VICTORIA UNIVERSITY OF WELLINGTON
Te Whare Wānanga o te Ūpoko o te Ika a Māui

School of Engineering and Computer Science
Te Kura Mātai Pūkaha, Pūrorohiko

PO Box 600
Wellington
New Zealand

Tel: +64 4 463 5341
Fax: +64 4 463 5045
Internet: office@ecs.vuw.ac.nz

Richer Restricted Boltzmann Machines

Max Godfrey

Supervisors: Marcus Frean

Submitted in partial fulfilment of the requirements for
Bachelor of Engineering with Honours.

Abstract

Abstract. The abstract is your four sentence summary of the conclusions of your paper. Its primary purpose is to get your paper into the A pile. Most PC members sort their papers in an A pile and a B pile by reading the abstracts. The A pile papers get smiling interest, the B pile papers are a chore to be slogged through. By keeping your abstract short and clear, you greatly enhance your chances of being in the A pile. I try to have four sentences in my abstract. The first states the problem. The second states why the problem is a problem. The third is my startling sentence. The fourth states the implication of my startling sentence. An abstract for this paper done in this style would be: TODO WORDING I or We????

Acknowledgments

I would like to thank my supervisor Marcus Frean for the constant support and mentorship throughout the year. Also for teaching me so much, and the opportunity to work on something so neat. I would also like to thank Stephen Marsland, co-inceptor of the idea for this project.

Contents

Figures

Chapter 1

Introduction

1.1 A Problem

Consider an image of a face. At least two *systems* are at play in the image of a face, the face itself and the illumination. Both faces and illumination are complex and can vary greatly, but they are fundamentally acting independently of each other. Yet they combine to form the image.

A form of Deep Neural Networks, Deep Belief Networks, have achieved state of the art performance in facial recognition [17, 21], but this is only possible with a large amount of training data. This suggests that there is a disparity between the task and these networks, as the networks do not explicitly attempt to represent these *systems* independently. This project takes steps toward representing complex *systems/causes* separately with the primary task of decomposing images (more generally, vector based data).

Separating faces and illumination is too challenging for this project, as there is no way to verify/evaluate that the new approach is working. The concept of a face without illumination cannot be visualised. Instead I will start with the modest task of working with images where the source images being combined are known.

1.2 Deep Belief Networks can achieve state of the art performance

Deep Belief networks (DBNs) are powerful models that have proven to achieve state of the art performance in tasks such as image classification [31, 12], dimensionality reduction [24], natural language recognition [10], document classification [22] and semantic analysis [19]. Despite a DBN's expressiveness, there is no way to extract independent sources, the model instead learns how to represent the complex combination. This complex combination of sources is inherently richer than the individual sources acting alone. The DBN may learn features that correspond to each source during it's training process, however the architecture or training algorithm make no attempt to enforce this. Consider the face and illumination example, with 1 000 possible faces and 1 000 possible illuminations. If we only observe a combination of faces and illuminations there is 1 000 000 possible combinations we need to represent. It is more succinct to treat faces and illumination as independent *systems*, which results in 2000 possible combinations.

DBNs are built of shallow networks called Restricted Boltzmann Machines (RBMs). Despite being building blocks for the DBN, RBMs can be used as models for practical tasks, for instance representing handwritten digits [6].

1.3 A Proposed Solution and Contributions

Frean and Marsland propose an alternative architecture to that of an RBM, the ORBM, which aims to encode two complex sources independently. Frean and Marsland also propose an algorithm to put this alternative architecture to use.

This project contributes:

$\mathcal{C}1$. The first articulation of the proposed architecture, including the context needed to understand it.

$\mathcal{C}2$. A Python implementation of the new architecture and algorithm.

$\mathcal{C}3$. A suite of graded evaluations/tests that explore how the architecture and source separation algorithm work in practice.

The evaluations will not address separating faces and illumination, instead the more general case of two images overlaying to form the data. The implications of this are compelling as applications could include removing noise from images or extracting images that are composed of many subjects. These are aspirational goals and this project will be performing tasks such as separating two handwritten digits composed on each other (see figure 1.1).

Figure 1.1: An example composition of a handwritten four and three, illustrating a non-trival task where the ground truth is known.

Chapter 2

Backgroud

As the proposed architecture and algorithm extend existing work on Probabilistic Graphical Models, a substantial amount of background is required culminating with the new approach being derived. A robust understanding of these concepts is required for this project to implement and design appropriate evaluations for the proposed architecture.

2.1 Generative Models

This project works with neural network based generative models. Generative models are a powerful way to model data. With generative models, we aim to learn a model that can both create the training data, represent it, and reconstruct it using the representation the model has learnt to create. Generative models can map input data from raw values to higher level features. Hinton [11], gave a compelling argument for higher level features in the context of generative models.

> "Consider, for example, a set of images of a dog. Latent variables such as the position, size, shape and color of the dog are a good way of explaining the complicated, higher-order correlations between the individual pixel intensities, and some of these latent variables are very good predictors of the class label."

Generative models model a distribution over a collection of binary variables X, where X is comprised of variables which can be observed or unobserved. The observed variables are referred to as `visible` variables or units (v). Conversely, unobserved variables correspond to the hidden units of the neural network (h). With these terms defined, the joint distribution that generative models model can be expressed as $P(X)$ where X is comprised of h, v. Collections of these units, are often referred to as 'patterns' or 'vectors' in that they are represented by a vector or pattern of bits. For instance in the context of an image, a visible pattern is the pixels of the image flattened into a one dimensional vector.

2.2 Directed PGMs

The new approach proposed in this project uses both the two classes of Probabilistic Graphical Models, Directed and Undirected. A Directed PGM, is a notation for factorising over a collection of stochastic variables. Given X, the variables in the generative model, and $parent_i$ the parent unit of x_i, the distribution over X in a directed PGM is defined by the following factorisation:

$$P(X) = \prod_i P(x_i | parent_i) \tag{2.1}$$

Connections between units in the Directed PGM express causation [26]. They provide an expressive way to represent a collection of related, stochastic variables. See figure 2.1 for the minimal example, where variable A is dependent on B. As a result this network is often referred to as a Belief Network or Bayesian Network. A Belief Network's causal dependencies are expressed as a conditional probability table also know as 'factor'. For example in figure 2.1 the probabilities of A being in a given state are dependent on B. The joint distribution over A and B, as in equation 2.1, can be expressed as:

$$P(A, B) = P(A|B)P(B)$$

2.2.1 Explaining Away in Directed PGMs

A common task in generative models is given a parent unit (a cause) is in some state, inferring the state of child variable. In directed PGMs this is trivial to calculate. The opposite task is also desirable [28], inferring the state of causes given the effect of that cause. It becomes problematic in Directed PGMs as the causal relationships give rise to the effect of 'Explaining Away'. The canonical example that exemplifies 'Explaining Away', `Burglar, Earthquake, Alarm Problem` [2] is illustrated in figure 2.2. Knowledge of the state of the `alarm` makes `burglar` and `earthquake` dependent. The `alarm` is the observable variable (v) and the `burglar` and `earthquake` are the hidden 'causes' (h). For example if the `alarm` is true, and we see news of earthquake in the area, our belief that we have been burgled decreases. Expressed (again exemplifying equation 2.1) as a joint probability over A, B and E where these are the states of `alarm`,`burglar` and `earthquake` respectively:

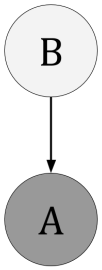


Figure 2.1: Minimal Directed PGM, showing an observed variable 'A' and it's hidden cause 'B'.

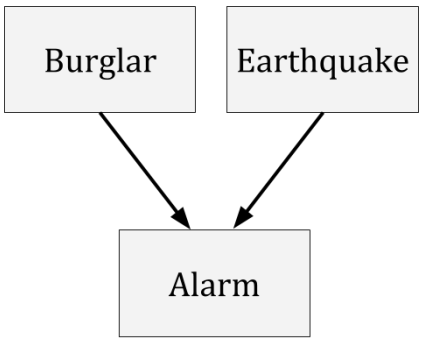


Figure 2.2: The famous `Burglar, Earthquake, Alarm` network showing a minimal case of explaining away.

$$P(A, B, E) = P(A|B, E)P(B)P(E)$$

2.2.2 Directed PGMs in Neural Networks: The Sigmoid Belief Network

A Belief network can be expressed as a neural network, where conditional probabilities are parameterised as weights. This network is called a Sigmoid Belief Network (SBN) as the probability of a variable x_i being 1 is dependent on a ancestor variable $parent_i$. The weighted

sum into x_i, ϕ_i passed through the sigmoid function ($\sigma(x) = 1/(1+e^{-x})$). This is equivalent to a Perceptron using a sigmoid activation function which ensures that the output is a valid probability (between 0 and 1). SBNs take a naive approach to causes, where each hidden unit represents a single, simple cause. Formally, ϕ_i is a weighted sum of the activations of parent nodes:

$$\phi_i = \sum_{j \in parent_i} W_{ij} x_j$$

and

$$P(x_i = 1 | parent_i) = \sigma(\phi_i)$$

2.3 Undirected PGMs:

Undirected PGMs do not represent causation, instead merely capturing a dependency between two units. These pairwise dependencies change the structure of the factorisation a factor Φ between each pair of variables x_i, x_j resulting in the factorisation:

$$P(X) = \frac{1}{Z} \prod_i \Phi(x_i, x_j)$$

The introduction of the normalisation Z (often referred to as the partition function) adds nontrivial complexity to performing inference in Undirected PGMs [29], a sum over all 2^N configurations of x is required.

Undirected PGMs do not capture causal relationships meaning computing the state of a variable given another is no longer hampered by the effect of explaining away. However, their reccurent structure, while expressive introduces an intractability in practice.

2.3.1 Undirected PGMs in Neural Networks: The Boltzmann Machine

A Undirected PGM expressed as nueral network is referred as Boltzmann Machine, where connections encode dependancies with an associated weight. We see this where W_{ij} is the weight between variables x_i and x_j the factor Φ is expressed as:

$$\Phi(x_i, x_j) = e^{x_i x_j W_{ij}}$$

The Boltzmann machine has been proposed in various forms from different domains throughout the years. For instance it was presented in a non-stochastic context of the Hopfield network in [16]. Hinton and Sejnowski [14] also proposed the Boltzmann machine. An example Boltzmann Machine is shown in figure 2.3, we see that the Boltzmann Machine can be recurrent, expressing complex dependencies between variables. This recurrence makes inferring the state of a subset variables based on knowledge of another subset non-trivial as the size of the network grows.

2.3.2 Restricted Boltzmann Machines

A Boltzmann Machine's architecture can be altered to alleviate inference shortcomings. The restriction, originally proposed by [30], and then later revitalised with a training algorithm that operates on the deeper architecture of the DBN [14]. The restriction requires the network to be a two layer bipartite network, each layer corresponding to the observed (visible) and latent (hidden) units. Connections are forbidden between the layer of hidden units and the layer of visible units respectively. An example Restricted Boltzmann Machine architecture is shown in figure 2.4 (with the biases omitted for simplicity.). The collection of hidden

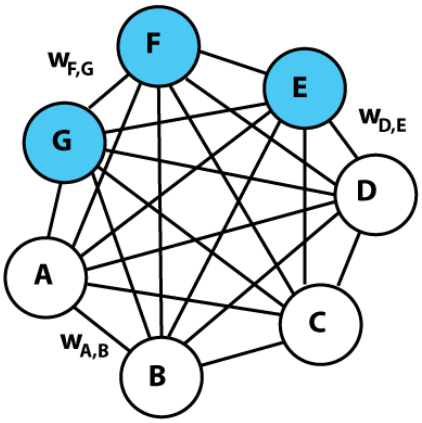


Figure 2.3: A Boltzmann Machine, the blue shaded nodes representing the observed variables, and the non-shaded nodes the latent variables.

units, forming a layer are referred to as the `hidden layer`. The collection of visible units are referred to as the `visible layer`.

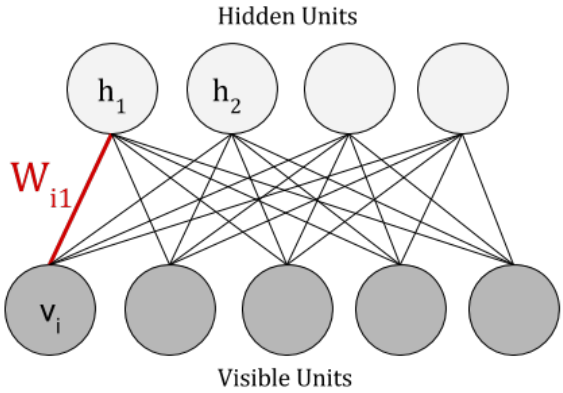


Figure 2.4: An example Restricted Boltzmann Machine with four hidden units, and five visible units. Note that the edges between units are not directed — representing a dependency not a cause. This means that the weights are symmetric.

The probability of the RBM (again ignore biases) being in a given configuration is the joint probability of h and v:

$$P(h,v) = \frac{1}{Z}\prod_{j,i} e^{h_j v_i W_{ji}}$$

Taking logs this becomes:

$$\log P(h,v) = \log \sum_{j,i} h_j v_i W_{ji} - \log Z$$

Z is the partition function, which normalises the probability of the joint. Calculating this would require summing over all 2^N configurations of h and v, which is intractable for practical numbers of units. For instance a 28 by 28 image corresponds to 784 visible units, and for say 10 hidden units, this would amount to $2^{784} * 2^{10}$ possible configurations. We opt to work in terms of $P^\star$ which is the non-normalised probability of the joint over h and v.

$$\log P^{\star}(h, v) = \log \sum_{i,j} h_j v_i W_{ji} \tag{2.2}$$

2.4 Sampling in Generative Models

2.4.1 Why sampling is important

Sampling is used when the distribution we want to work with is intractable to calculate analytically, which happens to be the case in the model proposed in this report. As mentioned in 2.1, the power of generative models is their ability to represent, reconstruct and be trained on data. These tasks all require sampling from configurations/distributions of the hidden and visible units, often conditioned one or the other. There are two cases:

- Sampling from $P(v|h)$, which is known as running the generative model, where the model *generates* data based on a hidden representation.
- Sampling from $P(h|v)$, which is known as *inverting* a generative model (This is also reffered to as *inference*). The process is called 'inverting' because instead of the model generating the data (sampling from $P(v|h)$) we instead try to infer a representation given data $P(h|v)$. It is the process of reasoning about what we do not know, given that of which we do.

