therefore, convolutional layers are capable of capturing smaller details/features. Being able to use larger images with HMs was also one of our primary goals to derive the NRWS for CNNs.

We start by evaluating the performance of CNNs in the non-geometric setting by comparing the use of convolutional and dense layers in an HM. Note that by construction, for an HM we have the restriction that layer sizes in the p and q networks must match. However, that does not mean that for every deconvolutional layer in p, a convolutional one in q is enforced or vice-versa. Using one type of layer in one network does not set any restriction on the other one as long as the size of the input/output layers is kept the same. Hence, one is free to choose combinations like convolutional/deconvolutional, convolutional/dense, dense/deconvolutional, or dense/dense for a given layer pair. In case we use the asymmetric pairs, we have to take care to flatten or reshape the input-output nodes for the respective layers.

We saw in our preliminary experiments that replacing dense layers with

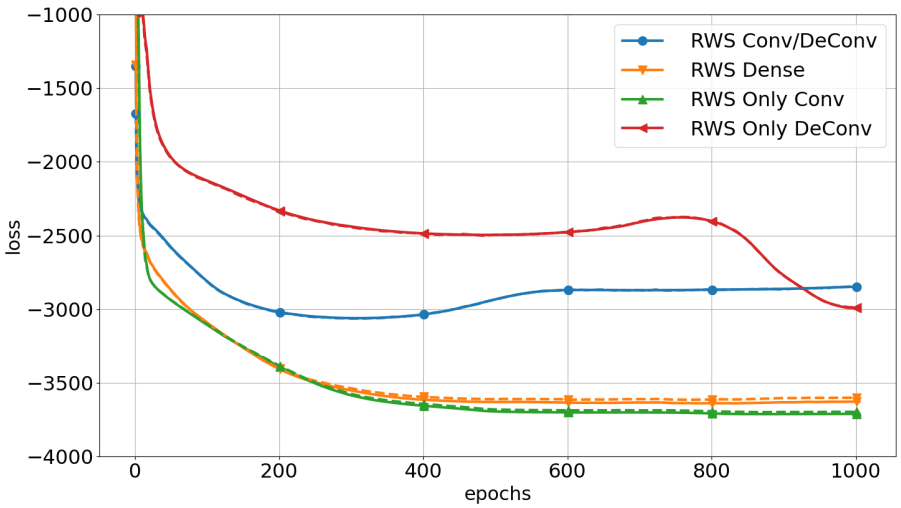


Figure 7.8: Curves of the loss in terms of epochs on the TFD dataset using the RWS with different network configurations for the bottom layer: Conv/ DeConv = convolutional q/ deconvolutional p, Dense = dense q/ dense p, Only Conv = convolutional q/ dense p and Only DeConv = dense q/ deconvolutional p .

convolutional and, particularly deconvolutional ones decreases the performance and introduces some unexpected behavior. We experimented with replacing multiple dense layers with convolutional/deconvolutional layers, where we saw that the more dense layers we replace, the worse the performance gets, even if we increased the number of filters per layer.

To illustrate this issue, we performed experiments using only RWS, with 4 different layer configurations for the bottom layer l_0: dense/dense, convolutional/dense, dense/ deconvolutional, and convolutional/deconvolutional for the q/p distribution pairs. In order to test the impact of the new types of layers, we keep the top of the HM by using the exact same network structure from layers l_1 to l_M, i.e., dense layers of the size $75, 50, 35, 25, 20, 20$. When we use the convolutional/deconvolutional layers for $q(h^1|x)$ and $p(x|h^1)$ we use 3×3 convolutional filters, with a stride of 2 and 8 output channels (8 input channels in the deconvolutional case). In Figure 7.8 we show the losses associated with the 4 experiments.

We notice that the network architecture using only dense layers (Dense) and the one where we have only one convolutional layer in the network q but no deconvolutional layer in p (Only Conv), perform comparably well, with the latter being slightly better. The network architectures that use

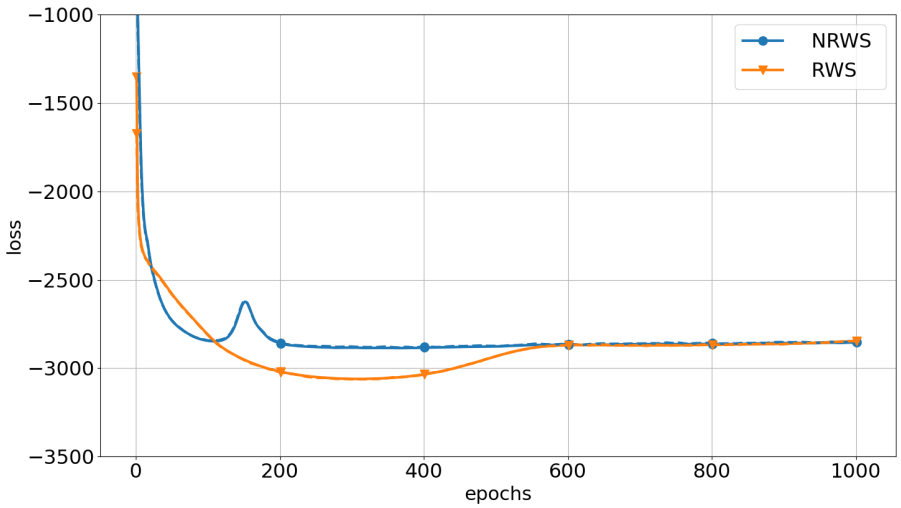


Figure 7.9: Loss curves for TFD in epochs using RWS and NRWS to train an HM with one convolutional/deconvolutional layer at the bottom of the network, continuous lines represent the quantities on the train set, and dashed lines the ones on validation.

both a convolutional and a deconvolutional layer (Conv/ DeConv) or just a deconvolutional one in p (Only DeConv) underperform significantly to the former two, with some unexpected behaviors in the curves of the loss.

Recall that these experiments were performed without using NG, which suggests that there is a problem with using convolutional and deconvolutional layers with HM independently of the training algorithm used. However, the sub-optimal performance of the Conv/ DeConv and Only DeConv could be explained by these structures being harder to train for HMs using training algorithms like the RWS.

We performed our next experiments with CNNs using the NRWS. We kept the top l_1 to l_M layers as before and we replaced the bottom layer l_0 with a convolutional layer in $q(h^1|x)$ and a deconvolutional layer in $p(x|h^1)$ with 3×3 convolutional filters, with a stride of 2 and 8 output channels (8 input channels in the deconvolutional case)[10].

In Figure 7.9, we compare the losses of RWS and NRWS trained on the same HM. One can notice that both algorithms converge to the same minimum of the loss, with the NRWS converging slightly faster, however, in

[10]We needed to restrict our test to using only the Conv/ DeConv setting because the size of the FIM of a dense layer, for the resolution we are using, ran into a memory limit on our testing hardware (which is described in Appendix A).

both cases, the curves have some unexpected behavior. The NRWS has a bump in the loss briefly around epoch 150, while the RWS seems to perform slightly better for a period between epochs 200 to 500 before converging.

7.5 Comments about the experiments

In the following, we form some hypotheses to explain the unexpected behavior we have noticed in our experiments. Such consideration could present insights useful for the design of more effective strategies for HMs employing CNNs.

If we take a look at the literature, we see that there are but a few examples of convolutional structures being used with stochastic networks, specifically Belief Networks, such as RBMs. The few examples we can find are typically based on the works of Norouzi et al. (2009) and Lee et al. (2009), and usually employ some atypical approaches to make use of convolutional layers with the statistical model, such as relaxing the weight-sharing requirement and using only a few layers (typically 1 or 2). However, there have also been stochastic models that use convolutional layers very successfully, in particular, the ones based on VAEs, starting from the work of Kingma and Welling (2014)[11], even if typically the stochastic layers are associated with the latent variables and do not appear in the encoder and decoder networks.

In our hypothesis, two main observations contribute to why the HM does not show improvement with convolutional and, in particular, deconvolutional layers. First, there is an inherent limitation given by the conditional independence structure of the HM, secondly, the convolutional and deconvolutional layers are constrained by their topology and the sharing of their weights.

Starting from the first observation, we can notice immediately that there is a structural difference between, from one side, the HM and RBM, where it appears that the convolutional structure cannot be employed in the topologies as is, and from the other the VAE, where it can be. The difference is that the former two typically employ a Markov-chain structure for each layer of both recognition and generative networks, while the latter is usually composed of a distribution per network, which does not decompose per layers[12].

The consequence of having a Markov-chain structure for the joint distri-

[11] In Sections 2.4.2 and 2.4.1 we present in details RBMs and VAEs

[12] In this section when we discuss the inherent structure of statistical models, we are referring to the typical formulation of those models, like HM, RBM and VAE. We are aware of more recent works, such as the ones of Sohn et al. (2015), Goyal et al. (2017), Chen et al. (2020), etc., which use hybrid solutions combined from the aforementioned structures, that blur the lines between these stochastic models, but we will not discuss them in this setting, as they would have to be individually addressed.

bution of $p(x, h)$, which we are using to learn $p_{\mathcal{D}}(x)$, is that the optimization can be executed layer-wise as in HMs, compared to VAEs, which use back-propagation. This could be viewed as an advantage for an HM since it allows us to formulate the FIM as a block-diagonal matrix and use NRWS efficiently to train our models. However, this layered structure introduces a constraint, that the distribution of each layer $p(h^i|h^{i+1})$ has to be as close as possible to its counterpart in the corresponding layer $q(h^{i+1}|h^i)$, through the optimization steps which minimize

$$\begin{aligned} &\mathrm{D}_{KL}\left[p(h^i|h^{i+1})||q(h^{i+1}|h^i)\right] \quad \text{for the wake phase and} \\ &\mathrm{D}_{KL}\left[q(h^{i+1}|h^i)||p(h^i|h^{i+1})\right] \quad \text{for the sleep phase.} \end{aligned} \tag{7.37}$$

The VAE based methods typically do not have this constraint, as $p(x, h)$ and $q(h|x)$ usually do not decompose into stochastic layers in such structures. As such, we can think of the VAE as a one-layer belief network, where the decoder and encoder networks are treated as one, very expressive, stochastic layer in p and q respectively. Note that there are variants of VAEs (e.g., Ladder Variational Autoencoder (Sønderby et al., 2016)), where the hidden layers decompose in some way, however, usually in these cases the first hidden layer in both distributions p and q correspond to the decoder and encoder network, and the rest of the layers form the bottleneck as a small Markov-chain. In the bottleneck layers, the constraint illustrated in our first point still applies.

For our second observation, let us compare the simplest possible cases of a dense layer versus a convolutional layer[13], where we have a feed-forward network with 3 input nodes and 2 output nodes, with a sigmoid activation. In one case we have the nodes fully connected (dense layer), and in the other, we connect them with a convolutional filter of size 2, as in Figure 7.10.

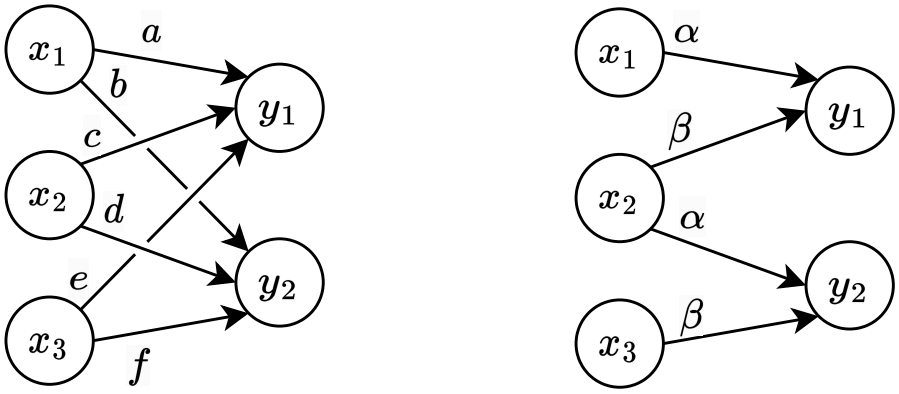
We can formulate both distributions with the formula

$$y \sim \text{Bernoulli}\left(s\left((W)^\top \cdot x\right)\right) \ ,$$

where y is the output vector, sampled from the Bernoulli distribution having as mean parameter the output of the sigmoid function s and the linear transformation[14] with the matrix W of the input vector x. The difference between dense and convolutional layers is the formulation of the matrix W.

[13] We illustrate our points on a convolutional layer for simplicity, but it applies analogously to a deconvolutional layer as well.

[14] A linear layer is typically formulated as $W \cdot x + b$, but for simplicity, we ignore the bias term b, as it is not important for our current argument.



(a) Dense layer with weights $\{a, b, c, d, e, f\}$.

(b) Convolutional layer with shared weights $\{\alpha, \beta\}$.

Figure 7.10: Toy examples of feed-forward networks with 3 inputs, vector x, and 2 outputs, vector y, with (a) dense layer and (b) convolutional layer.

In the case of a dense layer, the matrix can be described as a dense matrix

$$W_d = \begin{pmatrix} a & b \\ c & d \\ e & f \end{pmatrix} ,$$

with $\{a, b, c, d, e, f\}$ individual weights, while a convolutional layer can be described by a so-called Toeplitz matrix (Petersen et al., 2008) as

$$W_c = \begin{pmatrix} \alpha & 0 \\ \beta & \alpha \\ 0 & \beta \end{pmatrix} ,$$

with $\{\alpha, \beta\}$ individual weights. The individual weights of W_d and W_c are represented as arrows in Figure 7.10. Notice, that W_c for the convolutional layer contains 0-s in the corners of the matrix, which correspond to the missing connections compared to the dense layer. Furthermore, since a convolutional layer shares its weights through a sliding filter, we have only two weights $\{\alpha, \beta\}$ controlling the linear transformation, compared to the six degrees of freedom of the dense layer $\{a, b, c, d, e, f\}$. From this simple example, we can immediately tell that a convolutional layer is less expressive than a dense layer, in fact, it is a special case of the latter. Notice, that the effect of the lower expressivity of a single convolutional layer is amplified when increasing the sizes of the input/output layers for a fixed filter size.

