# Nonlinear Multilayered Sequence Models

by

Ilya Sutskever

A thesis submitted in conformity with the requirements
for the degree of Master of Science
Graduate Department of Computer Science
University of Toronto

*Your file Votre référence*
*ISBN: 978-0-494-27297-8*
*Our file Notre référence*
*ISBN: 978-0-494-27297-8*

# Abstract

Nonlinear Multilayered Sequence Models

Ilya Sutskever

Master of Science

Graduate Department of Computer Science

University of Toronto

2007

Feature discovery is a fundamental problem, for the performance of any learning algorithm depends on the features it is given. In this thesis we focus on the problem of feature discovery for sequential data, such as video or speech, which is important for many practical problems, including that of constructing an intelligent agent. The hidden variables of probabilistic models that are fitted to the data often act as useful features for discrimination tasks that are not known during learning.

We introduce several powerful probabilistic sequence models with efficient learning and inference algorithms that can learn accurate models of highly complex, high-dimensional sequence data and produce hierarchical features.

# Acknowledgements

I would like to thank my supervisor Geoffrey Hinton, who taught me a considerable amount on neural networks and research, devoted me much time, and shared with me his knowledge, experience and wisdom. Working with him was very enjoyable and he made my learning process a pleasant one.

I wish to thank Radford Neal for promptly reading my thesis and providing me with valuable comments and remarks.

I would like to thank my fellow grad students and my friends, (pseudorandomly permuted): Hadassa Brunschwig, Tijmen Tieleman, Keir Mierle, Roland Memisevic, Vinod Nair, Josh Susskind, Alex Levinshtein, Ruslan Salakhutdinov, Andriy Mnih, Arnold Binas, Graham Taylor, Paul Cook, Christopher Parisien, Mike Jamieson, Peter Liu, and anyone else I forgot to mention; they made my experience of grad school (and lunches) very much fun.

And finally, I wish to thank my grandparents, parents and my brother Noam for all their support and encouragement.

# Contents

# Chapter 1

# Introduction

Consider the following tasks:

1. Given a scanned image of a handwritten note with diagrams, convert it into a typeset file (e.g., Tex) with beautiful diagrams.

2. Given a photograph, determine the identities and positions of all the people it contains.

3. Given an audio recording, determine the words spoken in it.

4. Given an email, determine whether it is spam.

These problems are important from the practical point of view, require almost no effort for humans to solve and currently cannot be solved using conventional programming techniques.

Machine learning algorithms for pattern recognition, or classification, can learn reasonably good solutions to such problems using a collection of training examples (also called a training set, or training data). For instance, a learning algorithm applied to the optical character recognition problem may be given 10,000 images of handwritten characters and their identities, labeled manually. After the algorithm finishes learning it outputs an *extremely complicated* computer program that solves the task exemplified by the training set.

Most learning algorithms are much simpler than the programs they output. They work by assuming that the desired program can be implemented by a member of a family of smoothly parameterized functions. The learning algorithm finds, using numerical optimization, a setting of the parameters so that the function, represented by the found parameters, minimizes (a smooth variant of) the number of incorrectly classified examples in the training set. It is plausible that if the function found by the algorithm performs well on the training set then it performs well for unseen examples as well, provided that the function is not too irregular.

The complexity of the resulting function lies in its parameter settings, for there are often tens of thousands (sometimes, millions [21]) of parameters. The parameters of the found function may contain little regularity, and an implementation of the learned function in C would take thousands of lines of incomprehensible code, consisting mainly of numerical values.

It is unreasonable to expect perfect performance from the learned program because the learning algorithm extrapolates the answers to the datapoints not in the training set. The extrapolation is never perfect, yet since the vast majority of functions cannot be implemented by the learning algorithm, it is unlikely that the function found by the learning algorithm performs well on the training set, yet performs much worse on unseen data, provided that the training set is large enough to account for the "overfitting ability" of the algorithm. A considerable amount of research is directed at deriving quantitative variants of the above statement, based on bounding precise complexity measures of the family of functions (see, e.g., [2], and references therein).

There are many learning algorithms for pattern recognition, all of which differ in their performance characteristics. For example, Bayesian algorithms are usually the most accurate [37], but are impractical for very large sized problems due to their long running time. For moderately large training sets, the Support Vector Machine [9] is currently among the most practical general-purpose classification learning algorithm.

The performance of every learning algorithm depends on the way the data is represented. The kind representation we consider in this thesis is one where every representation is a function that assigns a *feature vector* ($\in \mathbb{R}^N$) to every datapoint. *The representation of a datapoint* is the vector assigned to the datapoint in question by the representation. If the representation extracts and emphasizes information that is useful for determining the label, then learning algorithms that use this representation will be more accurate. For example, images are usually represented with a vector of their pixel values. When an object in an image is moved or distorted, or the illumination changes, the difference (e.g., $L_2$ distance) between the pixel representations of the images can be substantial, even when the images look very similar to us. A better representation is one that produces nearby feature vectors (e.g., the feature vectors have small $L_2$ distance) for perceptually similar images, and distant feature vectors for dissimilar images[1]. Finding such a representation is difficult, however. Similarly, to use a learning algorithm to separate the spam from the non-spam we typically must convert the raw text, which is not a vector, to into a vector representation, which is the data-type virtually all learning algorithms work with. A representation that has turned out to be successful for spam classification is the bag-of-words [16], in which a document is represented with a (sparse) vector whose coordinates correspond to words, each coordinate containing the number of times its word appeared in the email. The success of this representation shows that the order of the words (and hence the meaning of the email) is not essential for spam recognition; for example, the presence of the word "mortgage" in any context provides a strong indication that the email is spam.

[1] It follows that the desired representation defines an extremely complex distance metric on images, and it is successful only if this distance metric matches our subjective notion of similarity between images.

All of the above problems, as described, are *static* and do not have a temporal component, which is important for many problems[2]. For example, tracking, surveillance, speech recognition, and monitoring a power plant have a temporal component. More importantly, if we wish to build an intelligent agent we must be able to deal with the sequential structure in at least video and speech. For such problems, it is advantageous to have an online sequential representation, one that assigns a feature vector to every arriving datapoint, based not only on the current datapoint, but on the previous datapoints and their representations as well.

A sequential representation is also useful for Reinforcement Learning. The goal of Reinforcement Learning [46] is to learn to choose actions that lead to as large a future reward (or reinforcement) as possible; the learning is done by interacting with an environment that occasionally yields reward to the agent. This problem is very interesting from the AI point of view, being about learning intelligent behavior from experience. A common reinforcement learning algorithm estimates the future reward of every action taken from every state by interacting with the environment. For small problems these estimates are stored in memory but larger problems require the use of function approximations, not only due to memory requirements, but also because the estimation of the values of a huge table requires tremendous amounts of data, which makes learning slow. For the reinforcement learning algorithm to learn useful behavior, the function approximation must be of high quality, which critically depends on the features it uses. It is known [28] that when the features are learned directly from the reinforcement learning signal, i.e., when a straightforward multilayered preceptron function approximation is used, then severe local minima problems occur during its optimization. Therefore, finding useful general-purpose sequential features is important for reinforcement learning. General-purpose sequential features are necessary, in general, for most problems for which data arrives in a sequence and has sequential structure.

[2] Spam detection has a sequential component which is ignored by the bag-of-words representation.

How can we find general-purpose representations that are suitable for many tasks without knowing[3] what these tasks are? When probabilistic models *that have a simple relationship between their visible and hidden variables* are fitted to the data, their hidden variables act as useful features. Let us provide some examples supporting this claim:

1. Mixture of Gaussians (see, for instance, [15]) has the "mixture-component-indicator" as a hidden variable, which provides a useful rough description (quantization) of every datapoint.

2. Principal Component Analysis (PCA) (e.g., [43]) has hidden variables which are linear functions of the data, and, as features, are useful in practice for recognition of faces and hands [48, 5].

3. Independent Component Analysis has hidden variables that are marginally independent, and resemble low level vision and acoustic filters found in animals [33, 3].

4. Restricted Boltzmann Machines and their multilayered extensions [24] learn hidden variables that when used as features, result in excellent discrimination [21] on the MNIST dataset (a dataset of handwritten digits).

These examples provide evidence for the claim that the hidden states of probabilistic models (that are fitted to the training data) act as reasonably good general-purpose features. To our knowledge, there is no theory as to why this is so.

Not every probabilistic model provides a good representation with its hidden variables, however, and in fact, most do not: a model whose hidden variables do not depend on the visible variables cannot, by definition, have a useful representation at all; and if the relationship between the visible and the hidden variables is highly obfuscated and contrived (e.g., when the visible variables depend on the 50-th bit in the binary expansion

[3] This is not accurate, for we know that the tasks to be solved are "natural" in a difficult to explain manner. Indeed, we feel certain that any vision-related problem is unlikely to depend on whether the number of red pixels in a certain region is prime.

of the real value taken by the hidden variables), then the hidden variables will be, as features, useless at best. Nevertheless, it plausible that "reasonable models" provide good features with their hidden variables when fitted to the training data.

In this thesis we address the problem of finding general-purpose sequential representations by introducing new powerful probabilistic sequence models with efficient learning algorithms that are capable of learning accurate models of the data and extracting sequential hierarchical features.

# Chapter 2

# Background Material and Review of Related Work

In this chapter we describe all the necessary background material that is needed to make this thesis mostly self-contained and review the related work.

## 2.1 Probabilistic models, inference and learning

A probabilistic model is a parametrized family of probability distributions $P = P(V, H) = P_\theta(V, H)$ over visible variables $V$ and hidden variables $H$, parametrized by $\theta$, with $V$ and $H$ taking values in a finite domain (it can be infinite, but we do not consider this case at all). A probability distribution assigns a probability $P(V, H)$ in $[0, 1]$ to every configuration of $V$ and $H$ that sums to 1: $\sum_{V,H} P(V, H) = 1$. $P(V) = \sum_H P(V, H)$ is the marginal distribution of $V$, which is the essence of the model. A key reason for introduction of the hidden variables $H$ is so that the marginal distribution $P(V)$ can be expressive enough to approximate the data we wish to model even when $P(V, H)$ has simple structure. In general, the marginal distribution $P(H)$ $(= \sum_V P(V, H))$ is called the *prior* distribution of the hidden variable and $P(H|V) = P(H, V)/P(V)$ is called the *posterior* distribution. The random variables $V$ and $H$ are usually high dimensional, and the marginal distribution with respect to any set of coordinates is obtained by summing over the remaining ones. The expectation of $f$ with respect to $P$ is denoted by and equal to $\langle f(V,H)\rangle_{P(V,H)} = \sum_{V,H} f(V,H)P(V,H)$. The notation arose from the observation that this sum is actually an inner product $\langle f, P\rangle$ when $f$ and $P$ are viewed as vectors indexed by $(V,H)$.

Every random variable is distributed according to a distribution $P(X)$, which is denoted by $X \sim P(X)$. Whenever there are many random variables, for instance $X_1,\ldots,X_n$, then there is an underlying joint distribution $P(X_1,\ldots,X_n)$ such that the marginal distributions $P(X_i)$ are the distributions that govern each $X_i$ separately. For brevity, we notationally identify the random variables with their values, thus when writing $P(V|H)$ we could mean either a probability (a number) or a probability distribution over $V$: $V \sim P(V|H)$. We also omit of the dependence of $P$ on the parameter $\theta$.

The purpose of the probabilistic model, as its name suggests, is to model the data distribution over the visible variables. The hidden variables $H$ make the task *easier*, as usually $P(H,V)$ is *simple*, but due to marginalization, $P(V)$ can be very complex. The reason we wish to learn the data is in order to gain a better understanding of it and to extract features. Indeed, if the probabilistic model is such that the hidden variables can represent very specific structure, then the learned model will extract that structure from the data distribution (e.g., the Mixture of Gaussians model and the ICA models). It is harder to a-priori justify modeling the data distribution using models whose hidden variables do not have an obvious semantic meaning, which is the case, for instance, for the Boltzmann Machine which is described soon. Nevertheless, we believe that any model that is *natural* in a hard to define way learns useful features with its hidden variables, which is a reason for fitting such a model to data. Another reason for learning probabilistic models is their possible usefulness for tasks such as monitoring the functioning of a power plant. For this problem, the only available data is that of normal operation of the power plant. By using a probabilistic model to learn this normal data it is possible to detect when the state of the power plant becomes too improbable, which indicates a possible malfunctioning.

A sample from the data distribution is a training set; it is obtained from the real world and denoted by

$$\tilde{P}(V) = \frac{1}{N}\sum_{i=1}^{N}\delta_{V_i}(V)$$

where $\delta_{V_i}$ is a distribution that assigns probability 1 to $V_i$ and zero to any other point, and $\{V_1, \dots, V_n\}$ is the data in the training set. Throughout, $\tilde{P}(V)$ always stands either for the training set or the data distribution.

When fitting the parameters, a possible goal of learning is to minimize the difference between the model's distribution and the data distribution. One way of learning is maximizing the likelihood. Given a training set $V_1, \dots, V_n$, the goal of maximum likelihood learning is to maximize $\prod_{i=1}^{n} P_\theta(V_i)$ with respect to $\theta$, which is the probability of the training set under the model; it is equivalent to maximizing $\frac{1}{N}\sum_{i=1}^{N}\log P_\theta(V_i) = \langle \log P_\theta(V)\rangle_{\tilde{P}(V)}$, the average log likelihood of the parameters.

For most models, finding the global maximum with respect to $\theta$ is computationally infeasible, so we resort to finding as good parameters as we can.

To maximize the average log likelihood, we use gradient ascent. According to this method, we repeatedly calculate the derivative

$$\Delta_\theta = \frac{1}{N}\cdot\frac{\partial}{\partial\theta}\sum_{i=1}^{N}\log P_\theta(V_i)$$

and update the weights in the direction of maximum local increase: $\theta \leftarrow \theta + \varepsilon\Delta_\theta$; this way a local maximum is found. The derivative is

$$\begin{aligned}
\frac{\partial \log P_\theta(V_i)}{\partial\theta} &= \frac{\partial \log \sum_H P_\theta(V_i, H)}{\partial\theta} \\
&= \sum_H \frac{\partial P_\theta(V_i, H)}{\partial\theta} \Big/ \sum_H P_\theta(V_i, H) \\
&= \sum_H P_\theta(V_i, H)\frac{\partial \log P_\theta(V_i, H)}{\partial\theta} \Big/ P_\theta(V_i) \qquad (*)
\end{aligned}$$

$$
\begin{aligned}
&= \sum_H P_\theta(H|V_i)\frac{\partial \log P_\theta(V_i,H)}{\partial\theta} \\
&= \left\langle \frac{\partial \log P_\theta(V_i,H)}{\partial\theta} \right\rangle_{P_\theta(H|V_i)},
\end{aligned}
$$

where in $(*)$ we used the identity $\frac{\partial f}{\partial\theta} = f\frac{\partial \log f}{\partial\theta}$. This derivative, cannot, in general, be feasibly computed, due to the need to find an expectation with respect to the posterior $P(H|V_i)$.

A conceptual breakthrough occurred when it was noticed that the posterior $P(H|V)$ can be approximated with some distribution $Q(H|V)$ and that the use of the approximate posterior provides a lower bound to the average log likelihood which can be used for learning [38, 50, 27]:

$$
\begin{aligned}
\langle \log P(V)\rangle_{\tilde{P}(V)} &= \left\langle \log\sum_H P(V,H)\right\rangle_{\tilde{P}(V)} \\
&= \left\langle \log\sum_H Q(H|V)\frac{P(V,H)}{Q(H|V)}\right\rangle_{\tilde{P}(V)} \\
&= \left\langle \log\left\langle\frac{P(V,H)}{Q(H|V)}\right\rangle_{Q(H|V)}\right\rangle_{\tilde{P}(V)} \\
&\geq \left\langle \left\langle\log\frac{P(V,H)}{Q(H|V)}\right\rangle_{Q(H|V)}\right\rangle_{\tilde{P}(V)} \qquad (2.1)\\
&= \left\langle \langle \log P(V,H)\rangle_{Q(H|V)} + \mathbb{H}(Q(H|V))\right\rangle_{\tilde{P}(V)},
\end{aligned}
$$

where $Q(H|V)$ is an arbitrary distribution that approximates the posterior $P(H|V)$, the inequality follows from the concavity of log by Jensen's inequality, and $\mathbb{H}(Q(H|V)) = -\sum_H Q(H|V)\log Q(H|V)$ is the entropy of $Q(\cdot|V)$ (this is *not* the conditional entropy).

