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 Nueral 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 these networks are missing a way to represent these *sources* independently. This project takes steps toward representing complex causes separately with the primary task of decomposing joint images (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.

DBNs are built of shallow networks called Restricted Boltzmann Machines.

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:

- The first articulation of the architecture, presenting the context needed to understand the new architecture.

- 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).

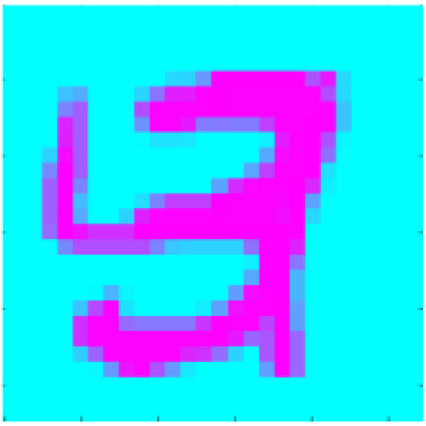


Figure 1.1: An example composition of a handwritten 4 and 3, illustrating a non-trival task where the ground truth is known.

Chapter 2

Backgroud

As the proposed architecture and algorithm extend existing work on RBMs, 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 and represent it. Generative models can map input data from raw values to higher level features. Hinton gave a compelling argument why higher level features are desirable in the context of generative models [2].

> 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 raveled into a one dimensional vector.

2.2 Directed PGMs

A Directed PGM, or in full a Directed Probabilistic Graphical Model (DPGMs), is a language for reasoning about generative models where connections between units express causation. They provide an expressive way to represent a collection of related, stochastic variables. See figure 2.1 for a simple, abstract 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 causal dependencies are expressed as a conditional probability table. For example in figure 2.1 the probabilities of A being in a given state are dependent on B, $P(A) = P(A|B)$.

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:

$$P(X) = \prod_i P(x_i | parent_i)$$

Note that a normalisation is not needed in DPGMs as the conditional probabilities enforce a value between 0 and 1.

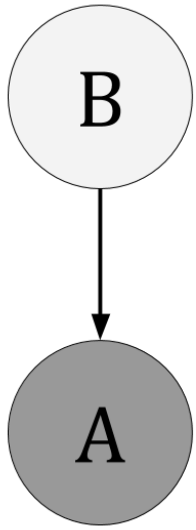


Figure 2.1: An example PGM, showing an observed variable 'A' and it's hidden cause 'B'.

2.2.1 Explaining Away in DPGMs

A common task in generative models is given a known parent unit (a cause), what is 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 task. 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 `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 in probabilities where A, B and E are the states of `alarm`,`burglar` and `earthquake` respectively:

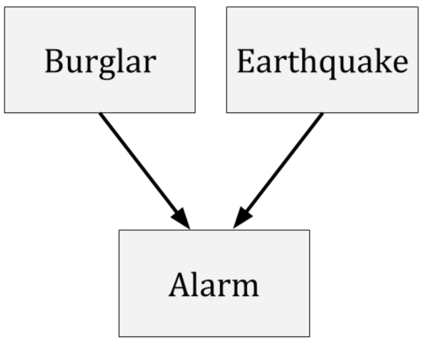


Figure 2.2: The famous Burglar, Earthquake, Alarm network showing a minimal case of explaining away.

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

2.2.2 DPGMs 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 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

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

and the facorisation of a DPGM is defined as:

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

2.3 Undirected PGMs:

Unlike DPGMs Undirected PGMs (UPGMs) do not represent causation, instead capturing a dependency between two units. These pairwise dependencies change the structure of the factorisation, requiring a normalisation constant Z, a factor Φ between two 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 the model as to compute Z, a sum over all configurations of x_i and x_j is required for al i and j.

As UPGMs do not capture causal data calculating the state of a variable given another is no longer hampered by the effect of explaining away.

2.3.1 UPGMs in Neural Networks: The Boltzmann Machine

A UPGM expressed as a neural network encodes dependancies as weights. A UPGM expressed as nueral network is referred as Boltzmann Machine or Markov Field. The Boltzmann machine has proposed in various forms throughout the years from different domains, for instance it was presented in a non-stochastic context of the Hopfield network [5]. Whereas Hinton and Sejnowski also proposed the Boltzmann machine in [4]. An example Boltzmann Machine is shown in figure 2.3. As shown in figure 2.3, the Boltzmann Machine can be recurrent, expressing complex dependencies between variables. This recurrence makes inferring the state of some variables based on knowledge of others non-trivial as the size of the network grows to be practical TODO CITE: .

