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

TODO WRITE THE ABSTRACT TODO WORDING I or We????

Acknowledgments

Any acknowledgments should go in here, between the title page and the table of contents. The acknowledgments do not form a proper chapter, and so don't get a number or appear in the table of contents.

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 face and illumination are complex and can vary greatly, but they are fundamentally acting independently of each other up until they compose to form the image.

A form of Deep Neural Networks, Deep Belief Networks, have achieved state of the art performance in facial recognition, 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 no explicitly attempt to represent these *systems* indedependently. This project takes steps toward representing complex causes separately with the primary task of decomposing joint images (more generally, vector based data).

Separating faces and illumination is too challenging for this project, there is no way to verify/evaluate that the new approach is working as 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, dimensionality reduction, natural language recognition, Document classification, Semantic Analysis.TODO CITE: Despite a DBN's expressiveness, there is no way to extract independant sources, the model instead learns how to represent the complex combination. The complex combination of sources is inheriently 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 face and illumination example, with 1000 possible faces and 1000 possible illuminations. If we only observe a combination of faces and illuminations there is 1000000 possible combinations we need to represent. It is more succint to treat faces and illuimination as independent *system*, resulting in 2000 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 handrwritten digits [2].

1.3 A Proposed Solution and Contributions

Frean and Marsland propose an alternative architecture to that of an RBM, that aims to encode two complex sources independantly. 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.

$\mathcal{C}3$. 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 only 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 Probablistic Graphical Models, a substantial amount of background is required culminating with the new approach being derived. An robust understanding of these concepts is required for this project to implement and design appropriate evaluations for the proposed architecture new approach.

2.1 Generative Models

This project works with nueral network based generative models. Generative models are a powerful way to model data. The rational behind them being that we aim to learn a model that can both create the training data represent it, and reconstruct it using it's learned representation. Generative models can map input data from raw values to higher level features. Hinton [4] 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 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

There are two classes of Probablisitic 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 [13]. 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 often referred to as a Belief Network or Bayesian Network where it's causal dependencies are expressed as a conditional probability table or 'factor'. For example in figure 2.1 the probabilities of A being in a given state are dependent on B. The joint distrubution over A and B, as in equation 2.1, can be expressed as:

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

TODO WORDING consider removing — Note that a normalisation is not needed in DPGMs as the conditional probabilities enforce that the factors sum to 1.

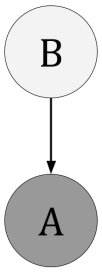


Figure 2.1: Minimal Directed PGM, showing an observed variable 'A' and it's hidden cause 'B'.

2.2.1 Explaining Away in DPGMs

TODO WORDING A common task in generative models is given a known parent unit (a cause), is inferring the probability of children variable dependent on that parent being in a given state. In directed PGMs this proves trivial to calculate. The opposite task of inferring the state of causes, given the effect of that cause is also a desirable taskTODO CITE: . It becomes problematic in DPGMs as these causal relationships gives rise to the effect of 'Explaining Away'. The canonical example `Burglar, Earthquake, Alarm Problem` is a exemplifies this effect effect[1] and is illustrated in figure 2.2. Knowledge of the state of the `alarm` makes

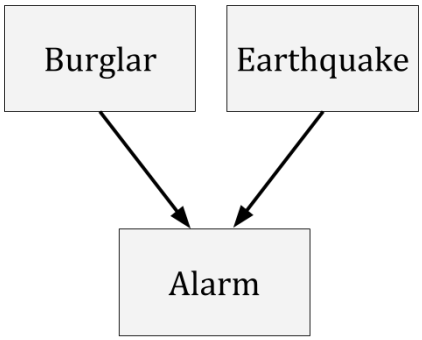


Figure 2.2: The famous Burglar, Earthquake, Alarm network showing a minimal case of explaining away.

`burglar` and `earthquake` dependent. The `alarm` is the observable variable here (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 exemplify equation 2.1) in probabilities where A, B and E are the states of `alarm`,`burglar` and `earthquake` respectively:

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

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. TODO WORDING Simplify — This network is called a Sigmoid Belief Network (SBN) as the probability of a variable x_i being 1 that 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 and ensures that the output is a valid probability (between 0 and 1). SBNs take a naive approach to causes, where each hidden unit represent 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 nontrival complexity to performing inference in Undirected PGMs TODO CITE: , a sum over all 2^N configuratiions of x is required.

On the other hand, Undirected PGMs do not capture causal relationships. Calculating 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 UPGM expressed as nueral network is referred as Boltzmann Machine or Markov Field, 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 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 [8]. Hinton and Sejnowski also proposed the Boltzmann machine in [6]. An example Boltzmann Machine is shown in figure 2.3. As shown in this figure 2.3, 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 growsTODO CITE: .

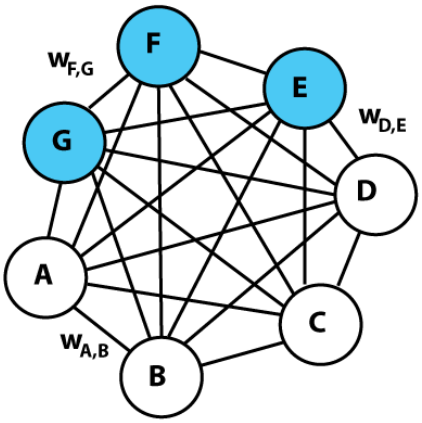


Figure 2.3: A Boltzmann Machine, the blue shaded nodes representing the observed variables, and the non-shaded nodes the latent variables.

2.3.2 Restricted Boltzmann Machines

TODO WORDING A Boltzmann Machine's architecture can be altered to alleviate inference shortcoming. The restriction, originally proposed by [14], and then later revitalised with a training algorithm that operates on the deeper architecture of the DBN [6]. 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. The collection of hidden units, forming a layer are referred to as the `hidden layer`. The collection of visible units are referred to as the `visible layer`.