Sampling from $P(v|h)$ and $P(h|v)$ is required to train generative models, as often the gradient to be climbed/descended involves calculating a probability over all the units in the generative model. Training a network based generative model involves calculating a weight update, which in turn requires inferring a hidden representation given a training item. The converse, sampling from a $P(v|h)$ is also required [4].

2.4.2 The sampling technique: Gibbs Sampling

Gibbs sampling is a special case of Markov Chain Monte Carlo [8], a technique for drawing samples from a complex distribution. Sampling from the probability mass (or 'joint distribution') of a generative model is a common use case for Gibbs sampling [26].

Gibbs sampling explores the desired probability distribution, taking samples of that distribution's state. One must leave iterations of 'exploration' (of the probability mass) between drawing of a sample to ensure that the samples are independent [3]. The process of taking a step between states is referred to as a `Gibbs iteration` or a `Gibbs Step`. Formally the algorithm is described in algorithm 1.

Data: A vector x indexed by j.
Result: Gibbs sampling algorithm
Let $x_{\setminus} j$ be all components that make up x vector except x_j;
initialization, begin with x, we are going to get a sample x';
for *k many iterations* **do**
  for *each component in* x, x_j **do**
    Draw a sample, x'_j from $P(x_j|x_{\setminus} j)$;
    Update the current value of x_j in x with x'_j;
  end
end

Algorithm 1: The Gibbs Sampling Algorithm

Mixing Time of the Gibbs Sampler

MCMC methods aim to approximate a distribution, by exploring likely states. As we often start this process from a random state, it's important that enough Gibbs steps are taken before a sample is drawn. This is because the random state may not be close to any part of the true distribution we want to sample from, so by running the chain for many iterations we increase the likelihood of samples being from the desired distribution.

This process of waiting for enough steps to before drawing samples is referred to as the `Mixing Time`. Gibbs sampling is used for performing inference in RBMs [12] and as result also in the ORBM. The `mixing time`, that is how many Gibbs iterations are needed to reach a satisfactory sample is an important part issue in the ORBM, in that one Gibbs step was sufficient in practice for training and using an RBM. The new generative model is not so fortunate.

2.4.3 Sampling in a Sigmoid Belief Network

Sampling from $P(v|h)$ (known as Ancestral Sampling in a SBN) is extremely efficient in a SBN, it does not need Gibbs sampling. It can be expressed as

$$P(v|h) = \sum_i v_i \log \sigma_i(h) + (1 - v_i) log(1 - \sigma_i(h)) \tag{2.3}$$

As this process is calculated by propagating probabilities from the root hidden causes, it is known in a SBN as Ancestral Sampling.

Conversly, sampling from $P(h|v)$ in an SBN is intractable. Performing inference in a Sigmoid Belief network would allow source separation, as each hidden unit could represent a simple cause. The SBN would be shown an input, and could extract representations of the seperate causes that likely gave rise to the input. There exist algorithms for performing inference in Sigmoid Belief Networks. For instance, the Belief Propagation algorithm proposed by Judea Pearl [25] operates on this encoding, calculating the probabilities of a given network state (i.e. the state of all the variables). As well as constraining the architecture to be a Directed Acyclic Graph, Belief Propagation is intractable to use as the number of variables grow [5]. As we want to work with cases with upwards of 100 visible and hidden nodes, these algorithms break down.

Despite the Sigmoid Belief Network being expressive and providing a succinct encoding of conditional probabilities, Gibbs sampling is intractable for SBNs of practical size [18]. The issue being that the Gibbs chain takes too long to mix [23], which arises from the the 'explaining away effect' [12]. Calling back to the example in section 2.2.1, instead of a single `burglar` or `earthquake` becoming dependent given the `alarm`, there is more in the realm of hundreds of burlgars and earthquakes all suddenly dependent.

Being able to efficiently sample from $P(h|v)$ is also required for training generative models [5] making Sigmoid Belief Networks impractical for not only inference, but also training.

2.4.4 Gibbs Sampling in a Boltzmann Machine

Performing Gibbs sampling appears trivial in a Boltzmann Machine, in that to find the probability of a given unit being active, a weighted input to that node is passed through a sigmoid function. However, in practice the recurrent nature of Boltzmann Machines makes sampling intractable, as updating a node will change the probabilities of those connected. This requires a long Gibbs chain to sample from, intractably long in practice.

Recall that $x_{\setminus j}$ be all components that make up x vector except x_j and that a Boltzmann Machine has symmetric weights ($W_{ji} = W_{ij}$),

$$P(x_j = 1, x_{\setminus j}) = \frac{1}{1 + e^{-\sum_i w_{ji} x_i}}$$

That is, Gibbs sampling in a Boltzmann Machine amounts to use the Sigmoid function of the weighted inputs [23].

2.4.5 Gibbs Sampling in RBMs

A big payoff for the restriction in an RBM is inverting the model becomes tractable, as the latent variables no longer become dependant given the observed ones. This is illustrated in figure 2.4 the hidden unit h_1 is not dependent on h_2, regardless if we know anything about the visible units. Firstly, this is the opposite of a SBN, where knowledge of the visible units makes the hidden units dependent. Secondly, by removing the recurrence present in Boltzmann Machines, it reduces the expressiveness of the RBM network. This does make the RBM useable in practice, as the Gibbs sampling process can stop after one Gibbs step[6].

In order to describe Gibbs sampling in the new architecture proposed, it must first be explained for a standard RBM — The process of Gibbs sampling is as follows:

- One must sample from $P(h|v)$ giving a hidden state h'
- Using this hidden state, a visible state is then generated, v', by sampling from $P(v'|h')$. This process of generating a hidden pattern, and subsequent visible pattern is referred to as a Gibbs step.
- This chain of Gibbs steps between sampling from $P(h|v)$ and $P(v|h)$ can then be repeated as desired, the longer the chain the closer the samples will be to the true joint distribution that the model has learnt. For training an RBM Hinton[12] showed that one step is often enough in practice, as one step is enough to infer a direction to adjust the weights in.

The process of updating the hidden, then visible layers forms what is referred to as the `Gibbs Chain` and is visualised at layer level in Figure 2.5.

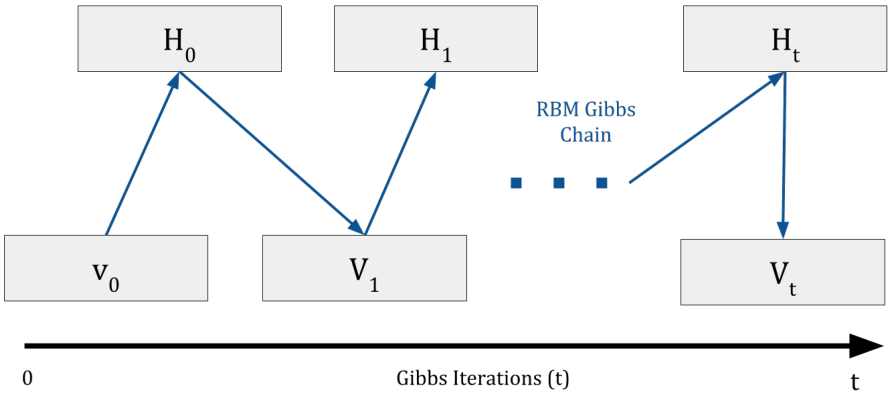


Figure 2.5: A figure illustrating a Gibbs chain where left to right indicates a Gibbs iteration. Note this is *not* a PGM.

The Gibbs update

Denote the Gibbs sampler's probability of setting a variable, x_j to 1 as:

$$P_{Gibbs}(h_j = 1|v, h_{k!=j}) \tag{2.4}$$

We can refer to the proability in equation 2.4 as p_1 and the converse $P_{Gibbs}(h_j = 0|v, h_{k!=j})$ can be referred to as p_0. Then, by the product rule of probabilities:

$$\begin{aligned}
\frac{p_1}{p_0} &= \frac{P^\star(h, v \text{ where } h_j = 1)}{P^\star(h, v \text{ where } h_j = 0)} \\
&= \frac{p_1}{1 - p_1} \\
p_1 &= \frac{1}{1 + \frac{P^\star(h,v \text{ where } h_j=0)}{P^\star(h,v \text{ where } h_j=1)}} \qquad = \frac{1}{1 + e^{-\psi_j}}
\end{aligned}$$

To update a hidden unit h_j we find $P(h_j = 1|v)$ where v is an input pattern. In the context of an image, v would be the pixel values where each pixel corresponds to a visible unit, v_i. The probability of a given hidden unit activating is:

$$P(h_j = 1|v) = \sigma(\psi_j) \tag{2.5}$$

Where ψ_j is the weighted sum into the jth hidden unit and $\sigma()$ is the Sigmoid function, or it also known as the Logistic function $\sigma(x) = 1/(1 + e^{-x})$. Figure 2.6 illustrates ψ_j for an example RBM. As the weights are symmetric, sampling from the visible layer, given a

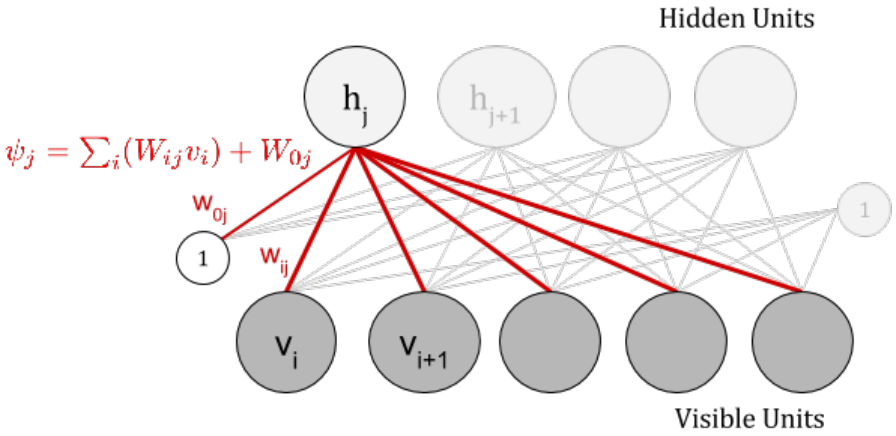


Figure 2.6: A diagram showing ψ_j, the weighted sum into the jth hidden unit. Note that W_{0j} is the hidden bias, represented as a unit that is always on with a weight into each hidden unit.

hidden state is similar. That is $P(v_i = 1|h)$, where h is the entire hidden vector is given by:

$$P(v_i = 1|h) = \sigma(\phi_i) \tag{2.6}$$

Where ϕ_i is the weighted sum into the ith visible unit, which is: $\phi_i = \sum(W_{ji} h_j) + W_{0i}$. Both ϕ_j and ψ_i can be expressed in alternative, but useful way:

$$\phi_j = \log P^\star(v, h|v_i = 1) - \log P^\star(v, h|v_i = 0) \tag{2.7}$$

$$\psi_i = \log P^\star(h, v|h_j = 1) - \log P^\star(h, v|h_j = 0) \tag{2.8}$$

Reconstructions: visualising what the RBM has learnt

RBMs create an internal representation given an input by sampling from $P(h|v)$. They can also generate a faux input given an internal representation (a h). Performing one Gibbs iteration, that is, sampling from the hidden units given a 'clamped' input $P(h|v)$ and then taking the generated hidden state and generating a faux input (sampling from $P(v|h_{sampled})$) results in a `reconstruction`. Clamped input is where the visible units are set to be an input pattern. The model tries to reconstruct the input based on the internal representation it has learnt to model.

RBM Fantasies: The Free-Phase of a Generative model

In the same way that a Generative model uses reconstructions to try and recreate the supplied input vector, performing many Gibbs iterations with no input pattern clamped and taking a sample is referred to as free-phase sampling of the model. It allows the samples to explore the probability mass that has been built by the model during training. Sampling from these wanderings creates what are referred to as 'fantasies' or 'dreams'.

Chapter 3

The ORBM: A model of independent complex causes

3.1 A network equivalent to an RBM

3.1.1 Unrolling a Gibbs Chain, a different perspective on RBM inference

Before the ORBMs architecture and inference algorithm can be introduced, Gibbs sampling in an RBM must be presented in a different, yet equivalent way. Hinton, when introducing the idea of training a Deep Belief Network showed that a Gibbs chain in an RBM is equivalent to a infinitely deep belief network, with an RBM on the top with tied weights [13]. An unrolling of a single Gibbs iteration is illustrated in figure 3.1, the U layer corresponding to the V layer after one Gibbs iteration. The top two layers (U and H) form a standard RBM, but the bottom connections (V and H) form a Sigmoid Belief Network. To further clarify, the U layer corresponds to V_1 in figure 2.5. Note in figure 3.1 the weights between the H and U layer are shared between the V and H layers.

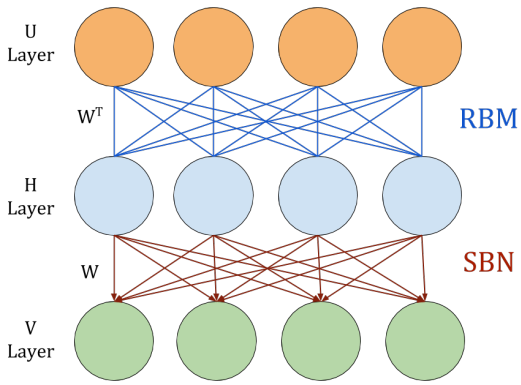


Figure 3.1: A diagram illustrating 'unrolling' an RBM by one Gibbs iteration. Note the connections between H and V layers are now directed.