While we know that a densely connected feed-forward network, with a single hidden layer, is a universal approximator (Hornik et al., 1989)[15], the same cannot be said about a convolutional network with **one** hidden layer. However, a convolutional network can be a universal approximator given enough layers (Zhou, 2020). Hence, the reduced expressivity of the convolutional layers is not a drawback, in fact, it is a strength because a convolutional network can trade off the expressivity of one layer, for extracting local information. Therefore a CNN can achieve the same performance as a dense layer using less weights per layer but multiple layers.

Our two observations then are as follows: First, the nature of the HM dictates that layers of the generation and recognition networks have to be forced to be close (in the Kullback-Leibler sense). Second, convolutional/deconvolutional layers are less expressive than dense ones. As a convolutional layer is trying to learn the inference problem at hand, it is also forced to learn an approximate distribution to the corresponding layer of the opposite network, see Eq. (7.37). If the corresponding layer is a deconvolutional layer, with similarly few weights, the problem is even more difficult, as their mutual reduced expressivity works to their detriment when trying to learn each others' inverse. The reduced expressivity of a convolutional layer combined with the layerwise constraint of HMs results in the sub-optimal generalization capabilities of layers including convolutional and deconvolutional layers compared to fully dense layers. Based on these observations, we hypothesize that unless a modification is done in the way CNNs are employed, HM cannot take advantage of convolutional and deconvolutional structures.

[15]In this context we understand the Universal Approximation Theorem as stating that any continuous function $f : [0, 1]^n \to \mathbb{R}$, where $y = f(x)$, with inputs $x = [0, 1]^n$, can be approximated arbitrarily well by a neural network with at least one sufficiently large hidden layer (Hecht-Nielsen, 1987).

Chapter 8

Conclusions

In this thesis, we showed how HMs can be effectively trained using the Natural Gradient (NG), thanks to properties of Sigmoid Belief Networks (SBNs), which allow efficient computation of the Fisher Information Matrix (FIM). Indeed, by exploiting the locality of the connection matrix given by the network topology of SBNs, the structure of the FIM is a smaller-grained block-diagonal matrix than what is generally used in the literature. In such models, it is not required to introduce any extra conditional independence assumption between random variables for computing the Natural Gradient (NG) efficiently, due to the sparse structure of the FIM and the use of coarser representations of its blocks in terms of low-rank updates of diagonal matrices.

We introduced the Natural Reweighted Wake-Sleep (NRWS) algorithm and we demonstrated an improvement of the convergence during training for stochastic gradient descent. NRWS was not only faster to converge, both in time and number of epochs, but the obtained optimum resulted in better values for the likelihood estimation compared to the Wake-Sleep (WS)(Hinton et al., 1995), Reweighted Wake-Sleep (RWS) (Bornschein and Bengio, 2015) and Bidirectional Helmholtz Machine (BiHM) (Bornschein et al., 2016) algorithms. Our findings have been corroborated by experiments on MNIST as well as on continuous datasets such as FashionMNIST and Toronto Face Dataset (TFD). Furthermore, on these datasets, NRWS outperformed RWS and BiHM also exhibiting a better generalization gap. These results defined new state-of-the-art performance for HMs on these datasets, concerning not only WS and RWS, but also in comparison to the more recent BiHM. The faster convergence in terms of wall-clock time is indeed a key aspect of using NRWS, as it is not common in the literature that a training algorithm based on NG can compete with non-geometric ones in this aspect.

The K-step update version of the NRWS algorithm showed considerable speed-up in terms of training time, without a decrease in performance, with respect to its baseline ($K = 1$). Noticeably, we showed how in our experiments a delayed, and thus less accurate estimation of the FIM, was enough to achieve state-of-the-art performances for HMs. Additionally, the damping factor introduced in training effectively acts as a regularizer reducing the gap between train and validation.

We studied the computational complexity of the algorithm which is dominated by $\mathcal{O}\left(l_0\left(l_1 n + n^{2.376}\right)\right)$ where l_0 and l_1 are the bottom two layers of the HM, which are usually the largest, and $n = B * S$ is the minibatch size times the sample size. Notice that the computation of the NG requires storing the FIM, which has increased memory usage compared to the vanilla gradient, even if the compact representations of the matrix are used. Hence, the current version of the algorithm designed for dense SBNs is unsuitable for datasets with high resolution and color images due to the computational complexity depending on $l_0 * l_1$. This is not considered a limiting factor, since dense layers are not usually employed for such datasets in favor of more efficient networks based on convolutional filters.

We introduced the Diagonal Natural Reweighted Wake-Sleep (DNRWS), which has a simplified diagonal FIM, to check whether the FIM of the HM can be approximated more efficiently. We have shown that the exact FIM used in the NRWS cannot be further simplified and approximated without loss of accuracy since the DNRWS significantly underperformed compared to the NRWS but also to the RWS, as it failed to converge to a reasonable minimum of the loss in all tested settings.

We developed the Natural Bidirectional Helmholtz Machine (NBiHM), by using the FIM computed for the NRWS with the training steps and losses of the BiHM. While the FIM is not the appropriate one, because it should be derived from the special p^* loss of the BiHM instead, we can still reasonably use it because of the underlying HM structure, for which the matrix was defined. We showed through experiments, that the NBiHM, still improves on the BiHM in terms of convergence speed and reached optimization minima, however, it does not reach the performance of the NRWS.

While we studied the convergence properties of the NRWS we found a simple refinement, which we named the Rescaled-sleep (RS), that we can use on all methods based on the WS to improve the convergence compared to their baseline. The RS is based on the fact that we approximate the performance of the Sleep-well (SW) variant of the WS algorithm (which is guaranteed to converge to the optimum), by simply scaling up the gradients of the sleep phase, so that the algorithm mimics the *em* algorithm. Hence, the RS can take larger steps in the sleep phase, which is much less computationally expensive, instead of doing multiple in the case of SW, while achieving almost identical performance.

Next, we adapted gradient acceleration and regularization to the geometric NRWS. We noticed that with a couple of simple assumptions, we can use first-order Accelerated Gradients (AG) methods on NRWS, such as the Momentum (Jacobs, 1988) and Nesterov Momentum (Nesterov) (Nesterov, 1983) to improve the convergence rate of the algorithm, without noticeable

drawbacks, still outperforming its non-geometric counterpart using the same method. Using second-order AG methods on NRWS, like Adam (Kingma and Ba, 2015), underperformed compared to the first-order methods, while still improving on the non-accelerated version. We hypothesized that the reason why the second-order methods did not improve is due to the fact that our assumptions about the local regularity of the manifold are too strong beyond the first-order.

During our experiments with the regularization of the NRWS, we have found that $\mathsf{L}^1/\mathsf{L}^2$ Regularizerss (LRs) did not improve on the performance of our algorithm because of the effects the constraining the norm of the weights has on stochastic networks. Therefore, we introduced a new regularization method called the Entropy Regularizer (ER), based on the Maximum Entropy Principle (Jaynes, 1957). We showed through experiments that, by adding a term to the loss that boosts the entropy of the model, the ER can improve on all WS based models, including the NRWS, with no noticeable drawbacks.

We evaluated the NRWS with both AG and ER, and we showed that we could improve on our state-of-the-art result, on all tested datasets: MNIST, FashionMNIST and TFD.

Finally, we explored the possibilities of expanding the capabilities of the NRWS by exploring how the algorithm behaves when using continuous random variables instead of binary ones, and how it could profit from using convolutional network structures.

We derived the FIM for a layer of the HM using Normally distributed random variables and we found a very similar structure compared to the binary case, which preserves the property of being finely-grained block-diagonal, with 2 blocks per node in a layer. We could efficiently compute the inverse of the FIM, thanks to its structure, which enabled us to use continuous variables with the NRWS. We showed through experiments done on TFD that continuous random variables can help us to train a HM in fewer steps to similar results as with binary variables.

We proved that by using convolutional and deconvolutional layer structures in an HM we can still describe the model's geometry with a finely-grained block-diagonal FIMs. In practice, however, using a HM with convolutional layers resulted in lower performance than expected. We formulated two hypotheses about why we think these types of layers do not work well with HMs, and why other generative models do not have this constraint.

We started this thesis with the goal of exploring the possibilities and benefits of training Helmholtz Machines using Information Geometry. Throughout it, we have developed a novel algorithm called the Natural Reweighted Wake-

Sleep, that introduced Natural Gradient learning to HMs by building on the Wake-Sleep algorithm. During our research, we have designed geometric variants of common machine learning techniques in training, such as gradient acceleration and regularization, which we could use to improve our algorithm and reach state-of-the-art performance on HMs. We improved the expressiveness of HMs, through the use of continuous variables, while maintaining the benefits of geometric learning and we derived the geometry of their convolutional and deconvolutional layers.

Although this thesis does not exhaust all the possibilities of the proposed research question, we have made significant contributions to the field of IG and HM, with numerous findings that could be further explored, or just reused in other research fields. Our results can represent a starting point for future research on improving training algorithms for Neural Networks and Deep Learning models using geometric algorithms, such as the NG.

Bibliography

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. (2015). TensorFlow: Large-scale machine learning on heterogeneous systems. *URL https://www.tensorflow.org*.

Absil, P.-A., Mahony, R., and Sepulchre, R. (2009). Optimization algorithms on matrix manifolds. In *Optimization Algorithms on Matrix Manifolds*. Princeton University Press.

Alimisis, F., Orvieto, A., Bécigneul, G., and Lucchi, A. (2020). A continuous-time perspective for modeling acceleration in Riemannian optimization. In *International Conference on Artificial Intelligence and Statistics*.

Alimisis, F., Orvieto, A., Becigneul, G., and Lucchi, A. (2021). Momentum improves optimization on Riemannian manifolds. In *International Conference on Artificial Intelligence and Statistics*.

Amari, S.-i. (1985). Differential-geometrical methods in statistics. *Lecture Notes on Statistics*, 28.

Amari, S.-i. (1995). Information geometry of the EM and em algorithms for neural networks. *Neural Networks*, 8(9).

Amari, S.-i. (1997). Neural learning in structured parameter spaces-natural Riemannian gradient. In *Advances in Neural Information Processing Systems*.

Amari, S.-I. (1998). Natural gradient works efficiently in learning. *Neural Computation*, 10.

Amari, S.-i. (2016). *Information geometry and its applications*, volume 194. Springer.

Amari, S.-i. and Nagaoka, H. (2000). *Methods of information geometry*, volume 191. American Mathematical Soc.

Arjovsky, M., Chintala, S., and Bottou, L. (2017). Wasserstein generative adversarial networks. In *International Conference on Machine Learning*.

Audiffren, J., Valko, M., Lazaric, A., and Ghavamzadeh, M. (2015). Maximum entropy semi-supervised inverse reinforcement learning. In *International Joint Conference on Artificial Intelligence*.

Ay, N. (2002). Locality of global stochastic interaction in directed acyclic networks. *Neural Computation*, 14(12).

Ay, N. (2020). On the locality of the natural gradient for learning in deep Bayesian networks. *Information Geometry*.

Ay, N., Jost, J., Vân Lê, H., and Schwachhöfer, L. (2017). *Information geometry*, volume 64. Springer.

Bahamou, A., Goldfarb, D., and Ren, Y. (2022). A mini-block natural gradient method for deep neural networks. *arXiv:2202.04124*.

Bécigneul, G. and Ganea, O.-E. (2018). Riemannian adaptive optimization methods. *arXiv preprint arXiv:1810.00760*.

Bengio, Y. (2009). Learning deep architectures for AI. *Foundations and Trends® in Machine Learning*, 2(1).

Bishop, C. M. (2006). *Pattern recognition and machine learning*. springer.

Bornschein, J. and Bengio, Y. (2015). Reweighted Wake-Sleep. *International Conference on Learning Representations*.

Bornschein, J., Shabanian, S., Fischer, A., and Bengio, Y. (2016). Bidirectional Helmholtz Machines. In *International Conference on Machine Learning*.

Bourlard, H. and Kamp, Y. (1988). Auto-association by multilayer perceptrons and singular value decomposition. *Biological Cybernetics*, 59(4).

Braverman, M. (2011). *COS 597D: Information theory in computer science Course*, chapter 3. Princeton University – Department of Computer Science.

Čencov, N. (1978). Algebraic foundation of mathematical statistics. *Statistics: A Journal of Theoretical and Applied Statistics*, 9(2).

Chen, J., Yu, Z., and Gu, Z. (2020). Semi-supervised deep learning in motor imagery-based brain-computer interfaces with stacked variational autoencoder. In *Journal of Physics: Conference Series*, volume 1631.

Chirco, G., Malagò, L., and Pistone, G. (2022). Lagrangian and hamiltonian dynamics for probabilities on the statistical bundle. *International Journal of Geometric Methods in Modern Physics*, 19(13).

Cowell, R. G., Dawid, P., Lauritzen, S. L., and Spiegelhalter, D. J. (2007). *Probabilistic networks and expert systems: Exact computational methods for Bayesian networks*. Springer Science & Business Media.

DasGupta, A. (2011). The exponential family and statistical applications. In *Probability for statistics and machine learning*. Springer.

Dayan, P. (2000). Helmholtz Machines and Wake-Sleep learning. *Handbook of Brain Theory and Neural Network. MIT Press, Cambridge, MA*, 44(0).

Dayan, P. and Hinton, G. E. (1996). Varieties of Helmholtz Machine. *Neural Networks*, 9(8).

Dayan, P., Hinton, G. E., Neal, R. M., and Zemel, R. S. (1995). The Helmholtz Machine. *Neural computation*, 7(5).

Decell, Jr, H. P. (1965). An application of the Cayley-Hamilton theorem to generalized matrix inversion. *SIAM review*, 7(4).