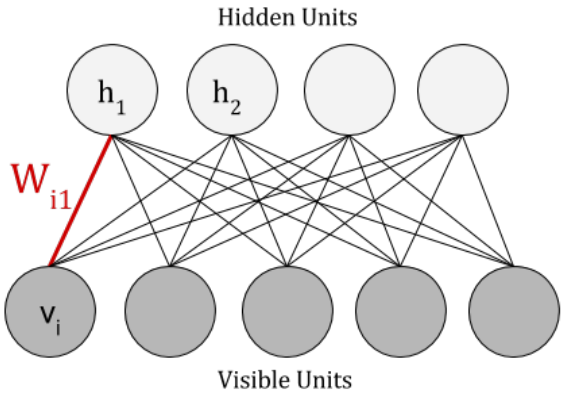


Figure 2.4: An example Restricted Boltzmann Machine with four hidden units, and five visible units. Note that the edges between units are not directed - representing a dependency not a cause.

The probability of the RBM 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 from is intractable to calculate analytically. 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 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 refferred 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. TODO CITE: EM

TODO WORDING inference for our purpose is used for (somehting about not training)

2.4.2 The sampling technique: Gibbs Sampling

Gibbs sampling is a special case of Markov Chain Monte Carlo [3], a technique for drawing sampling from a complex distribution. Sampling from the probability mass (or 'joint distribution') of a generative model is a common use case for Gibbs sampling [12].

Gibbs sampling explores the desired probability distribution, taking samples of that distribution's state, allowing iterations of exploration between drawing of a sample to ensure that the samples are independentTODO CITE: . The process of taking a step between states is referred to as a `Gibbs iteration` or a `Gibbs Step`. Formally the algorithm is descibed 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 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`. When Hinton when proposed a fast training for algorithm for RBMs and DBNs [5], Gibbs sampling is used for performing inference in RBMs 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) 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 calcuated by propagating probabilites 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 [11] 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 Directed Acyclic Graph, Belief Propagation is intractable to use as the number of variables growTODO CITE: . 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 [9]. The issue being that the Gibbs chain takes too long to mix [10], which arises from the the 'explaining away effect' [5]. Calling back to the example in section **??**, instead of a single

`burglar` or `earthquake` becoming dependant given the `alarm`, there is more in the realm of hundreds of burlgars and earthquakes all suddenly depedendant.

Being able to efficiently sample from $P(h|v)$ is also required for training generative modelsTODO CITE: em 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. However, it was shown that given unlimited training time Boltzmann Machines could be trained, out performing the state of the art models of the time TODO CITE: This.

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. TODO CITE: neal1992:connectionist

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 variables. This is illustrated in figure 2.4 the hidden unit h_1 is not dependent on h_2 wether or not 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 dependant. Secondly, by removing the recurence present in Boltzmann Machines, it reduces the expresiveness of the RBM network while making the RBM useable in practice as the Gibbs sampling process can stop after one Gibbs stepTODO CITE: .

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 $\tilde{h}'$
- Using this hidden state, a visible state is then generated, $\tilde{v}'$, by sampling from $P(\tilde{v}'|\tilde{h}')$. 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 HintonTODO CITE: CD-1Paper showed that 1 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.

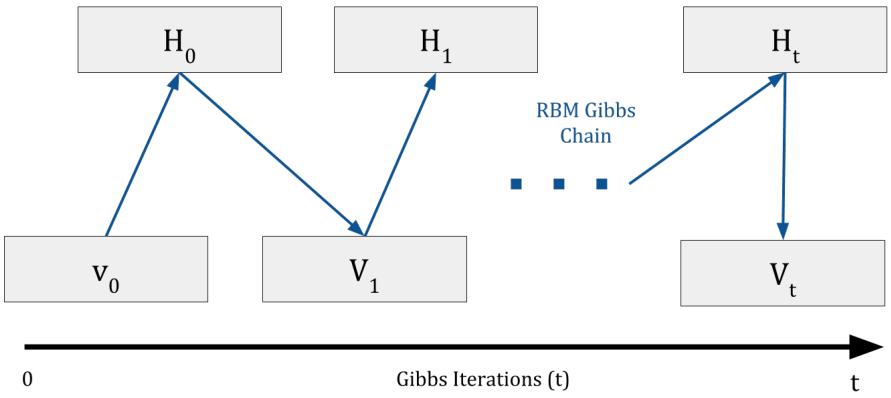


Figure 2.5: A figure illustrating a Gibbs chain where left to right indicates a Gibbs iteration. Note this is *not* a PGM.

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: TODO CITE: Gibbs sampling would be good here.

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

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 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:

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

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

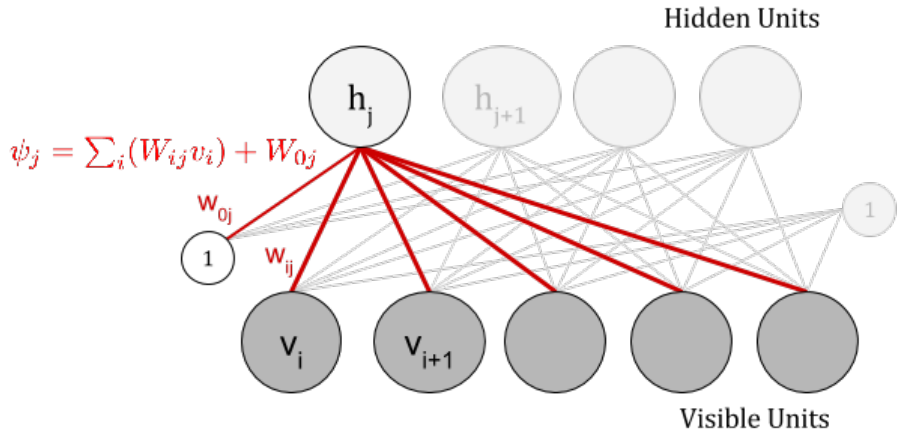


Figure 2.6: A diagram showing ψ_j, the weighted sum into the jth hidden unit. Note that W_{oj} is the hidden bias, represented as a unit that is always on with a weight into each hidden unit.

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. 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. This has applications in increasing the size of a dataset by introducing variation by generatin these faux inputsTODO CITE: .

RBM Fanstasies: 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, many (greater than 100) Gibbs iterations with no input pattern clamped allows the reconstructions to explore the probability mass that has been built by the model during training. Sampling from these wanderings creates what are refferred to as 'fantasies' or 'dreams'. These give a sense of what the model has learnt, and can act a smoke test for if the model has actually capture anything. TODO CITE: (TODO-CITE-PAPER-WITH-MNIST-DREAM-EVALUATION, they were crappy).