We can now show that Gibbs sampling in this RBM unrolled with a Sigmoid belief network is equivalent to Gibbs Sampling in a standard RBM.

Sampling in this equivalent network

Sampling in the ORBM network behaves a similar way to RBM sampling, where a Gibbs chain is run between the top two layers, U and H, until the last iteration. At the last iteration a hidden pattern is sampled and then is pushed through the Sigmoid belief network between H and V. This is illustrated in figure 3.2.

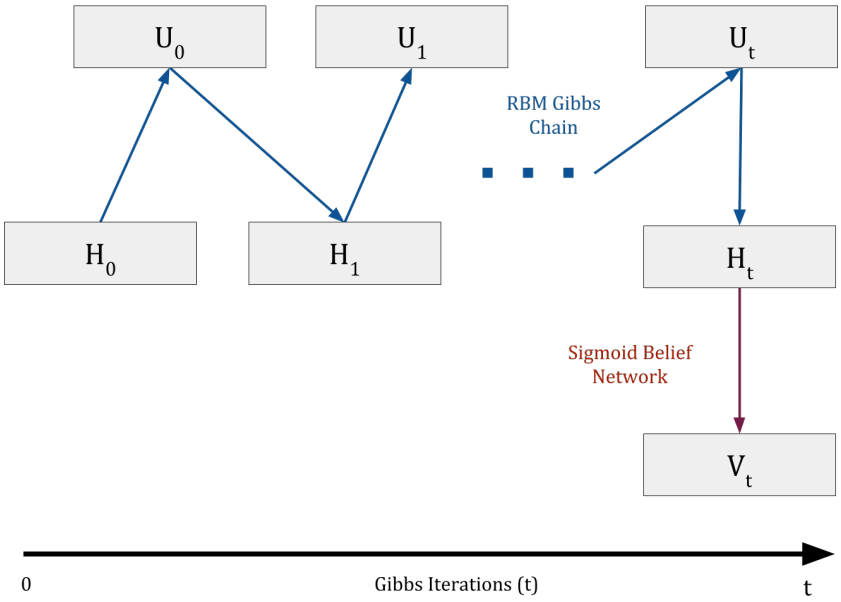


Figure 3.2: A diagram showing sampling in the equivalent network, where normal sampling in the top 2 layers is performed until Gibbs iteration t, and the hidden state is pushed down through the bottom layers of the Sigmoid Belief Network.

To generate a sample from h is equivalent we can use equation 2.5, by running the Gibbs chain between h and u. To sample from v in the SBN is by definition:

$$P(v_i = 1|h) = \sigma_i(h)$$

Thus Gibbs sampling from a simple RBM ending in a sample for v is equivalent to sampling h from the same RBM and using a SBN for the last step. However another useful way to draw samples in such a network that we will leverage in the ORBM. The log likelihood of the joint can be written as:

$$\log P^\star(h, v) = \log P^\star(h) + \log P(v|h) \tag{3.1}$$

The second term is defined in equation 2.3. To find the first term, $P^\star(h)$, we need to marginalise the joint (eq 3.1) over all $\mathbf{U}$ layer configurations:

$$
\begin{aligned}
P^\star(h) &= \sum_{v_1=0}^{1} \cdots \sum_{v_n=0}^{1} \exp\left[\log P^\star(h,v)\right] \\
&= \sum_{v_1=0}^{1} \cdots \sum_{v_n=0}^{1} \exp\left[\sum_i \sum_j h_j W_{ji} v_i \;+\; \sum_i W_{0i} v_i \;+\; \sum_j W_{j0} h_j\right] \\
&= \sum_{v_1=0}^{1} \cdots \sum_{v_n=0}^{1} \exp\left[\sum_i v_i \phi_i(h) \;+\; \sum_j W_{j0} h_j\right] \\
&= \exp\left[\sum_j h_j W_{j0}\right] \;\times \sum_{v_1=0}^{1} \cdots \sum_{v_n=0}^{1} \prod_i \exp\left[v_i \phi_i(h)\right] \\
&= \exp\left[\sum_j h_j W_{j0}\right] \;\times \prod_i \left(1 + e^{\phi_i(h)}\right) \\
\text{and taking logs, } \log P^\star(h) &= \sum_j h_j W_{j0} \;+ \sum_i \log\left(1 + e^{\phi_i(h)}\right) \\
&= \sum_j h_j W_{j0} \;+\; \sum_i \phi_i(h) \;-\; \sum_i \log \sigma_i(h)
\end{aligned}
$$

$\log P^\star(h)$ for the RBM that is the 'top layer' has now be defined. Given this, another way to write $\log P^\star(h,v)$ is

$$
\log P^\star(h,v) = \underbrace{\sum_j h_j W_{j0} \;+\; \sum_i \phi_i(h) \;-\; \sum_i \log \sigma_i(h)}_{\log P^\star(h)} \;+\; \underbrace{\sum_i v_i \log \sigma_i(h) + (1 - v_i) \log(1 - \sigma_i(h))}_{\log P(v|h)} \tag{3.2}
$$

By collecting terms and simplifying this matches the earlier form in equation 2.2.

3.2 A New Approach, The ORBM

3.2.1 Architecture

Frean and Marsland extend on this idea of a 'One Gibbs iteration unrolled RBM'. They propose adding another U and H layers to represent another rich source of data, which then combines with the original U and H layer via a SBN to form the visible layer, V. This architecture is illustrated in figure 3.3, where A and B are used to denote the two independent causes we are modeling. To clarify the ORBMs architecture, there are two RBMs, one for each cause. These RBMs are connected to a SBN into the visible layer, the weights of which are tied to the respective RBMs. Performing inference in this architecture is slightly simpler for performing inference, as the U layers are not involved.

By building on RBMs and an SBN we can leverage existing algorithms and work on RBMs. Also the potential for extending this work to deeper architectures could be possible. Also the architecture supports 'plug and play' with the models in that existing RBMs can be plugged into the architecture. The implications of this are exciting in that an RBM that has been trained on hand written digits could be plugged into an RBM that has been trained on a paper texture (the bumps and ridges of paper). Then using the ORBM a 'clean' representation of the text and paper could be separated out given an image of a digit on paper.

The ORBM, like the RBM is difficult to evaluate empirically, but the same techniques that can be used to evaluate RBMs can be applied to the ORBM.

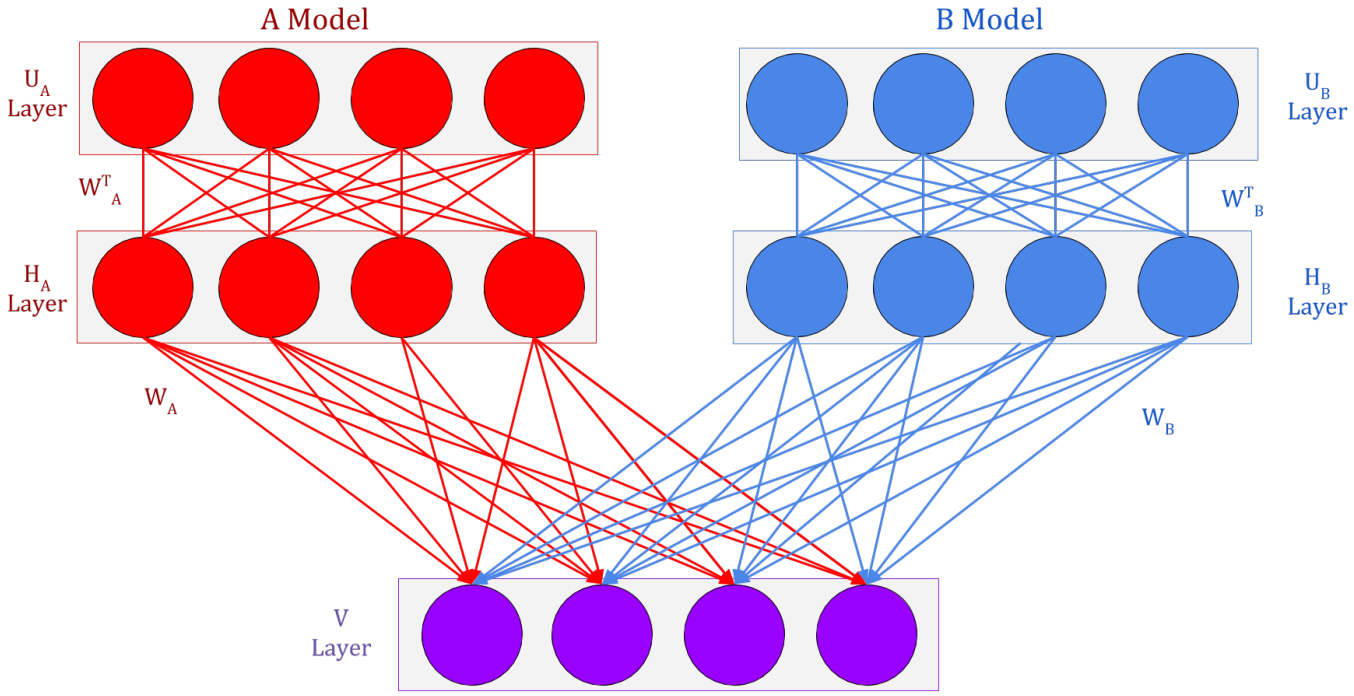


Figure 3.3: The full ORBM architecture, where A and B are the two causes that combine to form the data.

3.2.2 Gibbs Sampling in the ORBM

Sampling in the ORBM

Sampling from the generative model in an RBM (Section 2.4.5) involved running a Gibbs chain and then taking the last visible pattern in that chain. To perform Sampling in the ORBM we can leverage the generative power of the two RBMs and then combine their inputs in a simple way. We know that joint in the unrolled RBM is expressed as eq 3.1. That is, the ORBM probability of v given h^A and h^B is defined by:

$$\log P(v|h^A, h^B) = \sum_i v_i \log \sigma(\phi_i^A + \phi_i^B) + (1 - v_i) \log(1 - \sigma(\phi_i^A + \phi_i^B)$$

Where ϕ_i^A and ϕ_i^B are the weighted sums into the ith visible units from the models A and B respectively. In this generative model a visible unit is created by taking the weighted sum from both sources, adding their contribution and then passing through a Sigmoid function to give a probability.

Inference in the ORBM

In a standard RBM sampling from $P(h_j = 1)$ is given by $P(h_j = 1) = \sigma(\psi_i)$, where ψ_i is defined in equation 2.8. In an ORBM we cannot consider the single RBM alone when trying to find $P(h_j)$, as the hidden layers of the two RBMs in the ORBM are dependent given a visible layer v. This amounts to explaining away as described in section 2.2.1. There is a nice feature of the ORBM in that the hidden representations (h^A and h^B) we extract from the visible layer require no interaction with the U layers to sample from. The U layers are only needed to generate a composite V pattern (or independent reconstructions).

We aim to use Gibbs sampling to generate h^A and h^B given a v: We need to calculate

$$\psi_j^A = \log P^\star(h, v|h_j^A = 1) - \log P^\star(h, v|h_j^A = 0)$$

We will use that fact that the weighted sum into a visible unit i where some hidden unit h_j is on, is equivalent to the same configuration except h_j is off plus the weight between these two units. This is by definition true, and expressed below:

$$\phi_i^A(h|h_j^A = 1) = \phi_i^A(h|h_j^A = 0) + W_{ji}^A$$

We will abbreviate $\phi_i^A(h|h_j = 0)$ to ϕ_i^{Aj0}. Given these we obtain:

$$\psi_j^A = \sum_i v_i \log\left(\frac{1+e^{-\phi_i^{Aj0}-\phi_i^B}}{1+e^{-\phi_i^{Aj0}-W_{ji}-\phi_i^B}}\frac{1+e^{\phi_i^{Aj0}+W_{ji}+\phi_i^B}}{1+e^{\phi_i^{Aj0}+\phi_i^B}}\right) \;+\; \sum_i \log\left(\frac{1+e^{\phi_i^{Aj0}+W_{ji}}}{1+e^{\phi_i^{Aj0}}}\frac{1+e^{\phi_i^{Aj0}+\phi_i^B}}{1+e^{\phi_i^{Aj0}+W_{ji}+\phi_i^B}}\right)$$

Now $\phi = \log\frac{1+e^{\phi}}{1+e^{-\phi}}$, which is$= \log\frac{\sigma(\phi)}{\sigma(-\phi)}$. So the first term simplifies to $\sum_i v_i W_{ji}$, which is the same as that in an RBM. The second term can also be simplified, using the identity $\log(1-\sigma(\phi)) = \phi - \log(1+e^{\phi})$. This leads to the following Gibbs Sampler probability of the j-th hidden unit in network A being 1: $p_j = \sigma(\psi_j^A)$ with

$$\psi_j^A = \underbrace{\sum_i W_{ji}^A v_i}_{\text{vanilla RBM}} \;+\; \underbrace{\sum_i C_{ji}^A}_{\text{correction}}$$

where the 'full' Correction is:

$$\begin{aligned} C_{ji}^A \;&=\; \log\left[\frac{\sigma(\phi_i^{Aj0})}{\sigma(\phi_i^{Aj0}+W_{ji}^A)}.\frac{\sigma(\phi_i^{Aj0}+W_{ji}^A+\phi_i^B)}{\sigma(\phi_i^{Aj0}+\phi_i^B)}\right] \\ &= \log\sigma(\phi_i^{Aj0}) \;+\; \log\sigma(\phi_i^{Aj0}+W_{ji}^A+\phi_i^B) \;-\log\sigma(\phi_i^{Aj0}+W_{ji}^A) \;-\; \log\sigma(\phi_i^{Aj0}+\phi_i^B) \\ &= \log\left[\frac{\sigma(\phi_i^A - h_i^A W_{ji}^A)}{\sigma(\phi_i^A + (1-h_i^A)W_{ji}^A)}\right] \;-\; \log\left[\frac{\sigma(\phi_i^{AB} - h_i^A W_{ji}^A)}{\sigma(\phi_i^{AB} + (1-h_i^A)W_{ji}^A)}\right] \end{aligned} \tag{3.3}$$

where $\phi_i^{AB} = \phi_i^A + \phi_i^B$. Note that v plays no role in the correction. It is clear that adding ϕ_i^B has introduced a dependency between the entirety of h. This means that a Gibbs chain will need to be run, in practice this chain proves to be short, even in the larger dimension tasks like MNIST.

