Multi-level Data Representation For Training Deep Helmholtz Machines

José Miguel Penedo Ramos

Thesis to obtain the Master of Science Degree in

Information Systems and Computer Engineering

Supervisor: Prof. Andreas Miroslaus Wichert

Examination Committee

October 2022

Acknowledgments

I would like to start by expressing my gratitude to Professor Andreas Wichert and Luís Sá Couto for introducing me to the world of research and teaching me how to overcome its quintessential non-linearity.

As this thesis represents the conclusion of a long chapter, I want to thank all the people who were with me in my University journey, especially Carlota, Maria, Maria, Marta, Lara, João, and Gonçalo, whose names feel mandatory in this script.

I am also grateful to Rodrigo, Maria, and Luís for their help, but most importantly for their simple presence in the hermitic times involved in the making of a thesis.

Transversal to work and essential to my life, I would like to thank my Mother, Father, my grandparents, Diniz, Guilherme, Miguel, and Zé.

Last but not least, to my best friend João – Thank you.

Abstract

A vast majority of the current research in the field of Machine Learning is done using algorithms with strong arguments pointing to their biological implausibility such as Backpropagation, deviating the field's focus from understanding its original organic inspiration to a compulsive search for optimal performance. Yet, there have been a few proposed models that respect most of the biological constraints present in the human brain and are valid candidates for mimicking some of its properties and mechanisms. In this thesis, we will focus on guiding the learning of a biologically plausible generative model called the Helmholtz Machine in complex search spaces using a heuristic based on the Human Image Perception mechanism. We hypothesize that this model's learning algorithm is not fit for Deep Networks due to its Hebbian-like local update rule, rendering it incapable of taking full advantage of the compositional properties that multi-layer networks provide. We propose to overcome this problem, by providing the network's hidden layers with visual queues at different resolutions using a Multi-level Data representation. The results on several image datasets showed the model was able to not only obtain better overall quality but also a wider diversity in the generated images, corroborating our intuition that using our proposed heuristic allows the model to take more advantage of the network's depth growth. More importantly, they show the unexplored possibilities underlying brain-inspired models and techniques.

Keywords

Helmholtz Machine; Biologically-inspired Models; Deep Learning; Generative Models; Hebbian Learning; Wake-Sleep.

Resumo

A grande maioria dos estudos no âmbito de Machine Learning são feitos usando algoritmos com fortes argumentos que apontam para a sua implausibilidade biológica tais como Backpropagation, desviando assim o foco desta área de entender a sua original inspiração orgânica para uma procura compulsiva por melhor performance. No entanto, existem alguns modelos propostos que respeitam a maioria das restrições existentes no cérebro humano, e são validos candidatos para simulações de algumas das suas propriedades e mecanismos. Nesta tese, iremo-nos focar em guiar a aprendizagem de um modelo generativo biologicamente plausível chamado Helmholtz Machine em espaços de procura complexos, usando uma heurística baseada no mecanismo de percepção de imagens humano. A nossa hipótese é que o algoritmo de aprendizagem deste modelo não é adequado para redes profundas devido à sua regra de aprendizagem local semelhante à regra de Hebb, tornando-o incapaz de tirar partido das propriedades composicionais que redes com várias camadas oferecem. Propomos uma solução para resolver este problema, oferecendo informações visuais a diferentes resoluções às camadas internas da rede do modelo, usando representações dos dados a vários níveis de resolução. Os resultados em diferentes datasets de imagens mostram que o modelo foi capaz de não só obter melhor qualidade, mas também uma maior diversidade nas imagens geradas, o que corrobora a nossa intuição de que ao usar a heurística proposta, o modelo é capaz de tirar maior partido da profundidade da rede. Mais importante, mostram as possibilidades inexploradas por detrás de modelos e técnicas inspiradas no cérebro humano.

Palavras Chave

Helmholtz Machine; Modelos inspirados na Biologia; Aprendizagem Profunda; Modelos Generativos; Hebbian Learning; Wake-Sleep.

Contents

List of Figures

List of Tables

List of Algorithms

Acronyms

BM	Boltzmann Machine
DBN	Deep Belief Network
HM	Helmholtz Machine
HN	Hopfield Network
IID	Independent and Identically Distributed
KL	Kullback-Leibler
KNN	K-Nearest Neighbor
LR	Logistic Regression
RBM	Restricted Boltzmann Machine
WS	Wake-Sleep

1 Introduction

Contents

Most recent machine learning models have shown great effectiveness at solving a wide range of complex cognitive tasks [2, 3], and back-propagation algorithms seem to be at the core of the majority of those models, proving it to be one of the most reliable and fast ways for machines to learn [4–6]. Visual pattern recognition is one of the many fields in which back-propagation algorithms thrive [3, 7, 8]. The evolution of these models' quality has been impressively swift, but as we get closer to perfection, the possible improvements get evermore difficult [9]. For some of the more simple visual tasks like image classification of handwritten digits in the famous MNIST dataset [10], these models have surpassed the brain capabilities, performing better than human participants [9, 11].

The surpassing of the human brain's accuracy is an amazing scientific mark and allows for more reliable and robust technology.

In the midst of this search for better and more powerful models, grew a firmer and firmer connection between the two concepts of intelligence and accuracy. We seem to have been intuitively led to the conclusion that the better a model performs at a certain task, the more intelligent it is. In a sense, we deviate from trying to mimic the brain's biological way of thinking and focus instead on neural network models that perform better [5, 12].

Nonetheless, even if there are models that compete with the human brain at performing specific tasks, there is no model that comes close to the robustness and flexibility of the human brain when dealing with general image classification and pattern recognition problems.

Therefore, a large part of the scientific community is still focused on the biologically plausible side of machine learning, proposing new competitive models that remain an arguably plausible implementation of some human brain mechanisms and properties [12–16].

1.1 Back-propagation's Biological Plausibility

Despite the obvious biological inspiration of the Back-propagation algorithm [5, 17], its biological plausibility has been questioned very early on from its appearance [18, 19]. In recent years, although there have been many attempts to create biologically plausible and empirically powerful learning algorithms similar to back-propagation [4, 12, 20], there is an overall consensus that some fundamental properties of back-propagation are too difficult for the human brain to implement [5, 13].

The first and most relevant argument is related to the fact that back-propagation synaptic weight updates depend on computations and activation on an entire chain of neurons whereas biological synapses change their connection strength solely based on local signals. Furthermore, for the Gradient-based algorithm to work, biological neurons' updates would have to be frozen in time waiting for the signal to reach its final destination where the error comparison is made, and only after the signal travels backwards the membrane permeability would be changed in accordance to its success or failure [14].

The second is the fact that back-propagation uses the same weights when performing forward and backwards passes, which would require identical bidirectional connections in biological neurons that are not present in all parts of the brain.

And lastly, the fact back-propagation networks propagate firing probabilities, whereas biological neurons only propagate neuron spikes [2].

1.2 Helmholtz Machines' Biological Inspiration

We propose to look at an older Generative model called Helmholtz Machine (HM), which uses the Wake-Sleep (WS) algorithm [21] instead of Back-propagation.

The Wake-Sleep is an unsupervised learning algorithm that uses two different networks to simultaneously learn a predictive Recognition Model and a generative Generation Model. Despite not being a completely Hebbian algorithm, its activation and learning rules are as local as the Hebb rule [22].

Hebbian learning algorithms respect the original proposition made by Hebb [23], that learning and memory in the brain would arise from increased synaptic efficacy, triggered by the coordinated firing of the pre- and post-synaptic neurons [24], and more importantly, they solve the previously mentioned locality problem because the synaptic weight updates only depend on the previous layer. Thus, the locality of WS also helps to avoid that problem in a similar way to the Hebbian rule.

The unsupervised nature of the algorithm, also contributes to its plausibility, since the human brain's learning is mostly done with unsupervised data. And unlike in Back-propagation where it is very difficult to find an implementation that works by propagation neuron activations instead of firing probabilities, the WS algorithm can work effectively with both options, solving the third mentioned back-propagation implausibility argument.

Furthermore, the learning algorithm of these machines is based on the biological idea of being awake and asleep. Its intuition is that after we experience an event, we also produce our own variations of those events. This idea can be easily extrapolated to what happens on a big scale daily, where we experience reality during our wake phase, and then recreate it in our sleep, but there is a shorter scale example that perhaps compares better to the actual behavior of the model that occurs, for example, in the interaction between the human eyes and the brain. Our brain receives continuous streams of images that our eyes are capturing, and while we are receiving them, we subconsciously try to predict what will happen in the next frame, and when the reality does not match your expectation, for example, when a magician pulls a rabbit out of the hat, we become surprised. The HM network also mimics this behavior, and after receiving an observation from the world, it will produce a dream, then the network will adjust its weights in order to create more plausible dreams, and try to reduce the surprise when experiencing the next event. Likewise, if you see the same magic trick performed enough times, you will learn to expect what was previously unexpected.

This ”reduce of surprise” corresponds to minimizing a quantity very imminent in neuro-scientific research called Free Energy [25, 26], which is ”an information theory measure that bounds the surprise on sampling some data, given a generative model” [27]. Thus, the minimization of Free Energy corroborates the hypothesis that ”a biological agent resists the tendency toward disorder through a minimization of uncertainty” [24, 27, 28] alluded to in the previous example.

2 Related Work

Contents

In this chapter, we want to explain the successive models that continuously introduced key concepts essential to the conceptualization of Helmholtz Machines, starting from simple Associative Memory Networks, and eventually growing into more complex models such as Boltzmann Machines.

2.1 Associative Memory Networks

Associative Memory Networks try to embed in themselves a set of known patterns, and given slightly different ones, maybe even never seen before, will match them to the closest known pattern. These networks hope to create a similar content addressable memory to the one that our brain seems to have, which is highly tolerant to perturbations and corruptions [29]. Although this form of intelligence may not seem as straightforward as a network that given a certain pattern, for example, a blood test analysis of a patient, can determine with a close to 100% accuracy if the patient has a certain disease, this network addresses something more conceptual, which is related to the idea of memory.

We can paraphrase this problem in a more intuitive yet very similar way, which is Plato's theory of Ideas and Forms [30]. Although this theory is very complex, we will address it in a much simpler way, that does not scratch the surface of the philosophical complexity that it bears. We believe this paradigmatic example will be helpful for the readers to get a more down-to-earth example of the way these networks intend to operate. In Plato's view, we can separate all existing things into two worlds, the world of Ideas, and the world of Forms. The world of Ideas is the one we have inside our brain, that is related to the concepts of the things we see in our physical world. Plato refers to the latter as the world of Forms. For example, if you close your eyes and try to see what image your brain associates with the word "horse", you will get what Plato considered the "Idea" of a horse, which belongs to the world of Ideas. Of course, the horse in your head is not real, and is probably different, even if by the smallest detail, from every horse that ever existed in the "world of forms", which in this case could be our real world. Nevertheless, whenever you encounter a horse in the real world, even if the size or the color is different from the generic one in your brain, it quickly associates it with your idea of a horse. Which is precisely what the Associative Memory Networks try to achieve. The tolerance to perturbations and corruptions can also be visually interpreted with our example, if you imagine, for instance, a picture of a horse with 6 legs or one that only shows half the horses' body, even though these things are not actual horses, we can still make this connection to the concept of horse in our head.

2.2 Hopfield Networks

The Hopfield Network (HN) model was proposed by Hopfield in 1982 [31] and is a neural network model that is based on the Associative Memory Network but designed in a way that resembles a lot of physical and magnetic models. In this model, each pair of neurons i and j are connected with a weight w_{ij} [32,33]. The activation values of the network units have values of +1 (active) or -1 (inactive) denoted by S_i, and can be defined by:

$$S_i := sgn\left(\sum_j w_{ij} S_j - \Theta_i\right) \tag{2.1}$$

where the Θ_i corresponds to the activation threshold, which in this model, can be incorporated into the weights as a bias, leaving us with a simpler function:

$$S_i := sgn\left(\sum_j w_{ij} S_j\right) . \tag{2.2}$$

This equation defines the dynamic updates of the neurons on the network. There are two imminent ways to approach the order of the neurons that will be updated. The synchronous update, where all units are updated at each time step, and asynchronous, where a randomly chosen unit is updated each time. Both approaches are valid and are better fitted for distinct tasks [34]. However, the "asynchronous choice is more natural for both brains and artificial networks" [1], since we don't have an internal clock coordinating our neurons' simultaneous synapses.

2.2.1 Storing a pattern

For simplicity of exposition, we will explain how to assign the weights to the network, so that a single pattern is recognizable and later try to generalize this example for multiple stored patterns. First, we need to assure that a stored pattern is stable, meaning once the network evolves to that pattern it remains unchanged. In other words, we must guarantee that:

$$sgn\left(\sum_j w_{ij} \xi_j\right) = \xi_j \tag{2.3}$$

if the above is true, the rule (2.2) will not cause any more updates to the network, meaning that the network has converged. If we consider $w_{ij} \propto \xi_i \xi_j$ and take the proportionality constant to be $1/N$ we get:

$$w_{ij} = \frac{1}{N} \xi_i \xi_j \tag{2.4}$$

and therefore, starting from a pattern S_i we get:

$$h_i = \sum_j w_{ij} S_j . \tag{2.5}$$

Where $sgn(h_i)$ will be the value of S_i. With the connections defined by the previous rule, we can see that even if not all, but the majority of the bits from a certain pattern S_i are equal to the ξ_i, the “wrong” bits will be overwhelmed by the majority of the similar bits since $sgn(h_i)$ will be ξ_i. So in the end, we can prove that the pattern not only is stable but can be reached from another similar pattern.

When the network evolves from a random state to another stable one, we call the latter an attractor [35], thus, by storing the pattern ξ_i, it will become an attractor. For this “one pattern” example, there will be two attractors, one is ξ_i, which is reached when the majority of the input bits are similar to the stored pattern, and we can formulate similar reasoning for when the majority of the bits differ from ξ_i making the pattern $-\xi_i$ the second attractor.

2.2.2 Storing many patterns

Extrapolating the first example to multiple stored patterns leads to more complex scenarios, where we want many patterns to be attractors [36]. A simple way to achieve this, is to make the weight rule, to be a sum of the values for all stored patterns

$$w_{ij} = \frac{1}{N}\sum_{\mu=1}^{p} \xi_i^{\mu}\xi_j^{\mu} \tag{2.6}$$

where p is the total number of stored patterns.

This is usually denominated as the “Hebb rule” because it follows the hypotheses suggested by Hebb [23,37] that the changes of the weights should be proportional to the correlation between the firing of the pre- and post-synaptic neurons.

Now, once again, we must prove that a certain pattern ξ_i^{ν} is stable. And just like previously, we want to make sure

$$sgn(h_i^{\nu}) = \xi_i^{\nu} \tag{2.7}$$

where the input h_i^{ν} is given by

$$h_i^{\nu} \equiv \sum_j w_{ij}\xi_j^{\nu} = \frac{1}{N}\sum_j \sum_{\mu=1}^{p} \xi_i^{\mu}\xi_j^{\mu}\xi_j^{\nu} \tag{2.8}$$

we now separate the term where $\mu = \nu$ from the inner sum and obtain

$$h_i^{\nu} = \xi_i^{\nu} + \frac{1}{N}\sum_j \sum_{\mu\neq\nu} \xi_i^{\mu}\xi_j^{\mu}\xi_j^{\nu} \; . \tag{2.9}$$

In order for the initial condition to be verified, we only need to make sure that the module of the last term of (2.9), that is denominated as the crosstalk term is smaller than 1, so that the value of the sign function for ξ_i^{ν} plus that term remains the same as ξ_i^{ν}. It turns out that if we choose a number of stored patterns that is small enough, around $0.138 \times \frac{number\ of\ patterns}{number\ of\ units}$ [38], the crosstalk term will indeed be inferior to one almost every time. So the stability requirement is met [39].

The capacity of a HN is not trivially obtainable, and small variations of the learning rule lead to different storage capacities [40, 41], so we will not go into much detail about the calculations behind the definition of the "small enough" threshold. However, we want to leave the reader with a general idea of how that term is reached.

The stability of the model depends on the absolute value of the crosstalk term being smaller than one. If we examine this crosstalk term, we can see that it is a sum of many neuron values ξ_i. We know that most of the time, some of these values will be +1 and some will be -1, so in the end, they would somewhat nullify each other, and the crosstalk term would be close to 0. However, there is a small probability that, for example, most of the terms are positive, and that all the ξ_i factors would sum up to a value higher than 1. That probability always exists, and assuming all ξ_i are independent, it can be described by the binomial distribution, and can then (if the number of units is high enough) be approximated by a Gaussian distribution. The more units there are (bigger pattern length), the less likely it is for them to "align" on the same value, and the higher the number of patterns, the higher the value the crosstalk term is, so we can easily understand that the probability of the crosstalk term is related to these two variables. The value for the threshold will depend on the probability we allow for an error to occur when storing a pattern, that is given by the Gaussian distribution. For example, the general value $0.138 \times \frac{number\ of\ patterns}{number\ of\ units}$ mentioned before, is related to an error probability of 0.0036$\%$.

2.3 Energy Function

One of the biggest changes Hopfield brought to Neural Network theory, was the introduction of an energy function. This function can be seen as a "hilly" surface that stores a certain pattern at each local minima [42], where the dynamical system will navigate, eventually reaching one of these patterns. Therefore, the idea of training a network can be compared to the idea of strategically shaping the surface of this landscape, so that a certain pattern corresponds to local minima. This energy function can be more formally described by the following function:

$$H = -\frac{1}{2}\sum_{ij} w_{ij} S_i S_j \tag{2.10}$$

moreover, it is proven when applied to networks where the connection strengths w_{ij} and w_{ji} are the same, it does create a local minima for each starting pattern given to the network [1].

The purpose of this function is to have a dynamical rule which makes the total energy of the system always decrease or remain the same, eventually converging to our desired attractor state. A good visual interpretation would be a sphere left on top of the energy landscape whose movement would be defined by the force of gravity, eventually falling into a dip, meaning the system reached a stable state, and therefore (hopefully) the stored pattern that more closely resembles the initial state.

The most common name for this function is the **Lyapunov function** [43], but there are other terminologies that we might be more familiarized with, such as the **Hamiltonian**, **cost function** or **fitness function**. One thing that is important to note, is that for the energy function to exists, the connections between weights must be symmetric $w_{ij} = w_{ji}$, which despite being biologically implausible for our neurons [1] is not an obstacle for the model since the Hebbian rule (2.6) assures this equality.

2.4 Spurious States

The previous section might have given us the idea that we successfully created a perfect Associative Memory Network that can perfectly recall all the patterns we want. However, this is not completely true, since it is also proven that there are other local minima apart from the desired ones. These undesired minima are called spurious states, and they emerge due to three different reasons [1, 44]. First, the reversed states, since the pattern $-\xi^\mu$ are also local minima, as we showed in section (2.2.1). Second, the mixture states, that consist of a combination of multiple stored patterns. And finally, some more complex minima that do not fit in the first, nor the second reason, and are usually called spin glass states [45].