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. TODO CITE: Hintons paper on unrolling the Gibbs chain. 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.

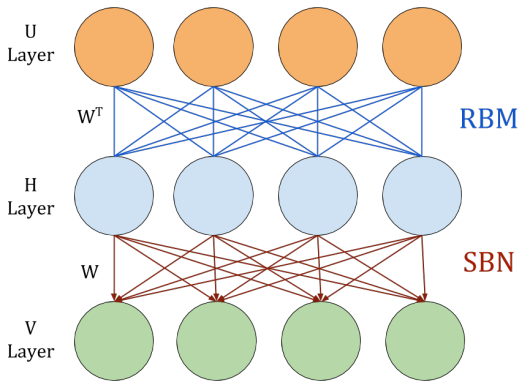


Figure 3.1: A diagram illustrating 'unrolling' an RBM by one Gibbs iteration. Note the connections between H and V layers are now directed.

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 TODO CITE: as described in the section where I talk about dreams... 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.

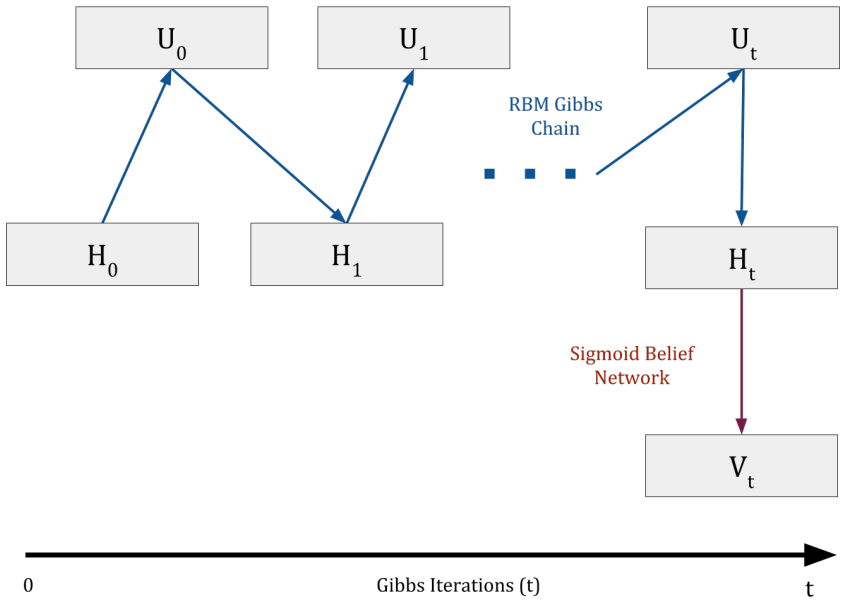


Figure 3.2: A diagram showing Ancestral sampling in the equivalent network, where normal sampling in the top 2 layers is performed until Gibbs iteration t the hidden state is pushed through the bottom layers Sigmoid Network.

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 the same as sampling h from the same RBM and using a SBN for the last step.

There is 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:

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

So far we've figured out $\log P^\star(h)$ for the RBM that is the 'top layer'.
Therefore 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 orignal U and H layer via a SBN to form V. This architecture is illustrated in figure 3.3, where A and B are used to denote the two independant 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. This architecture is simpler in practice as the U layers can be captured in the inference algorithm.

By building on RBMs and SBN we can leverage existing algorithms and work on RBMs and also the potential for extending this work to deeper architectures is 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.

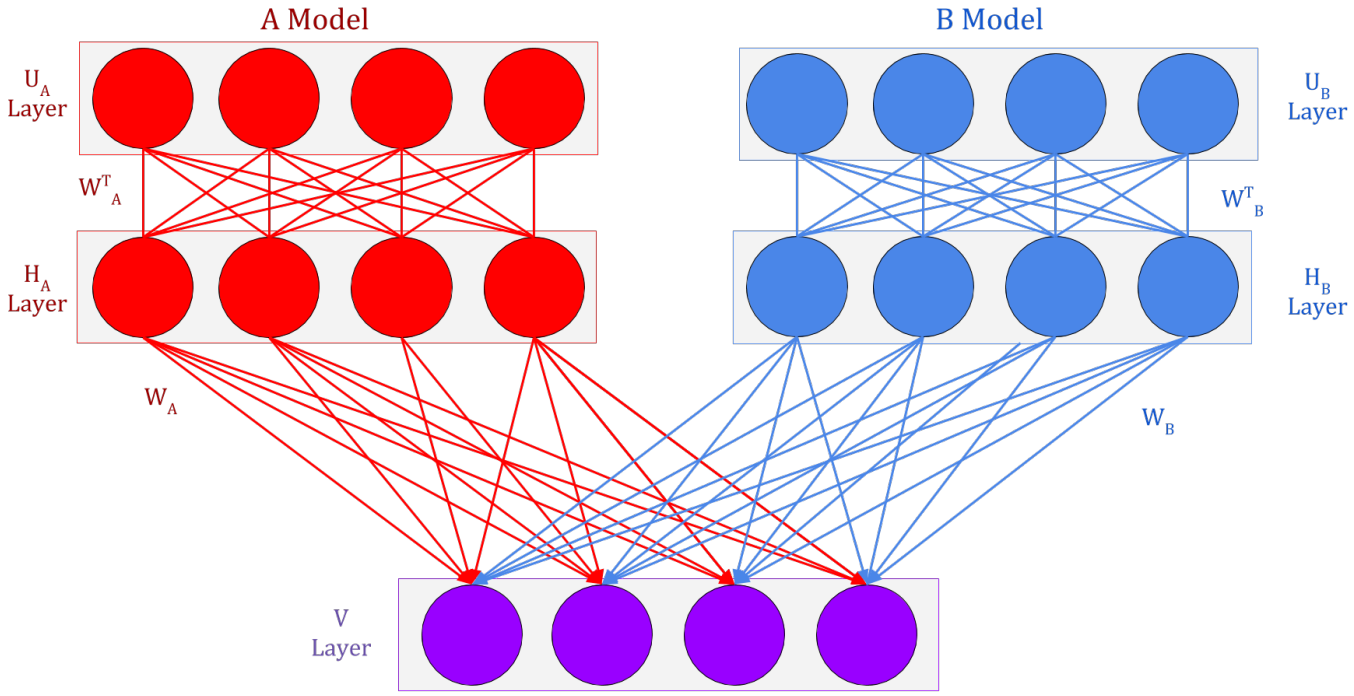


Figure 3.3: The full ORBM architecture, where A and B are the two causes that combine to form the data.

