Dynamic gradient estimation in machine learning

Thomas Flynn
Abstract

The optimization problems arising in machine learning form some of the most theoretically challenging and computationally demanding problems in numerical computing today. Due to the complexity of the models and the problem domains to which they are applied, approximation methods are required during optimization. This review focuses on optimization schemes involving dynamic gradient estimation. In these algorithms, gradient estimation runs in parallel with the parameter adaptation process. We survey a number of problems from machine learning that admit such approaches to optimization, including applications to deterministic and stochastic neural networks, and present these algorithms in a common framework of stochastic approximation.
## Contents

1 Introduction ................................................. 3  
1.1 Notation .................................................. 4  
2 Boltzmann machine ........................................... 5  
  2.1 Model ....................................................... 5  
  2.2 Optimization problem ................................. 7  
  2.3 Application: A Joint Model of Images and Text ...... 8  
  2.4 Optimization algorithm ............................... 9  
  2.5 Numerical experiment ................................. 12  
  2.6 Variants of the Boltzmann Machine .................. 13  
3 Stochastic approximation .................................... 14  
  3.1 Weak convergence to an ODE ......................... 15  
  3.2 Applying SA to the Boltzmann machine .............. 19  
  3.3 Application to online Bayesian learning .......... 19  
4 Sigmoid belief networks ..................................... 22  
  4.1 Model ....................................................... 22  
  4.2 Optimization problem ................................. 23  
  4.3 Optimization algorithm ............................... 23  
  4.4 Numerical experiment ................................. 25  
  4.5 Extensions ............................................... 26  
5 Attractor networks ........................................... 29  
  5.1 Model ....................................................... 29  
  5.2 Optimization problem ................................. 31  
  5.3 Optimization algorithm ............................... 33  
  5.4 Numerical experiment ................................. 35  
6 Chemical reaction networks ................................ 36  
  6.1 Model ....................................................... 37  
  6.2 Optimization problem ................................. 39  
  6.3 Optimization algorithm ............................... 40  
  6.4 Numerical experiment ................................. 40  
7 Conclusion .................................................... 41
1 Introduction

We will review several network-based models useful for applications in machine learning and other areas, touching upon a number of topics in each case. This includes how the networks operate, what they are used for, and issues related to optimization. The networks are diverse in terms of their dynamical features: some operate probabilistically while others are deterministic; some run in a continuous state space and some have discrete states. In terms of optimization, we discuss what is the typical optimization problem associated with the network, describe the sensitivity analysis procedure (that is, how to compute the necessary gradients), and also mention what are some theoretical challenges associated with the optimization. Typically, the parameters of the model relate either to the local behavior of a unit or how units interact. These parameters determine things like affinity for a certain state, or how one unit inhibits or excites another. For several of the problems the results of numerical experiments are presented.

Many of the models have the property that computing their derivatives is computationally difficult, and one must resort to (either deterministic or probabilistic) iterative procedures to do so. The resulting optimization algorithms then have a “two-timescale”
form, where derivative estimation and parameter update steps are parallel processes that must be calibrated correctly to achieve convergence. A schematic for this type of procedure is shown in Figure 1. For example, one situation where gradient estimation becomes non-trivial is when the optimization problem concerns the long-term behavior of a system. In this case, the sensitivity analysis procedure must discover the way long-term behavior is affected by changes to local parameters, but typically one only has a description of how the network evolves over the short term. A framework to analyze these multiple time-scale stochastic adaptive algorithms is provided by the theory of stochastic approximation, another topic which we review below.

The remainder of this survey is organized as follows. In Section 2 we consider the Boltzmann machine, a discrete time, discrete state space, stochastic neural network. In Section 3 we review the theory of Stochastic Approximation. This provides a framework for analyzing the asymptotic and transient properties of stochastic optimization algorithms as parameters such as the step size are varied. In Section 4 we consider another model, the Sigmoid Belief Network, which is similar to the Boltzmann machine but has an acyclic and directed connectivity graph. Section 5 considers continuous state space models that may have cycles in the connectivity graph, known as attractor networks. These are also known as fixed-point or recurrent neural networks. The last model we consider, in Section 6, is a chemical reaction network. We finish with a discussion in Section 7

1.1 Notation

For reference, we record some of the notation that is used in the rest of this survey.

- \( n \) - dimensionality of the state space of a model. In a network based model, this will be the number of nodes in the network.
- \( V \) - a subset of \( \{1, \ldots, n\} \) defining the indices of the visible units.
- \( n_V \) - number of visible units in a model.
- \( n_H \) - number of hidden or latent variables in a model.
- \( \mathcal{X} \) - state space of the model.
- \( x \# U \) - projection of the vector \( x \) onto the components \( U \). Formally, the vector \( (x_{U_1}, x_{U_2}, \ldots, x_{U_{|U|}}) \).
- \( m \) - number of training examples.
- \( w(1), w(2), \ldots \) - sequence of parameters generated by optimization algorithm.
- \( w^\epsilon(1), w^\epsilon(2), \ldots \) - sequence of parameters generated using specific step size \( \epsilon \).
2 Boltzmann machine

The Boltzmann machine is a network of stochastic units that operates in discrete time. The introduction of the Boltzmann machine as a machine learning model can be traced to [1]. We will consider a Boltzmann machine with a binary state, but variations involving units that take values in other discrete sets or even continuous values are also important and have been studied [2]. We first describe the model and the associated optimization problem. To illustrate how the Boltzmann machine is used we review a recent application where the Boltzmann machine is used to define a joint model of image statistics and textual descriptions. We then consider algorithms for the Boltzmann machine optimization problem, and present the results of a simple numerical experiment.