Dekel, O., Gilad-Bachrach, R., Shamir, O., and Xiao, L. (2012). Optimal distributed online prediction using mini-batches. *Journal of Machine Learning Research*, 13(1).

Deng, L. (2012). The MNIST database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine*, 29(6).

Desjardins, G., Pascanu, R., Courville, A., and Bengio, Y. (2013). Metric-free natural gradient for joint-training of Boltzmann Machines. *International Conference on Learning Representations*.

Desjardins, G., Simonyan, K., Pascanu, R., et al. (2015). Natural neural networks. In *Advances in Neural Information Processing Systems*.

Dozat, T. (2016). Incorporating Nesterov momentum into ADAM. *Workshop track - International Conference on Learning Representations 2016*.

Duchi, J., Hazan, E., and Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12.

Fornalski, K., Parzych, G., Pylak, M., Satuła, D., and Dobrzyński, L. (2010). Application of Bayesian reasoning and the maximum entropy method to some reconstruction problems. *Acta Physica Polonica A*, 117(6).

Fujiwara, A. and Amari, S.-i. (1995). Gradient systems in view of information geometry. *Physica D: Nonlinear Phenomena*, 80(3).

Glasmachers, T., Schaul, T., Yi, S., Wierstra, D., and Schmidhuber, J. (2010). Exponential natural evolution strategies. In *Proceedings of the 12th Conference on Genetic and Evolutionary Computation*.

Glorot, X. and Bengio, Y. (2010a). Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the 13th International Conference on Artificial Intelligence and Statistics*.

Glorot, X. and Bengio, Y. (2010b). Understanding the difficulty of training deep feedforward neural networks. In Teh, Y. W. and Titterington, M., editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014). Generative adversarial nets. In *Advances in Neural Information Processing Systems*.

Goyal, P., Hu, Z., Liang, X., Wang, C., and Xing, E. P. (2017). Nonparametric variational auto-encoders for hierarchical representation learning. In *Proceedings of the IEEE International Conference on Computer Vision*.

Grathwohl, W., Choi, D., Wu, Y., Roeder, G., and Duvenaud, D. (2018). Backpropagation through the void: Optimizing control variates for black-box gradient estimation. In *International Conference on Learning Representations*.

Grosse, R. and Martens, J. (2016). A Kronecker-factored approximate Fisher matrix for convolution layers. In *International Conference on Machine Learning*.

Grosse, R. and Salakhudinov, R. (2015). Scaling up natural gradient by sparsely factorizing the inverse Fisher matrix. In *International Conference on Machine Learning*.

Guiasu, S. and Shenitzer, A. (1985). The principle of maximum entropy. *The Mathematical Intelligencer*, 7(1).

Haarnoja, T. (2018). *Acquiring diverse robot skills via maximum entropy deep reinforcement learning*. University of California, Berkeley.

Haarnoja, T., Zhou, A., Abbeel, P., and Levine, S. (2018). Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International Conference on Machine Learning*.

Hecht-Nielsen, R. (1987). Kolmogorov's mapping neural network existence theorem. In *Proceedings of the international conference on Neural Networks*, volume 3, pages 11–14. IEEE press New York, NY, USA.

Helmholtz, H. v. (1882). On the thermodynamics of chemical processes. *Physical memoirs selected and translated from foreign sources*, 1.

Hewitt, L., Le, T. A., and Tenenbaum, J. (2020). Learning to learn generative programs with Memoised Wake-Sleep. In *Conference on Uncertainty in Artificial Intelligence*.

Hinton, G. E. (2002). Training products of experts by minimizing contrastive divergence. *Neural Computation*, 14(8).

Hinton, G. E., Dayan, P., Frey, B. J., and Neal, R. M. (1995). The "Wake-Sleep" algorithm for unsupervised neural networks. *Science*, 268(5214).

Hinton, G. E., Osindero, S., and Teh, Y.-W. (2006). A fast learning algorithm for deep belief nets. *Neural Computation*, 18(7).

Hinton, G. E. and Salakhutdinov, R. R. (2006). Reducing the dimensionality of data with neural networks. *Science*, 313(5786).

Hinton, G. E. and Zemel, R. (1993). Autoencoders, minimum description length and Helmholtz free energy. *Advances in Neural Information Processing Systems*, 6.

Hornik, K., Stinchcombe, M., and White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5).

Ikeda, S., Amari, S.-i., and Nakahara, H. (1999). Convergence of the Wake-Sleep algorithm. In *Advances in Neural Information Processing Systems*.

Jacobs, R. A. (1988). Increased rates of convergence through learning rate adaptation. *Neural Networks*, 1(4).

Jaynes, E. T. (1957). Information theory and statistical mechanics. *Physical review*, 106(4).

Jeffreys, H. (1946). An invariant form for the prior probability in estimation problems. *Proceedings of the Royal Society of London. Series A. Mathematical and Physical Sciences*, 186(1007).

Karras, T., Laine, S., and Aila, T. (2019). A style-based generator architecture for generative adversarial networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*.

Kass, R. E. and Vos, P. W. (2011). *Geometrical foundations of asymptotic inference*. John Wiley & Sons.

Keskar, N. S. and Socher, R. (2017). Improving generalization performance by switching from ADAM to SGD. *arXiv:1712.07628*.

Kingma, D. P. and Ba, J. (2015). Adam: A method for stochastic optimization. *International Conference on Learning Representations*.

Kingma, D. P. and Welling, M. (2014). Auto-encoding variational bayes. *International Conference on Learning Representations*.

Kirby, K. G. (2006). A tutorial on Helmholtz Machines. *Department of Computer Science, Northern Kentucky University*.

Kool, W., van Hoof, H., and Welling, M. (2020). Estimating gradients for discrete random variables by sampling without replacement. In *International Conference on Learning Representations*.

Kramer, M. A. (1991). Nonlinear principal component analysis using autoassociative neural networks. *American Institute of Chemical Engineers Journal*, 37(2).

Lauritzen, S. L. (1987). Statistical manifolds. *Differential Geometry in Statistical Inference*, 10.

Lauritzen, S. L. (1996). *Graphical Models*. Oxford University Press.

Le, T. A., Kosiorek, A. R., Siddharth, N., Teh, Y. W., and Wood, F. (2020). Revisiting Reweighted Wake-Sleep for models with stochastic control flow. In *Uncertainty in Artificial Intelligence*.

LeCun, Y. (1987). Phd thesis: Modeles connexionnistes de l'apprentissage (connectionist learning models). *Universite P. et M. Curie (Paris 6)*.

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11).

Lee, H., Grosse, R., Ranganath, R., and Ng, A. Y. (2009). Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. In *Proceedings of the 26th annual International Conference on Machine Learning*.

Lee, J. (2010). *Introduction to topological manifolds*, volume 202. Springer Science & Business Media.

Lee, J. M. (2013). Smooth manifolds. In *Introduction to smooth manifolds*. Springer.

Lee, J. M. (2018). *Introduction to Riemannian manifolds*. Springer.

Lin, W., Khan, M. E., Hubacher, N., and Nielsen, D. (2017). Natural-gradient stochastic variational inference for non-conjugate structured variational autoencoder. *Proceedings of the International Conference on Machine Learning 17 Workshop on Deep Structured Prediction*.

Lin, W., Nielsen, F., Emtiyaz, K. M., and Schmidt, M. (2021). Tractable structured natural-gradient descent using local parameterizations. In *International Conference on Machine Learning*.

Liu, Y., Shang, F., Cheng, J., Cheng, H., and Jiao, L. (2017). Accelerated first-order methods for geodesically convex optimization on riemannian manifolds. *Advances in Neural Information Processing Systems*, 30.

Lloyd, S. and Penfield, P. (2003). Information and entropy. *Course 6.050 J/2.110 J, Spring*.

Ma, G. (2021). A remark on the maximum entropy principle in uncertainty theory. *Soft Computing*, 25(22).

Martens, J. (2020a). New insights and perspectives on the natural gradient method. *Journal of Machine Learning Research*, 21(146).

Martens, J. (2020b). New insights and perspectives on the natural gradient method. *Journal of Machine Learning Research*, 21(1).

Martens, J. and Grosse, R. (2015). Optimizing neural networks with Kronecker-factored approximate curvature. In *International Conference on Machine Learning*.

McLachlan, G. J. and Krishnan, T. (2007). *The EM algorithm and extensions*, volume 382. John Wiley & Sons.

Mnih, A. and Gregor, K. (2014). Neural variational inference and learning in belief networks. In *International Conference on Machine Learning*.

Montúfar, G. (2015). Deep narrow Boltzmann Machines are universal approximators. *International Conference on Learning Representations*.

Montúfar, G. (2016). Restricted Boltzmann Machines: Introduction and review. In *Information Geometry and Its Applications IV*. Springer.

Montúfar, G., Ay, N., and Ghazi-Zahedi, K. (2015). Geometry and expressive power of conditional restricted Boltzmann Machines. *Journal of Machine Learning Research*, 16(1).

Montúfar, G. and Morton, J. (2015). Discrete restricted Boltzmann Machines. *Journal of Machine Learning Research*, 16(1).

Murray, M. and Rice, J. (1993). *Differential Geometry and Statistics*. Chapman.

Neal, R. M. (1990). Learning stochastic feedforward networks. *Department of Computer Science, University of Toronto*, 64(1283).

Neal, R. M. (1992). Connectionist learning of belief networks. *Artificial Intelligence*, 56(1).

Nesterov, Y. E. (1983). A method of solving a convex programming problem with convergence rate $O\left(k^2\right)$. In *Doklady Akademii Nauk*, volume 269-3. Russian Academy of Sciences.

Nielsen, F. (2020). An elementary introduction to information geometry. *Entropy*, 22(10).

Norouzi, M., Ranjbar, M., and Mori, G. (2009). Stacks of convolutional restricted Boltzmann Machines for shift-invariant feature learning. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE.

Ollivier, Y. (2015). Riemannian metrics for neural networks I: Feedforward networks. *Information and Inference: A Journal of the IMA*, 4(2).

Ollivier, Y., Arnold, L., Auger, A., and Hansen, N. (2017). Information-geometric optimization algorithms: A unifying picture via invariance principles. *Journal of Machine Learning Research*, 18(18).

Pajarinen, J., Thai, H. L., Akrour, R., Peters, J., and Neumann, G. (2019). Compatible natural gradient policy search. *Machine Learning*, 108.

Petersen, K. B., Pedersen, M. S., et al. (2008). The matrix cookbook. *Technical University of Denmark*, 7(15).

Polyak, B. T. (1964). Some methods of speeding up the convergence of iteration methods. *USSR Computational Mathematics and Mathematical Physics*, 4(5).

Ramesh, A., Pavlov, M., Goh, G., Gray, S., Voss, C., Radford, A., Chen, M., and Sutskever, I. (2021). Zero-shot text-to-image generation. In *International conference on machine learning*, pages 8821–8831. Pmlr.

Razavi, A., Van den Oord, A., and Vinyals, O. (2019). Generating diverse high-fidelity images with VQ-VAE-2. *Advances in Neural Information Processing Systems*, 32.

Rényi, A. (1961). On measures of entropy and information. In *Proceedings of the Fourth Berkeley Symposium on Mathematical Statistics and Probability, Volume 1: Contributions to the Theory of Statistics*, volume 4. University of California Press.

Rezende, D. J., Mohamed, S., and Wierstra, D. (2014). Stochastic back-propagation and approximate inference in deep generative models. In *International Conference on Machine Learning*.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature*, 323(6088).

Shore, J. and Johnson, R. (1980). Axiomatic derivation of the principle of maximum entropy and the principle of minimum cross-entropy. *IEEE Transactions on Information Theory*, 26(1).

Smolensky, P. (1986). Information processing in dynamical systems: Foundations of harmony theory. Technical report, Colorado Univ at Boulder Dept of Computer Science.

Sohn, K., Lee, H., and Yan, X. (2015). Learning structured output representation using deep conditional generative models. *Advances in Neural Information Processing Systems*, 28.

Sønderby, C. K., Raiko, T., Maaløe, L., Sønderby, S. K., and Winther, O. (2016). Ladder variational autoencoders. *Advances in Neural Information Processing Systems*, 29.

Srinivasan, R. (2013). *Importance sampling: Applications in communications and detection*. Springer Science & Business Media.

Sun, K. and Nielsen, F. (2017). Relative Fisher information and natural gradient for learning large modular models. In *Proceedings of the 34th International Conference on Machine Learning*.

Susskind, J. M., Anderson, A. K., and Hinton, G. E. (2010). The Toronto face database. *Department of Computer Science, University of Toronto, Toronto, ON, Canada, Tech. Rep*, 3.

Tucker, G., Mnih, A., Maddison, C. J., Lawson, D., and Sohl-Dickstein, J. (2017). Rebar: Low-variance, unbiased gradient estimates for discrete latent variable models. In *31st Conference on Neural Information Processing Systems*.

Van Den Oord, A., Vinyals, O., et al. (2017). Neural discrete representation learning. *Advances in Neural Information Processing Systems*, 30.

Várady, C., Volpi, R., Malagò, L., and Ay, N. (2022). Natural reweighted wake–sleep. *Neural Networks*, 155.

Vaserstein, L. N. (1969). Markov processes over denumerable products of spaces, describing large systems of automata. *Problemy Peredachi Informatsii*, 5(3).

Wainwright, M. J., Jordan, M. I., et al. (2008). Graphical models, exponential families, and variational inference. *Foundations and Trends® in Machine Learning*, 1(1–2).

Wenliang, L., Moskovitz, T., Kanagawa, H., and Sahani, M. (2020). Amortised learning by wake-sleep. In *International Conference on Machine Learning*.

Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3-4).

Williams, V. V. (2012). Multiplying matrices faster than coppersmith-winograd. In *Proceedings of the forty-fourth annual ACM symposium on Theory of computing*, pages 887–898.

Williams, V. V., Xu, Y., Xu, Z., and Zhou, R. (2024). New bounds for matrix multiplication: from alpha to omega. In *Proceedings of the 2024 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 3792–3835. SIAM.