3.2.2 Gibbs Sampling in the ORBM

Ancestral Sampling in the ORBM

Ancestral sampling in an RBM involved running a Gibbs chain and then taking the last visible in that chain. To perform ancestral sampling in the ORBM we can leverage the generative power of the RBM 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 $P(h_j)$ is given by $P(h_j) = \sigma(\psi_i)$, 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 **??**. 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. They 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 $\phi = \log\frac{1+e^\phi}{1+e^{-\phi}}$ (Marcus' magic identity), 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 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 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 correction.

It is useful to see the correction 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 $h_j^A = 0, 1$ These are plotted in figure 3.4. From these plots we can see that this function has defined 'plataus' where the height of these platuas is $+_-$ the weight respectively. The correction essentailly adjusts the weight between v_i and h_j allowing it to be turned off, turned on, intensified, deintensified, 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:

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

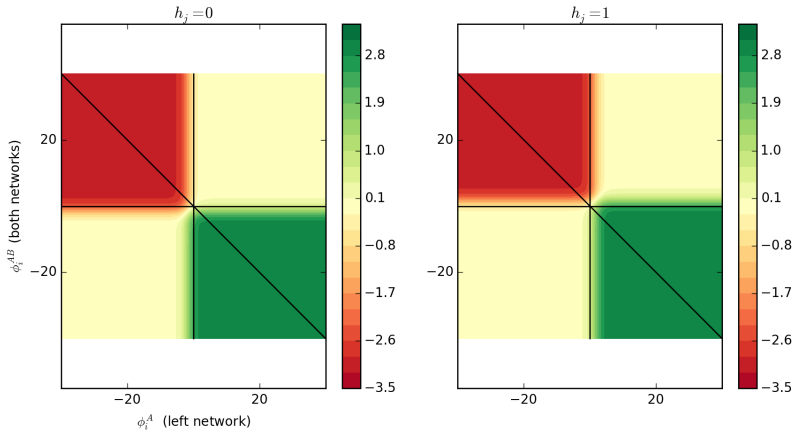


Figure 3.4: A diagram that shows the structure of the correction over the weighted sums into the hiddens of one of the RBMS versus both. Note the platuaes that are $+ -$ *weight* in maganitude

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.

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) uses 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 bugs in the implementation are a threat to validity.

4.1 Implementation Design

- Ordering of evaluation tasks made unit testing, with hand made test cases possible. Also less risk, remove the uncertainty around the RBM.
- testing approach

4.1.1 Language Choice

TODO WORDING The implementation of the ORBM and inference algorithm is in *Python*, with *Matlab* and *Java* being the other languages 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 environmentTODO CITE: and has a lot of online resources as well as machine learing papers TODO CITE: 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 sucinct syntax and shallow learning curveTODO CITE: , as well as being the langauge of choice of my supervisor. TODO WORDING This is favourable as I can have support for translating the algorithm into python correctly. Through libraries that supply wrappers for C-bindings, efficient code can be written but overal being an interpreted language Python is slower than Matlab and JavaTODO CITE: . Three main factors influenced the choice for 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 expeirence 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 contendors in this factor, as a lot of matlab machine learning code is avaliable with supporting papers TODO CITE: . Python has libraries such as NumPy, SciPy and Matplotlib TODO CITE: which mirror Matlabs 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 Sklearn's TODO CITE: allowed for some performance improvements.

Ease of Evaluation Python provides strong plotting library that is inspired by Matlab and R — MatplotLib TODO CITE: . Matplotlib is interoperable with the NumPy library, meaning that turning results into plots was really easy.

Also the evalation requires repeateldy running a test to gain more confidence in the stochastic result. This is made possible by the ECS-Grid TODO CITE: . Python is supported with very minimal extra work required. A simple script that manages input and output is required. However the Java support on the ECS grid requires significantly more setup TODO CITE: . TODO WORDING This seemed like an unnesccariy risk to introduce to the project at the later stage in the project where evaluation is 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 CorrectionTODO CITE: Equation ... By using Python, multiple inheritance could be leveraged to achieve this composability. For instance adding continuous pixel value support 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 is trivial to compose samplers, which was required for the ORBM samplers.

4.2 Evaluation Design: Evaluating RBMs and ORBMs

A challenge faced by this project and work with RBMs is that they are non-trivial to evaluate TODO CITE: . Being unsupervised black box, one cannot merely inspect the hidden units

