Online unsupervised learning and inference with networks of spiking neurons

by

Russell Tod

A thesis
submitted to the Victoria University of Wellington
in fulfilment of the
requirements for the degree of
Master of Science
in Computer Science.

Victoria University of Wellington
2006

Abstract

As animals interact with their environments, they need to cope with considerable ambiguity and uncertainty inherent in their sensory input. Recent psychophysical experiments suggest that at some level the brain does this by implementing Bayesian inference. This thesis considers algorithms from Machine Learning which perform Bayesian inference on graphical models, and investigates how these algorithms may be implemented with neural components. In this view, neurons are the variable nodes of a graphical model and synapses represent the dependencies between variables. These algorithms are based on the propagation of messages that convey probabilities. Furthermore, learning of the probabilistic dependencies between variables of a belief network is based on observed data and performed in an online and unsupervised manner using a variation of the expectation-maximization (EM) algorithm. The feasibility of these algorithms is demonstrated on a motion detection task.

Acknowledgments

I would like to thank my supervisor, Dr Marcus Frean, for all his guidance and support, and for allowing me the freedom to explore ideas in probabilistic machine learning and neural computation.

Contents

List of Figures

Chapter 1

Introduction

Animals face the challenging problem of converting a two-dimensional retinal image of light intensities to meaningful percepts from which they must make decisions affecting their very survival. Consider the two slightly different views of the world received by each eye from which depth, motion and other cues must be perceived. The fact that this is seemingly achieved so effortlessly belies the true difficultly of the problem. This difficultly arises from uncertainty inherent in perception - sensors are noisy and receive local ambiguous information. For instance, local motion information is ambiguous in determining global motion of an object, a situation known as the "aperture problem" [67]. Furthermore, the processes of perceptual organization must combine information from multiple sensory systems each with different and changing degrees of uncertainty, and which are in different frames of reference. Thus, the two basic information processing goals of the brain are: first, combining uncertain information from several sources, and second, the transformation from one representation to another [62]. Animals which are better at perceiving and acting in an uncertain world have survived. Thus, our sensory systems have evolved to deal with uncertainty, and developed ways to represent, manipulate and reason with uncertain quantities.

Probability theory provides a consistent framework in which to represent and manipulate uncertain quantities, via estimates of the probability of unknown variables. The concepts of probability theory have begun to be applied to problems in biological perception and action [39]. An important observation from this approach is the way in which human observers behave as optimal Bayesian observers. According to Bayesian theory an optimal estimate results from combining prior knowledge with sensory evidence. This process is known as Bayesian inference due to the use of Bayes' rule which states that,

$$P(H|D) = \frac{P(H|D)\ P(H)}{P(D)}$$

where H is some hypothesis and D is the observed data. The terms of Bayes' theorem are frequently given names. The marginal probability $P(H)$ is called the **prior** and $P(D|H)$ is called the **likelihood** concerning hypothesis H. The term $P(D)$ is sometimes called the **evidence**. Thus, Bayes' rule computes $P(H|D)$, the **posterior** probability of the hypothesis given the observed data, by multiplying the likelihood and the prior and normalizing.

There is evidence that brains utilize Bayesian principles for inference. For instance, Bayesian models have been used to explain results in a number of psychophysical experiments [40]. One of the simplest demonstrations of Bayesian inference in the perceptual system is the way contrast influences speed perception. Using Bayes rule, a likelihood and prior are multiplied to compute posterior distribution of perceived speed where speed is the hypothesis H. The likelihood always peaks near the true velocity. At high contrast, the likelihood function is narrow as the image provides reliable information. Therefore, the likelihood dominates the product and the posterior peaks at the same speed as the likelihood. However, at low contrast, the likelihood is wide because the image provides unreliable information. Therefore, when this likelihood is combined with the

prior the prior has a greater influence. As the prior favors slow speeds, the resulting posterior has its peak at a slower velocity. This prediction is verified by experimental evidence of human perception of speed [8]. Many visual illusions can also be explained by Bayesian models of perception [40]. Other experiments on how humans combine visual and haptic cues to assess the shape of objects [17] and combine visual and proprioceptive cues on reaching tasks [42] provide further evidence that the human brain represents uncertainty and performs Bayesian inference.

An interesting and important question arises as to how a brain composed of many distributed processing units, known as neurons, might perform Bayesian inference. One could simply take the view that the combined activity of many neurons indirectly represent and combine probabilities and that only at a behavioural level is Bayes' rule observed, its underlying mechanism being implicit. An alternative view has been suggested that networks of neurons implement Bayesian inference directly [57][15][68]. In this approach neurons explicitly represent probabilities and their firing rates correspond to estimates of hidden variables. As Bayesian inference requires multiplying likelihoods and a prior, a possible implementation with neurons is that this computation be performed in the log domain [57][28][67] where multiplication and division become addition and subtraction, since these operations are easily implemented by excitation and inhibition.

Real world problems might involve thousands of variables, and consequently the task of representing and manipulating probability distributions over all these variables is daunting. Fortunately, for many problems, the vast majority of the possible dependencies between variables need not be modeled, that is, the variables can be assumed to be conditionally independent. Thus, a large joint probability distribution can be broken into a more manageable number of smaller conditional probability distributions. Graphical models are a way to represent the conditional dependencies be-

tween variables graphically and are a useful tool for visualizing these relationships. If the graph is a tree, exact Bayesian inference can then be performed using message-passing algorithms such as belief propagation [55]. One approach for dealing with non-tree structured graphs, that is graphs with loops, is to use a stochastic simulation method known as Gibbs sampling [23] which can also be considered a kind of message passing.

In this direct neural implementation of Bayesian inference view, the brain can be seen loosely as a kind of graphical model where neurons are the nodes of the graph. Furthermore, the correlations between nodes of the graph are represented by the synapses which connect neurons [15][28]. As neurons propagate spikes it seems natural to draw a connection to message-passing algorithms that operate on graphical models such as the belief propagation algorithm [55] and the Gibbs sampling algorithm [51].

The probabilistic view of vision is experiencing a surge of interest from vision theorists. In a generative model of vision it is assumed that an object or event in the external world generates the sensory data observed. Thus, perception can be seen as the process of inverting a generative model. That is, an object or event is inferred from sensory data by assuming that the data was produced by the generative model. Learning, is usually viewed as the process of finding the conditional probabilities of the generative model that maximize the likelihood that the generative model produced the observed data. For problems with hidden variables, learning can be performed with the Expectation-Maximization (EM) algorithm [14]. This learning process is unsupervised, that is, it does not require values for the hidden variables to be provided. However, learning with the EM algorithm is typically performed as a batch process where the observed data is repeatedly presented to the model after every update of the parameters. This makes the algorithm slow, particularly where there is a lot of observed data. Ideally, a learning algorithm would be capable of learning online, that is in a single pass of the data.

In this thesis, a stochastic belief propagation algorithm is proposed to perform approximate Bayesian inference on graphical models. This algorithm is a variation on the belief propagation algorithm where the messages are instead propagated as spikes and is therefore similar to the stochastic feedforward networks of Neal [51]. This algorithm is then applied to the problem of motion detection, using an online version of the EM algorithm to perform unsupervised learning from data as it is observed. Thus this thesis addresses two important issues in machine learning: reasoning with uncertainty, and learning from experience.

The outline of this thesis is as follows. An introduction to graphical models and the belief propagation algorithm is given in chapter 2, and then unsupervised learning in these belief networks with the EM algorithm is described in chapter 3. Spiking neuron models and networks of spiking neurons are introduced chapter 4. Stochastic simulation and a stochastic version of belief propagation for Bayesian inference are introduced in chapter 5 and online learning with the stochastic EM algorithm in chapter 6. These approaches are then demonstrated on a motion detection problem in chapter 7. The conclusion is given in chapter 8.

Chapter 2

Graphical Models

This chapter introduces graphical models in more detail and describes inference on such graphical models using the belief propagation algorithm.

2.1 Introduction to Graphical Models

Graphical models are a combination of graph theory and probability theory [37]. They are a convenient way of representing probabilistic dependencies between variables and reasoning with them. A probability distribution can be represented as a graph in which each node corresponds to a variable. The edges of the graph represent the dependencies between the variables.

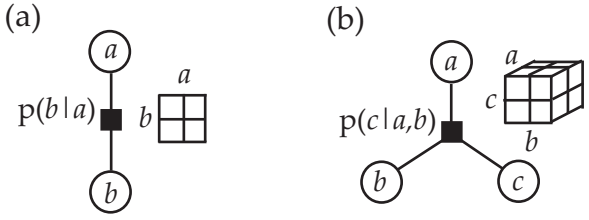

Figure 2.1: A factor connecting two nodes (a) and three nodes (b).

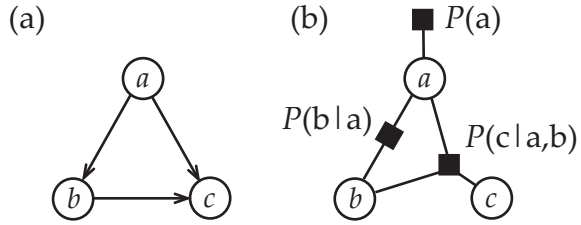

Figure 2.2: Belief network (a) and corresponding factor graph (b). Variable nodes are shown as circles and Factor are denoted by black squares. Arrows on links indicate the conditioned varable. For example, the probability of b given a, $P(b|a)$, is shown by an arrow from a to b.

Consider a joint probability distribution over a set containing discrete variables a, b and c, that is $P(a, b, c)$. For discrete variables the joint probability distribution is a big look-up table of M^N entries that sum to 1, where M is the number of states each variables can take on and N is the number of variables. So, if a, b and c are binary variables then there are 2^3 entries in the joint probability distribution table (see figure 2.1(b)). Clearly, this table becomes immense and intractable to work with for any problem of practical size. Furthermore, as the size of the table grows exponentially so does the amount of data required to learn the joint probabilities.

Using the product rule, $P(a, b) = P(b|a)P(a)$, the joint probability distribution can be broken into factors. This is known as factorization or factorizing the joint. For example, the joint $P(a, b, c)$ can be factorized as follows

$$P(a, b, c) = P(c|a, b)P(b|a)P(a) \tag{2.1}$$

using the product rule.

This factorization can be represented graphically (see figure 2.2(a)) by a directed graph also known as a Belief Network. A factor graph (see figure 2.2(b)) is an alternative rendition of the factorized distribution in which the factors are explicit. A factor graph is a 'bipartite' graph with two kinds of vertices; variables and factors. There are other possible factorizations for

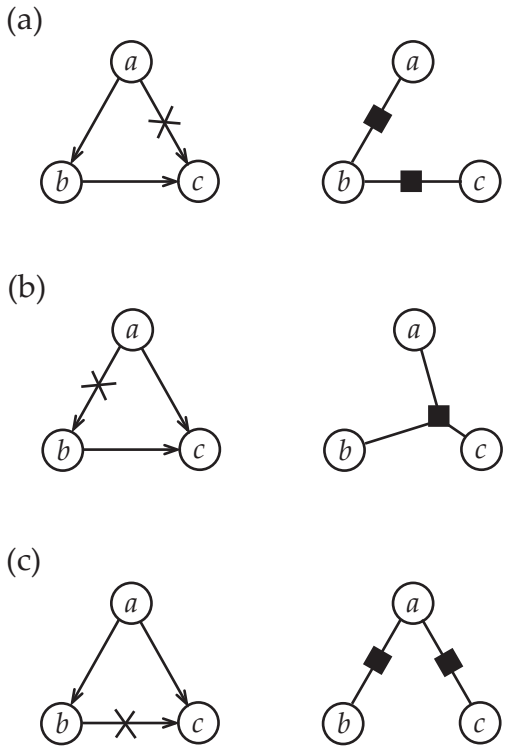

Figure 2.3: Three conditional independence assumptions applied to the same three node graph and the resulting factor graphs. In (a) node c is assumed conditionally independent of a given b, that is $P(c|a,b) = P(c|b)$.

this joint that result from choosing a different order in which the product rule is applied.

The factors, such as $P(b|a)$, are look-up tables. The factorized distribution in equation 2.1 requires exactly the same number of parameters to fill the entries in the conditional probability tables as were required to fill the full joint probability table. However, by making conditional independence assumptions more compact representations are possible. Graphically, making conditional independence assumptions involves removing edges from the graph. This edge removal reduces the size of factors or eliminates factors altogether. Figure 2.3 shows conditional independence

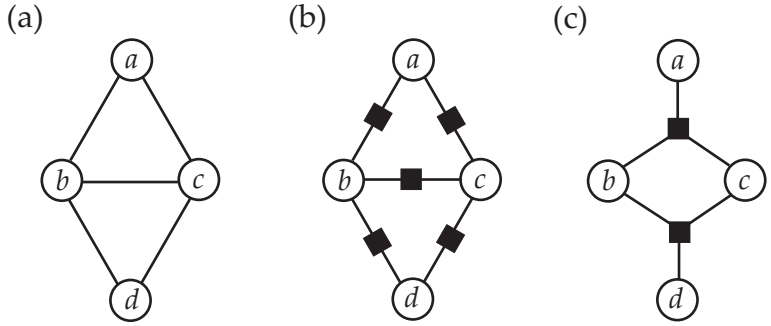

Figure 2.4: This figure shows (a) an undirected graph of four nodes representing the distribution $P(a, b, c, d)$, of which (b) is a pairwise factorization, $P(a, b, c, d) \propto \psi(a, b)\psi(a, c)\psi(b, c)\psi(b, d)\psi(c, d)$, and (c) is another possible factorization, $P(a, b, c, d) \propto \psi(a, b, c)\psi(b, c, d)$.

assumptions and the reduced factor graphs they result in.

Consider a joint probability distribution $P(\mathbf{x})$ over a set of variables $\mathbf{x}$. The general form for a factorization of $P(\mathbf{x})$ that corresponds to a directed graph is

$$P(\mathbf{x}) = \prod_i P(x_i|\mathbf{x}_{\pi_i})$$

where $\mathbf{x}_{\pi_i}$ are the parents of the x_i variable node.

Another variety of graphical models are undirected graphs, which are also known as Markov random fields or Markov networks. An undirected graph is factorized into cliques, where a clique is a fully connected subgraph. A clique is maximal if it is not contained within another clique. For example, the pairwise cliques of figure 2.4 (b) are $\{a, b\}$, $\{a, c\}$, $\{b, c\}$, $\{b, d\}$ and $\{c, d\}$, and the maximal cliques of figure 2.4 (c) are $\{a, b, c\}$ and $\{b, c, d\}$. From these cliques an undirected graph is factorized into non-negative potential functions ψ. For example, a pairwise Markov network factorization of probability distribution $P(a, b, c, d)$ is

$$P(a, b, c, d) = \frac{1}{Z}\psi(a, b)\psi(a, c)\psi(b, c)\psi(b, d)\psi(c, d)$$

where the normalization constant Z is required to guarantee $P(a, b, c, d)$ is

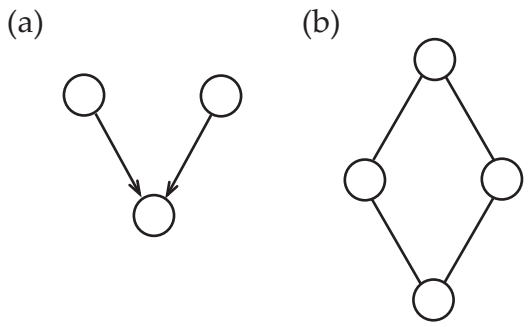

Figure 2.5: This figure shows (a) a directed graph for which there is no undirected graph which can represent these and only these independencies. Figure (b) shows an undirected for which there is no directed graph.

a valid probability distribution. For an undirected graph the general form is

$$P(\mathbf{x}) = \frac{1}{Z} \prod_{C \in \mathbf{C}} \psi(\mathbf{x}_C)$$

where $\mathbf{x}_C$ are the variables of clique C, and $\mathbf{C}$ is the set of all the cliques of the graph. The normalization of undirected graphs is global. However, for directed graphs normalization is local. Therefore, the factors of directed graphs have a direct probabilistic interpretation but factors of undirected graphs do not.

There are probabilities distributions that can only be represented by a directed or an undirected graph that cannot be represented by the other. For example, the directed graph of figure 2.5 (a) has no equivalent undirected graph that can capture only those conditional independencies specified by the directed graph. To convert the directed graph of figure 2.5 (a) into an undirected graph we must add a connection between the parents of c, nodes a and b; a process is known as 'moralization' of a graph. This moralization introduces conditional dependencies not present in the directed graph. Therefore, the directed graph has conditional independencies not exploited by the moralized undirected graph. In the case of the undirected

graph of figure 2.5 (b) it is impossible to add arrows to produce a corresponding directed graph such that the same conditional independencies are represented that are present in the undirected graph. Adding arrows to figure 2.5 (b) will result in the V-structure of figure 2.5 (a) which, as already mentioned, does not have the same independence interpretation as intended by the undirected graph. However, there are distributions that can be represented by both directed or undirected graphs. These are any distribution that does not have the V-structure of figure 2.5 (a).

2.2 Probabilistic inference

Typically, one seeks to find the marginal probability of a variable, or the conditional probability of a variable given some observations of other variables. Inferring these values involves summing over all the unknown variables using the sum rule, $P(b) = \sum_a P(a, b)$. This process is known as marginalization. From the marginals other values of interest can computed such as conditional probabilities, for instance, $P(a|b) = \frac{P(a,b)}{\sum_a P(a,b)}$. The brute force method of inferring a marginal probability is to marginalize the joint probability distribution over all the unknown variables. However, this sum becomes intractable as the computation grows exponentially in the number of unknowns.

By way of example consider the factor graph in figure 2.6. Suppose that we wish to calculate the marginal probability $P(x_4)$. The marginal $P(x_4)$ is computed by summing the joint over all variables except x_4, that is

$$P(x_4) = \sum_{x_3} \sum_{x_2} \sum_{x_1} P(x_1, x_2, x_3, x_4)$$

The factorization of the joint depicted in figure 2.6 is:

$$P(x_4) = \sum_{x_3} \sum_{x_2} \sum_{x_1} P(x_4|x_2)\ P(x_3|x_2)\ P(x_2|x_1)\ P(x_1)$$

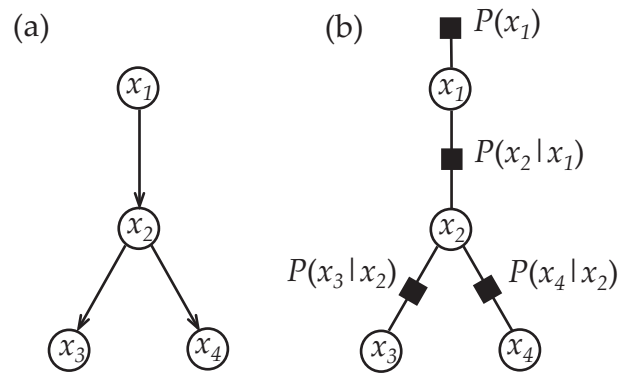

Figure 2.6: An example factor graph of the joint probability distribution $P(x_1, x_2, x_3, x_4)$ factorized such that x_1, x_3 and x_4 are conditionally independent given x_2.

Notice that the variable x_1 appears only in the two right-most factors and it is therefore possible to move the sum over x_1 to be in front of only these factors. Similarly the sum over x_3 can be moved further into the product, giving

$$P(x_4) = \sum_{x_2} P(x_4|x_2) \sum_{x_3} P(x_3|x_2) \sum_{x_1} P(x_2|x_1)\ P(x_1)$$

Here, we in introduce an intermediate term $m_{12}(x_2)$ to represent the vector of probabilities that result from summing $P(x_2|x_1)P(x_1)$ over x_1, for instance

$$P(x_4) = \sum_{x_2} P(x_4|x_2) \sum_{x_3} P(x_3|x_2)\ m_{12}(x_2)$$

Similarly, we substitute in an intermediate term $m_{32}(x_2)$ for the sum over x_3, that is

$$P(x_4) = \sum_{x_2} P(x_4|x_2)\ m_{32}(x_2)\ m_{12}(x_2)$$

And finally, by summing over x_2 we compute the marginal probability

$$P(x_4) = m_{24}(x_4)$$

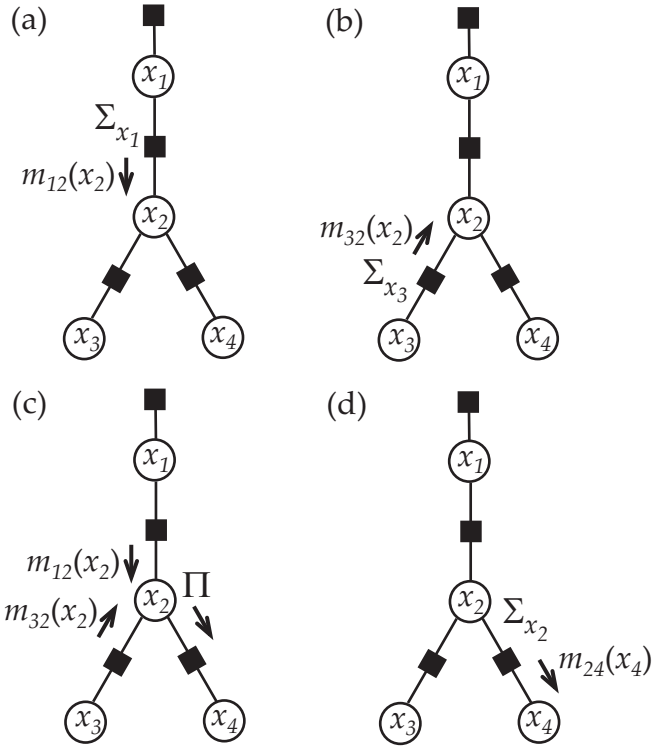

Figure 2.7: This figure shows the intermediate terms of the marginalization calculation which can be viewed as messages that are passed between nodes. In (a), the intermediate term is a message $m_{12}(x_2)$ that is computed by summing $P(x_2|x_1)P(x_1)$ over the values x_1. The message is a vector of values over x_2 and is passed to variable node x_2. Similarly in (b), by summing $P(x_3|x_2)$ over the values of x_1 a term $m_{32}(x_2)$ is computed and passed to variable node x_2. In (c), the intermediate terms from variables x_1 and x_3 are combined by element-wise multiplication of the message vector elements. And finally in (d), this combined vector and the factor $P(x_4|x_2)$ are multiplied together and then the sum over the values of x_2 is performed to get the message vector $m_{24}(x_4)$ over the values of x_4. This message is the desired marginal probability $P(x_4)$.