2.1 Model

A Boltzmann machine with $n$ units operates in the following way. The connections among the $n$-units is defined by an undirected collection of edges $E$. The state vector is an $n$-bit binary vector, and we can denote the state space $X = \{0, 1\}^n$. When the Boltzmann machine is running, it generates a sequence of $n$-bit binary vectors $X(1), X(2), \ldots$, where at each time $t + 1$ the vector $X(t + 1)$ is determined by $X(t)$ and some random input. We use subscripts to denote components of a vector, so $X_i(t)$ is the state of the $i$th node at time $t$. At time $t + 1$, a random index $I_{t+1}$ in $\{1, \ldots, n\}$ is chosen (i.e. a random node of the graph) to possibly be updated. We let

$$u_i(x) = \sum_{j: (i,j) \in E} w_{i,j} x_j + b_i$$

denote the input to unit $i$, at the network state $x$. Then the state of node $I_{t+1}$ is updated randomly depending on the value of $u_i(X(t))$ in the following way:

$$P(X(t + 1)_i = k \mid X(t) = x \text{ and } I_{t+1} = i) = \begin{cases} \sigma(u_i(x)) & \text{if } k = 1 \\ 1 - \sigma(u_i(x)) & \text{if } k = 0 \end{cases}$$ (1)

The parameters of the model are the weights $w$ and the biases $b$. These are arbitrary real numbers, with the important constraint that $w_{i,j} = w_{j,i}$. The function $\sigma$ is

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

From (1), we can see that if node $i$ is chosen for updating, and it receives a large positive input, meaning $u_i(X(t))$ is a large positive number, then it is very likely that it will have the value 1, or be “on” at the next time step. Likewise, if $u_i(X(t))$ is a large negative number, then it will most likely take the value 0, or be “off”, at the next time step.

The Markov chain that we have just described determines the operation of the Boltzmann machine. Applying basic results in the theory of Markov processes, one can show that the chain is ergodic, converging in measure to a unique stationary distribution. This follows from the fact that any state can be reached from any other after $n$
Furthermore, it can be shown based on the properties of the sigmoid function $\sigma$ and the update rule (1) that the stationary measure has a nice closed form solution. Let $E(x; w, b) = -\sum_{(i,j) \in E} w_{i,j} x_i x_j - \sum_{i=1}^{n} b_i x_i$ denote what will be called the energy of a state $x$. Then the stationary measure $\pi$ is the measure on $\{0, 1\}^n$ given by

$$\pi(x; w, b) = \frac{e^{-E(x; w, b)}}{\sum_{y \in \{0, 1\}^n} e^{-E(y; w, b)}}$$  (2)

The quantity $E(x)$ associated to each state $x$ is the known as the energy of the state, and in stationarity, the probability of a state is increases as the energy of that state decreases. This leads to an interpretation of the parameters $w$ and $b$ as constraints. For instance a value of +1 for a bias $b_i$ reflects the constraint that node $i$ should be on. A value of −1 for a weight $w_{i,j}$ means that node $i$ and node $j$ are not on at the same time, while $w_{i,j} = +1$ reflects the constraint that when one is on, so is the other. The energy of a state $x$ then reflects the degree to which the state violates the constraints; more constraints being violated means higher energy, and the process of running the Boltzmann machine can be seen as a stochastic search for a low energy state, or one which satisfies the constraints.

The denominator in (2), which is a function of the parameters $w$ and $b$, is known as the partition function. Based on the definition (2), one can see that to increase the probability of observing a particular state $x$ at stationarity, one must make $E(x; w, b)$ small while raising the energy $E(y; w, b)$ of other states $y \neq x$. We will have more to say about optimization below.

Typically, the Boltzmann machine is used to model a probability distribution over binary vectors. The formalism is similar in the case that some nodes are also continuous valued, see [2]. During training, the parameters of $w, b$ are tuned with the goal of bringing the distribution of the Boltzmann machine as close as possible to the distribution of binary vectors specified by the particular domain. At run time the Boltzmann machine is used to perform any number of tasks one would want to do with a probability distribution: Draw samples from the distribution, fill in missing data, sample from conditional distribution, or approximately compute probabilities of a specific binary vector or a subset of vectors.

In many applications, one would attempt to model a distribution over binary vectors of length $n_V$ with a Boltzmann machine with $n_V$ units, for $n_V < n$. This involves adding some auxiliary or “hidden” units in the Boltzmann machines. Such units increase the expressive power of the Boltzmann machine, giving it at least the possibility of being able to capture higher-order (i.e. more than pairwise) statistical properties in a probability measure over length $n_V$ binary vectors. We can partition the state space of the Boltzmann machine into a group of $n_V$ visible units and $n_H = n - n_V$ visible units. Let $V$ be a subset of $\{1, \ldots, n\}$ of size $n_V$ which consists of the indices of the visible units. Correspondingly we define $H = \{1, \ldots, n\} \setminus V$. Given a vector $x$ of
dimension \( n \) we will use the notation \( x^\#V \) to denote the subvector of \( X \) defined by \( x^\#V = (x_{V_1}, x_{V_2}, \ldots, x_{V_n}) \). For a vector \( v \in \{0,1\}^n \) we let \( \pi(X^\#V = v; w, b) \) be the probability that the visible units take the values \( v \). For ease of notation, sometimes we will leave it implicit that we are marginalizing out the hidden units and simply write \( \pi(v; w, b) \). Usually it will be clear, by using variables like \( v, v^1, v^2 \), that we are talking about a marginal probability and not the full joint probability of the Boltzmann machine.

2.2 Optimization problem

Let us formally state the problem one wants to solve when training the Boltzmann machine. Let \((E, (w, b), V)\) define a Boltzmann machine on \( n \) nodes, with visible units \( V \), connections \( E \), and parameters \((w, b)\). Let \( Q \) be a distribution on \( \{0,1\}^n \). Define \( J(w, b) \) as

\[
J(w, b) = \sum_{v \in \{0,1\}^n} Q(v) \log \frac{Q(v)}{\pi(v; w, b)}
\]

This is the Kullback-Liebler divergence between the measure \( Q \) and the measure on visible vectors defined by \( \pi \), and the goal of optimization is to minimize this divergence.