2.5 Ising Model

There is a clear similarity between these types of networks and some physical systems since they were one of the inspirations for its conception. Perhaps one of the most paradigmatic examples that confirm this claim is a magnetic system called the Ising model [46]. The simplest representation of this model consists of a finite number of points (atomic magnets) across a two-dimensional plane, where each point can have a value of +1 if the magnet has a “spin up” or -1 if it has a “spin down”. The parallelism between this model and HN is trivial, we can think of these values for each magnet as the S_i values of each neuron. In the Ising model, the value of each spin is influenced by the spin values of their neighbours, called internal field, and the external field h^{ext} that is present at it’s location [47]. Yielding the following rule:

$$h_i = \sum_j w_{ij} S_i + h^{ext} \tag{2.11}$$

where w_{ij} corresponds to the strength of the influence of spin S_j on the field of S_i. One particularity of the properties of the magnets is that the influential strengths are symmetric, therefore $w_{ij} = w_{ji}$, which as mentioned before, is a fundamental requirement for the existence of an energy function.

2.5.1 Thermal fluctuation in the Ising models

Explaining this model serves two different purposes, the first one, stated in the previous section, is to show this intrinsic resemblance that HN have with physical models, and the second, is to introduce the idea for an extension of the previous deterministic networks, called Stochastic Networks. One can think of Thermal fluctuations on Ising models, as a force that works against the alignment of the spins with the magnetic field, working against the actual field fluctuations. However, this Thermal fluctuation is only noticeable for high temperatures, being nullified at the absolute zero temperature. The mathematical way to describe it's influence on the Ising model is using the Gauber dynamics Ising model [48], that replaces the rule for S_i described in the previous section with the following stochastic rule:

$$S_i := \begin{cases} +1 & \text{with probability } g\left(h_i\right) \\ -1 & \text{with probability } 1-g\left(h_i\right) \end{cases} \tag{2.12}$$

where $g(h)$ is a sigmoid function related to the temperature.

$$g(h) = \sigma_\beta(h) = \frac{1}{1+e^{-2\beta h}} \tag{2.13}$$

with β defined as:

$$\beta = \frac{1}{k_B T} \,. \tag{2.14}$$

By analyzing the effect of the β factor on the equation (2.13), we can see that as it tends to infinity (decreasing temperature), the sigmoid function starts to polarise more abruptly the results eventually becoming equal to a deterministic network with the sign function, whereas when it tends to 0 (increasing temperature) it starts to resemble a flat line, which means the value of S_i would be completely random.

2.6 Stochastic Networks

Stochastic Networks use a similar concept to the idea of Thermal Fluctuations on the Ising model to determine the probability of a neuron to update. Yielding

$$Prob(S_i = \pm 1) = \sigma_\beta(\pm h_i) = \frac{1}{1-e^{\pm 2\beta h_i}} \,. \tag{2.15}$$

Once again, as the parameter β grows to infinity, this sigmoid function starts to resemble the sign function used for the deterministic Hopfield Network.

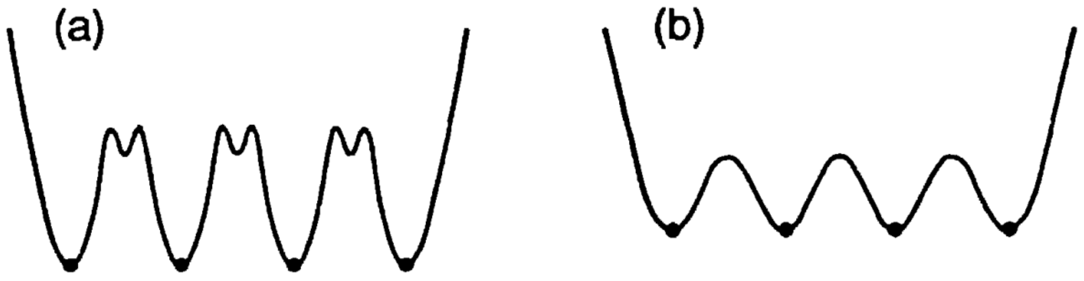

Figure 2.1: (a) Illustration of an energy landscape with spurious states as small dips between the desired pattern valleys. (b) Less detailed interpretation of the exact energy landscape due to higher stochasticity. Taken from [1]

Biological neurons fire with variable strengths and there is also a delay between synapses, therefore creating what we will broadly classify as noise. This noise can also be compared to the temperature factor of the Ising model and is essentially a force, that works against the stability of the model. By bringing this idea of noise to the network, ideally, we also bring the model one step closer to the way the human brain works.

Having a factor that disturbs the convergence of the model may seem counterintuitive at first, but is very useful when trying to avoid the local spurious minima [49, 50]. We will try to give an intuition for this occurrence by considering a simple energy function (see Fig. 2.1), where (a) would be the actual energy landscape, and only the global minima (represented with a dot) were the stored patterns, whereas the other minima would correspond to spurious states. A non-stochastic model, when trying to decrease the overall energy of the system, might be tempted to slowly fall into one of the spurious states, if the model started in a state that belonged inside one of the small dips between the desired patterns. The introduction of stochasticity on the model will essentially allow it to traverse this landscape in a less predictable way, and eventually, unconventionally move away from the center of small dips, and allow it to find the desired patterns. The higher the temperature (meaning a high level of stochasticity) the more likely it is for these unconventional paths to be taken, therefore, at a high enough temperature, it is actually likely the model completely avoids local minima, since it would introduce some noise in the energy function (a) making it look similar to the energy landscape shown on function (b).

One can think of increasing the temperature as losing some details of the exact energy function and change it into a “smoother” line. Of course, raising the temperature to infinity is not a valid solution to avoid local minima, since it would turn the energy landscape closer and closer to a flat line, preventing the model from converging at all and losing all the information it stored. So the actual solution would be to start at a high enough temperature, to avoid local minima at first, and slowly reduce this temperature along with its evolution, in order to eventually settle at a specific state.

One linguistic pitfall we might be tempted to fall into is this idea of temperature as the actual temperature of our brain, which is not at all related to what we are considering as the β factor. The temperature we consider is just the overall noise that exists in the model at a certain time-step, similarly to the stability of the Ising models with a certain temperature.

2.7 Boltzmann Machines

The Boltzmann Machine (BM) derives from stochastic networks and also incorporate the idea of hidden units as an extension of the visible units present on the HN [49]. Consequentially, another common name for these machines is Stochastic Hopfield Networks with hidden units.

The visible units have similar behavior as the neurons in the HN model but now, besides being connected between themselves they are also connected to some hidden units. The hidden ones could also be connected between themselves and to the visible units, but have no connection to the environment [51]. The hidden units, allow this model to learn more complex correlations between the visible units and ultimately provide higher expressive power.

Both the hidden and the visible units have a stochastic behavior, and the probability for a neuron to fire is determined by equation (2.15). The connections between the units are symmetric, allowing the existence of an energy function (2.10) where each state can be assigned to a certain energy level. “The energy of a [state] can [then] be interpreted as the extent to which that combination of hypotheses violates the constraints implicit in the problem domain” [49]. So by minimizing this energy, we are finding states that better satisfy the constraints of our domain.

2.7.1 Restricted Boltzmann Machines

As the name suggests, the Restricted Boltzmann Machine (RBM) [52] is a simpler version of the original Boltzmann Machines. It consists of a two-layer stochastic network, where the first layer $\mathbf{v}$ has all the visible units, and the second one $\mathbf{h}$ has all the hidden ones. In addition, there must be no connections within the same layer, so a visible unit can only be connected to hidden units, and vice-versa [53]. The goal of this machine is to learn the joint probability $p(\mathbf{v}, \mathbf{h})$ for all configurations of $\mathbf{v}$ and $\mathbf{h}$ [54].

This model is not particularly complex, and despite being significantly faster to train than a normal Boltzmann Machine [55], this machine alone is not able to describe as complex domains as its predecessor.

2.7.2 Deep Belief Networks

A Deep Belief Network (DBN) uses stacked RBMs to create a deep network that has a powerful generative model [56]. The way these networks are built is by simply learning and fixing the weights for a single RBM. Then, the representations obtained in the hidden layers will serve as an input for the next RBM. So the hidden units of the first machine correspond to the visible ones of the second one. Once the second RBM is trained, we can repeat this process, and continuously stack RBMs on top of each other, creating a deep network that was trained layer-by-layer [57]. This approach has seen a lot of success in capturing higher-level representations of input features [51].

3

Helmholtz Machine

Contents

To fully understand Helmholtz Machines, we need to use the concepts of the energy function, stochastic neural networks, and other probabilistic backgrounds. These networks are in many ways different from Boltzmann Machines, however, the foundations for both designs are very similar, hence the reason for explaining the Boltzmann Machines as a stepping stone to understanding the HM. Although we could start explaining the Helmholtz Machine from the Boltzmann checkpoint, we believe that, since these machines are the core of our work, it is better to explain them from scratch, referring to previously explained concepts along the way. We will be following the trail of thought described by Kevin G. Kirby [58] since it describes not only the model itself but the mathematical background that led to its definition in a clear and intuitive way.

3.1 The Motivation

Instead of heading straight to the detailed definition of the HMs, we believe it is better to start with a less technical explanation of the way this machine works. Helmholtz Machines are neural network models that try to understand the world by creating a generative model that explains it. The world for a Helmholtz Machine consists of many different bit inputs we call patterns. We can consider, for example, a 4 by 4 grid with either black or white squares, where the input would be a bit vector of size 16 $\{0,1\}^{16}$, where the first four bits would refer to the top row, and the subsequent sequences of 4 bits would refer to the other rows beneath the first one. Each bit would then have a value of 0 if the square was black, and 1 if it was white. Of course, if we assume that every combination of black and white squares was possible and had an equal probability of appearing, it would be impossible for the model to understand a way to make any inference about a certain pattern it observed from the world since the world itself would be completely random. In this example, if we wanted the machine to produce an image that it thinks is likely to exist in this world, it would be as accurate as a machine that chose random values for each bit. This may lead us to believe this model is useless, however, there is a quintessential presumption this model has about the world, which is the fact that there are certain rules that are not explicit, but the world follows them [58,59]. If we assume the world only had rows either all black or all white, a HM would in theory understand this particularity of the world, and when it tried to reproduce an image of the world, ideally, it would only return grids with black or white rows, and never, for example, a chess-like pattern. This idea of a world with only rows makes it plausible for the HM to work, but this strict restriction might lead us to think that it could never be applied in a real-world scenario, where there are countless possibilities and patterns, and not only a finite number of them. Perhaps a better example that meets the simplicity of the first example halfway with the complexity of the real world, is the example of a chessboard. The chessboard example is very commonly recalled when explaining machine learning algorithms since it has enough configurations for a human to think of them as almost infinite and perhaps make the connection for its

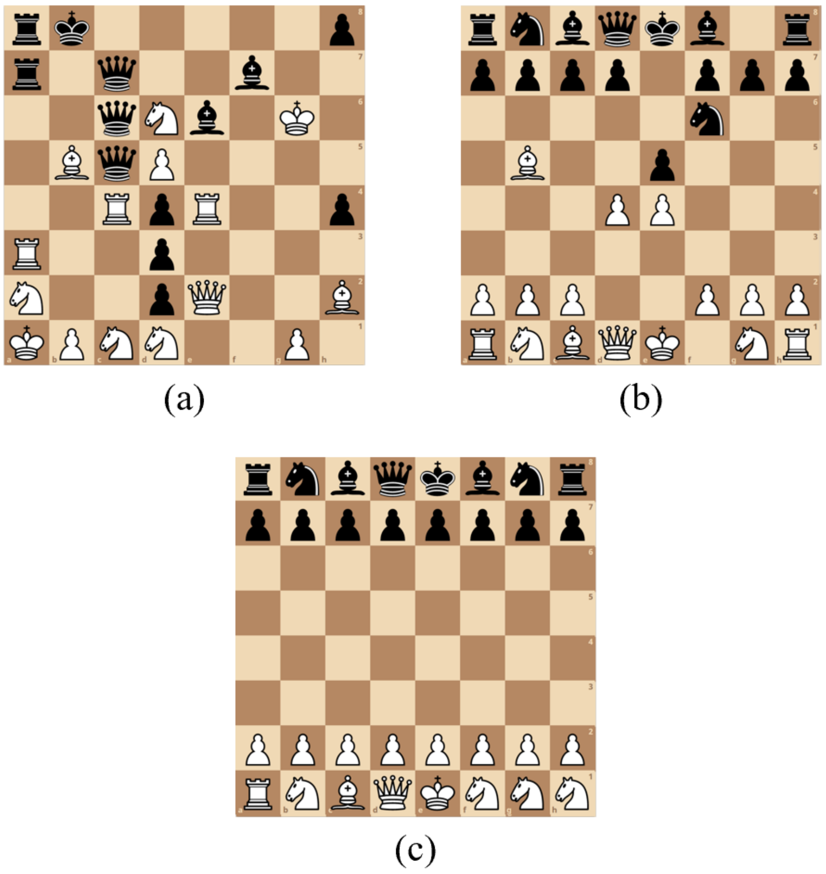

Figure 3.1: Different chessboard configurations. Configuration (a) is visually chaotic and impossible to be reached in a normal chess game, configuration (b) is a plausible and relatively common to be reached in a chess game, configuration (c) is very similar to the starting board configuration only with two pieces different at the bottom right corner, therefore, a human is not so shocked when looking at it at first glance when compared to configuration (a) despite both being impossible to reach.

applicability from simple to practical life scenarios more smoothly. The board configuration, in a way, is very similar to the first grid example, in a sense that there is a slightly bigger 8×8 grid, but now we have more options than black or white since it can either be empty or have a certain piece on it. In this example, we can think that the HM will look at a board and try to understand how reasonable it is to assume that specific board configuration could appear in a chess game, or in other words, how surprised it is about that certain pattern. We can look at the figure below, and try to understand, how reasonable it would for that position to be reached in a chess game. It can be completely unreasonable (Fig. 3.1a), completely reasonable (Fig. 3.1b), or something in between (Fig. 3.1c).

As we can see, even though they all belong to the enormous amount of board configurations, by understanding the patterns and rules of piece positions and movements, we can infer something about its plausibility to appear in our world.

This example hopefully shows the type of patterns that our brain has subconsciously learned from looking at chessboards throughout our lives, which makes it capable of attributing a level of plausibility to a certain board, even though you might have never seen that exact configuration. These types of patterns are exactly what this model tries to understand when it creates the generative model of the world.

3.2 Probabilistic Notions

Just like other well known Machine Learning Models, HMs receive and generate bit vectors of variable size N. This vectors are commonly represented in the literature with bold lowercase $\mathbf{d} \in \{0,1\}^N$ and in this Thesis we will follow the same paradigm.

These vectors withhold the information received and generated by the machine, while the machine itself holds what we call a probability functions, which assign a certain probability to a certain occurrence of that pattern and are described as $p(\mathbf{d}) : \{0,1\}^N \rightarrow [0,1]$. Another way to look at this function is as a distribution of a discrete random variable $\mathbf{D}$, resulting in the probability function:

$$p(\mathbf{d}) = Prob[\mathbf{D} = \mathbf{d}] \ . \tag{3.1}$$

This probability function can be easily extended to a multiple vector input such as $\mathbf{xy} \in \{0,1\}^L \times \{0,1\}^M$. Where the function $p(\mathbf{xy}) = Prob[\mathbf{X} = \mathbf{x} \wedge \mathbf{Y} = \mathbf{y}]$ describes the joint probability distribution of both bit vectors.

This allows us to use the basic probability notions of marginalization and conditional probability and the Bayesian Theorem to arrive at the following definitions:

$$p(\mathbf{xy}) = p(\mathbf{x}|\mathbf{y})p(\mathbf{y}) \ , \tag{3.2}$$

$$p(\mathbf{x}) = \sum_{\mathbf{y}} p(\mathbf{x}|\mathbf{y})p(\mathbf{y}) \ , \tag{3.3}$$

$$p(\mathbf{xy}|\mathbf{d}) = p(\mathbf{x}|\mathbf{yd})p(\mathbf{y}|\mathbf{d}) \ , \tag{3.4}$$

$$p(\mathbf{y}|\mathbf{x}) = \frac{p(\mathbf{y})}{p(\mathbf{x})} p(\mathbf{x}|\mathbf{y}) \ . \tag{3.5}$$

This can be particularly useful, if we consider the initial input $\mathbf{d}$ as N different vectors that consist of a single bit. Our initial function (3.1) becomes:

$$p(d_1 d_2 ... d_N) = Prob[D_1 = d_1 \ \wedge \ D_2 = d_2 \ \wedge \ ... \ \wedge \ D_N = d_N] \ . \tag{3.6}$$

Under the assumption that each bit is independent we can simplify the previous equation as $p(d_1 d_2 ... d_N) = p(d_1)p(d_2)...p(d_N)$. And by assuming they are identically distributed, we can conclude that there is a certain probability p_0 that for all $i = 1, 2...N$:

$$p(d_i) = p0 \qquad if\ d_i = 1 \tag{3.7}$$

$$p(d_i) = 1 - p0 \quad if\ d_i = 0\ . \tag{3.8}$$

By assuming both of this conditions, we get what we call an Independent and Identically Distributed (IID) random variable. We can rewrite the probability function once again as:

$$p(\mathbf{d}) = \prod_i p_0^{d_i}(1 - p_0)^{1-d_i}\ . \tag{3.9}$$

Furthermore, we will define a function $s(\mathbf{d}) = -\log p(\mathbf{d})$ and refer to it as the surprise of $\mathbf{d}$. Since the probability values only range from 0 to 1, the surprise will range from 0, when the probability is equal to 1, to infinity, when the probability is 0.

With this surprise function, we can define the function $H(\mathbf{D})$ called entropy, which is just the expected value of the surprise of $\mathbf{D}$ [60].

$$H(\mathbf{D}) \equiv \langle -\log\, p(\mathbf{D}) \rangle = -\sum_{\mathbf{d}} p(\mathbf{d})\, \log\, p(\mathbf{d})\ . \tag{3.10}$$

This latter function is useful, because it helps us define a way of comparing two different probability distributions $p_A(\mathbf{d})$ and $p_B(\mathbf{d})$ using the Kullback-Leibler (KL) divergence [61], also referred to as the relative entropy.

$$KL[p_A(\mathbf{D}), p_B(\mathbf{D})] = \sum_{\mathbf{d}} p_A(\mathbf{d})\, \log\, \frac{p_A(\mathbf{d})}{p_B(\mathbf{d})} \tag{3.11}$$

using the property of the logarithm division, we can rewrite the previous equation as:

$$KL[p_A(\mathbf{D}), p_B(\mathbf{D})] = \langle -\log\, p_B(\mathbf{D}) \rangle_A - \langle -\log\, p_A(\mathbf{D}) \rangle_A \tag{3.12}$$

and after looking at equation (3.10) we can see that it is the difference in surprise averaged by A.

3.3 The Network

Although there is no rule for the exact number of layers a HM must have, we will usually associate it with a 3 layer network, and from then, the generalization to a higher-layer structure is trivial. Considering the proposed number of layers, we can draw the network as shown in Fig. 3.2.