Figure 2.7 shows this computation graphically. The intermediate terms m can be viewed as messages to be passed between variables. It is these messages that form the basis of an algorithm known as belief propagation.

2.3 Belief propagation

Belief propagation (BP) [55] is an algorithm for performing inference in graphical models by passing messages; and is essentially a generalization of the preceding example to any graph that is tree structured. The goal of BP is to compute the marginal probabilities of a distribution over a set of variables. In the example used in section 2.2 it was shown that by exploiting the factorization of the joint it is possible to convert an apparently intractable sum over the joint into a product of partial sums. The messages that form the basis of the BP algorithm are the results of these partial sums. Thus, the global computation required to compute the marginal probabilities is transformed into local computation and the sending of messages.

The BP algorithm can be described in terms of operations on a factor graph. In the BP algorithm there are two kinds of messages; those from variable-to-factor and those from factor-to-variable. The message $m_{ai}(x_i)$ sent from factor a to variable i is a vector over the possible states of x_i. Similarly the message $n_{ia}(x_i)$ sent from variable i to factor a is a vector over the possible states of x_i. The set of neighbours of variable node i except factor a is denoted $N(i)\backslash a$. Similarly, the set of neighbours of factor node a except for variable i is denoted $N(a)\backslash i$. The set of variables $\mathbf{x}$ of a factor node a are denoted $\mathbf{x}_a$, and the potential function taking these variables as arguments is $\psi_a(\mathbf{x}_a)$. For a directed graph the $\psi_a(\mathbf{x}_a)$ is a conditional probability, which can be obtained from $\psi_a(\mathbf{x}_a)$ by normalizing over the appropriate variable of $\mathbf{x}_a$. The sum over variables $\mathbf{x}_a$ except x_i is denoted $\sum_{\mathbf{x}_a \backslash x_i}$.

The BP algorithm can be summarized by two rules to update these two

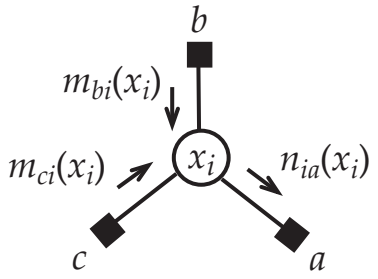

Figure 2.8: This figure shows the message update rule of equation 2.2 for a message $n_{ia}(x_i)$ sent from variable node i to factor node a. The message $n_{ia}(x_i)$ is a vector over the states of variable x_i. This message is computed at a variable node i by the element-wise multiplication of incoming messages $m_{bi}(x_i)$ and $m_{ci}(x_i)$.

message sets, m and n [19]. These rules are as follows.

1. The variable-to-factor messages are computed by:

$$n_{ia}(x_i) = \prod_{b \in N(i) \setminus a} m_{bi}(x_i) \tag{2.2}$$

2. The factor-to-variable messages are computed by:

$$m_{ai}(x_i) = \sum_{\mathbf{x}_a \setminus x_i} \psi_a(\mathbf{x}_a) \prod_{j \in N(a) \setminus i} n_{ja}(x_j) \tag{2.3}$$

The computation to be performed at a variable node is given by equation 2.2, which prescribes the message that a variable node i should send to its neighbouring factors a. Each neighbouring factor of a node is sent a different message by the variable node, since in each case a different term is left out of the product. The computation to be performed at a factor node is given by equation 2.3. Equations 2.2 and 2.3 are shown graphically in figures 2.8 and 2.9.

Some factor nodes in the graph are connected to only one variable, and so, the message product of equation 2.3 is empty and is taken to be one. These factor nodes perpetually send a message that is the potential $\psi(x_i)$

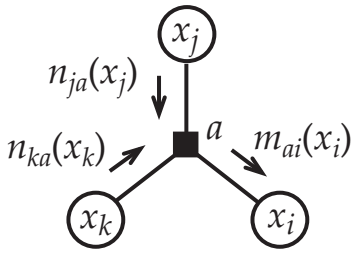

Figure 2.9: This figure shows the message update rule of equation 2.3 for a message $m_{ai}(x_i)$ sent from factor node a to variable node i. The message $m_{ai}(x_i)$ computed at factor node a is a vector over the states of variable x_i. This message $m_{ai}(x_i)$ is computed by summing over the values of variables x_j and x_k for the product of potential $\psi_a(x_i, x_j, x_k)$ and incoming messages $n_{ja}(x_j)$ and $n_{ka}(x_k)$.

for a variable x_i. Other variable or factor nodes are able to propagate their message once they have received all their incoming messages. In this way messages propagate across the graph.

The belief $b(x_i)$ at variable node i is the marginal probability $P(x_i)$, for a tree structured graph. The belief is the product of all incoming messages to a node

$$b_i(x_i) \propto \prod_{a \in N(i)} m_{ai}(x_i) \tag{2.4}$$

where the proportional symbol $\propto$ indicates that the beliefs must be normalized. As noted by Pearl [55] normalizing the messages does not effect the beliefs computed by the belief propagation algorithm.

The variable-to-factor message *to* factor a (n_{ia}) is computed by multiplying together all incoming messages except the message *from* factor a (equation 2.2). Equivalently, n_{ia} could be computed by multiplying *all* the incoming messages and then dividing out the message from factor a, that is

$$n_{ia}(x_i) = b_i^*(x_i)/m_{ai}(x_i) \tag{2.5}$$

where b_i^* indicates an unnormalized belief, $b_i^*(x_i) = \prod_a m_{ai}(x_i)$.

For non-tree structured factor graphs, that is those with loops, the beliefs computed by belief propagation are not the exact marginal probabilities. However, the BP algorithm performed on graphs with loops, sometimes referred to as loopy belief propagation, often gives a reasonable approximation and excellent experimental results have been reported on networks with loops [50].

The preceding message-passing algorithm is known by various names among which are the sum-product algorithm in coding theory [22], where it has been shown equivalent to turbocodes [47], the forward-backward algorithm in Hidden Markov models [56], and belief-propagation [55]. Message-passing algorithms are a practical and powerful way to solve many problems in a variety of disciplines, including statistical physics, signal processing, artificial intelligence, and digital communications [19][37].

2.3.1 Special case: binary variable nodes

For binary nodes x is either true (denoted X) or not true (denoted $\neg X$). Suppose that $b_i^*(X_i)$ is the unnormalized belief computed from the product of incoming messages, $b_i^*(X_i) = \prod_{a \in N(i)} m_{ai}(X_i)$. The local normalization of beliefs is therefore,

$$b_i = \frac{b_i^*(X_i)}{b_i^*(\neg X_i) + b_i^*(X_i)} \tag{2.6}$$

Dividing the numerator and denominator by $b_i^*(\neg X_i)$ gives,

$$b_i = \frac{\frac{b_i^*(X_i)}{b_i^*(\neg X_i)}}{1 + \frac{b_i^*(X_i)}{b_i^*(\neg X_i)}} \tag{2.7}$$

Thus, by defining function f to be $f(z) = z/(1+z)$ we can re-write this equation as

$$b_i = f\left(\frac{b_i^*(X_i)}{b_i^*(\neg X_i)}\right) \tag{2.8}$$

Furthermore, by equation 2.4 we have

$$b_i = f\left(\prod_{a \in N(i)} \frac{m_{ai}(X_i)}{m_{ai}(\neg X_i)}\right) \tag{2.9}$$

Similarly we can normalize the variable-to-factor messages using the function $f(z) = z/(1+z)$ applied to the ratio of variable-to-factor message values for the two values of x_i, X_i and $\neg X_i$. This gives

$$n_{ia} = f\left(\frac{b_i^*(X_i)/m_{ai}(X_i)}{b_i^*(\neg X_i)/m_{ai}(\neg X_i)}\right) \tag{2.10}$$

With the division re-arranged this becomes

$$n_{ia} = f\left(\frac{b_i^*(X_i)m_{ai}(\neg X_i)}{b_i^*(\neg X_i)m_{ai}(X_i)}\right) \tag{2.11}$$

These forms of the belief equation 2.9 and of the variable-to-factor message equation 2.11 will be useful later in section 5.1.1.

Chapter 3

Learning in Graphical Models

Unsupervised learning of the conditional probabilities tables (CPTs) of belief networks from data is essential, as it is typically prohibitive for an expert (or in the biological world, a genetic encoding) to specify all parameters of the network. For a belief network, there are two kinds of learning we might wish to perform. One is learning of the structure of the network and the other is learning of the network parameters (CPTs) Here we assume that the structure of the belief network is known and has either been provided by an expert or it has been learnt by a method such as model-selection EM [20]. There are two cases for learning the parameters of a network, where there is either have complete data or incomplete data.

3.1 Learning with complete data

Assuming we have complete data and a known network structure we can learn the parameters θ of the network by Maximum Likelihood (ML). Let $\mathbf{d}$ be the training data set $\mathbf{d} = \{\mathbf{x}^1, ..., \mathbf{x}^T\}$, where $\mathbf{x}^t = \{x_1^t, ..., x_d^t\}$. The data set is complete if it assigns a value to every variable of the network. In the case of known structure and complete data, learning is inference of the parameters. The goal of learning then is to find the ML estimate

of the parameters for each conditional probability table of every factor in the network. The probability of all the observed data given the current parameters is referred to as the likelihood

$$P(\mathbf{d}|\theta) = \prod_{t=1}^{T} P(\mathbf{x}^t|\theta) \tag{3.1}$$

Equivalently, we can estimate the parameters which maximize the log-likelihood $\mathcal{L}(\theta)$. The log-likelihood decomposes according to the factorization of the belief network, that is

$$\mathcal{L}(\theta) = \log\ P(\mathbf{d}|\theta) = \sum_{t=1}^{T} \log\ P(\mathbf{x}^t|\theta) = \sum_{t=1}^{T} \sum_{i=1}^{d} \log\ P(x_i^t|\mathbf{x}_{\pi_i}^t, \theta_i) \tag{3.2}$$

where $\mathbf{x}_{\pi_i}^t$ denotes the parents of i. Therefore, because of the factorization we can maximize the log-likelihood of each variable node independently.

The goal is to learn the entries of the conditional probability tables (CPT). The frequency that a CPT entry occurs in the data, normalized accordingly, corresponds to the Maximum Likelihood estimate of that CPT entry. For example, consider two binary nodes x_1 and x_2 and say we want to find the CPT entry $P(x_2{=}1|x_1{=}0)$. The number of occurrences in the data set $\mathbf{d}$ where x_2 is in state 1 and x_1 is in state 0 is denoted $c(x_2{=}1, x_1{=}0)$

$$c(x_2{=}1, x_1{=}0) = \sum_t I[x_2^t{=}1] I[x_1^t{=}0] \tag{3.3}$$

where $I[x_i{=}j]$ is an indicator function which is 1 if x_i is in state j and zero otherwise. These counts are known as the sufficient statistics. The number of times that $P(x_2{=}1|x_1{=}0)$ occurs in the log likelihood is equal (with normalization) to the count $c(x_2{=}1|x_1{=}0)$ where the normalization constraint is $P(x_2{=}0|x_1{=}0) = 1 - P(x_2{=}1|x_1{=}0)$. Therefore, the total contribution of $P(x_2{=}1|x_1{=}0)$ to the log likelihood is

$$c(x_2{=}1, x_1{=}0)\ \log\ P(x_2{=}1|x_1{=}0)\ +\ c(x_2{=}0, x_1{=}0)\ \log\ (1 - P(x_2{=}1|x_1{=}0)) \tag{3.4}$$

Differentiating equation 3.4 with respect to $P(x_2{=}1|x_1{=}0)$ gives,

$$P(x_2{=}1|x_1{=}0) = \frac{c(x_2{=}1, x_1{=}0)}{c(x_2{=}1, x_1{=}0) + c(x_2{=}0, x_1{=}0)} \tag{3.5}$$

Equation 3.5 is the maximum likelihood estimate of CPT entry for $P(x_2{=}1|x_1{=}0)$. By the same method the other CPT entries can be calculated.

The Bayesian approach to learning combines a prior over the parameters with the likelihood of the data. For a maximum a posteriori (MAP) estimate "pseudo counts" are added to the empirical counts. This avoids the problem that arises when an event does not occur in the training set and is therefore assigned a probability of zero. Pseudo-counts correspond to a Dirichlet prior over the parameters. For example, using a uniform Dirichlet prior, the MAP estimate for CPT entry $P(x_2{=}1|x_1{=}0)$ is

$$P(x_2{=}1|x_1{=}0) = \frac{c(x_2{=}1, x_1{=}0) + 1}{c(x_2{=}1, x_1{=}0) + c(x_2{=}0, x_1{=}0) + 2} \tag{3.6}$$

Therefore, ML and MAP learning in a complete data setting for the directed graph simply involves counting occurrences in the data set and normalizing to get the conditional probability.

3.2 Learning with incomplete data

In the case where observed data does not include the values of some variables in the graphical model the data is referred to as incomplete. We assume that for the whole data set it is the same variables for which data is not observed. These variables are often referred to as hidden variables. Consider a graph of two sets of variables; observed (visible) variables $\mathbf{v}$ and unobserved (hidden) variables $\mathbf{h}$. That is, the data set $\mathbf{d}$ is $\{\mathbf{v}^1, ..., \mathbf{v}^T\}$. In this case, the log-likelihood cannot be decomposed as it was in equation 3.2 [53]. Rather, the likelihood involves a sum over the states of the hidden

variables, that is

$$\mathcal{L}(\theta) = \log\ P(\mathbf{d}|\theta) = \sum_{t=1}^{T} \log\ P(\mathbf{v}^t|\theta) = \sum_{t=1}^{T} \log \sum_{\mathbf{h}^t} P(\mathbf{v}^t, \mathbf{h}^t|\theta) \tag{3.7}$$

The log of the sum potentially couples all the parameters of the graph together, making learning of the parameters difficult. Furthermore, the sum is intractable for models with many hidden variables.

Neal and Hinton [53] simplified this problem with the insight that any distribution over the hidden variables defines a lower bound on the likelihood $\mathcal{L}$ and therefore learning involves maximizing this lower bound on the likelihood rather than the likelihood itself. In this approach an approximating distribution Q is introduced to equation 3.7 as follows

$$\mathcal{L}(\theta) = \sum_{t=1}^{T} \log \sum_{\mathbf{h}^t} Q^t(\mathbf{h}) \frac{P(\mathbf{v}^t, \mathbf{h}^t|\theta)}{Q^t(\mathbf{h})} \tag{3.8}$$

Applying Jensen's inequality [13], which follows from the $\log$ function being concave, gives the lower bound

$$\mathcal{L}(\theta) \geq \sum_{t=1}^{T} \sum_{\mathbf{h}^t} Q^t(\mathbf{h}) \log\ \frac{P(\mathbf{v}^t, \mathbf{h}^t|\theta)}{Q^t(\mathbf{h})} \tag{3.9}$$

This decouples the parameters and there is no longer a need to take the log of a sum over h^t. We can now maximize the lower bound to the log-likelihood of each variable node independently.

Equation 3.9 can be re-written as

$$\mathcal{L}(\theta) \geq \underbrace{\sum_{t=1}^{T} \sum_{\mathbf{h}^t} Q^t(\mathbf{h}) \log\ P(\mathbf{v}^t, \mathbf{h}^t|\theta)}_{\text{Energy}} - \underbrace{\sum_{t=1}^{T} \sum_{\mathbf{h}^t} Q^t(\mathbf{h}) \log\ Q^t(\mathbf{h})}_{\text{Entropy}} = \mathcal{F}(Q, \theta) \tag{3.10}$$

where $\mathcal{F}(Q, \theta)$ is the quantity known in statistical physics as the free energy. The free energy is the expected energy under distribution Q minus the entropy of Q.

The Expectation-Maximization (EM) algorithm [5][14] iterates two steps which are referred to as the Expectation (E) step and the Maximization (M) step. The E step maximizes the free energy $\mathcal{F}$ with respect to approximating distribution Q with parameters θ fixed,

$$Q^{t+1} = \arg\max_{Q} \mathcal{F}(Q, \theta^t) \tag{3.11}$$

The M step maximizes the free energy $\mathcal{F}$ with respect to parameters θ with distribution Q fixed,

$$\theta^{t+1} = \arg\max_{\theta} \mathcal{F}(Q^{t+1}, \theta) \tag{3.12}$$

When these steps are iterated the EM algorithm is guaranteed to increase a lower bound on the log likelihood.

The EM algorithm is a very general approach and can be applied to learning the CPTs of belief networks [7][27]. For a belief network, the E involves inferring the approximating distribution Q over the hidden variables given the current parameters. This inference can be performed by the belief propagation algorithm which is exact for tree structured graphs. Exact inference results in the optimal Q distribution of $Q(\mathbf{h}) = P(\mathbf{h}|\mathbf{v}, \theta)$ and the lower bound becomes an equality $\mathcal{F} = \mathcal{L}$. However, for a belief network with loops inference with the BP algorithm is not exact. In this case an approximation of $Q(\mathbf{h}) \approx Q(\mathbf{x}) = \prod_i Q(x_i|\mathbf{x}_{\pi_i}, \theta)$ is used, which takes advantage of the belief network factorization of the distribution.

The maximization (M) step of the EM algorithm finds a maximum likelihood estimate for θ. For a belief network, the ML estimate is computed from the expected sufficient statistics obtained in the E step. That is, the hidden variables inferred by the E step are treated as if they were observed and the ML estimate can then be performed as in the complete data case (section 3.1). The M step maximizes the energy term of equation 3.10. As the entropy does not depend on θ it does not contribute to the ML esti-

mate. Thus, the parameter update computed by the M step is

$$\theta^{t+1} = \arg\max_{\theta} \sum_{t} Q^t(\mathbf{h}|\mathbf{v}) \log\ P(\mathbf{h}^t, \mathbf{v}^t|\theta) \tag{3.13}$$

Applying the belief network factorization to the joint distribution gives

$$\theta^{t+1} = \arg\max_{\theta} \sum_{t} \sum_{i} Q^t(\mathbf{h}|\mathbf{v}) \log\ P(x_i^t|\mathbf{x}_{\pi_i}^t, \theta) \tag{3.14}$$

The following example, adapted from [4], should hopefully clarify this approach to learning of the conditional probability table values. Consider the factor graph in figure 2.6 which has the following factorization

$$P(x_1, x_2, x_3, x_4) = P(x_4|x_2)\ P(x_3|x_2)\ P(x_2|x_1)\ P(x_1) \tag{3.15}$$

Suppose that variables x_2 and x_4 are hidden, $\mathbf{h} = \{x_2, x_4\}$, and the x_1 and x_3 are observed, $\mathbf{v} = \{x_1, x_3\}$. Then, the contribution to the energy of equation 3.10 is

$$\sum_{t} Q^t(x_2, x_4|x_1, x_3) \log \left[\ P(x_4|x_2)\ P(x_3^t|x_2)\ P(x_2|x_1^t)\ P(x_1^t)\ \right] \tag{3.16}$$

Equation 3.16 may be re-written as

$$\sum_{t} Q^t(x_2, x_4|x_1, x_3) \log\ P(x_4|x_2) \tag{3.17}$$

$$+ \sum_{t} Q^t(x_2|x_1, x_3) \log\ P(x_3^t|x_2) \tag{3.18}$$

$$+ \sum_{t} Q^t(x_2|x_1, x_3) \log\ P(x_2|x_1^t) \tag{3.19}$$

$$+ \sum_{t} \log\ P(x_1^t) \tag{3.20}$$

where each of the CPTs, such as $P(x_3|x_2)$, can be maximized independently. Learning the CPT for $P(x_1^t)$ is straight-forward as variable x_1 is observed. Consider learning the CPT entry for $P(x_2^t{=}i|x_1{=}j)$. This table

entry occurs in the energy whenever variable x_2 is in state i. Hence the contribution to the energy of term $P(x_2^t{=}i|x_1{=}j)$ is

$$\sum_t I[x_1^t{=}j]\ Q^t(x_2{=}i|x_1,x_3)\ \log\ P(x_2{=}i|x_1{=}j){+}\lambda\left\{1-\sum_k P(x_2{=}k|x_1{=}j)\right\} \tag{3.21}$$

where the last term is a Lagrange term required to ensure normalization of the table. The value for $Q^t(x_2{=}i|x_1,x_3)$ is inferred by the E step and approximates $P(x_2{=}i|x_1,x_3)$. Differentiating equation 3.21 with respect to $P(x_2{=}i|x_1{=}j)$ gives

$$P(x_2{=}i|x_1{=}j)=\frac{\sum_t I[x_1^t{=}j]\ Q^t(x_2{=}i|x_1,x_3)}{\sum_t\sum_k I[x_1^t{=}j]\ Q^t(x_2{=}k|x_1,x_3)} \tag{3.22}$$

Similar updates can be derived for the other CPTs of equation 3.15. The M step update for CPT entry $P(x_3^t{=}i|x_2{=}j)$ is

$$P(x_3{=}i|x_2{=}j)=\frac{\sum_t I[x_3^t{=}i]\ Q^t(x_2{=}j|x_1,x_3)}{\sum_t\sum_k I[x_3^t{=}k]\ Q^t(x_2{=}j|x_1,x_3)} \tag{3.23}$$

The M step update for CPT entry $P(x_4^t{=}i|x_2{=}j)$, where both variables are hidden, is

$$P(x_4{=}i|x_2{=}j)=\frac{\sum_t Q^t(x_4{=}i|x_1,x_3)\ Q^t(x_2{=}j|x_1,x_3)}{\sum_t\sum_k Q^t(x_4{=}k|x_1,x_3)\ Q^t(x_2{=}j|x_1,x_3)} \tag{3.24}$$

Equation 3.22 is the hidden variable equivalent of the complete data case of equation 3.5. Comparing these equations we see that the deterministic count $I[x_2{=}i]$ is replaced by the inferred hidden variable value that x_2 is in state i, that is, $Q(x_2{=}i|\mathbf{v})$. Therefore, in the EM algorithm we replace the deterministic functions I with their inferred hidden variable equivalents Q.