In a Boltzmann machine, where W_{ij} is the weight between variables x_i and x_j the factor Φ is defined as:

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

2.3.2 Restricted Boltzmann Machines

A Boltzmann Machine's architecture can be altered to alieviate shortcomings around inference. The restriction, originally proposed by [8], and then later revitalised with a deeper architecture training algorithm in [4]. 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`.

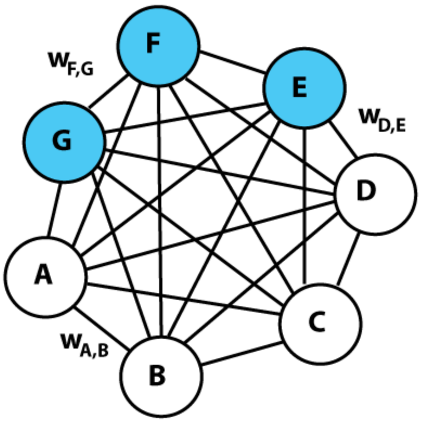


Figure 2.3: A Boltzmann Machine, the blue shaded nodes representing the observed variables, and the non-shaded nodes the latent variables.

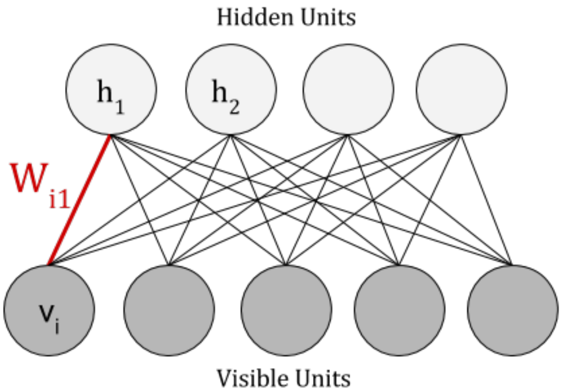


Figure 2.4: An example Restricted Boltzmann Machine with four hidden units, and five visible units. Note that the edges between units are not directed - representing a dependency not a cause.

2.3.3 Energy, and the log likelihood of the joint

An RBM models the joint distribution of hidden and visible states. The RBM assigns to every configuration of h and v an `Energy`, where the lower the energy, the more likely the RBMs configuration is to *fall* into that state. Hopfield, in the context of what is now called the Boltzmann Machine [5], presented this energy as defined by the function as

$$E(\tilde{v}, \tilde{h}) = -\sum_{i \in visible} W_{0i} v_i - \sum_{j \in hidden} W_{0j} h_j - \sum_{i,j} v_i h_j W_{ji}$$

The probability of the RBM being in a given configuration is the joint probability of $\tilde{h}$ and $\tilde{v}$.

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

If we move to log space that becomes

$$\log P(\tilde{h}, \tilde{v}) = \frac{1}{Z} \log \sum_{j,i} h_j W_{ji} v_i$$

$\frac{1}{Z}$ is the partition function, which normalises the probability of the joint. Calculating this would require summing over all possible 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. So we arrive at

$$\log P^{\star}(h, v) = \sum_i \sum_j e^{h_j v_i W_{ji}} \tag{2.1}$$

2.4 Sampling in Generative Models

Sampling is the process of drawing samples from a distribution. It is used when the distribution we want samples from is intractable to calculate analytically. This 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.

2.4.1 Why sampling is important

2.4.2 The sampling technique: Gibbs Sampling

Gibbs sampling is a special case of Markov Chain Monte Carlo, 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.

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 independent. The process of taking a step between states is referred to as a `Gibbs iteration` or a `Gibbs Step`.

Let $x \diagdown j$ be all components that make up x vector except x_j.

Data: A vector x indexed by j.
Result: Gibbs sampling algorithm
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—$x \diagdown j$);
        Update the current value of x_j in x with x'_j;
    end
end

Algorithm 1: How to write algorithms

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 more than one may be required.

Mixing Time

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 likihood 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 [3], one Gibbs step was sufficient in practice for training and using an RBM. The ORBM is not so fortunate.

2.4.3 Gibbs Sampling in a Sigmoid Belief Network

The dependance in belief networks means that sampling from the network requires a longer Markov Chain to mix, as changing the value of `Earthquake`, effects the value of `Burglar`. In a network with many connected nodes the dependence introduced makes sampling take longer. In the context of images, where there may be upwards of 1000 observable values, all with different dependancies this becomes intractable. Neal showed this by comparing the number of gibbs iterations required for small enough error rates in [6].

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.

2.5 The cost of sampling from P(h—v)

2.5.1 Inverting the generative model

Inverting a generative model can be referred to as inference, the process of reasoning about what we do not know, given that of which we do know. In generative models this is the 'posterior', the probability distribution of the hidden (latent) variables, given we know the state of the observable (visible) units. The algorithm introduced and evaluated in this report aimed to invert the ORBM, generating two separate hidden representations given an input.