Maximizing this lower-bound with respect to both $P$ and $Q$ is useful due to its relation to maximum likelihood. By maximizing the bound with respect to $Q$ we tighten it, as equality is attained when $Q(H|V) = P(H|V)$ for all $V$ in the training set. In fact, the tighter the bound, the closer $Q(H|V)$ is to the posterior $P(H|V)$, so maximization of this bound is equivalent to approximating the posterior by the $Q$ distribution which is the closest (with respect to a specific distance measure). In general, $Q(H|V)$ is restricted to be a member of a simple parametric family such as the factorial distributions, which are the distributions $Q(H|V)$ that have the coordinates $H^{(i)}$ independent for each choice of $V$ (i.e., $Q(H^{(1)}, \dots, H^{(m)}|V) = \prod_i Q(H^{(i)}|V)$).

By maximizing the bound with respect to $P$, on the other hand, we find better model parameters.

The original EM algorithm [13] iteratively maximizes this bound with respect to $P$ and with respect to $Q$ as well; however, instead of merely *increasing* it with respect to $P$ and $Q$, at each iteration the EM algorithm finds the global maximum of $Q$ given $P$ and the global maximum of $P$ given $Q$.

## 2.2 Markov Chain Monte Carlo

The Monte Carlo method is a stochastic method for evaluating expectations of functions with respect to complex distributions over very high-dimensional spaces. It is based on a very simple observation: if $X_i \sim Q$ for $1 \le i \le n$ are independent random variables, then $\langle \sum_i f(X_i)/N \rangle_{X_i \sim Q} = \langle f(X) \rangle_{Q(X)}$, yet $\sum_i f(X_i)/N$ has less variance (i.e., it is closer to its mean with higher probability). The simplest Monte Carlo method is to simply randomly *sample* independent variables $X_i \sim Q$ and to use $\sum_i f(X_i)/N$ as an estimate of the expectation $\langle f(X) \rangle_{Q(X)}$. If $f$ is of small range (for instance, $0 \le f(x) \le 1$), then this average is a good estimate of the expectation for most random draws of the $X_i$'s from $Q$. Sampling from a distribution is done by producing pseudorandom bits and algorithmically manipulating them to produce a random variable whose distribution is $Q$.

For many realistic distributions, such as the posterior distribution of some complex probabilistic model, sampling is computationally difficult and there is no obvious procedure for obtaining a sample from the random bits. Nevertheless, by performing *Gibbs sampling* [39] we can obtain a sample from a distribution as close to $Q$ as desired using a simple procedure. Gibbs sampling applies when the random variable $X = (X^{(1)}, \ldots, X^{(n)})$ is high-dimensional and each coordinate $X^{(i)}$ has a small finite range. At each step of Gibbs sampling, we pick an index $i$ and sample the $i$th coordinate of $X$ from its conditional distribution $X^{(i)} \sim Q(X^{(i)}|X^{(1)}, \ldots, X^{(i-1)}, X^{(i+1)}, \ldots, X^{(n)})$, which is usually relatively easy. By doing so for a sufficiently large number of steps, the resulting state of $X$ will be a sample from $Q$, which is what is needed to approximate the expectation. It is not, unfortunately, feasible to know how large "sufficiently large" is. Moreover, there are many distributions that require a truly infeasible amount of Gibbs sampling in order to yield a sample. The Gibbs sampling method is simulating a Markov chain that has $Q$ as its stationary distribution, which explains the correctness of the method.

A very important property of the Monte Carlo method is its unbiasedness. A Monte Carlo estimate may not be very accurate, yet the mean of the estimate is the exact value (the expectation is taken with respect to the randomness in the sample $X$). Therefore, the sum or average of Monte Carlo estimates of different quantities results in an unbiased Monte Carlo estimate of their sum or average, which is extremely useful for our algorithms, for we often need to add and take averages of Monte Carlo estimates.

## 2.3 Probabilistic Sequence Models with Hidden State

The Kalman filter [29] and the Hidden Markov Model (HMM) [42] are the oldest and the most popular probabilistic sequence models, having the largest number of applications (see references in [41, 42]). These models have a hidden and a visible variable for every time step, and they define a joint distribution via the equation

$$P\left(V_1^T, H_1^T\right) = P(H_0) \prod_{t=1}^{T} P(V_t|H_t)P(H_t|H_{t-1}), \tag{2.2}$$

where $V_t$ and $H_t$ are the visible and hidden variables at time $t$, $T$ is the total number of time steps, and $X_i^j$ stands for $X_i, \ldots, X_j$. The HMM and the Kalman filter are equivalent, and differ only in their specification of $P(V_t|H_t)$ and $P(H_t|H_{t-1})$. The HMM, having multinomial (1-of-$n$) variables, allows the distributions $P(V_t|H_t)$ and $P(H_t|H_{t-1})$ to be arbitrary, and the Kalman filter requires $P(V_t|H_t)$ and $P(H_t|H_{t-1})$ to be Gaussian distributions with fixed covariance and with the mean being a linear function of $H_t$ and $H_{t-1}$, respectively. As a result, the marginal posterior distribution $P(H_t, H_{t-1}|V_1^T)$ is Gaussian as well, with covariance depending on $t$.

Inference, the problem of calculating expectations with respect to the posterior distribution $P\left(H_1^T|V_1^T\right)$, can be efficiently done for both models[1]. Inference is necessary for using the models when a hidden variable contains desirable (yet hidden) information, such as the location of a tracked object, and for maximum likelihood learning with the EM algorithm. The EM algorithm [13, 50] maximizes the likelihood of a model with hidden variables by reducing this hard maximization problem to a simpler one: iteratively, the values of the hidden variables are inferred using the posterior distribution (which is done algorithmically by inference), and then the expected log probability of the training data and the inferred hidden variables is increased. EM suits the HMM and the Kalman Filter particularly well because the optimal parameters can be computed very efficiently when there are no hidden variables in these models.

Despite the attractive properties of these models, they typically suffer from insufficient modeling power. The Kalman filter has *linear* dynamics in its hidden state and a *Gaussian* noise model, assumptions which are very often severely violated in practice. At the other extreme, the HMM, while being highly non-linear, has a very small hidden state space due to its explicit representation of the probability distributions. Even if the state space were made large enough (for instance, by using a computer with unlimited

[1] There is also the MAP-inference, in which the setting of the hidden variables that has maximal probability is found; despite its usefulness, we ignore this type of inference here.

memory), the HMM could not be learned from a feasible amount of data, simply due to the sheer number of parameters it would have. But if the state space is sufficiently small (e.g., of size $n \sim 1000$) then the past can constrain the future only with only $\log_2 n \sim 10$ bits of information, and even in this case, an HMM with a full transition matrix has $10^6$ parameters. As an example, the HMM is unable to efficiently model several interacting, yet only loosely coupled objects, because the space of their joint configurations is large.

To overcome the disadvantages of the HMM and the Kalman filter, more powerful probabilistic models must be used. These models need to have high-dimensional hidden variables that can carry much information; they may also need to have complex dynamics. Defining such models is easy, but doing inference for these models efficiently is hard. Models that have many richly interacting variables may have posteriors so complex that some expectations cannot even be tractably approximated [12], which seems to render the models unusable.

Nevertheless, it is possible to approximate the posterior, and when learning or using the model, assume that the approximate posterior is the actual true posterior.

There are many methods that find approximate posteriors. The Sum-Product algorithm [52, 30] is a general-purpose method for finding approximate marginals of complex distributions. It is an unprincipled adaptation of the sum-product algorithm for tree-structured distributions which is exact (see [50]) to general distributions, which is inexact, but often works well in practice, with little theoretical insight into why. It functions by iteratively updating certain equations that have multiple fixed points, and may not even converge. The paper [52] was one of the first to notice that the sum-product algorithm finds fixed points of some objective function (called the Bethe free energy), allowing some theoretical properties of it to be studied. A convex variant of the sum-product algorithm [49] appears to perform even better and to have a unique solution.

Using the Markov Chain Monte Carlo method [39] (see also section 2.2) it is possible to perform exact inference in virtually any model (necessarily making the method computationally intensive). The method obtains a sample from the desired distribution by simulating a Markov chain that has the posterior as its stationary distribution. Using several independent samples, it is possible to estimate the required expectations reasonably accurately. The problem with the Markov chain method is its inefficiency; it is also very hard to estimate, a-priori, the amount of simulation necessary for the Markov chain to yield a sample.

The classical variational approximation to the posterior distribution [38, 27] uses the approximate posterior and model to create a lower bound to its log likelihood. This lower bound has the property that the better the approximate posterior approximates the true posterior, the larger the value of the lower bound, so its maximization with respect to the posterior yields a method of approximate inference. It differs from exact inference because we do not maximize the lower bound over *all* possible distributions but only over distributions from a parametric family that probably does not contain the true posterior. When we maximize it with respect to the model parameters, we perform a step of learning.

The classical variational method was derived for a very large collection of models in a unified way [51] that results in a message-passing algorithm similar to the sum-product that maximizes a lower bound to the log likelihood.

This kind of variational inference is used by the Factorial HMM sequence model [14]. This model has several hidden Markov chains which are combined to produce a visible sequence. The approximate posterior is restricted to have the form of a product of independent Markov chains.

An unusual sequence model is the Product of HMMs [7]. This model, as its name suggests, is a product of the distributions defined by several HMMs. Unlike the other models, inference *is easy* for this model, and it is more powerful than a standard HMM. It is not trivial to learn due to the presence of a global normalizing constant, making learning difficult even when the values of the latent variables are given. It is learned by the Contrastive Divergence algorithm [23], which will be described soon.

There are many more sequence models. For many non-trivial sequence problems handcrafted models have been created, thus enriching the large family of sequence models and approximate inference algorithms. We refer the reader to [35] for a more complete discussion of such sequence models.

Sometimes, instead of performing complete inference we merely wish to find $P\left(H_t|V_1^t\right)$ (recall that $V_1^t = V_1, \ldots, V_t$), which is the filtering problem. It is common in online tracking where it is known that several hidden variables represent the position of the tracked object. The filtering problem is often easier than the full inference problem. The Particle Filter [1] is a Monte Carlo method that approximates the filtering distribution with a mixture of "spikes", or dirac-delta distributions. Whenever a new timestep is incorporated to the posterior, the spikes are drifted by some proposal distribution and are reweighted by the new likelihood that arises from the new datapoint. It often works well in practice, but can fail for very high dimensional hidden variables.

A different approximate filtering scheme is Assumed Density Filtering (ADF) [6]. In this method, the filtering distribution is "assumed" to be from a parametric family of distributions. Whenever a likelihood term that arises from a new datapoint is incorporated, the resulting "true filtering distribution" is approximated with a member from this parametric family. The inference method of one of our models resembles the ADF method.

## 2.4 Non-probabilistic Sequence Models

There are several sequence models that do not have probabilistic semantics. The Recurrent Neural Network (RNN) [44] is a neural network model with a continuous non-probabilistic hidden state that is a deterministic function of the previous hidden state and the inputs. It learns using Backpropagation though time [44]. It works well for some problems, although there are theoretical and practical difficulties in training RNNs [4].

A modification of the RNN is the Echo State Network [26]. It has many hidden units with *fixed random* connections that are not too strong and *never change.* Learning is done *only* for the connections from the hidden variables to the output variables. Using the fact that the output variables act as inputs to the hidden variables at the next time step, this model can learn to predict very complex sequences for large numbers of timesteps.

Another variation of the RNN is the Long-Short term memory [25] model. It is an RNN that has memory cells that can store and change values. This model works well in practice, detecting regularities over very large temporal ranges. It learns, similarly to the RNN, with the Backpropagation through time algorithm.

## 2.5 Discriminative Sequence Models

If we wish to classify every element of a sequence, when the labeling depends on the context, we could use a discriminative model. Such a model does not attempt to model the underlying distribution over sequences and focuses all its modeling power on predicting the labels. The Conditional Random Field [31] is a probabilistic model that learns by maximizing the *conditional* likelihood, and the Max Margin Markov Network [47] attempts to maximize the weighted margin of the correct labeling from the wrong labellings using the marginals of the dual parameters (which happen to have a form of a probability distribution); otherwise there are exponentially many dual parameters. See the respective papers for details. These methods can be kernelized (see, e.g., [11]), and thus have better features. However, the learning time of kernel methods scales quadratically with the size of the training set, a fact that limits their usability for problems with very large training sets.

# Chapter 3

# Boltzmann Machines

The Boltzmann Machine is a building block of the models we will introduce, so we discuss it in detail in this chapter.

## 3.1 The General Boltzmann Machine

The Boltzmann Machine [22] is a probabilistic model that has several attractive properties. It defines a probability distribution over $X \in \{0,1\}^N$ via the equation

$$P(X) = \frac{\exp\left(\frac{1}{2}X'WX + b'X\right)}{Z}, \tag{3.1}$$

where $W$ and $b$ are the parameters of the BM, and $X'$ is $X$ transpose. $W$ is the weights of the connections between the coordinates of $X$ satisfying $W_{ii} = 0$ and $W_{ij} = W_{ji}$, $b$ the *bias* vector and $Z = Z(W, b)$ is the partition function that is necessary for $P(X)$ to be well-defined. We partition $X$ into a set of visible and hidden variables $X = (V, H)$, $V \in \{0,1\}^{N_V}, H \in \{0,1\}^{N_H}$.

The BM has several attractive properties:

1. If we repeatedly pick a variable $X^{(i)}$ and set $X^{(i)} \leftarrow 1$ with probability $\sigma((WX)^{(i)} + b^{(i)})$, where $\sigma(z) = (1+\exp(-z))^{-1}$ is the logistic function, then we are performing Gibbs sampling from the distribution $P(X)$.

In addition, if we fix the value of $V$ and pick the indices $i$ only from variables that are in $H$, then we are performing Gibbs sampling from $P(H|V)$.

2. Finding a factorial distribution that approximates $P(H|V)$ is straightforward as well. If we represent $Q(H|V)$ by $\mathbf{x} \in [0,1]^N$ via the relation $Q(X_i = 1|V) = \mathbf{x}_i$ and set $\mathbf{x}_i = X_i$ for indecies $i$ in $V$, then by selecting a variable $\mathbf{x}^{(i)}$ for $i$ in $H$ and setting $\mathbf{x}^{(i)} \leftarrow \sigma((W\mathbf{x})^{(i)} + b^{(i)})$ we are increasing the lower bound in equation 2.1 with respect to $Q$.

3. The derivative of the average log likelihood has a very simple form:

$$\begin{aligned}\frac{\partial \langle \log P(V) \rangle_{\tilde{P}(V)}}{\partial W_{ij}} &= \left\langle X^{(i)} X^{(j)} \right\rangle_{P(H|V)\tilde{P}(V)} - \left\langle X^{(i)} X^{(j)} \right\rangle_{P(V,H)} \\ \frac{\partial \langle \log P(V) \rangle_{\tilde{P}(V)}}{\partial b_i} &= \left\langle X^{(i)} \right\rangle_{P(H|V)\tilde{P}(V)} - \left\langle X^{(i)} \right\rangle_{P(V,H)}\end{aligned}$$

where $\tilde{P}(V)$ is the distribution of the training data.

This learning rule (gradient ascent on the average log likelihood) has a biologically plausible, yet a speculative, interpretation: what our senses observe is $V$ and our hidden (i.e., mental) state is $H$. During the day, $V$ is fixed by the inputs from our senses. The regular neural activity resembles Gibbs sampling, so it is plausible that it results in a sample from $P(H|V)$, allowing each synapse to approximately evaluate the "positive correlations", the expectation $\left\langle X^{(i)} X^{(j)} \right\rangle_{P(H|V)\tilde{P}(V)}$.

During the night there are no inputs to the senses, so $V$ can change as well, which provides a justification for dreams. As before, we assume that the standard neural behavior performs Gibbs sampling and results with a sample from $P(V, H)$. Once the sample is obtained, the correlations $\left\langle X^{(i)} X^{(j)} \right\rangle_{P(H,V)}$ is evaluated for each connection locally, and is subtracted from its positive correlation in order to update the weights; this partially explains that it is hard to remember dreams because they are meant to be forgotten (see also [10]).

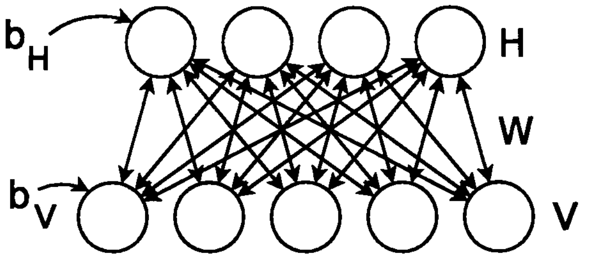

Figure 3.1: A Restricted Boltzmann Machine

Despite the simplicity and the appeal of the derivative of the log likelihood, learning general Boltzmann Machines is not feasible because obtaining the desired samples from the posterior and from the model (for the "sleep" phase) takes a prohibitively large amount of time.