In most cases of interest one cannot directly compute \( J \) or its derivatives. This is not only because it requires computing \( \pi \) but also because \( Q \) is not directly available. Usually one only has the option of obtaining many samples from \( Q \) by some experiment. A common approach in machine learning is to use a large number of samples \( v^1, \ldots, v^m \) to form the random measure

\[
\hat{Q} = \frac{1}{m} \sum_{i=1}^m \delta_{v^i}
\]

and from here define the random function

\[
\hat{J}(w, b) = \sum_{v \in \{0,1\}^n} \hat{Q}(v) \log \frac{\hat{Q}(v)}{\pi(v; w, b)}
\]

and then send the random function \( \hat{J} \) to an optimization algorithm, to solve

\[
\min_{(w, b)} \hat{J}(w, b)
\]

Remark 2.1. Since the function to minimize is random, the output of any optimization routine to solve this problem is a random parameter \( \hat{w} \). In general, the asymptotic behavior of this random variable \( \hat{w} \) as the number of samples \( m \) that define our empirical distribution \( \hat{Q} \) tends to infinity should be of importance for practitioners. For instance, one might want to know that the result of optimizing the approximate function \( \hat{J} \) approaches the result of optimizing the true function \( J \) as the number of samples grows, and also to know the speed of convergence. This type of analysis would be allow one to say something about generalization. One approach is the stochastic-counterpart method of [3], but we do not pursue this further in the current survey.
2.3 Application: A Joint Model of Images and Text

A recent application of the Boltzmann machine can be found in [4], where the authors consider a multimodal learning problem. The task is to build a useful joint probability model on image features and textual descriptions of images. Such a model would facilitate tasks such as the assignment of tags to images, and, going the other way, should be useful for image searches. Srivastava et. al. use a variant of Boltzmann machine, termed the Deep Boltzmann Machine (DBM) to approach this problem. We present a somewhat simplified account of their work, referring the reader to [4] for a full description.

To attempt to capture this distribution with a Boltzmann machine, they consider an architecture with multiple layers, each connected symmetrically with the layer before and after. The nodes of the first layer collectively represent image features while the last layer represents text. (Note that unlike a feed-forward network, notions of first and last are arbitrary in the DBM, as all connections are undirected). The middle layers consist of hidden variables $H$ that, hopefully, through optimization, are able to capture the complex, higher-order correlations between the video and text representation of a stimulus. Due to the multilayer structure they impose on the Boltzmann machine, they refer to it as the Deep Boltzmann machine. A schematic of the architecture is shown in Figure 2. Corresponding to this, the visible units would be the units in the first and last layers, and we refer to these as the image units and text units, respectively.

In the work of [4] there are 2000 text units, each one corresponding to a possible word that can be used as an image tag. We let $v^{(m)}$ denote an assignment of values to the image units, and let $v^{(t)}$ be a state of the text units. We likewise partition the weights $w$ into three matrices: $w^{h,h}$, consisting of the weights between the text units and the hidden units, $w^{h,h}$ consisting of weights along connections among the hidden units, and $w^{h,m}$, the weights between the image units and the hidden units. The biases are split into 3 vectors $b^{m}, b^{h}$, $b^{t}$ for the image, the hidden, and the text units respectively.

A state vector in the DBM can be represented as a triple \( \{v^{(m)}, v^{(t)}, h\} \), and the
energy function takes the form

$$E(\{v^{(m)}, v^{(t)}, h\}; w, b) = -\sum_{k,j} w^{h,m}_{k,j} v^{(m)}_k - \sum_{j,l} w^{h,h}_{j,l} h_j h_l - \sum_{i,j} w^{h,t}_{i,j} v^{(t)}_i$$

In fact, the matrix $w^{h,h}$ has further structure, as the hidden units themselves are broken into various layers (see Figure 2), but we omit this for clarity.

The optimization problem is to get the distribution of $(v^{(m)}, v^{(t)})$ specified by the Boltzmann machine to match the empirical distribution, as described in Section 2.2 above. The dataset used to define the empirical distribution consists of images and tags for those images from the online image sharing website Flickr. Each image is preprocessed into image features, resulting in a vector of size $3857$ per image. The tags for images were restricted to the $2000$ most common tags identified from a larger database of $1$ million images. The tags for a given image are then represented with a vector of $2000$ bits, where the $i$th bit is on if the image has the corresponding tag. Combining these, a single training case is a vector of length $3857 + 2000$, consisting of an image feature vector and a word tag vector. The number of visible image units in the DBM was $3857$ while the number of visible text units was $2000$. A total of $10,000$ images were used during optimization to define the empirical distribution $\hat{Q}$.

We describe the optimization procedure in more detail below.

After optimization, the model can be put to practical use as follows. To generate a sample word vector for an image, one samples from the distribution $\pi(v^{(t)}|v^{(m)})$. The algorithm for generating a sample from a conditional distribution is similar to the algorithm for running the Boltzmann machine. One simply “clamps” the image units to the value $v^{(m)}$, and otherwise runs the Boltzmann machine normally. An example of the results the authors obtain is shown in Figure 3.

Likewise, given a set of tags among the $2000$ most common words, their system is able to recover a reasonable figure with that description. A slight complication is that their model doesn’t directly describe a joint probability measure on images, only a higher level representation of the image, in terms of features. To map the features back into image space, a database of correspondences between features and images is stored, and a nearest neighbor search is performed to associate an image to a set of features generated by the model. This is necessary as the “full” task which tries to model the images directly would be too difficult.

### 2.4 Optimization algorithm

The sum over exponentially many states needed to compute the partition function (that is, the denominator in equation (2)) precludes the possibility of direct, non-simulation based computation for many problems relating to $\pi$. We now describe a particular approach to optimization involving stochastic approximation. In the context of the Boltzmann machine it is called Persistent Contrastive Divergence [5]. The origin of the
procedure in the context of Boltzmann machine seems to be the work [6]. In this section we will describe the optimization procedure, and in the next section we will consider the question of convergence and place the algorithm in the more general setting of Stochastic Approximation algorithms.