2.5.2 Inverting a Sigmoid Belief Network

Performing inference in a Sigmoid Belief network would allow source separation in that each hidden unit could represent a simple cause. Meaning if a cause's state could be inferred from an input item, individual causes could be examined for an input. For example if the input was an n by n image, the Sigmoid Belief Net makes the assumption that independent causes gave rise to a given pixel.

Despite the Sigmoid Belief Network being expressive and providing a succinct encoding of inter-variable dependancies. There is too much dependency in it's expressiveness, to the extent that performing inference is intractable for a network of practical size. There do exist algorithms for performing inference in Sigmoid Belief Networks. For instance, the Belief

Propagation algorithm proposed by Judea Pearl [7] operates on this encoding, calculating the probabilities of a given network state (i.e. the state of all the variables). Belief Propagation is intractable to use as the number of variables grow. This intractability arises from the SBNs dependencies and the 'explaining away effect' as Hinton touches on in light of training stacked RBMs (DBNs) [3]. Inference is required for training generative models making Sigmoid Belief Networks impractical to train.

2.5.3 Inverting a Restricted Boltzmann Machine

The effect of this restriction is inference 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 dependant on h_2 wether or not we know anything about the visible units. This is the opposite of a Sigmoid Belief Network where knowledge of the visible units makes the hidden units dependant. By removing the recurence present in Boltzmann Machines, it reduces the expresiveness of the RBM network but makes the RBM useable in practice.

2.5.4 Reconstructions, visualising what the model has learnt

Generative Models can create an internal representation given an input. They can also generate a faux input given an internal representation. Performing one Gibbs iteration, that is sampling from the hidden units given an input $P(\tilde{h}|\tilde{v})$ and then taking the generated hidden state and generating a faux input. The model tries to reconstruct the input.

2.5.5 Fanstasies of the model

In the same way that a generative model uses reconstructions to try and recreate the supplied input based purely on how it's represented that input, performing many, many (greater than 100) Gibbs iterations with no input pattern clamped allows the reconstructions to explore the probability mass that the model has built up 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 **??**. Note in figure 3.1 the weights between the H and U layer are shared between the V and H layers.

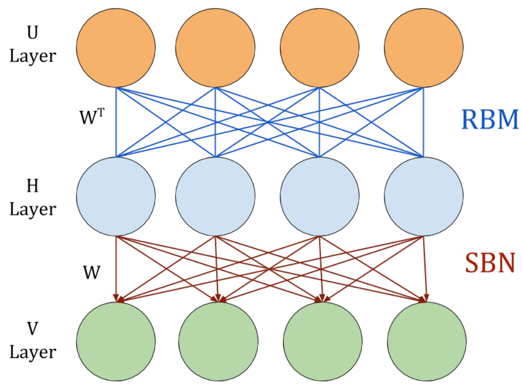


Figure 3.1: A diagram illustrating 'unrolling' an RBM by one Gibbs iteration. Note the connections between H and V layers are now directed.

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.

Ancestral Sampling in this equivalent network

Ancestral sampling in an RBM TODO CITE: as described in the section where I talk about dreams.... This network behaves a similar way, 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.

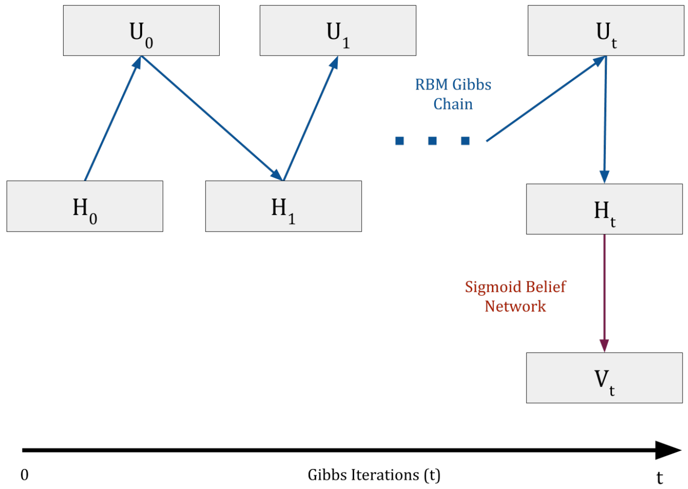


Figure 3.2: A diagram showing Ancestral sampling in the equivalent network, where normal sampling in the top 2 layers is performed until Gibbs iteration t the hidden state is pushed through the bottom layers Sigmoid Network.

To generate a sample from h is equivalent we can use equation **??**, 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:

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

We have the second term already:

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

To find the remaining term, $P^{\star}(h)$ we need to marginalise the joint (eq 3.1) over all $\mathbf{v}$ configurations:

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

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

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

By collecting terms and simplifying one can readily those that this matches the earlier form in equation 2.1.