In this network, each node will be represented by a bit that takes the value 1 if the node has fired, or 0 otherwise. The number of neurons on layer $\mathbf{x}$ and $\mathbf{y}$ can vary, but usually, each layer has a less or equal amount of neurons as the layer immediately below, whereas the bottom layer $\mathbf{d}$ has a fixed size that

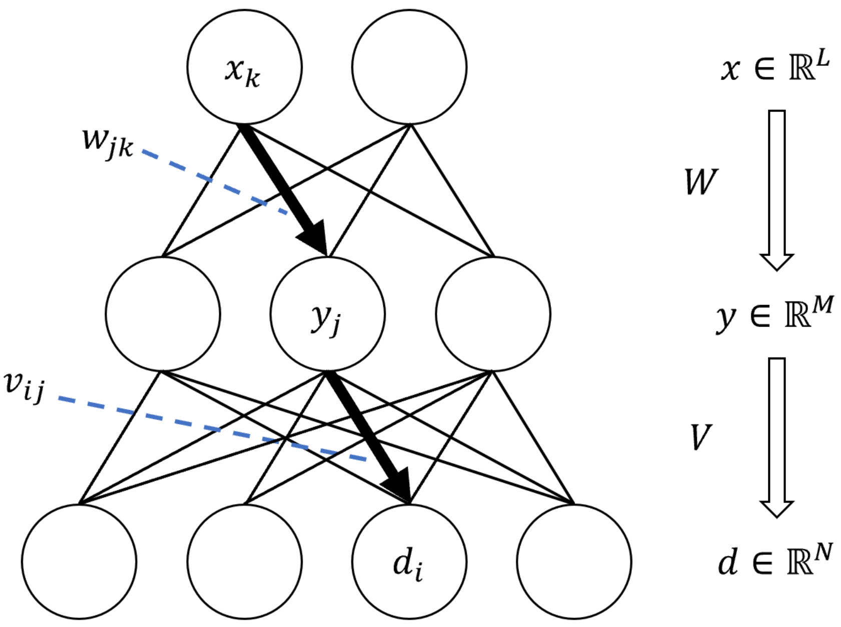

Figure 3.2: Example of a 3-layer Helmholtz Machine generative neural network. In the Helmholtz Machine model there is a similar network with connections from top to bottom, in addition to the represented one.

corresponds to the size of the input. The layer of nodes $\mathbf{x}$ will correspond to a bit vector that declares which nodes are being triggered. The network will then use the weights between the top and middle layers to determine which nodes will fire on the middle layer. For each node on the second layer $\mathbf{y}$ we will then calculate the sum of the weights $\mathbf{W}$ that are attached to that node plus a bias value and use that value in a certain function to determine wether they fire or not. From the middle to the bottom row, we will do the same process, but now we use $\mathbf{y}$ as the “input” and determine the activation of neurons in layer $\mathbf{d}$ with the weights $\mathbf{V}$ plus a bias. Which yields the following equations:

$$net\ y_j = \sum_{k=i}^{L} W_{jk}x_k + W_{j,L+1} \tag{3.13}$$

$$net\ d_j = \sum_{j=i}^{M} V_{ij}y_k + V_{j,M+1} \tag{3.14}$$

This functions can be simplified using matrix notation to simply $\mathbf{y} = W[\mathbf{x}|1]$ and $\mathbf{d} = V[\mathbf{y}|1]$, where $[\mathbf{a}|1]$ is the bit vector $\mathbf{a}$ with a 1 attached at the end.

After getting the weighted sum for each node, just like we did with Stochastic Networks at section (2.6) we will use a sigmoid function to determine the probability of a neuron firing. The sigmoid function is defined as:

$$\sigma(u) = \frac{1}{1+e^{-u}} \tag{3.15}$$

Using this probability, we will then sample a value of 0 or 1 for each node, taking into consideration the value obtained from the sigmoid function.

$$\mathbf{y} = SAMPLE[p_y] \quad where\ p_y = \sigma(W[\mathbf{x}|1]) \tag{3.16}$$

$$\mathbf{d} = SAMPLE[p_d] \quad where\ p_d = \sigma(V[\mathbf{y}|1]) \tag{3.17}$$

3.4 Generative Models

Generative models try to learn how the world generates patterns. Given their flexibility, these models can be used for multiple purposes depending on the type of data we have available and the problem we have to solve. Perhaps their most intuitive application would be for generating new data given a set of previously known examples, much like a task in a previous chapter where we wanted to create "normal" looking chessboards. There can be a wide range of examples for its applicability, such as text generation [62, 63], image inpainting [64, 65], texture generation [66] or generating realistic images [67]. Although these examples are mostly associated with unlabelled data, generative models can be used as classifiers when they learn from labeled examples. They can learn the joint distribution over inputs and labels, and be used as a normal classifier [68].

In HM, the Generative Model is symbiotically used with a Recognition model in what is described as a "wake-sleep" algorithm [21], where we perform a Top-down pattern Generation and a Bottom-up Pattern Recognition.

3.5 Top-down Pattern Generation

For top-down pattern generation, we will use the neural network example described in Fig. 3.3 and create a chain $1 \rightarrow \mathbf{x} \rightarrow \mathbf{y} \rightarrow \mathbf{d}$. This chain will try to communicate a distribution $p_G(\mathbf{d})$ learned from $p(\mathbf{d})$ not starting from $\mathbf{x}$ but a constant bias input of value 1. To be able to do this, the network is required to learn three different distributions: $p_G(\mathbf{x})$, $p_G(\mathbf{y}|\mathbf{x})$ and $p_G(\mathbf{d}|\mathbf{y})$.

By following the chain $\mathbf{x} \rightarrow \mathbf{y} \rightarrow \mathbf{d}$ we create a conditional independence from **x** and **d** given **y**, which allows us to separate $p_G(\mathbf{xyd})$ into the three distributions [58, 59], giving us:

$$p_G(\mathbf{xyd}) = p_G(\mathbf{d}|\mathbf{y})p_G(\mathbf{y}|\mathbf{x})p_G(\mathbf{x}) \ . \tag{3.18}$$

In the end we obtain a compact function $p_G(\mathbf{xyd})$ that stores the information of the distribution of all

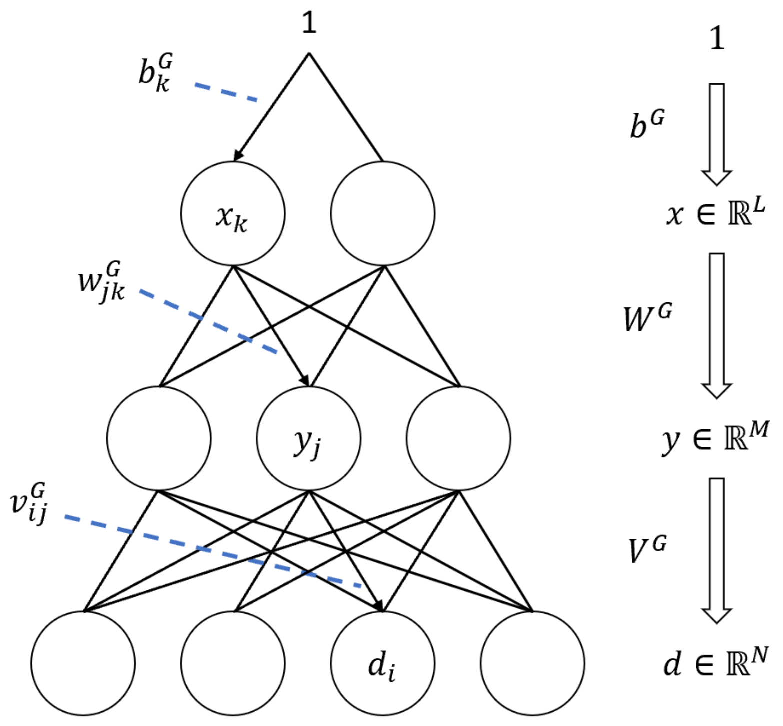

Figure 3.3: Top-down Generative network for a Helmholtz Machine.

three variables, and can be used to get the distribution we are most interested in, which is

$$p_G(\mathbf{d}) = \sum_{\mathbf{xy}} p_G(\mathbf{xyd}) \ . \tag{3.19}$$

The goal of this network will then be to approximate this distribution $p_G(\mathbf{d})$ as close as possible to the real data distribution $p(\mathbf{d})$.

In the HM, the layer **d** is called the data layer, whereas layers **x** and **y** are seen as hidden layers (See Fig. 3.3).

So we get from matrix $\mathbf{b}^G$ a distribution $p_G(\mathbf{x})$, from matrix $\mathbf{W}^G$ we get $p_G(\mathbf{y}|\mathbf{x})$ and from $\mathbf{V}^G$ we get $p_G(\mathbf{d}|\mathbf{y})$.

Finding the exact distribution of $p(\mathbf{d})$ may require a lot of information, and even trying to come up with an approximation can be exponentially hard [59, 69]. It has been argued that deeper networks have the potential to capture higher-level abstractions [70, 71]. So of course, given the low number of hidden layers, we are restraining the model in a way that will likely make it unable to describe the real $p(\mathbf{d})$ to perfection, but its goal will still be to get it as close as possible.

The probability values will be obtained according to the following rules that derive from (3.9) and (3.18):

$$p_G(\mathbf{x}) = \prod_k p_G(x_k)^{x_k}[1 - p_G(x_k)]^{1-x_k} \quad where\, p_G(x_k) = \sigma(b_k^G)$$

$$p_G(\mathbf{y}|\mathbf{x}) = \prod_j p_G(y_j|\mathbf{x})^{y_j}[1 - p_G(y_j|\mathbf{x})]^{1-y_j} \quad where\, p_G(y_j|\mathbf{x}) = \sigma(\sum_{k=1}^{L} w_{jk}^G x_k + w_{j,L+1}^G)$$

$$p_G(\mathbf{d}|\mathbf{y}) = \prod_i p_G(d_i|\mathbf{y})^{d_i}[1 - p_G(d_i|\mathbf{y})]^{1-d_i} \quad where\, p_G(d_i|\mathbf{y}) = \sigma(\sum_{j=1}^{M} v_{ij}^G y_j + v_{i,M+1}^G)\ .$$

One definition that will be helpful to retain is to call **xy** (determined by all the hidden layers) as the "explanation" of **d**.

3.6 Bottom-Up Pattern Recognition

After understanding the Top-Down approach, this will be much easier to understand, since we are doing what we did previously but with a reversed chain $\mathbf{d} \to \mathbf{y} \to \mathbf{x}$ described in Fig. 3.4. Note that, since we will be using a $\mathbf{d}$ that was generated by the world itself, we do not need to add that extra bias factor that we used to obtain the initial vector $\mathbf{x}$. Now, the quantity of interest would be $p_R(\mathbf{xy}|\mathbf{d}) = p_R(\mathbf{x}|\mathbf{y})p_R(\mathbf{y}|\mathbf{d})$. We will denominate the matrices that determine $p_R(\mathbf{x}|\mathbf{y})$ and $p_R(\mathbf{y}|\mathbf{d})$, as $\mathbf{W}^R$ and $\mathbf{V}^R$, respectively, in a similar way as we did for Pattern Generation.

The equation for recognition, would also be similar to the ones defined in the previous section:

$$p_R(\mathbf{x}|\mathbf{y}) = \prod_k p_R(x_k|\mathbf{y})^{x_k}[1 - p_R(x_k|\mathbf{y})]^{1-x_k} \quad where\, p_R(x_k|\mathbf{y}) = \sigma(\sum_{j=1}^{M} w_{kj}^R y_j + w_{k,M+1}^R)$$

$$p_R(\mathbf{y}|\mathbf{d}) = \prod_j p_R(y_j|\mathbf{d})^{y_j}[1 - p_R(y_j|\mathbf{d})]^{1-y_j} \quad where\, p_R(y_j|\mathbf{d}) = \sigma(\sum_{i=1}^{N} v_{ji}^R d_i + v_{j,N+1}^R)\ .$$

3.7 Energy

Now that we have the model defined, we will try to explain how this model evolves, more specifically, explain how the learning phase of these machine works. For that, we will revisit a topic mentioned in section (2.3), which is the energy function. We will first look at a probability of an explanation **xy** given a fixed piece of generated data **d**

$$p_G(\mathbf{xy}|\mathbf{d}) = \frac{p_G(\mathbf{xyd})}{p_G(\mathbf{d})} = \frac{p_G(\mathbf{xyd})}{\sum_{xy} p_G(\mathbf{xyd})}\ . \tag{3.20}$$

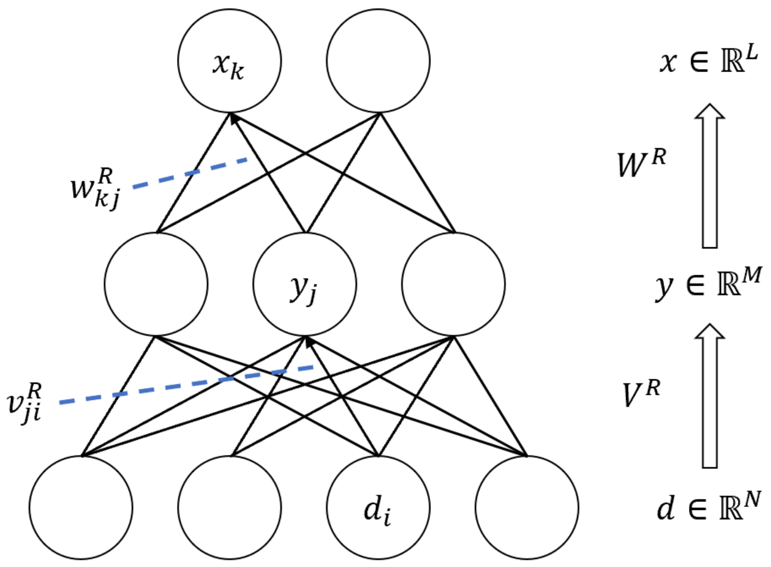

Figure 3.4: Bottom-Up Recognition network for a Helmholtz Machine.

We will define the energy function as:

$$E_G(\mathbf{xy}|\mathbf{d}) \equiv -\log\, p_G(\mathbf{xyd}) \tag{3.21}$$

and call $E_G(\mathbf{xy}|\mathbf{d})$ the energy of the explanation $\mathbf{xy}$ of the data pattern $\mathbf{d}$. Equation (3.20) now becomes:

$$p_G(\mathbf{xy}|\mathbf{d}) = \frac{e^{E_G(\mathbf{xy}|\mathbf{d})}}{\sum_{xy} e^{E_G(\mathbf{xy}|\mathbf{d})}} \tag{3.22}$$

This expression resembles the Boltzmann distribution which is present in the Boltzmann Machine.

3.8 Free Energy

The ultimate goal of the machine, is to find a $p_G(\mathbf{d})$ that is the same as $p(\mathbf{d})$, however, as stated before, this might be an impossible task, since the model actually restrains the complexity of $p_G(\mathbf{d})$. So with that in mind, we will try to get as close as possible to $p(\mathbf{d})$ as we can. So will will try to find a $p_G(\mathbf{d})$, meaning a $\mathbf{b}^G$, $\mathbf{W}^G$ and $\mathbf{V}^G$ that minimize:

$$\Phi(G) \equiv KL[p(\mathbf{D}), p_G(\mathbf{D})]\ . \tag{3.23}$$

So, by using the actual KL function we get:

$$\Phi(G) = \sum_d p(\mathbf{d})\, \log\, \frac{p(\mathbf{d})}{p_G(\mathbf{d})} = \sum_d p(\mathbf{d})\, \log\, p(\mathbf{d}) - \sum_d p(\mathbf{d})\, \log\, p_G(\mathbf{d})\ . \tag{3.24}$$

We can see that the left argument of the last expression does not depend on G, so therefore can be discarded when trying to find a $p_G(\mathbf{b})$ that minimizes the expression. So the function we want to minimize is simply:

$$\Phi_0 = -\sum_{d} p(\mathbf{d}) \log p_G(\mathbf{d}) = \langle -\log p_G(\mathbf{d}) \rangle_D \ . \tag{3.25}$$

To minimize this function we will do a well know gradient descent approach and compute the gradients ∇ of Φ_0

$$\nabla\Phi_0 = \sum_{d} p(\mathbf{d})\, \nabla[-\log p_G(\mathbf{d})] \ . \tag{3.26}$$

The gradient descent approach allows us to bypass the necessity of having the value of $p(\mathbf{d})$ since we will sample a $\mathbf{d}$ from the world in each iteration, and by using the multiple generated $\mathbf{d}$'s we will slowly make changes proportional to $\nabla\Phi_0$ that would end up giving us the same result as the one in equation (3.26).

We can see the quantity we ultimately want to minimize is $-\log p_G(\mathbf{d})$, or in other words, the surprise of $p_G(\mathbf{d})$.

Now, we reformulate this surprise function to get this curious equivalence

$$\begin{aligned}
-\log p_G(\mathbf{d}) &= -\log p_G(\mathbf{d})1 \\
&= -\log p_G(\mathbf{d}) \left[\sum_{\mathbf{xy}} p_G(\mathbf{xy} \mid \mathbf{d})\right] \\
&= -\sum_{\mathbf{xy}} p_G(\mathbf{xy} \mid \mathbf{d}) \log p_G(\mathbf{d}) \\
&= -\sum_{\mathbf{xy}} p_G(\mathbf{xy} \mid \mathbf{d}) \log \left[p_G(\mathbf{xyd})/p_G(\mathbf{xy} \mid \mathbf{d})\right] \\
&= -\sum_{\mathbf{xy}} p_G(\mathbf{xy} \mid \mathbf{d}) \log p_G(\mathbf{xyd}) + \sum_{\mathbf{xy}} p_G(\mathbf{xy} \mid \mathbf{d}) \log p_G(\mathbf{xy} \mid \mathbf{d}) \\
&= \sum_{\mathbf{xy}} p_G(\mathbf{xy} \mid \mathbf{d}) E_G(\mathbf{xy}|\mathbf{d}) - H_G(\mathbf{XY} \mid \mathbf{d}) \\
&= \langle E_G(\mathbf{XY}|\mathbf{d}) \rangle_G - H_G(\mathbf{XY} \mid \mathbf{d}) \ .
\end{aligned} \tag{3.27}$$

The first term is the average energy of the explanations given a generated pattern **d**, and the second term is the entropy of the explanations given a generated pattern $\mathbf{d}$.

In thermodynamics, the Helmholtz Free Energy [72] of a system is given by:

$$F = \langle E \rangle - TH \tag{3.28}$$

which is similar to expression (3.27) when we consider a temperature of 1. So we can call $F_G(\mathbf{d})$ as the free energy of **d** in G.

$$F_G(\mathbf{d}) = -log\, p_G(\mathbf{d}) = \langle E_G(\mathbf{XY}|\mathbf{d})\rangle_G - H_G(\mathbf{XY}|\mathbf{d}) \tag{3.29}$$

We obtain this curious equivalence of the terms “free energy of a pattern”, “surprise of a pattern” and “average energy minus entropy”.

So the problem of minimizing the KL divergence is the same as minimizing $\langle F_G(\mathbf{D})\rangle$. In order to do that, we would need to calculate:

$$\frac{\partial}{\partial b_k^G} F_G(\mathbf{d}) \qquad \frac{\partial}{\partial w_k^G} F_G(\mathbf{d}) \qquad \frac{\partial}{\partial v_k^G} F_G(\mathbf{d}) \tag{3.30}$$

to express $F_G(\mathbf{d})$ via the weights $\mathbf{b}^G$, $\mathbf{W}^G$ and $\mathbf{V}^G$. However, if we try to calculate these derivatives, we will see they do not give simple expressions, and would require a derivative for sums over all $\mathbf{X}$ and $\mathbf{Y}$ in $\sum_{\mathbf{xy}} p_G(\mathbf{xyd})$ which makes it intractable.