Wilson, A. C., Roelofs, R., Stern, M., Srebro, N., and Recht, B. (2017). The marginal value of adaptive gradient methods in machine learning. In *Advances in Neural Information Processing Systems*.

Woodbury, M. A. (1950). *Inverting modified matrices*. Statistical Research Group.

Xiao, H., Rasul, K., and Vollgraf, R. (2017). Fashion-MNIST: A novel image dataset for benchmarking machine learning algorithms. *arXiv:1708.07747*.

Zeiler, M. D. (2012). ADADELTA: an adaptive learning rate method. *arXiv:1212.5701*.

Zhang, G., Sun, S., Duvenaud, D., and Grosse, R. (2018a). Noisy natural gradient as variational inference. In *International Conference on Machine Learning*.

Zhang, N., Ding, S., Zhang, J., and Xue, Y. (2018b). An overview on restricted Boltzmann Machines. *Neurocomputing*, 275.

Zhou, D.-X. (2020). Universality of deep convolutional neural networks. *Applied and Computational Harmonic Analysis*, 48(2).

Zuoqin, W. (2016). Riemannian Geometry Course.

Acronyms

AE Autoencoder. 10, 37

AG Accelerated Gradients. 52, 93, 94, 96, 97, 100–107, 124, 125, 160, 161

BGD Batch Gradient Descent. 27

BiHM Bidirectional Helmholtz Machine. 13, 14, 17, 32, 36, 59, 75, 76, 80–85, 94, 130, 159, 160, 182, 192

CNN Convolutional Neural Network. 139, 142, 143, 152, 154, 155, 158, 179, 205, 212

DAG Directed Acyclic Graph. 53, 55–57, 59, 142–146

DNRWS Diagonal Natural Reweighted Wake-Sleep. 71–73, 75, 76, 80, 160

EM Expectation-Maximization. 85

ER Entropy Regularizer. 14, 120–122, 124, 125, 161, 199, 200

FIM Fisher Information Matrix. 9, 12–16, 44–48, 53, 55–57, 59–64, 66, 68–73, 80, 82, 83, 87, 97, 98, 100, 106, 107, 110, 127, 130, 132–137, 139, 142, 143, 145–147, 150–152, 154, 156, 159–161, 179, 193, 195–197

GAN Generative Adversarial Network. 10, 39

GD Gradient Descent. 44–46, 88, 93, 94

GEM Generalized Expectation-Maximization. 86

GR Geometric Regularizer. 93

HFE Helmholtz Free Energy. 17, 21–26

HM Helmholtz Machine. 9–20, 22, 23, 25–27, 29–32, 36–39, 42, 46, 47, 53–57, 59–61, 63, 64, 70, 71, 76, 80, 94, 96, 97, 99, 100, 102, 106, 108–110, 113, 114, 116–120, 124, 125, 127–130, 133, 134, 137, 139, 143, 144, 146, 147, 152–156, 158–162, 189, 191, 214

IG Information Geometry. 10, 11, 14, 41, 43, 96, 116, 161, 162

KL Kullback-Leibler. 20, 21, 23, 25, 31, 46, 47, 84, 87, 117, 119, 158, 189

LR $\mathsf{L}^1/\mathsf{L}^2$ Regularizers. 101, 109–114, 116, 120, 121, 124, 161

MBGD Minibatch Gradient Descent. 27, 28, 66, 74–77, 82

MLE Maximum Likelihood Estimation. 86

NBiHM Natural Bidirectional Helmholtz Machine. 59, 80–85, 160, 192

Nesterov Nesterov Momentum. 14, 93–97, 100–104, 106, 107, 160

NG Natural Gradient. 9, 11–13, 15, 44, 45, 53, 56, 59–61, 63, 64, 68, 69, 71, 73, 75, 82, 93, 97, 100, 101, 106, 107, 114, 130, 139, 154, 159, 160, 162

NRWS Natural Reweighted Wake-Sleep. 13–15, 59, 61, 63, 64, 70–77, 79, 80, 82–84, 87, 89–91, 93, 96, 100–103, 105–107, 109–112, 114, 116, 120–122, 124, 125, 127, 128, 130, 136, 137, 139, 152, 154–156, 159–161, 189, 190, 195, 197–199, 201, 202

NWS Natural Wake-Sleep. 195

pdf Probability Density Function. 128

RBM Restricted Boltzmann Machine. 38, 39, 155

RS Rescaled-sleep. 59, 88–91, 160

RWS Reweighted Wake-Sleep. 13–15, 17, 30, 32, 35, 36, 60–64, 72, 74–77, 79, 80, 82, 87, 94, 100–107, 110–112, 114, 116, 120–122, 124, 125, 130, 152–155, 159, 160, 182, 199, 201, 202

SBN Sigmoid Belief Network. 10, 12, 13, 18, 19, 26, 38, 53, 55–57, 99, 116, 144, 147, 159, 160, 201

SGD Stochastic Gradient Descent. 27, 83, 122, 202

SW Sleep-well. 59, 86–89, 91, 160

TFD Toronto Face Dataset. 73, 76–80, 91, 102, 105–107, 110–112, 116, 120–122, 124, 125, 137, 152, 159, 161, 190, 191, 199

VAE Variational Autoencoder. 10–12, 23, 24, 32, 38, 39, 76, 155, 156

WS Wake-Sleep. 11, 14, 17, 24, 25, 29, 30, 32, 36–38, 61, 75, 76, 80, 84, 86–89, 91, 101, 119, 129, 159–162, 189, 195, 196, 201, 202

List of Symbols

The next list describes several symbols that are used within the body of the document. A general rule is, that we use calligraphic symbols for the exact values of the given mathematical quantities and simple capital letters for the empirical estimations of those same values. In particular, we used it for the Loss, Entropy and the Fisher Information Matrix. Furthermore, we use a `Monospace` font for constants and indices used for Convolutional Neural Networks.

Convolutional Neural Networks

`c`	Index commonly used for channels of input in CNN's
`C`	Input node channels
`H`	Height of input node
`W`	Width of input node
`f`	Index commonly used for channels of output in CNN's
`F`	Output node channels
`H'`	Height of output node
`W'`	Width of output node
`s`	Stride
`K`	Height of weights
`k`	Index commonly used for height of weights in CNN's
`L`	Width of weights
`l`	Index commonly used for width of weights in CNN's

Fisher Information Matrix

$\widetilde{F}$	Augmented empirical estimation of Fisher information matrix
$\mathcal{F}$	Fisher information matrix

F Empirical estimation of Fisher information matrix

U Matrix of samples used for the Empirical estimation of Fisher information matrix, when using the Woodbury formula

A The "inner-inverse matrix" when using the Woodbury formula $A = \alpha Q^{-1} + U^\top U$

$F_p^{i,j}$ Empirical estimation of the Fisher information matrix for the distribution p, of the j-th element of the i-th layer.

$\mathcal{F}_p^{i,j}$ Fisher information matrix for the distribution p, of the j-th element of the i-th layer.

Q Diagonal matrix of probabilities used for the Empirical estimation of Fisher information matrix, when using the Woodbury formula

Helmholtz Free Energy

$\tilde{\mathbf{F}}_G^R$ Augmented free energy of the generative network with respect to the recognition network, which is used only in the sleep step of the Wake-Sleep Algorithm

$\mathbf{E}_i$ Energy of some state i

$\mathbf{E}_G$ Energy of a state of the generative network

$\mathbf{H}$ Entropy of the system

$\mathbf{H}_G$ Entropy of the generative network

$\mathbf{H}_R$ Entropy of the recognition network

$\mathbf{F}$ Free energy

$\mathbf{F}_G$ Free energy of the generative network

$\mathbf{U}$ Internal energy of the system

$\mathbf{T}$ Temperature of the system

$\mathbf{F}_G^R$ Variational free energy of the generative network with respect to the recognition network, also called Evidence Lower Bound (ELBO)

Information Geometry

γ A curve

D	A divergence function
$\mathrm{D}_{KL}[\cdot\|\|\cdot]$	The Kullback-Leibler divergence function
$\mathcal{M}$	Manifold
$\mathcal{X}$	Set on which the manifold is defined
$\mathcal{S}$	Submanifold

K-step

α	Damping factor hyperparameter of NRWS
K	K-step hyperparameter of NRWS

Loss

$\mathcal{H}$	The entropy calculated for the HM
H	The estimation of the entropy
$\mathcal{L}$	Loss
L	Empirical estimation of Loss

Probability distributions

ρ	The activation value after applying the sigmoid function to the inputs of the layer.
$h^{i,j}$	j-th element of hidden layer i
$\mathcal{N}(\mu, \sigma)$	Normal distribution parameterized by μ and σ
μ	Mean (of Normal distribution)
θ	Generic parameters of the generation network p
W	Parameters of the generation network p used in linear function of the node
ϕ	Generic parameters of the recognition network q
V	Parameters of the generation network q used in linear function of the node
σ	Standard deviation

s The sigmoid function $s(x) = \frac{1}{(1+e^{-x})}$.

N Set of all nodes in the network, visible and hidden $\{x, h\}$

h Hidden nodes

p_θ Probability distribution p parameterized by θ, sometimes written also as $p(\ldots;\theta)$

$p_{\mathcal{D}}(x)$ The true distribution of data from some dataset $\mathcal{D}$

q_ϕ Probability distribution q parameterized by ϕ, sometimes written also as $q(\ldots;\phi)$

x Visible nodes

Regularization

L^p A p-norm type regularizer

β Parameter that controls the strength of regularization

Run-time constants

n Sample size times Mini-Batch size

B Mini-Batch size

$\mathcal{D}$ Dataset

M Depth / total number of layers of the network

D Dimension of the parameter space of a statistical model

η Learning rate

ω Complexity factor for matix inversion $2 \leq \omega \leq 3$

η_r Rescaled learning rate parameter (in the Rescaled-sleep method)

S Sample size

$\tilde{\omega}_k, \omega_k$ Weights for the RWS and BiHM

List of Figures

List of Tables

Appendix

Chapter A

Experimental Details

In this section, we present all the experimental details for the experiments in this thesis. The goal of the experiments is to train an HM on a dataset by learning to generate images that look indistinguishable from the ones that come from the dataset. To perform such experiments in a repeatable fashion, we define which datasets we use, what hyperparameters the training algorithms use and what kind of hardware are we using.

A.1 Datasets

In the following, we present the datasets that are being used for the experiments, to evaluate which competing training algorithm performs better, and also for tuning the hyperparameters of the algorithms.

- The **ThreeByThree** (denoted in the following as **3by3**) is a synthetic dataset, which was introduced by Kirby (2006). It consists of 12 vertical and horizontal patterns on a 3×3 grid, represented in Figure A.1, where vertical patterns are two times more likely to appear as horizontal ones. In our experiments, as a dataset, we used $10,000$ samples for each horizontal pattern and $20,000$ for each vertical one. The advantage of this dataset is that the KL divergence value between the *generation distribution* p and the *true distribution* of the data $p_{\mathcal{D}}$ can be calculated exactly, and we don't have to rely solely on the approximation by importance sampling, traditionally used to evaluate generative models. The algorithms training on 3by3 converge very quickly to a minimum, so the rate of convergence can be monitored in steps as well as epochs. 3by3 was mainly used for preliminary tests to explore the ranges of the hyperparameters quickly and serve as preliminary comparisons between the different variants of WS.

- For the final performance evaluation of NRWS we use the binarized version of the **MNIST** dataset of handwritten digits (Deng, 2012) as a standardized benchmark. The dataset consists of $60,000$ black and white images of handwritten digits of size 28×28, shown in Figure A.1.

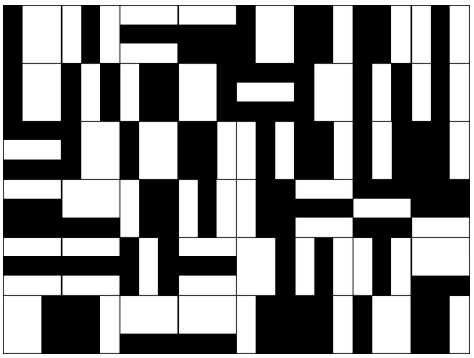
MNIST has been long used as a benchmark by the ML community, and as such it can be used to compare with different methods from the literature, however, nowadays it is considered a somewhat small dataset. We use MNIST for some of the hyperparameter tuning however we mostly use it to benchmark the performance of the algorithms. The network used for experiments with MNIST is usually composed o dense layers of size $300, 200, 100, 75, 50, 35, 30, 25, 20, 15, 10, 10$ (identical to the one used by Bornschein et al. (2016)), unless specified otherwise. In specific use cases, we also use the non-binarized, continuous version of the dataset.

- In addition to the datasets mentioned above, we used the **miniMNIST** dataset for hyperparameter tuning. The miniMNIST is a downsized binarized version of the MNIST dataset, from 28×28 to 14×14 (Figure A.1), similar to the one used by Hinton et al. (1995). Because of its reduced size compared to MNIST, but increased complexity from 3by3, it is well suited as a middle-ground between the two datasets. This makes it ideal for tuning the more complex hyperparameters such as the damping factor α, the effects of which could not be clearly seen on the 3by3. We usually used a model consisting of $100, 50, 20, 10, 10$ layers to test with this dataset.

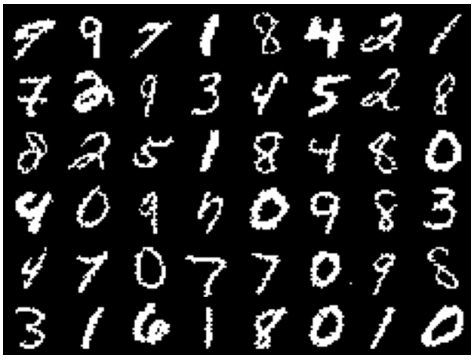
- In addition to MNIST, we show the efficacy of the NRWS on the **FashionMNIST** dataset (Xiao et al., 2017). It is also a very well-known dataset for benchmarking with the same size images as MNIST but with more difficulty, represented in Figure A.1. We used it for benchmarking and hyperparameter tuning. A similar network was used for FashionMNIST as for MNIST.