The EM algorithm handles hidden data in an intuitive way. The E-step "fills-in" the hidden variables, and the M-step computes the parameters by a ML estimate as if the hidden variables were observed. This is typically done as a batch process, where in the E-step BP is run over the entire

dataset. Next, the parameters are updated based on the inferred hidden values obtained from the E-step, and the E-step is re-run over the entire dataset again. In a later chapter, variations on the EM algorithm are introduced that operate in an online rather than batch manner, on stochastic neural networks that are introduced in the next chapter.

Chapter 4

Spiking Neural Networks

This chapter introduces spiking neuron models of real neurons. Then, these complex models are reduced to simpler, more abstract mathematical models that perform computational operations of which real neurons may be capable of implementing.

4.1 Spiking Neuron Models

Spiking neuron models are an attempt to capture the computational capabilities of real neurons which communicate via electrical pulses known as spikes. The name 'spiking neuron' refers to any neuron model that emits spikes as its primary means of communication. The emission of a spike is sometimes also referred to as the neuron 'firing'. There are three functionally distinct parts to a typical neuron: the dendrites, the soma and axons. The dendrite is the input of a neuron, that collects signals from other neurons and transmits them to the soma. The soma performs a thresholding operation, generating an output signal if the total input exceeds a certain threshold. The axon is the output of a neuron which carries a signal to other neurons. A neuron is essentially an electrical device, such that there is a potential difference between the interior of a neuron and its surround-

ings, called the membrane potential u. When the membrane potential u reaches the threshold u^θ a spike is emitted.

A synapse is the junction between two neurons, where the axon from of one neuron connects to the dendrite of another neuron. Neurons send signals to one another across the synapse. The sending neuron may also be referred to as the pre-synaptic neuron and the receiving neuron may also be referred to as the post-synaptic neuron. Neuronal signals sent down the axon of a neuron are short electrical pulses, commonly referred to as action potentials or spikes. These spikes have a typical duration of about 1 millisecond and have roughly the same amplitude as each other. Thus it is not the form of the spike which conveys information, but rather, it is the frequency and timing of spikes. When a spike arrives at a synapse, neurotransmitter is released from the pre-synaptic side of the synapse. These transmitter molecules are detected on the post-synaptic side of the synapse by specialized receptors which open ion channels leading to a change in membrane potential called a post-synaptic potential. The amplitude of a post-synaptic potential is referred to as the efficacy or weight of the synapse and is denoted by w.

There have been many models of neuron that model neurons at differing levels of detail. A well known detailed neuron model is that of Hodgkin and Huxley [35], which models the opening and closing of ion channels that affect the neuron's membrane potential.

4.1.1 Spike Response Model

The Spike Response Model is a spiking neuron model that captures the dynamics of the membrane potential of a neuron [24][25], but are less bio-physically accurate than the Hodgkin and Huxley model. The SRM_0 model describes the momentary value of the membrane potential u_i^t of neuron i at time t. A neuron's membrane potential is affected by the

spikes it receives, such that each spike evokes a post-synaptic potential. Each post-synaptic potential has an amplitude w_{ji} and a temporal evolution described by the kernel ϵ. Consider a pre-synaptic neuron j which sends spikes at times $\{t_j^{(1)}, t_j^{(2)}, ...\}$ to post-synaptic neuron i. It is assumed that spikes received at post-synaptic neuron i have an effect on the membrane potential u_i that is the sum of the individual post-synaptic potentials. Thus, the input to neuron i is the sum of all the current post-synaptic potentials $w_{ij} \sum_f \epsilon(t - t_j^{(f)})$, sent by neuron j where f indexes the spikes received by neuron i. In the SRM_0 model, the membrane potential of a neuron is also affected by its own most recent firing event, at time $\hat{t}$. This effect is described by the kernel $\eta(t - \hat{t})$, which incorporates what is known as the refractory period - the short period after firing in which the membrane potential is suppressed. Thus, with all these components put together, the SRM_0 equation for the membrane potential is,

$$u_i^t = \eta(t - \hat{t}) + \sum_j w_{ij} \sum_f \epsilon(t - t_j^{(f)}) \tag{4.1}$$

The membrane potential is the sum of all spikes f from all neighbouring pre-synaptic neurons j of neuron i. The state of a neuron i is denoted s_i^t, and is either firing $s_i^t{=}1$ or not $s_i^t{=}0$, at each time step t. A spike is generated when the threshold u^θ is exceeded. The thresholding condition to generate a spike is,

$$s_i^t = \Theta(u_i^t - \theta_i) \tag{4.2}$$

where Θ is the Heaviside step function with $\Theta(u) = 1$ if $u \geq 0$ and $\Theta(u) = 0$ otherwise.

The full SRM model involves an additional term for an external current input and has an ϵ kernel that also depends on the time the neuron last fired $\hat{t}$, $\epsilon(t - \hat{t}, t - t_j^{(f)})$ [25]. Another common spiking neuron model, the leaky integrate-and-fire model, has been shown to be a special case of SRM where the kernel ϵ is an exponential decay function [25, p109].

4.1.2 Stochastic neuron model

It has been observed that the spike trains of real neurons are highly irregular and that they approximate a random (Poisson) process [64]. There is much debate in neuroscience on the meaning of this irregularity [63]. On the one hand, this irregularity may be due to noise in which case it is the spike rate which conveys information. The alternative view is that the irregularity conveys information and the precise timing of spikes is important [60]. Despite this controversy, the view taken here is that information is conveyed primarily by a neuron's firing rate.

Stochastic neuron model are another form of spiking neurons where, instead of generating spikes by a deterministic threshold process, spikes are generated stochastically. In the stochastic neuron model, the output is a rate-modulated Poisson process with time varying rate parameter $r^{(t)}$. For a Poisson process, the instantaneous rate $r^{(t)}$ is the probability of firing at time t. In a discrete time model time is discretized into intervals of length Δt, where $\Delta t = 1/r_{\text{max}}$ and r_{max} is the maximum firing rate. In each time step a neuron i is either firing $s_i^t = 1$ or not firing $s_i^t = 0$. In discrete time, the probability that a stochastic neuron is firing, $s_i^t = 1$, at time interval t is a function of the total synaptic input to the neuron, u_i^t, that is

$$P(s_i^t = 1 | u_i^t) = g(u_i^t)\Delta t \tag{4.3}$$

where g is the gain function. Alternatively, if we take the kernel ϵ of the SRM_0 model equation 4.1 to be a Dirac δ-pulse, such that, when receiving a spike the value of ϵ is $1/\Delta t$ and zero otherwise, and ignore refractoriness η then we get

$$u_i^t = \sum_j w_{ij} s_j^t \tag{4.4}$$

This model still captures the behaviour that incoming spikes evoke a post-synaptic potential, but the detailed time course of that response has been

reduced to a Dirac δ-pulse. Equations 4.3 and 4.4 define a stochastic neuron model that performs a linear weighted sum of the spikes it has received from its neighbouring neurons.

The advantage of using stochastic neurons is that methods exist for performing probabilistic inference and Maximum Likelihood learning on networks of such neurons.

4.1.3 Networks of stochastic neurons

One of the earliest models of networks composed of stochastic neurons is the 'Boltzmann machine'. The Boltzmann machine [33][34] is a fully connected undirected graph of stochastic binary units with symmetric connections. Stochastic simulation is a method of sampling from conditional distributions of a graphical model. Stochastic simulation is also known as Gibbs sampling [23]. Gibbs sampling involves repeatedly selecting a new value for each variable node from its distribution conditional on the rest of the network. Typically, inference is performed by clamping the observation nodes to their observed values. Then, once the simulation has been run for a sufficient length of time to reach equilibrium the generated samples approximate the posterior distribution. However, it is difficult to know how long the simulation will require to reach equilibrium. Another drawback of the Boltzmann machine is that learning is generally slow and requires an unclamped phase (observation nodes are free) which is not biologically plausible.

Sigmoid belief networks [51], also known as stochastic feedforward networks, are an attempt to overcome some of the limitations of the Boltzmann machine. A sigmoid belief network is composed of stochastic binary neurons connected in a directed graph with pairwise factors. These connections have a direct probabilistic interpretation, which the Boltzmann machine does not. Stochastic simulation of belief networks was intro-

duced by Pearl [54] and computes posterior probabilities by generating samples. For instance, consider a belief network defined over the set of binary variables $\mathbf{s} = \{s_1, s_2, .., s_n\}$. The parents of variable s_i are denoted by $\mathbf{s}_{\pi_i}$ and is the set of nodes such that the conditional distribution of the i-th node is,

$$P(s_i|s_1, s_2, .., s_{i-1}) = P(s_i|\mathbf{s}_{\pi_i}) \tag{4.5}$$

In this stochastic feedforward network, samples are generated based on the conditional probability of node i state given the state of its parents. That is, the probability that the i-th node is active is,

$$P(s_i = 1|\mathbf{s}_{\pi_i}) = \sigma\left(\sum_j w_{ij}s_j + \theta_i\right) \tag{4.6}$$

where w_{ij} and θ_i and the weights and biases of the network. Thus, given the state of its parents each node performs a weighted sum to determine its probability of being active. Equation 4.6 is equivalent to equations 4.3 and 4.4 where the gain function g is the sigmoid function σ. Learning has the advantage of not requiring a negative (unclamped) phase [52] and hence is more biologically plausible than the Boltzmann machine.

However, sigmoid belief networks only consider graphs of pairwise factors [52]. The networks in this thesis extend this to consider higher-order factors, as described in the next chapter.

Chapter 5

Online Inference with Spiking Neurons

To interact with the real world a system must perform inference continuously from a continuous stream of information. That is, probabilistic inference must be performed online, at the speed at which data arrives. In the real world, it is often better to have an inaccurate but quick answer than an accurate but delayed one. For instance, fast inference that a collision may be imminent is more important than knowing the exact probability.

In this chapter, two methods for performing probabilistic inference online with networks of spiking neurons are discussed, under the two constraints that inference be online and performed with spikes. The two methods are, first, a stochastic form of the belief propagation algorithm and second, an online form of Gibbs sampling. These methods perform inference on graphical models consisting of binary variable nodes.

The neural network models discussed so far assume only pairwise interactions between neurons. In this case, the factor nodes of the corresponding graphical model may be interpreted as synapses. However, graphical models may also contain higher-order interactions. It is suggested that dendritic processing involving non-linear interactions between

synapses may implement the higher-order factor nodes.

5.1 Spiky Belief Propagation

In this section an algorithm is proposed for performing probabilistic inference on a graph of spiking neurons. The proposed algorithm is a variation on the belief propagation algorithm, and as it sends messages via spikes it will be referred to as 'spiky' belief propagation. As with belief propagation, the goal is to compute the marginal probabilities of all of the variable nodes. Spikes and post-synaptic potentials are described in terms of the messages from variable-to-factor and factor-to-variable of the belief propagation algorithm. Thus, neural activity is described in terms of the belief propagation algorithm. The spiky belief propagation algorithm operates on a factor graph of binary variables and factors of arbitrary arity (pairwise or higher-order factors).

5.1.1 Variable-to-factor messages

The variable-to-factor messages of the spiky belief propagation algorithm are sent as a spike train such that the underlying firing rate r of the spike train s approximates the variable-to-factor messages n of the belief propagation algorithm that were computed by equation 2.2. The spike train from variable i to factor a is denoted $s_{ia}(x_i)$ and the value of t-th element of the sequence is $s_{ia}^t{=}1$ for a spike and $s_{ia}^t{=}0$ for no spike. Figure 5.1 shows a spike train sent to convey the variable-to-factor message. In the BP algorithm, the variable-to-factor message is the unnormalized belief b^* divided by the incoming factor-to-variable message, that is $n_{ia}(x_i) = b_i^*(x_i)/m_{ai}(x_i)$. Thus, the instantaneous firing rate r_{ia} for the individual

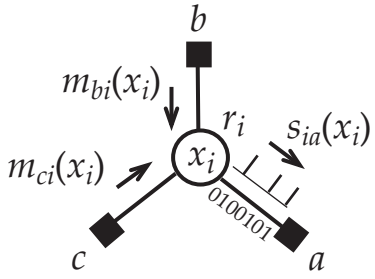

Figure 5.1: This figure shows the spike sequence s_{ia} emitted by variable node i towards factor a. The firing rate r_i of the variable node i is computed from its incoming messages to the variable node.

variable-to-factor messages, as derived from BP, is

$$r_{ia} = f\left(\frac{b_i^*(X_i)m_{ai}(\neg X_i)}{b_i^*(\neg X_i)m_{ai}(X_i)}\right) \tag{5.1}$$

where X_i is shorthand for x_i=1 and $\neg X_i$ is shorthand for x_i=0. However, this would imply that spiking neurons send a different spike train to each of their downstream neighbours, which is generally accepted not to be the case for real neurons. Real neurons have a single axonal output which sends the same spike train to all of the neuron's downstream neighbours.

5.1.2 Factor-to-variable messages

The spiky belief propagation algorithm sends factor-to-variable messages where the message values are simply entries from the factor's potential function. Thus, computing the factor-to-variable messages is just a look-up operation. The same operation in the standard belief propagation algorithm involves a sum over variables and messages sent to the factor. For example, consider a graph consisting of binary variables x_i and x_j connected by a pairwise factor a. In this case, equation 2.3 can be re-written with the sum expanded for the two values of the j-th variable, that is

$$m_{ai}(x_i) = \psi_a(x_i, X_j)n_{ja}(X_j) + \psi_a(x_i, \neg X_j)n_{ja}(\neg X_j) \tag{5.2}$$

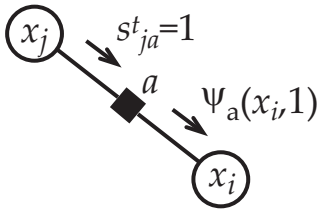

Figure 5.2: This figure shows an incoming spike to pairwise factor a at time t, i.e $s^t_{ja}(X_j) = 1$, and the resulting message $m_{ai}(x_i)$ to variable node i which is the vector $\psi_a(x_i, 1)$. The variable x_i is binary so the elements of the message vector $m_{ai}(x_i)$ are $\psi_a(1, 1)$ and $\psi_a(0, 1)$.

where X_j and $\neg X_j$ are the two values of x_j. For notational convenience, the potential function ψ is used, as factor a may represent a conditional probability of either $P(x_i|x_j)$ or $P(x_j|x_i)$ but the message computation is identical in either case.

In the previous section (5.1.1) the variable-to-factor messages $n_{ja}(x_j)$ were replaced with a spike train $s_{ja}(x_j)$. Thus, the message arriving at factor a at time t is the value of spike $s^t_{ja}(x_j)$. As spikes are generated from a normalized firing rate, $s^t_{ja}(\neg X_j) = 1 - s^t_{ja}(X_j)$. Accordingly, $s^t_{ja}(X_j)$ may be written as s^t_{ja} and $s^t_{ja}(\neg X_j)$ as $1 - s^t_{ja}$. Substituting in these spikes, s^t_{ja} for $n_{ja}(X_j)$ and $1 - s^t_{ja}$ for $n_{ja}(\neg X_j)$, into equation 5.2 we get

$$m^t_{ai}(x_i) = \psi_a(x_i, X_j)s^t_{ja} + \psi_a(x_i, \neg X_j)(1 - s^t_{ja}) \tag{5.3}$$

As the value of a spike is either 0 or 1, one of the terms in the sum in equation 5.3 is zero and the other equates to the value of the potential function. For example, when a spike is received, $s^t_{ja} = 1$, then the message $m_{ai}(x_i)$ to variable node i is the vector $\psi_a(x_i, 1)$. This is shown graphically in figure 5.2. Similarly, when no spike is received ($s^t_{ja} = 0$) the message $m_{ai}(x_i)$ is the vector $\psi_a(x_i, 0)$. Thus, the factor-to-variable message equation for a

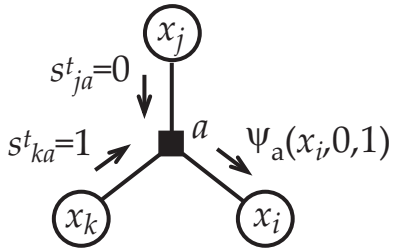

Figure 5.3: This figure shows two incoming spikes to factor a at time t. From variable j no spike is received, i.e $s^t_{ja}(x_j) = 0$ and from variable k a spike is received, i.e $s^t_{ka}(x_k) = 1$. For this particular combination of incoming spikes to factor a, the message to variable node i is the vector $\psi_a(x_i, 0, 1)$.

pairwise factor can be written in the form of a case statement, for example

$$m^t_{ai}(x_i) = \begin{cases} \psi(x_i, X_j), & s^t_{ja} = 1 \\ \psi(x_i, \neg X_j), & s^t_{ja} = 0 \end{cases} \tag{5.4}$$

Equation 5.4 can also be written as a look-up operation where the value of the spike s^t_{ja} selects a value from the potential function ψ_a. Thus, we can write the message equation as

$$m^t_{ai}(x_i) = \psi(x_i, s^t_{ja}) \tag{5.5}$$

Figure 5.3 shows another example, this time for a factor connecting three variables. In this example, the factor-to-variable message equation written in the form of a case statement is

$$m^t_{ai}(x_i) = \begin{cases} \psi_a(x_i, X_j, X_k), & s^t_{ja} = 1 \wedge s^t_{ka} = 1 \\ \psi_a(x_i, X_j, \neg X_k), & s^t_{ja} = 1 \wedge s^t_{ka} = 0 \\ \psi_a(x_i, \neg X_j, X_k), & s^t_{ja} = 0 \wedge s^t_{ka} = 1 \\ \psi_a(x_i, \neg X_j, \neg X_k), & s^t_{ja} = 0 \wedge s^t_{ka} = 0 \end{cases} \tag{5.6}$$

where $\wedge$ denotes the logical AND operation. The product of the incoming spikes to factor a is performed by the logical AND operation $\wedge$, as the

product of binary values is equivalent to a logical AND operation. Equation 5.4 can also be expressed as a look-up operation, for example,

$$m_{ai}^t(x_i) = \psi_a(x_i, s_{ja}^t, s_{ka}^t) \tag{5.7}$$

where the value of the incoming spikes, s_{ja}^t and s_{ka}^t, select a value from the potential function ψ_a.

More generally, for arbitrary factors, the message equation of spiky belief propagation is,

$$m_{ai}^t(x_i) = \psi_a(x_i, \mathbf{s}_{a\setminus i}^t) \tag{5.8}$$

where $\mathbf{s}_{a\setminus i}$ is the vector of incoming spikes to factor a except from variable node i. A look-up operation is performed in which the values of the spikes $\mathbf{s}_{a\setminus i}$ are used to select a value from the potential function.

The messages computed by equation 5.8 are only an approximation to the BP message $m(x_i)$, such that the messages $m^t(x_i)$ averaged over time are equal to the BP messages $\langle m^t(x_i)\rangle_T = m(x_i)$, in the limit of infinitely long time T. Therefore, at time t, the SBP message m^t is only a rough approximation to the message m of the standard BP algorithm. The instantaneous message m^t can be made a closer approximation by using a running average, such that

$$m_{ai}^t(x_i) = \gamma\ m_{ai}^{t-1}(x_i) + (1-\gamma)\ \psi_a(x_i, \mathbf{s}_{a\setminus i}^t) \tag{5.9}$$

where γ is the smoothing factor between zero and one. Then equation 5.8 is simply the case where $\gamma = 0$.

5.1.3 Beliefs in spiky belief propagation

The spiky belief propagation algorithm computes an approximate marginal probability for each variable node, $b_i = p(X_i)$. As with the belief propagation algorithm, the belief of a variable node is computed by multiplying together the incoming messages from the neighbouring factors, $a \in N(i)$, of

the variable node i, that is, $b_i(x_i) \propto \prod_a m_{ai}$. Substituting in the smoothed factor-to-variable messages $m^t_{ai}(x_i)$ of equation 5.9 into the belief equation we get

$$b^t_i = f\left(\prod_{a \in N(i)} \frac{m^t_{ai}(X_i)}{m^t_{ai}(\neg X_i)}\right) \tag{5.10}$$

where normalization is performed with function $f(z) = z/(1+z)$.

If we cast equation 5.10 in the log domain we can see how it could potentially be implemented by a neuron. We use the identity that a product is equivalent to an exponential of a sum of logs, $\prod \omega = e^u$ where $u = \sum \log \omega$ is used. Furthermore, the argument to the function f is an exponential, $f(e^u) = e^u/(1+e^u)$, which is equivalent to the sigmoid function, $\sigma(u) = 1/(1+e^{-u})$, that is, $f(e^u) = \sigma(u)$. Therefore the belief equation can be written as

$$b^t_i = \sigma\left(\sum_{a \in N(i)} \log \frac{m^t_{ai}(X_i)}{m^t_{ai}(\neg X_i)}\right) \tag{5.11}$$

This sum has an obvious analog in the summation performed at the soma of a neuron. In this interpretation the post-synaptic potential is the log of the ratio of message components for and against the neuron's underlying binary hypothesis. The log of the ratio is positive where the ratio is greater than one. This happens when the message component for the neuron's hypothesis is greater than the message component against, $m^t_{ai}(X_i) > m^t_{ai}(\neg X_i)$. Thus, in this case, the effect on the neuron is excitatory. Similarly, there is an inhibitory response at the neuron when the log of the ratio is negative which happens when $m^t_{ai}(X_i) < m^t_{ai}(\neg X_i)$. The temporal smoothing of messages over time is not unreasonable from a neural plausibility standpoint, as a spike arriving at a synapse typically evokes a post-synaptic response that has a slower time course than spikes themselves. The problem with this belief propagation interpretation of neural activity is that a different outgoing message is sent to each node by the

belief propagation algorithm which is not the case with real neurons.

5.2 Gibbs sampling

This section introduces a second algorithm which is shown to be similar to the spiky belief propagation algorithm. Gibbs sampling, also known as stochastic simulation, is another method for performing inference on graphical models. The Gibbs sampler is a Markov chain sampling method [23] which generates samples from a probability distribution. To do this the Gibbs sampler repeatedly replaces the value of each variable node with a value picked from its distribution conditional on the current values of all other variable nodes in the distribution. This is done for all variable nodes sequentially, in any order. The probability of any event or combination of events can be computed by counting the number of times the event occurs in the samples.