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.

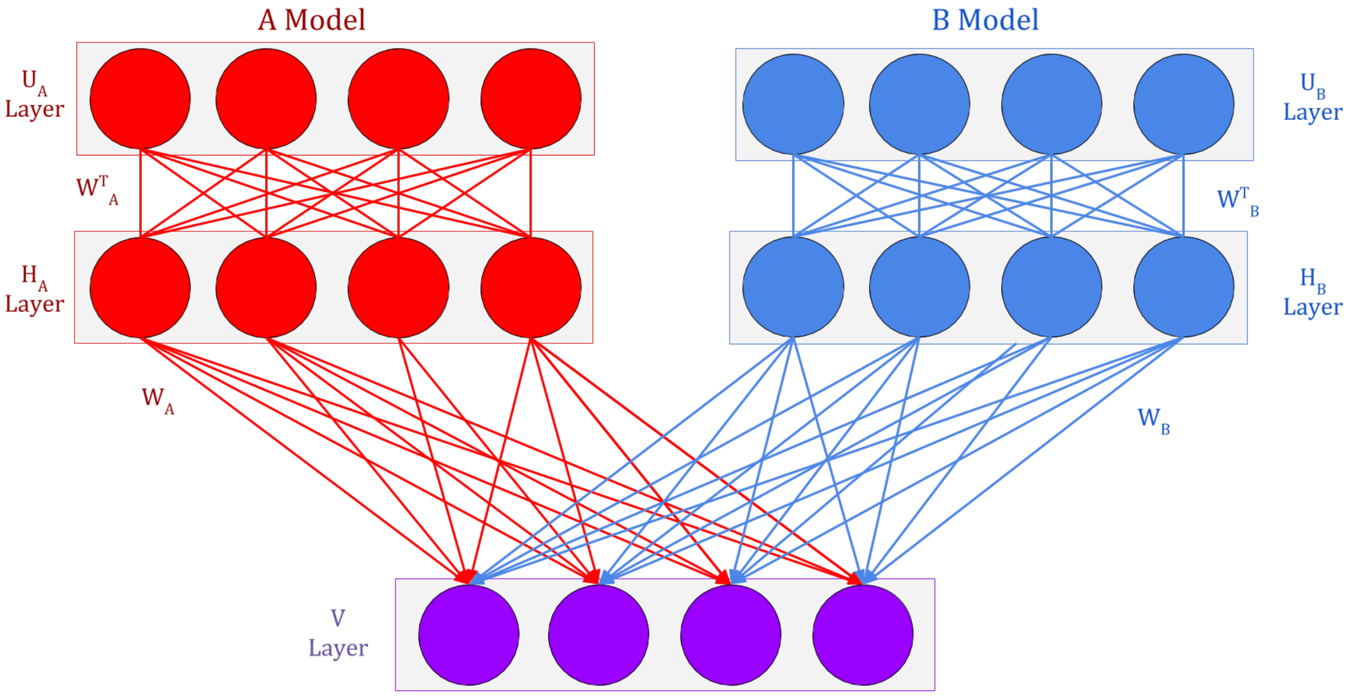


Figure 3.3: The full ORBM architecture, where A and B are the two causes that combine to form the data.

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 **??**. 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:

$$\begin{aligned} C_{ji}^A \; &= \; \log \left[\frac{\sigma(\phi_i^{Aj0})}{\sigma(\phi_i^{Aj0} + W_{ji}^A)} . \frac{\sigma(\phi_i^{Aj0} + W_{ji}^A + \phi_i^B)}{\sigma(\phi_i^{Aj0} + \phi_i^B)} \right] \\ &= \log \sigma(\phi_i^{Aj0}) \; + \; \log \sigma(\phi_i^{Aj0} + W_{ji}^A + \phi_i^B) \; - \log \sigma(\phi_i^{Aj0} + W_{ji}^A) \; - \; \log \sigma(\phi_i^{Aj0} + \phi_i^B) \\ &= \log \left[\frac{\sigma(\phi_i^A - h_i^A W_{ji}^A)}{\sigma(\phi_i^A + (1 - h_i^A) W_{ji}^A)} \right] \; - \; \log \left[\frac{\sigma(\phi_i^{AB} - h_i^A W_{ji}^A)}{\sigma(\phi_i^{AB} + (1 - h_i^A) W_{ji}^A)} \right] \end{aligned} \tag{3.2}$$

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 3.2) 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.

The proposed approximation uses a Sigmoid to approximate TODO WORDING Marcus help! I don't understand where the approximation comes from. Do you have to jusitfy it or can I just drop it in and be like, proposed approximation will also be tested?