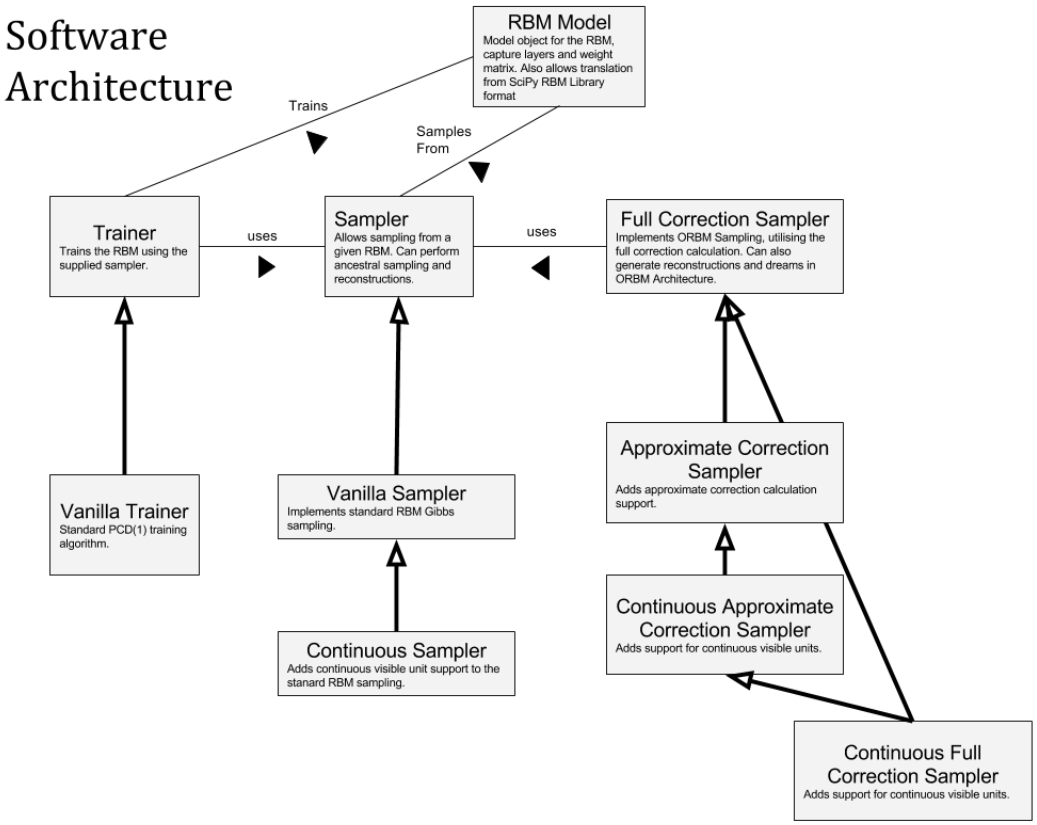


Figure 4.1: A figure showing the architecture for the implemented test suite in UML notation.

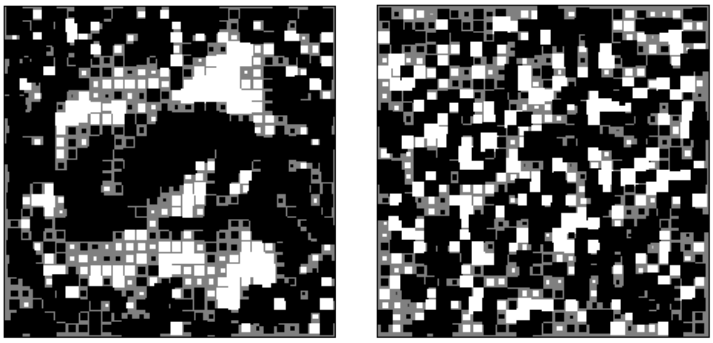
for an input and be sure that a good model has been learnt. As I plug RBMs into the ORBM architecture it is important for the overal results that the RBMs are well trained.

4.2.1 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. This was first used by Hinton in the context of Boltzmann Machines in [7]. 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.

4.2.2 Reconstructions As a measure of Performance

- Cite literature where reconstructions are used, starting from early RBM training papers by Hinton, up to more recent in deeper networks - the goal being to show the reader that reconstructions are used well in practice.



(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.

4.2.3 Evaluating RBMs: Problem dependent

Evaluating RBMs is problem dependant, we can examine different the different approaches avaliable by splitting problems into two cases:

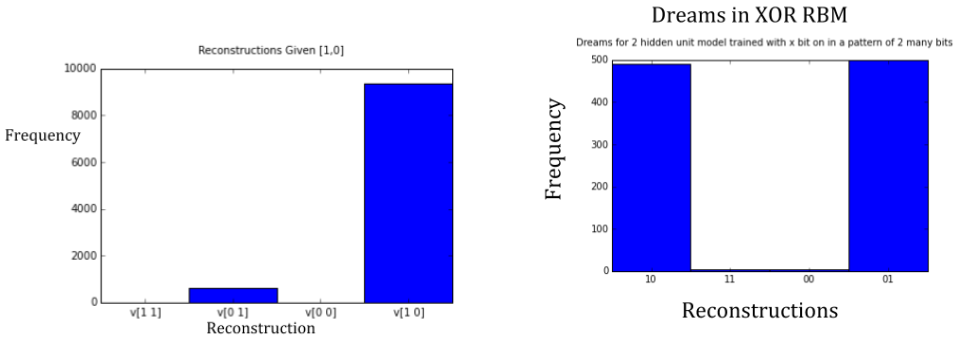
Small Dimentionality When the dimentionality of the input is very small (< 10) and all possible inputs are known we can perform analytical evaluations of the RBM. By clamping the visible units of the RBM to each input in the training set create reconstructions can be created. This process can be repeated for each input showing the frequency that each reconstruction occurs on a bar graph. We can then ensure that the RBM is reconstructing the input perfectly. For instance, an RBM trained on two bit XOR and clamped to the input $[1, 0]$ bar graph of reconstructions illustrated in figure 5.2.

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 dreams visible patterns from the training set approximately an equal amount. Using the two bit XOR example once more, an example bar graph is shown in figure5.4.

Large Dimentionalities In non-trivial cases, with larger datasets, reconstructions can be inspected and compared to the training dataset, but empirically detecting if a model is trained is difficult given the unsupervised, black box nature of RBMs. Alternatively a 'score' can be assigned to each item in the trainin set, based on how well the RBM can reconstruct it. However in large cases your training set will not be complete — you will not have every possible valid image like you would in a case where your image is 2 pixels. Dreams can also be examined in larger cases, but again you cannot be as sure as in the smaller cases that you have a well trained RBM.

4.2.4 Evaluation Methods

- Same models
- Growing dimensionality, bit strings to 2d to MNIST
- Allows comparison of Approx corrections in smaller scale - fail there don't continue using them...



(a) Example reconstructions for the Handcrafted RBM, For 10000 independant reconstructions given the input $[1, 0]$.

(b) Dreams in the XOR RBM, note how only the training data is present.

Figure 4.3: Bar graphs exmplifying the use of reconstructions and dreams for small numbers of RBMs.

TODO WORDING My TODO WORDING x experiements all followed the same high level process, working from trivial cases to more challenging tasks. The reasoning 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.

Growing Complexity of tasks

The following evaluations aimed to evaluate the ORBM and it's inference algorithm by examining reconstructions given a multi-cause input. An example of a muli-cause input with two quadrilaterals is shown in figure 4.4. The optimal reconstructions are equivalent to the two images that combined form the input.

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

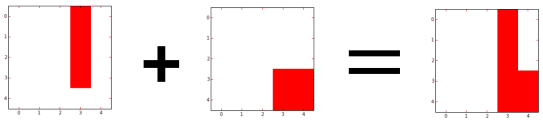


Figure 4.4: A figure illustrating two five by five pixel images combining to form a composite/multicause input. The images are binary.