Recall that the optimization problem is to minimize the function \( \hat{J} \), defined by Equation 3, which is the KL divergence between an empirical distribution and the distribution specified by the Boltzmann machine. The standard gradient descent algorithm follows the recursion

\[
\Delta(t + 1) = \frac{\partial \tilde{J}}{\partial w}(w(t)) \\
w(t + 1) = w(t) - \epsilon \Delta(t + 1)
\]

for some small step size \( \epsilon \). As mentioned above, exact computation with the Boltzmann machine is in general not possible, but we shall see how sampling methods can be used to approximate the derivatives. Taking the gradient of \( \tilde{J} \) with respect to the parameters, we see that

\[
\frac{\partial \tilde{J}}{\partial \theta}(w, b) = -\frac{1}{m} \sum_{i=1}^{m} \log \pi(v^i; w, b)
\]

Referring to equation (2) for the stationary measure \( \pi \), we can obtain the following formulas for the derivatives

\[
\frac{\partial \tilde{J}}{\partial w_{j,k}}(w, b) = E_{\pi}[X_j X_k] - \frac{1}{m} \sum_{i=1}^{m} E_{\pi}[X_j X_k \mid X \#V = v^i] \\
\frac{\partial \tilde{J}}{\partial b_i}(w, b) = E_{\pi}[X_j] - \frac{1}{m} \sum_{i=1}^{m} E_{\pi}[X_j \mid X \#V = v^i]
\]

In the above equation (4) and (5) we can see there are two types of expectations: Expectations with respect to \( \pi_{\theta} \), which we call model expectations and expectations conditioned on the data, which we call data dependent expectations. If we define the
measure
\[ \mu(\{h, v\}) = \pi(X#H = h \mid X#V = v)\hat{Q}(v) \]
we can write the derivative more compactly as
\[ \frac{\partial \hat{J}}{\partial w_{j,k}}(w, b) = E_\pi [X_jX_k] - E_\mu [X_jX_k] \]

To compute the data dependent expectation, we can proceed as follows. Introduce \( m \) Markov chains \( X^1(1), X^1(2), \ldots, X^2(1), X^2(2), \ldots \), up to \( X^m(1), X^m(2), \ldots \), where the chain \( \{X^j(1), X^j(2), \ldots\} \) is a copy of the Boltzmann machine with the state of the visible units clamped to the value \( v^j \). The distribution of the \( X^j(1) \) converges to \( \pi(X \mid X#V = v) \) as \( k \to \infty \). This reflects a nice property of the Boltzmann machine that conditioning on any subset of the nodes yields a distribution of the same type as that specified by the Boltzmann machine itself.

Likewise, to approximate the model expectation one runs the Boltzmann machine, starting from an arbitrary initial condition, for a large number of steps. When we combine the estimators for the model expectation and the data dependent expectation we get that
\[ E_\pi \left[ X(t),X(t)_j - \frac{1}{m} \sum_{k=1}^{m} X^k_j(t)X^k_j(t) \right] \to \frac{\partial \hat{J}}{\partial w_{i,j}}(w) \]
as \( t \to \infty \). Based on this, the idea behind the optimization procedure is to continually run the \( n + 1 \) copies of the Boltzmann machine, the first being unclamped and the remaining \( n \) copies of the Boltzmann machine having their visible units clamped to the vectors \( v^1, v^2, \ldots, v^n \), respectively. At each time step, one performs a parameter update based on the states of this auxiliary process.

We can formally describe the algorithm as follows. Let \( X \) be the state space of the Boltzmann machine. Let \( P_U(\cdot \mid X; (w, b)) \) be the Markov kernel governing the Boltzmann machine, in the normal “unconditioned” mode. Let \( P_C(\cdot \mid X; v, (w, b)) \) be the Markov kernel governing the Boltzmann machine when the visible units are clamped the vector \( v \). Recall that \( \mathcal{X} = \{0,1\}^n \) is the state space of the Boltzmann machine. Then the state space of the auxiliary system is \( \mathcal{Z} := \mathcal{X}^{m+1} \). The auxiliary system consists of the \( \mathcal{Z} \)-valued random variables \( Z(1), Z(2), \ldots \), where \( Z(k) = (X(k), X^1(k), \ldots, X^m(k)) \) and the \( Z(k) \) are distributed as follows:
\[ P(Z(t+1) \mid w(0), Z(1), \ldots, Z(t)) = P_U(X(t+1) \mid X(t); w(t)) \Pi_{j=1}^{n} P_C(X^j(t+1) \mid X^j(t); w(t), v^j) \]
and the parameters \( w(t) \) are defined as:
\[ w(t+1)_{i,j} = w_{i,j}(t) - \epsilon \left[ X(t)_iX(t)_j - \frac{1}{m} \sum_{k=1}^{m} X^k_i(t)X^k_j(t) \right] \]

Equations (6) and (7) along with the initial parameter \( w(0) \), initial states \( \{X(0)_i; 0 \leq i \leq n\}, X(0) \) and the step size \( \epsilon \), determine the optimization procedure. It was shown in [5] that this method was superior to other approaches when applied to a handwritten digit recognition problem. In Section 3 we will discuss convergence of such procedures.
Figure 4: Structure of the Boltzmann machine used in the experiment. The network is partitioned into hidden and visible units (the $h_i$ and $v_i$ respectively). There are connections between visible and hidden units but no intralayer connections, a configuration known as the Restricted Boltzmann Machine.

The mixture distribution to be learned.

| $m$ | $P(M = m)$ | $P(V_i = v_i | M = m)$, $i = 1, \ldots, 9$ |
|-----|-------------|--------------------------------------------|
| 1   | 0.25        | 0.8 0.8 0.8 0.8 0.2 0.2 0.2 0.2 1.0       |
| 2   | 0.25        | 0.2 0.2 0.2 0.2 0.8 0.8 0.8 0.8 1.0       |
| 3   | 0.25        | 0.8 0.8 0.2 0.2 0.8 0.8 0.2 0.2 0.0       |
| 4   | 0.25        | 0.2 0.2 0.8 0.8 0.2 0.2 0.8 0.8 0.0       |

Figure 5: Figure from [7]. A sample of 250 vectors from this mixture distribution is used to define the empirical distribution that we attempt to model with the Boltzmann machine. The variable $M \in \{1, 2, 3, 4\}$ defines the mixture component. Within each component, the visible units are independent, distributed according to the probabilities in the corresponding row.