3.9 Variational Free Energy

So far, we have not yet mentioned the utility of the Recognition model. But in this section, we will use it to solve the previously mentioned intractability problem. Instead of trying to compare the Generative distribution with the real distribution, we will compare it instead with the Recognition distribution $p_R(\mathbf{d})$. However, for what we will do next, there is no need to know the whole $p_R(\mathbf{xyd})$, but only the distribution $p_R(\mathbf{xy}|\mathbf{d})$, that can be easily obtained from the Recognition model weight vectors.

First, we will compute the Kullback-Leibler divergence of explanations from p_R to the generative distribution p_G given a fixed pattern $\mathbf{d}$:

$$\mathrm{KL}\left[p_R(\mathbf{XY} \mid \mathbf{d}), p_G(\mathbf{XY} \mid \mathbf{d})\right] \equiv \sum_{\mathbf{xy}} p_R(\mathbf{xy} \mid \mathbf{d}) \log \frac{p_R(\mathbf{xy} \mid \mathbf{d})}{p_G(\mathbf{xy} \mid \mathbf{d})} \; . \tag{3.31}$$

Which can be rewritten as (See appendix A.1):

$$\mathrm{KL}\left[p_R(\mathbf{XY} \mid \mathbf{d}), p_G(\mathbf{XY} \mid \mathbf{d})\right] = -H_R(\mathbf{XY} \mid \mathbf{d}) + \langle \mathrm{E_G}(\mathbf{XY}|\mathbf{d})\rangle_R - F_G(\mathbf{d}) \; . \tag{3.32}$$

So now we can write the free energy $F_G(\mathbf{d})$ in two different ways:

$$F_G(\mathbf{d}) = \langle E_G(\mathbf{XY}|\mathbf{d})\rangle_G - H_G(\mathbf{XY} \mid \mathbf{d}) \tag{3.33}$$

$$F_G(\mathbf{d}) = \langle E_G(\mathbf{XY}|\mathbf{d})\rangle_R - H_R(\mathbf{XY} \mid \mathbf{d}) - \mathrm{KL}\left[p_R(\mathbf{XY} \mid \mathbf{d}), p_G(\mathbf{XY} \mid \mathbf{d})\right] \; . \tag{3.34}$$

We end up with two different expressions for the generative free energy of a HM. Where the second expression involves a new distribution p_R and is related to the concept of variational free energy.

We define variational free energy (from R to G) as:

$$F_G^R(\mathbf{d}) \equiv \langle E_G(\mathbf{XY}|\mathbf{d})\rangle_R - H_R(\mathbf{XY}|\mathbf{d}) \tag{3.35}$$

$$= F_G(\mathbf{d}) + KL[p_R(\mathbf{XY}|\mathbf{d}), p_G(\mathbf{XY}|\mathbf{d})] \; . \tag{3.36}$$

So we can safely say, after realizing that $KL[p_G(\mathbf{XY}|\mathbf{d}), p_G(\mathbf{XY}|\mathbf{d})] = 0$ the last equation gives us:

$$F_G^G(\mathbf{d}) = F_G(\mathbf{d}) \; . \tag{3.37}$$

Another important factor, is that the KL divergence is never smaller than zero, so we can write (3.34) as:

$$\langle E_G(\mathbf{XY}|\mathbf{d})\rangle_R - H_G(\mathbf{XY}|\mathbf{d}) - F_G(\mathbf{d}) \geq 0 \tag{3.38}$$

or equivalently,

$$F_G(\mathbf{d}) \leq \langle E_G(\mathbf{XY}|\mathbf{d})\rangle_R - H_G(\mathbf{XY}|\mathbf{d}) = F_G^R(\mathbf{d}) \; . \tag{3.39}$$

This essentially means that $F_G^R(\mathbf{d})$ is an upper bound to $F_G(\mathbf{d})$ and by pushing the variational free energy down, the free energy will also have to go down. So by minimizing $F_G^R(\mathbf{d})$ we would get $p_R(\mathbf{d})$ closer to $p_G(\mathbf{d})$ also minimizing $F_G(\mathbf{d})$ which was the initial goal. Moreover, we will see in later sections that the intractability problem that appeared in the previous section, does not exist for the derivatives of this function.

This idea of minimizing the variational free energy is very important and is used in a lot of machine learning algorithms [73, 74].

3.10 Learning

The actual learning of the HM will consist ultimately of minimizing the function $F_G(\mathbf{d})$. However, we will be doing it by minimizing $F_G^R(\mathbf{d})$ instead, as stated in the previous section. The learning algorithm will be divided into two different phases, one that makes changes to $p_R(\mathbf{d})$ and another one that changes $p_G(\mathbf{d})$ in what is defined as a "wake-sleep" algorithm.

In the wake phase, we will make small changes to the generative weights, to minimize equation (3.36)

$$F_G^R(\mathbf{d}) = F_G(\mathbf{d}) + KL[p_R(\mathbf{XY}|\mathbf{d}), p_G(\mathbf{XY}|\mathbf{d})] \; . \tag{3.40}$$

Where **d** will be a sample from the real world (created with the actual $p(\mathbf{d})$).

In the sleep phase, we will minimize a similar function:

$$\widetilde{F}_G^R(\mathbf{d}) = F_G(\mathbf{d}) + KL[p_G(\mathbf{XY}|\mathbf{d}), p_R(\mathbf{XY}|\mathbf{d})] \ . \tag{3.41}$$

Note that since the KL function is not symmetric, these functions will be slightly different.

The adjustments done on each step will then be determined using gradient descent.

3.11 Wake Phase

In this phase, we want to find the derivatives for the variational free energy acording to the generative weights:

$$F_G^R(\mathbf{d}) \equiv \langle E_G(\mathbf{XY}|\mathbf{d})\rangle_R - H_R(\mathbf{XY}|\mathbf{d}) \ . \tag{3.42}$$

Since the second factor does not depend on G. The derivative of $F_G^R(\mathbf{d})$ is simply:

$$\nabla_G F_G^R(\mathbf{d}) = \langle \nabla_G E_G(\mathbf{XY}|\mathbf{d})\rangle_R \ . \tag{3.43}$$

The gradient of the free energy is given by:

$$\nabla_G E_G(\mathbf{XY}|\mathbf{d}) = -\nabla \log\, p_G(\mathbf{xyd}) \tag{3.44}$$

$$= -\nabla \log\, p_G(\mathbf{x}) - \nabla \log\, p_G(\mathbf{y}|\mathbf{x}) - \nabla \log\, p_G(\mathbf{d}|\mathbf{y}) \ . \tag{3.45}$$

By recalling the expressions for these three factors defined on section (3.5) we get:

$$\begin{aligned}
\log p_G(\mathbf{x}) &= \log \prod_k \xi_k^{x_k} \left(1-\xi_k\right)^{1-x_k} \quad \text{where } \xi_k \equiv \sigma\left(b_k^G\right) \\
&= \sum_k x_k \log \xi_k + \sum_k \left(1-x_k\right) \log\left(1-\xi_k\right) \\
\log p_G(\mathbf{y} \mid \mathbf{x}) &= \log \prod_j \psi_j^{y_j} \left(1-\psi_j\right)^{1-y_j} \quad \text{where } \psi_j \equiv \sigma\left(\sum_{k=1}^{L} w_{jk}^G x_k + w_{j,L+1}^G\right) \\
&= \sum_j y_j \log \psi_j + \sum_j \left(1-y_j\right) \log\left(1-\psi_j\right) \\
\log p_G(\mathbf{d} \mid \mathbf{y}) &= \log \prod_i \delta_i^{d_i} \left(1-\delta_i\right)^{1-d_i} \quad \text{where } \delta_i \equiv \sigma\left(\sum_{j=1}^{M} {v^G}_{ij} y_j + v_{i,M+1}^G\right) \\
&= \sum_i d_i \log \delta_i \quad + \sum_i \left(1-d_i\right) \log\left(1-\delta_i\right) \ .
\end{aligned}$$

With these in mind, after the calculation for the derivatives of all the connection weights matrices (See appendix A.2), we get these simple equations in vector form:

$$\nabla_{\mathbf{b}^G} E_{\mathrm{G}}(\mathbf{xy}|\mathbf{d}) = -(\mathbf{x} - \xi)$$
$$\nabla_{\mathbf{w}^G} E_{\mathrm{G}}(\mathbf{xy}|\mathbf{d}) = -(\mathbf{y} - \psi)[\mathbf{x} \mid 1]^{\mathrm{T}}$$
$$\nabla_{\mathbf{v}^G} E_{\mathrm{G}}(\mathbf{xy}|\mathbf{d}) = -(\mathbf{d} - \boldsymbol{\delta})[\mathbf{y} \mid 1]^{\mathrm{T}} \ .$$

The gradient descent for each layer, with learning rate ε will be:

$$\begin{array}{l}\mathbf{b}^{\mathrm{G}} + = \varepsilon(\mathbf{x} - \xi)\\ \mathbf{W}^{\mathrm{G}} + = \varepsilon(\mathbf{y} - \Psi)[\mathbf{x} \mid 1]^{\mathrm{T}}\\ \mathbf{V}^{\mathrm{G}} + = \varepsilon(\mathbf{d} - \boldsymbol{\delta})[\mathbf{y} \mid 1]^{\mathrm{T}} \ .\end{array}$$

So we can now write a pseudo code for the whole wake phase like shown in algorithm 3.1.

Algorithm 3.1: Wake Phase

```
$\mathbf{d} = getImageFromWorld()$

// Pass information up through recognition network

$\mathbf{y} = SAMPLE(\sigma(\mathbf{V}_R[\mathbf{d}|1]^T)$
$\mathbf{x} = SAMPLE(\sigma(\mathbf{W}_R[\mathbf{y}|1]^T)$

// Pass back down explanation xy through the generation network

$\xi = \sigma(\mathbf{b}^G)$
$\boldsymbol{\Psi} = \sigma(\mathbf{W}^G[\mathbf{x}|1]^T)$
$\delta = \sigma(\mathbf{V}^G[\mathbf{y}|1]^T)$

// Adjust generation weights with learning rate $\varepsilon$

$\mathbf{b}^G \mathrel{+}= \varepsilon(\mathbf{x} - \xi)$
$\mathbf{W}^G \mathrel{+}= \varepsilon(\mathbf{y} - \boldsymbol{\Psi})[\mathbf{x}|1]^T$
$\mathbf{V}^G \mathrel{+}= \varepsilon(\mathbf{d} - \delta)[\mathbf{y}|1]^T$
```

3.12 Sleep Phase

For the sleep phase, we will follow the same trail of thoughts we had on the wake phase, but now the function we want to minimize is:

$$\widetilde{F}^R_G(\mathbf{d}) = F_G(\mathbf{d}) + \mathrm{KL}\left[p_G(\mathbf{XY} \mid \mathbf{d}), p_R(\mathbf{XY} \mid \mathbf{d})\right] \ . \tag{3.46}$$

However, unlike in the previous section, we will be interested in the dependence on R. On this function, only the last factor depends on R, so the first factor can be regarded as a constant. Giving us:

$$\begin{aligned}\nabla_R \widetilde{F}^R(\mathbf{d}) &= \nabla_R \mathrm{KL}\left[p_G(\mathbf{XY} \mid \mathbf{d}), p_R(\mathbf{XY} \mid \mathbf{d})\right]\\ &= -\left\langle \nabla_R \log p_R(\mathbf{XY} \mid \mathbf{d})\right\rangle_G \ .\end{aligned}$$

The derivatives for the logaritmic function are:

$$\begin{aligned}\nabla_R \log p_R(\mathbf{xy} \mid \mathbf{d}) &= \nabla_R \log p_R(\mathbf{y} \mid \mathbf{d}) p_R(\mathbf{x} \mid \mathbf{y}) \\ &= \nabla_R \log p_R(\mathbf{y} \mid \mathbf{d}) + \nabla_R \log p_R(\mathbf{x} \mid \mathbf{y}) \ .\end{aligned}$$

By recalling once again the values of $p_R(\mathbf{y}|\mathbf{d})$ and $p_R(\mathbf{x}|\mathbf{y})$ from section (3.6) we obtain:

$$\begin{aligned}\log p_R(\mathbf{x} \mid \mathbf{y}) =& \log \prod_k \xi_k^{x_k} \left(1-\xi_k\right)^{1-x_k} \quad \text{where } \xi_k = \sigma\left(\sum_{j=1}^{M} w_{kj}^R y_j + w_{k,M+1}^R\right) \\ =& \sum_k x_k \log \xi_k + \sum_k \left(1-x_k\right) \log\left(1-\xi_k\right) \\ \log p_R(\mathbf{y} \mid \mathbf{d}) =& \log \prod_j \psi_j^{y_j} \left(1-\psi_j\right)^{1-y_j} \quad \text{where } \psi_j = \sigma\left(\sum_{i=1}^{N} v_{ji}^R d_i + v^R{}_{j,N+1}\right) \\ =& \sum_j y_j \log \psi_j \quad + \sum_j \left(1-y_j\right) \log\left(1-\psi_j\right)\end{aligned}$$

and after calculating for the derivatives of all the connection weights of the recognition matrices (See appendix A.3) we end up once again with a simple vector form:

$$\begin{aligned}\nabla_{\mathbf{w}^R} \log p_R(\mathbf{xy} \mid \mathbf{d}) = (\mathbf{x} - \xi)[\mathbf{y} \mid 1]^{\mathrm{T}} \\ \nabla_{\mathbf{v}^R} \log p_R(\mathbf{xy} \mid \mathbf{d}) = (\mathbf{y} - \boldsymbol{\psi})[\mathbf{d} \mid 1]^{\mathrm{T}}\end{aligned}$$

and a gradient rule:

$$\begin{aligned}\mathbf{W}^R{+}= \varepsilon(\mathbf{x} - \xi)[\mathbf{y} \mid 1]^{\mathrm{T}} \\ \mathbf{V}^R{+}= \varepsilon(\mathbf{y} - \boldsymbol{\psi})[\mathbf{d} \mid 1]^{\mathrm{T}}.\end{aligned}$$

Note that unlike before, the factor **d** is not a sample from the real world. We get this **d** from the chain $1 \to \mathbf{x} \to \mathbf{y} \to \mathbf{d}$, by sampling the vector **x** according to weights of the matrix $\mathbf{b}^G$.

We can now describe the pseudo code for the sleep phase like shown in algorithm 3.2.

3.13 The Wake-Sleep Algorithm

To compute the whole algorithm, we just need to initialize all the weight vectors to 0 and combine the two algorithms described in the previous sections, until the generative distribution is close enough to the real distribution.

The idea behind this algorithm is that when you get an image from the world, you will get an intuition about the reasons behind that image, so you update what you think the rules of the world are. Some of those intuitions will be true, and others will not. For example, if you see a round red apple, you could be tempted to believe that all apples are round, which is true, and that all apples are red, which is false. So after seeing a pattern, you will most likely get some good insights on the world's probability distribution and some over-fitted ones.

When you create a dream, you will be doing the inverse, from what you know of the world, you will try to create an image that exists in the world, so continuing with the apple example, you could, for example,

Algorithm 3.2: Sleep Phase

```
// Create a dream
$\mathbf{x} = SAMPLE(\sigma(\mathbf{b}^G))$
// Pass down dream through the generation network
$\mathbf{y} = SAMPLE(\sigma(\mathbf{W}^G[\mathbf{x}|1]^T)$
$\mathbf{d} = SAMPLE(\sigma(\mathbf{V}^G[\mathbf{y}|1]^T)$
// Pass information back up through the recognition network
$\mathbf{\Psi} = \sigma(\mathbf{V}^R[\mathbf{d}|1]^T)$
$\xi = \sigma(\mathbf{W}^R[\mathbf{y}|1]^T)$
// Adjust recognition weights with learning rate $\varepsilon$
$\mathbf{V}^R \mathrel{+}= \varepsilon(\mathbf{y} - \mathbf{\Psi})[\mathbf{d}|1]^T$
$\mathbf{W}^R \mathrel{+}= \varepsilon(\mathbf{x} - \xi)[\mathbf{y}|1]^T$
```

Algorithm 3.3: Wake-Sleep Algorithm

```
InitializeWeights($\mathbf{V}^G, \mathbf{W}^G, \mathbf{b}^G, \mathbf{V}^R, \mathbf{W}^R$)
while Model hasn't converged (or until a certain number of iterations) do
    Wake Phase to update Generative Weights
    Sleep Phase to update Recognition Weights
```

dream of a squared red apple. If the Recognition model was good, it would find that odd, readjusting its perceptions. If for instance, you had dreamed of a round green apple, hopefully, the Recognition model would not find that odd and reinforce its beliefs.

The dreams you have, will reinforce the good features you found, and undermine the bad ones. While the wake phases simply find more and more explanations for the images the world has.

To summarize, the Wake Phase is adding more information to the Generative Model, from what it has perceived in the real world. And in the Sleep Phase, it will remove the useless or wrong information it got, and only keep the useful one. By continuously doing these iterations of adding and trimming information, we end up with a generative distribution that is very close to the real one.

3.14 How to use a trained Helmholtz Machine

After a Helmholtz Machine is trained, we can focus on the Generative Network and use it to create new patterns that will be similar to the ones that exist in the world. We do this in a very similar way to section (3.5) where we described top-down pattern generation. After the machine is trained, it would only dream about things that exist in the world, so the dreams it produces will be in accord with the real world's distribution [58].

4

Problem and Proposed Solution

Contents

4.1 Improving Wake-Sleep

In spite of the WS algorithm being interesting from a neuro-scientific perspective, its' lack of efficiency [75] and ability to perform as well as other learning algorithms have led it to be less and less explored in recent years. One of its biggest disadvantages is that when the complexity of the network increases, the algorithm's performance starts to be less impressive. If the complexity of the world we are trying to mimic increases, our model needs to be able to capture higher-level abstractions and generalize better, which can be done by increasing the size of its network [70, 76]. However, by increasing the number of neurons on a model's network, the size of the search space also grows. When any model is searching through the energy surface it can easily get stuck at a sub-optimal local minima [1], and we believe this is the main problem of the HM with a large hidden network.

Our proposition to overcome this problem is to provide the algorithm with a heuristic for it to be more consistently led to optimal solutions.

Heuristics consist of ways to navigate the search space, that guide the algorithm to either find a better solution, find a solution faster, or both. They can be seen as generic rules that apply to a majority of the cases, allowing the agent to avoid exploring search paths that seem unpromising.

4.1.1 Multi-level Data Representation and Human Image Perception

One thing that might help humans understand what they see in a better and more structured way, is the ability to evaluate a given visual image at different scales. Many studies point to the fact that the human brain processes visual inquiries at different resolutions [77, 78]. This multi-level biological visual analysis could be one of the many keys that enable the human brain to capture the world it perceives in such a robust and accurate way despite the obvious extreme complexity of its neural network.