5.2.1 Gibbs sampling for belief networks

For belief networks, the use of Gibbs sampling was introduced by Pearl [54]. From the conditional independence assumptions implied by the belief network, the conditional probability that a variable node is 1 given the state of all other variables is equal to the conditional probability given the state of the variable node's Markov blanket,

$$P(s_i|\mathbf{s}_{\setminus i}) = P(s_i|\mathbf{s}_{\beta_i})$$

where $\mathbf{s}_{\setminus i}$ is all variables of $\mathbf{s}$ except s_i and $\mathbf{s}_{\beta_i}$ is the Markov blanket of variable node i. A variable node's Markov blanket is its parents, its children and its children's parents. Or put another way, a variable node's Markov blanket is every node one factor away in the graph. For the belief networks then, the Gibbs sampler repeatedly replaces the value of each

variable node with a value picked from its distribution conditional on the current values of the node's Markov blanket. That is, the probability of selecting the value 1 for variable i is

$$P(s_i{=}1|\mathbf{s}_{\beta_i}) = \frac{P(s_i{=}1|\mathbf{s}_{\pi_i}) \prod_{j:i\in\pi_j} P(s_j|s_i{=}1, \mathbf{s}_{\pi_j\setminus i})}{\sum_{s_i} P(s_i|\mathbf{s}_{\pi_i}) \prod_{j:i\in\pi_j} P(s_j|s_i, \mathbf{s}_{\pi_j\setminus i})} \tag{5.12}$$

where $\mathbf{s}_{\pi_i}$ is the parents of variable node i.

Marginal probabilities can be computed from the samples, by counting the number of times the value occurs in the samples. However, faster convergence results from taking the average over time of the variable's conditional probability given the state of its neighbours [54]. Thus, marginal $P(s_i)$ can be computed with

$$P(s_i) = \sum_t P(s_i^t|\mathbf{s}_{\beta_i}^t)\ P(\mathbf{s}_{\beta_i}^t) \tag{5.13}$$

Similarly, if there are observed nodes $\mathbf{s}_V$ then the posterior probability of s_i given the observations is

$$P(s_i|\mathbf{s}_V) = \sum_t P(s_i^t|\mathbf{s}_{\beta_i}^t, \mathbf{s}_V^t)\ P(\mathbf{s}_{\beta_i}^t, \mathbf{s}_V^t) \tag{5.14}$$

5.2.2 Gibbs sampling as message-passing

Gibbs sampling on belief networks can also be considered as a message passing algorithm. For a node to determine its conditional probability given the state of its neighbours it must receive from its neighbours their current state. This is exactly the operation that is performed by nodes in the spiky belief propagation algorithm when they send spikes to their neighbouring factors (see section 5.1.1). Factors then select the conditional probability from their conditional probability tables based on the values of their incoming spikes, as the factor-to-variable message in section 5.1.2 were. The numerator of equation 5.12 is the product of incoming messages to the node. Using the potential notation of the previous section

(5.1) equation 5.12 may be written as

$$P(s_i{=}1|\mathbf{s}_{\beta_i}) = f\left(\prod_{a\in N(i)} \frac{\psi_a(s_i{=}1, \mathbf{s}^t_{a\setminus i})}{\psi_a(s_i{=}0, \mathbf{s}^t_{a\setminus i})}\right) \tag{5.15}$$

This equation is equivalent to the equation 5.10 of the spiky belief propagation algorithm where no message smoothing is performed ($\gamma = 0$).

5.2.3 Online Gibbs sampling

Gibbs sampling on belief networks is typically performed with observation nodes clamped to a single set of observations. The simulation is run collecting many samples to take an average over to get the marginal probabilities of the hidden nodes. However, it is hard to know how long to run a simulation for to achieve a desired level of accuracy.

In an online version of Gibbs sampling, observations are updated at every timestep. This requires no change to the basic algorithm. However, it does require making the assumption that the hidden process being captured changes only slowly compared to the rate at which the data is observed. In this case, the state of the belief network can be assumed to be in a converged or nearly converged state. This will only be an appropriate strategy on certain problems, such as the motion detection task of chapter 7.

5.3 Summary of the two approaches

There are three differences between spiky belief propagation and online Gibbs sampling. The first difference is when temporal averaging (smoothing) is performed to compensate for stochastic nature of sampling based methods. In spiky belief propagation the smoothing is performed on the messages, such that the belief is the product of smoothed messages. In

online Gibbs sampling the belief is the smoothed product of messages. The second difference concerns the outgoing messages of variable nodes. In spiky belief propagation a different outgoing message is sent to each neighbouring variable node; it must divide out the incoming message on that edge. In online Gibbs sampling the same outgoing message is sent to each neighbouring variable node. The third difference is parallelism. In spiky belief propagation nodes can be updated in parallel. In online Gibbs sampling variable nodes must be updated sequentially. Although some parallelism is allowed, neighbouring nodes can not be updated simultaneously. Thus, the two methods are closely related but differ in their details. The difference between these approaches is demonstrated using the ”Burglar alarm” problem in the next section.

5.4 The Burglar Alarm Problem

This section walks through both algorithms on a classic inference task. The burglar alarm problem is a common example used to illustrate Bayesian inference and the phenomenon known as ’explaining away’ [55]. The basic problem is given by the following story.

> ”Whilst at work, Fred receives a phone call informing him that his burglar alarm is ringing. What is the probability that there is a burglar in his house? Soon after Fred hears that there was a small earthquake near his house. Fred is relieved after hearing this, knowing that it was probably the earthquake that set the alarm off. What is the probability that there is a burglar in his house, after hearing of the earthquake?”

Let, variable a be the burglar alarm, variable b be a burglar in Fred’s house, and variable e be a small earthquake near Fred’s house. The prob-

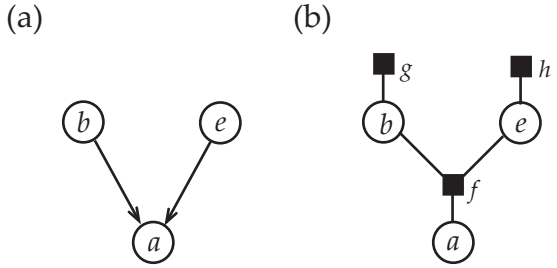

Figure 5.4: This figure shows a graphical model of the burglar alarm problem. Figure (a) shows the directed graph for this problem and (b) shows the equivalent factor graph. Variable nodes 'a', 'b' and 'e' represent the alarm, burglar and earthquake variables respectively. The factor nodes are labeled 'f', 'g' and 'h'.

ability of these variables factorizes as follows:

$$P(b, e, a) = P(a|b, e)\ P(b)\ P(e)$$

Figure 5.4 shows the belief network for this factorization. The following values are assigned to each of these probability tables. The probability of a burglar at Fred's house is $P(b{=}1) = 0.01$. The probability of an earthquake near Fred's house is $P(e{=}1) = 0.01$. The burglar alarm will be triggered by any of the following events: a burglar enters the house; a small earthquake occurs near the house; or a some other event causes a false alarm. The probability the alarm is triggered by a burglar is $\alpha_b = 0.99$. The probability the alarm is triggered by an earthquake is $\alpha_e = 0.1$. The probability the alarm is triggered by a false alarm is $f = 0.01$. Thus the probability of the alarm ringing given b and e is

$$\begin{aligned}
P(a{=}1|b{=}0, e{=}0) &= f && = 0.01 \\
P(a{=}1|b{=}1, e{=}0) &= 1 - (1 - f)(1 - \alpha_b) && = 0.9901 \\
P(a{=}1|b{=}0, e{=}1) &= 1 - (1 - f)(1 - \alpha_e) && = 0.109 \\
P(a{=}1|b{=}1, e{=}1) &= 1 - (1 - f)(1 - \alpha_b)(1 - \alpha_e) && = 0.99109
\end{aligned}$$

The belief network of figure 5.4 is tree structured (no loops) and therefore the belief propagation algorithm computes the conditional probabil-

ity of a burglar exactly. The probability of a burglar given that the alarm is ringing, $P(b{=}1|a{=}1)$, computed by the belief propagation algorithm is

$$P(b{=}1|a{=}1) = 0.476$$

The probability of a burglar given that the alarm is ringing and there was an earthquake, $p(b{=}1|a{=}1, e{=}1)$, computed by the belief propagation algorithm is

$$P(b{=}1|a{=}1, e{=}1) = 0.084$$

Thus, knowledge of the earthquake has explained away a burglar as the probable cause of the burglar alarm.

5.4.1 Burglar alarm results

The two stochastic approximation methods are compared with the exact inference of burglar probability in this section. These are the spiky belief propagation (SBP) of section 5.1 and the online Gibbs sampling (OGS) of section 5.2.

The results shown here are the mean and standard deviation of the beliefs for 10000 runs where the belief from each run is the probability of a burglar given the alarm after a settling period of 1000 timesteps. The probability of a burglar given the alarm is ringing is, for the SBP algorithm,

$$P(b{=}1|a{=}1) = 0.476 \pm 0.01$$

and for the OGS algorithm,

$$P(b{=}1|a{=}1) = 0.476 \pm 0.01$$

The probability of a burglar given the alarm is ringing and an earthquake has occurred is, for the SBP algorithm,

$$P(b{=}1|a{=}1, e{=}1) = 0.084 \pm 0.0$$

and for the OGS algorithm,

$$P(b{=}1|a{=}1, e{=}1) = 0.084 \pm 0.0$$

The variance is zero because both the alarm and earthquake variable nodes are clamped to their observed values, so the message sent to the burglar variable node is the same every timestep.

By comparing these results to the exact values it can be seen that both the stochastic form of belief propagation and the online form of Gibbs sampling give accurate approximations to the exact values.

5.4.2 Burglar alarm analysis

This section shows how the two algorithms arrive at the approximately correct marginal probabilities. Firstly, in the standard belief propagation algorithm the factor-to-variable $m_{fb}(b{=}1)$ message to variable b is computed by summing over the values of e for incoming messages from variable e, $n_{ef}(e)$ multiplied by the CPT entries $P(a|b, e)$, that is,

$$\begin{aligned} m_{fb}(b{=}1) &= P(a{=}1|b{=}1, e{=}1)\ n_{ef}(e{=}1) + P(a{=}1|b{=}1, e{=}0)\ n_{ef}(e{=}0) \\ &= P(a{=}1|b{=}1, e{=}1)\ P(e{=}1) + P(a{=}1|b{=}1, e{=}0)\ P(e{=}0) \\ &= P(a{=}1, e{=}1|b{=}1) + P(a{=}1, e{=}0|b{=}1) \\ &= P(a{=}1|b{=}1) \end{aligned}$$

Similarly the message for $b{=}0$ is $m_{fb}(b{=}0) = P(a{=}1|b{=}0)$. The degree of belief in a burglar given that the alarm is ringing, $P(b{=}1|a{=}1)$, is computed by multiplying the incoming messages to variable node b and normalizing.

That is,

$$\begin{aligned}
\text{Bel}(b{=}1|a{=}1) &= \frac{m_{fb}(b{=}1)\; m_{gb}(b{=}1)}{m_{fb}(b{=}1)\; m_{gb}(b{=}1) + m_{fb}(b{=}0)\; m_{gb}(b{=}0)} \\
&= \frac{P(a{=}1|b{=}1)\; P(b{=}1)}{P(a{=}1|b{=}1)\; P(b{=}1) + P(a{=}1|b{=}0)\; P(b{=}0)} \\
&= \frac{P(a{=}1, b{=}1)}{P(a{=}1)} \\
&= P(b{=}1|a{=}1)
\end{aligned}$$

Thus, the belief is the conditional probability of a burglar given the alarm is ringing.

In both stochastic algorithms, SBP and OGS, the variable-to-factor messages are spikes s^t. The instantaneous factor-to-variable message to variable node b at time t is denoted, m^t_{fb}. This instantaneous message is simply an entry from the conditional probability table

$$m^t_{fb} = \begin{cases} P(a{=}1|b{=}1, e{=}1), & s^t_{ef} = 1 \\ P(a{=}1|b{=}1, e{=}0), & s^t_{ef} = 0 \end{cases}$$

where s^t_{ef} is a spike - the sampled state of variable node e at time t.

For the SBP algorithm the outgoing messages n are replaced by samples s, such that samples s^t_{ef} are drawn with a probability of n_{ef}, which is $P(e{=}1)$. Thus, as the firing rate r is the average sample value over time, $r = \langle s^t \rangle_T$, the firing rate r_{ef} approximates $P(e{=}1)$. The message $m^t_{fb}(b{=}1)$ averaged over time is

$$\begin{aligned}
\left\langle m^t_{fb}(b{=}1) \right\rangle_T &= P(a{=}1|b{=}1, e{=}1)\; r_{ef} + P(a{=}1|b{=}1, e{=}1)\; (1 - r_{ef}) \\
&\approx P(a{=}1|b{=}1, e{=}1)\; P(e{=}1) + P(a{=}1|b{=}1, e{=}1)\; P(e{=}0) \\
&= P(a{=}1|b{=}1)
\end{aligned}$$

where $\langle . \rangle_T$ denotes an average over T timesteps. Therefore, the average of the instantaneous message over time is an approximation to the belief propagation message m_{fb}. This spiky belief propagation algorithm

approximates the belief propagation algorithm by averaging the instantaneous factor-to-variable messages m^t over time.

In the case of the OGS algorithm no averaging is performed on messages. In the OGS algorithm the conditional probability is computed at variable node b from the incoming messages m^t_{fb} and m^t_{gb}. This conditional probability is denoted $\text{Bel}^{(t)}(b{=}1)$ and may also be referred to as the instantaneous belief. Thus, the instantaneous belief at time t for $s^{(t)}_{ef} = 1$ is

$$\begin{aligned}
\text{Bel}^t(b{=}1) &= \frac{m^t_{fb}(b{=}1)\; m_{gb}(b{=}1)}{m^t_{fb}(b{=}1)\; m_{gb}(b{=}1) + m^t_{fb}(b{=}0)\; m_{gb}(b{=}0)} \\
&= \frac{P(a{=}1|b{=}1,e{=}1)\; P(b{=}1)}{P(a{=}1|b{=}1,e{=}1)\; P(b{=}1) + P(a{=}1|b{=}0,e{=}1)\; P(b{=}0)} \\
&= \frac{P(a{=}1,b{=}1|e{=}1)}{P(a{=}1,b{=}1|e{=}1) + P(a{=}1,b{=}0|e{=}1)} \\
&= \frac{P(a{=}1,b{=}1|e{=}1)}{P(a{=}1|e{=}1)} \\
&= P(b{=}1|a{=}1,e{=}1)
\end{aligned}$$

The incoming messages to variable node b compute the conditional probability $P(b{=}1|a{=}1,e{=}1)$. Similarly for $s^t_{ef} = 0$ the instantaneous belief $\text{Bel}(b{=}1)^t$ is $P(b{=}1|a{=}1,e{=}0)$. Thus, $\text{Bel}^t(b{=}1)$ is

$$\text{Bel}^t(b{=}1) = \begin{cases} P(b{=}1|a{=}1,e{=}1), & s^t_{ef} = 1 \\ P(b{=}1|a{=}1,e{=}0), & s^t_{ef} = 0 \end{cases}$$

In this case the firing rate is approximately $r_{ef} \approx P(e{=}1|a{=}1)$. Taking the average of the instantaneous belief over time gives

$$\begin{aligned}
\left\langle \text{Bel}^t(b{=}1) \right\rangle_T &= P(b{=}1|a{=}1,e{=}1)\; r_{ef} + P(b{=}1|a{=}1,e{=}0)\; (1 - r_{ef}) \\
&\approx P(b{=}1|a{=}1,e{=}1)\; P(e{=}1|a{=}1) + P(b{=}1|a{=}1,e{=}0)\; P(e{=}0|a{=}1) \\
&= P(b{=}1,e{=}1|a{=}1) + P(b{=}1,e{=}0|a{=}1) \\
&= P(b{=}1|a{=}1)
\end{aligned}$$

which is the conditional probability of a burglar given the alarm is ringing.

This shows the difference in the way the SBP and OGS algorithms compute the degree of belief.

5.5 Neural plausibility

Real neurons are highly complex in their workings. A common idealization is to treat the computation performed by a neuron as a linear weighted sum of its input. The variable nodes, in both of the approaches above, multiply their incoming messages to compute their posterior probability. This multiplication in the log domain becomes a sum, an operation that is easily performed by neurons. Furthermore, where the nodes of the belief network are connected by pairwise factors the conditional probabilities of these factors reduce to a single weight value. Thus, a network of stochastic neurons performing a linear weighted sum is equivalent of a belief network of pairwise factors. This equivalence is shown in the following section.

Both algorithms lack neural plausibility in many respects. For example, the online Gibbs sampling algorithm lacks plausibility in that it requires that nodes be updated sequentially, whereas neurons in the brain change their states simultaneously, that is, in parallel. The spiky belief propagation algorithm implements parallelism but requires that nodes send out messages at different spike rates to their different neighbours. Real neurons have a single axonal output and a single output spike train.

5.5.1 Synaptic weights and pairwise factors

This section shows the connection between pairwise factors and synaptic weights. Or, expressed another way, it shows that a network of spiking neurons that compute using only weighted sums implements approx-

imate Bayesian inference. Pairwise factors only reduce to a simple weight in the case where there is no smoothing applied to messages. That is, the spiky belief propagation algorithm with $\gamma = 0$ or the online Gibbs sampling algorithm.

Consider a graph consisting only of pairwise factors. In this case, message equation 5.3 can be re-written as $m_{ai}^t = \psi_a(\neg X_j)^{(1-s_{ja}^t)}\psi_a(X_j)^{s_{ja}^t}$. Therefore, the belief equation is

$$b_i^t(x_i) \propto \prod_{j \in N(i)} \psi_a(x_i, \neg X_j)^{(1-s_{ja}^t)}\psi_a(x_i, X_j)^{s_{ja}^t} \tag{5.16}$$

where $\propto$ indicates that normalization is required. The normalized belief b_i^t can be written as the product of the ratio of messages where the function $f(x) = x/(1+x)$ converts a ratio into a normalized value. The ratio of messages is denoted by $\phi_a(\neg X_j) = \frac{\psi_a(X_i, \neg X_j)}{\psi_a(\neg X_i, \neg X_j)}$ and $\phi_a(X_j) = \frac{\psi_a(X_i, X_j)}{\psi_a(\neg X_i, X_j)}$. The belief equation to compute a normalized belief is then

$$b_i^t = f\left(\prod_{j \in N(i)} \phi_a(\neg X_j)^{(1-s_{ja}^t)}\phi_a(X_j)^{s_{ja}^t}\right) \tag{5.17}$$

Equation 5.17 can be re-arranged such that the product of $\phi_a(\neg X_j)$ for all neighbours is computed and then $\phi_a(\neg X_j)$ is divided out where a spike is received, $s_{ja}^t = 1$, that is

$$b_i^t = f\left(\prod_{j \in N(i)} \phi_a(\neg X_j)\left[\frac{\phi_a(X_j)}{\phi_a(\neg X_j)}\right]^{s_{ja}^t}\right) \tag{5.18}$$

Let $k_i = \prod_{j \in N(i)} \phi_a(\neg X_j)$ and $\omega_{ij} = \frac{\phi_a(X_j)}{\phi_a(\neg X_j)}$, then substituting k_i and ω_{ij} into the belief equation is,

$$b_i^t = f\left(k_i \prod_{j \in N(i)} [\omega_{ij}]^{s_{ja}^t}\right) \tag{5.19}$$

Recall equation 5.11 in which the belief equation was considered in the log domain. The belief equation 5.19 in the log domain is

$$b_i^t = \sigma\left(\log k_i + \sum_{j \in N(i)} \log(\omega_{ij}) s_{ja}^t\right) \tag{5.20}$$

Equation 5.20 is identical to the sigmoid belief network defined by equation 4.6, where $\theta_i = \log k_i$, $w_{ij} = \log(\omega_{ij})$ and the gain function g is the sigmoid function σ. Thus, spikes propagating in a network of stochastic neurons are equivalent to the spiky belief propagation algorithm on a factor graph of pairwise factors, however with the added benefit that the synaptic efficacy w_{ij} can now be related to the potential functions ψ_a,

$$w_{ij} = \log\left(\frac{\psi_a(X_i, X_j)\psi_a(\neg X_i, \neg X_j)}{\psi_a(\neg X_i, X_j)\psi_a(X_i, \neg X_j)}\right) \tag{5.21}$$

Therefore the factor a connecting nodes i and j can be viewed as a synapse with weight w_{ij}. The bias θ_i is the probability of firing when no spikes are received, that is, $\log k$, which is

$$\theta_i = \sum_j \log\left[\frac{\psi_a(X_i, 0)}{\psi_a(\neg X_i, 0)}\right] \tag{5.22}$$

The firing rate r is the probability of emitting a spike is, which is

$$p(s_i = 1 | \mathbf{s}_j) = r_i = \sigma\left(\sum_j w_{ij} s_j + \theta_i\right) \tag{5.23}$$

This confirms the equivalence of stochastic neurons and stochastic binary variable nodes in OGS and SBP where $\gamma = 0$.

Thus, in this context, the synaptic weights and neuron bias have probabilistic interpretations.

5.5.2 Dendritic processing and higher-order factors

This section discusses running OGS algorithm (or SBP with $\gamma = 0$) on graphical models with higher-order factors and the possible neural inter-

pretation of such higher-order factors. A higher-order factor is a factor connecting three or more nodes. An example is the 3-node factor shown in figure 5.3. As described in the previous section 5.5.1 pairwise factors were interpreted as synapses. This corresponds to the assumption that neurons perform a linear weighted sum of their input as given by equation 5.23. However, there is evidence that there may in fact be highly nonlinear interactions between nearby synapses on the dendritic tree of a neuron [48][49]. For instance, an inhibitory synapse can cancel the effect of a neighbouring excitatory synapse which is the analog equivalent of a logical AND-NOT operation [41]. Therefore, a neuron may not simply add post-synaptic responses of synapses independently but rather the combined effect of several synapses should be considered. Accordingly, a higher order factor is interpreted as a subunit of the dendritic tree to which several synapses attach and where those synapses interact non-linearly. These dendritic subunits have also be referred to as synapse clusters [65].