## 3.2 Restricted Boltzmann Machines

The Restricted Boltzmann Machine (RBM) is a special case of the Boltzmann Machine which *can* be learned efficiently. The Restricted Boltzmann Machine has all the weights among the visible and hidden variables set to zero. Thus, the RBM is defined by the equation

$$P(V,H) = \frac{\exp(V'WH + b_V'V + b_H'H)}{Z}, \tag{3.2}$$

for $V \in \{0,1\}^{N_v}, H \in \{0,1\}^{N_h}$, where now $W$ is the matrix of connection weights between the visible and the hidden variables and $b_V$ and $b_H$ are the visible and hidden biases respectively and as before, $Z = Z(W, b_V, b_H)$ is the partition function.

As a result of the lack of connections among $V$ and among $H$, the distributions $P(V|H)$ and $P(H|V)$ are factorial, satisfying

$$\begin{aligned} P\left(V^{(i)} = 1|H\right) &= \sigma\left((WH)^{(i)} + b_V{}^{(i)}\right) & (3.3) \\ P\left(H^{(j)} = 1|V\right) &= \sigma\left((W'V)^{(j)} + b_H{}^{(j)}\right), & (3.4) \end{aligned}$$

where $\sigma(z) = (1+\exp(-z))^{-1}$. Thus, unlike most complex models, inference is trivial in an RBM.

The derivative of the average log likelihood given a training set distribution $\tilde{P}(V)$ is essentially identical to that of the general Boltzmann Machine:

$$\begin{aligned}
\frac{\partial \langle \log P(V) \rangle_{\tilde{P}(V)}}{\partial W_{ij}} &= \left\langle V^{(i)} H^{(j)} \right\rangle_{P(H|V)\tilde{P}(V)} - \left\langle V^{(i)} H^{(j)} \right\rangle_{P(V,H)} && (3.5)\\
\frac{\partial \langle \log P(V) \rangle_{\tilde{P}(V)}}{\partial b_{Vi}} &= \left\langle V^{(i)} \right\rangle_{P(H|V)\tilde{P}(V)} - \left\langle V^{(i)} \right\rangle_{P(V,H)} && (3.6)\\
\frac{\partial \langle \log P(V) \rangle_{\tilde{P}(V)}}{\partial b_{Hi}} &= \left\langle H^{(i)} \right\rangle_{P(H|V)\tilde{P}(V)} - \left\langle H^{(i)} \right\rangle_{P(V,H)}. && (3.7)
\end{aligned}$$

Evaluating these derivatives is easier for the RBM, for it is easy to compute the expectations with respect to $P(H|V)$ (i.e., the positive correlations), because $P(H|V)$ is factorial and can easily be sampled from to obtain a Monte Carlo estimate of the expectation. The negative expectations, however, are hard to estimate for the RBM.

The Contrastive Divergence [23] (CD) parameter estimation method can find good parameters efficiently. It approximately minimizes the difference between divergences of two distributions, hence its name. According to CD, instead of evaluating the negative expectation with respect to $P(V,H)$, we perform very brief Gibbs sampling *that has $V$ initialized from the data distribution.* Specifically, having estimated the expectation with respect to $P(H|V)\tilde{P}(V)$ using a sample $(V,H)$, we modify the sample by updating the visible variables, which amounts to sampling from $V \sim P(V|H)$, and then updating the hidden variables, which amounts to sampling $H \sim P(H|V)$. The resulting $(V,H)$ is a sample from the *reconstruction* distribution $P_{\text{recon}}$. CD uses the following weight updates:

$$\begin{aligned}
\Delta W_{ij} &\propto \left\langle V^{(i)} H^{(j)} \right\rangle_{P(H|V)\tilde{P}(V)} - \left\langle V^{(i)} H^{(j)} \right\rangle_{P_{\text{recon}}(V,H)} && (3.8)\\
\Delta b_{Vi} &\propto \left\langle V^{(i)} \right\rangle_{P(H|V)\tilde{P}(V)} - \left\langle V^{(i)} \right\rangle_{P_{\text{recon}}(V,H)} && (3.9)\\
\Delta b_{Hi} &\propto \left\langle H^{(i)} \right\rangle_{P(H|V)\tilde{P}(V)} - \left\langle H^{(i)} \right\rangle_{P_{\text{recon}}(V,H)}. && (3.10)
\end{aligned}$$

Notice that $P_{\text{recon}}$ satisfies $P_{\text{recon}}(V,H) = \sum_{V_0,H_0} \tilde{P}(V_0)P(H_0|V_0)P(V|H_0)P(H|V)$.

For CD to work, it is crucial to initialize $V_0$ with the training data distribution $\tilde{P}(V_0)$ in the reconstruction distribution.

The fixed points of CD learning do not coincide with the local maxima of the likelihood, since performing CD learning from a local maximum in the likelihood results in change in the parameters [8]. However, if the amount of training data is sufficiently large and there are reasons to suppose that there is an RBM $P(V, H)$ that almost exactly matches the training data distribution $\tilde{P}(V)$ (i.e., $\tilde{P}(V) \approx P(V)$) then the solution found by CD will almost certainly be close to the parameters found by maximum likelihood. To see why, assume that the parameters found by maximum likelihood satisfy $P(V) = \tilde{P}(V)$. In this case, the reconstruction distribution of CD equals to the model distribution, from which it follows that the parameters need not be updated. Unfortunately, it might be that CD has additional suboptimal solutions even in this case.

We hope to prove a qualitative version of this statement in the near future: if $\tilde{P}(V)$ is close, according to some distance measure, to some RBM distribution $P(V)$, then CD will not change the maximum likelihood solution too much.

### 3.2.1 Using Probabilities for Real Values

In practice, not all data is binary; much of the data we consider is real valued, i.e., $V \in [0,1]^{N_V}$. The Restricted Boltzmann Machine, as described, is not suitable for modeling such data. However, there is an unprincipled adaptation of the Restricted Boltzmann Machine that works reasonably well in practice, though only for some datasets.

The modification is as follows: we let $V$ take real values in a simple-minded fashion. When we sample $H \sim P(H|V)$, equation 3.4 is used with the real values of $V$. On the other hand, when we sample $V \sim P(V|H)$, we use equation 3.3 in the following way: we *set* $V^{(i)} = \sigma\left((WH)^{(i)} + b_V{}^{(i)}\right)$ for all $i$, *deterministically*. This is the only way in which the modified Boltzmann Machine differs from the standard Boltzmann Machine. By doing the "Gibbs-sampling" outlined above we obtain a sample from the modified

RBM, a sample from its posterior and a sample from the reconstruction distribution that is required for the CD weight update.

As a result, these updates lose their clean probabilistic semantics, but this modification can work well in practice. This is precisely the way in which we use our sequence models on real-valued data.

## 3.3 A Multilayered Extension of the RBM

The RBM achieves its many excellent properties by having only one layer of hidden variables with no connections among them. The hidden variables, as features, are linear functions squashed by the sigmoid nonlinearity (see equation 3.4), so the representational power of the hidden variables of the RBM is limited. A much more desirable representation is one in which there are several layers of hidden variables that are arranged in a hierarchical fashion. This is what this section is about.

Let $\tilde{P}(V)$ denote the data distribution and $P(V, H)$ denote the joint distribution defined by the RBM that is already fitted to the data. The idea is to get another RBM, $Q(H, U)$, which has $H$ as its visible and $U$ as its hidden variables, to learn to model the aggregated posterior distribution, $\tilde{Q}(H)$, of the first RBM:

$$\tilde{Q}(H) = \sum_{V} P(H|V)\tilde{P}(V). \tag{3.11}$$

Provided $Q(H)$ models $\tilde{Q}(H)$ better than $P(H)$ does, the augmented model

$$M_{PQ}(V, H, U) = Q(U, H)P(V|H) \tag{3.12}$$

is a better model of the original data than the $P(V, H)$ defined by the first RBM alone [24]. It thus inherits $P(V|H)$ from the first RBM but instead of its prior $P(H)$ it uses $Q(H)$. To sample from the augmented model, we sample from $Q(H, U)$, discard the value of $U$, and then sample from $P(V|H)$ to obtain $V$. Provided $N_U \geq N_V$ ($U$ has more dimensions that $V$), $Q$ can be initialized by using the parameters from $P$ to ensure that

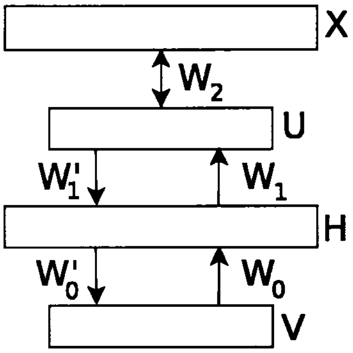

Figure 3.2: A multilayered extension of the RBM. The weights $W$ are used for sampling and $W'$ are used for approximate inference.

the two RBMs define the same distribution over $H$. This is done by using the weight of the connection between $V_i$ and $H_j$ for the weight of the connection between $H_j$ and $U_i$. If $i > N_v$, then the connection between $H_j$ and $U_i$ is set to zero. Starting from this initialization, proper optimization ensures that $Q(H)$ models $\tilde{Q}(H)$ better than $P(H)$ does.

The second RBM, $Q(H, U)$, learns by fitting the distribution $\tilde{Q}(H)$, which is not equivalent to maximizing $\langle \log M_{PQ}(V) \rangle_{\tilde{P}(V)}$. Following [24], we will show that this learning procedure maximizes a variational lower bound to $\langle \log M_{PQ}(V) \rangle_{\tilde{P}(V)}$. We can view this way of learning $Q$ as using $P(H|V)$ to approximate the posterior distribution $M_{PQ}(H|V)$. The standard variational bound yields

$$\langle \log M_{PQ}(V) \rangle_{\tilde{P}(V)} \geq \left\langle \langle \log Q(H) P(V|H) \rangle_{P(H|V)} + \mathbb{H}(P(H|V)) \right\rangle_{\tilde{P}(V)} \tag{3.13}$$

where $\mathbb{H}(P(H|V))$ is the entropy of $P(H|V)$ for each value of $V$. Maximizing this lower bound with respect to the parameters of $Q$ whilst holding the parameters of $P$ and the approximating posterior $P(H|V)$ fixed is precisely equivalent to fitting $Q$ to $\tilde{Q}(H)$. Note that the details of $Q$ are unimportant; $Q$ could be any kind of a model, and not just an RBM. The main advantage of using another RBM is that it makes it possible to initialize $Q(H)$ to be equal to $P(H)$, so the bound starts as an equality and any improvement in the bound guarantees that $M_{PQ}(V)$ is a better model of the data than $P(V)$.

This procedure can be repeated recursively as many times as desired, creating very deep hierarchical representations. For example, a third RBM, $R(U, X)$, can be used to model the aggregated approximate posterior over $U$ obtained by

$$\tilde{R}(U) = \sum_{V} \sum_{H} Q(U|H)P(H|V)\tilde{P}(V) \tag{3.14}$$

Provided $R(U)$ is initialized to be the same as $Q(U)$, $M_{QR}(H)$ will be a better model of $\tilde{Q}(H)$ than $Q(H)$, but this does not mean that $M_{PQR}(V)$ is necessarily a better model of $\tilde{P}(V)$ than $M_{PQ}(V)$. It does mean, however, that learning $R$ will improve the variational bound obtained by using the distribution $P(H|V)Q(U|H)$ to approximate the posterior distribution $M_{PQR}(U, H|V)$, and this lower bound will be higher than the lower bound to the log likelihood of $M_{PQ}$ in equation 3.13.

The idea outlined in this section will be used in the following chapters to make the models we introduce multilayered.

# Chapter 4

# The Temporal Restricted Boltzmann Machine

In this chapter we describe the first of our models, the Temporal Restricted Boltzmann Machine (the TRBM). The TRBM is a sequence model that has a componential state space with highly non-linear dynamics, simple approximate inference and learning algorithms, and a hierarchical multilayered extension similar to that of the RBM. The TRBM has an RBM for each timestep, and inherits some of the RBM's desirable properties.

## 4.1 Model definition

The TRBM sequence model is constructed from the conditional distribution

$$P\left(V_t, H_t | V_{t-m}^{t-1}, H_{t-m}^{t-1}\right) = \exp\left(V_t' C_0 H_t + B_V\left(V_{t-m}^{t-1}\right)' V_t + B_H\left(V_{t-m}^{t-1}, H_{t-m}^{t-1}\right)' H_t\right) / Z, \tag{4.1}$$

where $V_i^j = V_i, \ldots, V_j$, $B_V\left(V_{t-m}^{t-1}\right)$ and $B_H\left(V_{t-m}^{t-1}, H_{t-m}^{t-1}\right)$ are vector valued bias functions that produce a dynamic bias for every setting of their arguments, and $Z$ is the partition function that ensures that the above equation defines a probability distribution for every value of $C_0$ and the arguments to the bias functions. Conditioned on $V_{t-m}^{t-1}, H_{t-m}^{t-1}$, this

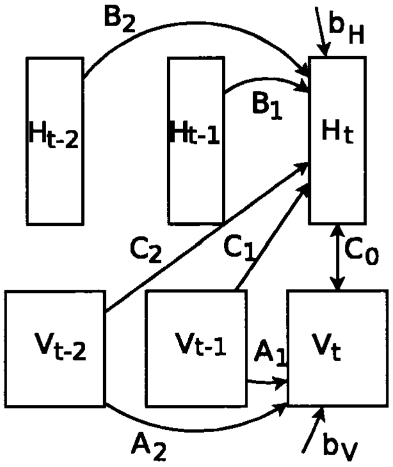

Figure 4.1: The Temporal Boltzmann Machine. In this TRBM, $m = 2$.

distribution is an RBM (see section 3.2) whose weight matrix is $C_0$ and whose biases are given by the bias functions. The variables $V_1^T$ and $H_1^T$ are binary, and $m$ is a structural parameter that determines the number of timesteps this conditional distribution can directly access. We call the conditional distribution a TRBM as well.

The TRBM model is given by the product

$$P\left(V_1^T, H_1^T\right) = \prod_{t=1}^{T} P\left(V_t, H_t | V_{t-m}^{t-1}, H_{t-m}^{t-1}\right), \tag{4.2}$$

where $T$ is the length of the sequence to be modeled.

To obtain a sample from the TRBM distribution, we let $t$ range from 1 to $T$ and sample $(V_t, H_t)$ (using Gibbs sampling) from the RBM distribution $P\left(V_t, H_t | V_{t-m}^{t-1}, H_{t-m}^{t-1}\right)$ which is conditioned on the values previously generated for $V_{t-m}^{t-1}$ and $H_{t-m}^{t-1}$. The resulting $\left(H_1^T, V_1^T\right)$ is a sample from the TRBM.

To complete the specification of the TRBM, we define the bias functions by the equations

$$B_V\left(V_{t-m}^{t-1}\right) = A_1' V_{t-1} + \cdots + A_m' V_{t-m} + b_V' \tag{4.3}$$

$$\begin{aligned} B_H\left(V_{t-m}^{t-1}, H_{t-m}^{t-1}\right) &= B_1' H_{t-1} + \cdots + B_m' H_{t-m} + \\ &\quad C_1' V_{t-1} + \cdots + C_m' V_{t-m} + b_H' \end{aligned} \tag{4.4}$$

The reason for this definition of the bias functions is its simplicity. Any reasonable definition of the bias functions would result in a useful model.

According to the definition of the TRBM, in some cases the bias function needs to access a variable with a nonpositive index, which does not exist (it happens for TRBMs that are at the beginning of the sequence, modeling the frames $1, \dots, m$). This is handled by replacing the product, for instance, of $C_i V_{t-i}$ by a special bias $c_i$ whenever $t - i$ is nonpositive; the matrices $A_i$ and $B_i$ also have similar special biases and are treated analogously.

## 4.2 Approximate filtering

The TRBM is defined so that $P\left(H_t | V_1^t, H_1^{t-1}\right)$, the distribution of the hidden state $H_t$ given all past hidden and past and present visible variables, is factorial, and is given by the equation

$$P\left(H_t{}^{(i)} = 1 | V_1^t, H_1^{t-1}\right) = \sigma\left((C_0' V_t)^{(i)} + B_H\left(V_{t-m}^{t-1}, H_{t-m}^{t-1}\right)^{(i)}\right), \tag{4.5}$$

where recall that $\sigma(z) = (1 + \exp(-z))^{-1}$. This equation suggests that a factorial approximation to the *filtering distribution*, $P\left(H_t | V_1^t\right)$, can be accurate.