- A small version of **Toronto Face Dataset (TFD)** (Susskind et al., 2010) was also used for final benchmarking. The dataset contains images of black and white faces cropped to 48×48 size. Since NRWS has a memory requirement that increases with the size of the images (see Section 5.3), we needed to use the reduced 24×24 size version of the dataset, shown in Figure A.1. For benchmarking a similar network was used for TFD as for MNIST, only with larger bottleneck layers with $300, 200, 100, 75, 50, 35, 30, 25, 20, 20$ nodes. For other experiments, where benchmarking is not the aim, we specify what network we use.

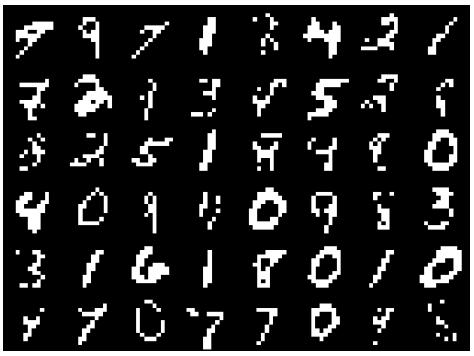
In order to test NRWS in Section 5.3.7, not only on binary datasets but continuous ones as well, we resorted to a form of data augmentation, where the gray values of the images were taken as probabilities for the bottom layer



(a) 3by3



(b) MNIST



(c) miniMNIST

(d) FashionMNIST

(e) TFD

Figure A.1: Random samples from each dataset used in the thesis: (a) 3by3; (b) MNIST; (c) miniMNIST; (d) FashionMNIST; (e) TFD.

of the HM. In Appendix D we go into more detail about how this works and present some results on the miniMNIST dataset.

A.2 Basic Settings for the Experiments

All experiments were run with CUDA-optimized Tensorflow 1.15 (Abadi et al., 2015). Benchmarks were run on Nvidia GTX1080 Ti GPUs; for other experiments, where clock-time was not as important a comparison, and hyperparameter tuning we used also an Nvidia RTX2080 Ti GPU or an Nvidia Titan X GPU. Whenever we compare algorithms based on time or epochs, these were run always on the same GPUs without other tasks running in parallel, to ensure a fair comparison.

In all of our experiments (unless explicitly specified otherwise) we used the following basic settings:

- mini-batch size $B = 32$
- sample size $S = 10$
- $K = 1000$
- $\alpha = 0.2$ unless specified otherwise
- all datasets are normalized to $-1, 1$ and all output functions of the layers (i.e. the output of sampling from Bernoulli distributions) are scaled to $[-1, 1]$ as well
- Glorot-normal initialization (Glorot and Bengio, 2010b) for the weights of the network, and a constant initialization of -1 for the biases, except in the case of the weights of the BiHM and NBiHM[1], where we used a truncated normal initialization, with mean 0 and standard deviation 0.001.

To favor comparisons between models and features, in our plots we report as a loss function the Negative Log-Likelihood (NLL) averaged over mini-batches and samples for all algorithms. We chose the NLL since it plays a fundamental role in the training of HMs, see Eq. (2.9), even if some of the algorithms have a different loss function, which we also present.

Our implementation is publicly available at:
https://github.com/szokejokepu/natural-rws .

[1]The reason for the different initialization method is because that was used in the original sourcecode associated to the paper (Bornschein et al., 2016), and we found that these algorithms also performed better with the different initialization.

Chapter B

Efficient Matrix Multiplication for the Inverse of the Fisher Information Matrix

When calculating the product $\widetilde{F}^{-1} \cdot \nabla L$, between the inverse of regularized FIM (see Eq.(5.9)) and the gradient of the loss, the order of the operations is important from a computational complexity perspective. When using the Woodbury formula our matrices that make up the FIM have the following sizes

$$\left(\tilde{F}_p^{i,j}\right)^{-1} \cdot \nabla L = \frac{1+\alpha}{\alpha} \left[I_{l_i} - \underbrace{U^{i+1}}_{l_{i+1} \times n} \underbrace{\left(A^{i,j}\right)^{-1}}_{n \times n} \underbrace{\left(U^{i+1}\right)^{\top}}_{n \times l_{i+1}} \right] \underbrace{\nabla L^{i+1}}_{l_{i+1} \times 1} . \tag{B.1}$$

Knowing that multiplying two matrices of size $a \times b$ and $b \times c$ have the computational complexity $\mathcal{O}(abc)$ and results in a matrix of size $a \times c$, we can find the optimal order in which to calculate the matrix products in (B.1), since matrix multiplication is an associative operation. The optimal order is to multiply the matrices from right to left, as this minimizes the computational complexity of the sequence of multiplications to $\mathcal{O}(l_{i+1}n + n^2)$. In the following formulation, we can observe how this is calculated

$$\underbrace{U^{i+1} \cdot \underbrace{\left(A^{i,j}\right)^{-1} \cdot \underbrace{\left(U^{i+1}\right)^{\top} \cdot \nabla L^{i+1}}_{n \times l_{i+1} \cdot l_{i+1} \times 1 \to n \times 1 \,:\, \mathcal{O}(l_{i+1}n)}}_{n \times n \cdot n \times 1 \to n \times 1 \,:\, \mathcal{O}(l_{i+1}n + n^2)}}_{l_{i+1} \times n \cdot n \times 1 \to l_{i+1} \times 1 \,:\, \mathcal{O}(l_{i+1}n + n^2)} .$$

All other orders of multiplying the same matrices would lead to worse results, than the optimal order. For example, multiplying from left to right would lead to a computational complexity of $\mathcal{O}(l_{i+1}n^2 + l_{i+1}^2 n)$.

Chapter C

Hyperparameter tuning

In this section, we will present more in detail some preliminary experiments to determine the right configuration of hyperparameters for the NRWS, depending on datasets, performance, and time to converge. The hyperparameters that we are interested in are the damping factor α, the learning rate η, and K.

C.1 Learning Rate and Damping Factor

Using a large damping factor leads to an optimization similar to the non-natural gradient algorithms while using damping factors that are too small leads to a large conditioning number in the estimated FIM. The inversion of which then carries serious numerical issues. This behavior can be seen clearly in Figure C.1. Ideally, we want to find the smallest damping factor that still maintains the optimization stable.

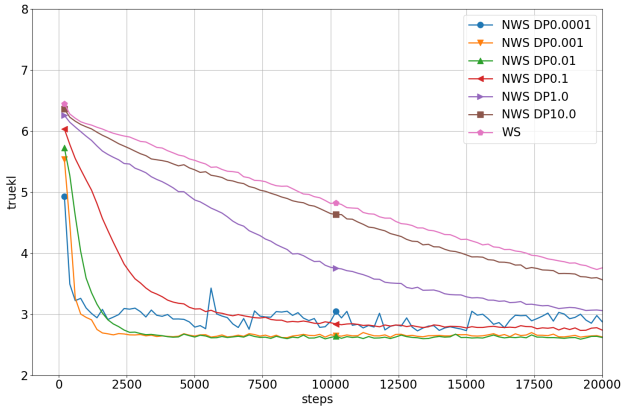


Figure C.1: The loss for WS and NWS with different damping values for the 3by3 dataset.

We searched empirically for the right combination of learning rate η and damping factor α leading to the best convergence rate. For the 3by3 dataset, we determined a range of $0.001 - 0.1$ for damping values that outperform the vanilla algorithm, which is suitable for different learning rates η of interest.

As one can observe, very low damping leads the algorithm to converge almost instantly but the loss remains noisy, as the inverse of the FIM is less regularized which is applied to the gradient. Large values however lead to convergence that is similar to the one of WS. From this general range, the parameter has to be fine-tuned for each dataset/model, to determine the appropriate combination of learning rate versus damping factor.

In Figure C.2 and Figure C.3 we compare different learning rates and damping factors on a smaller architecture to determine the appropriate values for optimal convergence in the case of the miniMNIST and MNIST. We found that the appropriate damping factor for this dataset for similar models is around $0.005 - 0.1$, with a learning rate in the neighborhood of $0.005 - 0.02$.

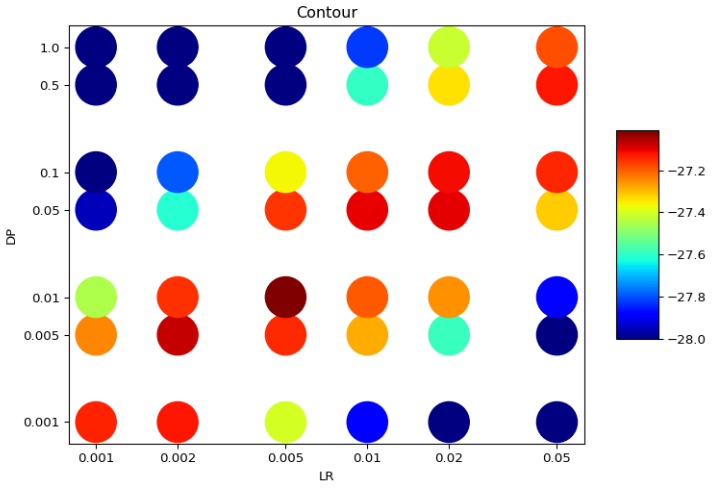


Figure C.2: The minimum loss of experiments comparing the learning rate η vs the damping factor α, on a model with 100, 50, 20, 10, 10 layers on miniMNIST.

We can notice a linear relation between the learning rate and the damping factor. When we grow the damping factor, we can also grow the learning rate up to a given point. The explanation for this phenomenon is that the smaller the damping, the closer the algorithm follows the geometry of the manifold defined by the statistical model (as presented in Subsection 5.3.2). Thus, smaller steps lead to better improvements, also a larger damping mitigates instabilities due to few-samples statistical estimations.

We also observe a correlation between the size of the image and the magnitude of the damping factor. The larger the image in the first layer, the more samples are needed to estimate the FIM of the network accurately.

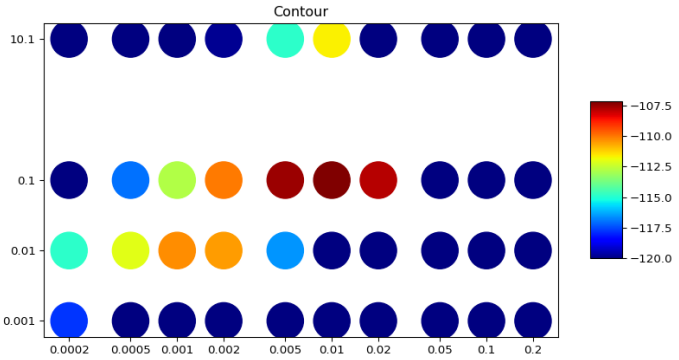


Figure C.3: The minimum loss of experiments comparing the learning rate η vs the damping factor α, on a model with 200,100,10 layers on MNIST.

When the number of samples cannot be grown anymore for practical reasons (the complexity of the algorithm grows quadratically with the number of samples, see Subsection 5.3.3), a larger damping factor is needed for the matrix to have a reasonable conditioning number.

C.2 K-step

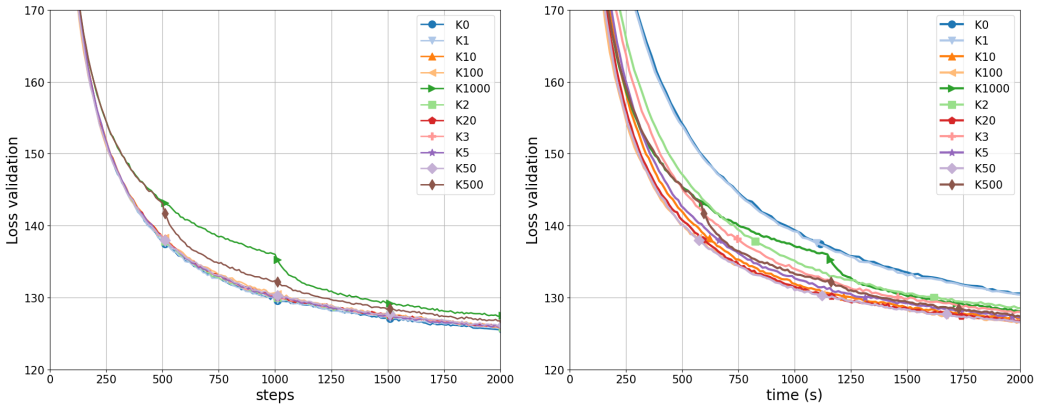


Figure C.4: Loss curves for MNIST for different values of K={0,1,2,3,5,10,20,50,100,500,1000}; left: in epochs; right: in seconds

Once we found the best combination of hyperparameters for the NRWS, we explored the possibility of computing the FIM only every K-step and thus speeding up the computational time of the algorithm. In Figure C.4 we see

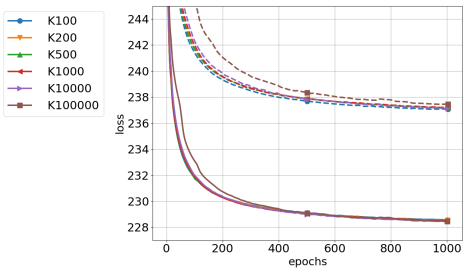


Figure C.5: Loss for train and test on FashionMNIST for different values of K

K	median. time (s)
100	286.17
200	234.74
500	207.63
1000	201.74
10000	191.1
100000	184.28

Table C.1: Average time per epoch for different values of K

the changes in the loss on the validation set for MNIST for a fixed learning rate and damping factor. We observe that when using the K-steps technique, the algorithm can speed up significantly but loses stability at very high values of K, which is most noticeable in the case of $K = 1,000$ steps, and to a lesser degree for $K = 500$. Consequently, we can use K-step in the range of $[1, 100]$ without a noticeable impact on the behavior of the algorithm. In these experiments, after $2,000$ steps all curves for $K < 500$ seem to converge to the same minimum, however, we consider this short time interval to not be enough to conclude on the optimal value for K.