Examining and approximating the Correction

This 'full' correction (eq 3.3) is a non-trivial computation to make in that for every weight, a series of semi-large matrix operations have to be performed. As a result, Frean and Marsland propose an approximation for this 'full' correction.

It is useful to see the correction for the A model as contours on axes ϕ_i^A versus ϕ_i^{AB}, for a positive weight W_{ij}^A. There are two plots for the two cases $h_j^A = 0$ and $h_j^A = 1$ These are plotted in figure 3.4. This function has defined 'plateaus' where the height of these plateaus is $\pm$ the weight respectively. The correction essentially adjusts the weight between v_i and h_j allowing it to be turned off, turned on, intensified, de-intensified, or have no effect. This is similar to adjusting the visible activation based on what the two RBMs are doing and hence Frean and Marsland propose the following approximation:

$$C_{ji}^A = \sigma(\phi_i^{A_{j0}} + W_{ji}^A) - \sigma(\phi_i^{A_{j0}} W_{ji}^A + \phi_i^B) \tag{3.4}$$

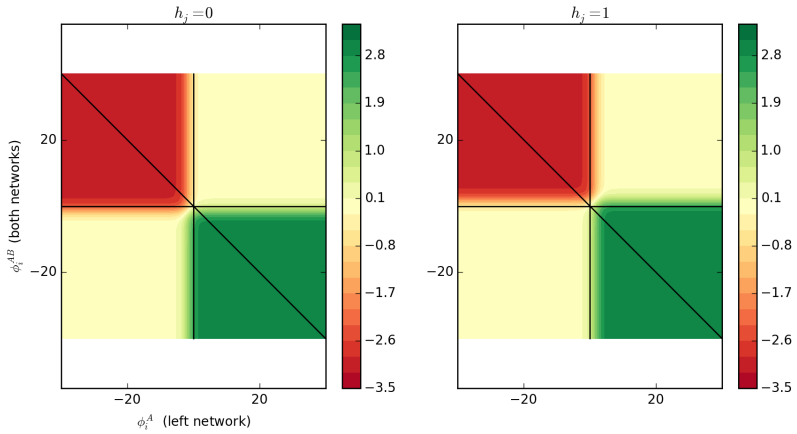


Figure 3.4: A diagram that shows the structure of the correction over the weighted sums into the hiddens of one of the RBMS versus both. Note the plateaus that are $\pm weight$ in magnitude

What happened to the biases?

The ORBM utilises the hidden biases present on the RBMs when calculating the hidden states h^A and h^B. However, the visible biases are not used in sampling from the visible. The reasoning behind this being that the RBMs visible bias acts like the 'background rate', i.e. what visible reconstruction would we see from an all zero hidden activation. As visible bias are not captured in the ORBM, it is important that RBMs plugged into it's structure do not use a visible bias when being trained.

Reconstructions and Dreams in the ORBM

Reconstructions in the ORBM are a natural extension of the inference algorithm.

- First hidden representations h^A and h^B are generated given an input v.
- The RBMs (RBM A and B) use their respective hidden representations to generate visible patterns independently. That is, we can use the same way of performing Gibbs sampling we saw in equation 2.6.

This means that for a visible pattern there are potentially two reconstructions, one from each model.

Chapter 4

Design and Implementation

A significant hurdle for the project was gaining enough understanding of the existing work on RBMs to be able to implement the ORBMs algorithm and architecture. This was crucial as an incorrect implementation invalidates Contributions 2 and 3 of this project.

4.1 Implementation Design

4.1.1 Language Choice

The implementation of the ORBM and inference algorithm is in *Python*, with *Matlab* and *Java* being the other languages originally considered.

I have spent my university career working in Java, and nowdays it is efficient to perform machine learning tasks. Python's module and class system promotes composition, and multiple inherietance in particular allows for composition of different classes. Matlab is a popular numerical computing environment and has a lot of online resources as well as machine learing papers [13] that include snippets or full Matlab programs that were used in the paper. Also Matlab is built around strong, efficient support for matrix operations which are very prolific in the RBM and ORBM implementations and evaluations.

Python has a very succinct syntax and shallow learning curve, as well as being the language of choice of my supervisor. This is favourable as I can have langauge level support for translating the algorithm into python correctly. Through libraries that supply wrappers for C-bindings, efficient code can be written in Python. Overall, being an interpreted language Python is slower than Matlab and Java. Three main factors outweighed this and resulted in the choice of Python:

Up front learning Given the amount of up front learning of concepts required to implement the solution, Python was favoured over Java or Matlab because is has a very shallow learning curve. Despite having experience in Java as well as Python, I had never used machine learning or linear algebra libraries in either languages. Given the amount of up front learning required to understand the concepts in this project let alone implement it, I opted for the shallower learning curve of Python. Also my supervisor has experience using Python and the libraries involved with this project. A shallower learning curve meant that more time could be allocated to the evaluation which is the foremost contribution of this project.

Library Support Matlab and Python were two contenders in this factor, as a lot of matlab machine learning code is avaliable with supporting papers. Python has libraries such as NumPy, SciPy and Matplotlib [1] which mirror Matlab functionality. However, NumPy and SciPy are all open source, while there is argument for treating an

API as a black box — i.e. not writing code that is dependant on the underlying implementation, having the option to view source helped me better grasp the concepts. In particular being able to compare my implementation of the RBM to the Sklearn implementation [27] allowed for some performance improvements to my implementation.

Ease of Evaluation Python provides a strong plotting library that is inspired by Matlab and R — MatplotLib. Matplotlib is interoperable with the NumPy library, meaning that visualising results was really easy.

Also the evaluation requires repeatedly running a test to gain more confidence in the stochastic result. This is made possible by the ECS-Grid. Python only required a simple script that manages input and output. However the Java support on the ECS grid requires significantly more setup. This seemed like an unnecessary risk to introduce to the project at the later stage when the evaluation would being carried out.

4.1.2 Program Architecture

The implementation of the RBM, ORBM and algorithm is implemented with unit testing and composability in mind. It is important that the design supports comparing the Full and Approximated Correction (Equations [**?**] and [**?**]. By using Python, multiple inheritance could be leveraged to achieve this composability. For instance adding support for continuous pixel values to the `Full Correction Sampler`, one simply needed to extend `Continuous Approximate Correction Sampler` and the `Full Correction Sampler`, with no code actually in the new class. The architecture is pictured in the class diagram 4.1. There were three main roles these classes filled:

1. The `Trainer` was used to train the weights of a supplied `RBM`. It would do so using a supplied `sampler`, decoupling how samples were generated from the training process.

2. The `RBM` was the model of an RBM, storing the weight matrix, as well as parameter information. Also this supported conversion of the SciPy Sklearn libraries' RBM implementation into the implementation used in this project. Decoupling the concept of an `RBM` from the `Sampler` and the `Trainer` meant that RBMs could be 'plugged' into the ORBM architecture with ease.

3. The `Sampler` defined how to perform Gibbs sampling, the subclasses defining whether it is standard RBM sampling or ORBM sampling. This made it trivial to compose `samplers`, which was required for the ORBM samplers.

4.1.3 Testing Approach

The inference algorithm is the most crutial part of the implementation to test. In larger dimensional tasks, unit testing to ensure the algorithm produces the correct values becomes difficult. In smaller cases I was able to ensure the probabilities of $P(h|v)$ and $P(v|h)$ calculated by the implementation matched calculations I made by hand. Being an unsupervised black boxes, determining if an RBM (and by extension the ORBM) hidden representation is 'correct' for larger tasks such as the MNIST handwritten digit dataset is non-trivial. Concerned by this risk, I designed my evaluations to build confidence in the algorithm and the implementation.

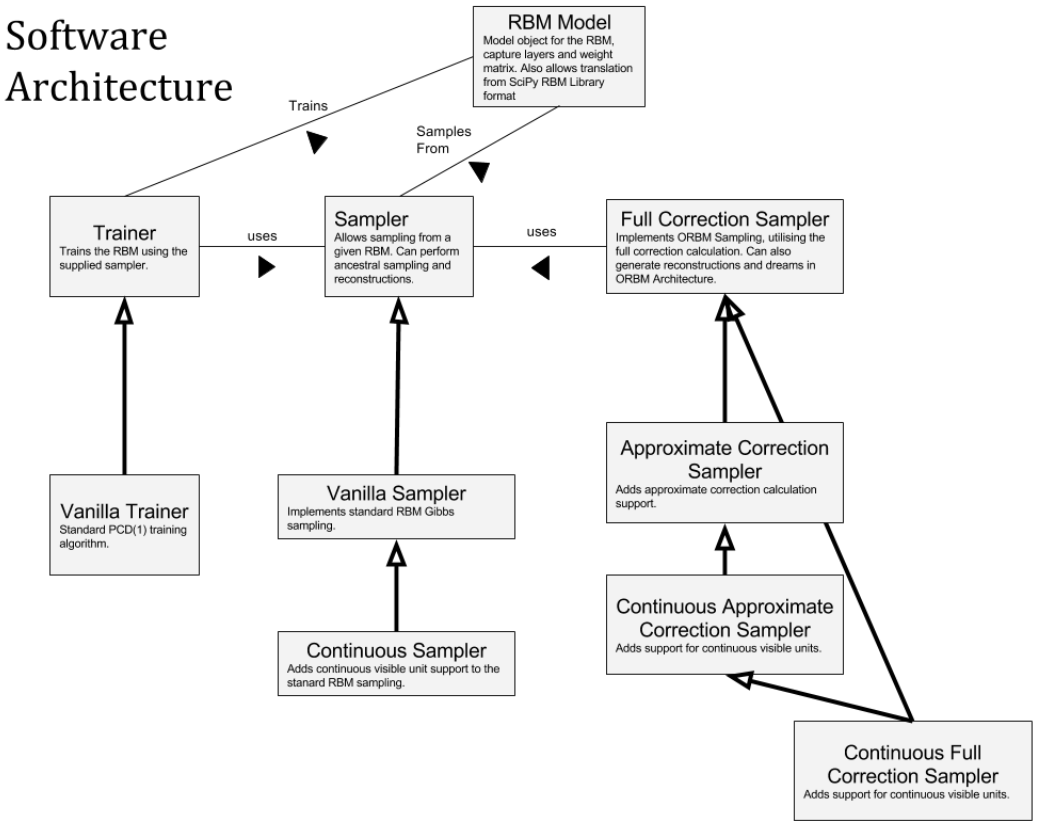


Figure 4.1: A figure showing the architecture for the implemented test suite in UML notation.

4.2 Evaluation Design

4.2.1 Examining the mixing time of inference in the ORBM

As described in Section 3.2.2, sampling from h^A and h^B suffers from the effect of explaining away (Section 2.2.1) which requires a Gibbs chain to be run. While intractable for a large SBN, as only two causes are being modeled, ideally the Gibbs chain for this process won't take many iterations to mix. As the correction needs to be computed for each Gibbs iteration, a long Gibbs chain could be detrimental to performance. Finding this *optimal mixing time* for the ORBM is only an issue as the number of dimensions and training examples increase. In smaller dimensions mixing can be overcome by using a large (> 1000) number of Gibbs iterations, as it computational feasible to do so. However in the larger problems this becomes intractable, so an optimal mixing time is desired.

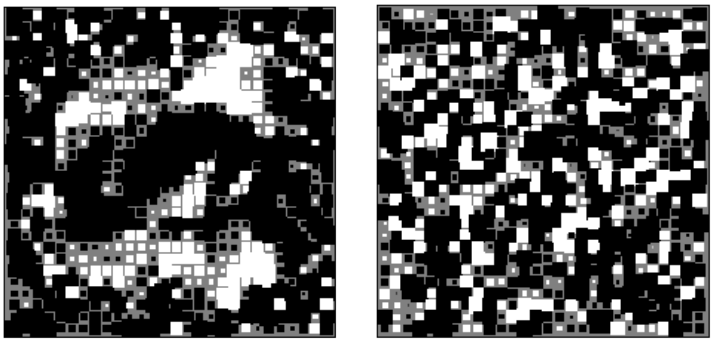
This project examines the mixing time by way of examining reconstructions at various points of the Gibbs chain. This is also interesting in that we can observe the dynamics of the network and the effect of applying the *correction* has on reconstructions. Hinton [9] has had success observing reconstructions in an animated setting.

4.2.2 Evaluating RBMs and ORBMs

A challenge faced by this project and work with RBMs is that they are non-trivial to evaluate. Being an unsupervised black box, one cannot merely inspect the hidden units sampled after a given input and be sure that a good model has been learnt. As I plug RBMs into the ORBM architecture it is important for the overall results that the RBMs are well trained.

4.2.3 Hinton diagrams

We can examine the weights into a given hidden unit in the shape of the visible vector. For instance in the context of images, `Hinton Diagrams` allow visualisation of what a given hidden unit is 'doing' by visualising the matrix of weights into that unit. This was first used by Hinton in the context of Boltzmann Machines in [15]. They also give insight into hidden unit utilisation. Weights are often initialised to small random values resulting in a Hinton diagram with little structure to it. This is illustrated in figures 4.2a and 4.2b. The former showing a hidden units weights where some structure has been learnt, and the latter showing the opposite.



(a) Utilised hidden unit Hinton Diagram

(b) Unutilised hidden unit Hinton Diagram

Figure 4.2: Hinton Diagrams illustrating a trained hidden unit, versus that of an untrained/unutilised hidden unit. This is with no measures to enforce sparsity.