Let $Q\left(H_t | V_1^t\right)$ denote such a factorial approximation to the filtering distribution which we soon describe how to compute. It is not a variational approximation in the standard sense, for it does not have free parameters and it is not being optimized over. The filtering distribution is used as an approximation to the posterior as well, so it is denoted by $P\text{approx}\left(H_1^T | V_1^T\right)$ in other contexts. This is indeed a drastic approximation, and it is conceivable that in some cases it might be too inaccurate to be useful.

let $\mathbf{p}_t \in [0, 1]^{N_H}$ be a vector that represents $Q$ via $Q\left(H_t{}^{(i)} = 1 | V_1^t\right) = \mathbf{p}_t{}^{(i)}$. Recall that we generically denote the $i$th coordinate of $X$ by $X^{(i)}$. To compute $\mathbf{p}_t$ from $\mathbf{p}_1^{t-1}$ and $V_1^t$, we modify equation 4.5 by replacing $H_1^t$ wih $\mathbf{p}_1^t$ throughout, namely

$$Q\left(H_t{}^{(i)} = 1 | V_1^t\right) = \mathbf{p}_t{}^{(i)} = \sigma\left((C_0' V_t)^{(i)} + B_H\left(V_{t-m}^{t-1}, \mathbf{p}_{t-m}^{t-1}\right)^{(i)}\right), \tag{4.6}$$

where we note that equation 4.4, defining the bias function $B_H$, is valid for real values. Equation 4.6 is analogous to equation 4.5, and implements non-iterative mean-field dynamics of the Boltzmann machine, but it does not correspond to maximization of a lower bound to the log likelihood (see section 3.2). This approximation is accurate whenever the number of hidden variables is large and the strength of the connections between them are weak [40]. This kind of approximate filtering is reminiscent of the Assumed Density Filtering [6] (see also section 2.3), in that we assume that the filtering distribution is factorial, and approximate the filtering distribution at the next timestep with a factorial distribution as well.

## 4.3 Learning

To learn the TRBM model we randomly select a vector $V_1^T$ from the training set, use approximate filtering to obtain the hidden states $H_1^T$ (i.e., a sample from the approximate posterior distribution), and once the hidden variables become known, we compute (approximately, using Contrastive Divergence; section 3.2) the gradients of the log probabilities of each conditional distribution $P\left(V_t, H_t | V_{t-m}^{t-1}, H_{t-m}^{t-1}\right)$ given the sampled $V_1^T, H_1^T$.

For learning we approximately maximize the lower bound of the log likelihood

$$\left\langle \log P\left(V_1^T\right)\right\rangle_{\tilde{P}(V_1^T)} \geq \left\langle \left\langle \log P\left(V_1^T, H_1^T\right)\right\rangle_{P_{\text{approx}}(H_1^T|V_1^T)} + \mathbb{H}\left(P_{\text{approx}}\left(H_1^T|V_1^T\right)\right)\right\rangle_{\tilde{P}(V_1^T)} \tag{4.7}$$

with respect to $P$. We perform the maximization using gradient ascent, by repeatedly computing the derivative of the lower bound with respect to the parameters of $P$, while assuming that $P_{\text{approx}}$ *remains constant.* This assumption is false, for $P_{\text{approx}}$ depends on the parameters of $P$, hence the value of the lower bound can, in principle, *decrease* due to the changes in $P_{\text{approx}}$. The fact that learning succeeds suggests that this approximation does not produce a large error.

To find the above derivative, by equation 4.2 we merely need to differentiate with respect to the parameters of $P$ the expression

$$\sum_{t=1}^{T} \left\langle \log P\left(V_t, H_t | V_{t-m}^{t-1}, H_{t-m}^{t-1}\right)\right\rangle_{P_{\text{approx}}(H_1^T|V_1^T)\tilde{P}(V_1^T)} .$$

Since taking an expectation and a derivative commute, the derivative of the above expression with respect to every parameter $\theta$ is

$$\left\langle \sum_{t=1}^{T} \frac{\partial \log P\left(V_t, H_t | V_{t-m}^{t-1}, H_{t-m}^{t-1}\right)}{\partial \theta} \right\rangle_{P_{\text{approx}}(H_1^T|V_1^T)\tilde{P}(V_1^T)}, \tag{4.8}$$

which can be estimated by a Monte Carlo estimate (see section 2.2) by randomly selecting $V_1^T$ from the training set, $H_1^T$ from $P_{\text{approx}}\left(H_1^T|V_1^T\right)$ (which is easy, for $P_{\text{approx}}$ is factorial) and evaluating the gradient given $V_1^T$ and the sampled value of $H_1^T$. Given $H_1^T$, the conditional distributions become RBMs, the derivatives of which are easily computed using CD. Let $R_t(V_t, H_t) = P\left(V_t, H_t | V_{t-m}^{t-1}, H_{t-m}^{t-1}\right)$ be the distribution of the TRBM at time $t$, conditioned on the *sampled* values of $H_1^T$. In this case, the derivatives are

$$\begin{aligned}
\frac{\partial \log P\left(V_1^T, H_1^T\right)}{\partial (C_0)_{ij}} &= \sum_{t=1}^{T} \left\langle V_t^{(i)} H_t^{(j)} \right\rangle_{R_t(H_t|V_t)} - \left\langle V_t^{(i)} H_t^{(j)} \right\rangle_{R_t(H_t,V_t)} \\
\frac{\partial \log P\left(V_1^T, H_1^T\right)}{\partial (C_k)_{ij}} &= \sum_{t=k+1}^{T} \left\langle H_t^{(j)} \right\rangle_{R_t(H_t|V_t)} V_{t-k}^{(i)} - \left\langle H_t^{(j)} \right\rangle_{R_t(H_t,V_t)} V_{t-k}^{(i)} \\
\frac{\partial \log P\left(V_1^T, H_1^T\right)}{\partial (B_k)_{ij}} &= \sum_{t=k+1}^{T} \left\langle H_t^{(j)} \right\rangle_{R_t(H_t|V_t)} H_{t-k}^{(i)} - \left\langle H_t^{(j)} \right\rangle_{R_t(H_t,V_t)} H_{t-k}^{(i)} \\
\frac{\partial \log P\left(V_1^T, H_1^T\right)}{\partial (A_k)_{ij}} &= \sum_{t=k+1}^{T} \left\langle V_t^{(j)} \right\rangle_{R_t(H_t|V_t)} V_{t-k}^{(i)} - \left\langle V_t^{(j)} \right\rangle_{R_t(H_t,V_t)} V_{t-k}^{(i)} \\
\frac{\partial \log P\left(V_1^T, H_1^T\right)}{\partial (b_V)_{i}} &= \sum_{t=1}^{T} V_t^{(i)} - \left\langle V_t^{(i)} \right\rangle_{R_t(H_t,V_t)} \\
\frac{\partial \log P\left(V_1^T, H_1^T\right)}{\partial (b_H)_{i}} &= \sum_{t=1}^{T} \left\langle H_t^{(i)} \right\rangle_{R_t(H_t|V_t)} - \left\langle H_t^{(i)} \right\rangle_{R_t(H_t,V_t)}
\end{aligned}$$

for $1 \leq k \leq m$. The calculation of the expectations with respect to $R_t$ are replaced with their CD version the way it is done for an RBM, namely, by replacing $R_t(H_t, V_t)$ with a sample from the reconstruction distribution, $R_{t,\text{recon}}(H_t, V_t)$, which is obtained exactly as it is for a standard RBM; see section 3.2. The special biases are learned similarly to the standard biases.

In practice, instead of evaluating the expectations $\langle \cdot \rangle_{P_{\text{approx}}}$ using a Monte Carlo estimate we followed a mean-field approach: instead of using the sampled value of $H_1^T$, we use $\mathbf{p}_1^T$, the probabilities that the respective coordinates in $H_1^T$ take the value 1 as *the inputs to the bias functions* that create the distribution $R_t$. Even though this is not theoretically justified, it works well in practice and reduces the amount of noise in the parameter updates.

## 4.4 Multilayered TRBM

In this section we show how to increase the modeling power of the TRBM by adding hidden layers the way it was done for the RBM (see section 3.3).

Let $P\left(V_1^T, H_1^T\right)$ be the distribution defined by a TRBM fitted to the data distribution $\tilde{P}\left(V_1^T\right)$. The posterior $P\left(H_1^T|V_1^T\right)$, being a very complicated distribution, is approximated by $P_{\text{approx}}(H_1^T|V_1^T)$. Let $Q(H_1^T, U_1^T)$ be another TRBM that learns the aggregated approximate posterior $\tilde{Q}\left(H_1^T\right) = \sum_{V_1^T} P_{\text{approx}}(H_1^T|V_1^T)\tilde{P}(V_1^T)$, where $U_1^T$ is the sequence of the hidden variables and $H_1^T$ is the sequence of the *visible* variables of the TRBM $Q$. The approximate posterior of $U_1^T$, $Q_{\text{approx}}\left(U_1^T|H_1^T\right) P_{\text{approx}}\left(H_1^T|V_1^T\right)$, allows $U_1^T$ to represent higher-level features that can be computed on-line, since neither $P_{\text{approx}}$ nor $Q_{\text{approx}}$ make use of future frames. The resulting augmented model, $M_{PQ}\left(V_1^T\right)$, is one where we first sample from $Q\left(H_1^T\right)$, then from $P\left(V_1^T|H_1^T\right)$, so $M_{PQ}\left(V_1^T\right) = \sum_{H_1^T} P\left(V_1^T|H_1^T\right) Q\left(H_1^T\right)$. If we can initialize $Q$ so that $Q\left(H_1^T\right) = P\left(H_1^T\right)$, then the augmented model is identical to the $P$ and has the same likelihood. By making $Q$ learn $\tilde{Q}$, the aggregated approximate posterior of $P$, we increase the lower bound

$$\left\langle \log M_{PQ}\left(V_1^T\right)\right\rangle_{\tilde{P}(V_1^T)} \geq \left\langle \left\langle \log Q\left(H_1^T\right) P\left(V_1^T|H_1^T\right)\right\rangle_{P_{\text{approx}}(H_1^T|V_1^T)} + \mathbb{H}\left(P_{\text{approx}}\left(H_1^T|V_1^T\right)\right)\right\rangle_{\tilde{P}(V_1^T} \tag{4.9}$$

with respect to $Q$, the way it is done for the RBM. Unfortunately, we are unable to make $Q$ learn $\tilde{Q}$ by increasing the likelihood, not even in a CD sense, so this bound does not quite apply for the algorithm we use.

This is very similar to the bound in equation 3.13, except for the use of the approximate posterior. At the beginning of the optimization this bound is strictly *smaller* than $\langle \log P\left(V_1^T\right)\rangle_{\tilde{P}\left(V_1^T\right)}$ even when $Q\left(H_1^T\right) = P\left(H_1^T\right)$, because of the approximate posterior: the lower bound is equal to the average log likelihood only if the $Q$ *is* the posterior. It could, therefore, remain less than $\langle \log P\left(V_1^T\right)\rangle_{\tilde{P}\left(V_1^T\right)}$ indefinitely. This is not the case for RBMs, since the exact posterior $P(H|V)$ is easily computable, so the lower bound is equal to $\langle \log P(V)\rangle_{\tilde{P}(V)}$ at the beginning of the optimization.

Although our learning procedure maximizes a lower bound that is initially smaller than $\langle \log P\left(V_1^T\right)\rangle_{\tilde{P}\left(V_1^T\right)}$, it is plausible that by the end of learning the bound will exceed $\langle \log P\left(V_1^T\right)\rangle_{\tilde{P}\left(V_1^T\right)}$. In addition, since we use an approximate posterior during the learning of $P\left(V_1^T\right)$ (recall that inference is intractable), we are performing approximate maximization of a lower bound on $\langle \log P\left(V_1^T\right)\rangle_{\tilde{P}\left(V_1^T\right)}$ (this is also equation 4.7 which we reproduce here):

$$\langle \log P\left(V_1^T\right)\rangle_{\tilde{P}\left(V_1^T\right)} \geq \left\langle \langle \log P\left(H_1^T\right) P\left(V_1^T|H_1^T\right)\rangle_{P_{\text{approx}}\left(H_1^T|V_1^T\right)} + \mathbb{H}\left(P_{\text{approx}}\left(H_1^T|V_1^T\right)\right) \right\rangle_{\tilde{P}\left(V_1^T\right)} \tag{4.10}$$

By introducing $Q$ initialized so that $Q\left(H_1^T\right) = P\left(H_1^T\right)$, the new lower bound of equation 4.9 is equal to the lower bound of equation 4.10. Therefore, the augmented model will have a larger lower bound of its average log likelihood than the lower bound to the average log likelihood of the model $P$, provided that $Q$ can learn $\tilde{Q}$ by increasing its likelihood.

To summarize, the learning procedure of the augmented model (fitting $Q\left(H_1^T\right)$ to $\tilde{Q}\left(H_1^T\right)$) results in approximate maximization of a lower bound that might be larger than the lower bound approximately maximized when learning $P$.

For RBMs, learning one hidden layer at a time works well even if $Q(H)$ is not initialized to be equal to $P(H)$ [24], so in our experiments we did not initialize $Q\left(H_1^T\right) = P\left(H_1^T\right)$.

We can also add further hidden layers in the same way as it is done for RBMs and each time another layer is added we should get a better model, but without the likelihood guarantees analogous to those of the RBM.

If the TRBM does not have any connections between the hidden variables (i.e., if the bias functions $B_H$ does not depend on $H_{t-m}^{t-1}$; we call this the TRBM-HH), then the posterior distribution $P\left(H_1^T|V_1^T\right)$ is factorial. This is a better situation: not only does this model have an exact learning procedure (considering CD an exact learning procedure), but its augmented model *always* has a greater likelihood since the lower bound (equation 4.10) *is equal* to the log likelihood if $Q\left(H_1^T\right) = P\left(H_1^T\right)$, because $P\text{approx}\left(H_1^T|V_1^T\right) = P\left(H_1^T|V_1^T\right)$. Modulo the CD approximation, the TRBM-HH learns by increasing likelihood. The TRBM-HH has the obvious disadvantage of having no direct dependencies among its hidden variables.

A drawback of all of our TRBM models is that it is difficult to obtain a sample from $P\left(V_1^T|H_1^T\right)$ because of the directed connections between $V_1^T$ and the directed connections from $V_1^T$ to $H_1^T$; therefore, sampling from $M_{PQ}$ is difficult as well, and a further approximation must be used. Gibbs sampling can be used in principle, but it is inefficient, for the variables of $V_t$ are *dependent* given $H_1^T$ and the rest of the visible variables, which forces the use of a slower, variable-wise Gibbs sampling. In an RBM we also perform Gibbs sampling, but it is easier, because the conditional distributions over large clusters of variables are factorial (i.e., $P(V|H)$ and $P(H|V)$), so Gibbs sampling is performed in parallel, which cannot be done when sampling $P\left(V_1^T|H_1^T\right)$ from a TRBM.

We use the following approximation when sampling from $P\left(V_1^T|H_1^T\right)$: If $V_1^{t-1}$ is known and $H_{t+1}^T$ is unknown then sampling is easy, for $P\left(V_t|V_1^{t-1}, H_1^t\right)$ is factorial. Therefore, to approximately sample from $P\left(V_1^T|H_1^T\right)$ we let $t$ range over the values 1 to $T$, sampling $V_t$ from $P\left(V_t|V_1^{t-1}, H_1^t\right)$ and ignoring the values of $H_{t+1}^T$ as we loop over $t$.

# Chapter 5

# The Windowed Restricted Boltzmann Machine

In this chapter we describe the Windowed Restricted Boltzmann Machine (WRBM). It is a powerful sequence model that has high-dimensional hidden variables, a simple exact inference procedure and a reasonably efficient learning algorithm. It is very closely related to the Restricted Boltzmann Machine, so learning a multilayered extension of the WRBM is straightforward. The multilayered WRBM can potentially detect and model extremely long range temporal structure.

## 5.1 Model definition

The WRBM is a probabilistic model with hidden variables, defining a joint distribution over visible and hidden sequences of binary vectors.

We define a local probability distribution over short sequences (i.e., windows), and let the product of the local probabilities of all short contiguous subsequences of the long sequence be its unnormalized probability[1].

[1] An unnormalized probability is a quantity proportional to the probability in question.

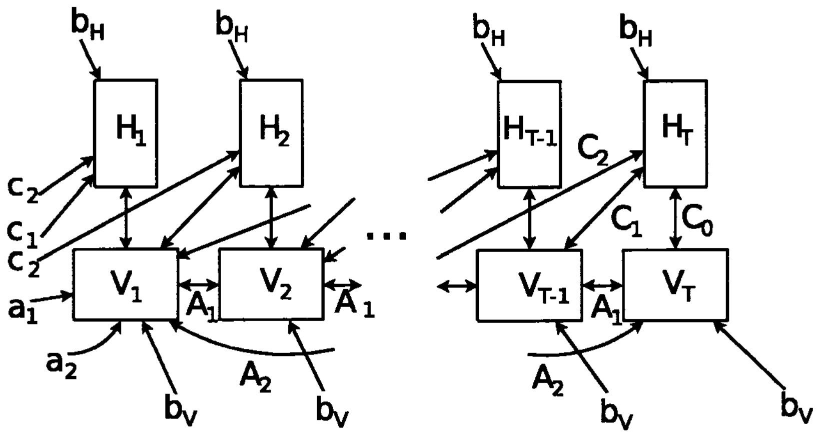

Figure 5.1: A generic WRBM. It is an undirected model: all the connections are symmetric.