To thoroughly explore the effects of the K-step, we analyzed the convergence of the NRWS on the FashionMNIST dataset, we fixed the learning rate $\eta = 0.001$ and damping factor $\alpha = 0.1$, while varying K. In Figure C.5 we can see that $K = 100,000$ is pushing the algorithm too far, as it breaks down on both the train and test curves. All-in-all we notice that most of the curves converge to the same minimum, contrary to what we might have expected from the previous experiment. Furthermore, we can notice a slight advantage in the test curve for $K = 100$, which hints that keeping the K value as low as possible might still bring some advantages. However, there is a significant difference in the time an epoch takes in Table C.1 to prefer larger values in order to make the algorithm as fast as possible. We concluded that K values in the range of $100 - 1000$ are the most preferable, which are acceptable from a speed and final convergence perspective. This result seems quite surprising and might hint at the fact that for some problems the metric on the manifold varies slowly from point to point.

C.3 Warm-up coefficient

When testing the Entropy Regularizer (ER), we noticed that at the beginning of the training procedure, the loss of the algorithm gets a huge bump in the loss compared to the non-regularized version, thanks to the regularization term. Such a bump eventually levels out. This is not a huge problem, as over time the loss of the regularized algorithm eventually outperforms the non-regularized version. However, this behavior could still be of concern, because it suggests that the regularization is too strong at the start of the training, where the parameters of the model are at a state where they do not need to be regularized yet. A common technique to circumvent this is to use a so-called warm-up coefficient, which scales up the regularization rate β from 0 until its specified value during a given number of epochs.

We can formulate the warm-up of β in the following way

$$\tilde{\beta}(t) = \begin{cases} \frac{1}{t_w - t}\beta & \text{if } t < t_w \\ \beta & \text{if } t \geq t_w \end{cases}$$

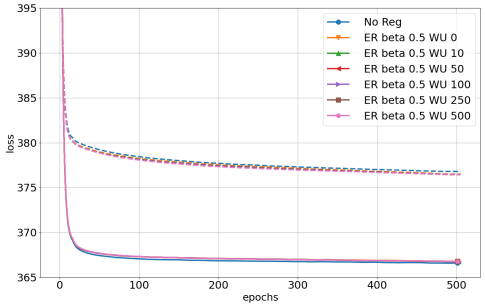
where $\tilde{\beta}(t)$ is the adjusted β scaled at time t, which represents the current time-step, and t_w the target warm-up time-step when β should reach its specified maximum value.

To test the effects of the warm-up, we compared different periods for t_w when training with ER to test the impact it has on the algorithm. In Figure C.6 we compare multiple values for the warm-up period t_w (defined in epochs), for relatively weak ($\beta = 0.5$) and strong ($\beta = 5$) ER for the RWS and NRWS on the TFD for 500 epochs.

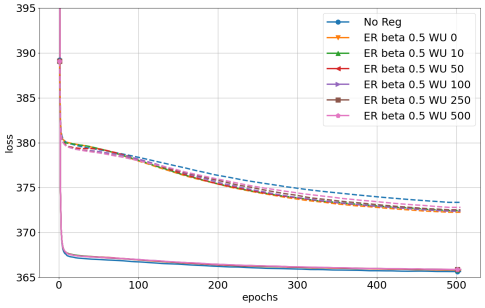
Our findings for low β (in C.6a and C.6b) show that the warm-up barely affects the training of the RWS regardless of the period. The warm-up affects similarly the NRWS but with a slightly better effect of the smaller periods t_w on the loss.

When looking at larger β (in C.6c and C.6d) all the effects of the warm-up get amplified. For the RWS we see that for the short term analyzed in these experiments, the warm-up does not help the ER overtake the non-regularized version, and all losses seem to end up at roughly the same value at the end of the 500 epochs. We will show in Section 6.3.2 that for larger training periods the warm-up does not affect negatively the training with ER.

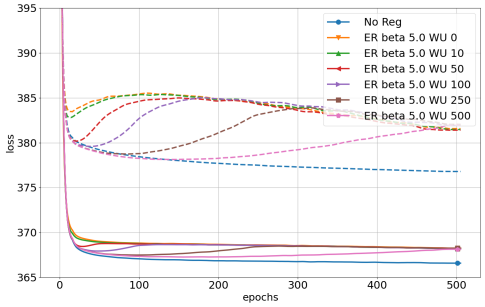
In the case of NRWS we see a similar delay of the ER, but no improvement overall on the value of the loss, as seemingly all the regularized versions of the algorithms seem to coalesce to a similar curve, which is better than the non-regularized version, usually before the first 100 epochs. However, for



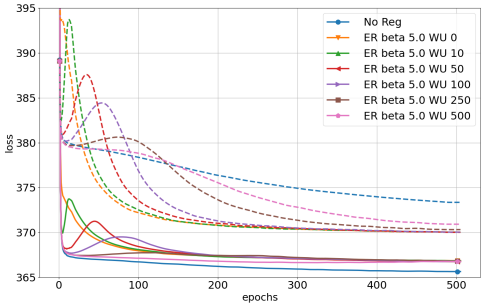
(a) Warm up coefficients for ER 0.5 on TFD for RWS



(b) Warm up coefficients for ER 0.5 on TFD for NRWS



(c) Warm up coefficients for ER 5.0 on TFD for RWS



(d) Warm up coefficients for ER 5.0 on TFD for NRWS

Figure C.6: Loss of RWS and NRWS with different warm up coefficients for ER. [**WU** - warm up time-step (in epochs); **ER beta** - Entropy Regularization β]

larger values of t_w (ex. $250 - 500$), we do not notice any benefit for the training, only the delay of the regularized value.

We conclude that using the warm-up for the ER does not have any immediate benefits on the training of the algorithms, other than the smoother behavior of the curves, however, it is also not detrimental to the training so we will continue using them when training with ER, in the range $t_w \leq 50$. For larger warm-up periods (> 100 epochs) the curves of the loss seem to be delayed too much, without any benefits..

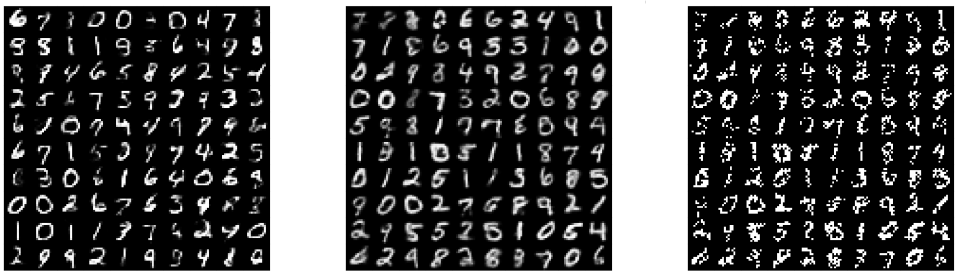
Chapter D

Data augmentation

In the thesis, we use two techniques to learn binary datasets. The first approach, which we call **B**, is to simply binarize all the samples from the dataset once, by rounding to $\{0, 1\}$. This is used for standard benchmarking usually and on average leads to smaller log-likelihoods.

The second technique, Binary Stochastic or Continuous **C**, consists of taking the gray values of the images as the means of Bernoulli distributions, for each sample from the dataset and each pixel in the image. **C** enables us to learn continuous data with an SBN and it is a technique that is widely used in the literature, for example in the paper by Bornschein et al. (2016). At each training step, we sample from the distributions, thus we get a range of samples, from a single image, which together approximates the original continuous sample better than **B**. In Figure D.1 *(a)* and *(b)* we see samples from models trained with the two differing techniques, while in *(c)* we see samples from the Bernoulli distributions with the respective means.

In Figure D.2 we compare the loss curves for three different models: WS, RWS and NRWS, comparing the two strategies for the miniMNIST dataset. In Table D.1 we report the importance-sampled estimation of the log-likelihood. We can see that in all cases the NRWS eventually outperforms both models in achieving the better minimum, as well as in the rate of convergence and wall-clock time. Moreover, a clear advantage is visible early on in the training,



(a) miniMNIST **B** probs (b) miniMNIST **C** probs (c) miniMNIST samples

Figure D.1: miniMNIST examples learned by our model.

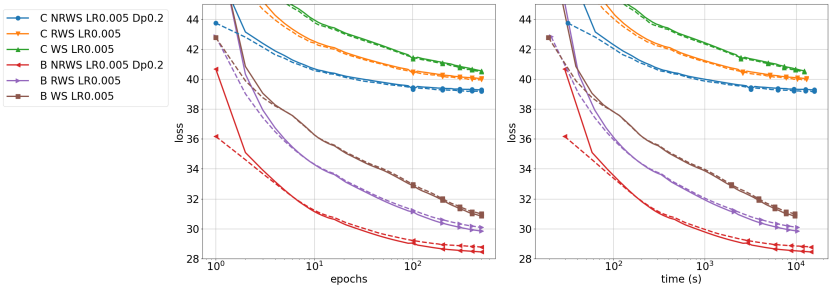


Figure D.2: The loss of training miniMNIST until convergence with **B** and **C**, with the algorithms WS, RWS and NRWS with the layers of size 100,50,20,10,10. On the left the convergence in epochs and on the right convergence in wall-clock time (s), both in log-scale.

DS	**ALG**	$\boldsymbol{\eta}$	**LL**	**T/E**
B	WS	0.04	-29.337	19s
	RWS	0.02	-28.695	21s
	NRWS	0.004	**-27.606**	31s
C	WS	0.02	-38.232	23s
	RWS	0.01	-37.811	25s
	NRWS	0.004	**-36.578**	32s

Table D.1: Importance Sampling estimation of the log-likelihood (**LL**) with 10,000 samples for different algorithms after 500 epochs of training with SGD. $\boldsymbol{\eta}$ is the learning rate, **T/E** is the average time per epoch. The damping factor used for NRWS is 0.05. For all algorithms, the number of samples used in training is 10.

in the case of **B** from the 12th epoch and in **C** from the very beginning.

Our preliminary analysis shows on miniMNIST a very small standard deviation for multiple runs of the same experiment, with different seeds. We tested the experiment with 24 different seeds and the best hyperparameters (LR 0.002 DP 0.05 as in Table D.1 and Figure D.2). After 100 epochs we obtain mean -28.3891 and std 0.0396 while after 200 epochs mean -28.2024 std 0.0299. This shows that the variance is relatively small for different seeds and gets smaller over time. Based on this observation we only present one run per experiment with the confidence that they behave as an average run.

Chapter E

Introduction to Convolutional Neural Networks

In this Appendix, we present Convolutional Neural Networks (CNNs) through a step-by-step buildup, to understand the underlying operations, constants, and indices. We start from the simplest possible case, and introduce complicating elements, until we arrive at the final form, that we use in Chapter 7.

Simple 2D Convolutional Layer

The simplest 2-dimensional convolutional case involves an input x, an output y, and a set of weights V and bias d. In this simplest of cases these variables x, y, and V are matrices. Let us define these quantities with sizes as:

- $x : \mathtt{H} \times \mathtt{W}$, where $\mathtt{H}$ is the height and $\mathtt{W}$ is the width of the input x;
- $y : \mathtt{H}' \times \mathtt{W}'$, where $\mathtt{H}'$ is the height, $\mathtt{W}'$ is the width of the output y;
- $V : \mathtt{K} \times \mathtt{L}$, where $\mathtt{K}$ is the height and $\mathtt{L}$ is the width of the weights V;
- d : scalar, which is the bias;

In the case when $\mathtt{H} = 4$, $\mathtt{W} = 4$, $\mathtt{K} = 2$ and $\mathtt{L} = 2$

$$x = \begin{bmatrix} x^{11} & x^{12} & x^{13} & x^{14} \\ x^{21} & x^{22} & x^{23} & x^{24} \\ x^{31} & x^{32} & x^{33} & x^{34} \\ x^{41} & x^{42} & x^{43} & x^{44} \end{bmatrix} .$$

$$V = \begin{bmatrix} V^{11} & V^{12} \\ V^{21} & V^{22} \end{bmatrix}$$

Convolution is an operation where we take the weights V and we apply it multiple times to the input x, as a sliding window, horizontally and vertically.

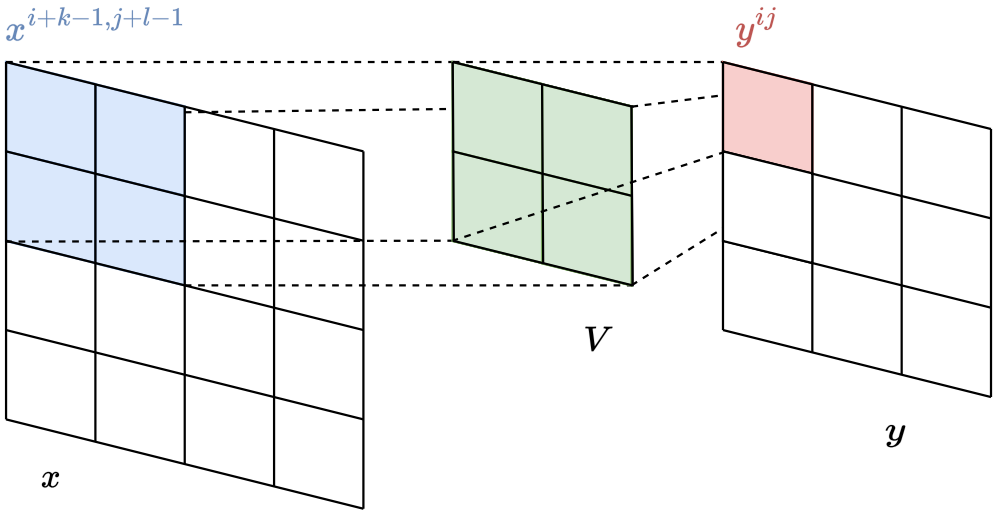


Figure E.1: 2D Convolutional layer, the blue area shows which area of x is involved with the weights V to define y^{ij}, in the case where $i = 1$ and $j = 1$.