4.2.4 Reconstructions: a measure of performance

Reconstructions, as described in Section 2.4.5, are a good measure of performance. Reconstructions will be the main way I evaluate the RBM in comparison to the ORBM. To be able to use this reconstruction based evaluation, knowledge of the ground truth is required. This is so:

- RBMs could be trained and then plugged into the ORBM network.
- Reconstructions could be compared directly (by score) to the ground truth.

4.2.5 Evaluating RBMs: Problem dependent

Evaluating RBMs is problem dependent, we can examine the different approaches available by splitting problems into two classes:

Problems with small dimensionality When the dimensionality of the input is very small (< 10) and all possible inputs are known, we can perform analytical evaluations of

the RBM. By clamping the visible units of the RBM to each input in the training set, reconstructions can be created. This process can be repeated for each input, recording the frequency that each reconstruction occurs on a bar graph. We can then ensure that the RBM is reconstructing the input perfectly.

In a similar way to the reconstructions, samples can be drawn from RBM in a 'free phase' without the input clamped to a given visible. The bar graph should exhibit dream visible patterns from the whole training set approximately an equal proportion to that of the training set.

Problems with large dimensionality In non-trivial cases, with larger datasets, reconstructions can be inspected and compared to the training dataset. However, empirically detecting if a model is trained is difficult, especially given the unsupervised, black box nature of RBMs. Alternatively, a 'score' can be assigned to each item in the training set based on how well the RBM can reconstruct it. This project will use two scores the *Cross Entropy* [7] and the *Negative Cosine Angle* to compare the reconstructions to the ground truth images. Dreams can also be examined in larger cases, but empirically detecting if they are 'correct' is infeasible. Nevertheless, Dreams are useful to examine, as if they look like items in the training set, this gives confidence the RBM is capturing the model.

All evaluations followed the same high level process, working from trivial cases to more challenging tasks. By changing only a few aspects from problem to problem it allows me to identify what qualities of the problem make the algorithm less or more effective. These asepcts were typically the dimensionality and then ultimately the difficultly of the problem. The justification for this being that trivial cases make unit testing feasible, therefore ensuring conclusions can be drawn with regard to the algorithm and model, not an incorrect implementation.

The evaluations in this project aimed to evaluate the ORBM and it's inference algorithm by examining reconstructions given a multi-cause image. Because the multi-cause inputs were images composed of known images, I could then compare the reconstructions to the ground truth. An example of a multi-cause input with two quadrilaterals is shown in Figure 4.3. The optimal reconstructions are equivalent to the two images that combined form the input.

In the smaller dimensional cases the reconstructions are inspected by plotting the reconstructions and the frequency with which they occurred after a large amount of repetitions. In the larger dimensional tasks, the two 'scores' are used to evaluate reconstructions against the corresponding item of the training set.

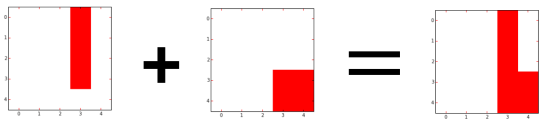


Figure 4.3: A figure illustrating two five by five pixel images combining to form a composite/multicause input. The images are binary.

4.2.6 Choice of Evaluation Datasets

As my evaluations follow the same outline of training RBMs, plugging them into the ORBM, and evaluating the reconstructions, the key aspect of the evaluation that needs to be designed is the datasets for these tasks. It is important the images of the datasets are *multi-subject* and that the individual images used to compose the training images are known. This is to allow the reconstructions of

the ORBM (and RBM) to be compared to these underlying images, exploring how well the ORBM and RBM can perform source separation.

2 Bit XOR The minimal case of image source separation. The RBMs are trained on a single bit being on in a two bit pattern. The training set then resembles a 2 bit XOR truth table. Because the dimensions of this task are so small, reconstructions and dreams can be evaluated empirically. Also the algorithm can be unit tested ensuring the outputs of the different steps are correct, comparing them to values calculated by hand. Also as the dimensionality is so small, the Approximated Correction and the Full Correction can be compared to ensure that the approximated correction works well in practice.

X **neighboring bits on in** Y **bit pattern** This is a natural next step from 2 bit XOR, and is effectively the same task but in larger dimensions. It is trivial to train an RBM to represent this problem, and the algorithm is quick to run on such a small dimensionality creating a quick feedback loop for development.

2 by 2 squares in a 5 by 5 pixel image This dataset extends the previous increasing to 25 dimensions. The dataset is trivial to construct and corresponds to a square of 2 by 2 pixels being on in a 5 by 5 pixel image. Interesting cases can be explored and then inspected visually as images, which are easier to interpret than bit strings (especially as the bit strings get larger than 10).

Rectangles in a 5 by 5 pixel image This dataset builds on the previous by introducing different shaped *subjects* to the images. This means that two RBMs need to be trained, one for each subject. This dataset, much like the previous allows for compelling cases to be separated. It sets the scene for the larger dimension cases.

MNIST Handwritten Digit Dataset This is a prolific machine learning dataset [20]. By using the dataset in this project it aids in reproducibility and as there has been previous work it is known that RBMs can be trained successfully on the data. The dataset is comprised of 28 by 28 pixel handwritten digit images, where a pixel value is between 0 and 1. There are digits 0 through 9. To make a multi-cause input, each digits it is composed with every other digit, making the novel, yet non-trival task of separating two digits from a single image.

Chapter 5

Evaluation

5.1 Two bit XOR: The minimal case

Description

The starting point for the evalatuion is examining the ORBM reconstructions in a two bit pattern, where two of the same model are combining to form the data. This model makes one bit on in a two bit pattern, i.e. $[1,0]$ or $[0,1]$. The training set is the two bit XOR truth table and it provides the minimal case to compare the full correction calculation, versus that of the proposed approximation (see Equation 3.3 and 3.4 respectively).

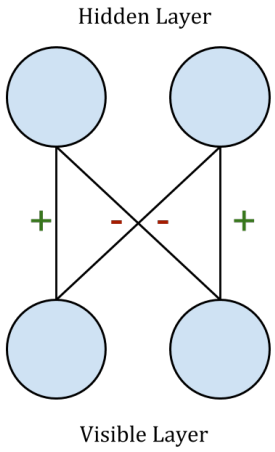


Figure 5.1: The two bit RBM for modelling a single bit on at a time in a two bit pattern. The diagonal weights are negative and the non-diagonal weights are positive.

Architecture and Parameters

Being a trivial dimensionality, the RBMs weights were constructed by hand, meaning that only the ORBMs inference algorithm was being evaluated and not the training of the RBMs plugged into it. This network is picture in Figure 5.1.

Method

This RBM was checked to ensure that it behaved in practice by ensuring it's reconstructions matched the input for a large frequency of reconstructions. In a similar way to the recon-

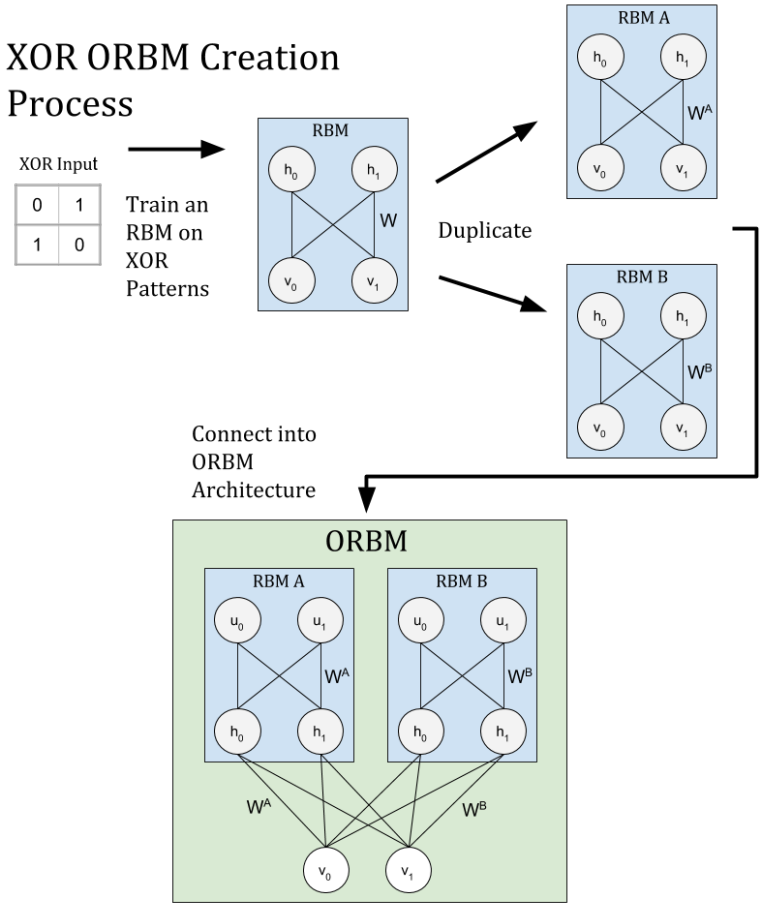


Figure 5.2: The process used to create the ORBM for evalating composite XOR inputs.

structions, free-phase visible pattern samples can be taken from the model and evaluated. In the small dimensions the dreams of the RBM should match the training set. A bar graph of frequencies of dreams was created and the RBM behaves as expected, generating dream patterns that match the training set in the correct proportion. With a valid model established, the XOR RBM was duplicated and plugged into the ORBM structure as illustrated in Figure 5.2.

The inference algorithm was run in the ORBM architecture for various visible inputs, giving two hidden vectors for the two representations h^A and h^B. For each pair of hidden vectors, a reconstruction was created, the process illustrated in Figure 5.3. This process was repeated in a similar way to how the reconstructions were evaluated in the lone RBM, counting the frequency each reconstruction occurred over 1500 runs.

XOR ORBM Analysis

The results of this process are shown in figure 5.4. The ORBM is able to separate the causes with both types of correction, as the model is being duplicated, it produces $[1, 0]$ and $[0, 1]$

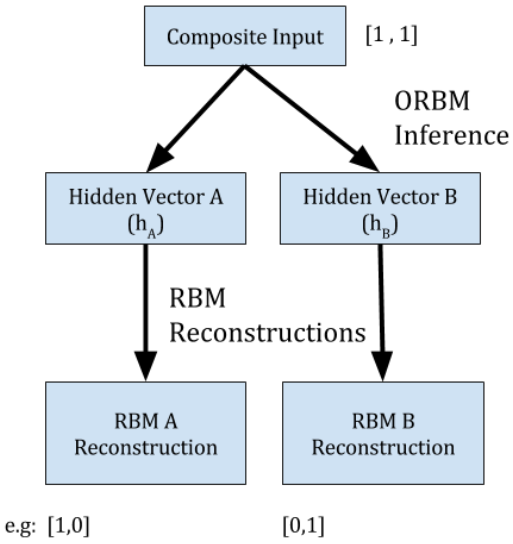


Figure 5.3: The process of generating reconstruction is shown in this diagram.

Visible Input	Expected Reconstructions	
$[1,1]$	$[1,0]$	$[0,1]$
$[1,0]$	$[1,0]$	$[1,0]$
$[0,1]$	$[0,1]$	$[0,1]$

Table 5.1: A Table showing the expected reconstructions from performing ORBM inference with various input patterns. The left and right hand column of the Expected Reconstructions columnn indicate the reconstructions from the left and right RBMs in the ORBM.

approximately half the time and symmetrically $[0,1]$ $[1,0]$ the other half of the time. These reconstructions were compared to one of the RBMs trained to recognised a single pattern being on in two bits. As expected a machine that has been trained to recognise one bit, has no concept of two bits being on and hence the reconstructions show no mechanism for separating the sources. This is illustrated in the bottom right of Figure 5.4.

The results of repeating this process with the input $[1,0]$ yielded successful results. We would hope that ORBM architecture could also handle inference with just a single subject. The results of this are shown in the top right and bottom left of Figure 5.4.

The ORBM was able to extract the target patterns $[1,0]$ and $[0,1]$ given the input $[1,1]$. This was the case for both the Approximated and Full corrections, which gave confidence that the more computationally efficient Approximated correction could be relied on going forward — as for larger datasets it is a lot faster in practice. The generative model also copes with the subject overlapping, which in the two bit case arised when $[1,0]$ is composed with $[1,0]$. In both the Approximated and Full corrections the highest occurring reconstruction is the target $[1,0]$, however there appears to be a lot less confidence.