For higher-order factors it is not possible to perform the re-arrangement to equation 5.18 of the previous section, and so the factor does not reduce to a single synaptic weight value. Instead, the factor is defined by a synaptic weight function, where this weight is a function of incoming spikes. Consider variable node i to which factor a sends messages. Belief equation 5.15 for the OGS algorithm in the log domain is

$$b_i^t = \sigma\left(\sum_{a\in N(i)} \log\left[\frac{\psi_a(X_i, \mathbf{s}_{a\setminus i}^t)}{\psi_a(\neg X_i, \mathbf{s}_{a\setminus i}^t)}\right]\right) \tag{5.24}$$

where the set of incoming spikes to factor a is denoted $\mathbf{s}_{a\setminus i}$ and $a\setminus i$ is all variable nodes of factor a except the i-th node. Therefore, the synaptic weight as a function of spikes $\mathbf{s}_{a\setminus i}$ is denoted

$$w_a(\mathbf{s}_{a\setminus i}) = \log\left(\frac{\psi(X_i, \mathbf{s}_{a\setminus i})}{\psi(\neg X_i, \mathbf{s}_{a\setminus i})}\right) \tag{5.25}$$

where the synapse evokes a response $w_a(\mathbf{s}_{a\setminus i})$ at variable node i for the

spikes $\mathbf{s}_{a\setminus i}$ arriving at the factor a. For example, consider two spikes s_j and s_k arriving at factor a, where the spikes are $s_j{=}1$ and $s_k{=}0$. The change in membrane potential Δu_i is the synaptic weight $w(s_j{=}1, s_k{=}0)$. If s_j is an excitatory synapse and s_k is an inhibitory synapse then the AND-NOT operation would be satisfied by a positive value for $w(s_j{=}1, s_k{=}0)$ and zero for the other weights. This weight function captures the non-linear interaction between spikes s_j and s_k, that could not be captured with individual weight functions, that is $w(s_j, s_k) \neq w(s_j) + w(s_k)$.

By adding additional variable nodes to the graph it is possible to create an equivalent graph of only pairwise connections. However, graphs of higher-order factors may be considerably more compact (they contain fewer nodes).

5.5.3 Population coding

In the stochastic inference methods discussed above, a neuron's firing rate represents its degree of belief in a binary hypothesis. However, the world consists of multi-valued and continuous-valued variables that an agent would need to model. A probability density function of a continuous variable is perhaps the most natural way to represent uncertainty in that variable [39]. Any probability density function, $p(x)$, can be represented by its value at a few sampled points along x, where using more points gives a better approximation to $p(x)$. Thus, to represent continuous variables a population of neurons is used where each neuron represents a particular value of the variable. This kind of representation is known as a 'population code'. Neurons in the cortex have been observed to have bell-shaped tuning functions centered on their preferred value [46]. The tuning function of a neuron defines the relationship between the firing rate of the neuron and the variable value it represents [62]. A receptive field is a particular example of a tuning function where the neuron response is a function of

position on the retina [36]. If neurons have overlapping tuning functions then the activity of a population of neurons can accurately approximate a continuous variable.

5.6 Probabilistic Inference with Spiking Neurons

Several models for probabilistic inference with networks of spiking neurons recently proposed by Rao [58], Deneve [15], and Zemel et al [68]. A method for performing belief propagation in networks of spiking neurons was described by Rao in [58]. In Rao's approach a population of neurons is used where each neuron represents one state of the hidden variable of a Hidden Markov model. Neural activity is then directly related to the belief propagation algorithm, where the output firing of a neuron conveys an element of the message vector. Thus, as each neuron is recurrently connected to all others every neuron receives the complete message vector. Rao suggests that belief propagation be performed in the log domain, so that the product of messages at a node becomes a sum. However, the operation performed at a factor is to sum over the elements of the message vector. Therefore, considering belief propagation in the log domain leads to the problem of having to approximate this log-sum with a sum-of-logs. In the spiky belief propagation algorithm, this is not a problem because all but one element in the sum is zero.

In the model of Deneve [15] a neuron's firing rate does not represent a probability. Instead, a neuron represents the log "odds" ratio for a preferred binary-valued state. This ratio is the log of the probability the state is true over the probability the state is not true, given all the input the node has received. Each synaptic spike gives evidence to the logs odds ratio where the synaptic weight describes how informative it was to receive a spike. These synaptic weights are similar to those used in the SBP algorithm (see section 5.5.1) where the weight is a log of the ratio of condi-

tional probabilities. However, this model differs from SBP in that neurons send only new information. In Deneve's model, neurons fire when the difference between a neuron's log odds ratio and a prediction of the log odds ratio (based on the output spikes) reaches a threshold. However, this model is restricted to graphs of pairwise factors and thus makes additional conditional independence assumptions, such that the joint probability can be pairwise factorized.

In a related work by Zemel et al [68] a population of nodes encodes a log probability distribution for a continuous variable. In their approach, a recurrent network takes spikes as input to capture the probabilistic dynamics of an underlying hidden variable. Zemel et al's approach can be seen as a population extension to Deneve and is similar to the approach of Rao's in that it uses a population code. However, Zemel et al use a spatio-temporal decoder to define the relationship between output spikes over time and the probability distribution. For instance, in their experiment they use a spatial kernel which is Gaussian and a temporal kernel which is an exponential decay. Thus, their method can represent continuous values. Their decoder can reproduce Rao's where the spatial and temporal kernel are delta functions.

Chapter 6

Online Learning in Networks of Stochastic Neurons

This chapter introduces an online unsupervised algorithm for the stochastic belief networks of the previous chapter. This learning algorithm is based on the frequency of the correlated firing of neurons and is an online and stochastic variation of the well known EM algorithm [14], which was introduced in chapter 3. In this chapter a stochastic version of the EM algorithm is used to learn synaptic weights. As the synaptic weight can be computed from the conditional probability table (CPT) entries (see section 5.5.1), this is really the problem of finding the Maximum Likelihood estimate of the CPT entries.

6.1 Stochastic EM

The stochastic EM algorithm (SEM) [11] is a variation on the standard EM algorithm where the inferred hidden variables of the E step are replaced with samples. The E step involves generating a sample for each hidden variable from the posterior probability of the hidden variable given the observed variables, $P(h_i|\mathbf{v})$. Then, in the M step the parameters (CPTs) are

updated from the sampled values of the hidden variables as if the samples were observed values. Intuitively, the sample "fills-in" the missing data and learning updates the parameters based on the samples. The E step and M step must be iterated as they are in the standard EM algorithm. Because of the stochastic nature of generating samples it is possible that in some iterations the likelihood may actually decrease, although overall the likelihood will increase. An advantage of the stochastic EM algorithm is that it may escape local maxima.

Other stochastic variations of EM have been proposed such as Monte Carlo EM (MCEM) [66] where the average of several samples is used in the E step, and the SAEM (stochastic approximation EM) algorithm [12] which is an annealed variant that guarantees almost-sure convergence [16].

For the stochastic inference algorithms introduced in the previous chapter, the spikes generated by neurons are samples from the approximating posterior, $Q^t(h_i|\mathbf{v}) \approx P(h_i|\mathbf{v})$, where each spike s_i^t is a sample. The stochastic E step involves running the inference algorithm over the entire dataset generating samples for each of the hidden variables. The samples $\mathbf{s}_H^t$ for the hidden variables replace the inferred hidden variable value Q^t. The M step update of CPT entries is based on the counts of the neuron states $\mathbf{s}$ generated for the entire dataset. Consider the example of section 3.2 where the CPT update for the standard EM algorithm was $P(x_2{=}i|x_1{=}j) \propto \sum_t\ I[x_1^t{=}j]\ Q^t(x_2{=}i|x_1, x_3)$. For the SEM algorithm the inferred hidden variable value $Q^t(x_2{=}i|x_1, x_3)$ is replaced by the sample $s_2^t{=}i$. That is, the update for the CPT is

$$P(x_2{=}i|x_1{=}j) = \frac{\sum_t\ I[s_1^t{=}j]\ I[s_2^t{=}i]}{\sum_t \sum_k\ I[s_1^t{=}j]\ I[s_2^t{=}k]} \tag{6.1}$$

where s_1 is observed and I is the indicator function, as before. The sums over the indicator functions are counts of the samples and can be written as

$$P(x_2{=}i|x_1{=}j) = \frac{c(s_2{=}i, s_1{=}j)}{c(s_2{=}i, s_1{=}j) + c(s_2{=}(1{-}i), s_1{=}j)} \tag{6.2}$$

where $c(.)$ denotes a count.

For each iteration of the EM algorithm the E step is run over the entire dataset. Thus, this EM process may be quite slow especially if the dataset is large.

6.2 Online SEM

To overcome the inefficiency of batch versions of the EM algorithm, an online EM algorithm [3][18] is introduced. The online EM algorithm is run over blocks of the dataset, updating parameters after observing each block of data. In the extreme case a block size of 1 is used, such that the E and M steps are iterated as each item of data is observed. The E step, generates samples for the hidden variables from the observations and current parameters using the stochastic inference algorithm. Then, the expected sufficient statistics, that is the current counts, are updated with the new sample. For example, each of the counts for the CPT connecting variable nodes x_1 and x_2 are updated with

$$c^t(s_2{=}i, s_1{=}j) = c^{t-1}(s_2{=}i, s_1{=}j) + I[s_1^t{=}j]\ I[s_2^t{=}i] \tag{6.3}$$

Then, in the M step, the CPT entries are computed from the counts by equation 6.2. Next, with the updated parameters and a new observation the E step is performed again, updating the counts, then the next M step is performed and so on - iterating the E and M steps. This may be referred to as the accumulated-counts (AC) learning rule.

A potential problem of this approach is that the earlier samples are generated from parameters that have not yet converged on their correct ML estimate. However, the influence of those earlier counts fades as the total counts rise [18]. Alternatively, a learning rate could be introduced to decay the counts. This is the approach taken by Andrieu and Doucet [3]

in their online SEM algorithm, where the counts are updated by

$$c^t(s_2{=}i, s_1{=}j) = (1-\eta)\; c^{t-1}(s_2{=}i, s_1{=}j) + \eta\; I[s_1^t{=}j]\; I[s_2^t{=}i] \tag{6.4}$$

This learning rule mixes the previous counts c^{t-1} with the current sample at a proportion given by the learning rate η. This may be referred to as the decayed-counts (DC) learning rule. The accumulated-counts learning rule of equation 6.3 corresponds to the case where the learning rate η exponentially decreases to zero as time increases to infinity.

The synaptic weight can be computed from the counts. By substituting the CPT entries from equation 6.2 and the pairwise synaptic weight equation 5.21 gives

$$w_{ij} = \log\left(\frac{c(s_i{=}1, s_j{=}1)\; c(s_i{=}0, s_j{=}0)}{c(s_i{=}0, s_j{=}1)\; c(s_i{=}1, s_j{=}0)}\right) \tag{6.5}$$

where the normalization terms have canceled. Thus, the counts are the basic quantity of interest from which the synaptic weight can be computed. For higher-order synapses, the synaptic weight functions (equation 5.25) can also be computed from their counts. For instance, the synaptic weight w_a received by variable node i is

$$w_a(\mathbf{s}_{a\setminus i}) = \log\left(\frac{c(s_i{=}1, \mathbf{s}_{a\setminus i})}{c(s_i{=}0, \mathbf{s}_{a\setminus i})}\right) \tag{6.6}$$

where $\mathbf{s}_{a\setminus i}$ is the incoming spikes to synapse a.

6.3 Spike-timing dependent plasticity

The method of learning typically prescribed for networks of spiking neurons is spike-timing dependent plasticity (STDP) [45][6], in which synaptic weight is adjusted based on the timing of the pre-synaptic and post-synaptic spikes. In STDP the synaptic weight is increased where the pre-synaptic spike precedes the post-synaptic spike. This is learning of the

kind Donald Hebb postulated [32], and can be summarized with the phrase "neurons that fire together wire together". For this reason it is often referred to as Hebbian learning. A complementary anti-Hebbian learning rule has also been suggested [29], where when the pre-synaptic neuron fires after the post-synaptic neuron the synaptic weight is decreased, to counteract the Hebbian rule.

The online stochastic EM learning of section 6.2 is also a form of Hebbian learning. In STDP the difference in timing determines the weight increase which is at its greatest when the firing times of pre-synaptic and post-synaptic neurons coincide, and decreases the earlier the firing time of the pre-synaptic neuron. Consider a discrete time model where the weight increase occurs when the firing times of pre-synaptic and post-synaptic neurons coincide. For example, consider two neurons i and j connected by synapse weight w_{ij}. If both neurons fire at time t, $s_i{=}1$ and $s_j{=}1$, then the count $c(s_i{=}1, s_j{=}1)$ increments by 1 and the other counts remain unchanged. Therefore, the synaptic weight of equation 6.5 increases. Conversely, the synaptic weight decreases when only one of the neurons fire, that is, their firing is not correlated. Thus, the synaptic weight captures the degree of correlation between the firing of the synapse's pre-synaptic and post-synaptic neurons. Where a pair of neurons' firing is correlated, that is both neuron tend to fire at same time or both do not fire at same time, then the ratio in equation 6.5 is greater than one. That is, the synaptic weight is positive or excitatory. Conversely, the synaptic weight is negative or inhibitory where the ratio of counts in equation 6.5 is less than one. The online stochastic EM algorithm is a form of Hebbian learning where a pair of neurons fire together, and anti-Hebbian where the pre-synaptic neuron fires either before or after the post-synaptic neuron.

Chapter 7

Motion Detection

Perhaps the most fundamental task performed by the visual system is the detection of motion. There are two principal methods that can be used to infer motion from a sequence of images. One method, common in computer vision, is to identify features, edges or objects in each image and to infer motion from changes in the spatial positions of these elements over time. This approach is known as the 'long-range' motion scheme [9][10] and gives rise to the 'correspondence problem'. The correspondence problem, in this case, is the problem of finding the corresponding feature in both images. An alternative method is known as 'short-range' or intensity-based motion. In this approach a low-level process extracts a motion signal based on light intensity at two spatially and temporal nearby points on the image, without explicit reference to spatial features. The most well-known computational model of this type is the 'motion-energy' model [1]. This 'motion-energy' model is the commonly accepted model of how humans and animals perform visual motion detection. This approach gives rise to the aperture problem in which a local reading of velocity is ambiguous in determining the global velocity. Motion detection is a difficult problem because of ambiguity resulting from the aperture problem and uncertainty imposed by noisy sensors.

In this chapter the stochastic inference algorithms and online EM learning algorithms are demonstrated by applying them to the task of detecting visual motion. The basic task discretizes the moving image into a sequence of static images. The proposed solution to this motion detection task uses a belief network composed of higher-order factors to correlate observed image intensities between images. On this task there is little distinction between the online Gibbs sampling algorithm and the spiky belief propagation and the inference algorithm could be considered as either OGS with parallel updating or SBP with no message smoothing ($\gamma = 0$).

7.1 Motion Detection Problem

A simplified and abstracted motion detection problem is considered here. This task is somewhat artificial compared to that faced in the real world domain, but has the advantage that optimal solution can be derived analytically. The basic task consists of inferring the direction of motion from a sequence of 1D images where each image is a random pattern of binary pixel values. At every timestep the image is shifted with an unknown velocity for which it is the goal of the belief network to infer. However, the problem is made challenging by corrupting the observed image with noise. The belief network observes only the noise corrupted image pixels where each observation node of the belief network observes a corrupted pixel of the image.

The true image $\mathbf{x}$ is an array of random binary pixels where each image pixel x_i has a probability α of being the value 1. The observed image $\mathbf{y}$ is a noise corrupted copy of the true image $\mathbf{x}$. Each observed pixel y_i has a probability β of being corrupted. A corrupted pixel y_i is redrawn and has probability α of being 1. Otherwise, y_i is uncorrupted and takes the value of x_i. Thus, the noise is unbiased as it leaves the proportion of 0's to 1's in the image unchanged.

The true image $\mathbf{x}$ is shifted by a discrete number of pixels v, such that the image has a velocity v. For instance, a leftward velocity of 1 pixel per timestep copies the new image pixel value from current pixel value at the position 1 pixel to the right, $x_i^{t+1} = x_{i-1}^t$. At every timestep the velocity has the probability ν of changing to a new value. The true image $\mathbf{x}$ remains the same (except shifted), however the observed image $\mathbf{y}$ is generated from $\mathbf{x}$ every timestep.

7.2 Generative Model of Image Motion

A first-order generative model of motion is proposed in which the velocity and current image are considered to generate the subsequent image. The factors of the belief network are the conditional probabilities $P(y_i|x_i)$ and $P(x_j^{t+1}|x_i^t, v_k)$ (see figure 7.1). The conditional probability $P(y_i|x_i)$ is known as the observation model. The following table shows the observation model where the probability of an observation y_i given the actual image value x_i is computed from parameters α and β.

$P(y_i|x_i)$

	x_i=0	x_i=1
y_i=0	$1-\alpha\beta$	$(1-\alpha)\beta$
y_i=1	$\alpha\beta$	$1-(1-\alpha)\beta$

The velocity node v_k represents a velocity of $V_k = j - i$, such that it takes 1 timestep for an image pixel to move from i-th to j-th position (see figure 7.1). Thus, if the current image velocity is V_k (v_k=1) then the value observed at the j-th pixel at time $t+1$ is the same as that observed at the i-th pixel at time t, that is, $P(x_j^{t+1}{=}a|x_i^t{=}a, v_k{=}1) = 1$, where a is observed image pixel value. Otherwise, if the current image velocity is not V_k (v_k=0) then the value observed at the j-th pixel does not depend on x_i^t and therefore $P(x_j^{t+1}{=}a|x_i^t{=}a, v_k{=}0) = \alpha$. The following table shows the

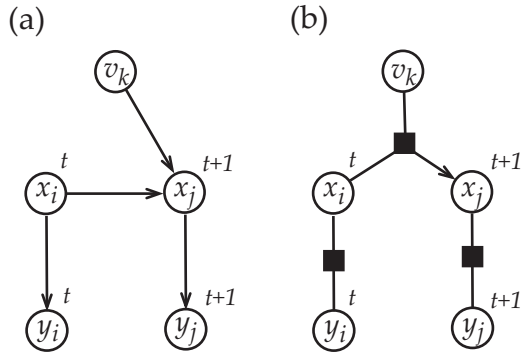

Figure 7.1: This figure shows a generative model of the motion detection task. Binary variable node v_k represents a velocity of V_k where $v_k = 1$ if velocity is V_k and v_k is zero otherwise. Binary variable node x_i^t represents the value of the i-th pixel at time t, from the true image, and similarly binary variable node x_j^{t+1} represents the value of the j-th pixel at time $t+1$. Binary variable nodes y_i^t and y_j^{t+1} represent the noise corrupted pixels values and are the observation nodes.

position-velocity transition model for the conditional probability of pixel value x_j^{t+1} given the velocity value v_k and current pixel value x_i^t.

$P(x_j^{t+1}|x_i^t, v_k)$

	v_k=0, x_i^t=0	v_k=0, x_i^t=1	v_k=1, x_i^t=0	v_k=1, x_i^t=1
x_j^{t+1}=0	$1-\alpha$	$1-\alpha$	1	0
x_j^{t+1}=1	α	α	0	1

7.3 Recognition Model of Image Motion

A recognition model is proposed that simplifies the generative model of figure 7.1. The motion detection subgraph in figure 7.2 perform inference in a single step. In this subgraph the true image pixels x_j^{t+1} and x_i^t are not explicitly modeled. Each motion detecting subgraph is composed of two sensors, y_i and y_j, that detect the light intensity at the i-th and j-th position of the visual field, and a variable node v_k that represents velocity of motion

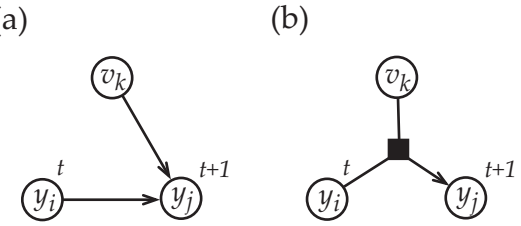

Figure 7.2: This figure shows a graphical model for motion detection. Variable nodes y_i and y_j observe sensory input and variable node v_k represents the velocity of motion. The factor $P(y_j^{t+1}|y_i^t, v_k)$ is the conditional probability of observing an object at y_j^{t+1} given an observation at y_i^t and an object velocity of v_k.

V_k where V_k is the velocity this subgraph is tuned to detect (see figure 7.2). The conditional probability $P(y_j^{t+1}|y_i^t, v_k)$ represents the correlation between the observations at the sensors nodes and the velocity of motion (the factor of the subgraph of figure 7.2). This conditional probability can be computed from $P(y_i|x_i)$ and $P(x_j^{t+1}|x_i^t, v_k)$ of the full generative model, as follows.

The conditional probability, $P(y_j^{t+1}|y_i^t, v_k)$, can be calculated from $P(y_j^{t+1}, y_i^t|v_k)$ using

$$P(y_j^{t+1}|y_i^t, v_k) = \frac{P(y_j^{t+1}, y_i^t|v_k)}{P(y_i^t|v_k)} \tag{7.1}$$

Therefore, we need to compute the conditional probability of the sensors given the velocity, $P(y_j^{t+1}, y_i^t|v_k)$. This can be done by summing the joint $P(y_j, y_i, x_j, x_i|v_k)$ over the true image pixels x_i and x_j. That is,

$$P(y_j, y_i|v_k) = \sum_{x_j} \sum_{x_i} P(y_j, y_i, x_j, x_i|v_k)$$

Note that the t subscripts have been omitted for notational clarity. Applying the product rule this can be re-written as

$$P(y_j, y_i|v_k) = \sum_{x_j} \sum_{x_i} P(y_j, y_i|x_j, x_i, v_k)\ P(x_j, x_i|v_k)$$

By further application of the product rule to $P(y_j, y_i|x_j, x_i, v_k)$ and to $P(x_j, x_i|v_k)$ we get

$$P(y_j, y_i|v_k) = \sum_{x_j} \sum_{x_i} P(y_j|x_j, x_i, v_k)\ P(y_i|x_j, x_i, v_k)\ P(x_j|x_i, v_k)\ P(x_i|v_k)$$

This can be simplified by noting that variable y_j is conditionally independent of x_i and v_k given x_j. Similarly, variable y_i is conditionally independent of x_j and v_k given x_i. The conditional independence assumptions implicit in the generative model (figure 7.1) allow the simplifications of $P(y_j|x_j, x_i, v_k)$ to $P(y_j|x_j)$ and of $P(y_i|x_j, x_i, v_k)$ to $P(y_i|x_i)$ giving

$$P(y_j, y_i|v_k) = \sum_{x_j} \sum_{x_i} P(y_j|x_j)\ P(y_i|x_i)\ P(x_j|x_i, v_k)\ P(x_i|v_k) \tag{7.2}$$

Thus, the conditional probabilities of the slightly more compact graph of figure 7.2 can be written in terms of the conditional probabilities of the full generative model of figure 7.1. These conditional probabilities, $P(y_j|x_j)$ and $P(x_j|x_i, v_k)$, can written in terms of parameters α and β.