Each application of the window results in a different output y^{ij}. With the x and V as above, we can define the convolutional operation for y:

$$\begin{aligned} y^{11} &= V^{11}x^{11} + V^{12}x^{12} + V^{21}x^{21} + V^{22}x^{22} + d \\ y^{12} &= V^{11}x^{12} + V^{12}x^{13} + V^{21}x^{22} + V^{22}x^{23} + d \\ &\dots \end{aligned}$$

by extension, the y will have shape $\mathtt{H}' = 3$, $\mathtt{W}' = 3$ as

$$y = \begin{bmatrix} y^{11} & y^{12} & y^{13} \\ y^{21} & y^{22} & y^{23} \\ y^{31} & y^{32} & y^{33} \end{bmatrix} .$$

We can define the operation of convolution for indices i and j of y generally as

$$y^{ij} = \left(\sum_{\mathtt{k}=1}^{\mathtt{K}} \sum_{\mathtt{l}=1}^{\mathtt{L}} V^{\mathtt{kl}} x^{i+\mathtt{k}-1, j+\mathtt{l}-1} \right) + d \quad \forall i, j \in \{1, 2, 3\} , \tag{E.1}$$

which we show in Figure E.1.

Suppose that the convolutional layer is part of a larger network, and we know the derivative of the loss $\mathcal{L}$ w.r.t. y, defined as:

$$\mathrm{d}y^{ij} = \left(\frac{\partial \mathcal{L}}{\partial y^{ij}} \right) ,$$

then we can compute the derivatives of V, d and x w.r.t. the loss $\mathcal{L}$. It is straightforward to see that the derivative w.r.t. d is

$$\frac{\partial \mathcal{L}}{\partial d} = \sum_{i=1}^{\mathtt{H}'} \sum_{j=1}^{\mathtt{W}'} \mathrm{d}y^{ij} \ . \tag{E.2}$$

The derivative w.r.t. an element m, n of V is

$$\frac{\partial \mathcal{L}}{\partial V^{mn}} = \sum_{i=1}^{\mathtt{H}'} \sum_{j=1}^{\mathtt{W}'} \mathrm{d}y^{ij} \sum_{\mathtt{k}=1}^{\mathtt{K}} \sum_{\mathtt{l}=1}^{\mathtt{L}} \frac{\partial V^{\mathtt{kl}}}{\partial V^{mn}} x^{i+\mathtt{k}-1, j+\mathtt{l}-1} \ , \tag{E.3}$$

knowing that

$$\frac{\partial V^{\mathtt{kl}}}{\partial V^{mn}} = \begin{cases} 1 & \text{if } m = \mathtt{k} \text{ and } n = \mathtt{l} \\ 0 & \text{otherwise} \end{cases}$$

and since $(n, m) = (\mathtt{k}, \mathtt{l})$ is only true for one pair of indices, we can reduce the Eq. (E.3) to

$$\frac{\partial \mathcal{L}}{\partial V^{\mathtt{kl}}} = \sum_{i=1}^{\mathtt{H}'} \sum_{j=1}^{\mathtt{W}'} \mathrm{d}y^{ij} x^{i+\mathtt{k}-1, j+\mathtt{l}-1} \ .$$

Note, that the derivative w.r.t. V can be done with a convolutional operation as well. In Figure E.2 we present which areas are involved in the derivative of $V^{\mathtt{kl}}$.

The gradients of the loss w.r.t. x are

$$\frac{\partial \mathcal{L}}{\partial x^{\mathtt{kl}}} = \sum_{i=1}^{\mathtt{H}'} \sum_{j=1}^{\mathtt{W}'} \mathrm{d}y^{ij} V^{\mathtt{k}-i+1, \mathtt{l}-j+1} \ .$$

2D Convolutional Layer with Stride

We can introduce the concept of stride to convolutional layers, by having a variable $\mathtt{s}$, which defines how much the window of V slides over the input x, where for $\mathtt{s} = 1$ we have the case defined in the previous section. Let us reiterate these quantities with sizes as:

- $x : \mathtt{H} \times \mathtt{W}$, where $\mathtt{H}$ is the height and $\mathtt{W}$ is the width of the input x;
- $y : \mathtt{H}' \times \mathtt{W}'$, where $\mathtt{H}'$ is the height, $\mathtt{W}'$ is the width of the output y;
- $V : \mathtt{K} \times \mathtt{L}$, where $\mathtt{K}$ is the height and $\mathtt{L}$ is the width of the weights V;
- d : scalar, which is the bias;

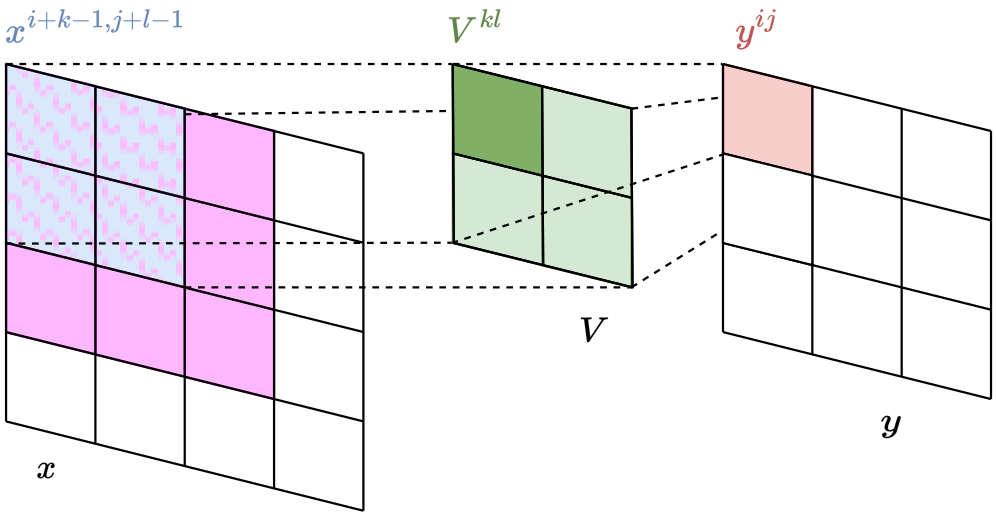


Figure E.2: 2D Convolutional layer, the blue area shows which area of x is involved with the weights V to define y^{ij}, in the case where $i = 1$ and $j = 1$ and the pink area shows which elements of x influence the derivative w.r.t. $V^{\mathtt{kl}}$ when $\mathtt{k} = 1$ and $\mathtt{l} = 1$.

- $\mathtt{s}$: scalar, is the stride.

We can formalize the output function of a convolutional layer using stride as

$$y^{ij} = \left(\sum_{\mathtt{k}=1}^{\mathtt{K}} \sum_{\mathtt{l}=1}^{\mathtt{L}} V^{\mathtt{kl}} x^{\mathtt{s}i+\mathtt{k}-1,\mathtt{s}j+\mathtt{l}-1} \right) + d \, .$$

The derivative of the loss with respect to d is unaffected by this change and it stays as defined in Eq. (E.2), while the derivatives with respect to V become

$$\frac{\partial \mathcal{L}}{\partial V^{\mathtt{kl}}} = \sum_{i=1}^{\mathtt{H}'} \sum_{j=1}^{\mathtt{W}'} \mathrm{d}y^{ij} x^{\mathtt{s}i+\mathtt{k}-1,\mathtt{s}j+\mathtt{l}-1} \, ,$$

and derivatives w.r.t. x is

$$\frac{\partial \mathcal{L}}{\partial x^{\mathtt{kl}}} = \sum_{i=1}^{\mathtt{H}'} \sum_{j=1}^{\mathtt{W}'} \mathrm{d}y^{ij} V^{\mathtt{k}-\mathtt{s}i+1,\mathtt{l}-\mathtt{s}j+1} \, .$$

The consequence of using a stride $\mathtt{s} > 1$ to shift the sliding window is that the application of V to x can be non-overlapping, which can be advantageous for two reasons. First, it means that elements of the input x contribute equally to each element of the output y^{ij}, i.e., no element of x is involved in the computation of two distinct elements of y^{ij}. Secondly, the size of the

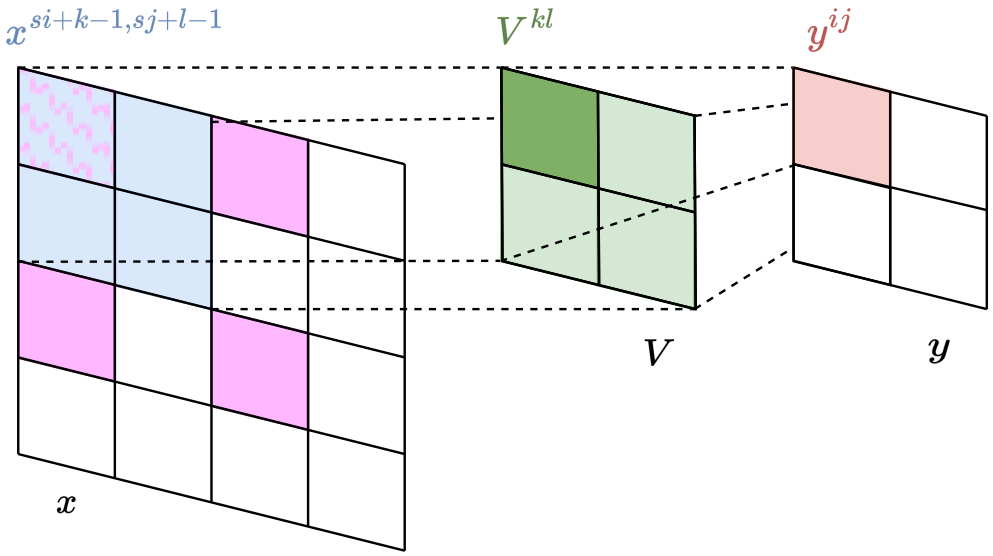


Figure E.3: 2D Convolutional layer with stride, where the blue area shows which area of x is involved with the weights V to define y^{ij}, in the case $i = 1$ and $j = 1$ and the pink area shows which elements of x influence the derivative $V^{\mathtt{kl}}$ when $\mathtt{k} = 1$ and $\mathtt{l} = 1$, for a stride of $\mathtt{s} = 2$.

output y is shrunk, which can be understood as compressing the information from x through the mapping of the convolutional layer. The convolutional operation with stride can be seen in Figure E.3.

2D Convolution with Multiple Output Layers and Stride

To increase the capability of a convolutional layer, we can add multiple layers to the weights V, which results in multiple layers for the output y. We usually refer to the depth of the y as output channels. The quantities of a convolutional layer with sizes then are:

- $x : \mathtt{H} \times \mathtt{W}$, where $\mathtt{H}$ is the height and $\mathtt{W}$ is the width of the input x;
- $y : \mathtt{H}' \times \mathtt{W}' \times \mathtt{F}$, where $\mathtt{H}'$ is the height, $\mathtt{W}'$ is the width and $\mathtt{F}$ are the depth of the output y;
- $V : \mathtt{K} \times \mathtt{L} \times \mathtt{F}$, where $\mathtt{K}$ is the height, $\mathtt{L}$ is the width and $\mathtt{F}$ is the depth (output channels) of the weights V;

- $d : \mathtt{F}$, which is the bias;
- $\mathtt{s}$: scalar, is the stride.

Notice, that V and y are no longer matrices, but 3-dimensional tensors, where each layer $\mathtt{f} \in \mathtt{F}$ of V can be understood as learning a different "feature" (as commonly referred to in the literature) or property of the input x. We can define the output function of a convolutional layer with multiple outputs and stride as

$$y^{ij\mathtt{f}} = \left(\sum_{\mathtt{k}=1}^{\mathtt{K}} \sum_{\mathtt{l}=1}^{\mathtt{L}} V^{\mathtt{klf}} \; x^{\mathtt{s}i+\mathtt{k}-1,\mathtt{s}j+\mathtt{l}-1} \right) + d_{\mathtt{f}}$$

Now the derivative of the loss is

$$\mathrm{d}y^{ij\mathtt{f}} = \left(\frac{\partial \mathcal{L}}{\partial y^{ij\mathtt{f}}} \right) \ ,$$

which we treat as known. The derivatives of d and V become dependent on the output channels $\mathtt{f}$

$$\frac{\partial \mathcal{L}}{\partial d^{\mathtt{f}}} = \sum_{i=1}^{\mathtt{H}'} \sum_{j=1}^{\mathtt{W}'} \mathrm{d}y^{ij\mathtt{f}} \ ,$$

$$\frac{\partial \mathcal{L}}{\partial V^{\mathtt{klf}}} = \sum_{i=1}^{\mathtt{H}'} \sum_{j=1}^{\mathtt{W}'} \mathrm{d}y^{ij\mathtt{f}} x^{\mathtt{s}i+\mathtt{k}-1,\mathtt{s}j+\mathtt{l}-1} \ ;$$

while the gradients for x are summed over the extra dimension $\mathtt{F}$

$$\frac{\partial \mathcal{L}}{\partial x^{\mathtt{kl}}} = \sum_{i=1}^{\mathtt{H}'} \sum_{j=1}^{\mathtt{W}'} \sum_{\mathtt{f}=1}^{\mathtt{F}} \mathrm{d}y^{ij\mathtt{f}} V^{\mathtt{k}-\mathtt{s}i+1,\mathtt{l}-\mathtt{s}j+1,\mathtt{f}} \ .$$

In Figure E.4 we present how a single layer $\mathtt{f}$ of the weights $V^{\cdot\cdot\mathtt{f}}$ interacts with the input x and output y layers[1].