2.5 Numerical experiment

A program was written to replicate the experiments from [7]. The problem is to train a Boltzmann machine to model a distribution over 9-bit binary vectors. The model has 9 visible units and 6 hidden units. The structure is shown in Figure 4. The only connections allowed are between the visible and hidden units; there are no connections within the visible group or within the hidden group. A Boltzmann machine with such bipartite structure is known as a Restricted Boltzmann Machine (RBM). One reason these models are of interest is that they allow for sampling procedures that are more efficient than the random update procedure described above. Roughly speaking, the bipartite structure means that all hidden units, or all visible units, can be updated at once and the long term behavior is still governed by the same distribution. This computation can take advantage of fast linear algebra routines or parallelism.

The distribution that the Boltzmann machine is tasked with modeling is shown in Figure 5. It is a mixture model with four components. Conditioned on the component indicator variable, the 9 bits of the state vector are independent, distributed according
Figure 6: Evolution of the KL-divergence between the distribution \( \hat{Q} \) and the distribution \( \pi \) of the Boltzmann machine, as optimization progresses.

to the probabilities shown in the figure.

From this distribution, 250 random vectors are generated and these define the empirical distribution \( \hat{Q} \). The optimization problem is then set up and solved as in Sections 2.2 and 2.4. That is, optimization involved an auxiliary process for estimating gradients, and alternates between running the auxiliary process and updating parameters. The auxiliary process is made up of 250+1 parallel Markov chains; the first 250 used to calculate the data dependent expectations, and a single “unclamped” copy of the Boltzmann machine for the data independent expectations.

A constant step size of \( \epsilon = 0.1 \) was used. Each weight \( w_{i,j} \) was initialized randomly by sampling from the uniform distribution on \([-0.1, 0.1]\). The initial bias \( b_i \) at each node was also taken from this distribution. The optimization was run for \( N = 1000 \) iterations.

Since the network has only \( 9 + 6 = 15 \) nodes, exact calculation of probabilities, and hence the objective function \( \hat{J} \), is possible. Figure 6 shows how the value of \( \hat{J}(\theta_n) \), which is the KL-divergence between \( \hat{Q} \) and the stationary measure of the Boltzmann machine, evolves as training progresses. One can see that optimization proceeds smoothly in the beginning, and oscillations in the objective function begin around iteration 400.

### 2.6 Variants of the Boltzmann Machine

In the previous sections, we derived the optimization procedure for the Boltzmann machine based on the expression for the stationary measure \( \pi \), which is based on the energy function \( E \). One shortcoming of this approach is that it does not give any indication of how the optimization procedure must change when the model itself changes. For example, say that one wished to use directed connections, instead of undirected ones. Or perhaps one wants to use a different updating strategy when running the net-
work, such as synchronous updates instead of choosing one node at random each time step. Based on the methodology of the present section, one would need to re-derive the stationary measure in each case in order to find a formula for the derivatives. Other possible modifications could involve using functions other than the $\sigma$ to define the transition probabilities. While it is not difficult to invoke abstract results guaranteeing a nice long term behavior, to pursue the above approach, one needs a closed form expression for the stationary measure. Therefore, it would be desirable to have a derivation of the optimization rule that proceeds directly from a description of the networks short term behavior, and does not required a detailed description of the stationary behavior.

We note that some of these variants have been analyzed; the stationary measure for the synchronous Boltzmann machine has been derived in [8]. However, there does not seem to be corresponding results for models, either synchronous or asynchronous, that lack symmetry of the weights. Some preliminary work studying the Boltzmann machine with asymmetric weights and synchronous updates, can be found in [9], but this work only considered optimizing over a finite time-horizon.

3 Stochastic approximation

Many stochastic optimization algorithms can be written in the following general form:

\begin{align*}
Z(t + 1) &= F(Z(t), \xi_{t+1}; w(t)) \\
\Delta(t + 1) &= G(Z(t + 1); w(t)) \\
w(t + 1) &= w(t) + \epsilon \Delta(t + 1)
\end{align*}

(8a)

(8b)

(8c)

Here, the sequence \(\{\xi_t, t = 1, 2, \ldots\}\) determines the random input to the algorithm. The \(\xi_t\) may be, for example, i.i.d uniform random vectors, of whatever dimensionality is appropriate. The \(w(t)\) is the sequence of parameters and \(\Delta(t)\) are the parameter update directions. \(X(t)\) represents some auxiliary process used to compute the parameter update directions, and \(G\) is a function that prepares the update direction from the state of the auxiliary process. For instance, the Boltzmann machine optimization procedure defined in the previous section is of this form.

One of the settings of this algorithm is the step size \(\epsilon\). When we want to differentiate between executions of the algorithm with different step sizes, we will use a superscript \(\epsilon\). For example, \(\Delta^\epsilon(t + 1)\) indicates the random variable which is the update direction at step \(t + 1\) when using step size \(\epsilon\). Likewise, \(w^\epsilon(t)\) is the (random) parameter obtained at step \(t\) when using step size \(\epsilon\).

Generalizing the discussion of optimization in the Boltzmann machine, we can state some general conditions that should be satisfied so that the stochastic parameter adaptation procedure of (8a-8c) can be used for optimization. Consider a function \(J(w)\) that needs to be minimized. Say that one can formulate a stochastic process to approximate the derivative \(\frac{\partial J}{\partial w}\) at each point \(w\). Specifically, assume the following setup:

i. There are measures \(\pi_w\) on a state space \(Z\), and a measurable function \(G : Z \times \)
\( W \rightarrow \mathbb{R} \) such that the derivative can be represented as an expectation

\[
\frac{\partial J}{\partial w}(w) = \mathbb{E}_{\pi_w}[G(Z; w)]
\]

ii. The measures \( \pi_w \) can be realized as invariant measures of ergodic Markov kernels \( P_w \) having the representation

\[
(P_w e)(z) = \int_{\Xi} e(F(z, \xi, w))d\nu(\xi)
\]  \hspace{1cm} (9)