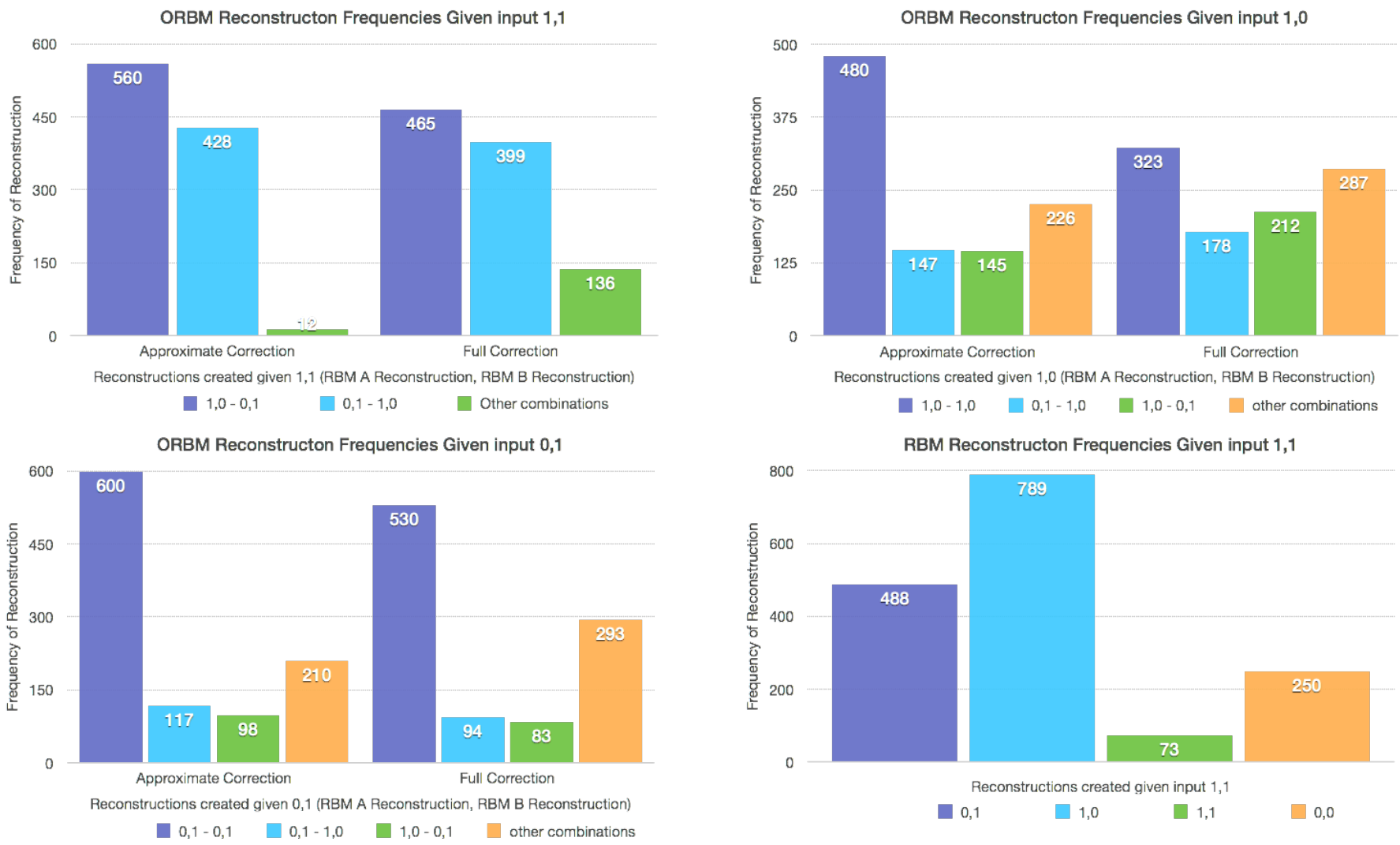


Figure 5.4: A figure showing the result of generating 1000 reconstructions with an RBM and ORBM on various inputs.

5.2 Y bit pattern, X neighbouring bits on

Description

A natural next step from a single bit on in a 2 bit pattern, is moving up to X bits side by side on in a Y bit pattern. In affect this is modelling a X bit subject, in a Y bit pattern. For example if $Y = 4$ and $X = 2$ then a valid inputs are the rows of the following table:

$$dataset = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix} \tag{5.1}$$

This allows for some interesting cases, for instance using the same example as above of $Y = 4$ and $X = 2$, we can examine how the ORBM handles separating interesting patterns. For instance where *subjects* partially overlap such as $[1, 1, 1, 0]$, which is a composition of $[1, 1, 0, 0]$ and $[0, 1, 1, 0]$. This evaluation will explore these interesting cases over a subset of the possible combinations of Y and X, ensuring that the ORBM is able to reconstuct the combination of images correctly.

Architecture and Parameters

- $|Y|$ Hidden units per RBM, any less and I was unable to train the RBM to make the target reconstructions or dreams.
- The RBM was trained with 1000 epochs and a learning rate of 0.002.
- A sample of 10,000 dreams were sampled from the RBMs, and then plotted on a bar graph. The frequency of produced dreams was ensured to be approximately equal and that the dreams should directly match the training set. This is feasible as the ground truth patterns are known.

Method and Results

For a subset of values of $y \in Y, x \in X | 2 < x < y < 10$ the following process was repeated:

1. A single RBM was trained on all possible permutations of X neighbouring bits being on in a Y bit string (as seen in the Matrix 5.1).
2. The trained RBM was duplicated and plugged into the ORBM architecture.
3. For a subset of possible compositions of two patterns, generate reconstructions using the ORBM (and the approximate correction). Where the cases examined are interesting cases such as partial overlapping and completely separate subjects.
4. Finally reconstructions are plotted on bar graphs ensuring they match the expected results.

A subset of the larger (but still small) dimension task results are shown in Figure 5.5. These illustrate the more interesting cases and the ORBM correctly separates the sources in all of them. The top left graph in Figure 5.5 extends the work in section 5.1 by recognising a single bit in a 5 bit pattern and the ORBM is successful. The top right graph illustrates the case of a 1 bit overlap of data made up of 2 neighboring bits. The bottom left graph also exemplifies a 1 bit overlay but this time with a 'wider' pattern of 3 bits. The bottom right graph shows the case where the two 'subjects' are side by side, not overlapping. This is a disparity here that would likely be solved creating more reconstructions.

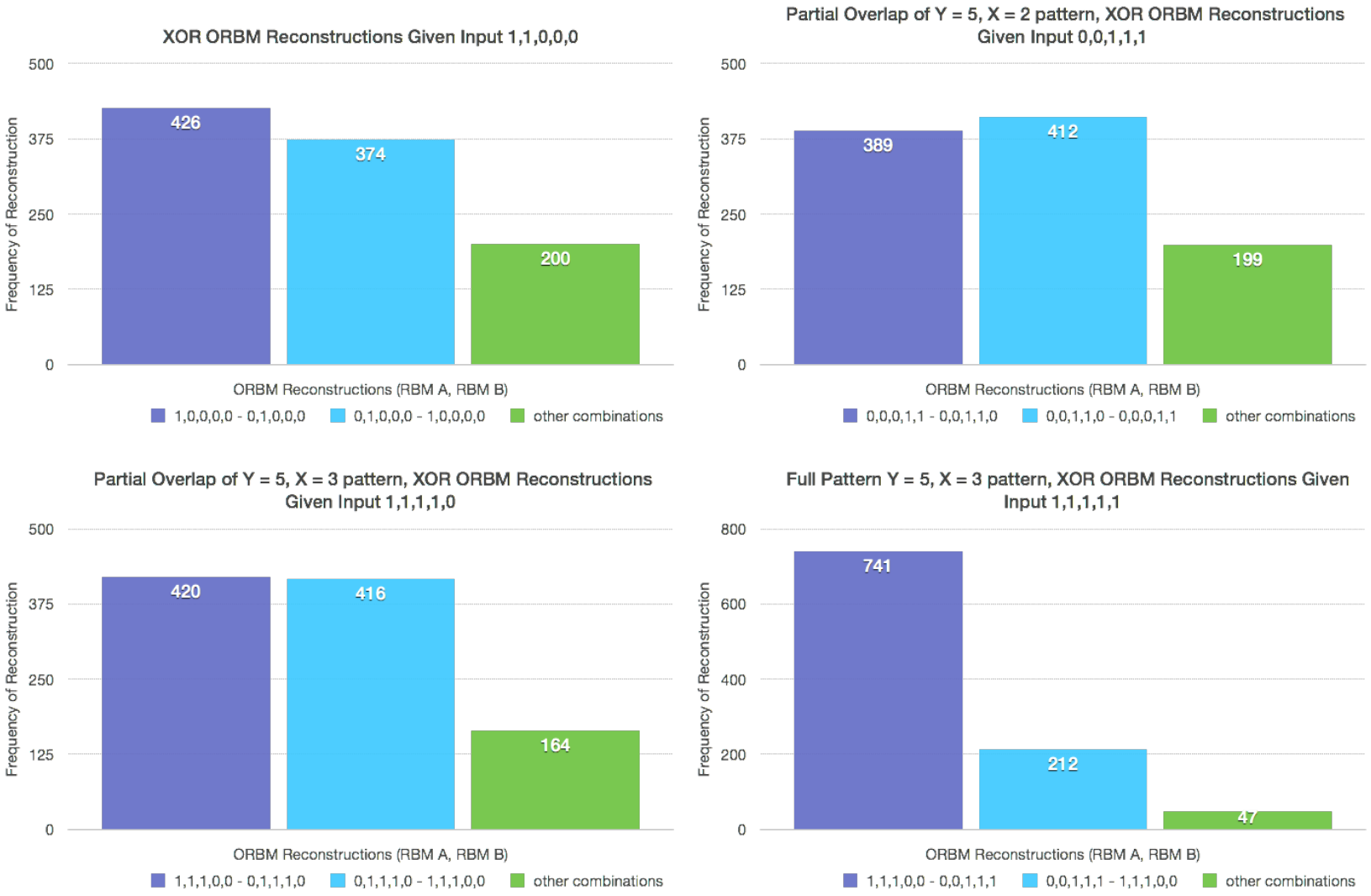


Figure 5.5: A figure showing interesting cases of ORBM reconstructions in Y bit pattern, X neighbouring bits on.

5.3 2D Patterns in a small dimensional space

Description

This task continues to work with a single RBM being duplicated, instead increasing the dimensionality from 10 to 25 in the form of a 5 by 5 pixel image. Then another similar model is introduced so the ORBM has to separate a 2 by 2 square and various sized rectangles.

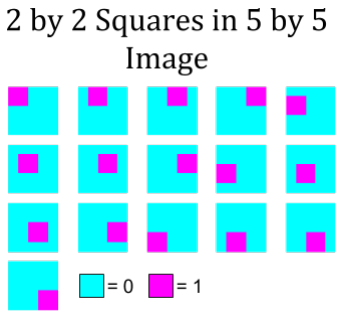


Figure 5.6: A figure illustrating the dataset used for this task. 2 by 2 pixel squares in a 5 by 5 pixel image.

Architecture and Parameters

- Uses a 25 hidden unit RBM trained on a dataset of every permutation of a 2 by 2 square in a 5 by 5 pixel image. This dataset is illustrated in Figure 5.6.
- The trained RBMs reconstructions and dreams were visually inspected, as well as Hinton diagrams to ensure an effective model.
- When performing inference in the ORBM, 500 Gibbs iterations were used. In practice this number is quite large however as the dimensions are small it is not computational intractable, and this gives more confidence that the ORBM is performing to the best of its ability.

Method and Results

The method followed a similar process as previous evaluations: Generate the training data and training and verfiying the RBM to make sure it performs well. Next the whole dataset was composited with every other item in that dataset, and ORBM reconstructions were then generated (using 500 Gibbs iterations when calculating the correction), as were RBM reconstructions. A subset of compelling results from this process were extracted and are shown in Figure 5.7. I only show a single RBM reconstruction here as the same model is being used for both subjects.

Square results

Column 1 Shows two squares sitting side by side. As expected the RBM, only modeling a single square reconstructs a square in the middle of the two. The ORBM is able to successfully extract and reconstruct the separate squares.

Columns 2 and 3 Here we see a single pixel overlap, the ORBM successfully reconstructing the ground truth. The RBM, despite having identical weights to the RBM that is plugged into the ORBM there is no mechanism to separate the sources. The ORBM does not perform perfectly in column 3, in that one if it's reconstructions is correct but the other is noiser than the RBM.

Columns 4, 5 and 6 Much like column 1 we have disjoint subjects, the ORBM successfully separates them, however the RBM performs much, much worse.

This process was repeated except a different RBM was introduced, one that represents a rectangle instead of square. Plugging the original square model and the new rectangle into the ORBM architecture. The results for this aspect are shown in 5.8.

Rectangle results

Columns 1 and 2 These show two instances of the ORBM being able to separate a square and rectangle of different sizes.

Columns 3 This shows a confusing case where the square is completely occluded by the the larger square. The ORBM reconstructions look valid in that it reconstructs the correct shapes, unlike the RBM, however the placing of the 2 by 2 square is in fact incorrect.

Columns 4, 5 and 6 Here we see partially overlapping shapes of various sizes. The ORBM generates less noisy reconstructions compared to the RBM.

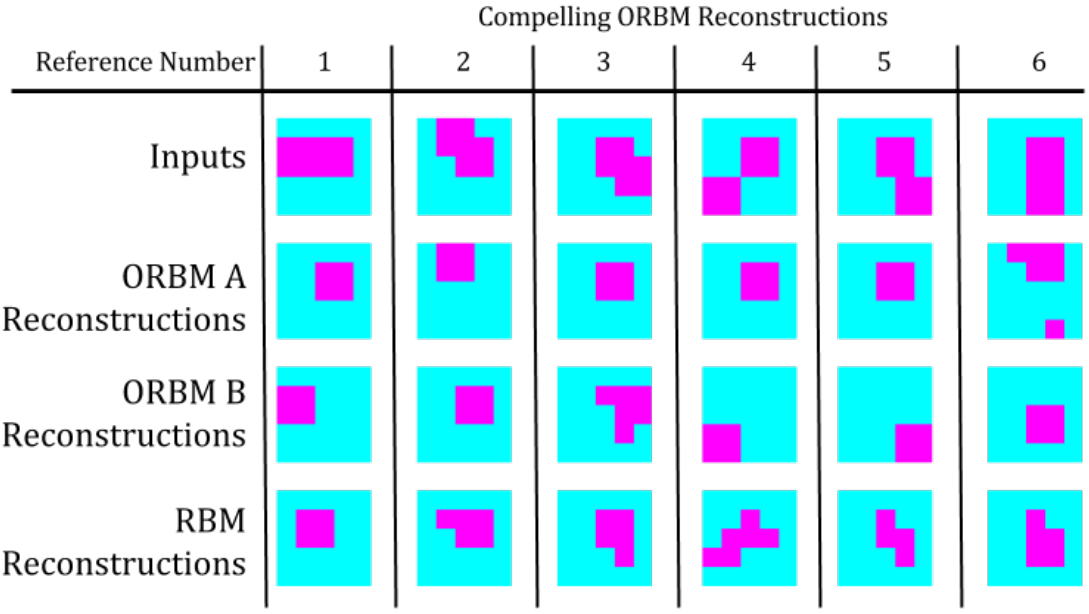


Figure 5.7: Figure illustrating a subset of the the results from ORBM inference on 2 By 2 square images.