2D case Multi-Channel Multi-Output with stride

We can further increase the capability of a convolutional layer, by processing multiple layers $\mathtt{C}$ of the input x simultaneously, by expanding the weights V by a 4-th dimension. However, the sliding window of the convolutional operation is kept on the first two dimensions of V which we refer to as the

[1] We use the notation $\cdot$ for a variable, to represent a span over all indices in the given position. Therefore, $V^{\cdot\cdot\mathtt{f}}$ represents the $\mathtt{f}$-th $2D$ matrix slice of the 3 dimensional tensor that is V in this case.

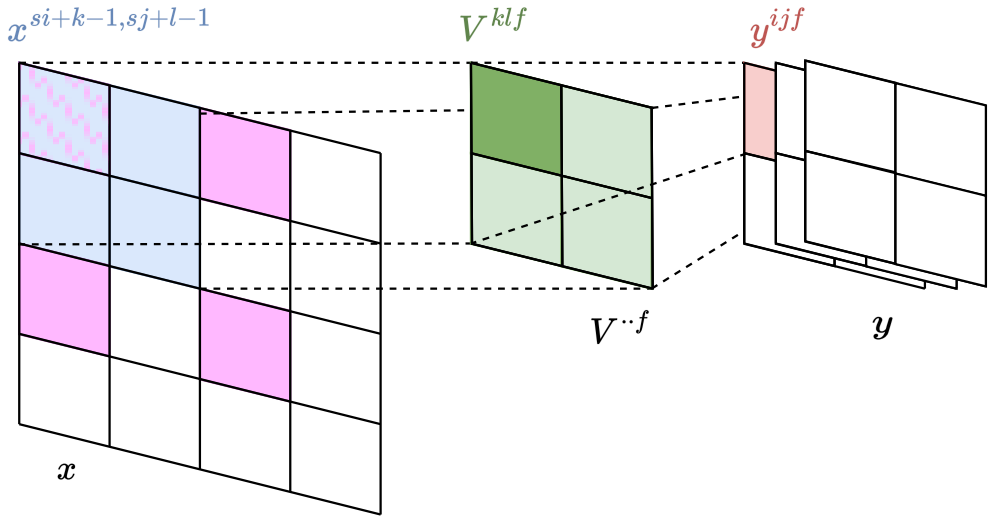


Figure E.4: 2D Convolutional layer with one input channel and multiple output channels with stride. The blue area shows which area of x is involved with the $\mathtt{f}$-th layer of the weights, $V^{\cdot\cdot\mathtt{f}}$, to define $y^{ij\mathtt{f}}$, in the case where $i = 1$, $j = 1$ and $\mathtt{f} = 1$. The pink area shows which elements of x influence the derivative $V^{\mathtt{klf}}$ when $\mathtt{k} = 1$ and $\mathtt{l} = 1$, for a stride of $\mathtt{s} = 2$.

width and height, while spanning through the whole depth $\mathtt{C}$ of the input x. We usually refer to the depth of the x as input channels, and we can define the elements with sizes as:

- $x : \mathtt{H} \times \mathtt{W} \times \mathtt{C}$, where $\mathtt{H}$ is the height, $\mathtt{W}$ is the width and $\mathtt{C}$ is the depth of the input x;
- $y : \mathtt{H}' \times \mathtt{W}' \times \mathtt{F}$, where $\mathtt{H}'$ is the height, $\mathtt{W}'$ is the width and $\mathtt{F}$ are the depth of the output y;
- $V : \mathtt{K} \times \mathtt{L} \times \mathtt{C} \times \mathtt{F}$, where $\mathtt{K}$ is the height and $\mathtt{L}$ is the width of the weights V, while $\mathtt{C}$ and $\mathtt{F}$ correspond to the input and output channels respectively;
- $d : \mathtt{F}$, which is the bias;
- $\mathtt{s}$: scalar, is the stride.

Since the weights V is now a 4-dimensional tensor, we can think of the convolutional operation as sliding $\mathtt{F}$ distinct cuboids windows horizontally and vertically over the input x. We can define the output function of a

convolutional layer with multiple input and output layers as

$$y^{ij\mathtt{f}} = \left(\sum_{\mathtt{k}=1}^{\mathtt{K}} \sum_{\mathtt{l}=1}^{\mathtt{L}} \sum_{\mathtt{c}=1}^{\mathtt{C}} V^{\mathtt{klcf}} \, x^{\mathtt{s}i+\mathtt{k}-1,\mathtt{s}j+\mathtt{l}-1,\mathtt{c}} \right) + d^{\mathtt{f}} \tag{E.4}$$

Treating $\mathrm{d}y$ as known, the gradients then can be defined as

$$\frac{\partial \mathcal{L}}{\partial d^{\mathtt{f}}} = \sum_{i=1}^{\mathtt{H}'} \sum_{j=1}^{\mathtt{W}'} \mathrm{d}y^{ij\mathtt{f}} \ , \tag{E.5}$$

$$\frac{\partial \mathcal{L}}{\partial V^{\mathtt{klcf}}} = \sum_{i=1}^{\mathtt{H}'} \sum_{j=1}^{\mathtt{W}'} \mathrm{d}y^{ij\mathtt{f}} x^{\mathtt{s}i+\mathtt{k}-1,\mathtt{s}j+\mathtt{l}-1,\mathtt{c}} \ , \tag{E.6}$$

$$\frac{\partial \mathcal{L}}{\partial x^{\mathtt{klc}}} = \sum_{i=1}^{\mathtt{H}'} \sum_{j=1}^{\mathtt{W}'} \sum_{\mathtt{f}=1}^{\mathtt{F}} \mathrm{d}y^{ij\mathtt{f}} V^{\mathtt{k}-\mathtt{s}i+1,\mathtt{l}-\mathtt{s}j+1,\mathtt{c},\mathtt{f}} \tag{E.7}$$

Having multiple channels for both the input x and the output y results in the possibility of stacking convolutional layers, as this is commonly done in CNNs. Stacking multiple convolutional layers, each progressively adding more channels and condensing the width and height of the layers is employed usually to "compress" the size of the input, and split it into different features, so that more specific layers can deal with a more condensed representation of the input data of the network.

An example of multiple input/multiple output convolutional layer is presented in Figure E.5.

Deconvolutional Layer

We can define the inverse operation of a convolutional layer which is the deconvolutional layer, also known as the transposed convolution in the literature. In this case, the input is y, the elements of which we multiply by the weights W which results in the output x. In the deconvolutional layer, the weights W still slide over x with the stride $\mathtt{s}$, the same as in the convolutional case. Let us define these quantities with sizes:

- $x : \mathtt{H} \times \mathtt{W} \times \mathtt{C}$, where $\mathtt{H}$ is the height, $\mathtt{W}$ is the width and $\mathtt{C}$ is the depth of the output x;
- $y : \mathtt{H}' \times \mathtt{W}' \times \mathtt{F}$, where $\mathtt{H}'$ is the height, $\mathtt{W}'$ is the width and $\mathtt{F}$ are the depth of the input y;

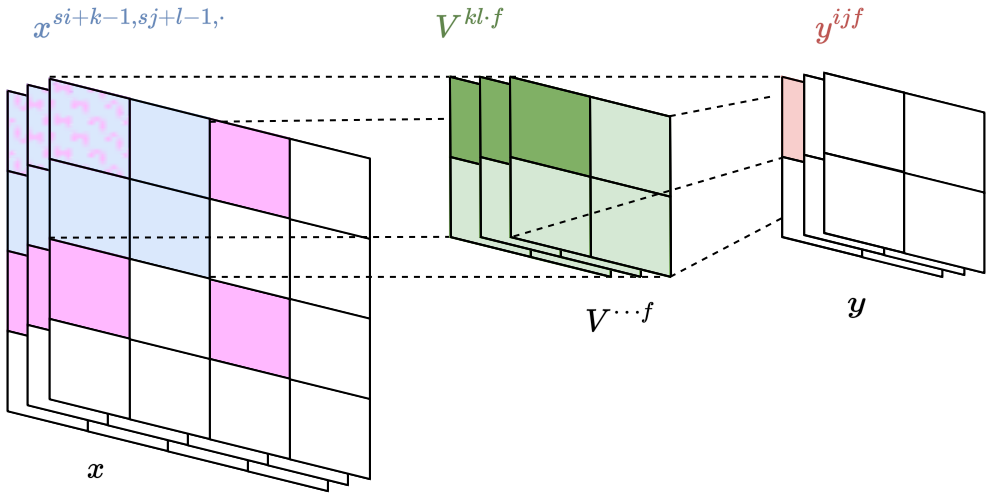


Figure E.5: 2D Convolutional layer with multiple input channels and multiple output channels with stride. The blue area shows which area of x is involved with the weights of the `f` layer, $V^{\cdots \mathtt{f}}$, to define $y^{ij\mathtt{f}}$, in the case where $i = 1$, $j = 1$ and $\mathtt{f} = 1$. The pink area shows which elements of x influence the derivative $V^{\mathtt{klcf}}$ when $\mathtt{k} = 1$ and $\mathtt{l} = 1$, for a stride of $\mathtt{s} = 2$.

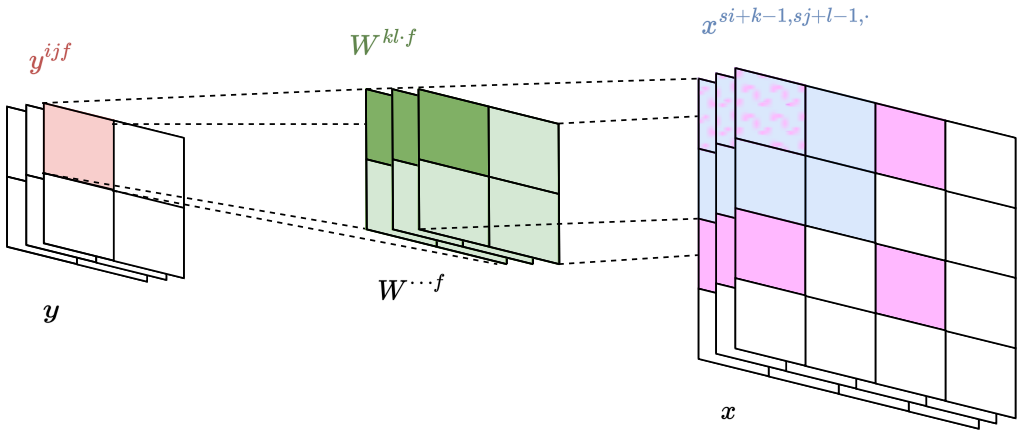


Figure E.6: 2D Deconvolutional layer with multiple input channels and multiple output channels with stride. The blue area shows which area of x is the result of the deconvolution of $y^{ij\mathtt{f}}$ with the weights of the `f` layer, $W^{\cdots \mathtt{f}}$, in the case where $i = 1$, $j = 1$ and $\mathtt{f} = \mathtt{F}$. The pink area shows which elements of x influence the derivative $W^{\mathtt{klcf}}$ when $\mathtt{k} = 1$ and $\mathtt{l} = 1$, for a stride of $\mathtt{s} = 2$.

- $W : \mathtt{K} \times \mathtt{L} \times \mathtt{C} \times \mathtt{F}$, where $\mathtt{K}$ is the height and $\mathtt{L}$ is the width of the weights W;
- $b : \mathtt{C}$, which is the bias;
- $\mathtt{s}$: scalar, is the stride.

The deconvolutional operation can be defined as follows

$$\begin{aligned} x^{ij\mathtt{c}} &= \sum_{\mathtt{f}=1}^{\mathtt{F}} \sum_{\mathtt{k}=1}^{\mathtt{K}} \sum_{\mathtt{l}=1}^{\mathtt{L}} W^{\mathtt{cklf}}\, y^{(i-\mathtt{k})/\mathtt{s}+1,(j-\mathtt{l})/\mathtt{s}+1,\mathtt{f}} + b^{\mathtt{c}} \quad \text{where} \\ y^{(i-\mathtt{k})/\mathtt{s}+1,(j-\mathtt{l})/\mathtt{s}+1,\mathtt{f}} &= 0 \quad \text{if} \quad \frac{(i-\mathtt{k})}{\mathtt{s}} + 1, \frac{(j-\mathtt{l})}{\mathtt{s}} + 1 \in \mathbb{R} \backslash \mathbb{N} \,. \end{aligned} \tag{E.8}$$

If we know the derivative of the loss with respect to x

$$\mathrm{d}x^{ij\mathtt{c}} = \left(\frac{\partial \mathcal{L}}{\partial x^{ij\mathtt{c}}} \right) \,,$$

we can define the gradients

$$\frac{\partial \mathcal{L}}{\partial b^{\mathtt{c}}} = \sum_{i=1}^{\mathtt{H}} \sum_{j=1}^{\mathtt{W}} \mathrm{d}x^{ij\mathtt{c}} \,,$$

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial W^{\mathtt{klcf}}} &= \sum_{i=1}^{\mathtt{H}} \sum_{j=1}^{\mathtt{W}} \mathrm{d}x^{ij\mathtt{c}} y^{(i-\mathtt{k})/\mathtt{s}+1,(j-\mathtt{l})/\mathtt{s}+1,\mathtt{f}} \,, \\ \frac{\partial \mathcal{L}}{\partial y^{ij\mathtt{f}}} &= \sum_{\mathtt{k}=1}^{\mathtt{K}} \sum_{\mathtt{l}=1}^{\mathtt{L}} \sum_{\mathtt{c}=1}^{\mathtt{C}} W^{\mathtt{klcf}}\, \mathrm{d}x^{\mathtt{s}i+\mathtt{k}-1,\mathtt{s}j+\mathtt{l}-1,\mathtt{c}} \end{aligned} \tag{E.9}$$

Notice that the derivative of w.r.t. y is a convolutional operation, as well as that the derivative of x in the convolutional step in Eq. (E.7) is a deconvolutional operation.

The deconvolutional layer can be used opposite to the convolutional layer to serve as a decoder pair, to the encoding role of the latter. Indeed in Chapter 7 we use both types of layers to be able to process larger input data for HMs. The deconvolutional operation is illustrated in Figure E.6.