A way to incorporate this multi-level perception into the HM is by using an Image Pyramid representation of the dataset [79]. The Image Pyramid is a simple way of having multi-level data representation that enables models to detect patterns on different scales. It consists of creating lower-level representations of the original images in a convolutional fashion, reducing an image by a factor each time, and creating a "sequence of copies of an original image in which both sample density and resolution are decreased in regular steps" [80], like shown in Fig. 4.1. Introducing this data representation to the training of the network would be in accordance with the high biological plausibility that motivated the interest in the HM model and by doing so we hope to guide its learning, in a way that first detects high-level patterns, and then as we add details to the samples, it would learn more correlations on different scales, acting as a heuristic to overcome the exponential increase of the search space that inevitably comes with the increase of the number of hidden layers.

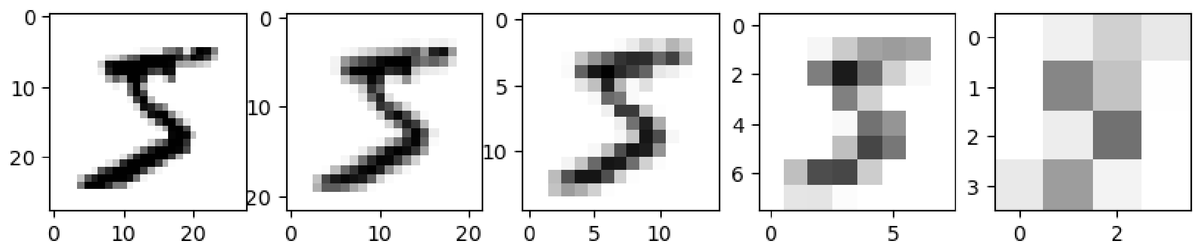
Figure 4.1: Example of an Image Pyramid representation of a handwritten number 5 generated by continuously downsampling the original image on the left.

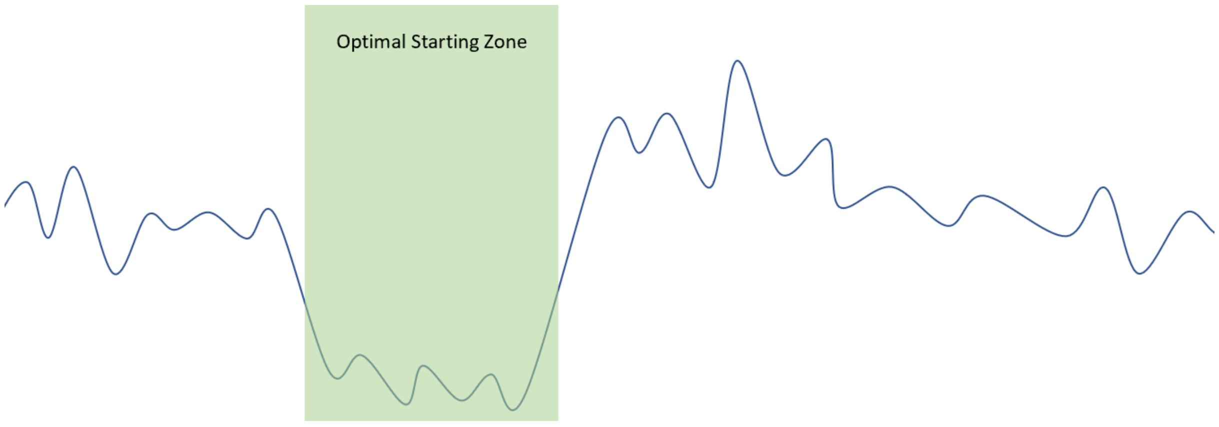

Figure 4.2: Example of a two-dimensional energy landscape described by a blue curve. When traveling the energy surface with a non-stochastic gradient method, our model would move in a way similar to a sphere being dropped in the said landscape, moved by the force of gravity. We can understand that the starting configuration of our model, meaning, the starting position on the landscape, would have a major impact on the absolute value of the minima achieved. There is in this case an optimal starting zone that we highlighted in green, where if the initial configuration corresponds to a point in that zone, the minima reached would be generally better.

4.1.2 Image Pyramid Heuristic for Helmholtz Machines

One way to guide our model's training is by configuring its initial position on the search space, to a zone where we believe the probability of finding a smaller local minima is higher like the one highlighted in Fig. 4.2.

Weight initialization has been known to have a significant impact on the model's convergence state when training with deep neural networks [7,81]. The idea of the heuristic we want to apply to the learning of the HM is to initialize the weights of the network so that the initial configuration contains queues of the image particularities at different scales.

We propose to create a network with multiple hidden layers, with increasing sizes from top to bottom where each layer must correspond to the size of a down-sampled image.

Then we iteratively train the machines layer by layer, starting from a low-resolution sample of the original world's images [82–84], and add additional layers while increasing the images' resolution as

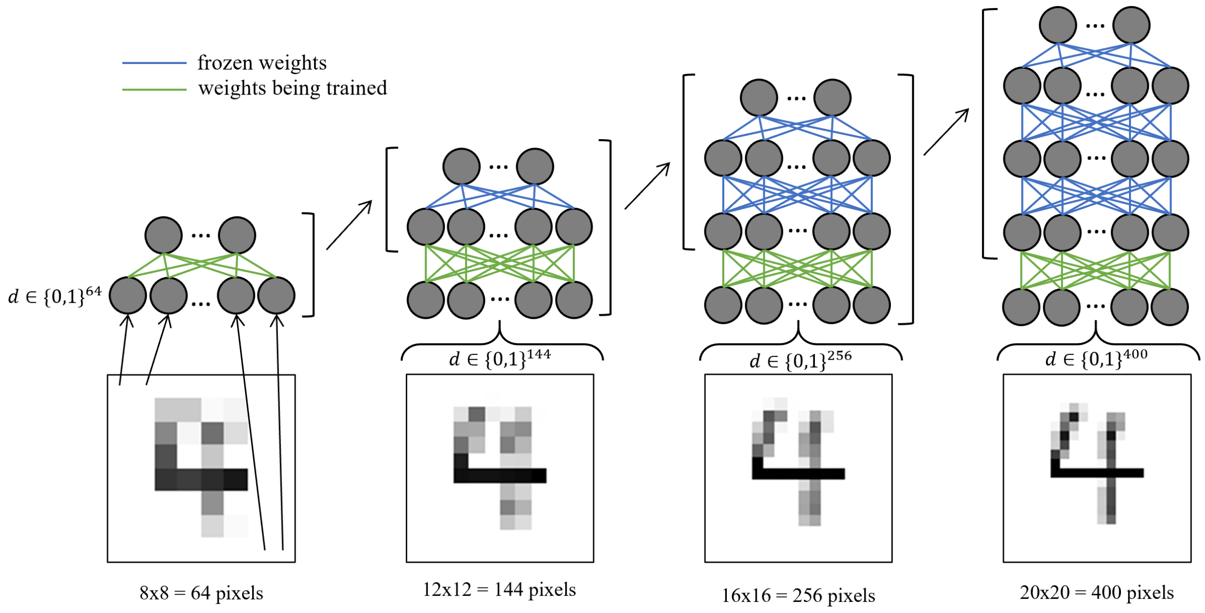

Figure 4.3: Proposed Weight Initialization for a Helmholtz Machine. We use down-sampled images to train the smaller hidden layer and proceed to freeze the learned weights, then we up-sample the previously used images and train a newly added layer, then we freeze the new layer's weights repeating the process until we reach the original images' resolution. This way, information on all detail levels should be present in the initialization of the weights.

described in Fig. 4.3. The downsampled images would be equivalent to the idea of a blurred image where only the global details could be retrieved, and therefore, the first layer trained would in theory be able to recognize global features of the world's distribution. After learning a good distribution for a certain level of resolution, the model would then freeze the weights it learned for this layer, preventing it from losing its global perception when learning with more detailed data. Then we would increase the resolution and train an additional layer the same way we did with the previous one.

When we reach the last layer, we should have incorporated in our machine's weights the information of all resolution levels, and after it, we would conventionally train the HM, with the predetermined initial configuration.

5 Results and Experiments

Contents

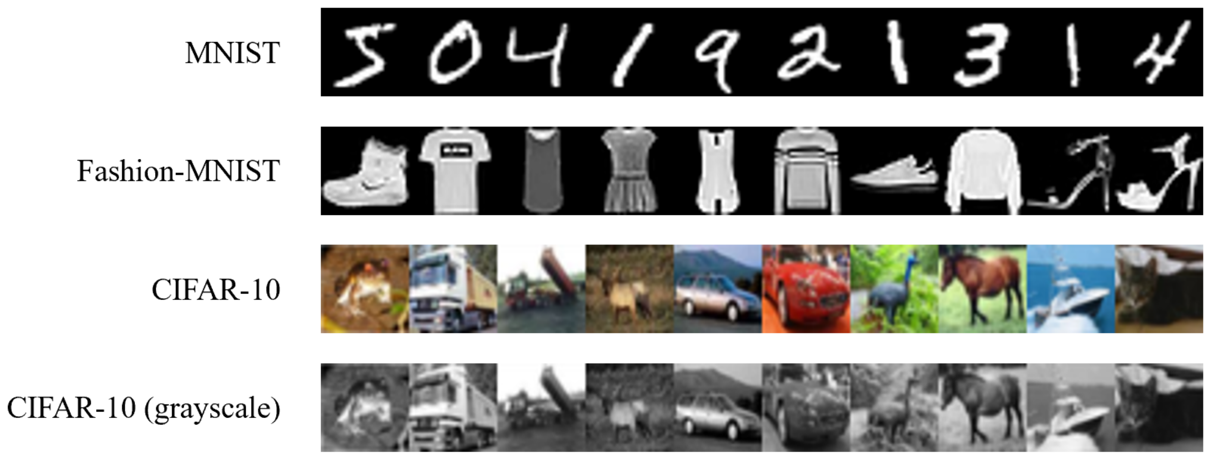

Figure 5.1: Ten samples taken from the Train Set of all used Datasets, including a grayscale transformation of the original RGB triplets from the CIFAR-10 images.

In this chapter, we will propose and perform several experiments to confirm our previously stated hypothesis and test if our proposed heuristic provides significant advantages in the generative performance of the Helmholtz Machine.

We will first use the MNIST dataset of handwritten digits [10] to train our models and perform our experiments. This dataset has a relatively small complexity but still allows us to compare results for different implementations in a permissive environment and to gather insights that could otherwise become imperceivable intricacies in more complex domains. Moreover, results on this dataset motivate future experiments on more complex datasets and act as a perfect stepping stone from conception to practical usage of any model.

After performing the proposed experiments on this dataset, we will test our heuristic on two other datasets, Fashion-MNIST [85] and CIFAR-10 [86]. Both of these datasets have higher complexity than the MNIST dataset of handwritten digits, with CIFAR-10 having the highest complexity of the three. In Fig. 5.1 we present 10 random samples of each datasest, so you can visually see the complexity difference.

5.1 Is the locality of Wake-Sleep a problem when training Deep Networks?

One factor that may penalize the Helmholtz Machine's performance with deep architectures is the locality of the Wake-Sleep algorithm.

When adding hidden layers to our model, we are increasing the number of free parameters, so in theory, we would be increasing the network's potential to represent the world's data. However, we believe that the HM does not take full advantage of this augmentation in capacity, due to the fact the local updates present on the learning rule make the learning progressively harder to be propagated throughout consecutive hidden layers.

5.1.1 Proposed Experiment:

To test this hypothesis, we trained a Helmholtz Machine with a deep architecture, and used its recognition layers' activations as inputs for a simple Logistic Regression (LR) Model, to see how well the HM's hidden representations are able to linearly separate the problem space. This approach takes advantage of the fact that our model is simultaneously training a recognition and a generative model. The quality of the generative model is related to the capability of our model to generate good lower-level explanations of the observed samples with its recognition network. So by testing our model's input representation at different steps of the recognition chain, we can see what hidden layers are responsible for identifying the majority of the learned features. For this experiment, we chose an architecture with 6 hidden layers of size 625 (25×25), 484 (22×22), 289 (17×17), 196 (14×14), 100 (10×10), 16 (4×4) starting from the input layer.

5.1.2 Results:

From the results described in Fig. 5.2 we can see that the majority of the separation of the problem space is done in the first layer. This suggests that with the local WS learning rule, as the size of the network increases, a large part of the information will not be propagated through the network, and will store most of the information regarding the learned features at the surface of the deep network, meaning that even tho we are adding more descriptive power to the model by increasing its depth, it is incapable of taking advantage from it.

5.2 Does the Multi-level Data Representation solve this problem?

5.2.1 Proposed Experiment:

We repeat the experiment proposed in the previous section, training a HM with the same architecture as the previously described one, but this time initialized with the proposed Image Pyramid method.

5.2.2 Results:

In this experiment, the results presented in Fig. 5.3 show not only higher overall accuracy values, but more importantly, a smooth decrease of the layer's descriptive power as we reach higher layers, which

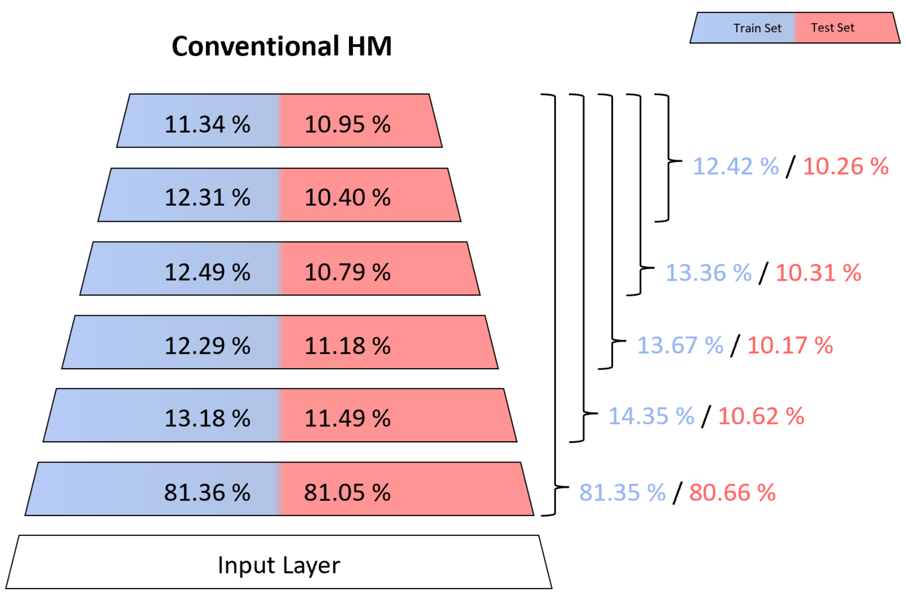

Figure 5.2: Accuracy of the LRs trained with different subsets of Helmholtz Machine's Recognition Network's hidden representations of the MNIST samples, both in the train and test set. The values on each layer correspond to the usage of the neuron activations on that single layer, while the brackets correspond to the concatenation of activations on the layers they aggregate. We can see that most of the class separation is done in the first layers, whereas layers that are further away from the input layer bear almost no information about the world's distribution.

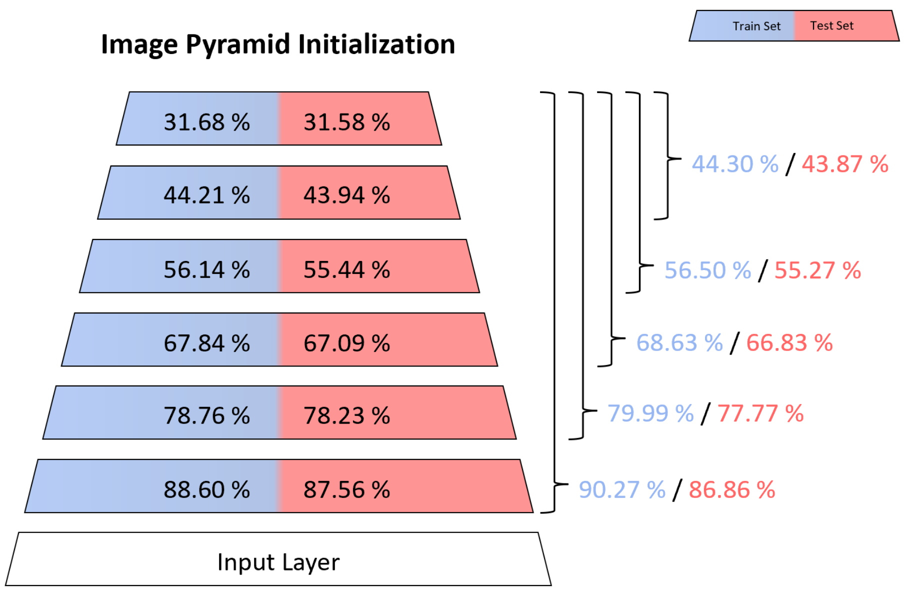

Figure 5.3: Accuracy of the LRs trained with different subsets of Helmholtz Machine's Recognition Network's hidden representations of the MNIST samples, both in the train and test set. The values on each layer correspond to the usage of the neuron activations on that single layer, while the brackets correspond to the concatenation of activations on the layers they aggregate. Image Pyramid Initialization shows a progressive increase of the class separation capability as we get closer to the input layer, suggesting a better use of the network as a whole to define the main features of the samples.

is expected since the number of neurons on each layer is smaller as we go up the network. These results are a good indicator that our heuristic provides an advantage for the recognition network's world representation, rendering it capable of fully using its deep hidden layers to store meaningful information.

5.3 Does Multi-level Data Representation provide a generative advantage?

In the previous section, we focused solely on the evaluation of the Recognition Network. We presented evidence for our claim that the Image Pyramid Initialization allows for better usage of the capacity of the deep network, and hypothesized that a better Recognition Model would also be translated into a better Generative one.

Consequentially, we should be able to see a similar improvement when testing the Generative Network and prove that when using Image Pyramid Initialization, we take full advantage of the network's increase in size.