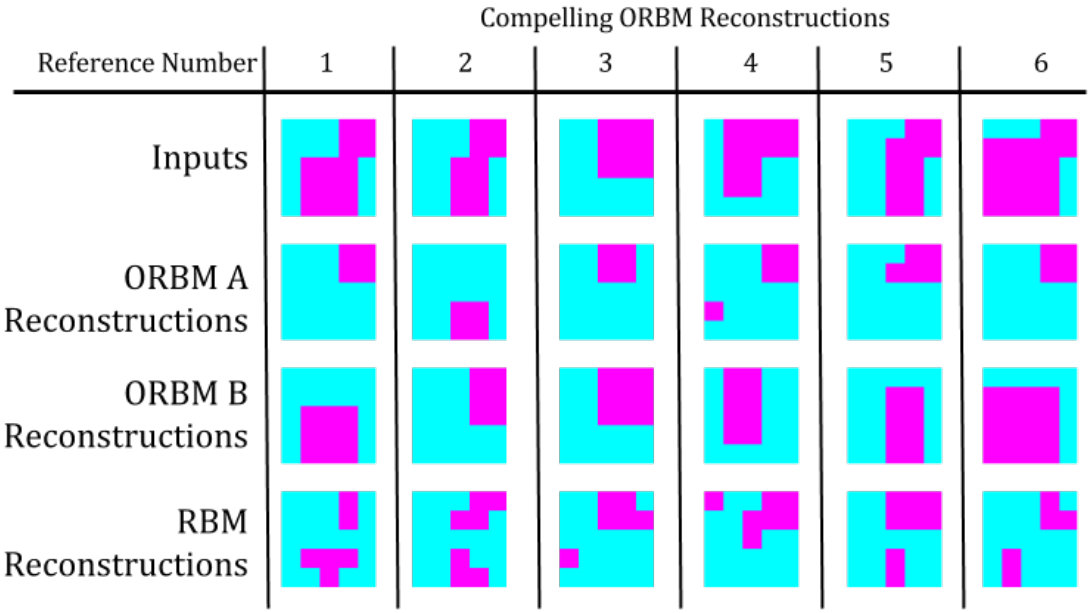


Figure 5.8: Figure illustrating a subset of the the results from ORBM inference on 2 by 2 squares combined with x by y pixel rectangles. Note that there is a different RBM (and corresponding ORBM) being used for different values of x and y in the figure.

5.4 MNIST Digits Mixing Time and Reconstructions

Description

MNIST is a widely used dataset of handwritten digits (0 – 9). This evaluation explored the non-trivial task of given two handwritten digits composited to form one image, how effectively can the ORBM separate the sources compared to the naive RBM?

Architecture and Parameters

- MNIST Digit images are 28 by 28 pixel images, with pixel values between 0 – 1.
- Ten RBMs each with 100 Hidden units trained on 500 examples of each digit respectively. Reconstructions and dreams of these RBMs were inspected by hand to ensure that they resembled the dataset.

Method and Results

For every digit dataset (of size 500), each digit in each dataset was composed with every other digit in every other dataset. Given this set of composite datasets, the corresponding RBMs were plugged in the ORBM architecture and used to create reconstructions.

As an MNIST image has 784 (28 by 28 pixels) features, more consideration is need for how many Gibbs iterations should be used to generate the hidden states (h^A and h^B) given the input. By examining reconstructions at different points in the Gibbs chain we can get a sense of how quickly the chain is mixing. A subset of these chains are illustrated in Figure 5.9. By visually inspecting the reconstructions and the associated hidden activations (the hidden activations are not pictured) at Gibbs iterations 1,2,5,10,50,100, and 500 it can be seen the Gibbs chain mixes very quickly. Viewing these reconstructions as animations it is

easier to see how the reconstructions stabilise and this lead to the choice of using 100 Gibbs iterations, just to be ensure the chain is definitely mixed while not being such a long chain that it is intractable to run.

This diagram also reveals an interesting case where the hidden activations have turned completely off. The reconstruction however appears grey with all values pixel being 0.5. As mentioned in Section 3.2.2, the ORBM does not use a visible bias which would be responsible for making an all zero reconstruction given no activations in the hidden layer.

Given the results of the exploration of the mixing time, 100 Gibbs iterations were used when generating the hidden states (h^A and h^B) for a given input. The RBMs plugged into the ORBM were also used to create standard RBM reconstructions, to compare against the ORBM. Given the ORBMs reconstructions (two per image) and the RBM reconstructions (also two per image), two scores were applied to compare these reconstructions to the ground truth:

Cross Entropy The cross entropy was calculated between the reconstruction and the ground truth.

Cosine Angle The angle between the flattened reconstruction vectors was computed, then negated to give a 'score'. The higher the score the smaller the angle between the reconstruction vector and the ground truth.

These scores can then be summed over each digit and over the entire dataset to find a single score for each digit composed with every other digit forming a 9 by 9 matrix. A cell of this matrix (say j, i) corresponds to the difference between the ORBM score and RBM scores for digits j and i composited together. This entire process was repeated 10 times gain certainty in the results — given the stochasticity of the RBM and ORBM. This process is shown in Algorithm 2.

Data: MNIST Digit datasets, each with 500 examples, 10 RBMs trained on MNIST data 0–9 respectively
Result: Composite Datasets for every digit
for *10 repetitions to gain confidence in results (via the ECS Grid)* **do**
  for *All MNIST digits datasets* **do**
    for *Every digit example in the MNIST digit* **do**
      composite the current digit dataset with every other dataset;
      Generate reconstructions on that dataset using the ORBM, RBM;
      Calculate the Cosine Angle score and Cross Entropy for both the RBM's and ORBM's reconstructions versus the ground truth;
    end
    Sum the scores over every digit example;
  end
  Tabulate the summed scores in a several matrices indexed by the digits being composited;
end

Algorithm 2: The algorithm explaining how the scores matrices were computed.

By finding the difference between the ORBMs reconstructions and the RBMs reconstruction we can create another matrix, in which a positive value represents a 'win' for the ORBMs reconstructions and a negative value represents a 'loss' (where the RBM outperformed the ORBM). These score difference matrices are seen in for the Cross Entropy and Cosine Angle scores in 5.10a and 5.10b respectively, where the colour encodes the score. For

the Cosine score, using the approximate correction, zero composed with every other digit yielded the best results for ORBM relative to the RBM, and nine the converse yielding particularly bad results (worse by a factor of 10 compared to zero). Given this, I plotted the RBMs Score versus the ORBM Score as a scatter plot, where each point corresponds to a score. This allows us to examine if the ORBM is performing worse as a whole, or if in some cases it is performing better than the RBM. These plots for zero and nine are shown in 5.11.

Lastly, the top two scoring ORBM reconstructions are shown in Figure 5.12 for the composition of a four and a five image and a two and four image respectively. In the four composited with five input, the two closest five ORBM reconstruction compared to the ground truth is shown along with the reconstruction from the other model (RBM B) connected to the ORBM. In the two composited with the four image, the two highest scoring four ORBM reconstructions are shown a long with the corresponding RBM reconstructions. The opposite, worst scores are illustrated in 5.13

I selected four composited with five and two composited with four as the four is a challenging digit in that there are strikingly different ways to draw a four. As a human looking at the composition inputs, it is not immediately clear what parts of the image are caused by the four, and which are caused by the five or two respectively.

5.4.1 MNIST Analysis

It is clear that RBM has better scores than the ORBM when summed over the dataset, and surprising the Approximated correction results in better scores than the Full Correction calculation. For the dataset wide scores 5.10 I would expect to see symmetry in that, for example four composited with five, should score the same as five composited with four.

5.5 Evaluation Analysis

It is clear that the ORBM outperforms the RBM in source separation in the smaller tasks, such as XOR. However, as the size of the problems increased I had less confidence until ultimately it performed worse by the scores defined in the MNIST digit dataset. This amounts to the inference algorithm performing worse in large dimensions, with harder tasks. It is not related to the mixing time as that was explored on the same task that the ORBM had the most trouble with. In fact in examining the mixing time it appears it takes very few Gibbs iterations to actually arrive and settle at the wrong answer. Yet in examining the individual scores for the various compositions of the MNIST dataset, the ORBM is performing worse in the majority of cases with only a small subset of examples scoring better than the RBM acting alone. Looking at example reconstructions, I would not say that it is that clear cut that the ORBM is performing worse.

As possible explanation could be that by plugging separately trained RBMs, into the ORBM architecture means that there is no regularisation of weights between the two RBMs. The RBMs were trained independently until they produced accurate reconstructions and dreams that approximately matched items from the dataset, however the process was adhoc and not identical per model. The reason this could be an issue is that the correction adjusts the visible input to the other model by roughly the weight. If the range of weights between the models is drastically different then one model could "overpower" the other. This would explain the effect where the hidden representation generated is entirely zero (no activation) as the other RBM is "taking responsibility" for the entirety of the input.

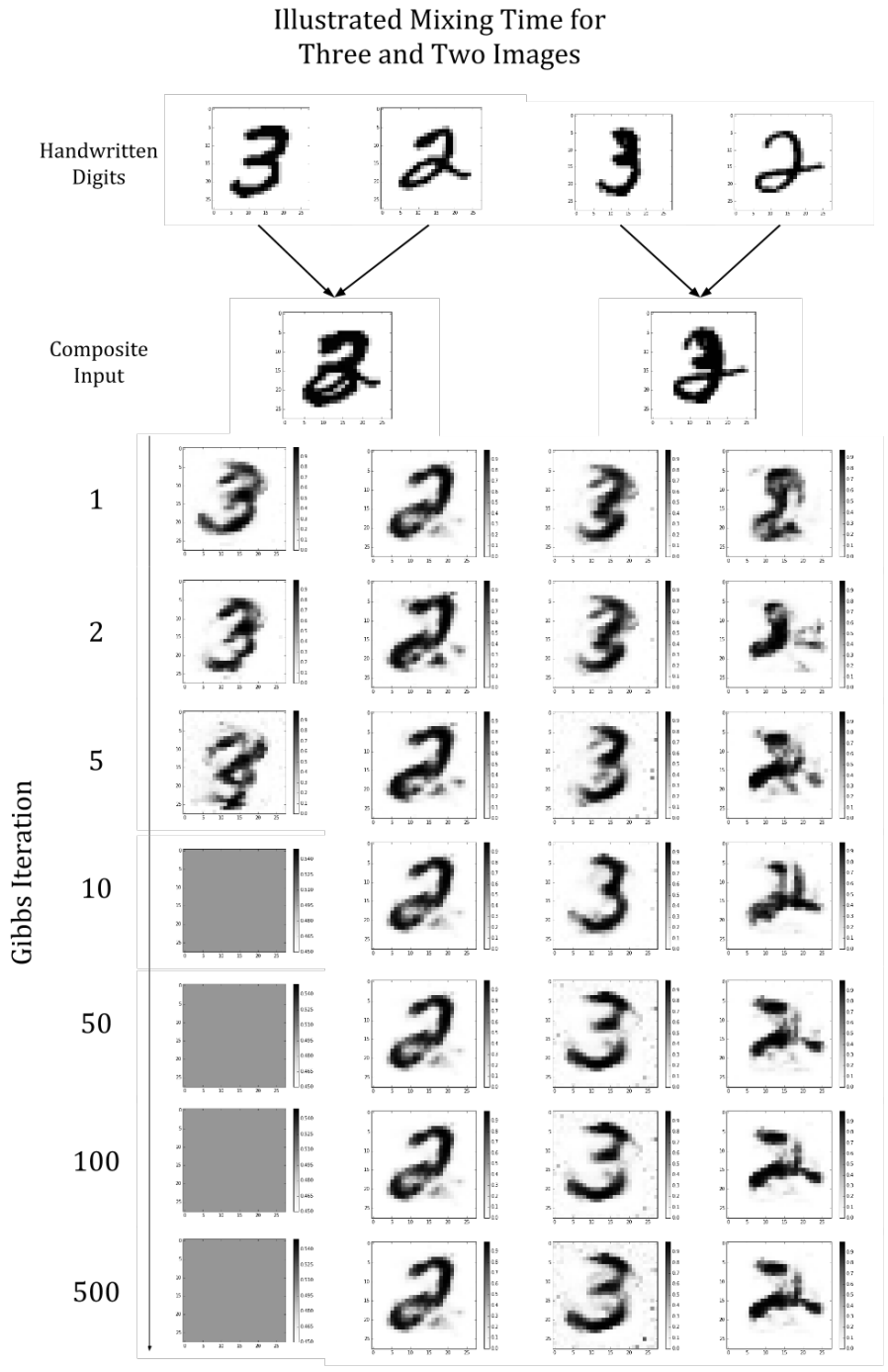
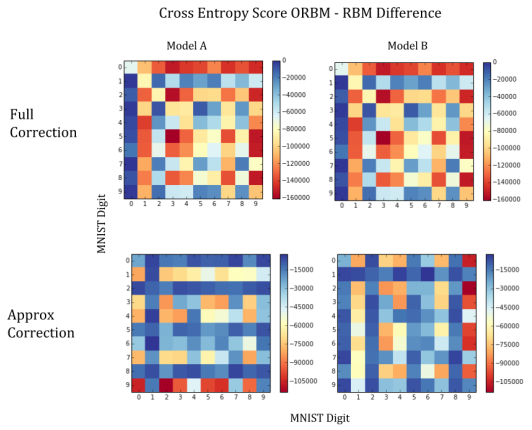
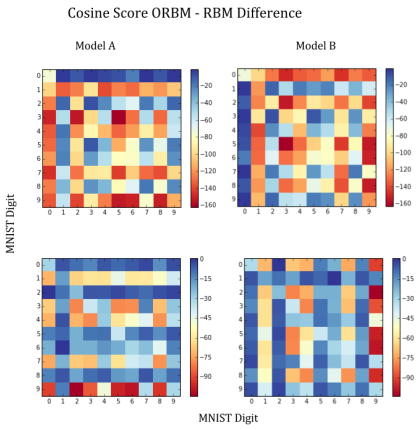


Figure 5.9: A figure visualising how reconstructions change in the ORBM as Gibbs iterations are run for two compositions of digits three and two.