Under some additional assumptions, in algorithm (8) one could expect that the variables \( \Delta(t+1) \) are, for all times \( t \), good approximations to the true gradients \( \frac{\partial J}{\partial w}(w(t)) \), in an appropriate probabilistic sense, assuming that the variables \( Z(t+1) \) are nearly distributed according to the stationary measure \( \pi_{w(t)} \). Furthermore, if the step size is chosen small enough, the trajectory of the \( w(t) \) should closely track that of the continuous time gradient system started from the same initial condition:

\[
\begin{align*}
\pi(0) &= w(0), \\
\frac{d}{dt} w(t) &= -\frac{\partial J}{\partial w}(w(t))
\end{align*}
\]  \hspace{1cm} (10)

The theory of stochastic approximation can formalize this. The technical conditions include those on the step size \( \epsilon \), the ergodicity and continuity of the chain (9), and growth conditions on \( G \). We give a precise statement of such a result in Proposition 3.2 below.

### 3.1 Weak convergence to an ODE

Probably the strongest guarantee about an algorithm for function minimization is that it will find a global minimum in a finite number of steps. When such an algorithm is not available, one can look for algorithms that satisfy weaker, but still useful properties. For instance, an algorithm whose only guarantee is that it eventually outputs a parameter value that is better than the initial one could still be useful in certain very difficult cases. There are also probabilistic guarantees that could be useful. A basic probabilistic guarantee is a statement that the random output of the program meets a given performance criteria with a certain level of probability. If the problem has an optimal parameter \( x^* \), then a useful guarantee may involve confidence intervals. An example of this would be a guarantee that the probability of the random output being within \( \epsilon \) distance of the optimum is at least \( 1 - \delta \). More sophisticated guarantees can involve the distribution of the output \( x \). For instance, if one knows that \( x^* - x \) is normally distributed (or nearly so), this provides a means of constructing confidence intervals for any desired level of confidence.

In the context of optimization, weak convergence formalizes the idea that the behavior of the iterations (8) resembles that of the trajectory of the continuous time gradient system (10) ever more closely as the parameter \( \epsilon \) tends to zero. This guarantee can be combined with information on the behavior of the ODE, such as its convergence to a stable equilibrium \( w^* \), to yield corresponding statements for the behavior of the
algorithm (8). We will assume that the ODE converges to a stable point \( w^* \), but results using more general assumptions on the ODE are also possible. We will describe the technical definition of what it means for the algorithm (8) to converge weakly to the solution of the ODE (10) below, but first we state one of its consequences to motivate why it is a useful criteria. This is the fact that weak convergence enables the construction of confidence intervals of any desired width, and for any level of probability. Let \( w^\varepsilon(t) \) be the random variable that gives the parameter value at step \( t \) of the algorithm, when using a step size of \( \varepsilon \). The following result is proved in [10]:

**Proposition 3.1** ([10], Theorem I.2.3). *Assume that the stochastic optimization algorithm (8) converges weakly to the solution of the ODE (10) as \( \varepsilon \) tends to 0 (in the sense of Definition 3.3 below). Then for any desired radius \( \gamma > 0 \) and any desired level of probability \( \delta > 0 \) there is a step size \( \varepsilon > 0 \) and an iteration number \( t \) so that\[
P( w^\varepsilon(t) \in B(w^*, \gamma) ) > 1 - \delta \]

Note that the step size \( \varepsilon \) is held constant during optimization. This result is most useful when formulas are given for the \( t \) and \( \varepsilon \) corresponding to choices of \( \varepsilon \) and \( \gamma \). In that situation, a practitioner could choose their distance tolerance \( \gamma \) and their probability level \( \delta \). In response, the step size \( \varepsilon \) and the run time \( t \) are determined. Then the practitioner knows that by running the algorithm for that amount of time, and at that step size, the end result \( w^\varepsilon(t) \) will be within \( \gamma \) distance of the optimal value with probability at least \( 1 - \delta \).

To get to the idea of weak convergence, first observe that each time we run a recursive parameter adaptation procedure like (8) a sequence of values \( w(0), w(1), \ldots \) is generated. Let the parameter take values in a set \( \mathcal{W} \). This can be, for instance, \( \mathcal{W} = \mathbb{R}^k \). Then we can think of the output as a point in the space of sequences \( \mathcal{W}^{\infty} \). When there is random input to the algorithm, this sequence will be random, and in this way the algorithm defines a probability distribution over sequences. We can call this measure \( \mu \). Note that the algorithm has parameters such as the step size \( \varepsilon \), so we let \( \mu_\varepsilon \) be the distribution over sequences when step size \( \varepsilon \) is used. Next, we would like to compare a sequence \( w(0), w(1), \ldots \) generated by the algorithm with the solution \( w^t(t) \) to the ODE (10) with the same initial condition. It is a bit awkward to do so at this point since these mathematical objects are living in different spaces: one is a sequence, i.e. a point of \( \mathcal{W}^{\infty} \), and the other is a continuous curve, an element of \( C([0, \infty), \mathcal{W}) \). One way to do a proper comparison is to embed the sequence via interpolation into the space of curves, and then use whatever method of comparison is available in that space, for instance computing the distance via a metric \( d \). There is more than one way to do the interpolation, and it is important that the timescale of the new curve be some how aligned with the timescale of the ODE if we want to do a comparison. For various reasons, a good choice turns out to be the the piecewise constant interpolation, denoted by \( \phi^\varepsilon \), which is constant on intervals of length \( \varepsilon \). Formally, it sends a sequence \( w \) to the curve \( \phi^\varepsilon(w) : [0, \infty) \to \mathbb{R} \) where

\[
\phi^\varepsilon(w)(t) = w^\varepsilon(n) \text{ when } t \in [n\varepsilon, n\varepsilon + \varepsilon)
\]

In this way, the curve \( \phi^\varepsilon(w) \) takes the value \( w^\varepsilon(n) \) for all times \( t \in [n\varepsilon, n\varepsilon + \varepsilon) \), and then jumps to the next item in the sequence for time \( \varepsilon \). Note that this is actually
a family of interpolations, one for each value of $\epsilon$. Since the curve $\phi^\epsilon(w)(t)$ is in general only piecewise continuous, it is more convenient to work in a larger space than $C([0,\infty),W)$, as this only contains continuous curves. An appropriate choice turns out to be the space $D([0,\infty),W)$, consisting of càdlàg paths.