To be able to use this reconstruction based evaluation, knowledge of the ground truth was required. This was 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 Choice of Evaluation Datasets

- 2 Bit XOR - It's the minimal case. Can examine reconstructions and dreams/fantansy emperically. Also can check that code performs as expect by manually calcating ex-

pected values for the algorithm.

- X Neighbouring Bits On in Y Bit Pattern - A natural next step from 2 bit XOR. Still trivial to train an RBM to represent, quick feedback to ensure the algorithm works. Allows comparing the different approximations tractable as number of hidden units is small enough.
- Square Patterns in a 5 by 5 Pixel Image - Another natural step from neighbouring bits, trival to construct a dataset. Very easy to compose. Can use the same model for both models in the ORBM Architecture.
- Rectangles in a 5 by 5 Pixel Image - Next step, builds on the previous test but adds a different model for each source. Also different shapes of model can be used.
- Continous Rectangles (Pixel values 0 to 1) in a 5 by 5 Image - Another variation working towards the MNIST dataset, same as the previous but has continous visible pixels. Allows to different intensitities, interesting cases where the images overlap and increase in intensity.
- MNIST Handwritten Digit Dataset - Pixel values from 0 to 1, 28 by 28 pixel image (784 pixel features.) Used extensiviely in previous studies. Gives me confidence that RBMs can learn these representations. Also allows metaparameters don't have to be tweaked as much as studies already have found reasonable values. Non-trivial to evaluate, but more of a real world case.

4.2.6 Mixing Time Evaluation

- List the types Traceplot, Densityplots, Gelbin and Rubin multiple sequence diagnostics
- Examine hidden state, reconstructions as gibbs iterations occur.

Chapter 5

Evaluation

5.1 Starting from the minimal case, two bit XOR

Description

The starting point for the evalatuion was examining the ORBM reconstructions in a two bit pattern, where two of the same model were combining to form the data. This model made one bit on in a two bit pattern at a time. That is it could either make $[1,0]$ or $[0,1]$. The training set is the two bit XOR truth table. Two bit case also provides the minimal case to compare the full correction calculation, versus that of the proposed approximation (see equation 3.3).

Architecture and Parameters

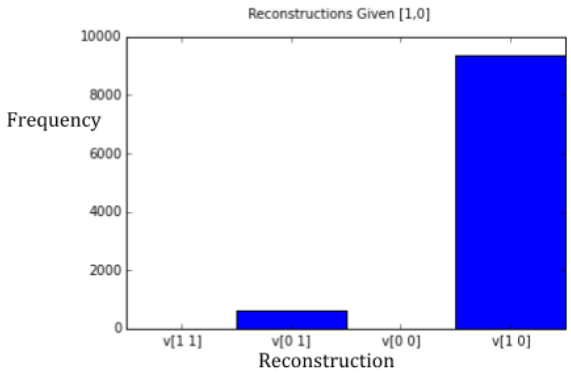


Figure 5.2: Example reconstructions for the Hand-crafted RBM, For 10000 independant reconstructions given the input $[1,0]$.

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, TODO WORDING with weights greater than 5 the networks stable state results in either $[1,0]$ or $[0,1]$. The RBMs had 2 hidden units and 2 visible units.

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. The results of trying this with the input $[1,0]$ are showing in figure 5.2. The most common reconstruction is $[1,0]$, which matches the input. We see that 500 reconstructions out of the $10,000$ reconstructions were incorrect, corresponding to $[0,1]$. This is due to the stochastic nature of the RBM.

h

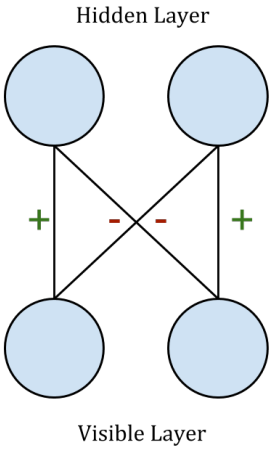


Figure 5.1: The two bit RBM for modelling a single bit on at a time in a two bit pattern. The diagonal weights are negative and the non-diagonal weights are positive.

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 columnmn indicate the reconstructions from the left and right RBMs in the ORBM.

In a similar way to the reconstructions, ancestral samples can be taken from the model and evaluated. TODO WORDING In larger dimensions Hinton has shown RBMs to be poor generative models without the extra layers above... however in the small dimensions of this task the dreams should match the training set:

A histogram of frequencies of reconstructions given the RBM is in the free phase, is shown in figure 5.3. Again the model behaves as expected, generating dream patterns that match the training dataset the majority of the time. Given this good model, the XOR RBM was duplicated and plugged into the ORBM structure as picture in figure 5.4.

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 5.5. 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.

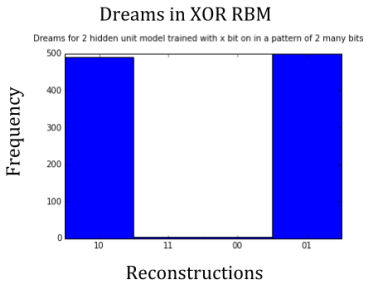


Figure 5.3: Dreams in the XOR RBM, note how only the training data is present.

XOR ORBM Analysis

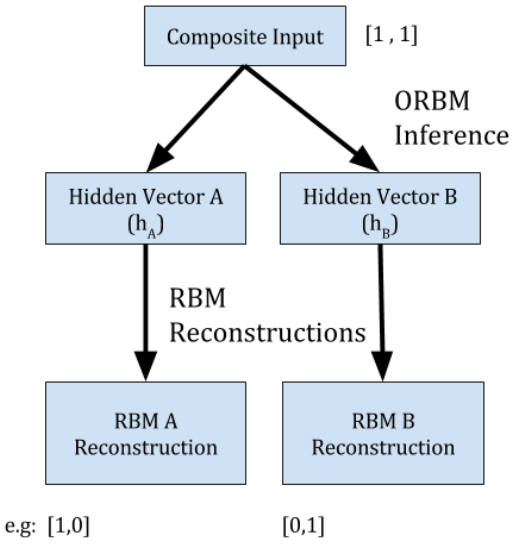


Figure 5.5: The process of generating reconstruction is shown in this diagram.