5.3.1 Proposed Experiment:

To test our hypothesis, we will define an architecture for a Neural Network and create two different machines with that same architecture. One of the machines will use the Image Pyramid Initialization (Fig. 5.4 a), and the other will use a classic Random Initialization (Fig. 5.4 b). After, we proceed to train them with a small train set of size N (e.g. 2 samples) and see if the network is able to generate it back. To do this, after the machine has been trained, we generate a large number of samples G, and find the euclidean distance from a given sample to the train set. Then, we choose the minimum distance observed, and claim that the generated sample corresponds to that particular train set image. We keep the smallest distance observed and the correspondent train set sample and repeat the same process for all generated samples. We end up with an array of closest distances, and an array of the correspondent train set samples. With the array of distances, we simply calculate the mean, and with the correspondence array, we first create an array with the size of the Train Set where each index corresponds to the representation fraction of the same index train sample in those G generations, creating a density vector (e.g. following the previous supposition that we only have 2 samples, x_0 and x_1, if the model generated 6 samples closer to x_0 and 4 samples closer to x_1, the corresponding density vector would be $[0.6, 0.4]$). From that density vector, we take two different measures, the first one being the Entropy, and the second the Number of Unrepresented Samples. The Entropy will be close to one if the machine generates the same number of samples for each Train Set image, and closer to zero as the model starts to replicate some of the world's images more frequently than others, so in general this measure relates to how well

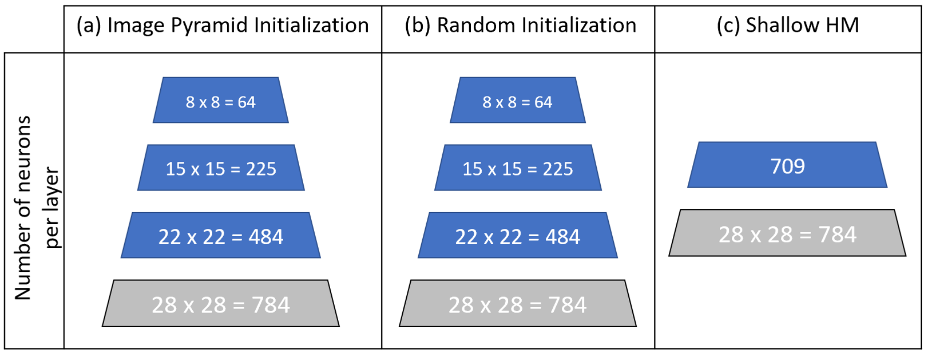

Figure 5.4: Proposed architecture for 3 different Helmholtz Machines. Machines (a) and (b) have three hidden layers and an equal architecture with a number of free weights $\rho \approx 64{+}64{\times}225{+}225{\times}484{+}484{\times}784 = 502820$, while machine (a) consists of a single hidden layer machine with $\rho \approx 709 + 709 \times 784 = 556565$. Thus, the descriptive power of the machines should follow the same order as the number of free parameters $\rho_{(a)} = \rho_{(b)} < \rho_{(c)}$. Machines (b), and (c) are initialized with random values, whereas machine (a) is initialized using our proposed multi-level representation method.

the machine is capturing the real world's distribution. The Number of Unrepresented Samples shows how many samples the machine has “forgotten”, meaning that it was unable to closely replicate a sample present during its training, despite having generated a large pool of samples. If this measure is 0, the machine was able to remember all learned samples, but as the number of samples in the world increases, the machine will inevitably become unable to represent some of them. With this measure, we can see the breaking point regarding the world's number of samples at which the model becomes unable to remember all samples seen and can be used as a comparison measure between different models.

In addition to our initial claim, we also believe that using a deep network with Image Pyramid Initialization with a certain number of free parameters has a generative advantage when compared to a shallow one-layer network with a higher number of free parameters given the compositional properties that multi-layer networks allow for. Therefore we include in the collection of machines an additional large shallow HM with a hidden layer size equal to the number of neurons of the two bigger hidden layers of the deep architecture (Fig. 5.4 c), assuring it has a higher number of free parameters.

5.3.2 Results:

The results in Fig. 5.5 are very promising and suggest that our initial intuition was true. Regarding the Number of Unrepresented Samples, we can see that up to $N = 64$ all machines can fully represent the dataset, from 128 to 256 the shallow machine starts to be unable to represent certain samples, while the deeper architectures can still fully represent them. From 512 to 1024 we can start to see a difference in the performance of the Random to the Multi-level based initialization, giving an edge to the latter one.

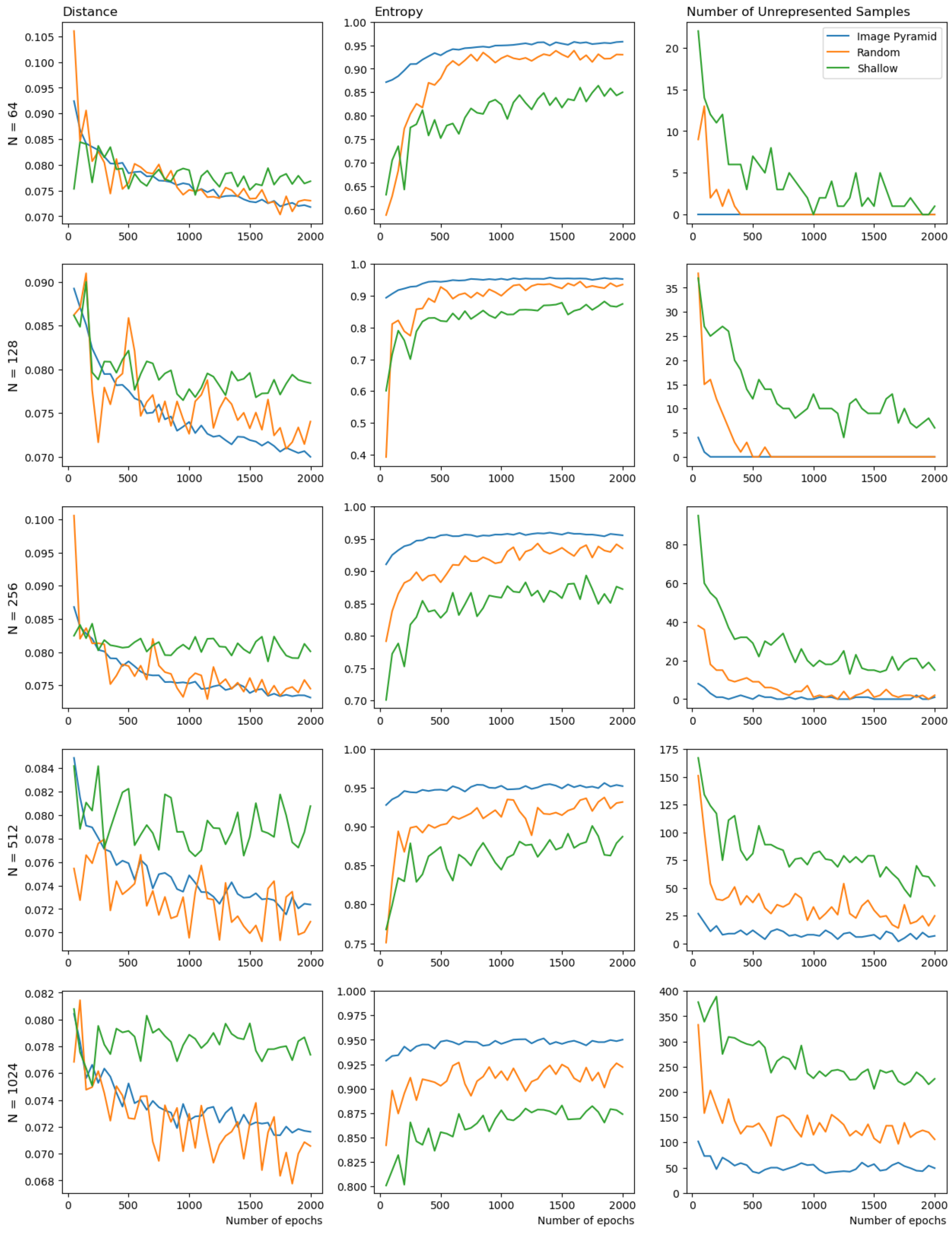

Figure 5.5: Measures for the three proposed machines regarding Mean Distance, Entropy, and Number of Unrepresented Samples on the first, second and third column respectively, with the number of train set samples N increasing on each row. The multi-level based approach showed better results on both Entropy and Number of Unrepresented Samples for all N values, while the shallow network performed worse on all measures for all experiments.

The Image Pyramid machine was able to have a higher entropy for all N values followed by the deep machine with Random initialization and lastly the shallow one, suggesting our approach is able to gather a better generalization of the world's distribution. The mean distance is similar in both deep architectures and consistently better than the shallow machine for all N values.

The poor performance of the shallow network with a higher number of free parameters indicates that the compositional properties of multi-layer can help a generative model capture the real world's distribution better.

The edge that machine (a) had over machine (b) when representing its training dataset suggests that using multi-level data representation provides advantages to the generative capabilities.

5.4 Can we quantify the generative advantage of Multi-level based Initialization?

From the previous experiment, we saw that the HM was able to replicate more samples on a given dataset when using our proposed initialization, which is a good indicator that the machine can understand the world's general distribution. However, we think mimicking a train set is not the goal of a generative model.

Testing a Generative model's performance is not a trivial task [87], and up to this date, there is no perfect definition of what can be considered a good generation, since different problems focus on different generative goals. Therefore, there is also no evaluation method devoid of criticism [88]. With this in mind, we decided to enumerate what we thought were the desired attributes our machine's generation required, in this particular experiment, with regards to handwritten digits' image generation. The most important attribute was the quality of the generated samples, more specifically, how similar the generated patterns were to real handwritten digits. The second attribute was diversity in the generated samples. And lastly, the propensity of generating new patterns. This last attribute might seem counter-intuitive, but the idea is that if our model produces completely different images of a digit that it did not see in the train set, but follow its digits general rules (eg. for number 8 two circles attached vertically), then our model effectively learned the core defining features of a digit.

5.4.1 Proposed Experiment:

We decided to take a common approach of tweaking the original generative model so that it can be used as a classification one. With classification, the model's performance becomes much easier to quantify, since we can get concrete measures such as error and accuracy. The fact that our machine learns with unlabelled data makes it hard for our model to be used to classify digits, so we decided to create

10 different machines, one for each digit. Hopefully, each machine's generation corresponds solely to good representations of its designated digit (what we call good quality) and produce a wide variety of that digit's possible representation (what we call variety). Then, we generate a fixed number of samples from each of the ten machines, and we end up with an entirely new generated dataset by combining all samples. After we create the new dataset, we can associate labels to the generated patterns, since each sample is associated with a certain digit's machine. Now, we can use a simple classification model like a K-Nearest Neighbor (KNN) trained with the generated dataset to classify the test set. We decided to use a KNN with $k = 1$ because of its simplicity. We believe it is a good choice because the score of the KNN's performance is purely related to the quality of the dataset, and our ultimate goal is not to create the best possible classifier but to test the quality of the generated dataset. If our machines can produce a wide range of variations of its designated digit, we should end up with a dataset that is able to produce the possible digit variations existent on the test set, and thus allow the KNN to have better accuracy during the test phase. We believe this evaluation method favors models that have the three requirements previously enumerated, but we can see scenarios where solutions that do not meet all the requirements still perform significantly well. For example, a machine generates the pixel distribution of a certain digit's class (similar to performing a mean of all the digit's samples), despite not having any variability, the KNN would still in most cases associate a test sample to the correct machine. To ensure variety amongst the generated samples we decided to measure the average euclidean distance to the mean of the newly generated dataset (ADM) similar to variance in a standard deviation, and to ensure that the generated instances are different from the training dataset we calculated another indicator called Novelty, that is obtained through the sum of the smallest distance of each Train Set sample to the generated ones.

Lastly, we performed said experiment on three different initialization methods, Zero Initialization that assigns all initial weight values to 0, Random Initialization that uses random values across a standard deviation centered on 0, and the Image Pyramid Initialization proposed.

5.4.2 Overall results:

The accuracy obtained with different network architectures and with different weight initialization methods described in Fig. 5.6, shows a clear advantage for initialization based on multi-level data representations, having not only a better average score but also smaller variance. We believe these results indicate that the Image Pyramid Initialization guides the networks' learning in a more robust way, by starting in an area of the energy surface where the local minima reached are generally better.

When looking at the variability measures for the same run of experiences in Fig. 5.7, the results show that our proposed method is able to generate samples more different from each other, and also samples less similar to the training set. This latter factor is a very promising indicator when combined with the previous observation that the accuracy performance also increased. This could indicate that the

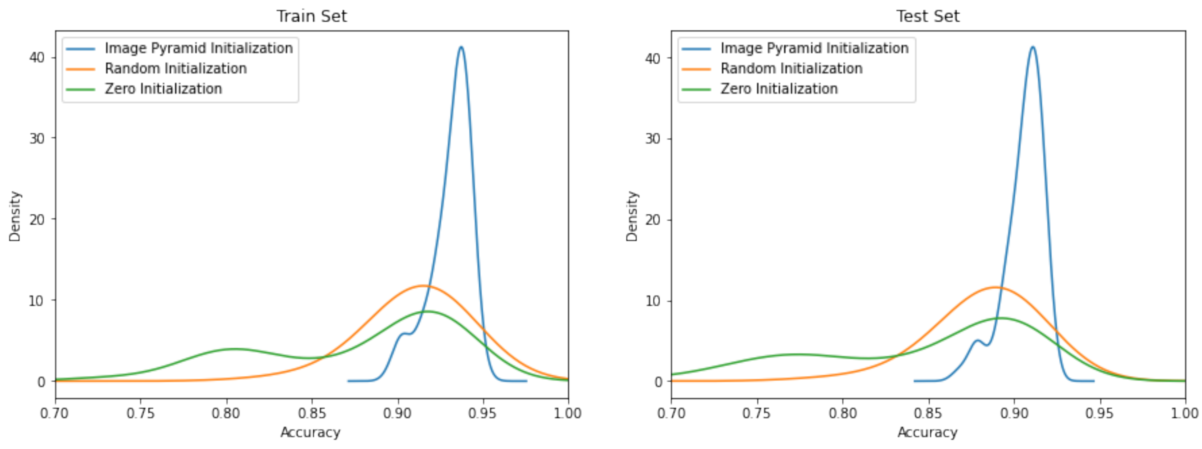

Figure 5.6: Density function for the train and test set accuracies of 70 randomly generated architectures trained with Random, Zero, and Image Pyramid Weight Initialization. Multi-level Representation Initialization shows not only a higher overall accuracy but also a lower variety, showing not only better performance but also more robustness.

model not only was able to produce new images, but those images are viable candidates for existing in the real world, possibly very similar to the unseen samples in the Test Set.

5.4.3 Results regarding dimensionality:

When we plot the accuracy with regards to the complexity of the model (either by the number of neurons or the number of hidden layers) in Fig. 5.8 we can see a clear decrease in the accuracy for the random and zero initialization, whereas the multi-level representation initialization remains consistent with the complexity increase. We believe these results show a clear inability for normal HM to perform well on deeper networks, which can easily be fixed by adding multi-level image representations to its learning. However, we expected the Image Pyramid Initialization to see an increase in accuracy with the increase of the number of free parameters, which did not happen on a meaningful scale. One of the reasons for this occurrence might be related to the small complexity of the dataset. We initially hypothesized that more free parameters were necessary for describing more complex worlds, however, if the world's complexity is small, there is no need for more free parameters. The MNIST dataset of handwritten digits is known to have relatively low complexity, and thus, as we surpass the required number of free parameters, we should not see any meaningful increases in the model's performance.

5.5 Can a Helmholtz Machine transcend the Train Set?

Encourage by the results mentioned in section 5.4.2, where we stated that the Helmholtz Machine using the Multi-level heuristic was able to produce a wider variety and more creative samples, we decided to

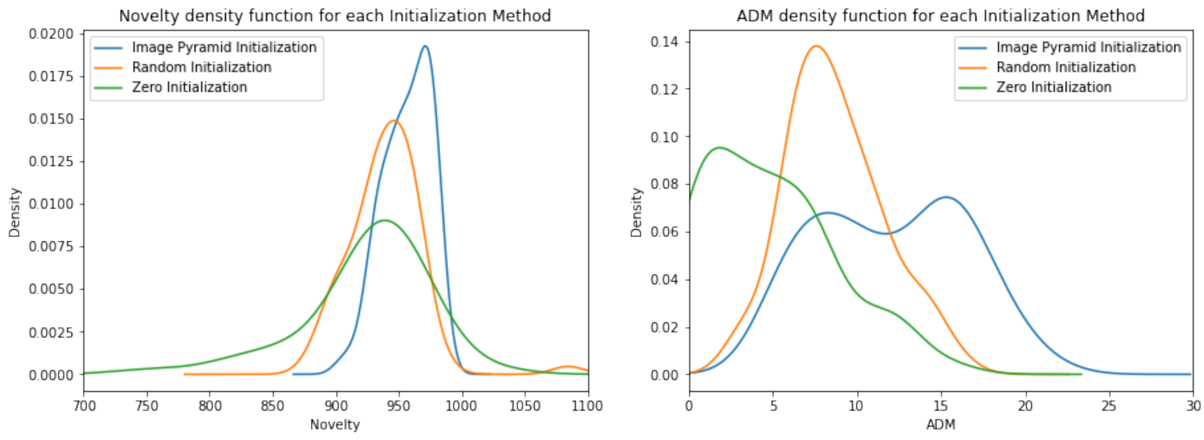

Figure 5.7: Density function of two different variability measures. Novelty relates to the difference between the generated data samples to the training samples, whereas ADM relates to variability among the generated samples. Our proposed Initialization method generated not only instances that were more different amongst themselves, but also less similar to the observed ones.

test if a HM was able to generate a dataset that could surpass the KNN performance of its original train set. We believe that after learning all digits, if a human spent an enormous amount of time generating labeled digits variations, eventually, a simple KNN using that generated data could almost perfectly classify the unseen MNIST test set. Likewise, if a HM produced enough samples it could cover a wider range of possible test set instances.

5.5.1 Proposed Experiment:

To test this hypothesis, we decided to train three 10-machine's models with the same architecture and each initialization method used previously. The architecture chosen was a 2 hidden layer network with layer sizes 400 (20×20) and 100 (10×10), while still having a visible layer of size 784 (28×28). We defined a Train Set of 10000 samples from the MNIST dataset and tested the performance of the chosen Train Set on the Test Set defining a threshold for our model to try to surpass. After we trained the machines with the chosen Train Set, we decided to calculate the accuracy of a KNN using datasets generated by the 10-machine model, and see what would happen with the increase of the size of the generated data, previously denoted as G.

5.5.2 Results:

From the results in Fig. 5.9 we can see that only the model trained with our multi-level heuristic was able to reach the same accuracy as the original Train Set, even surpassing its performance by a small margin, which strongly indicates that the model can generate samples more similar to the test set than the ones existing on the Train Set.