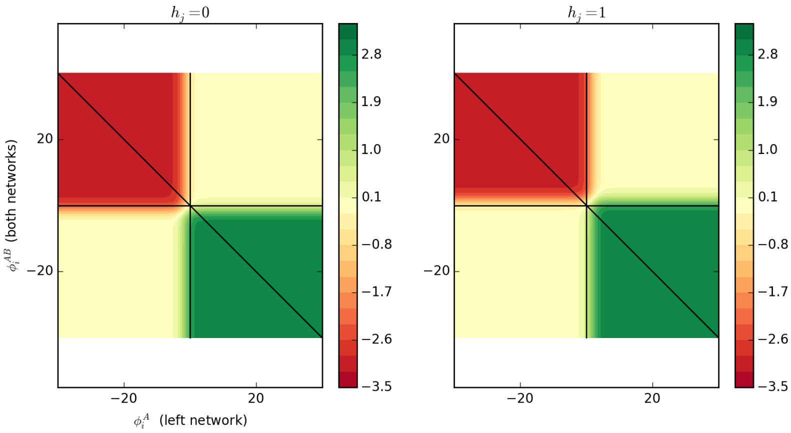


Figure 3.4: A diagram that shows the structure of the correction over the weighted sums into the hiddens of one of the RBMS versus both.

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 **??**.

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

Chapter 4

Design and Implementation

A large portion of 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.

TODO WORDING Make sure you talk about why the sigmoid is the most appropriate atrication function! Maybe this should go in background..

4.1 Evaluation Dataset Choice

- 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 expected 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 Implementation Design

4.2.1 Language Choice

- Library choice meant efficiency with benefits of python for quick dev
- Easy to deploy to grid (versus java higher risk, lack of expeirence)
- Ordering of evaluation tasks made unit testing, with hand made test cases possible. Also less risk, remove the uncertainty around the RBM.
- testing approach

4.2.2 Program Architecture

The program was implemented with unit testing and composability in mind. It was important that the design supported comparing the Full and Approximated correction. By using Python, multiple inheritance could be leveraged. 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.

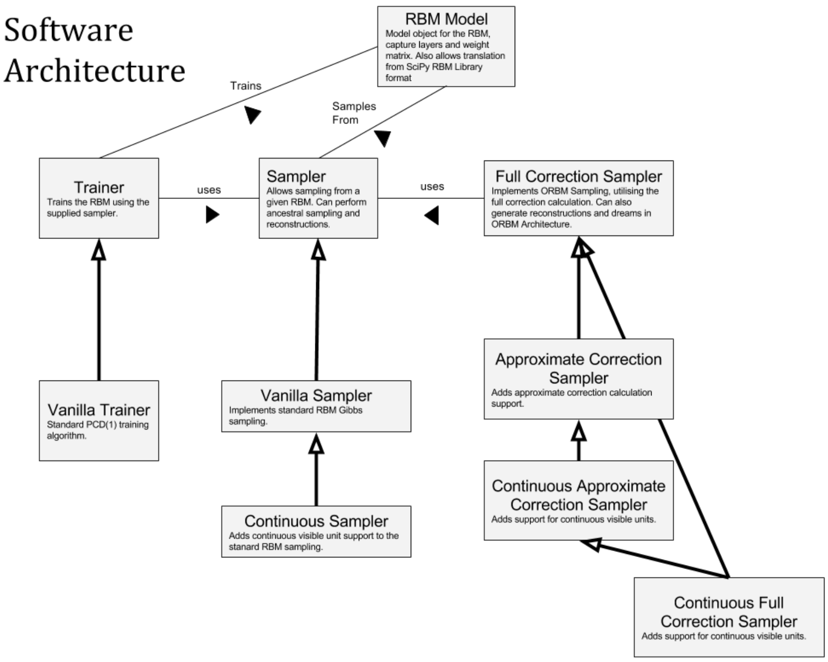


Figure 4.1: A figure showing the architecture for the implemented test suite in UML notation.

Chapter 5

Evaluation

5.1 Evaluation Design

5.1.1 Mixing Time Evaluation

- List the types Traceplot, Densityplots, Gelbin and Rubin multiple sequence diagnostics
- Performed Geweke Diagnostic

5.1.2 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.

5.1.3 Growing Complexity of tasks

- 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...

5.2 Evaluation Methods

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.

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 5.1. The optimal reconstructions are equivalent to the two images that combined form the input.