Substituting the values of the conditional probabilities $P(x_j^{t+1}|x_i^t, v_k)$ and $P(y_i|x_i)$ (shown in preceding CPTs) into equation 7.2 and simplifying we get

$P(y_j^{t+1}, y_i^t|v_k)$

	v_k=0	v_k=1
y_j^{t+1}=0, y_i^t=0	$(1-\alpha)^2$	$(1-\alpha)(1-\alpha\beta(2-\beta))$
y_j^{t+1}=0, y_i^t=1	$(1-\alpha)\alpha$	$\alpha(1-\alpha)\beta(2-\beta)$
y_j^{t+1}=1, y_i^t=0	$(1-\alpha)\alpha$	$\alpha(1-\alpha)\beta(2-\beta)$
y_j^{t+1}=1, y_i^t=1	α^2	$\alpha(1-(1-\alpha)\beta(2-\beta))$

From these conditional probabilities, $P(y_j^{t+1}, y_i^t|v_k)$, it is straightforward to obtain $P(y_j^{t+1}|y_i^t, v_k)$ by normalization using equation 7.1. These are the optimal values with which to perform inference. Therefore, they form a benchmark from which to compare the values obtained by the online EM learning algorithm.

7.4 Aside: Biological Models of Motion Detection

Visual motion detection has perhaps been one of the most extensively studied areas in neuroscience. Early experiments on beetles by Hassenstein and Reichardt [31] led to the development of a model for motion detection known as the 'Hassenstein-Reichardt' model or 'Reichardt detector' which remains the most widely accepted model of invertebrate vision [30][61]. The basis of a Reichardt detector is a delay-and-correlate computation where the brightness signal measured by one photoreceptor is delayed by a low-pass filter and correlated by multiplication with the instantaneous signal of a neighbouring photoreceptor. A directionally selective response is obtained by subtracting the output signals from two delay-and-correlate subunits arranged in a mirror-symmetrical fashion (see figure 7.3). Experiments have verified many of the counter-intuitive predictions of the purely mathematical treatment of the model by Reichardt [59], such as the response being tuned to an optimal velocity rather than increasing continuously with velocity.

Direction selective neurons that respond to image motion in a preferred direction are found in many species from insects to mammals. The mechanism of directional selectivity remains controversial, but in all cases a role for dendritic computation in directional selectivity has been proposed. Primarily, these proposals involve non-linear interactions and delays on the dendritic branches. Dendritic involvement in directional selectivity has been shown experimentally in flys by Single and Borst [61], and supports Reichardt-type models of motion detection.

Interestingly, the motion detecting subgraph of section 7.2 is a binary version of the delay-and-correlate subunit of the Reichardt detector. For example, to infer the velocity v_k using the SBP algorithm messages are passed from observation nodes y_i and y_j towards v_k. However, in the

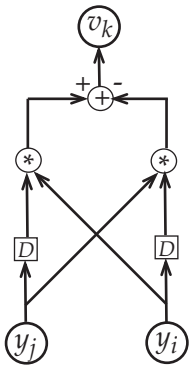

Figure 7.3: This figure shows the basic Reichardt detector. A multiplier unit is denoted by a circle with a $*$ and may also be considered a coincidence detector. Each coincidence detector receives a signal from two sensors, y_i and y_j, where one of the signals is delayed by D. The output of the two coincidence detectors is summed together. The coincidence detector for the preferred direction has a positive effect and the null direction has a negative effect on the motion neuron v_k.

graphical model observations are from different time intervals and therefore the messages (spikes) from y_j^{t+1} must be delayed such that the spikes from y_i and y_j, s_i^t and s_j^t, arrive at the factor at the same time t. Therefore, by combining two mirror symmetric graphical models like those of figure 7.2, that is, $P(y_j^{t+1}|y_i^t, v_k)$ and $P(y_i^{t+1}|y_j^t, v_k)$, we obtain a graphical model similar to the full Reichardt detector.

7.5 Belief Network for Motion Detection

The belief network for inferring the velocity of motion consists of multiple instantiations of the subgraph of figure 7.2. Different belief networks are constructed for each variation of the motion detection task where the number of image pixels and the number of velocities to detect may vary. A belief network consists of N_{obs} observation nodes, one for each image

pixel, together with N_{vel} velocity nodes, one for each possible velocity of motion specified for the task. Factors of the belief network connect pairs of observation nodes to velocity nodes. A pair of observation nodes observe two spatially and temporally separate pixels, so they may detect a particular image velocity. Every pair of observation nodes is connected to every velocity node. That is, each velocity node is wired up symmetrically. For example, consider a two velocity problem with motion leftwards or rightwards at 1 pixel per timestep. The belief network for this two velocity problem is shown in figure 7.4. Velocity node v_k has factors that detect both leftwards and rightwards motion. Thus, velocity node v_k could represents either leftwards or rightwards motion depending on the values of the conditional probabilities $P(y_j^{t+1}|y_i^t, v_k)$ and $P(y_i^{t+1}|y_j^t, v_k)$. For velocity node v_k to represent leftwards motion the conditional probability $P(y_i^{t+1}|y_j^t, v_k)$ would be positively correlated and for velocity node v_k to represent rightwards motion the conditional probability $P(y_j^{t+1}|y_i^t, v_k)$ would be positively correlated. The factor values are positively correlated if when v_k is 1 the probability that $y_j^{t+1} = y_i^t$ is high and the probability that $y_j^{t+1} \neq y_i^t$ is low. That is, the preferred motion a velocity node is determined by the values in its conditional probability tables, rather than by the connectivity of the network.

7.5.1 Dynamical model

In the generative model of figure 7.1 there is no dynamical model of how velocities change over time. That is, velocities are inferred from the immediate evidence available (from current and previous timesteps) to the velocity nodes, ignoring the velocity inferred from previous evidence. In a more elaborate model, one could include a transition model of how velocities change over time. Figure 7.5 (a) shows such a model with conditional

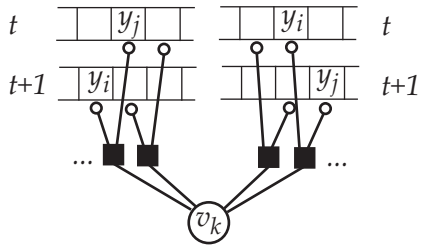

Figure 7.4: This figure shows factors connecting one velocity node to sensors for a belief network that detects two velocities; leftwards or rightwards at 1 pixel per timestep. The factors $P(y_i^{t+1}|y_j^t, v_k)$ are connected to sensors such that they can detect leftwards motion. The factors $P(y_j^{t+1}|y_i^t, v_k)$ are connected to sensors such that they can detect rightwards motion.

probabilities

$$P(v_l^{t+1}|v_k^t, a_i)$$

where a_i is an acceleration node. Figure 7.5 (b) shows a simplified model that does not include an acceleration model, with conditional probabilities

$$P(v_l^{t+1}|v_k^t)$$

This could be interpreted as having a single acceleration node representing an acceleration of zero that is always true.

A population code is used for velocity, such that a population of binary velocity nodes represents a multi-valued velocity variable, where each binary node represents a value of the velocity. Thus, the full velocity transition model would necessitate $P(v_k^{t+1}|\mathbf{v}^t)$ where $\mathbf{v}^t$ is the set of binary velocity nodes. As the number of velocity nodes increases the number of givens in the conditional probability $P(v_k^{t+1}|\mathbf{v}^t)$ increases and therefore the computation performed at the factors would increase exponentially. However, the velocity transition model used in these simulation makes the approximation

$$P(v_k^{t+1}|\mathbf{v}^t) \approx P(v_k^{t+1}|v_k^t)$$

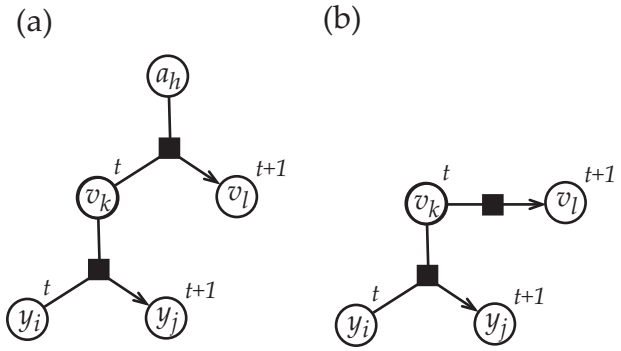

Figure 7.5: This figure shows a graphical model for motion detection with an acceleration model (a) and a velocity transition model (b). Variable node a_h represents the acceleration of an object. In (a) the factor $P(v_l^{t+1}|v_k^t, a_h)$ is the conditional probability of an objects new velocity given its current velocity and acceleration. In (b) the factor $P(v_l^{t+1}|v_k^t)$ is conditional probability of an objects new velocity given its current velocity.

In this model, a velocity node's degree of belief depends on its previous belief, but not on the beliefs at other velocity nodes. This velocity transition model is such that the velocity changes with probability ν at any given time. That is,

$P(v_k^{t+1}|v_k^t)$

	v_k^t=0	v_k^t=1
v_k^{t+1}=0	$1-\nu$	ν
v_k^{t+1}=1	ν	$1-\nu$

7.5.2 Motion inference

Inference is performed using the stochastic inference methods of chapter 5. This inference algorithm can be viewed as the spiky belief propagation algorithm with no temporal smoothing applied to the messages and the online Gibbs sampling algorithm. In this approach, a factor receives

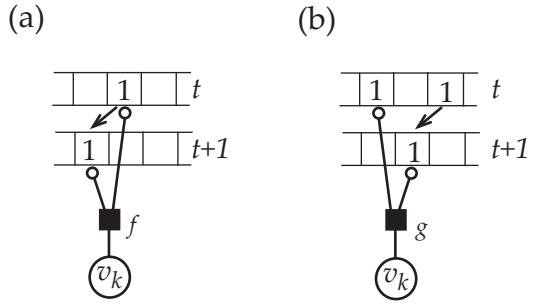

Figure 7.6: This figure shows observations that in (a) provide correct evidence of leftwards motion and in (b) provide incorrect evidence of rightwards motion.

messages (spikes) from a pair of observation nodes where one of the messages is delayed by one timestep. Factors then compute and send to the velocity node the ratio of the conditional probabilities for the values of the observations. That is,

$$\frac{P(y_j^t|y_i^{t-1}, v_k{=}1)}{P(y_j^t|y_i^{t-1}, v_k{=}0)}$$

A velocity node receives these likelihood ratio messages from its factors and multiplies them together with the prior $P(v_k^t)$. By normalizing this product the velocity node computes its degree of belief that the actual velocity is its preferred velocity given the observations. That is, the velocity node v_k computes the posterior probability

$$P(v_k^t{=}1|\mathbf{y}^t, \mathbf{y}^{t-1})$$

where $\mathbf{y}^t$ is the current observations and $\mathbf{y}^{t-1})$ is the previous observations. With a dynamical model, the velocity transition model of the previous section, the prior is the posterior from the previous time step. Thus, the prior is $P(v_k^t|v_k^{t-1})$ instead of $P(v_k^t)$.

Figure 7.6 shows why inference based on these observations is ambiguous. In figure 7.6 the true motion is leftwards at 1 pixel per timestep and factor f (figure 7.6 (a)), which is configured to detect leftwards motion, correctly provides evidence to v_k of leftwards motion. However, factor

g (figure 7.6 (b)), which is configured to detect rightwards motion, incorrectly provides evidence to v_k of rightwards motion even though the true motion is leftwards. Thus, the evidence provided by each motion detecting factor is ambiguous. Furthermore, as observations are corrupted by noise evidence is even less reliable. To overcome this ambiguity and noise in the observations, evidence from many sensors must be combined.

7.6 Results for Motion Inference

This section shows the results of running the stochastic inference algorithm on the problem of inferring motion from 1D images. Motion inference is performed on several variations of the motion detection task with a varying number of velocities to infer. In the two velocity task, the velocities are leftwards and rightwards at 1 pixel per timestep. The three velocity task adds a 'no motion' velocity (V=0). In the four velocity task, the velocities are leftwards and rightwards at 1 pixel per timestep and leftwards and rightwards at 2 pixels per timestep.

The next section describes the error measure used to assess the performance of inference using the spiky belief propagation algorithm. Then the results are given for performing inference on belief networks with the optimal conditional probability table values described in section 7.2.

7.6.1 Motion inference error measure

To measure how well the stochastic inference algorithm performs at inferring motion the error between the actual and inferred velocity is calculated. For each velocity node v_k the error e_k is the square of the difference between the inferred degree of belief $b(v_k{=}1)$ that the velocity is V_k and an indicator function I_k that the velocity is V_k, that is

$$e_k = [I_k(V_{\text{actual}}{=}V_k) - b(v_k{=}1)]^2$$

where $I_k(V_{\text{actual}}{=}V_k)$ is 1 if V_k is V_{actual} and zero otherwise. The root mean squared error is the square root of the mean of the individual velocity node errors over a number of timesteps T, that is,

$$\text{r.m.s.e} = \sqrt{\frac{\sum_{t=1}^{T}\sum_{k}^{N_{\text{vel}}} e_k}{T.N_{\text{vel}}}}$$

where N_{vel} is the number of velocity nodes. The error is zero when the correct velocity is inferred.

7.6.2 Motion inference with no dynamical model

Motion inference was performed on a belief network with no dynamical model. This inference is based purely on the evidence immediately available (from current and previous timesteps) to the network. Figure 7.7 shows instantaneous belief in each of the two velocities inferred over time based on evidence from 20 observation nodes which were corrupted with noise with probability $\beta = 0.1$. The underlying image pixels values are 1 or 0 with equal probability, that is $\alpha = 0.5$. Similarly figure 7.8 shows instantaneous belief in each of the three velocities inferred over time based on evidence from 20 observation nodes and with the same parameter values.

In figures 7.7 and 7.8, the inferred velocity can be seen to flip between the different values. Furthermore, in these figures it can be seen that the correct velocity is inferred the majority of the time, as the error is zero most of the time. However, there is the occasional spike in the error where the velocity was not correctly inferred. The observations are noisy and the evidence is often ambiguous which explains the occasional error.

In the next scenario, simulations are run for belief networks of varying sizes and for varying levels of noise, on the two, three and four velocity problems. Figure 7.9 shows the root mean squared error of the inferred beliefs for belief networks where the number of observation nodes is varied

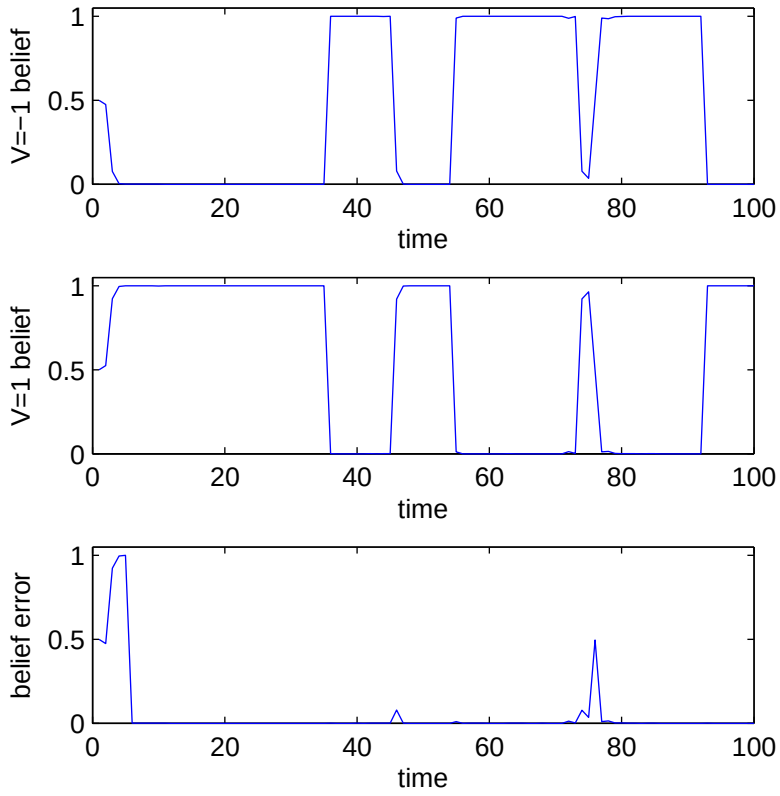

Figure 7.7: This figure shows the motion inference with two hidden velocity nodes over 100 time steps. The top plot shows the degree of belief in leftwards motion. The middle plot shows the degree of belief in rightwards motion. The bottom plot shows the error between the inferred belief in leftwards and rightwards velocity and the actual velocity.

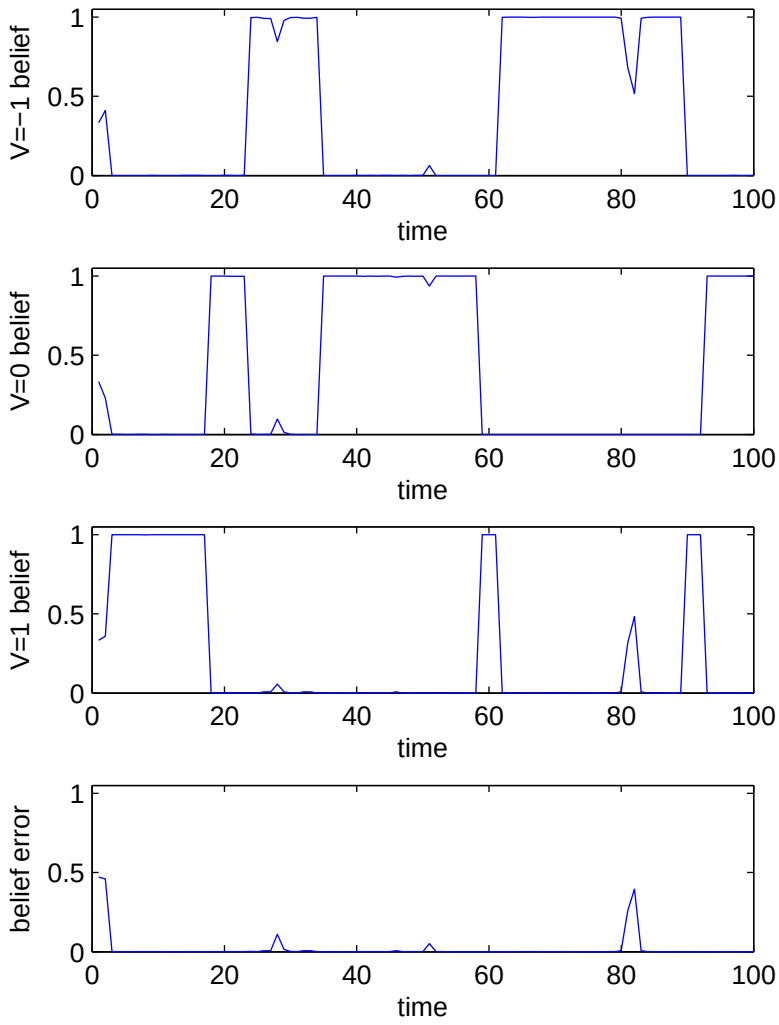

Figure 7.8: This figure shows the motion inference with three hidden velocity nodes over 100 time steps. The top three plots show the degree of belief in each of the three velocities. The bottom plot shows the error between the inferred and true velocity.

from 2 to 30 and the noise parameter β is varied from 0 to 1. Each point on a surface is mean of the error for 100 runs of 1000 timesteps.

The results shown in figure 7.9 shows three things. First, as the number of observation nodes increases the accuracy of the belief network improves. Second, the accuracy of the belief network degrades as the noise parameter β increases, as one would expect. Thus, the more evidence the belief network receives for a particular velocity of motion the more accurate the inference of that velocity of motion. Third, the error gets worse as the number of velocity nodes increases, as the problem becomes more difficult.

7.6.3 Motion inference with a dynamical model

Motion inference was performed on a belief network with the dynamical model for velocity transitions (see section 7.5.1). In this case inference is based not only on the immediate evidence but also on the inferred velocity based on past evidence. The simulations of the previous section were repeated for the three and four velocity problems with a velocity transition model.

The results shown in figure 7.10 show that as before the error decreases as the number of observation nodes increases and as the noise decreases. However in this case, the decrease in error happens slightly more quickly, because the inference is based on more evidence than without the dynamical model.