The local probability distribution is a Boltzmann Machine, given by the equation

$$P_{\text{local}}\left(V_{t-m}^{t}, H_t\right) = \exp\left(\sum_{i=0}^{m} V_{t-i}' C_i H_t + \sum_{i=1}^{n} V_{t-i}' A_i V_t + V_t' b_V + H_t' b_H\right) /Z, \tag{5.1}$$

where $Z$ is the normalizing constant needed for $P$ to be a proper probability distribution, $C_i$ is the weight matrix of the connections between the hidden and the visible variables, and $A_i$ is a weight matrix of the connections among the visible variables. $n$ and $m$ are structural parameters that determine the structure of the connections and the amount of influence a variable has on its neighbors.

These $P_{\text{local}}$ distributions are multiplied and renormalized (the product of distributions is not a distribution) to yield the sequence model:

$$P\left(V_1^T, H_1^T\right) = \prod_{t=1}^{T} P_{\text{local}}\left(V_{t-m}^{t}, H_t\right) /Z, \tag{5.2}$$

where $Z$ is another partition function. The parameters $A_i$ and $C_i$ are shared among $P_{\text{local}}$, allowing the model to capture sequential regularities. This model is reminiscent of a Convolutional Network [32] through time. Being a product of Boltzmann Machines, the resulting model is also a large Boltzmann Machine with certain parameter constraints, which is seen by rewriting equation 5.2:

$$P\left(V_1^T, H_1^T\right) = \exp\left(\sum_{t=1}^{T}\left(\sum_{i=0}^{m} V'_{t-i}C_iH_t + \sum_{i=1}^{n} V'_{t-i}A_iV_t + V'_tb_V + H'_tb_H\right)\right)/Z. \quad (5.3)$$

Whenever a product $V'_{t-i}C_i$ or $V'_{t-i}A_i$ is used with $t-i \leq 0$ it is replaced by a bias vector $c_i$ or $a_i$, respectively. See also figure 5.1.

This model is an extension of the Spiking Boltzmann Machine [18] that has connections between its visible variables. Not having any connections among its hidden variables, it can be learned efficiently using a variant of the Contrastive Divergence learning procedure. In order to obtain a sample from this model, standard Boltzmann Machine Gibbs sampling is used. The model becomes more interesting once its multilayered versions are introduced.

A disadvantage of this model is the fact that the marginal distribution $P(V_1^t)$ changes with the total number of modelled timesteps. More precisely, if there are $T_0$ frames in one product (i.e., equation 5.2), which we denote by $P_0$, and there are $T_1(\neq T_0)$ frames in another product, which we denote by $P_1$, then the marginal distributions over the first $t$ frames are unequal, i.e. $P_0(V_1^t) \neq P_1(V_1^t)$, even if the parameters of $P_0$ and $P_1$ are the same, which is counterintuitive for those used to directed models.

## 5.2 Learning and The Boltzmann Machine View

Due to the equivalence of our model with a Boltzmann machine that has shared weights, it is easier to work with the Boltzmann machine directly; the gradient with respect to a shared weight is the sum of the standard gradients of the weights computed as if the weights are not shared[2]. To describe learning an WRBM, we simply need to describe the variant of Contrastive Divergence that is used for RBMs with connections between the visible variables, which we do now.

[2] Let $g(x,y)$ be a function. Then the chain rule yields $\frac{\partial g(z,z)}{\partial z} = \frac{\partial g}{\partial x}\big|_{x=z} + \frac{\partial g}{\partial y}\big|_{y=z}$, which is precisely what weight sharing is: the weights $x$ and $y$ are shared, and are equal to $z$.

The derivatives of the Boltzmann Machine are given in section 3.1, and are the difference between the posterior expectations and model expectations: $\frac{\partial \langle \log P(V) \rangle_{\tilde{P}(V)}}{\partial W_{ij}} = \left\langle X^{(i)} X^{(j)} \right\rangle_{P(H|V)\tilde{P}(V)} - \left\langle X^{(i)} X^{(j)} \right\rangle_{P(V,H)}$.

The lack of connections among the hidden variables in the WRBM makes the posterior $P\left(H_1^T | V_1^T\right)$ factorial, so finding the posterior expectation in the expression for the derivative is easy using a Monte Carlo estimate.

Let us suppose that we have an RBM $P(V, H)$. In an RBM, we replace the expectation with respect to $P(V, H)$ by an expectation with respect to the reconstruction distribution $P_{\text{recon}}(V, H)$. To sample from $P_{\text{recon}}(V, H)$ we take the sample $(V, H)$ from the distribution $\tilde{P}(V)P(H|V)$ used to evaluate the posterior expectation, sample $V \sim P(V|H)$ and then $H \sim P(H|V)$, outputting $(V, H)$ as the resulting sample.

We follow a similar path to obtain a sample from $P_{\text{recon}}\left(V_1^T, H_1^T\right)$ of the WRBM. Having obtained a sample $\left(V_1^T, H_1^T\right)$ from $P\left(H_1^T | V_1^T\right) \tilde{P}\left(V_1^T\right)$ we ought to sample $V_1^T$ from $P\left(V_1^T | H_1^T\right)$, according to CD. The connections between the visible variables make sampling from $P\left(V_1^T | H_1^T\right)$ difficult, so instead we perform brief Gibbs sampling from $P\left(V_1^T | H_1^T\right)$, having initialized $V_1^T = 0$. From the new value of $V_1^T$ we sample $H_1^T$ from $P\left(H_1^T | V_1^T\right)$ which is easy, and use the pair $\left(V_1^T, H_1^T\right)$ as a sample from the reconstruction distribution used in CD.

This variant of CD for learning typically works effectively.

## 5.3 Multilayered WRBM

Introducing additional hidden layers to the WRBM model is straightforward and is identical to the way it is done for the RBM. The resulting augmented model together with the variational inequality it defines are *identical* to those of the RBM, so we do not reproduce them here. The only difference from the RBM case is that when sampling from the augmented model we must update $V_1^T$ by Gibbs sampling since we cannot sample

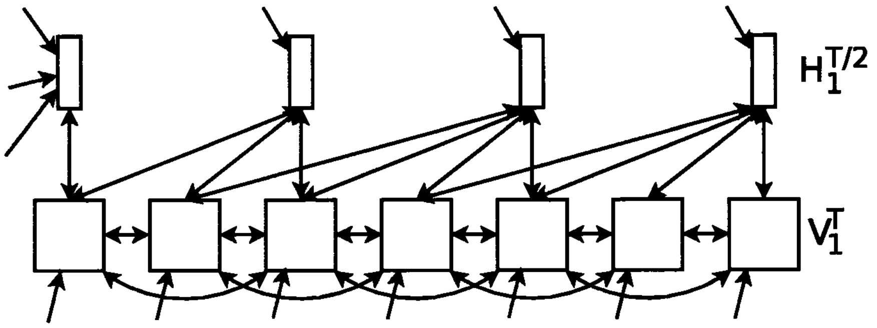

Figure 5.2: The Multigrid-WRBM, a single layer

directly from $P\left(V_1^T|H_1^T\right)$. For the TRBM, we approximate $P\left(V_1^T|H_1^T\right)$ rather than perform Gibbs sampling because it is much more complicated, whereas in the case of the WRBM we approximate $P\left(V_1^T|H_1^T\right)$ by initializing $V_1^T = 0$ and performing a few steps of Gibbs sampling.

## 5.4 Multigrid Multilayered WRBM

A drawback of both the WRBM and the TRBM is that they cannot capture very long range temporal regularities. For instance, these models are unable to learn the probability distribution concentrated on the *single* sequence $(1,0,0,0,0,0,0,0,0,0,0,0,0,0,1)$, even though the HMM learning algorithm can learn it easily if the HMM has 30 hidden states, and there does exist a setting of the parameters for the TRBM (but not the WRBM) that assigns very high probability to the above sequence by implementing counters in the hidden state.

If the hidden variables of our models were made to observe more timesteps, the models could find more structure in the data. It is possible to do this elegantly by slightly modifying the WRBM.

Recall equation 5.2:

$$P\left(V_1^T, H_1^T\right) = \prod_{t=1}^{T} P_{\text{local}}\left(V_{t-m}^t, H_t\right)/Z,$$

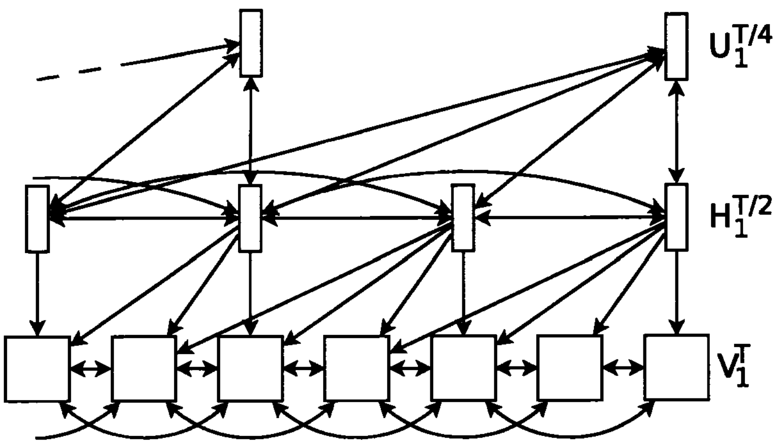

Figure 5.3: The Multilayered Multigrid-WRBM.

We can modify this equation to remove every second hidden state, which results in the Multigrid-WRBM:

$$P\left(V_1^T, H_1^{T/2}\right) = \prod_{t=1, t \text{ is odd}}^{T} P_{\text{local}}\left(V_{t-m}^t, H_{(t+1)/2}\right)/Z. \tag{5.4}$$

So for the standard WRBM, given a visible $V_1^T$ sequence there is a hidden sequence $H_1^T$ of the same length. In the *Multigrid-WRBM* there is a hidden sequence $H_1^{T/2}$ of half the length. This does not produce a better "stand-alone" WRBM. However, the difference between the WRBM and the Multigrid-WRBM becomes much more apparent when such Multigrid-WRBMs are stacked one upon another to create a deep hierarchical model. In this setting, the first hidden layer has a total of $T/2$ frames, the second has $T/4$, and in general, the $n$-th has $T/2^n$ frames, while the number of visible frames remains $T$. This means that the hidden variables of the higher-level Multigrid-WRBMs observe more timesteps, and can detect and model more sequential structure. As with other multilayered models, to obtain a sample from the deep Multigrid-WRBM we need to obtain a sample from the topmost Multigrid-WRBM. So a depth $d$ Multigrid-WRBM produces a $2^d$ reduction in the length of the whole sequence for the WRBM in the highest layer, and each timestep of the WRBM in the highest layer gets to view $2^d$ visible frames, though not directly. As a result, the Multigrid-WRBM model can model

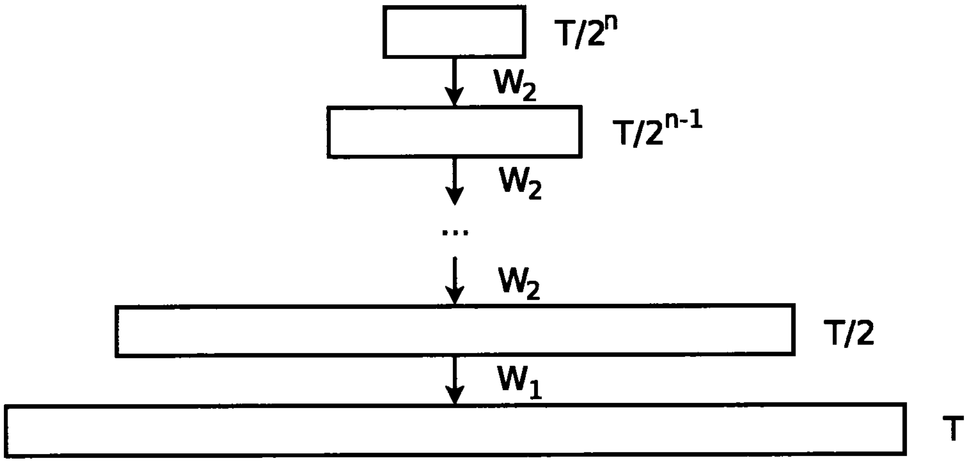

Figure 5.4: The Recursive Multilayered Multigrid-WRBM. All but a constant number layers use the same parameters.

long range structure more easily.

We chose to reduce the density of the hidden layer by a factor of 2 for simplicity. We could have reduced the number of hidden variables by a larger factor than 2 (e.g., the product is taken over $t$ divisible by 3) or by a factor between 1 and 2 (e.g., for every three time steps in $V$ there would be 2 time steps in $H$).

## 5.5 Recursive Multigrid Multilayered WRBM

The Multigrid-WRBM is an improvement over the standard WRBM. However, the Multigrid-WRBM cannot model temporal regularities of arbitrary length. Moreover, when we wish to sample a very long sequence from the Multigrid-WRBM, the Multigrid-WRBM in the topmost hidden layer must sample a $2^d$ times shorter sequence which can, in principle, also be long, and sampling long sequences from the single layered WRBM may be difficult.

If we allowed the number of hidden layers to vary with the length of the sequence, then the Multigrid-WRBM in the top layer would always have to generate short sequences, making sampling easier, and its variables would be able to gain information about the entire sequence, which enables them to capture sequential regularities of arbitrary length. This idea can be implemented by letting the deep Multigrid-WRBM have $\log T$ layers, with all but the first few of it having the same parameters. As a result, the WRBMs with the same parameters act as "sequence expanders": they expand the sequence in their hidden layer without altering the nature of the representation encoded in the sequence. See figure 5.4 for an illustration. To obtain a sample from this model, we first decide on the length of the sequence to be to be sampled, and then sample from a model with the appropriate number of layers. It is not possible to learn such a model with the layer-by-layer learning algorithm that is used so far; instead, the wake sleep algorithm, described in the next chapter, could be used. We did not conduct experiments on this model.

# Chapter 6

# Fine-tuning deep models

In the previous chapters we outlined a general method for learning hierarchical, multilayered models in a greedy, layer by layer fashion. This method does not provide us with a way of fine-tuning the lower layers once the new hidden layers are added. This is undesirable, for the lower layers can be adapted to the presence of the new higher layers to create a better model.

In this chapter we address the problem of fine-tuning the lower layers.

## 6.1 The Wake-Sleep Algorithm

The Wake-Sleep algorithm [19] is used for fine-tuning multilayered models similar to those we have. In the Wake-Sleep algorithm, a complex, multilayered model (which we call the *main model*) is supplemented by an auxiliary inference, or a recognition model, that is used to efficiently obtain an approximate sample from the posterior given a datapoint. The inference model is, by design, relatively easy to sample from, yet is sufficiently flexible that it can produce a good approximation to the true posterior. Given an inference model, learning the main model is easy, as the inference model provides values to all the hidden variables.

The harder part is the learning of the inference model, for as the main model changes,

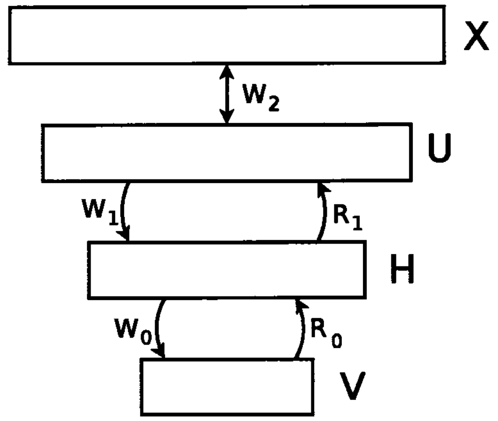

Figure 6.1: A generic model with a recognition network. The weights $W_i$ are those of the main model and the weights $R_0, R_1$ are used for fast approximate inference.

the inference model needs to track these changes in order to remain a good approximation to the posterior of the main model.

We would like the recognition model to output a sample from the posterior of the main model; a straightforward way of learning to do so is by providing the inference model with training examples that consist of datapoints with samples from their posteriors, so the recognition model learns the mapping from a datapoint to (a sample from) its posterior. Such examples can be obtained by sampling from the main model. Indeed, since $P(H|V)P(V) = P(H, V)$, we can view the sampling process from $P$ as first sampling $V$ from $P(V)$ and then sampling $H$ from $P(H|V)$, so $H$ is a sample from $V$'s posterior, even though in reality the sample $(V, H)$ is obtained by first sampling $H \sim P(H)$ and then $V \sim P(V|H)$. This works well whenever obtaining a sample from $P$ is easy, which is the case for the models considered in [19]. Obtaining a sample for each iteration of learning is infeasible for the models we consider, rendering this method unusable for our models. We will describe how to overcome this and other, less severe disadvantages of this method by using the Contrastive Wake-Sleep algorithm [24, 17].