In the smaller dimensonal cases the reconstructions are inspected by hand, manually plotting the reconstructions and the

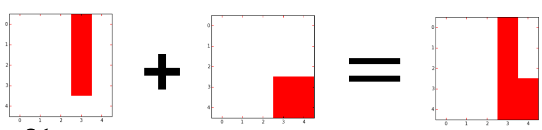


Figure 5.1: A figure illustrating two five by five pixel images combining to form a composite/multicause in-

h

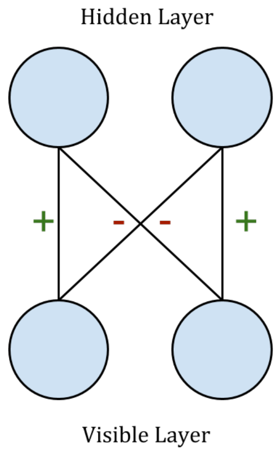


Figure 5.2: The two bit RBM for modelling a single bit on at a time in a two bit pattern. The diagonal weights are negative and the non-diagonal weights are positive.

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.

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.

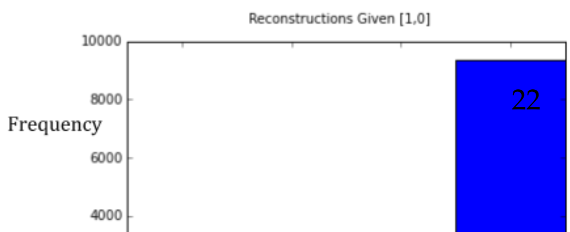
5.3 Starting from the minimal case, two bit XOR

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 3.2).

Architecture and Parameters

Being a trivial dimensionality, the RBMs weights were constructed by hand, meaning that only the ORBMs inference algorithm was



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

being evaluated and not the training of the RBMs plugged into it. This network is picture in figure 5.2, 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 5.3. 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.

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 5.4. 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 5.5.

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 5.6. 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.

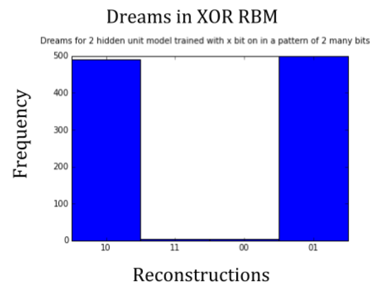


Figure 5.4: Dreams in the XOR RBM, note how only the training data is present.

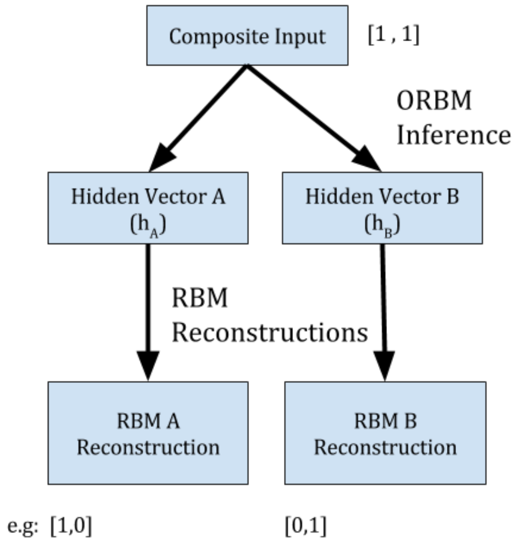


Figure 5.6: The process of generating reconstruction is shown in this diagram.

XOR ORBM Analysis

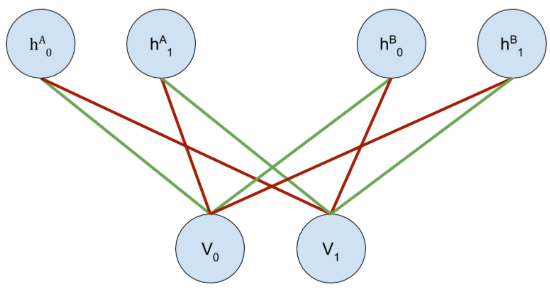


Figure 5.5: Figure showing how the XOR RBM fits into the ORBM architecture.

The results of this process are shown in figure 5.7. 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 5.9.

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 5.8.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

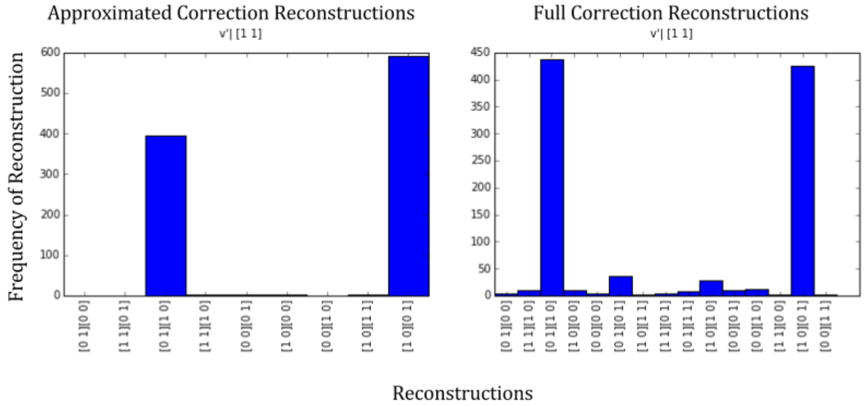


Figure 5.7: A figure with the results of running ORBM inference 1000 times, with the two different approaches to calculating the correction. The frequency for which a given reconstruction occurred is shown.