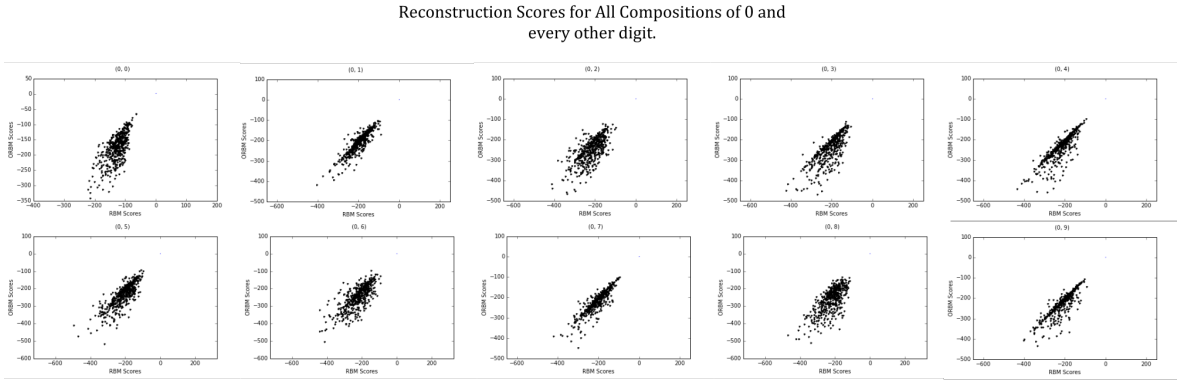


(a) The difference between the RBM and ORBM cross entropy score summed over every item in the composite dataset.

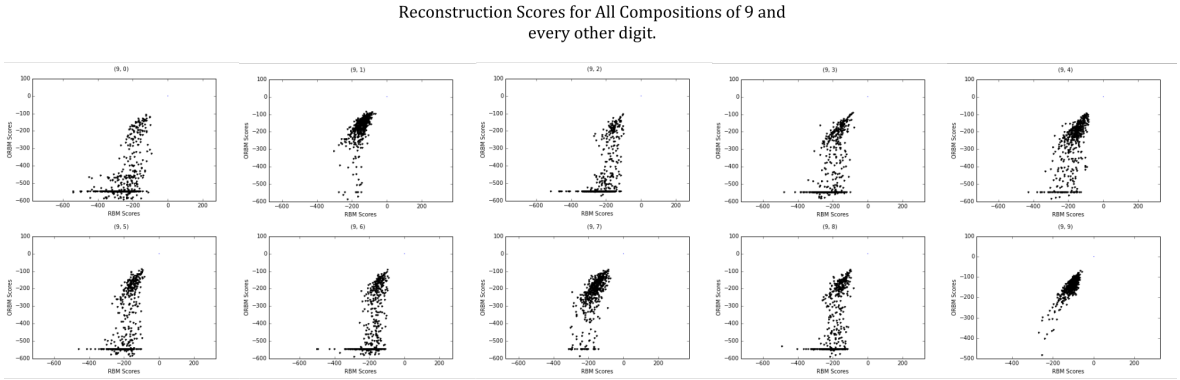


(b) The difference between the RBM and ORBM cosine score summed over every item in the composite dataset.

Figure 5.10: Dataset-wide MNIST Score Results



(a) The cosine score for all digits composited with a handwritten zero.



(b) The cosine score for all digits composited with a handwritten nine.

Figure 5.11: Cosine Score breakdown for the highest and lowest performing datasets, 0 and 9.

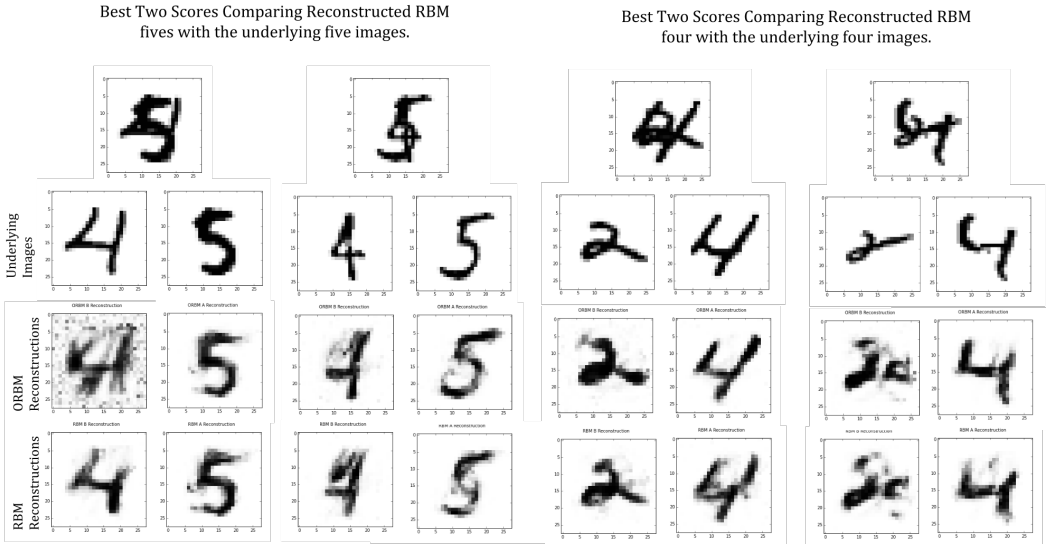


Figure 5.12: The top two ORBM reconstruction results for two different compositions. The highest scoring fives form the basis for the left part of the diagram. The highest scoring fours form the basis for the right part of the diagram.

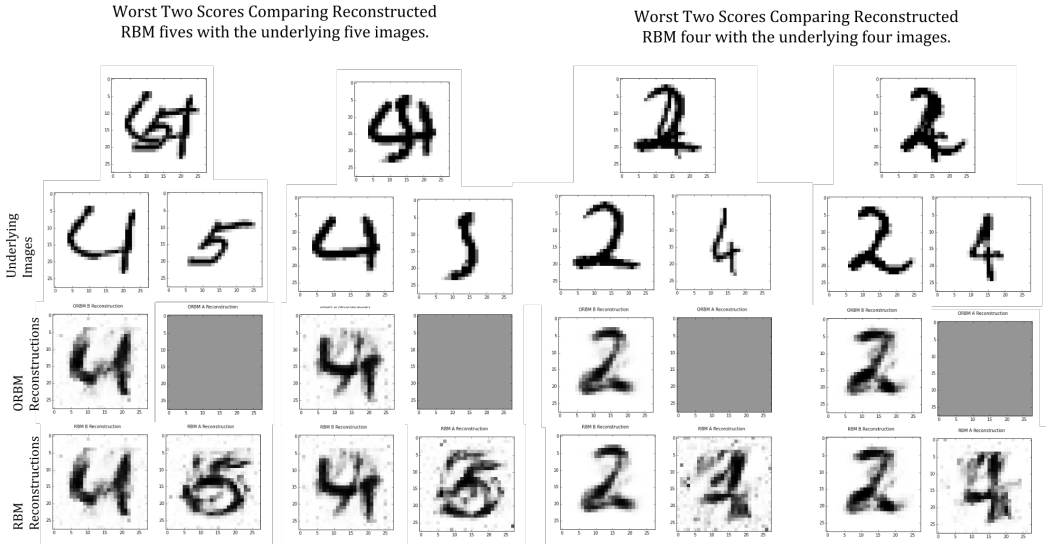


Figure 5.13: The worst two ORBM reconstruction results for two different compositions. The highest scoring fives form the basis for the left part of the diagram. The highest scoring fours form the basis for the right part of the diagram.

Chapter 6

Conclusions and Future Work

6.1 Conclusions

This project aimed to contribute:

$\mathcal{C}1$. The first articulation of the proposed architecture, including the context needed to understand it. It does, by way of this report, particular the second and third chapters.

$\mathcal{C}2$. A Python implementation of the new architecture and algorithm. This was created in order to meet the last contribution. The code is publicly available on Github.

$\mathcal{C}3$. A suite of graded evaluations/tests that explore how the architecture and source separation algorithm work in practice. The project explores how the architecture and algorithm work in practice. However more questions are raised as to why it seems to break down, performing less strikingly in the larger more challenging cases.

The main takeaway from the project is that the ORBM works in smaller problems, and the implications and applications if it does work in larger cases would be huge, so there is a motivation to continue this work going forward.

6.2 Future Work

Frean and Marsland should pursure exactly why the ORBM is not performing better than the RBM on the MNIST dataset. I would suggest ensuring both models have a similar range of weights, to ensure one does not over power the other. Or another approach would be to revert back to using a single model, except the model would be trained on all MNIST digits (0-9). Then a composite input would be shown to this model in the ORBM architecture and the reconstructions should separate the sources of that input. This would alleviate issues of weights having different ranges.

Bibliography

[1] History of SciPy. `http://wiki.scipy.org/History_of_SciPy/`. Accessed: 2015-07-20.

[2] BARBER, D. *Bayesian Reasoning and Machine Learning*. Cambridge University Press, New York, NY, USA, 2012.

[3] CASELLA, G., AND GEORGE, E. I. Explaining the gibbs sampler. *The American Statistician 46*, 3 (1992), 167–174.

[4] DEMPSTER, A. P., LAIRD, N. M., AND RUBIN, D. B. Maximum likelihood from incomplete data via the em algorithm. *Journal of the royal statistical society. Series B (methodological)* (1977), 1–38.

[5] ELIDAN, J. D. D. T. G., AND KOLLER, D. Using combinatorial optimization within max-product belief propagation. In *Advances in Neural Information Processing Systems 19: Proceedings of the 2006 Conference* (2007), vol. 19, MIT Press, p. 369.

[6] FISCHER, A., AND IGEL, C. Training restricted boltzmann machines: an introduction. *Pattern Recognition 47*, 1 (2014), 25–39.

[7] GOLIK, P., DOETSCH, P., AND NEY, H. Cross-entropy vs. squared error training: a theoretical and experimental comparison.

[8] HASTINGS, W. K. Monte carlo sampling methods using markov chains and their applications. *Biometrika 57*, 1 (1970), 97–109.

[9] HINTON, G. Animations showing reconstructions of a deep belief network. `http://www.cs.toronto.edu/~hinton/adi/` accessed 17 Aug 2015.

[10] HINTON, G., DENG, L., YU, D., DAHL, G., RAHMAN MOHAMED, A., JAITLY, N., SENIOR, A., VANHOUCKE, V., NGUYEN, P., SAINATH, T., AND KINGSBURY, B. Deep neural networks for acoustic modeling in speech recognition. *Signal Processing Magazine* (2012).

[11] HINTON, G. E. To recognize shapes, first learn to generate images. *Progress in brain research 165* (2007), 535–547.

[12] HINTON, G. E., OSINDERO, S., AND TEH, Y.-W. A fast learning algorithm for deep belief nets. *Neural Comput. 18*, 7 (July 2006), 1527–1554.

[13] HINTON, G. E., AND SALAKHUTDINOV, R. R. Reducing the dimensionality of data with neural networks. *Science 313*, 5786 (2006), 504–507.

[14] HINTON, G. E., AND SEJNOWSKI, T. J. Analyzing cooperative computation. *Proceedings of the 5th Annual Congress of the Cognitive Science Society* (may 1983).

[15] HINTON, G. E., AND SEJNOWSKI, T. J. Parallel distributed processing: Explorations in the microstructure of cognition, vol. 1. MIT Press, Cambridge, MA, USA, 1986, ch. Learning and Relearning in Boltzmann Machines, pp. 282–317.

[16] HOPFIELD, J. J. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences 79*, 8 (1982), 2554–2558.

[17] HUANG, G. B., LEE, H., AND LEARNED-MILLER, E. Learning hierarchical representations for face verification with convolutional deep belief networks. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on* (2012), IEEE, pp. 2518–2525.

[18] JENSEN, C. S., AND KONG, A. Blocking gibbs sampling in very large probabilistic expert systems. *Internat. J. HumanComputer Studies 42* (1995), 647–666.

[19] KIM, J., NAM, J., AND GUREVYCH, I. Learning semantics with deep belief network for cross-language information retrieval.

[20] LECUN, Y., AND CORTES, C. The MNIST database of handwritten digits.

[21] LIU, P., HAN, S., MENG, Z., AND TONG, Y. Facial expression recognition via a boosted deep belief network. In *Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on* (2014), IEEE, pp. 1805–1812.

[22] MCAFEE, L. Document classification using deep belief nets.

[23] NEAL, R. M. Connectionist learning of belief networks. *Artificial intelligence 56*, 1 (1992), 71–113.

[24] NOULAS, A. K., AND KRSE, B. Deep belief networks for dimensionality reduction. In *Belgian-Dutch Conference on Artificial Intelligence, Netherland* (2008).

[25] PEARL, J. Reverend Bayes on inference engines: A distributed hierarchical approach. In *Proceedings of the American Association of Artificial Intelligence National Conference on AI* (Pittsburgh, PA, 1982), pp. 133–136.

[26] PEARL, J. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1988.

[27] PEDREGOSA, F., VAROQUAUX, G., GRAMFORT, A., MICHEL, V., THIRION, B., GRISEL, O., BLONDEL, M., PRETTENHOFER, P., WEISS, R., DUBOURG, V., VANDERPLAS, J., PASSOS, A., COURNAPEAU, D., BRUCHER, M., PERROT, M., AND DUCHESNAY, E. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research 12* (2011), 2825–2830.

[28] SALAKHUTDINOV, R. *Learning deep generative models*. PhD thesis, University of Toronto, 2009.

[29] SALAKHUTDINOV, R., AND SALAKHUTDINOV, R. Learning and evaluating boltzmann machines. Tech. rep., 2008.

[30] SMOLENSKY, P. *Foundations of harmony theory: Cognitive dynamical systems and the subsymbolic theory of information processing*. Parallel distributed processing: Explorations in the ..., 1986.

[31] ZHOU, S., CHEN, Q., AND WANG, X. Discriminative deep belief networks for image classification. In *Image Processing (ICIP), 2010 17th IEEE International Conference on* (Sept 2010), pp. 1561–1564.