Let us denote the inference model by $Q(H|V)$.

The learning of $Q$ described above minimizes the cost function

$$\langle KL(P(H|V)\|Q(H|V))\rangle_{P(V)}, \tag{6.1}$$

where

$$KL(P(H|V)\|Q(H|V)) = \sum_H P(H|V)\log\frac{P(H|V)}{Q(H|V)}. \tag{6.2}$$

This is the KL-divergence which is a distance measure between two distributions; minimizing equation 6.1 with respect to $Q$ is equivalent to maximizing the average log likelihood

$$\langle \log Q(H|V)\rangle_{P(H,V)} \tag{6.3}$$

so maximum likelihood estimation of $Q$ is equivalent to minimizing a KL-divergence. The reason for writing it in the KL-divergence notation will become clear momentarily. We emphasize that by fitting $Q$ to $P(V,H)$ this way, the recognition model will learn to approximate the posterior distribution of $P$.

## 6.2 The Disadvantages of the Wake-Sleep Algorithm

When using an approximation to the posterior, it is more principled to maximize a lower bound to the average log likelihood with respect to $Q$ in order to find the approximate posterior. Indeed, recall that an approximate posterior and a model can cooperate to create a lower bound to the log likelihood, and the maximization of the lower bound is useful for finding an approximate posterior and for parameter estimation. This lower bound maximization with respect to $Q$ is equivalent to *minimizing* the cost function

$$\langle KL(Q(H|V)\|P(H|V))\rangle_{\tilde{P}(V)} \tag{6.4}$$

with respect to $Q$ while holding the parameters of $P$ fixed. Notice the similarity and the difference between this cost function and the cost function in equation 6.1.

The first difference is that in equation 6.1 the data distribution is $P(V)$ instead of $\tilde{P}(V)$. This implies that whenever $P(V)$ is far from $\tilde{P}(V)$, the datapoints that $Q(H|V)$

will learn will not be the ones it needs to find posteriors of for learning $P$. As a result, it could be that learning may never get beyond its initial stage. This problem becomes less severe, however, as $P(V)$ gets closer to the data distribution.

More severely, when we apply the Wake-Sleep algorithm to our sequence models, obtaining a sample from $P(V,H)$ for each iteration of learning becomes prohibitively expensive.

The second difference is that in equation 6.1 the objective function is an average of $KL(P(H|V)||Q(H|V))$, whereas in equation 6.4 the objective is the average of $KL(Q(H|V)||P(H|V))$. Minimizing $KL(P(H|V)||Q(H|V))$ leads to the mode averaging problem (see figure 6.2 for an informative illustration). For the discussion, assume that the true posterior $P(H|V)$ is multimodal while the approximate posterior $Q(H|V)$ is unimodal. $KL(P(H|V)||Q(H|V))$ is large whenever $Q(H|V)$ is small at locations $H$ at which $P(H|V)$ is large, which follows from the definition of the KL divergence (eq 6.2). As an extreme example, if $Q(H = \mathbf{h}|V) = 0$ while $P(H = \mathbf{h}|V) > 0$, then $KL(P(H|V)||Q(H|V)) = \infty$. Since $Q(H|V)$ cannot have extremely low probability at any region at which $P(H|V)$ is sufficiently large, $Q$ must distribute its single mode around all the modes of $P$, or else there will be regions at which $P(H|V)$ is large but $Q(H|V)$ is extremely small. As a result, if the posterior represents two alternatives, this approximation will not choose any one of them. This is illustrated in figure 6.2.

Equation 6.4, the one which we wish to, but cannot, minimize, has *exactly the opposite* behavior, for the function $KL(Q(H|V)||P(H|V))$ is minimized. By switching the KL-divergence $Q$ is forced to select a single mode and to put most of its mass on it, for $KL(Q(H|V)||P(H|V))$ is high whenever $Q(H = \mathbf{h}|V)$ is large but $P(H = \mathbf{h}|V)$ is very small, which is desirable.

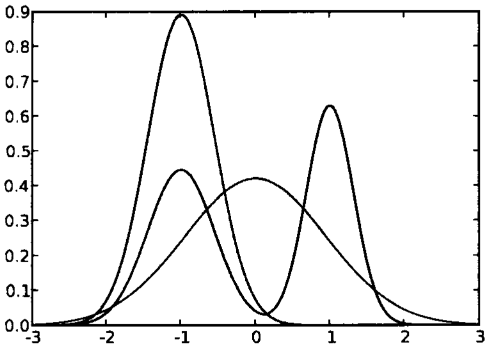

Figure 6.2: The mode averaging problem: The blue distribution represents the true posterior, the green distribution is $\text{argmin}_Q KL(Q(H|V)||P(H|V))$ and the red distribution is $\text{argmin}_Q KL(P(H|V)||Q(H|V))$. This is especially undesirable when each mode represents a choice, such as left or right. The red distribution will necessarily choose center.

## 6.3 The Contrastive Wake-Sleep Algorithm

There is an elegant way to alleviate the need to be sample from $P(V, H)$, the mode averaging problem, and the fact that the distribution used is $P(V)$ and not $\tilde{P}(V)$, in a single modification to the algorithm.

Recall that $P(H, V)$ is a multilayered model, where there are several layers of hidden variables (for instance, the deep models we obtain by augmenting the TRBM or the WRBM). The idea of Contrastive Wake-Sleep is the following: given a training point $V$ randomly selected from the training set, we first obtain a sample from the approximate posterior $Q(H|V)$ and update the parameters of $P$ which is also done in the standard Wake-Sleep algorithm. Then, we randomly (and slightly) perturb the variables of $H$ which are at the highest layer (which we will denote by $H_h$, a mnemonics for *highest*) and sample from $P(V, H|H_h)$ which is easy for the deep models we consider (it is sampling $P(H_h)$ that is hard for our models). This is similar to reconstructing all the variables $(V, H)$ from $H_h$. The reconstructed $(V, H)$ does not satisfy that $H$ is a sample from $V$'s posterior, yet it is plausible that $H$ is close to being one.

However, as the reconstruction is from $H_h$, there is no need to *sample* from $P(H_h)$, which is the main advantage gained from using the Contrastive Wake-Sleep algorithm. Moreover, as the inference model becomes accurate, the reconstructions will always be more similar to the respective datapoint, and thus closer to the data distribution than a sample from the model.

Secondly, $Q$ no longer gets penalized when it distributes its probability on only one mode of the marginal posterior $P(H_h|V)$. Indeed, if all the samples from $Q(H_h|V)$ are near only one mode of $P(H_h|V)$, then the reconstructions from $H_h$ will be near the respective datapoint and $Q$ will not be influenced by the presence of the other modes in $H_h$. Had we sampled $(V, H)$ from $P$, $Q$ would have to learn to output samples from the other modes of $P(H_h|V)$ as well, which is avoided. We do not make a claim regarding the variables not in the highest layer $H_h$.

## 6.4 Contrastive Wake-Sleep applied to Deep Sequence Models

Let us see how these ideas apply to our models. Recall that our learning of the deep models uses a certain approximation to the posterior, which we now consider as our recognition model. The inference and main models have the same parameters, but once we split the main model into the inference model and a main model explicitly, the above ideas apply directly.

Let us describe the multilayered models differently from before. There are $m$ layers of variables, $V = H^{\langle 0 \rangle}$, $H^{\langle 1 \rangle}, \ldots, H^{\langle m \rangle}$, where for uniformity, we denote $V$ by $H^{\langle 0 \rangle}$. In general, $H^{\langle i \rangle}$ is the $i$th hidden layer, with $P^{\langle i \rangle}$ being the TRBM or the WRBM that models the distribution of $H^{\langle i \rangle}$ and $H^{\langle i+1 \rangle}$. The joint distribution of the deep model $M$

is

$$M\left(V,H^{\langle 1\rangle},\ldots,H^{\langle m\rangle}\right)=\prod_{i=0}^{m-2}P^{\langle i\rangle}\left(H^{\langle i\rangle}|H^{\langle i+1\rangle}\right)P^{\langle m-1\rangle}\left(H^{\langle m-1\rangle},H^{\langle m\rangle}\right), \tag{6.5}$$

and inference is performed via

$$M_{\text{approx}}\left(H^{\langle 1\rangle},\ldots,H^{\langle m\rangle}|V\right)=\prod_{i=0}^{m-1}P^{\langle i\rangle}_{\text{approx}}\left(H^{\langle i+1\rangle}|H^{\langle i\rangle}\right) \tag{6.6}$$

We replace $M_{\text{approx}}$ with its copy whose parameters do not depend on the parameters of $M$, yet happen to be equal to them. Let us denote by

$$Q\left(H^{\langle 1\rangle},\ldots,H^{\langle m\rangle}|V\right)=\prod_{i=0}^{m-1}Q^{\langle i\rangle}\left(H^{\langle i+1\rangle}|H^{\langle i\rangle}\right) \tag{6.7}$$

the recognition model that replaces $M_{\text{approx}}$, whose parameters are initialized to those of $M_{\text{approx}}$ and will change during learning. Learning $M$ and $Q$ proceeds exactly according to the Contrastive Wake-Sleep algorithm. In particular, if $M$ has already been trained in a greedy way, then the wake-sleep algorithm starts with reasonably accurate main and recognition models.

. We reiterate that to learn $M$ we first randomly select $V$ from the training set, and use $Q$ to perform approximate inference so that all the variables receive values; $M$ increases the log probability it assigns to these values of $H^{\langle i\rangle}$. Then $H^{\langle m\rangle}$ is randomly perturbed and from the value new of $H^{\langle m\rangle}$ the values of $H^{\langle i\rangle}$ are reconstructed using $M$ (i.e., we sample $P(\{H^{\langle i\rangle}\}_{0\leq i\leq m-1}|H^{\langle m\rangle})$), and $Q$ learns by increasing the log probability it gives to the values of $H^{\langle i\rangle}$ obtained by the reconstruction.

To conclude this section, we provide the equations for the gradients needed for learning:

First, note that

$$\log M\left(V,H^{\langle 1\rangle},\ldots,H^{\langle m\rangle}\right)=\sum_{i=0}^{m-2}\log P^{\langle i\rangle}\left(H^{\langle i\rangle}|H^{\langle i+1\rangle}\right)+\log P^{\langle m-1\rangle}\left(H^{\langle m-1\rangle},H^{\langle m\rangle}\right) \tag{6.8}$$

so we merely need to find the gradients of $\log P^{\langle i\rangle}\left(H^{\langle i\rangle}|H^{\langle i+1\rangle}\right)$ and $\log P^{\langle m-1\rangle}\left(H^{\langle m-1\rangle},H^{\langle m\rangle}\right)$ with respect to $M$'s parameters. We have already shown how to approximately compute the derivatives of $\log P^{\langle m-1\rangle}\left(H^{\langle m-1\rangle}, H^{\langle m\rangle}\right)$ in the sections of the multilayered models of the TRBM and the WRBM, for $P^{\langle m-1\rangle}\left(H^{\langle m-1\rangle}, H^{\langle m\rangle}\right)$ is either a TRBM or an WRBM.

The derivatives of $\log P^{\langle i\rangle}\left(H^{\langle i\rangle}|H^{\langle i+1\rangle}\right)$ can be easily approximated for both the WRBM and the TRBM.

For the WRBM, $\log P^{\langle i\rangle}\left(H^{\langle i\rangle}|H^{\langle i+1\rangle}\right)$ is the distribution of a Boltzmann Machine without hidden variables. Conceptually, the derivative is that of the Boltzmann Machine, except that in practice the negative expectations are replaced by a reconstruction distribution, obtained from brief Gibbs sampling of the visible variables. In this equation, the fact that we condition on $H^{\langle i+1\rangle}$ implies that the biases to the variables $H^{\langle i\rangle}$ are parameterized similarly to the bias functions of the TRBM. As a result, the equations are the following:

$$\frac{\partial \mathcal{L}}{\partial (C_k)_{uv}} = \sum_{t=k+1}^{T} \left( H_{t-k}^{\langle i\rangle(u)} - \left\langle H_{t-k}^{\langle i\rangle(u)} \right\rangle_{P^{\langle i\rangle}\left(H^{\langle i\rangle}|H^{\langle i+1\rangle}\right)} \right) H_t^{\langle i+1\rangle(v)} \tag{6.9}$$

$$\frac{\partial \mathcal{L}}{\partial (A_k)_{uv}} = \sum_{t=k+1}^{T} H_{t-k}^{\langle i\rangle(u)} H_t^{\langle i\rangle(v)} - \left\langle H_{t-k}^{\langle i\rangle(u)} H_t^{\langle i\rangle(v)} \right\rangle_{P^{\langle i\rangle}\left(H^{\langle i\rangle}|H^{\langle i+1\rangle}\right)} \tag{6.10}$$

The WRBM is parameterized with the matrices $A_k$ and $C_k$ for some values of $k$. In this equation, $\mathcal{L}$ stands for $\log P^{\langle i\rangle}\left(H^{\langle i\rangle}|H^{\langle i+1\rangle}\right)$.

The TRBM is slightly different. When sampling from a deep TRBM, the conditional distribution $P^{\langle i\rangle}\left(H^{\langle i\rangle}|H^{\langle i+1\rangle}\right)$ is difficult to sample from, so it is approximated similarly to the filtering distribution of the TRBM: we loop over the timestep $t$ from 1 to $T$, sampling $H_t^{\langle i\rangle}$ from $P^{\langle i\rangle}\left({H^{\langle i\rangle}}_t|{H^{\langle i+1\rangle}}_1^t, {H^{\langle i\rangle}}_1^{t-1}\right)$ while ignoring ${H^{\langle i\rangle}}_{t+1}^T$. This approximate sampling process is that of a Sigmoid Belief Network [36]. The derivatives of $\log P^{\langle i\rangle}\left(H^{\langle i\rangle}|H^{\langle i+1\rangle}\right)$ when $P^{\langle i\rangle}$ is the approximate conditional TRBM are the following:

$$\frac{\partial \mathcal{L}}{\partial (C_0)_{uv}} = \sum_{t=1}^{T} \left( H_t^{\langle i\rangle(u)} - \left\langle H_t^{\langle i\rangle(u)} \right\rangle_{P^{\langle i\rangle}\left(H^{\langle i\rangle}|H^{\langle i+1\rangle}\right)} \right) H_t^{\langle i+1\rangle(v)} \tag{6.11}$$

$$\frac{\partial \mathcal{L}}{\partial (A_k)_{uv}} = \sum_{t=k+1}^{T} \left( H_t^{\langle i\rangle(v)} - \left\langle H_t^{\langle i\rangle(v)} \right\rangle_{P^{\langle i\rangle}\left(H^{\langle i\rangle}|H^{\langle i+1\rangle}\right)} \right) H_{t-k}^{\langle i\rangle(u)}, \tag{6.12}$$

During the approximate sampling the connections $C_k$ for $k > 0$ are not used. $\mathcal{L} = \log P^{\langle i\rangle}\left(H^{\langle i\rangle}|H^{\langle i+1\rangle}\right)$, as before.

For the inference model $Q$, the distributions $Q = P^{\langle i \rangle}_{\text{approx}}\left(H^{\langle i+1 \rangle} | H^{\langle i \rangle}\right)$ are exactly[1] the same (but reflected symmetrically) as the conditional of the main models, so the derivatives are very similar, conceptually. For the WRBM $P^{\langle i \rangle}_{\text{approx}}\left(H^{\langle i+1 \rangle} | H^{\langle i \rangle}\right)$ is a Boltzmann Machine without any connections which is a factorial distribution, and its derivatives are:

$$\frac{\partial \mathcal{L}}{\partial (C_k)_{uv}} = \sum_{t=k+1}^{T} H_{t-k}^{\langle i \rangle (u)} \left( H_t^{\langle i+1 \rangle (v)} - \left\langle H_t^{\langle i+1 \rangle (v)} \right\rangle_{Q^{\langle i \rangle}\left(H^{\langle i+1 \rangle} | H^{\langle i \rangle}\right)} \right), \tag{6.13}$$

where $\mathcal{L} = \log Q^{\langle i \rangle}\left(H^{\langle i+1 \rangle} | H^{\langle i \rangle}\right)$ and the $A_k$ matrices are not used in the approximate inference.