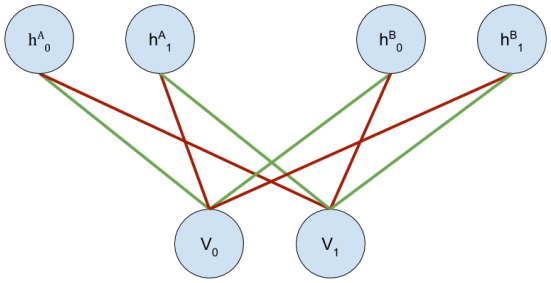


Figure 5.4: Figure showing how the XOR RBM fits into the ORBM architecture.

The results of this process are shown in figure 5.6. Where the x-axis shows the reconstructions of the form A reconstruction, B reconstruction. The results show how applying the full correction compares to the approximated correction.

The ORBM is able to separate the causes with both types of correction, as the model is being duplicated, it produces $[1,0]$ and $[0,1]$ approximately half the time and symetrically $[0,1]$ $[1,0]$ 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 seperating the sources. This is illustrated in figure 5.8.

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 figure 5.7.TODO WORDING Do the markers care that it works in this case... it's so trivial does it even mean anything for larger images..

Two bit XOR Conclusion

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 is was 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 occuring reconstruction is the

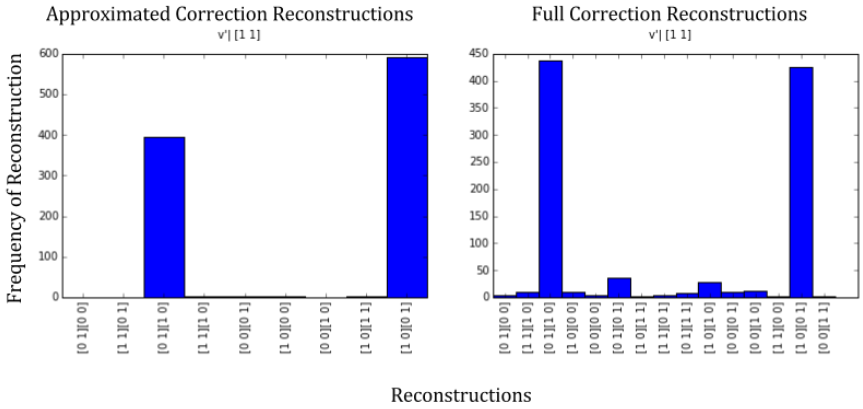


Figure 5.6: A figure with the results of running ORBM inference 1000 times, with the two different approaches to calculating the correction. The frequency for which a given reconstruction occurred is shown.

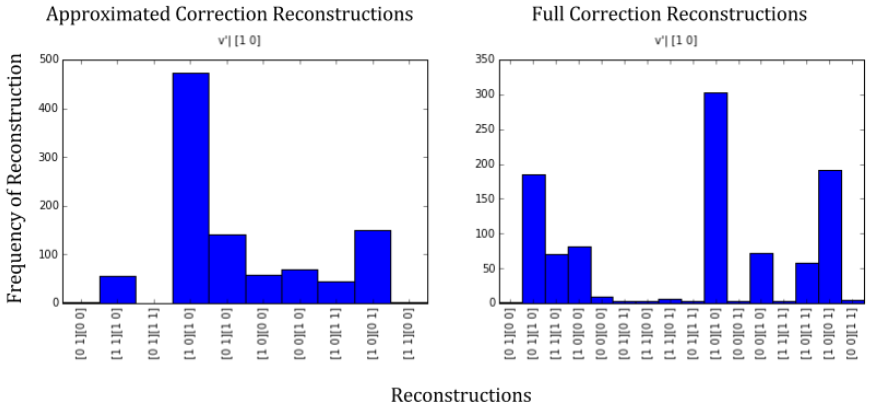


Figure 5.7: A figure showing the result of generating 1000 reconstructions with the ORBM with $[1, 0]$ as input, again comparing the Approximated correction and Full correction.

target $[1, 0]$, however there appears to be a lot less confidence.

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 is a next step. For example if $Y = 4$ and $X = 2$ then a valid inputs are the rows of the following:

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

This allows for some interesting cases, for instance using the same example as above of $Y = 4$ and $X = 2$, we can see how the ORBM is able to separate interesting patterns such as

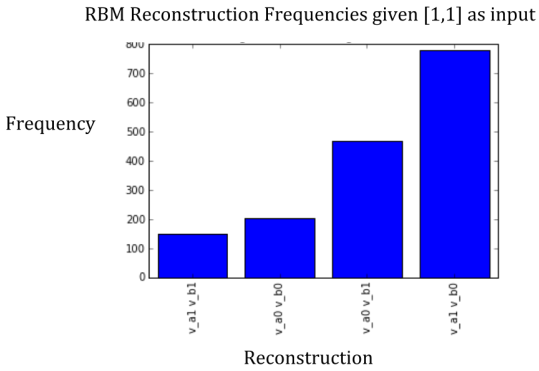


Figure 5.8: A figure showing the result of generating 1000 reconstructions with an RBM. It is clear the RBM has no mechanism to separate the sources. Hence showing it [1,1] when it has only been trained on XOR results in no separation, i.e. reconstructions including $[1, 1]$.

$[1, 1, 1, 0]$, which is a composition of $[1, 1, 0, 0]$ and $[0, 1, 1, 0]$.

Architecture and Parameters

- $|Y|$ Hidden units per RBM, any less and we were unable to train the RBM to make the target reconstructions or dreams.
- Random normally distributed, mean and standard deviation of one, starting weights. 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 histogram in a similar fashion to figure **??**. 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

For values of Y where $y \epsilon Y \big| 2 < y < 7$ and all values of X where $x \epsilon X \big| 2 < x < y < 7$ The process below is repeated.

The inference algorithm was run for 1000 reconstructions in the ORBM architecture for all unique compositions of the training set. The result was 1000 reconstructions for both the Full Correction and Approximate Correction, for some interesting compositions of the dataset.

TODO CITE: Need to grab the images out of the ipython notebook.