7.7 Results of Unsupervised Learning

This section shows the results of performing online unsupervised learning with the motion detection belief networks of section 7.5. Learning is unsupervised, that is, there is no teacher signal or correct answer given to the

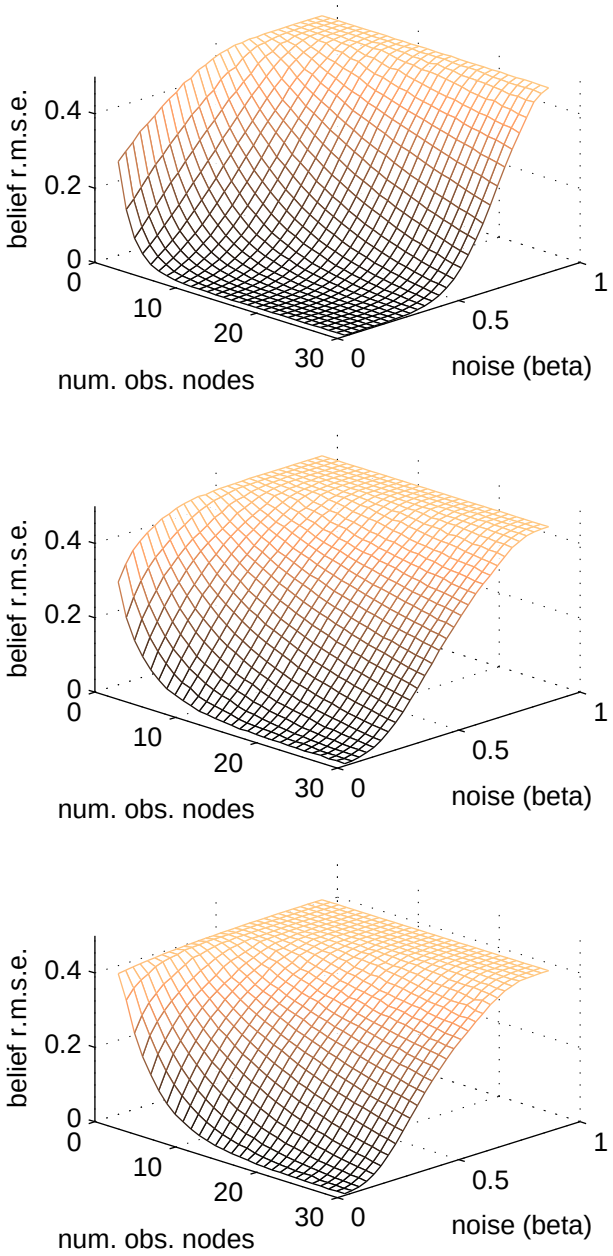

Figure 7.9: This figure shows the belief error (r.m.s.e) for simulations with two (top), three (middle) and four (bottom) velocity nodes using the optimal CPT values.

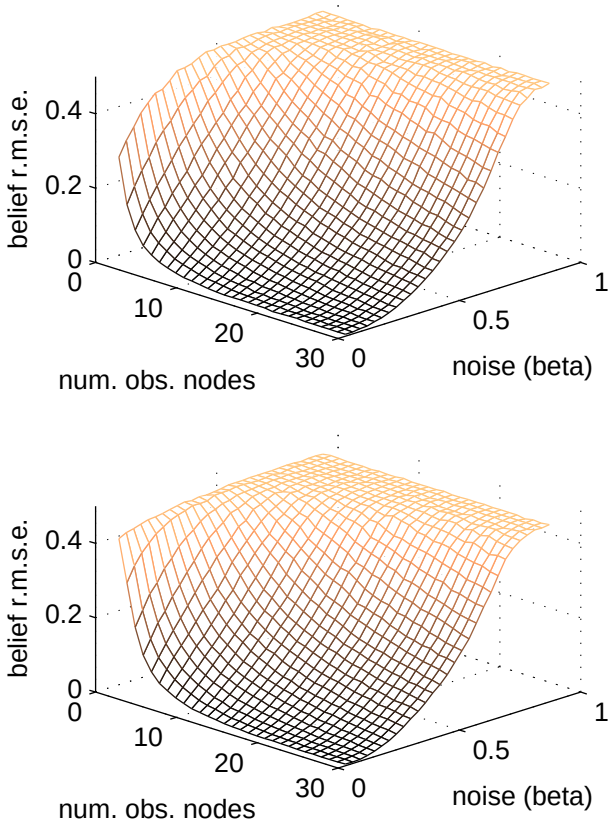

Figure 7.10: This figure shows the belief error (r.m.s.e) for simulations with three (top) and four (bottom) velocity nodes using the optimal CPT values and a dynamical model of velocity transitions.

networks. A belief network learns solely from the observed data where its goal is to learn the parameters of a generative model of the data. These parameters are the values of conditional probability tables, $P(y_j^{t+1}|y_i^t, v_k)$, of the belief network based on the generative model of section 7.2.

Learning is also online, that is, learning is performed in a single forward pass of the data. Two variations of the learning rules are compared. These learning rules are online versions of the EM algorithm and were described in section 6.2. The two learning rules are referred to as the accumulated-counts (AC) learning rule and as the decayed-counts (DC) learning rule depending on the method by which they update the counts over time.

A uniform prior is used for the initial conditions, that is, the pseudo-counts are all initially set at one, as if each event was observed once. Thus, the conditional probabilities are initially all equally likely, with a probability of 0.5. Therefore, velocity nodes initially start with no preferred velocity. As learning proceeds the preferred velocity of each velocity nodes is determined by inspecting the values in their conditional probability tables. This is done only so that the error can be measured.

Learning was performed on a sequence of randomly generated images in a single forward pass where each simulation is run for a period of 4000 timesteps to allow the algorithm to settle on a set of conditional probabilities. Next the error in inferred beliefs is calculated over the following 1000 timesteps. The results shown are the mean of 100 simulation runs, as they were in the previous section.

7.7.1 Unsupervised learning with no dynamical model

For belief networks with no dynamical model, the inference error resulting from inference with conditional probabilities learnt with the online unsupervised learning rules are shown in figures 7.11 and 7.12. These figures

show the inference error for the accumulated-counts learning rule and the decayed-counts learning rule on belief networks of various sizes and levels of noise. For the decayed-counts learning rule, a learning rate of 0.0025 is used.

By comparing figures 7.11 and 7.12 it can be seen that the decayed-counts learning rule performs better than the accumulated-counts learning rule on all three versions of the velocity detection problem. For both learning rules, the performance gets worse as the number of velocity nodes increases, as it does for the optimal parameters. For the two velocity task, the inference error is very similar for inference using the learnt and optimal conditional probabilities. This can be seen by comparing the inference error in figures 7.11 and 7.12 with figure 7.9 based on the optimal conditional probabilities. However, for the three and four velocity tasks, the inference error using the learnt conditional probabilities is considerably worse than the inference error using the optimal conditional probabilities.

The high inference error is because inference was performed using sub-optimal conditional probabilities. Learning does not always converge on the optimal conditional probabilities, but sometimes gets stuck in local maxima of the likelihood. This is a well known problem associated with Maximum Likelihood learning.

7.7.2 Convergence properties of learning

In this section, the convergence properties of the two learning rules are shown. To show the convergence properties the root mean squared error (r.m.s.e.) between the learnt conditional probabilities and optimal conditional probabilities was plotted over 2000 timesteps for multiple runs. These simulation runs all had 30 observation nodes and a noise parameter of $\beta = 0.1$.

Figure 7.13 shows the error between the learnt and optimal conditional

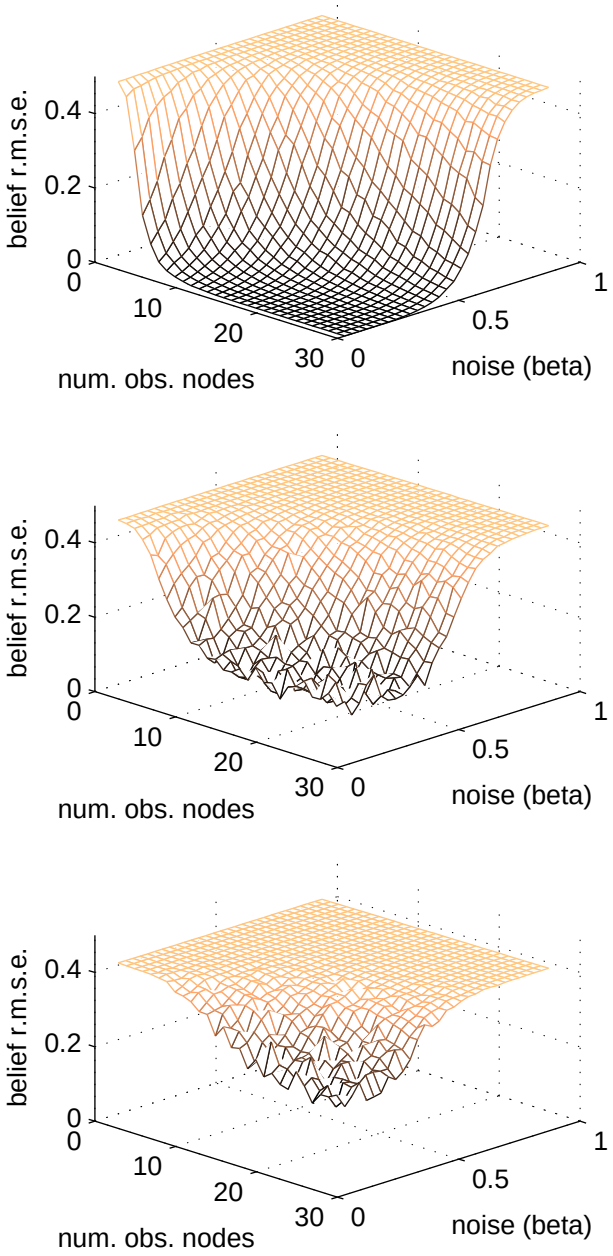

Figure 7.11: This figure shows the belief error (r.m.s.e) for accumulated-counts learning rule. The belief networks have varying numbers of observation nodes and varying levels of noise. These simulations have two (top), three (middle) and four (bottom) velocity nodes.

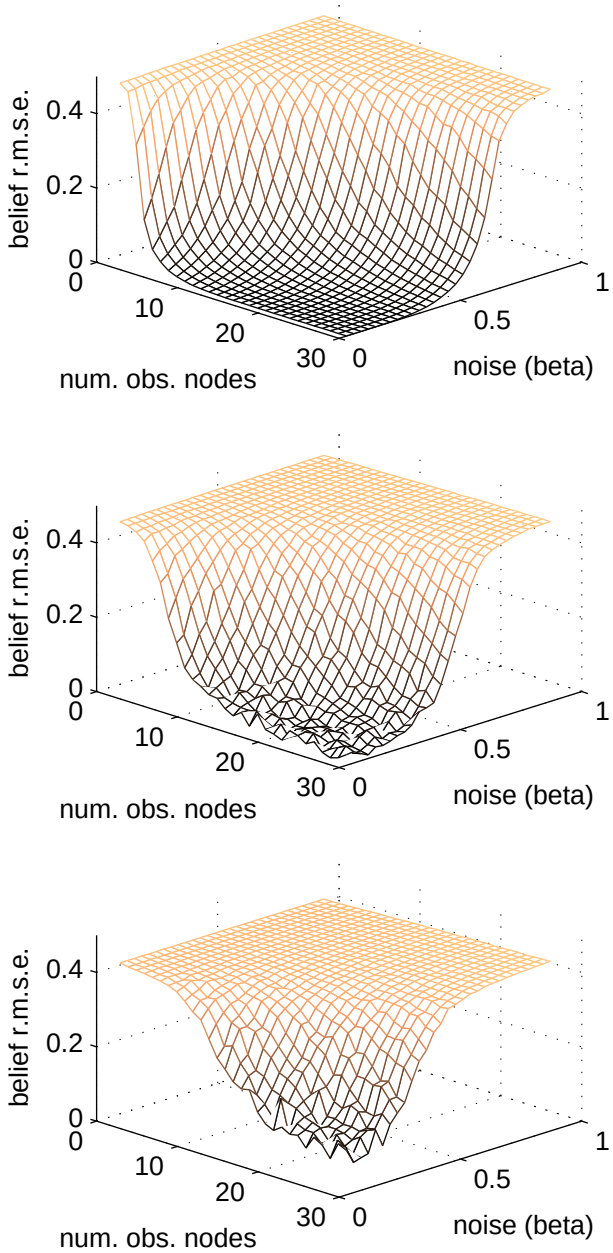

Figure 7.12: This figure shows the belief error (r.m.s.e) for decayed-counts learning rule. The belief networks have varying numbers of observation nodes and varying levels of noise. These simulations have two (top), three (middle) and four (bottom) velocity nodes.

probability values for the accumulated-counts learning rule. For a belief network with two velocities, the learnt conditional probabilities converge to near-optimal values which nevertheless produce a low belief error. The reason the accumulated-counts learning rule does not converge to the optimal parameters is because of the counts early in the run are based on incorrect parameters. For belief networks with three and four velocities the learnt conditional probabilities do not always converge on near-optimal values. Rather, there are local maxima in which the learning algorithm gets stuck. For instance, the cluster of runs with an error of around 0.2.

Figure 7.14 shows the error between the learnt and optimal conditional probability values for the decayed-counts learning rule with a learning-rate of 0.0025. The error between learnt and optimal conditional probabilities decreases much more slowly with the decayed-counts learning rule than with the accumulated-counts learning rule. That is, the speed of learning is considerably slower. This speed is of course adjustable with the learning-rate parameter. The decayed-counts learning rule has an element of 'forgetting' where the learning-rate specifies the amount of forgetting. That is, earlier counts become insignificant as time proceeds and become overwritten by more recent counts. As counts early in the run are based on the incorrect conditional probabilities, the decayed-counts learning rule has better convergence properties than the accumulated-counts learning rule, because its forgets the earlier, less reliable counts. Thus, the mean inference error is less with the decayed-counts learning rule, and it is less prone to getting stuck in local maxima than the accumulated-counts learning rule.

7.7.3 Decayed-counts learning rate

In order to find a reasonable learning-rate, simulation runs were performed over a range of rates. Figure 7.15 shows the error in the learnt conditional

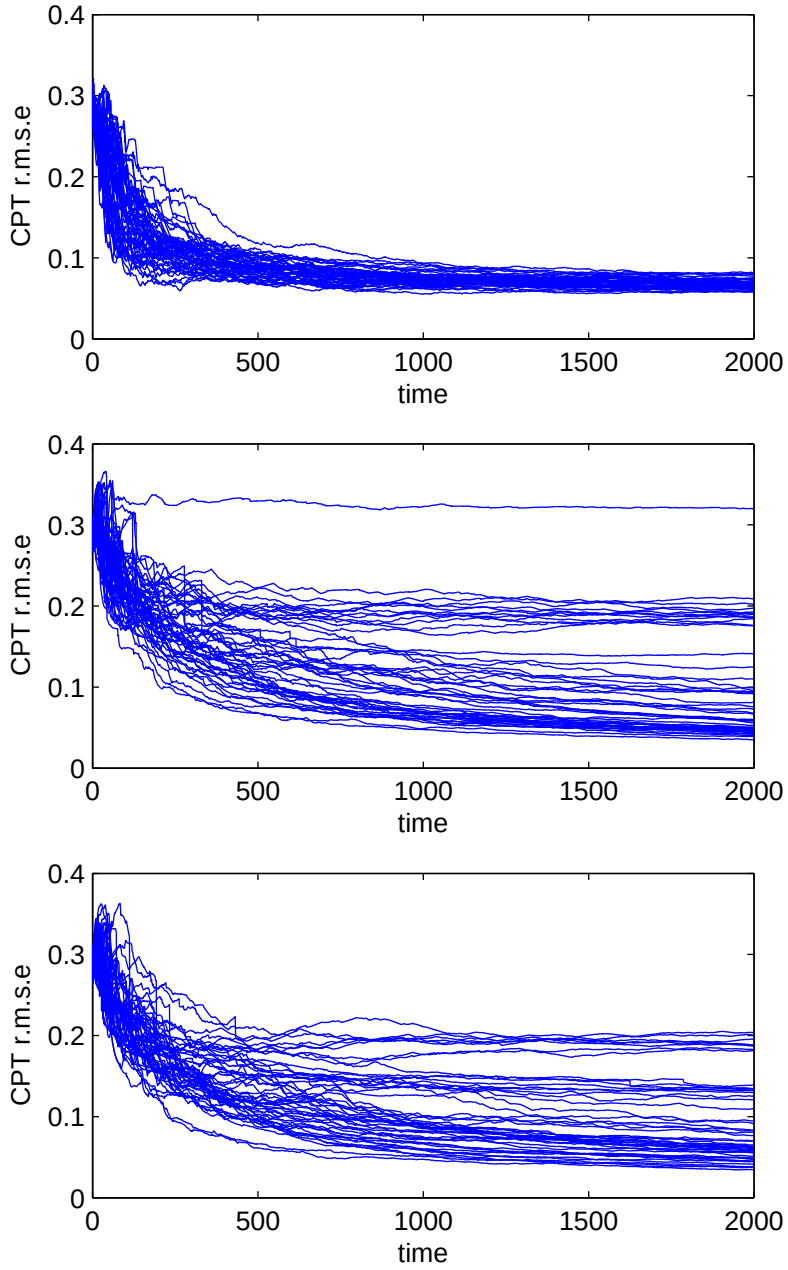

Figure 7.13: This figure shows the convergence properties of the accumulated-counts learning rule. The error (r.m.s.e) is shown between learnt and optimal parameters over time for 50 simulation runs. These simulations have two (top), three (middle) and four (bottom) velocity nodes.

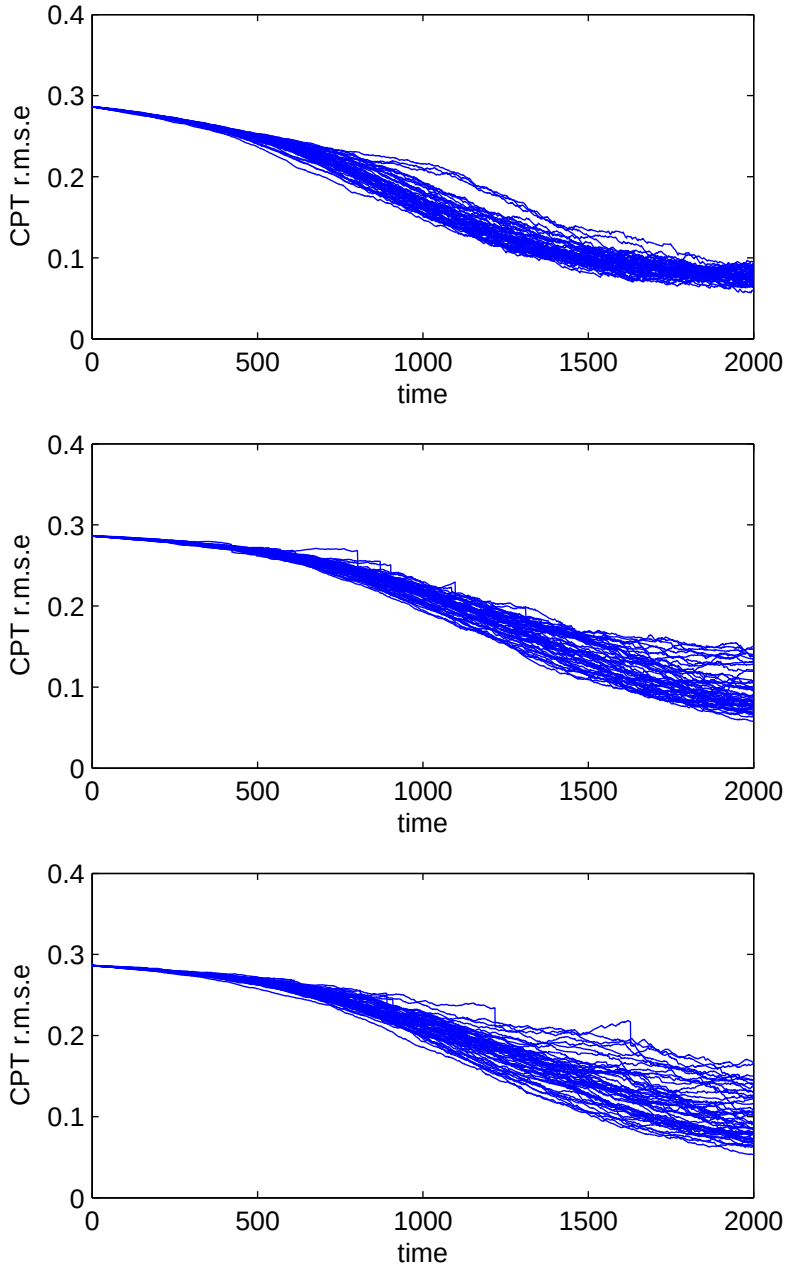

Figure 7.14: This figure shows the convergence properties of decayed-counts learning rule with learning rate 0.0025. The error (r.m.s.e) is shown between learnt and optimal parameters over time for 50 simulation runs. These simulations have two (top), three (middle) and four (bottom) velocity nodes.

probabilities decreasing rapidly as learning-rate increases before gradually increasing again. Thus, a learning-rate of 0.0025 was chosen for simulations involving the decayed-counts learning rule as it is roughly the lowest point on each plot of figure 7.15. This learning-rate value depends on the length of time learning is given. Obviously, a short learning time does not give a slow learning rate sufficient time to learn. A slower learning rate results in a closer approximation to the optimal conditional probabilities because the temporal averaging smoothes over more data.

7.7.4 Unsupervised learning with a dynamical model

For belief networks with a dynamical model, the simulations of section 7.7.1 were repeated on the three and four velocity motion detection tasks. The inference error resulting from inference with conditional probabilities learnt with online unsupervised learning are shown in figures 7.16 and 7.17. These simulations that include a velocity transition model, show considerable improvement over simulations without a velocity transition model, for both learning rules. This can be seen by comparing figures 7.11 and 7.16 for the accumulated-counts learning rule and figures 7.12 and 7.17 for the decayed-counts learning rule. As before, the decayed-counts learning rule performs better than the accumulated-counts learning rule.

With a dynamical model the results are much closer to the inference error for the optimal conditional probabilities, particularly with the decayed-count learning rule.