For the TRBM we get the derivatives

$$\frac{\partial \mathcal{L}}{\partial (C_k)_{uv}} = \sum_{t=k+1}^{T} H_{t-k}^{\langle i \rangle (u)} \left( H_t^{\langle i+1 \rangle (v)} - \left\langle H_t^{\langle i+1 \rangle (v)} \right\rangle_{Q^{\langle i \rangle}\left(H^{\langle i+1 \rangle} | H^{\langle i \rangle}\right)} \right) \tag{6.14}$$

$$\frac{\partial \mathcal{L}}{\partial (B_k)_{uv}} = \sum_{t=k+1}^{T} H_{t-k}^{\langle i+1 \rangle (u)} \left( H_t^{\langle i+1 \rangle (v)} - \left\langle H_t^{\langle i+1 \rangle (v)} \right\rangle_{Q^{\langle i \rangle}\left(H^{\langle i+1 \rangle} | H^{\langle i \rangle}\right)} \right) \tag{6.15}$$

Note that here the weights $C_k$ are used because they are used in the process of the approximate filtering, and the parameters $A_k$ are not used in the approximate inference.

## 6.5 Mixed multilayered models

In the description of the multilayered models, we have a model $P(V, H)$ and we use a model $Q(H)$ to fit the aggregated posterior of $P$ (see section 3.3; the notation differs from the previous section and is the one used in the descriptions of the multilayered models used throughout the text). In particular, $Q$ would be from the same model family as $P$ in our descriptions. However, $Q(H)$ can be an arbitrary model. The only reason for $Q$ to be similar to $P$ is so that $Q(H)$ could be initialized to be equal to $P(H)$. This initialization is desirable only because this way the lower bound to the log likelihood

[1] This is not quite true for the approximate filtering of the TRBM, but we nonetheless use the same derivatives for their simplicity.

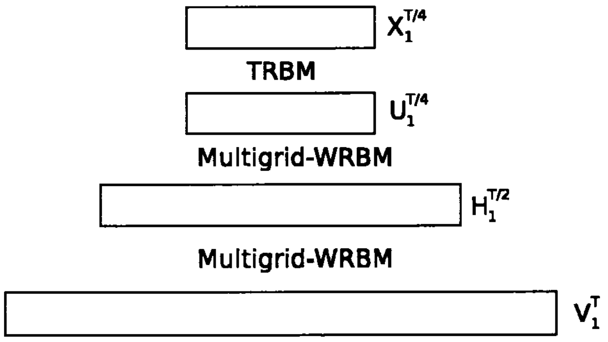

Figure 6.3: A Mixed Multilayered model, where every layer consists of either an (Multigrid-) WRBM or a TRBM

of the augmented model $\langle \log M_{PQ}(V) \rangle_{\tilde{P}(V)}$ is equal to the log likelihood of $P$; this way know that the augmented model fits the data better than the initial model. However, we do not initialize $Q$ this way, so it is possible for $Q$ to be any model, as long as it is sufficiently flexible.

All of this implies that when constructing multilayered models, we need not select all the models from one family. For example, we could fit an WRBM and then fit a TRBM for the second hidden layer, and then fit a Multigrid-WRBM for the third hidden layer. A particularly appealing model consists of several Multigrid-WRBMs for the first few hidden layers with a TRBM in the top layer.

This provides many more possible models, and we believe that any such combination produces good results. Moreover, the resulting deep model can be straightforwardly fine-tuned exactly the way it is described in the previous section.

# Chapter 7

# Experiments

In this chapter we demonstrate that the large number of models we introduced can learn accurate probability models of highly complex sequences efficiently. The implementation is available at the website www.cs.utoronto.ca/~ilya/MS_thesis/.

## 7.1 The Dataset

We used a synthetic dataset of videos of 2 balls bouncing in a bounded box. A typical sequence can be seen in the website. We can generate as many such sequences as necessary, so there is no danger of overfitting the data. For our experiments we used videos of resolution 20 by 20, thus each frame is very high-dimensional. The data is highly non-linear in the positions of the balls due to them bouncing off the walls and off each other, and indeed, the problem is difficult to model even when the balls are represented with the real-valued coordinates of their centers [34]. We apply our methods to the much harder problem of modeling the *pixels* of these videos.

We generate a video sequence by simulating the trajectories of the bouncing balls in the box whose dimensions are $10 \times 10$ units (not pixels), initializing their positions randomly and velocities to random vectors. For each frame, we let the vectors $x_1$, $x_2$ denote the positions of the centers of the balls and let $r = 2$ be their radius relative to the coordinates of the $10 \times 10$ box. We set the intensity of each pixel of the frame whose position in the $10 \times 10$ box is $p$ to the value $\min\left(1, \sum_{i=1}^{2} \exp\left(-\left(\|p - x_i\|^2/r^2\right)^4\right)\right)$.

## 7.2 The models we test

We have introduced a large number of models, and with the ways of creating multilayered models there is a very large space of possible models. Therefore, we experiment with the small number of representative models, enumerated below:

1. The first model has three layers of TRBMs. The TRBM in the first layer has 400 visible variables (this is the dimensionality of the input) and 200 hidden variables per timestep. The second TRBM has 200 visible variables and 400 hidden variables per timestep, and the third TRBM has 400 visible variables and 200 hidden variables. The number of variables per layer was chosen by trial and error. In particular, we chose the number of variables in the different layers to alternate (400,200,400,200) [24]. It is likely that other numbers of variables per layer would yield very similar results.

   Each variable of each TRBM can directly observe 4 previous timesteps, which amounts to $m = 4$ in the definition of the bias functions (equation 4.3).

2. The second model is identical to the model (1) above, except that each TRBM has no connections between the hidden variables; we denote this TRBM by TRBM-HH. This is a an interesting model, for it has an exact inference procedure and its multilayered theory is more sound.

3. The third model is identical to the model in (1), except that each TRBM has no connections among the visible variables, and the hidden variables cannot observe the previous visible time steps (i.e., $A_k = 0$ and $C_k = 0$ for $k > 0$). This model

is important, showing that the TRBM can learn to use the hidden state alone to propagate information through time.

4. The fourth model has three layers of WRBMs. The WRBMs used in this model have the same numbers of hidden and visible variables per timestep as the TRBMs in (1). The hidden variables of each WRBM can influence five nearby visible variables, and the visible variables can directly influence only their immediate neighbors. This is equivalent to having set $m = 4$ and $n = 1$ in equation 5.1 that defines the WRBM.

5. The fifth model is similar to (4), except that every WRBM used is a Multigrid-WRBM with the same dimensions, where each Multigrid-WRBM has a factor of 2 reduction in the length of the sequence.

6. The final model we test is a mixed model. This mixed model has three layers; the first two layers are the Multigrid-WRBMs that are in the first two layers of model (5), and the final layer is the full TRBM that is in model (1).

We fitted each model by learning the layers sequentially, then fine-tuning all the layers simultaneously using the Contrastive Wake-Sleep algorithm.

## 7.3 Learning details

The general method used for finding the parameters is gradient descent with a slightly nonstandard momentum. Let $\theta_n$ be the state of the parameters on the $n$-th iteration, and let the value of the momentum be $\mu = 0.9$. We maintain the velocities of the parameters $\theta_n$ at timestep $n$ which are called $v_n$. Let $d_n$ be the stochastic approximation to the gradient of the objective function we wish to optimize at time $n$, which is computed the way described in the previous chapters. For each timestep, we update the velocities

$$v_{n+1} = \mu v_n + \varepsilon d_n,$$

and then update the parameters $\theta_{n+1} = \theta_n + Sv_n$, where $\varepsilon$ is the learning rate and $S$ is the speedup factor that increases during learning, and is initially initialized to 1. The difficulty in using this optimization is that we need to initialize the parameters $\theta_0$, to choose $\varepsilon$, to decide when to increase $S$, and to choose the total number of iterations appropriately. These difficult problems are not addressed in this thesis; instead, these choices are made by trial and error, and are suboptimal.

We noticed that $S$ can be increased as learning progresses without undesirable side-effects. $S = 1$ at the beginning of learning when the parameters "break symmetry". After this, we can safely increase $S$ up to 10 to 100. We currently do not know how to detect automatically when this symmetry breaking occurs; we can observe it when the images of the incoming connections to the hidden variables start looking like filters for the model of the first hidden layer. Using this visual information, we can determine when to increase the speedup factor, and use the resulting schedule for the models at the higher layers as well. This has been applied very successfully to the sequential learning of the layers. The Contrastive Wake-Sleep algorithm is much less tolerant to changes in the speedup factor and often finds extremely bad solutions when the speedup factor is increased by such a factor, so we did not increase it during learning.

When training each layer separately, we ran the algorithm for 5000 iterations. At each iteration we compute the approximate gradients given a video of 100 frames, and normalize the learning rate by dividing it by the length of the video. For training the TRBMs, we initialized the learning rate with 0.005, and doubled the speedup factor at iterations 100,200,500, and 1000. For the WRBMs and the mixed model we initialized the learning rate to 0.001 and doubled the speedup factor at the same iterations as the TRBMs. As a result of using this schedule, learning requires approximately 10 times less iterations. The WRBMs turned out to be more sensitive to large learning rates than the TRBMs. Additionally, models that have connections among the visible variables fail to learn if these connections are updated with the same learning rate, so we reduced the learning rate of these connections by a factor of 10 for all models. The parameters of the models were initialized using very small random numbers.

When fine-tuning the models, we ran the Contrastive Wake-Sleep algorithm for 10000 iterations using learning rate of 0.0005 for the parameters of the main, generative model, except for the parameters of the TRBM/WRBM in top of the hierarchy, which had a learning rate of 0.0025. The learning rate 0.001 was used for the parameters of the recognition model. As before, the learning rate of the connections among the visible variables (i.e., the $A_k$ connections) of each model was a factor of 10 smaller than the corresponding learning rate of the other connections (i.e., the $B_k$ and the $C_k$ connections), and each learning rate was divided by the length of the training sequence, as before. The speedup factor of the Contrastive Wake-Sleep algorithm was fixed throughout learning, and the learning rates were of the TRBM and the WRBM were equal.

As a result, the Contrastive Wake-Sleep algorithm usually improved the quality of the model. However, we were unable to fit the models using the Contrastive Wake-Sleep algorithm starting with a random initialization.

We use different learning rates in order to speed up learning as much as possible, since different components of the model can learn in different speeds. Indeed, had we restricted all the learning rates to be 0.0005, we would have to increase the number of iterations to obtain analogous results.

Finally, even a single TRBM can learn an extremely accurate model of a similar dataset when trained for a very large ($> 10^6$) number of iterations; samples from this TRBM are shown on the website www.cs.utoronto.ca/~ilya/MS_thesis/.

## 7.4 Results

Samples from all the models are available at the website. The results show that the models are capable of learning reasonably accurate models of the training set, and that adding hidden layers and fine-tuning typically improves the quality of the samples from the models. We did not quantitatively evaluate the quality of the samples because the models can generate trajectories that are approximately correct but can considerably deviate from any real trajectory of true bouncing balls, particularly when the balls bounce off each other. Therefore, visual inspection is preferable in this situation.

Our models are to some degree undertrained, in part because we wished to keep learning time sufficiently low (less than a day). Indeed, by training for a considerably longer period of time (of the order of two weeks) we can get substantially more accurate models.

Figures 7.2 and 7.3 display the connections learned by the TRBM.

An additional task that can be done with our models is denoising. We provide the model with a video, and use the model to denoise it by performing infernece and reconstructing some chosen hidden layer. See figure 7.1.

The noise is constructed the following way: at each frame, each pixel is independently selected with probability 0.004. For each selected pixel, a ball of smaller radius (which fits in a 5×5 box) is shown for 6 frames in that location. The noise can overlap, and it is possible, for instance, for the same pixel to be selected in two consecutive frames (and thus to have the same smaller ball appear for 7 frames). The noise starts in steady state, which is why the noise appearing in the first frame may remain visible for less than 6 frames.

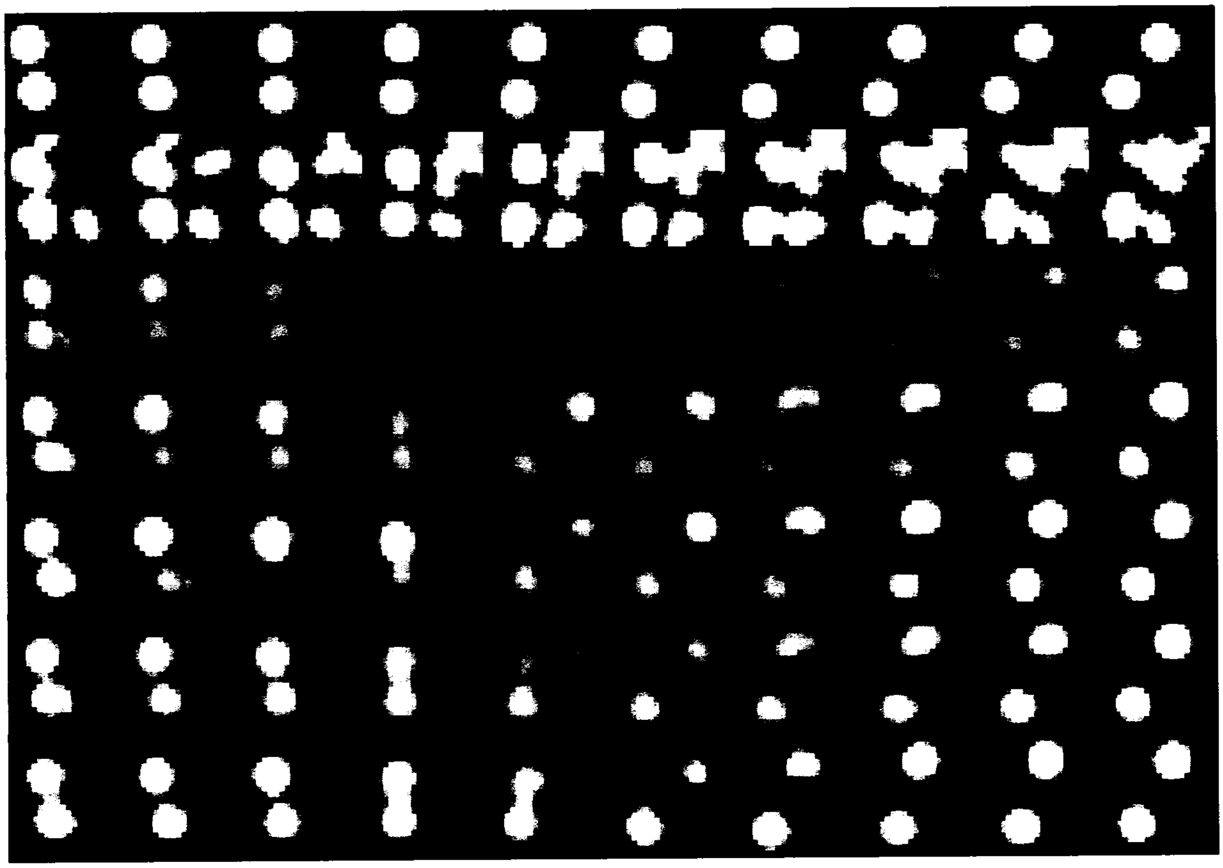

Figure 7.1: Denoising using a TRBM-HH. The first row shows the data to be denoised, the second row shows the noisy data, the third row shows the reconstruction using a single layer of the TRBM-HH, the fourth row shows the reconstruction from the second layer of the TRBM-HH, and the fifth row shows the reconstruction from the third hidden layer of the TRBM-HH. The sixth row shows the result of denoising the *third row* with the first layer of the TRBM-HH (i.e., the already denoised sequence using a single layer of the TRBM-HH) and the final row shows denoising using the first layer of the TRBM-HH of the sixth row. This demonstrates that "iterative denoising" with a single layered model has a similar effect to that of using additional hidden layers.