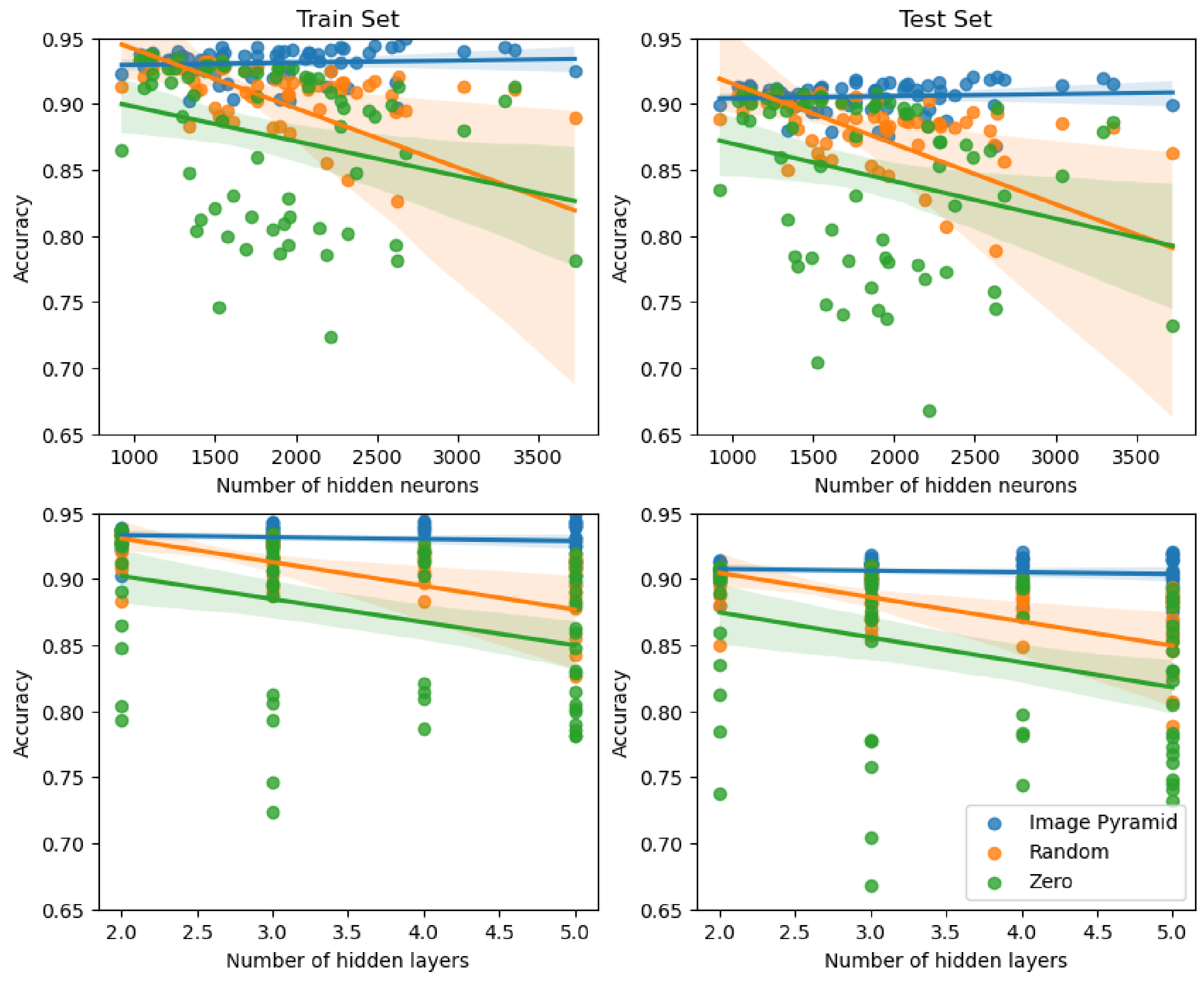

Figure 5.8: Scatter Plot with a Linear Regression Fit of the accuracy for Random, Zero, and Image Pyramid Initialization both on the Train and Test Set, plotted with regards to the number of neurons on the bottom axis on the first row, and to the number of hidden layers on the second one. Both Random and Zero Initialization show a progressive accuracy drop with the increase of the network's size, while the Image Pyramid's accuracy remains unchanged.

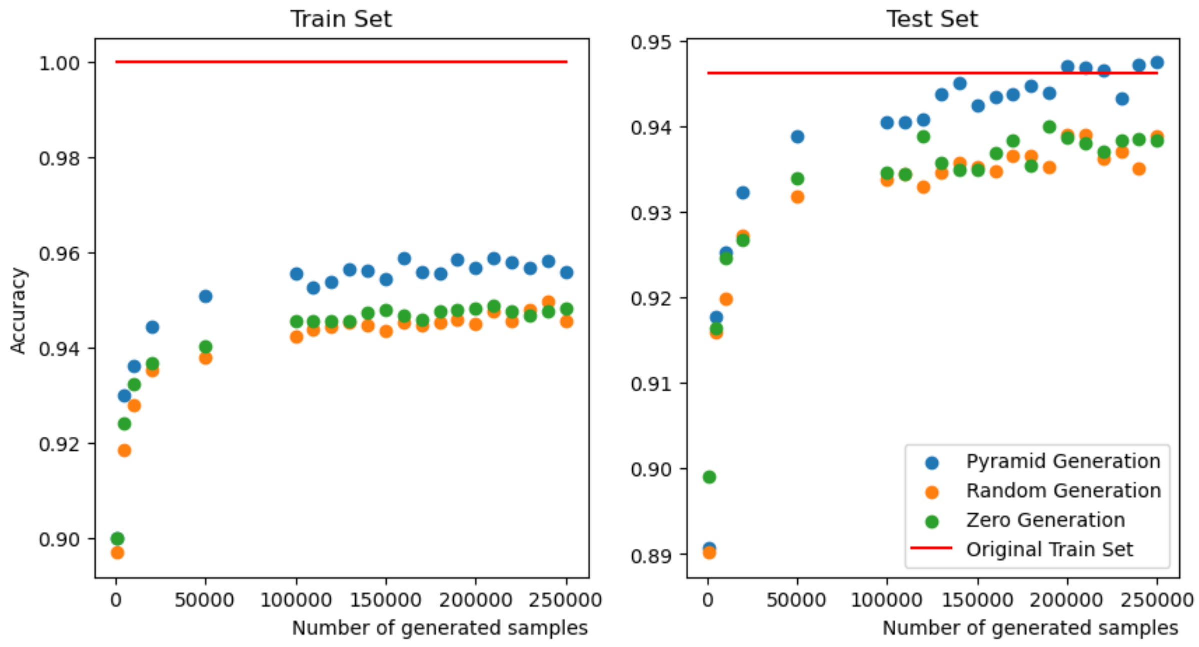

Figure 5.9: Scatter plot of the accuracy in the Train and the Test Set of a 1NN classifier using datasets generated by the HM models with three different initialization algorithms, plotted with regards to its number of samples. The blue horizontal line corresponds to the accuracy of a 1NN classifier using the same Train set the HM models used for training. We can see that only the model using the Image Pyramid initialization was able to surpass that horizontal line.

5.6 Is the Generative Advantage still present on more complex Datasets?

To conclude our experiments we believe it is important to understand if our heuristic still provides advantages in different and more complex domains. We decided to use the Fashion-MNIST and the CIFAR-10 datasets for this purpose. In the CIFAR-10, there are three color channels, and although it is possible to create a HM architecture to address this, we believe that changing the architecture of the model for this particular experiment would ravel the comparison to the other datasets, so we decided to change the RGB triplet of the images in CIFAR-10 to a grayscale, allowing us to have similar HM architectures for all domains.

5.6.1 Proposed Experiment:

We decided to perform a similar experiment as we did in section 5.4 using the 10-machine model to generate a dataset followed by a KNN using the generated dataset to classify the Test Set. Since the classification task is generally harder as we increase the complexity of the world, the overall accuracy values obtained will decrease as the complexity of the world grows. We believe the fairest comparison measure would not be the total KNN accuracy, but the accuracy improvement using the generated dataset, compared to the accuracy obtained using the Train Set. So the proposed metric to compare the results would be the Accuracy Improvement Factor, obtained by dividing the accuracy of the KNN using the generated dataset by the accuracy of the KNN using the original Train Set.

5.6.2 Results:

Table 5.1: Accuracy Improvement Factor on the Test Set for the three Initialization methods in each row, on different Datasets in each column. There is a clear advantage when using the Multi-level approach compared to the more classical implementations at all levels of domain complexity.

	MNIST	Fashion-MNIST	CIFAR-10
Image Pyramid	0.970	0.960	1.196
Random	0.938	0.929	1.042
Zero	0.905	0.909	0.872

From the results in Table 5.1, we can see that across all three datasets, each one with a different complexity, the performance of the model that is using the proposed heuristic was higher, which suggests that the advantages observed in the simpler domain of the MNIST dataset of handwritten digits are also present when the complexity of the world increases.

5.6.3 Visual analysis:

Through human visual analysis, in the CIFAR-10 dataset, although in some cases the Random Initialization produced similar results to the Image Pyramid generation, there were a lot of cases there was a noticeable quality difference favoring our Multi-level implementation, like the one shown in Fig. 5.10.

In the Fashion-MNIST dataset generation shown in Fig 5.11, the quality difference is not so noticeable, but there are still some less evident improvements, namely, the slight blurriness present in the Zero Initialization (more evident in the Ankle Boot class), and the disturbance in the Pullover texture (similar to Gaussian noise) present in the Random Initialization, which are not present in the Multi-level Initialization generated images.

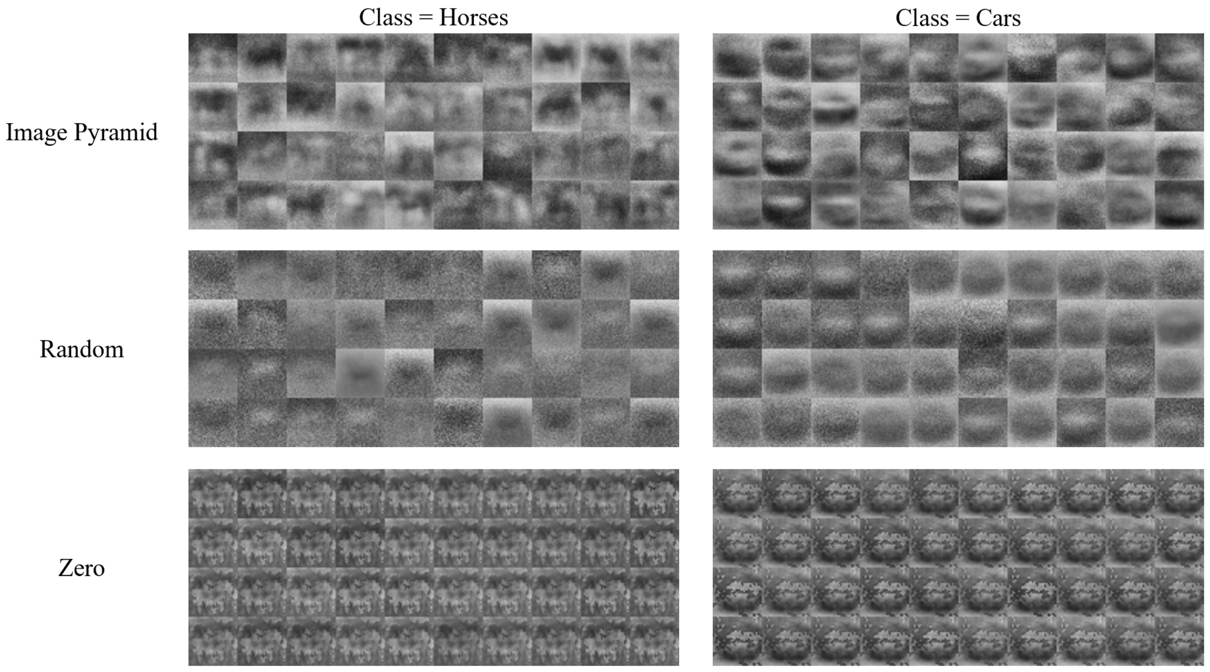

Figure 5.10: Generation of 40 samples by machines with similar architectures trained with different classes and different Initialization Methods on the CIFAR-10 dataset. In this particular example, despite all machines having the same architecture, our visual inspection of these two particular classes shows clear advantages for the generation model using the Multi-level heuristic.

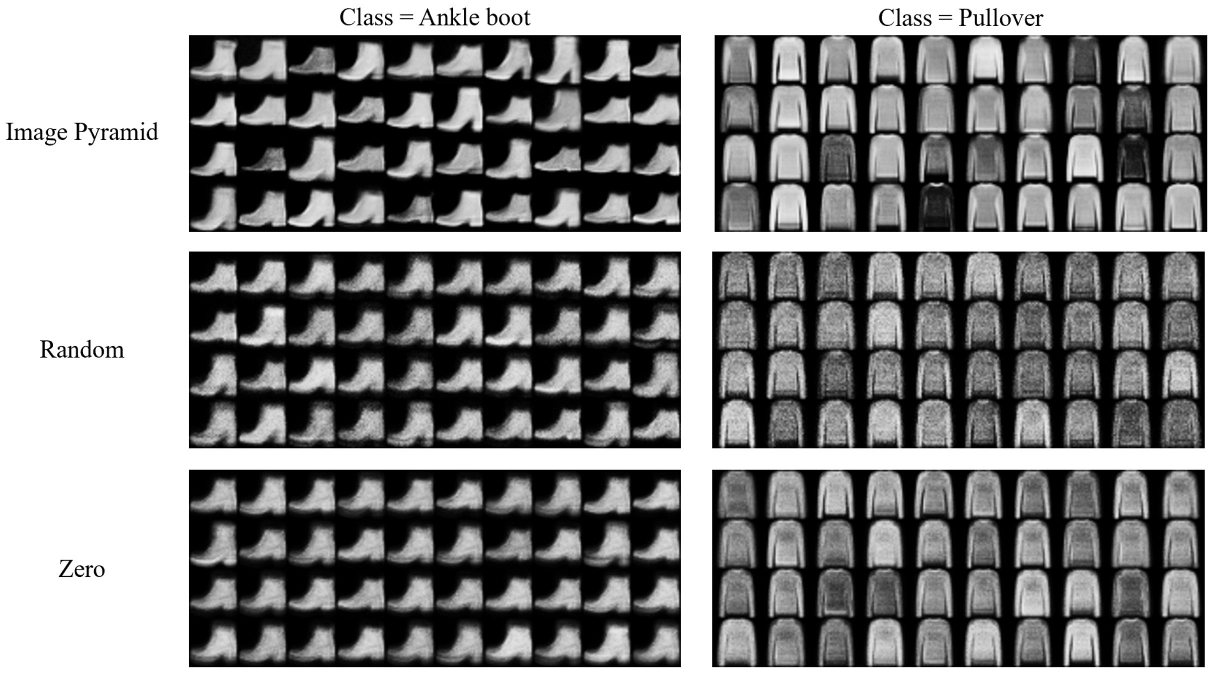

Figure 5.11: Generation of 40 samples by machines with similar architectures trained with different classes and different Initialization Methods on the Fashion-MNIST dataset. We believe the quality improvement using the Multi-level heuristic is not as accentuated as in the CIFAR-10 generation but is still present since Zero Initialization shows a higher level of blurriness in the boots generations and the Random Initialization shows less continuity (similar to Gaussian noise) in the texture of the pullovers generated.

6 Conclusion

Contents

6.1 Conclusions and Future Work

Despite the undeniable success of global Gradient-based algorithms, to understand the underlying mechanisms of biological intelligence it is necessary to develop models whose implementation could be plausible in a biological neural network. We believe a great candidate for said implementation is the Helmholtz Machine, due to the biological inspiration and locality of its training algorithm.

We hypothesize that, unlike gradient algorithms such as Back-propagation, this local learning algorithm does not perform well under deep network architectures, making it difficult to take advantage of the composition properties that multi-layer networks provide.

To test this hypothesis, we trained a HM on the MNIST dataset of Handwritten digits and used a linear classifier trained with a subset of the hidden representations of the HM's recognition model. The results were compliant with our hypothesis and showed that most of the separation was concentrated in the first layers, whereas deeper smaller layers had almost no relevant information regarding the problem space.

To avoid this limitation of the learning algorithm, we came up with a heuristic for the initialization of the machine's weight vectors by using a multi-level data representation based on the idea that humans process visual inquiries at different resolution levels. By iteratively training the smaller layers with down-sampled images of the real dataset, and increasing the resolution as we increase the size of the new layers, we believe that the information regarding high abstract levels of representation is rendered into each layer, obtaining the core defining features of the correspondent resolution.

Using our proposed solution, and repeating the previous experiment we saw that the separation of the problem space was done uniformly throughout the deep network, suggesting a better usage of the network as a whole in the recognition model of the HM.

We then saw that the model's improvement was also imminent in the generative model, showing that our proposed solution was able to take advantage of the compositional properties that multi-layer networks allow for, being able to replicate a more complex world than a one-layer network with a higher number of free parameters and an equal architecture trained conventionally.

We performed further experiments to test the generative advantages of adding multi-level information to the training of the neural network, by using the model's generated images to train a simple classifier and concluded that the machine's generated samples had not only better quality, since the classifier performed generally better when using our approach, but were also more diverse and creative when compared to the classic implementation.

From the latter experience, we were able to find further evidence that supports our claim that the WS algorithm is not fit for Deep Networks, showing a progressive decrease in performance as the network's depth increases when using classical implementation. The same occurrence was not imminent when using our proposed initialization step.

Encouraged by the creativity measures of our model, we decided to see if it was able to produce a dataset for training a classifier that would outperform an equal classifier that learned on the HM's initial training set. We saw that a such thing was possible when generating a large number of data samples but only with our proposed multi-level heuristic.

Finally, we detected that the Multi-level heuristic still provides generative advantages in more complex domains, by performing similar experiments on different datasets.

The results obtained were very promising and showed the innate potential of the Helmholtz Machine's generative model's capabilities. Moreover, we believe the general heuristic idea can be applied to other local learning algorithms training on two-dimensional data with similar success.

We compiled the experiments and results obtained in this thesis into a paper that can be accessed at `https://arxiv.org/abs/2210.14855` that will later be submitted to a scientific journal.

Bibliography

[1] J. Hertz, A. Krogh, and R. G. Palmer, *Introduction to the Theory of Neural Computation*. Addison-Wesley Longman Publishing Co., Inc., 1991.

[2] J. C. Whittington and R. Bogacz, “Theories of error back-propagation in the brain,” *Trends in Cognitive Sciences*, vol. 23, pp. 235–250, 2019.

[3] Y. LeCun, Y. Bengio, and G. Hinton, “Deep learning,” *nature*, vol. 521, pp. 436–444, 2015.

[4] S. Bartunov, A. Santoro, B. A. Richards, L. Marris, G. E. Hinton, and T. Lillicrap, “Assessing the scalability of biologically-motivated deep learning algorithms and architectures,” *arXiv Prepr.*, 2018.

[5] A. H. Marblestone, G. Wayne, and K. P. Kording, “Toward an integration of deep learning and neuroscience,” *Frontiers in Computational Neuroscience*, vol. 10, 2016.

[6] T. P. Lillicrap, A. Santoro, L. Marris, C. J. Akerman, and G. Hinton, “Backpropagation and the brain,” *Nature Reviews Neuroscience*, vol. 21, pp. 335–346, 2020.

[7] I. Sutskever, J. Martens, G. Dahl, and G. Hinton, “On the importance of initialization and momentum in deep learning,” in *Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28*, ser. ICML’13. JMLR.org, 2013, p. III–1139–III–1147.

[8] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT press, 2016.

[9] S. Ahlawat, A. Choudhary, A. Nayyar, S. Singh, and B. Yoon, “Improved handwritten digit recognition using convolutional neural networks (cnn),” *Sensors*, vol. 20, 2020.

[10] C. J. B. Yann LeCun, Corinna Cortes, “The mnist database of handwritten digits.” [Online]. Available: http://yann.lecun.com/exdb/mnist/

[11] D. Ciregan, U. Meier, and J. Schmidhuber, “Multi-column deep neural networks for image classification,” in *2012 IEEE Conference on Computer Vision and Pattern Recognition*, 2012, pp. 3642–3649.

[12] D. Krotov and J. J. Hopfield, “Unsupervised learning by competing hidden units,” *Proceedings of the National Academy of Sciences*, vol. 116, pp. 7723–7731, 2019.

[13] B. Illing, W. Gerstner, and J. Brea, “Biologically plausible deep learning — but how far can we go with shallow networks?” *Neural Networks*, vol. 118, pp. 90–101, 2019.

[14] Y. Bengio, D.-H. Lee, J. Bornschein, T. Mesnard, and Z. Lin, “Towards biologically plausible deep learning,” *arXiv Prepr.*, 2015.

[15] N. B. Ravichandran, A. Lansner, and P. Herman, “Learning representations in bayesian confidence propagation neural networks,” in *2020 International Joint Conference on Neural Networks (IJCNN)*, 2020, pp. 1–7.