**Definition 3.1.** The space $D = D([0,\infty),W)$ consists of the càdlàg paths from $[0,\infty)$ to $W$; these are curves that are continuous from the right with left limits: $\gamma \in D$ iff for all $t$,

i. $\lim_{s \to t^+} \gamma(s) = \gamma(t)$

ii. $\lim_{s \to t^-} \gamma(s)$ exists.

$D$ is equipped with a useful metric, the Skorohod metric, making it into a Polish space. We let $\mathcal{P}(D)$ be the space of probability measures on $D$, endowed with the topology of weak convergence.

This space includes the continuous curves, but also allows curves with jump discontinuities, such as our interpolations (11).

Composing the underlying algorithm with the interpolation process yields a measure in the path space, referred to as the interpolated process:

**Definition 3.2.** Let $\mu^\epsilon$ be the measure on $W^\infty$ defined by the algorithm (8), and let $\phi^\epsilon$ be the interpolation operator defined by equation (11). The interpolated process $\nu^\epsilon$ is the measure $\nu^\epsilon = \mu^\epsilon \circ (\phi^\epsilon)^{-1}$ on the space $D$.

Given this construction, one way of stating that the algorithm tends towards the solution of the ODE is to consider the convergence in probability of the interpolated paths to the solution of the ODE. We can state this as

\[ \forall \delta > 0, \quad \lim_{\epsilon \to 0} \mu^\epsilon \left( w \mid d(\phi^\epsilon(w), \bar{w}) > \delta \right) = 0 \]  

(12)

Equivalently, we can speak of the weak convergence of the interpolated process to $\delta_{\bar{w}}$ in the space $\mathcal{P}(D)$:

**Definition 3.3.** The algorithm (8) is said to converge weakly to the solution of the ODE (10) if (12) holds, or equivalently, the interpolated process $\nu^\epsilon$ converges weakly to $\delta_{\bar{w}}$ in the space $\mathcal{P}(D)$, as $\epsilon$ tends to 0.

Of course, to make use of these definitions in practice (e.g. to set the parameters of a computer program), one needs to know how the convergence occurs, and what exactly is meant by statements like $d(\gamma_1, \gamma_2) > \delta$, for the Skorohod metric $d$. One can show, simply by unrolling the definitions, that weak convergence of the algorithm implies statements like Proposition 3.1 on confidence intervals.

Next, we state a standard result on weak convergence of an algorithm to an ODE. This is from [11]. For simplicity we have strengthened some of the assumptions of the theorem. For each $\epsilon > 0$ and $n \geq 1$ we let $\mathcal{F}^\epsilon(n)$ be the $\sigma$-algebra generated by the random variables $\{w^\epsilon(j), \Delta^\epsilon(j), Z(j) \mid 1 \leq j \leq n\}$. 
Proposition 3.2 ([11], Theorem 8.4.4). Consider algorithm (8). For each $\epsilon > 0$, let $\nu^\epsilon$ be the corresponding interpolated process. Let $\pi$ be the solution to the ODE (10). Let the following assumptions hold:

i. (A1.1) The random variables $\{\Delta^\epsilon(t) \mid \epsilon > 0, t \geq 1\}$ are uniformly integrable,

ii. (A1.5) There is a continuous function $g : \mathcal{Z} \times \mathcal{W} \to \mathcal{W}$ such that

$$
\mathbb{E}[G(z^\epsilon(t + 1), w^\epsilon(t)) \mid \mathcal{F}(t)] = g(z^\epsilon(t), w^\epsilon(t)),
$$

iii. (A1.7) The collection $\{z^\epsilon(t) \mid t \geq 1, \epsilon > 0\}$ is tight,

iv. (A4.1) For each $w$ there is a Markov kernel $P_w$ on $\mathcal{Z}$ so that $(w, z) \mapsto P_w(z, A)$ is measurable, and

$$
P(z^\epsilon(t + 1) \in A \mid \mathcal{F}^\epsilon(t)) = P_w(z^\epsilon(t), A),
$$

v. (A4.3) For any bounded, continuous, real-valued function $H : \mathcal{Z} \to \mathbb{R}$, the function $(z, w) \mapsto (P_w H)(z)$ is continuous,

vi. (A4.13) For each $w$ the kernel $P_w$ has a unique invariant measure $\pi_w$, and for any compact subset $\mathcal{H} \subseteq \mathcal{W}$, the collection $\{\pi_w \mid w \in \mathcal{H}\}$ is tight,

vii. (A4.14) The collection of random variables

$$
\{g(w^\epsilon(t), z^\epsilon(t)) \mid t \geq 1, \epsilon > 0\}
$$

are uniformly integrable and, for any compact $\mathcal{H} \subseteq \mathcal{W}$, there is a number $K_0(\mathcal{H}) \geq 0$ such that

$$
\sup_{w \in \mathcal{H}} \int_\mathcal{Z} |g(w, z)| d\pi_w(z) \leq K_0(\mathcal{H}),
$$

viii. For all $w \in \mathcal{W},$

$$
\int_\mathcal{Z} g(z, w) d\pi_w(z) = -\frac{\partial J}{\partial w}(w).
$$

Then $\nu^\epsilon \to \pi$ as $\epsilon \to 0$.