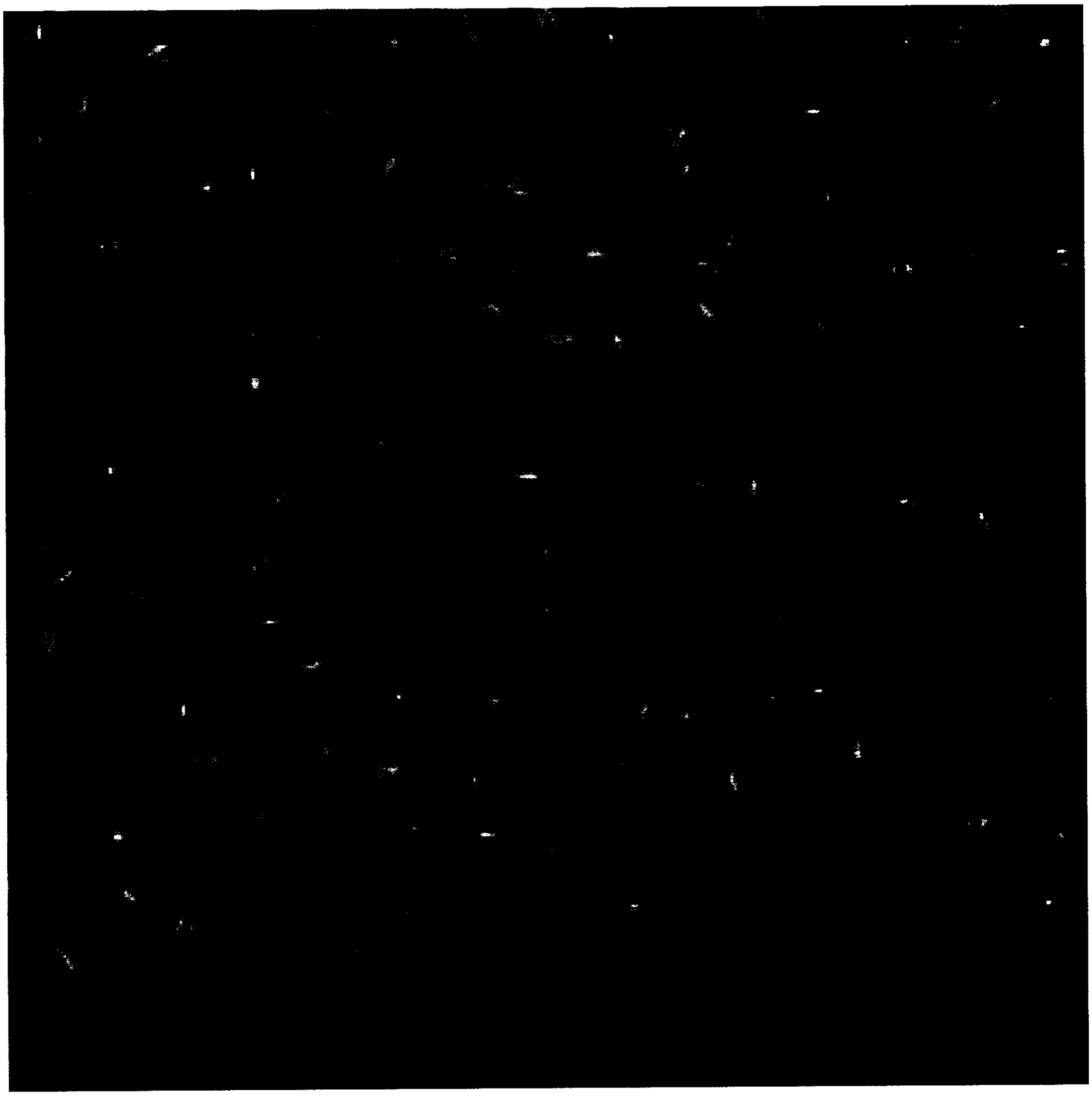

Figure 7.2: The connections between the frame and its corresponding hidden variables (i.e., the connections between $V_t$ and $H_t$, or $C_0$) in a fully connected TRBM. Each square corresponds to a hidden variable, and its incoming connections are aligned in the shape of the image; thus we can see what kind of inputs activate the hidden variables. The connections learned by the WRBM do not look as appealing.

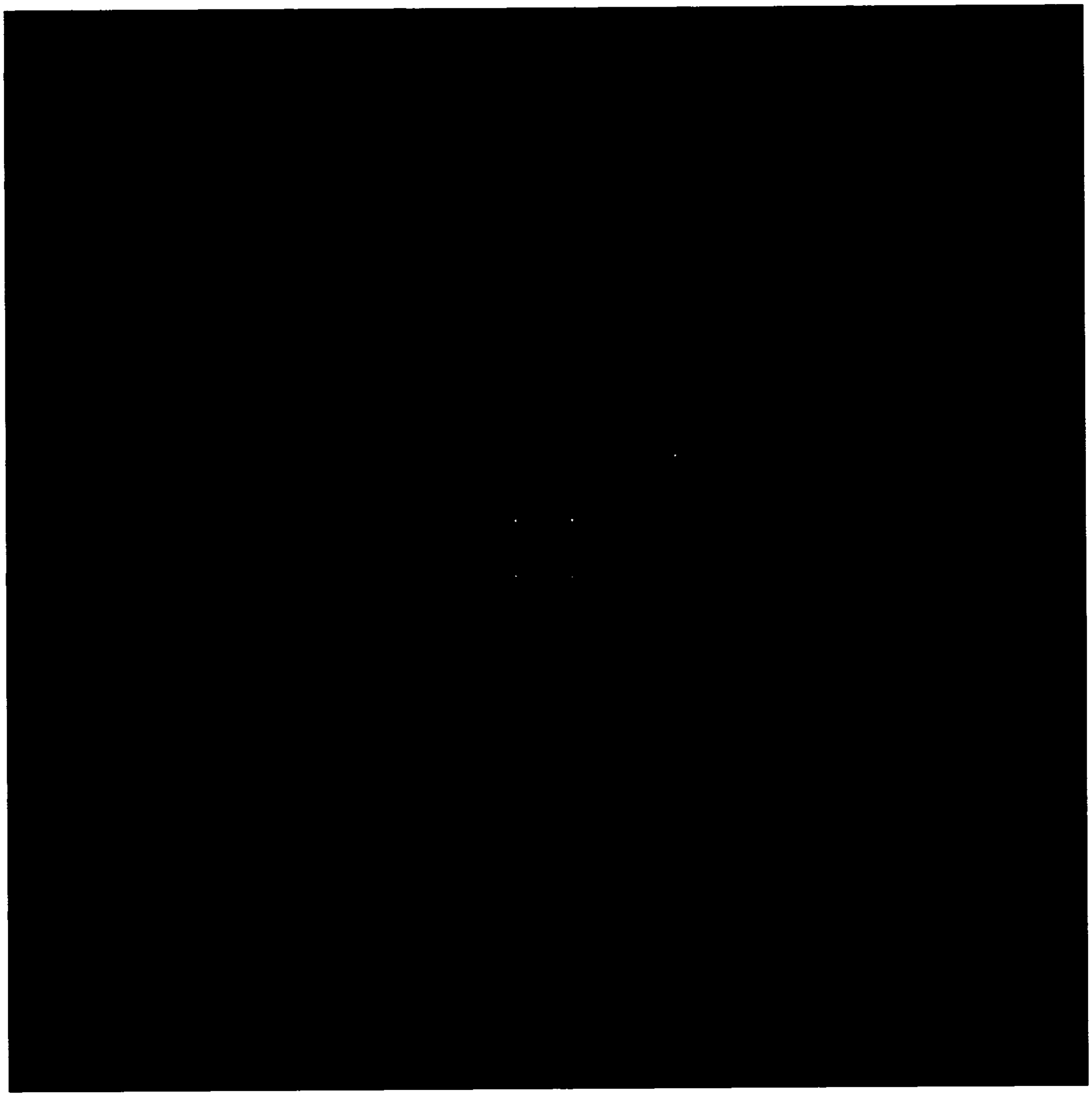

Figure 7.3: The connections between two adjacent visible frames (the connections between $V_t$ and $V_{t-1}$, or $A_1$) in a fully connected TRBM. Each square corresponds to a pixel in the image, and the image within shows the connections from this pixel to the pixel in the next frame.

# Chapter 8

# Conclusions and Discussions

In this thesis we introduced several probabilistic sequence models and demonstrated their power by fitting them to high-dimensional synthetic video sequences. As a result, the models learned accurate probabilistic models of the data.

A model similar to the multilayered TRBM could be useful for the construction of an intelligent agent, such as a mouse or a dog, for the agent needs to be able to model and predict the environment it is in and to choose actions that lead to the outcome it desires. The TRBM can learn online, so it is appropriate in this setting, and the features it learns may be useful for achieving the tasks of the agents. The TRBM, however, cannot learn to retain information over many timesteps, a fact that makes the TRBM less useful for this task. The Multigrid WRBM is more capable of learning long range temporal regularities, but it is not suitable for online learning and filtering the way a TRBM is.

Therefore, we wish to modify the TRBM model to have some kind of longer-term memory, similar to that of the Long-Short Term Memory model [25], or to have connections that change during the operation of the network and learn to store important contextual information for a large number of timesteps. A similar idea was introduced in [20].

The most important drawback of our models is that they are difficult to use in practice. While the algorithms are not (too) difficult to implement correctly, finding the appropriate learning rate and the correct number of iterations is a much more challenging task. In the reported experiments we used a fixed number of iterations and a learning rate found by trial and error. This is impractical since every task requires a large amount of manual labor for finding the the learning rate schedule. In our initial experiments we used a constant learning rate, and learning *a single layer* to reach performance similar to the one we have obtained took *several days*, for a large learning rate was unusable. We later realized that we could use a small learning rate for the beginning of learning and a much larger learning rate for the rest of learning. Since we cannot compute a cost function, it is difficult to implement adaptive learning rate, where the learning rate increases until the cost function being minimized starts exhibiting oscillatory behavior. One possible way of implementing an adaptive learning rate is by increasing the learning rate and checking whether the weights start oscillating, although checking this is non-trival; if they do, the learning rate is too large, but if they do not, the learning rate can be increased. Stochastic Meta Descent [45] tries to solve the same problem, but it is inapplicable to our models due to our inability to evaluate the objective function, the log likelihood.

In conclusion, a powerful time-series model needs to have

- a high-dimensional, distributed, hidden state
- complex non-linear dynamics in the hidden variables
- a hierarchical hidden representation, possibly with feedback connections
- an efficient learning algorithm

In this thesis we introduced models that, to some extent, satisfy these requirements.

# Bibliography

[1] M. Arulampalam, S. Maskell, N. Gordon, T. Clapp, D. Sci, T. Organ, and S. Adelaide. A tutorial on particle filters for online nonlinear/non-Gaussian Bayesian tracking. *IEEE Transactions on Signal Processing*, 50(2):174–188, 2002.

[2] P. Bartlett and M. Anthony. *Neural Network Learning: Theoretical Foundations.* Cambridge University Press, 1999.

[3] A. Bell and T. Sejnowski. The 'independent components' of natural scenes are edge filters. *Vision Research*, 37(23):3327–3338, 1997.

[4] Y. Bengio, P. Simard, and P. Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166, 1994.

[5] M. Black and A. Jepson. EigenTracking: Robust Matching and Tracking of Articulated Objects Using a View-Based Representation. *International Journal of Computer Vision*, 26(1):63–84, 1998.

[6] X. Boyen and D. Koller. Tractable inference for complex stochastic processes. In G. F. Cooper and S. Moral, editors, *Proceedings of the 14th Conference on Uncertainty in Artificial Intelligence (UAI-98)*, pages 33–42, San Francisco, July 24–26 1998. Morgan Kaufmann.

[7] A. Brown and G. Hinton. Products of hidden markov models. *Proceedings of Artificial Intelligence and Statistics*, pages 3–11, 2001.

[8] M. Carreira-Perpinan and G. Hinton. On contrastive divergence learning. *Artificial Intelligence and Statistics*, 2005.

[9] C. Cortes and V. Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, 1995.

[10] F. Crick and G. Mitchison. The function of dream sleep. *Nature*, 304(5922):111–114, 1983.

[11] N. Cristianini and J. Shawe-Taylor. *An Introduction to Support Vector Machines and Other Kernel-based Learning Methods.* Cambridge University Press, 2000.

[12] P. Dagum and M. Luby. Approximating probabilistic inference in Bayesian belief networks is NP-hard. *Artificial Intelligence*, 60(1):141–153, 1993.

[13] A. Dempster, N. Laird, and D. Rubin. Maximum Likelihood from Incomplete Data via the EM Algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, 39(1):1–38, 1977.

[14] Z. Ghahramani and M. I. Jordan. Factorial hidden Markov models. In D. S. Touretzky, M. C. Mozer, and M. E. Hasselmo, editors, *Advances in Neural Information Processing Systems, NIPS*, volume 8, pages 472–478. MIT Press, 1995.

[15] Z. Ghahramani and S. Roweis. A unifying review of linear Gaussian models. *Neural Computation*, 11(2):305–345, 1999.

[16] P. Graham. A plan for spam. http://www.paulgraham.com/spam.html, October 2002.

[17] G. Hinton. What kind of a graphical model is the brain? *IJCAI2005, Award lecture*, 2005.

[18] G. Hinton and A. Brown. Spiking Boltzmann Machines. *Advances in Neural Information Processing Systems*, 12, 2000.

[19] G. Hinton, P. Dayan, B. Frey, and R. Neal. The wake-sleep algorithm for unsupervised neural networks. *Science*, 268(5214):1158–1161, 1995.

[20] G. Hinton and D. Plaut. Using fast weights to deblur old memories. *Proceedings of the 9th Annual Conference of the Cognitive Science Society*, pages 177–186, 1987.

[21] G. Hinton and R. Salakhutdinov. Reducing the Dimensionality of Data with Neural Networks. *Science*, 313(5786):504, 2006.

[22] G. Hinton and T. Sejnowski. Learning and relearning in Boltzmann machines. *Mit Press Computational Models Of Cognition And Perception Series*, pages 282–317, 1986.

[23] G. E. Hinton. Training products of experts by minimizing contrastive divergence. *Neural Computation*, 14(8):1771–1800, 2002.

[24] G. E. Hinton, S. Osindero, and Y. W. Teh. A fast learning algorithm for deep belief nets. *Neural Computation*, 18, 2006.

[25] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

[26] H. Jaeger and H. Haas. Harnessing Nonlinearity: Predicting Chaotic Systems and Saving Energy in Wireless Communication. *Science*, 304(5667):78–80, 2004.

[27] M. Jordan, Z. Ghahramani, T. Jaakkola, and L. Saul. An Introduction to Variational Methods for Graphical Models. *Machine Learning*, 37(2):183–233, 1999.

[28] L. Kaelbling, M. Littman, and A. Moore. Reinforcement Learning: A Survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996.

[29] R. Kalman. A new approach to linear filtering and prediction problems. *Journal of Basic Engineering*, 82(1):35–45, 1960.

[30] F. Kschischang, B. Frey, and H. Loeliger. Factor graphs and the sum-product algorithm. *IEEE Transactions on Information Theory*, 47(2):498–519, 2001.

[31] J. Lafferty, A. McCallum, and F. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proc. 18th International Conf. on Machine Learning*, pages 282–289, 2001.

[32] Y. LeCun and Y. Bengio. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, pages 255–258, 1998.

[33] M. Lewicki. Efficient coding of natural sounds. *Nature Neuroscience*, 5(4):356–363, 2002.

[34] A. Mnih. Learning nonlinear constraints with contrastive backpropagation. Master's thesis, University of Toronto, Oct. 2004.

[35] K. Murphy. *Dynamic Bayesian Network : Representation, Inference and Learning.* PhD thesis, UC Berkeley, 2002.

[36] R. Neal. Learning stochastic feedforward networks. Technical Report CRG-TR-90-7, University of Toronto, 1990. In CRG library.

[37] R. Neal. *Bayesian Learning for Neural Networks.* Springer, 1996.

[38] R. Neal and G. Hinton. A view of the EM algorithm that justifies incremental, sparse, and other variants. *Learning in Graphical Models*, 89:355–368, 1998.

[39] R. M. Neal. Probabilistic inference using Markov chain Monte Carlo methods. Technical report, University of Toronto, 1993.

[40] C. Peterson and J. Anderson. A mean field theory learning algorithm for neural networks. *Complex Systems*, 1(5):995–1019, 1987.

[41] A. Poritz. Hidden Markov models: a guided tour. *International Conference on Acoustics, Speech, and Signal Processing*, pages 7–13, 1988.

[42] L. Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. *In Proceedings of IEEE*, 77(2):257–286, February 1989.

[43] S. Roweis. EM algorithms for PCA and SPCA. *Advances in Neural Information Processing Systems*, 10:626–632, 1998.

[44] D. Rumelhart, G. Hinton, and R. Williams. *Learning internal representations by error propagation.* MIT Press Cambridge, MA, USA, 1986.

[45] N. Schraudolph. Local gain adaptation in stochastic gradient descent. *Ninth International Conference on Artificial Neural Networks*, 2:569–574, 1999.

[46] R. Sutton and A. Barto. *Reinforcement Learning: An Introduction.* Bradford Book, 1998.

[47] B. Taskar, C. Guestrin, and D. Koller. Max-margin Markov networks. *Advances in Neural Information Processing Systems*, 16:25–32, 2004.

[48] M. Turk and A. Pentland. Face recognition using eigenfaces. *Computer Vision and Pattern Recognition*, pages 586–591, 1991.

[49] M. Wainwright, T. Jaakkola, and A. Willsky. A new class of upper bounds on the log partition function. *IEEE Transactions on Information Theory*, 51(7):2313–2335, 2005.

[50] M. Wainwright and M. Jordan. Graphical models, exponential families, and variational inference. *UC Berkeley Dept. of Statistics, Tech. Rep*, 629, 2003.

[51] J. Winn. *Variational Message Passing and its Applications.* PhD thesis, University of Cambridge, october 2003.

[52] J. Yedidia, W. Freeman, and Y. Weiss. Constructing free-energy approximations and generalized belief propagation algorithms. *IEEE Transactions on Information Theory*, 51(7):2282–2312, 2005.