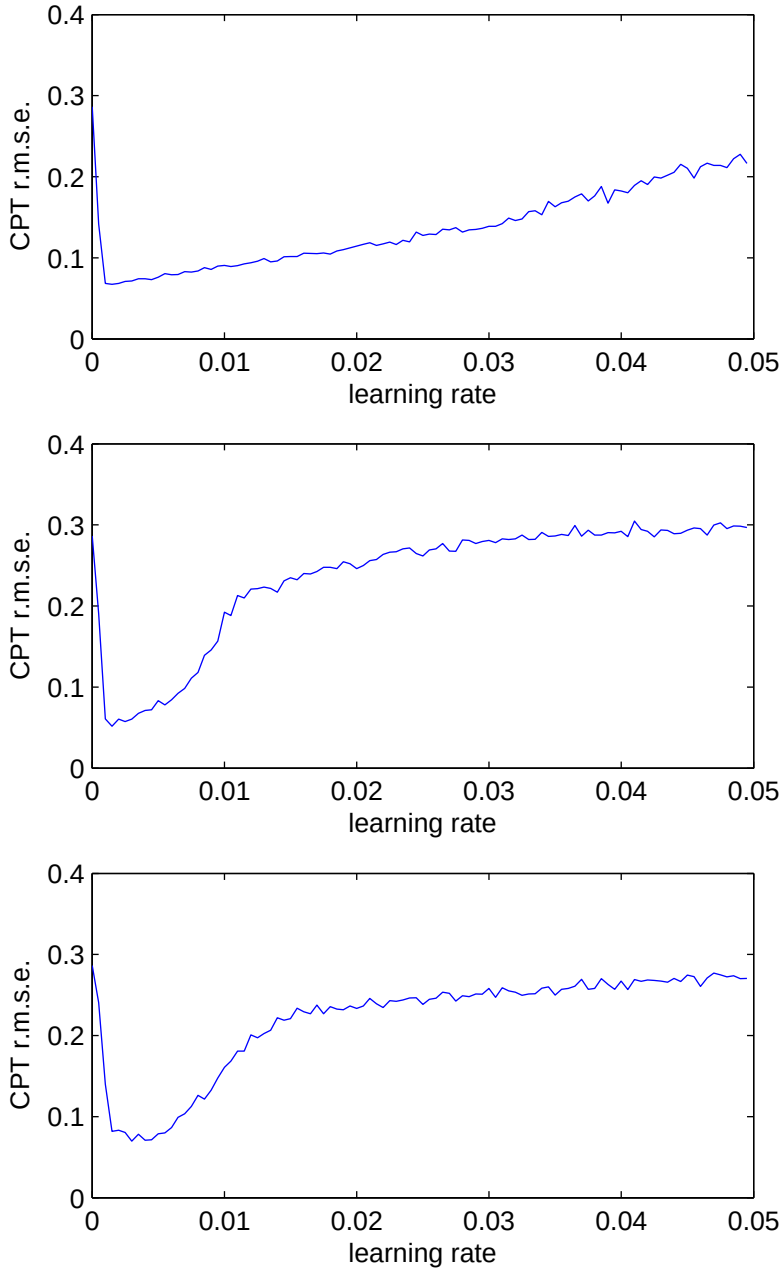

Figure 7.15: This figure shows CPT parameter error verses learning rate. The CPT parameter error is the average over 100 runs with noise of $\beta = 0.1$, 30 observation nodes. These simulations have two (top), three (middle) and four (bottom) velocity nodes.

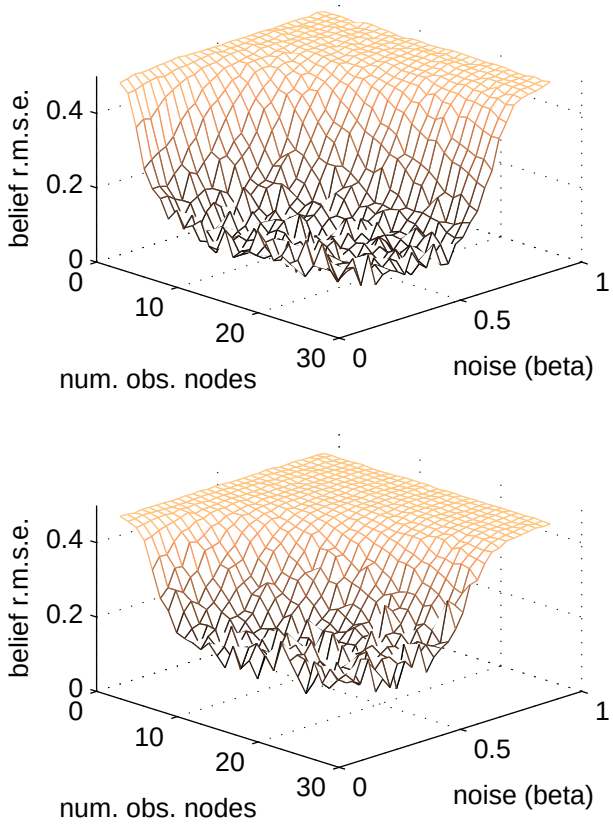

Figure 7.16: This figure shows the belief error (r.m.s.e) for accumulated-counts learning rule. The belief networks of these simulations include a transition model for velocities. These simulations have two (top), three (middle) and four (bottom) velocity nodes.

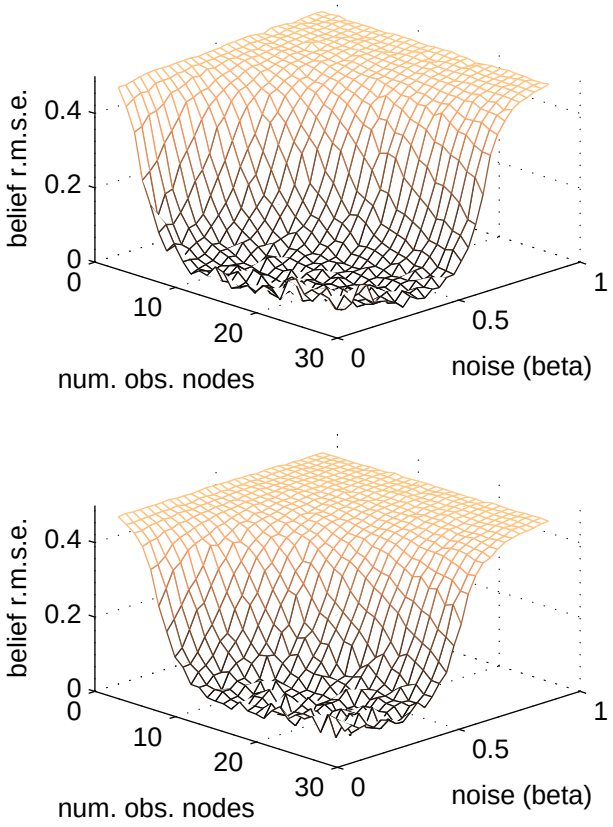

Figure 7.17: This figure shows the belief error (r.m.s.e) for decayed-counts learning rule. The belief networks of these simulations include a transition model for velocities. These simulations have two (top), three (middle) and four (bottom) velocity nodes.

Chapter 8

Conclusion

As experiments in psychophysics suggest the brain is representing and propagating uncertainty, the probabilistic framework has recently been suggested for neural processing [58][15][68][43]. Similarly, in this thesis several algorithms from machine learning were considered that perform reasoning with uncertainty and learning from experience. Ways in which these algorithms might be implemented in neural circuits were explored, and demonstrated on a filtering task. As the brain is extremely complex and far from being fully understood, these proposed algorithms make many assumptions and simplifications to actual information processing in the nervous system. However, the computations performed by these algorithms attempt capture the basic behaviour of real neurons.

Message-passing algorithms perform a global computation by sending messages between the variable nodes of a graph and by performing only local computations at those nodes. A contribution of this thesis was a message-passing algorithm, named spiky belief propagation, for performing Bayesian inference on directed graphical models. Furthermore, the connection between this algorithm and an online form of Gibbs sampling was shown. The appeal of online Gibbs sampling and spiky belief propagation is that they send particularly simple messages, and that the hidden

variables can be inferred online. The variable nodes of these belief networks are neurons, and the messages are spikes.

In this approach, the firing rate of a neuron represents the degree of certainty in an underlying binary hypothesis. A disadvantage of this is that the graphical models are then restricted to be composed of binary variable nodes. To overcome this limitation a population of nodes was introduced to capture multi-valued variables or approximate continuous ones. This has the advantage that multi-modal distributions can be represented. This is an improvement over standard techniques such as Kalman filtering [38] which can not represent multi-modal distributions.

A further contribution was to consider graphs composed of higher-order factors which produce more compact models than would be possible with graphs composed of pairwise factors. For instance, the proposed generative model for the motion detection task was composed of higher-order factors. However, higher-order factors may be applied much more generally than computing derivatives and could be used to combine information from multiple sources, such as from different sensory pathways or feedback from higher levels. Whether the higher-order factors described here could be implemented by dendritic processing is unknown. What is known is that neurons are complicated and have intricate dendritic trees with both active and passive dendritic processing. There are nonlinear mechanisms involved in the dendritic tree that have been suggested to play an important part in the overall computation performed by neurons [48][49][44]. For instance, there is evidence for dendritic processing playing a role in early vision and in auditory systems [2].

Learning was performed with a variation to the EM algorithm. This learning algorithm learns the conditional probabilities of a generative model based on counts produced by the stochastic inference algorithm. As some variables are unobserved this learning is unsupervised. Unsupervised learning is a difficult task made more difficult by the demand on it to be

performed online. Online learning is performed as the data is observed at the rate at which it arrives. The feasibility of this online learning algorithm was explored on a motion detection task. Learning on this task was difficult due to the large number of conditional probabilities to be learnt. A common problem with maximum likelihood approaches is convergence to local maxima and this problem was encountered on the motion detection task. Further work is required to determine if a 'better' prior could overcome this convergence problem.

There are several interesting directions for possible future work. An interesting question is whether this approach can be scaled up to deeper belief networks. For the motion detection task this might involve adding another layer of variables nodes representing acceleration which would be inferred from the estimated velocity. Also missing from this model is feedback. Incorporating feedback would involve passing messages both backwards towards input nodes as well as in the forward direction [43]. For instance, velocity could be inferred from both the lower-level sensory evidence and from the higher-level acceleration nodes. In this case, spikes would be passed around loops in the graph and a careful balance between excitation and inhibition is required to prevent runaway activity [15]. Further investigation would also be required to determine if learning is possible in the case where messages are passed around loops.

Another question is whether the structure of the belief networks can be learnt from data. The network structure and initial connection weights are a prior in the Bayesian approach. The belief networks applied to the motion detection task had a fixed network structure that was specified a priori. For many other problems, however the network structure may not be so obvious, a priori. Learning network structure is a difficult problem, as the associated search space is exponentially large. The typical approach is to define a 'score' that describes the fitness of the structure, such as Bayesian scoring or minimum description length (MDL), and search for

the structure with the highest score [20][21]. Finding the structure that maximizes the score is intractable, and so approximations are required [26]. Learning network structure is still an open problem. In the context of stochastic approximation, further work is needed to investigate if these methods can be used to learn network structure.

There are several aspects in which the simulations of this thesis could be more biologically plausible and more applicable to real world problems. The first is that motion detection be performed on natural images, rather than the random dots. The second is to use deterministic spiking neurons, such as Integrate-and-Fire neurons or Spike Response Model neurons, rather than the stochastic (Poisson) spike generating neurons that are used. The third is to incorporate actions and 'close the loop', such that the belief network is an agent that exists in an environment with which it interacts. Thus, the agent would be able see the effect of its actions, and to learn from them.

The brain has remarkable abilities of learning and information processing. It is hoped that a better understanding of the computational principles underlying these abilities may lead to biologically-inspired solutions to problems in computer-vision, robotics, and artificial intelligence. How the brain handles a rapidly changing world based on unreliable, noisy sensory information is an interesting question. The probabilistic framework was proposed for representing and propagating uncertainty using networks of stochastic neurons for neural computation. However, many aspects of this hypothesis remain unproven. Much remains to be discovered concerning how information processing can be performed by a distributed network of neurons. It remains a major challenge for neuroscience to uncover how Bayesian estimation might be implemented by the brain.

Bibliography

[1] ADELSON, E. H., AND BERGEN, J. R. Spatiotemporal energy models for the perception of motion. *J. Opt. Soc. Am. A 2*, 2 (1985), 284–299.

[2] AGMON-SNIR, H., CARR, C. E., AND RINZEL, J. The role of dendrites in auditory conincidence detection. *Nature 393* (1998), 207–208.

[3] ANDRIEU, C., AND DOUCET, A. Online expectation-maximization type algorithms for parameter estimation in general state space models. In *Proc. IEEE ICASSP* (2003).

[4] BARBER, D. Probabilistic modelling and reasoning variational learning and EM. http://anc.ed.ac.uk/∼dbarber/pmr/pmr.html, 2003.

[5] BAUM, L. E., AND PETRIE, T. Statistical inference for probabilistic functions of finite-state Markov chains. *Ann. Math. Stat. 37* (1966), 1559–1563.

[6] BI, G. Q., AND POO, M. M. Synaptic modifications in cultured hippocampal neurons: Dependence on spike timing, synaptic strength, and postsynaptic cell type. *Journal of Neuroscience 18* (1998), 10464–10472.

[7] BISHOP, C. M. Variational learning in graphical models and neural networks. In *Proceedings 8th International Conference on Artificial Neural Networks, ICANN'98* (1998), Springer, pp. 13–22.

[8] BLAKEMORE, M. R., AND SNOWDEN, R. J. The effect of contrast upon perceived speed: a general phenomenon? *Perception 28* (1999), 33–48.

[9] BRADDICK, O. J. A short-range process in apparent motion. *Vision Research 14* (1974), 519–527.

[10] BRADDICK, O. J. Low-level and high-level processes in apparent motion. *Philosophical Transactions of the Royal Society of London B 290* (1980), 137–151.

[11] CELEUX, G., AND DIEBOLT, J. The SEM algorithm: A probabilistic teacher algorithm derived from the EM algorithm for the mixture problem. *Computation Statistics Quarterly 2* (1985), 73–82.

[12] CELEUX, G., AND DIEBOLT, J. A stochastic approximation type EM algorithm for the mixture problem. *Stochastics and Stochastics Report 41* (1992), 127–146.

[13] COVER, T. M., AND THOMAS, J. A. *Elements of Information Theory*. Wiley, New York, 1991.

[14] DEMPSTER, A. P., LAIRD, N. M., AND RUBIN, D. B. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society B 39* (1977), 1–38.

[15] DENEVE, S. Bayesian inference in spiking neurons. *NIPS-17* (2004).

[16] DEYLON, B., LAVIELLE, M., AND MOULINES, E. Convergence of a stochastic approximation version of the EM algorithm. *The Annals of Statistics 27* (1999), 94–128.

[17] ERNST, M. O., AND BANKS, M. S. Humans integrate visual and haptic information in a statistically optimal fashion. *Nature 415* (2002), 429–433.

[18] FREAN, M. Online EM. personal communication, 2006.

[19] FREY, B. *Graphical Models for Machine Learning and Digital Communication*. MIT Press, 1998.

[20] FRIEDMAN, N. Learning belief networks in the presence of missing values and hidden variables. In *Proceedings of the Fourteenth International Conference on Machine Learning* (1997), Morgan Kaufmann, pp. 125–133.

[21] FRIEDMAN, N. The Bayesian structural EM algorithm. In *Proceedings of the Fourteenth International Conference on Uncertainty in Artificial Intelligence* (1998), Morgan Kaufmann, pp. 129–138.

[22] GALLAGER, R. G. *Low Density Parity Check Codes*. No. 21 in Research monograph series. MIT Press, 1963.

[23] GEMAN, S., AND GEMAN, D. Stochastic relaxation, gibbs distribution and the bayesian restoration of images. *IEEE Transactions on Pattern Analysis and Machine Intelligence 6* (1984), 721–741.

[24] GERSTNER, W. Time structure of the activity in neural network models. *Physical Review E 51* (1995), 738–758.

[25] GERSTNER, W., AND KISTLER, W. *Spiking Neuron Models*. Cambridge University Press, Cambridge, United Kingdom, 2002.

[26] GHAHRAMANI, M. J. B. Z. The variational Bayesian EM algorithm for incomplete data: with application to scoring graphical model structures. *Bayesian Statistics 7* (2003), 453–464.

[27] GHAHRAMANI, Z., AND BEAL, M. Propagation algorithms for variational Bayesian learning. *Advances in Neural Information Processing Systems* (2001), 507–513.

[28] GOLD, J. I., AND SHADLEN, M. N. Neural computations that underlie decisions about sensory stimuli. *Trends in Cognitive Sciences 5* (2001), 10–16.

[29] GOLDBERG, J., HOLTHOFF, K., AND YUSTE, R. a problem with Hebb and local spikes. *Trends in Neuroscience 25* (2002), 433–435.

[30] HAAG, J., DENK, W., AND BORST, A. Fly motion vision is based on reichardt detectors regardless of the signal-to-noise ratio. *PNAS 101*, 46 (2004), 16333–16338.

[31] HASSENSTEIN, B., AND REICHARDT, W. Z. Reihenfolgen- und vorzeichenauswertung bei der bewegungsperzeption des russelkafers chlorophanus. *Zeitschrift fur Naturforschung 11b* (1956), 513–524.

[32] HEBB, D. *The Organization of Behavior*. Wiley, New York, 1949.

[33] HINTON, G. E., AND SEJNOWSKI, T. J. Optimal perceptual inference. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition* (1983).

[34] HINTON, G. E., AND SEJNOWSKI, T. J. Learning and relearning in boltzmann machines. In *Rumelhart, D. E. and McClelland, J. L., editors, Parallel Distributed Processing: Explorations in the Microstructure of Cognition* (1986), MIT Press, pp. 282–317.

[35] HODGKIN, A. L., AND HUXLEY, A. F. A quantitative description of membrane current and application to conduction and excitation in a nerve. *Journal Physiol.* (1952), 500–544.

[36] HUBEL, D. H., AND WIESEL, T. N. Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. *Journal of Physiology 160* (1962), 106–154.

[37] JORDAN, M. *Learning in Graphical Models*. MIT Press, 1998.

[38] KALMAN, R. E. A new approach to linear filtering and prediction problems. *Transactions of the ASME - Journal of Basic Engineering 82* (1960), 35–45.

[39] KNILL, D., AND POUGET, A. The bayesian brain: the role of uncertainty in neural coding and computation. *Trends in Neurosciences 27*, 12 (December 2004), 2799–2802.

[40] KNILL, D. C., AND RICHARDS, W. *Perception as Bayesian Inference*. Cambridge University Press, 1996.

[41] KOCH, C., POGGIO, T., AND TORRE, V. Nonlinear interactions in a dendritic tree: Localization, timing, and role in information processing. *Proc. Natl. Acad. Sci. USA 80* (May 1983), 2799–2802.

[42] KORDING, K., AND WOLPERT, D. Bayesian integration in sensorimotor learning. *Nature 427* (2004), 244–277.

[43] LEE, T. S., AND MUMFORD, D. Heirarchical Bayesian inference in the visual cortex. *J. Opt. Soc. Am. A 20* (2003), 1434–1448.

[44] LONDON, M., AND HÄUSSER, M. Regulation of synaptic efficacy by coincidence of postsynaptic APs and EPSPs. *Science 275* (1997), 213–215.

[45] MARKRAM, H., LUBKE, J., FROTSCHER, M., AND SAKMANN, B. Regulation of synaptic efficacy by coincidence of postsynaptic APs and EPSPs. *Science 275* (1997), 213–215.

[46] MAUNSELL, J. H., AND VAN ESSEN, D. C. Functional properties of neurons in middle temporal visual area of the macaque monkey. i. selectivity for stimulus direction, speed, and orientation. *J. Neuroscience 49* (1985), 1127–1147.

[47] MCELIECE, R. J., MACKAY, D. J. C., AND CHENG, J. F. Turbo decoding as an instance of Pearl's 'belief propagation' algorithm. *IEEE Journal on Selected Areas in Communications 16*, 2 (February 1998), 140–152.

[48] MEL, B. W. Information processing in dendritic trees. *Neural Computation 6* (1994), 1031–1085.

[49] MEL, B. W. *Why have Dendrites? A computational perspective*. Oxford University Press, 1999, pp. 271–289.

[50] MURPHY, K., WEISS, Y., AND JORDAN, M. Loopy belief propagation for approximate inference: An empirical study. In *Uncertainty in AI* (1999), pp. 467–475.

[51] NEAL, R. Learning stochastic feedforward networks. *Technical Report CRG-TR-90-7, Dept. of Computer Science, University of Toronto* (1990).

[52] NEAL, R. Connectionist learning of belief networks. *Artificial Intelligence 56* (1992), 71–113.

[53] NEAL, R., AND HINTON, G. *A view of the EM algorithm that justifies incremental, sparse, and other varients*. MIT Press, 1998, pp. 355–368.

[54] PEARL, J. Evidential reasoning using stochastic simulation of causal models. *Artificial Intelligence 32* (1987), 245–257.

[55] PEARL, J. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, 1988.

[56] RABINER, L. R., AND JUANG, B. H. An introduction to hidden Markov models. *IEEE ASSP Magazine* (January 1986), 4–15.

[57] RAO, R. Bayesian computation in recurrent neural circuits. *Neural Computation*, 16 (2004).

[58] RAO, R. Hierarchical bayesian inference in networks of spiking neurons. *NIPS*, 17 (2005).

[59] REICHARDT, W. Z. Autocorrelation, a principle for the evaluation of sensory information by the central nervous system. *Principles of Sensory Communication* (1956), 303–317.

[60] RIEKE, F., WARLAND, D., DE RUYTER VAN STEVENINCK, R., AND BIALEK, W. *Spikes: Exploring the Neural Code*. MIT Press, 1997.

[61] S, S., AND BORST, A. Dendritic integration and its role in computing image velocity. *Science 281*, 5384 (1998), 1848–1850.

[62] SANGER, T. Neural population codes. *Current Opinion in Neurobiology*, 2 (2003), 238–249.

[63] SHADLEN, M. N., AND NEWSOME, W. T. Noise, neural codes and cortical organization. *Current Opinion in Neurobiology 4* (1994), 569–579.

[64] SOFTKY, W. R., AND KOCH, C. The highly irregular firing of cortical cells is inconsistent with temporal integration of random EPSPs. *Journal of Neuroscience 13* (1993), 775–785.

[65] SPRATLING, M. W., AND HAYES, G. M. Learning synaptic clusters for non-linear dendritic processing. *Neural Processing Letters 11*, 1 (2000), 17–27.

[66] WEI, G. C. G., AND TANNER, M. A. A Monte Carlo implementation of the EM algorithm and the poor man's data augmentation algorithms. *Journal of the American Statistical Association 85*, 411 (1990), 699–704.

[67] WEISS, Y., AND FLEET, D. J. *Velocity likelihoods in biological and machine vision*. MIT Press, 2002, pp. 77–96.

[68] ZEMEL, R. S., HUYS, Q. J. M., NATARAJAN, R., AND DAYAN, P. Probabilistic computation in spiking populations. *Advances in Neural Information Processing systems 17* (2005), 1609–1616.