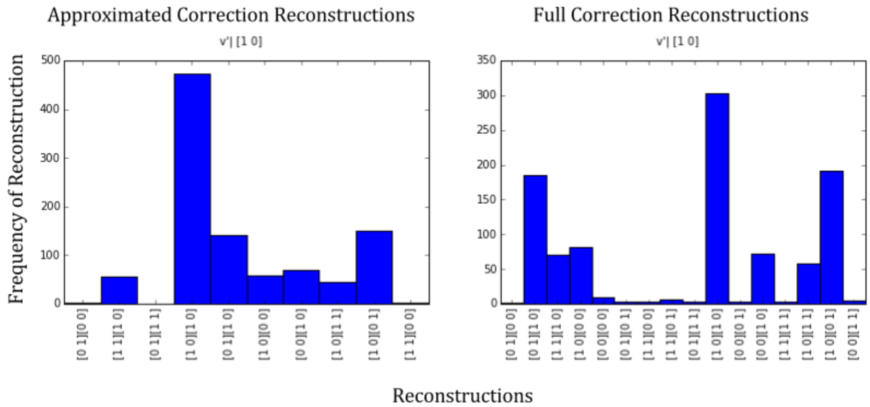


Figure 5.8: A figure showing the result of generating 1000 reconstructions with the ORBM with $[1, 0]$ as input, again comparing the Approximated correction and Full correction.

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

5.4 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:

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

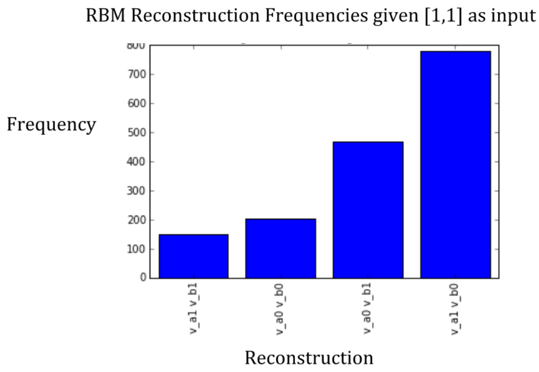


Figure 5.9: A figure showing the result of generating 1000 reconstructions with an RBM. It is clear the RBM has no mechanism to separate the sources. Hence showing it [1,1] when it has only been trained on XOR results in no separation, i.e. reconstructions including $[1, 1]$.

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 $[1, 1, 1, 0]$, 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 $y \epsilon Y | 2 < y < 7$ and all values of X where $x \epsilon X | 2 < x < y < 7$ 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.

5.5 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.

5.6 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.

5.7 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.

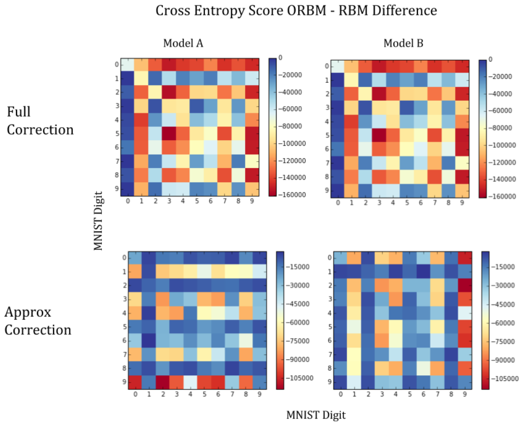
By finding the difference between the ORBMs reconstructions and the RBMs reconstruction we can create another matrix, in which a positive value represents a 'win' for the ORBMs reconstructions and a negative value represents a 'loss' (where the RBM outperformed the ORBM).

Results

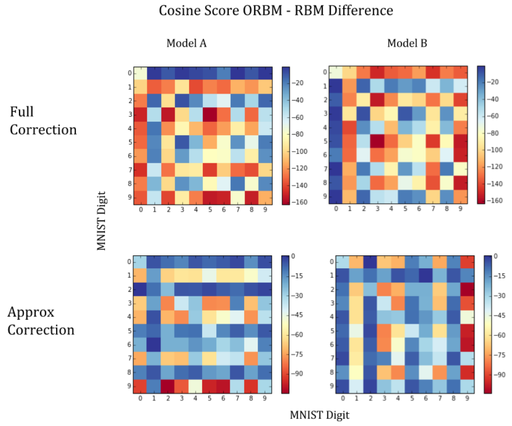
These score difference matrices are seen in for the Cross Entropy and Cosine Angle scores in 5.10a and 5.10b respectively, where the colour encodes the score. For the Cosine score, using the approximate correction, zero composed with every other digit yielded the best results for ORBM relative to the 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.