[16] L. Sa-Couto and A. Wichert, “Attention inspired network: Steep learning curve in an invariant pattern recognition model,” *Neural Networks*, vol. 114, pp. 38–46, 2019.

[17] W. S. McCulloch and W. Pitts, “A logical calculus of the ideas immanent in nervous activity,” *The bulletin of mathematical biophysics*, vol. 5, pp. 115–133, 1943.

[18] F. Crick, “The recent excitement about neural networks,” *Nature*, vol. 337, no. 6203, p. 129—132, 1989.

[19] D. G. Stork, “Is backpropagation biologically plausible,” in *International Joint Conference on Neural Networks*, vol. 2. IEEE Washington, DC, 1989, pp. 241–246.

[20] T. P. Lillicrap, D. Cownden, D. B. Tweed, and C. J. Akerman, “Random synaptic feedback weights support error backpropagation for deep learning,” *Nature communications*, vol. 7, pp. 1–10, 2016.

[21] G. E. Hinton, P. Dayan, B. J. Frey, and R. M. Neal, “The ”wake-sleep” algorithm for unsupervised neural networks,” *Science*, vol. 268, pp. 1158–1161, 1995.

[22] R. M. Neal and P. Dayan, “Factor analysis using delta-rule wake-sleep learning,” *Neural computation*, vol. 9, pp. 1781–1803, 1997.

[23] D. O. Hebb, “The organization of behavior: A neuropsychological theory. new york: John wiley and sons,” *Science Education*, vol. 34, pp. 336–337, 1949.

[24] R. L. Sumner, M. J. Spriggs, S. D. Muthukumaraswamy, and I. J. Kirk, “The role of hebbian learning in human perception: a methodological and theoretical review of the human visual long-term potentiation paradigm,” *Neuroscience & Biobehavioral Reviews*, vol. 115, pp. 220–237, 2020.

[25] K. Friston, “A theory of cortical responses,” *Philosophical transactions of the Royal Society B: Biological sciences*, vol. 360, pp. 815–836, 2005.

[26] K. Friston, J. Kilner, and L. Harrison, “A free energy principle for the brain,” *Journal of Physiology-Paris*, vol. 100, pp. 70–87, 2006.

[27] K. Friston, “The free-energy principle: a rough guide to the brain?” *Trends in cognitive sciences*, vol. 13, pp. 293–301, 2009.

[28] ——, “A free energy principle for biological systems,” *Entropy*, vol. 14, pp. 2100–2121, 2012.

[29] G. Palm, “Neural associative memories and sparse coding,” *Neural Networks*, vol. 37, pp. 165–171, 2013.

[30] J. R. Pelaez, “Plato’s theory of ideas revisited,” *Neural Networks*, vol. 10, no. 7, pp. 1269–1288, 1997.

[31] J. J. Hopfield, “Neural networks and physical systems with emergent collective computational abilities,” *Proceedings of the National Academy of Sciences*, vol. 79, no. 8, pp. 2554–2558, 1982.

[32] ——, “Neurons with graded response have collective computational properties like those of two-state neurons,” *Proceedings of the National Academy of Sciences*, vol. 81, no. 10, pp. 3088–3092, 1984.

[33] J. Šíma and P. Orponen, “Continuous-time symmetric hopfield nets are computationally universal,” *Neural Computation*, vol. 15, no. 3, pp. 693–733, 2003.

[34] J. K. Paik and A. K. Katsaggelos, “Image restoration using a modified hopfield network,” *IEEE Transactions on Image Processing*, vol. 1, no. 1, pp. 49–63, 1992.

[35] J. Milnor, “On the concept of attractor,” in *The theory of chaotic attractors*. Springer, 1985, pp. 243–264.

[36] A. Barra, A. Bernacchia, E. Santucci, and P. Contucci, “On the equivalence of hopfield networks and boltzmann machines,” *Neural Networks*, vol. 34, pp. 1–9, 2012.

[37] G. A. Carpenter, “Neural network models for pattern recognition and associative memory,” *Neural networks*, vol. 2, no. 4, pp. 243–257, 1989.

[38] K. Takasaki, “Critical capacity of hopfield networks,” *Department of Physics, MIT*, 2007.

[39] Y. Abu-Mostafa and J. St. Jacques, “Information capacity of the hopfield model,” *IEEE Transactions on Information Theory*, vol. 31, no. 4, pp. 461–464, 1985.

[40] A. Storkey, “Increasing the capacity of a hopfield network without sacrificing functionality,” in *International Conference on Artificial Neural Networks*. Springer, 1997, pp. 451–456.

[41] Y. Wu, J. Hu, W. Wu, Y. Zhou, and K. Du, “Storage capacity of the hopfield network associative memory,” in *2012 Fifth International Conference on Intelligent Computation Technology and Automation*, 2012, pp. 330–336.

[42] H. S. Seung, “Continuous attractors and oculomotor control,” *Neural Networks*, vol. 11, no. 7-8, pp. 1253–1258, 1998.

[43] M. A. Cohen and S. Grossberg, “Absolute stability of global pattern formation and parallel memory storage by competitive neural networks,” *IEEE Transactions on Systems, Man, and Cybernetics*, vol. SMC-13, no. 5, pp. 815–826, 1983.

[44] Y. Crama, P. Hansen, and B. Jaumard, “Detection of spurious states of neural networks,” *IEEE Transactions on Neural Networks*, vol. 2, no. 1, pp. 165–168, 1991.

[45] D. J. Amit, H. Gutfreund, and H. Sompolinsky, “Storing infinite numbers of patterns in a spin-glass model of neural networks,” *Physical Review Letters*, vol. 55, pp. 1530–1533, Sep 1985.

[46] B. A. Cipra, “An introduction to the ising model,” *The American Mathematical Monthly*, vol. 94, no. 10, pp. 937–959, 1987.

[47] D. J. Amit, H. Gutfreund, and H. Sompolinsky, “Spin-glass models of neural networks,” *Physical Review A*, vol. 32, no. 2, p. 1007, 1985.

[48] R. J. Glauber, “Photon correlations,” *Physical Review Letters*, vol. 10, no. 3, p. 84, 1963.

[49] D. H. Ackley, G. E. Hinton, and T. J. Sejnowski, “A learning algorithm for boltzmann machines,” *Cognitive science*, vol. 9, no. 1, pp. 147–169, 1985.

[50] C. Marzban and R. Viswanathan, “Stochastic neural networks with the weighted hebb rule,” *Physics Letters A*, vol. 191, no. 1-2, pp. 127–133, 1994.

[51] A. M. Wichert, *Principles of Quantum Artificial Intelligence: Quantum Problem Solving and Machine Learning*. World scientific, 2020.

[52] P. Smolensky, “Information processing in dynamical systems: Foundations of harmony theory,” Colorado Univ at Boulder Dept of Computer Science, Tech. Rep., 1986.

[53] R. Salakhutdinov, A. Mnih, and G. Hinton, “Restricted boltzmann machines for collaborative filtering,” in *Proceedings of the 24th international conference on Machine learning*, 2007, pp. 791–798.

[54] A. Meyder and C. Kiderlen, “Fundamental properties of hopfield networks and boltzmann machines for associative memories,” *Machine Learning, vt*, 2008.

[55] G. E. Hinton, “Boltzmann machine,” *Scholarpedia*, vol. 2, no. 5, p. 1668, 2007.

[56] R. Salakhutdinov and G. Hinton, “Deep boltzmann machines,” in *Artificial intelligence and statistics*. PMLR, 2009, pp. 448–455.

[57] G. E. Hinton, “Learning multiple layers of representation,” *Trends in cognitive sciences*, vol. 11, no. 10, pp. 428–434, 2007.

[58] K. G. Kirby, “A tutorial on helmholtz machines,” *Department of Computer Science, Northern Kentucky University*, 2006. [Online]. Available: https://www.nku.edu/~kirby/docs/HelmholtzTutorialKoeln.pdf

[59] T. J. van Dam, N. M. Neumann, F. Phillipson, and H. van den Berg, “Hybrid helmholtz machines: a gate-based quantum circuit implementation,” *Quantum Information Processing*, vol. 19, no. 6, pp. 1–14, 2020.

[60] C. E. Shannon, “A mathematical theory of communication,” *The Bell System Technical Journal*, vol. 27, no. 3, pp. 379–423, 1948.

[61] S. Kullback, *Information theory and statistics*. Courier Corporation, 1997.

[62] P. J. Liu, M. Saleh, E. Pot, B. Goodrich, R. Sepassi, L. Kaiser, and N. Shazeer, “Generating wikipedia by summarizing long sequences,” *arXiv Prepr.*, 2018.

[63] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, “Language models are unsupervised multitask learners,” *OpenAI blog*, vol. 1, no. 8, p. 9, 2019.

[64] Y. Tang, R. Salakhutdinov, and G. Hinton, “Robust boltzmann machines for recognition and denoising,” in *2012 IEEE Conference on Computer Vision and Pattern Recognition*, 2012, pp. 2264–2271.

[65] R. A. Yeh, C. Chen, T. Yian Lim, A. G. Schwing, M. Hasegawa-Johnson, and M. N. Do, “Semantic image inpainting with deep generative models,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

[66] J. Kivinen and C. Williams, “Multiple texture boltzmann machines,” in *Proceedings of the Fifteenth International Conference on Artificial Intelligence and Statistics*, ser. Proceedings of Machine Learning Research, vol. 22. PMLR, 4 2012, pp. 638–646.

[67] T. Karras, S. Laine, M. Aittala, J. Hellsten, J. Lehtinen, and T. Aila, “Analyzing and improving the image quality of stylegan,” in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 8110–8119.

[68] H. Larochelle, M. Mandel, R. Pascanu, and Y. Bengio, “Learning algorithms for the classification restricted boltzmann machine,” *The Journal of Machine Learning Research*, vol. 13, pp. 643–669, 2012.

[69] D. Roth, “On the hardness of approximate reasoning,” *Artificial Intelligence*, vol. 82, no. 1-2, pp. 273–302, 1996.

[70] J. Bornschein, S. Shabanian, A. Fischer, and Y. Bengio, “Bidirectional helmholtz machines,” in *Proceedings of The 33rd International Conference on Machine Learning*, vol. 48. PMLR, 2016, pp. 2511–2519. [Online]. Available: https://proceedings.mlr.press/v48/bornschein16.html

[71] G. E. Hinton, S. Osindero, and Y.-W. Teh, “A fast learning algorithm for deep belief nets,” *Neural computation*, vol. 18, no. 7, pp. 1527–1554, 2006.

[72] M. Campisi, “On the mechanical foundations of thermodynamics: The generalized helmholtz theorem,” *Studies in History and Philosophy of Science Part B: Studies in History and Philosophy of Modern Physics*, vol. 36, no. 2, pp. 275–290, 2005.

[73] J. An and S. Cho, “Variational autoencoder based anomaly detection using reconstruction probability,” *Special Lecture on IE*, vol. 2, no. 1, pp. 1–18, 2015.

[74] T. Isomura and K. Friston, “In vitro neural networks minimise variational free energy,” *Scientific reports*, vol. 8, no. 1, pp. 1–14, 2018.

[75] D. P. Kingma, M. Welling *et al.*, “An introduction to variational autoencoders,” *Foundations and Trends® in Machine Learning*, vol. 12, pp. 307–392, 2019.

[76] J. Bornschein and Y. Bengio, “Reweighted wake-sleep,” *arXiv Prepr.*, 2014.

[77] H. R. Wilson and J. R. Bergen, “A four mechanism model for threshold spatial vision,” *Vision Research*, vol. 19, pp. 19–32, 1979.

[78] F. W. Campbell and J. G. Robson, “Application of fourier analysis to the visibility of gratings,” *The Journal of physiology*, vol. 197, p. 551, 1968.

[79] A. Rosenfeld, “Some useful properties of pyramids,” in *Multiresolution Image Processing and Analysis*, 1984, pp. 2–5.

[80] E. H. Adelson, C. H. Anderson, J. R. Bergen, P. J. Burt, and J. M. Ogden, “Pyramid methods in image processing,” *RCA engineer*, vol. 29, pp. 33–41, 1984.

[81] X. Glorot and Y. Bengio, “Understanding the difficulty of training deep feedforward neural networks,” in *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, vol. 9, 2010, pp. 249–256. [Online]. Available: https://proceedings.mlr.press/v9/glorot10a.html

[82] A. Wichert, “Content-based image retrieval by hierarchical linear subspace method,” *Journal of Intelligent Information Systems*, vol. 31, no. 1, pp. 85–107, 2008.

[83] A. Wichert and C. Moreira, “On projection based operators in lp space for exact similarity search,” *Fundamenta Informaticae*, vol. 136, no. 4, pp. 461–474, 2015.

[84] A. Wichert, P. Teixeira, P. Santos, and H. Galhardas, “Subspace tree: High dimensional multimedia indexing with logarithmic temporal complexity,” *Journal of Intelligent Information Systems*, vol. 35, no. 3, pp. 495–516, 2010.

[85] H. Xiao, K. Rasul, and R. Vollgraf, “Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms,” 2017.

[86] A. Krizhevsky, G. Hinton *et al.*, “Learning multiple layers of features from tiny images,” 2009. [Online]. Available: https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf

[87] A. Alaa, B. Van Breugel, E. S. Saveliev, and M. van der Schaar, “How faithful is your synthetic data? Sample-level metrics for evaluating and auditing generative models,” in *Proceedings of the 39th International Conference on Machine Learning*, vol. 162, 2022, pp. 290–306. [Online]. Available: https://proceedings.mlr.press/v162/alaa22a.html

[88] L. Theis, A. v. d. Oord, and M. Bethge, “A note on the evaluation of generative models,” 2015.

A

Appendix A

A.1 Kullback-Leibler divergence of explanations from p_R to the generative distribution p_G given a fixed pattern d

$$
\begin{aligned}
&\mathrm{KL}\left[p_R(\mathbf{XY} \mid \mathbf{d}), p_G(\mathbf{XY} \mid \mathbf{d})\right] \equiv \textstyle\sum_{\mathbf{xy}} p_R(\mathbf{xy} \mid \mathbf{d}) \log \frac{p_R(\mathbf{xy}|\mathbf{d})}{p_G(\mathbf{xy}|\mathbf{d})} \\
&= \textstyle\sum_{\mathbf{xy}} p_R(\mathbf{xy} \mid \mathbf{d}) \log p_R(\mathbf{xy} \mid \mathbf{d}) - \sum_{\mathbf{xy}} p_R(\mathbf{xy} \mid \mathbf{d}) \log p_G(\mathbf{xy} \mid \mathbf{d}) \\
&= \textstyle\sum_{\mathbf{xy}} p_R(\mathbf{xy} \mid \mathbf{d}) \log p_R(\mathbf{xy} \mid \mathbf{d}) - \sum_{\mathbf{xy}} p_R(\mathbf{xy} \mid \mathbf{d}) \log \frac{p_G(\mathbf{xyd})}{p_G(\mathbf{d})} \\
&= -H_R(\mathbf{XY} \mid \mathbf{d}) - \textstyle\sum_{\mathbf{xy}} p_R(\mathbf{xy} \mid \mathbf{d}) \log p_G(\mathbf{xyd}) + \sum_{\mathbf{xy}} p_R(\mathbf{xy} \mid \mathbf{d}) \log p_G(\mathbf{d}) \\
&= -H_R(\mathbf{XY} \mid \mathbf{d}) + \textstyle\sum_{\mathbf{xy}} p_R(\mathbf{xy} \mid \mathbf{d}) E_G(\mathbf{xy}; \mathbf{d}) + \sum_{\mathbf{xy}} p_R(\mathbf{xy} \mid \mathbf{d}) \log p_G(\mathbf{d}) \\
\\
&= -H_R(\mathbf{XY} \mid \mathbf{d}) + \langle E_{\mathrm{G}}(\mathbf{XY}; \mathbf{d}) \rangle_R + \log p_G(\mathbf{d}) \textstyle\sum_{xy} \neq_k (\mathbf{x}y \mid \mathbf{d}) \\
\\
&= -H_R(\mathbf{XY} \mid \mathbf{d}) + \langle E_{\mathrm{G}}(\mathbf{XY}; \mathbf{d}) \rangle_R - F_G(\mathbf{d}).
\end{aligned}
$$

A.2 Calculation for the derivatives of all the connection weights matrices of the Generative Model

For matrix $\mathbf{b}^G$:

$$\begin{aligned}
\frac{\partial}{\partial b_k^G}\log p_G(\mathbf{x}) &= \frac{\partial}{\partial b_k^G}\sum_K x_K \log \xi_K + \frac{\partial}{\partial b_k^G}\sum_K (1-x_K)\log(1-\xi_K) \\
&= x_k - \xi_{k.} \\
\frac{\partial}{\partial b_k^G}\log p_G(\mathbf{y}\mid\mathbf{x}) &= 0 \\
\frac{\partial}{\partial b_k^G}\log p_G(\mathbf{d}\mid\mathbf{y}) &= 0
\end{aligned}$$

For matrix $\mathbf{W}^G$:

$$\begin{aligned}
&\tfrac{\partial}{\partial w_{jk}^G}\log p_G(\mathbf{x}) = 0 \\
&\tfrac{\partial}{\partial w_{jk}^G}\log p_G(\mathbf{y}\mid\mathbf{x}) = \begin{cases} (y_j-\psi_j)\,x_k, & k=1,2,\ldots,L \\ y_j-\psi_j, & k=L+1 \end{cases} \\
&\tfrac{\partial}{\partial w_{jk}^G}\log p_G(\mathbf{d}\mid\mathbf{y}) = 0
\end{aligned}$$

For matrix $\mathbf{V}^G$:

$$\begin{aligned}
&\tfrac{\partial}{\partial v_{ij}^G}\log p_G(\mathbf{x}) = 0 \\
&\tfrac{\partial}{\partial v_{ij}^G}\log p_G(\mathbf{y}\mid\mathbf{x}) = 0 \\
&\tfrac{\partial}{\partial v_{ij}^G}\log p_G(\mathbf{d}\mid\mathbf{y}) = \begin{cases} (d_i-\delta_i)\,y_j, & j=1,2,\ldots,M \\ d_i-\delta_i, & j=M+1 \end{cases}
\end{aligned}$$

A.3 Calculation for the derivatives of all the connection weights matrices of the Recognition Model

For $\mathbf{W}^R$:

$$\begin{aligned}
&\tfrac{\partial}{\partial w_{kj}^R}\log p_R(\mathbf{x}\mid\mathbf{y}) = \begin{cases} (x_k-\xi_k)\,x_j, & j=1,2,\ldots,M \\ x_k-\xi_k, & j=M+1 \end{cases} \\
&\tfrac{\partial}{\partial w_{kj}^R}\log p_R(\mathbf{y}\mid\mathbf{d}) = 0
\end{aligned}$$

For $\mathbf{V}^R$:

$$\begin{aligned}
&\tfrac{\partial}{\partial v_{ji}^R}\log p_R(\mathbf{x}\mid\mathbf{y}) = 0 \\
&\tfrac{\partial}{\partial v_{ji}^R}\log p_R(\mathbf{y}\mid\mathbf{d}) = \begin{cases} (y_j-\psi_j)\,d_i, & i=1,2,\ldots,N \\ y_j-\psi_j, & i=N+1 \end{cases}
\end{aligned}$$