5.3 2D Patterns in a small dimensional space

- Dataset:
  - 2x2 square in a 5x5 image. Dataset of every possible configuration of said square.
- Method:
  - train a single RBM to represent 2x2 squares in 5x5 space. Nice and small, can still inspect reconstructions, and dreams.

- Compose two square images with each other, can the ORBM architecture separate the images.

5.4 2D Pattern, Different Rectangle Separation

- Dataset
  - n by m rectangles in 5x5 Images (TODO-IMAGE-IN-HERE) where n and m are not always equal.
- Method
  - Superimpose two images as the same way as before. How well can they separate.

5.5 MNIST Digits Reconstructions

Description

MNIST is a widely used dataset of handwritten digits (0 – 9). This task 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? An example input is illustrated in figure **??**.

Architecture and Parameters

- MNIST Digit images are 28 by 28 pixel images, with pixel values between 0 – 1.
- 10, 100 Hidden unit RBMs 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

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. 100 Gibbs iterations were used when calculating the correction (generating the hidden states (h^A and h^B) for a given input). These RBMs were also used to create standard RBM reconstructions. 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* **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

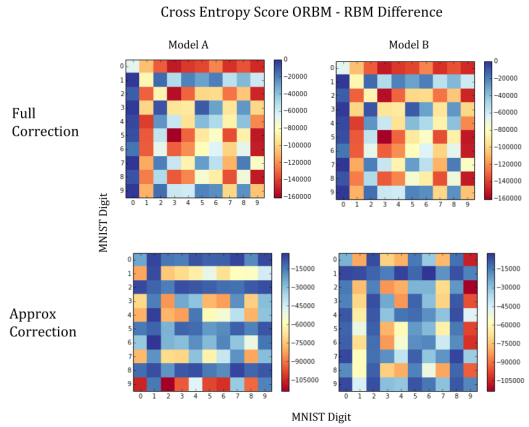
Algorithm 2: Algorithm explaining how the Scores matrices were constructed.

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).

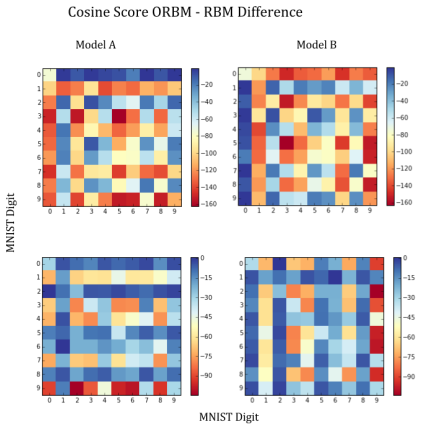
Results

These score difference matrices are seen in for the Cross Entropy and Cosine Angle scores in 5.9a and 5.9b 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 5.10.

5.5.1 MNIST Analysis

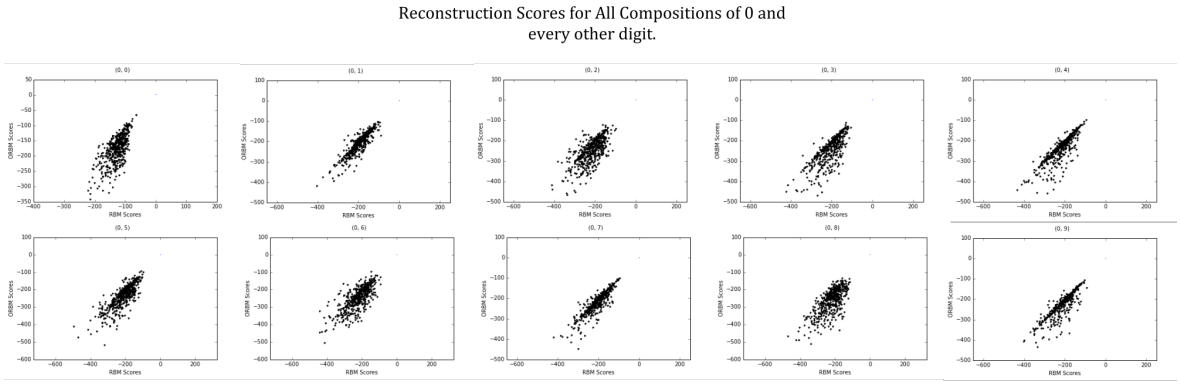


(a) The cross entropy score summed over every item in the composite dataset.

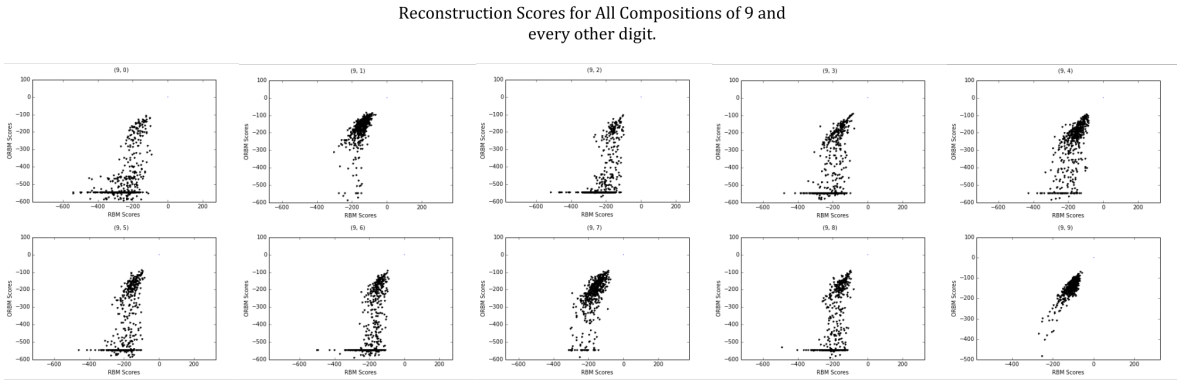


(b) The cosine score summed over every item in the composite dataset.

Figure 5.9: Dataset-wide MNIST Score Results



(a) $(0, x)$



(b) $(9, x)$

Figure 5.10: Cosine Score breakdown for the highest and lowest performing datasets, 0 and 9.

Bibliography

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

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

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

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

[5] 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.

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

[7] 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.

[8] 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.

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

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

[11] 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.

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

[13] PEARL, J. *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Morgan Kaufmann, 2014.

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