5.7.1 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, will be an equivalent dataset to five composited with four. There is a symmetry present in the Full Correction, but this is not the case for the Approximate Correction.

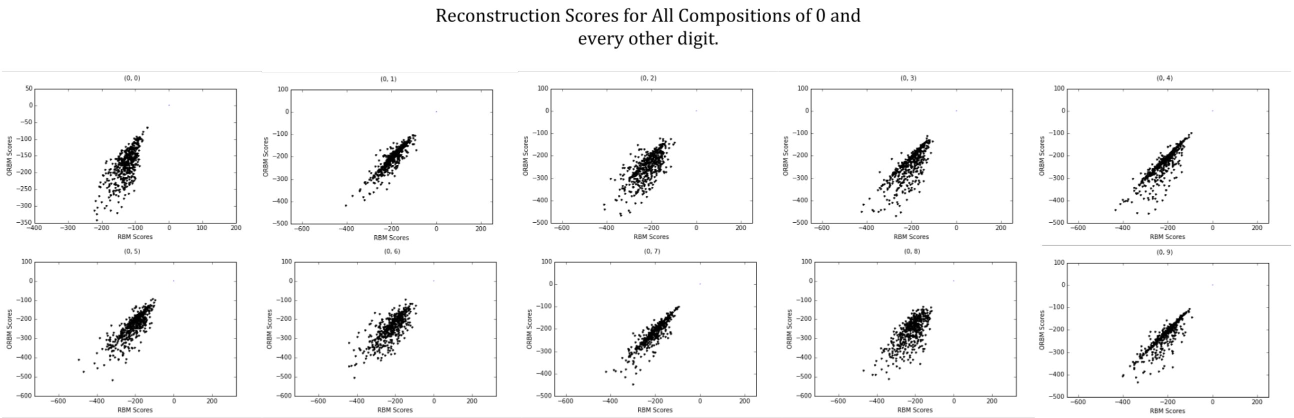


(a) The cross entropy score summed over every item in the composite dataset.

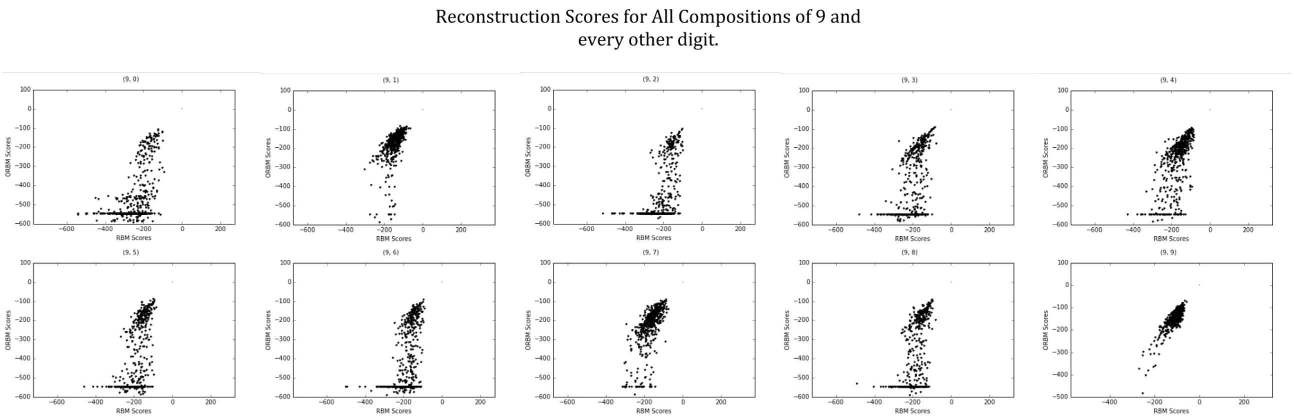


(b) The cosine score summed over every item in the composite dataset.

Figure 5.10: Dataset-wide MNIST Score Results



(a) (0,x)



(b) (9,x)

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

Chapter 6

Conclusions and Future Work

- ORBM inference is working in smaller cases really well. Even better than the RBM!
- Training algorithm for this generative model has been proposed. Haven't been able to get it going but it could be really promising - Amounts to blind source separation.

Bibliography

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

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

[3] 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.

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

[5] 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.

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

[7] 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.

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