To apply the above theorem, one needs to verify the rather technical conditions i-viii. The statement of this theorem does not indicate how the convergence might occur - meaning how one should choose the parameters of the algorithm to match the behavior of the ODE. More refined versions of the theorem would be needed for it to be practical.
3.2 Applying SA to the Boltzmann machine

One interesting direction would be to see if the conditions of Theorem 3.2 apply to the Boltzmann machine optimization procedure we have described. Several of the conditions should become trivial due to the discrete nature of the state space of the model, which we denoted $\mathcal{X}$, and that of the auxiliary space $\mathcal{Z}$. Working in a related convergence framework, that of convergence with probability one, the authors of [6] obtained a convergence result for a special case of the Boltzmann machine optimization problem - that of maximizing the probability of a single state vector in a Boltzmann machine with no hidden units. That author then extended their work in [12] to a more general setting. It seems that more work would need to be done in this direction to make these results useful for those who want to use Boltzmann machines. This includes things like finding out how the rates of convergence to the ODE depend on the problem one is trying to solve. This includes properties of the model (for example, the Boltzmann machine), and also probabilities of the distribution one is trying to model (the distribution $Q$).

3.3 Application to online Bayesian learning

In this section we discuss a recent application of stochastic approximation to online Bayesian learning. Bayesian learning formulates the problem of estimating a parameter $\theta$ governing a distribution as an inference problem. Online Bayesian learning is an iterative approach to this problem, where at each time step one receives a new data point and revises their belief based on this new evidence.

Formally, let $\Theta$ be a space of parameters and let $X$ be a space of observations. Each choice $\theta \in \Theta$ defines a probability distribution $p(x \mid \theta)$ on $X$, known as the likelihood. One's initial belief about the value of $\Theta$ is encoded in a prior distribution $p(\theta)$ on $\Theta$. After receiving data $x$, one updates their belief by forming the posterior distribution $p(\theta \mid x)$ using Bayes Rule:

$$p(\theta \mid x) = \frac{p(x \mid \theta) p(\theta)}{\int_{\Theta} p(\theta') p(x \mid \theta')}$$

Using Bayes rule, after data is observed one could estimate $\theta$ via the mean of $p(\theta \mid x)$, or as the the mode of the distribution, also known as the the maximum a posteriori (map) estimate.

In online methods, one considers observations $x_1, x_2, \ldots$, arriving sequentially. The work [13] allows for control inputs $b_n$ at each time to the process generating the data, with the assumption that the observations are independent given the control inputs, that is,

$$p(x_1, \ldots, x_n \mid \theta, b_1, \ldots, b_n) = \prod_{i=1}^n p(x_i \mid \theta, b_i)$$

The distributions $p(\theta \mid x_{[1,n]}, b_{[1,n]})$ can be computed recursively based on the formula

$$p \left( \theta \mid x_{[1,n+1]}, b_{[1,n+1]} \right) = p \left( \theta \mid x_{[1,n]}, b_{[1,n]} \right) \frac{p \left( x_{n+1} \mid \theta, b_{n+1} \right)}{\int_{\Theta} p(x_{n+1} \mid \theta', b_{n+1}) p(\theta' \mid x_{[1,n]}, b_{[1,n]})}$$

(13)
In some cases, one can prove consistency theorems for the iterations (13). A typical result might say that the posterior distributions become more and more concentrated around the true mean as the amount of data grows. However, in most cases the update rule (13) is not useful because the resulting probability distribution won’t have a compact representation. One possibility is to project the result of computing the update into a space of probability distributions that are computationally tractable, such as the normal distributions. To formally describe this, rewrite the update rule (13) as

\[
\mu_{n+1} = f(\mu_n, x_{n+1}, b_{n+1})
\]

where

\[
f(\mu, x, b)(\theta) = \mu(\theta) \frac{p(x \mid \theta, b)}{\int_{\Theta} p(x \mid \theta', b) \mu(\theta')}
\]

We can consider various ways of projecting \( f(\mu, x, b) \) into spaces of tractable distributions. For example, if at each step we find the normal distribution that minimizes the KL-divergence to \( f(\mu, x, b) \), then the resulting algorithm would be

\[
\mu_{n+1} = \Pi( f(\mu_n, x_{n+1}, b_{n+1}) )
\]

(14)

where \( \Pi \) is the projection operator

\[
\Pi(\mu) = N(\lambda, \sigma) \text{ where } (\lambda, \sigma) = \arg \min_{(\lambda, \sigma)} D_{KL}(\mu, N(\lambda, \sigma))
\]

(15)

In some cases, the closed loop consisting of inference and projection can be computed efficiently. However, we are introducing errors since we are no longer following Bayes rule, and therefore it is not clear how useful the long-term results of this will be.

The work of [13] provides a method for showing consistency of such approximate Bayesian learning models. They do so by interpreting algorithms of the form (14), (15) as instances of stochastic approximation with a bias term. They then apply a theorem on SA that can take into account such bias terms. The methodology is applied to a number of approximate Bayesian learning models of the form (14) and (15) from the literature. We consider two such examples below.

### 3.3.1 Posted-price auctions

The first example concerns posted-price auctions. In this setup, we assume a collection of buyers, characterized by i.i.d random variables \( Y_n \) indicating their valuation of a product. This means that buyer \( n \) will purchase a product for price \( q \) if \( Y_n > q \). The seller is interested in how the purchase probability \( P(Y > q) \) depends on \( q \), the offered price. The seller cannot directly observe the valuations held by the buyers, they can only perform experiments in the form of offers to individual buyers at various prices. This query process is described by a sequence of prices \( q_n \), for \( n = 1, 2, \ldots \), where at time \( i \) price \( q_i \) is offered to buyer \( i \). This defines the random variables

\[
I_{n+1} = 1_{Y_n > q_n}
\]

which are the “evidence” presented to the algorithm.
In the problem they assume that prices are numbers between 0 and 1. The problem assumes that the function \( q \mapsto P(Y > q) \), referred to as the demand curve and denoted \( p \), is of the form \( p(q) = 1 - \gamma q \), and the goal is to find \( \gamma \). This means the likelihoods are of the form

\[
\begin{align*}
P(I_{n+1} = 1 \mid \gamma, q) &= P(Y_n > q) = 1 - \gamma q \\
P(I_{n+1} = 0 \mid \gamma, q) &= \gamma q
\end{align*}
\]

(16)

The initial belief on \( \gamma \) is expressed using a Beta distribution with coefficients \((a_0, b_0)\).

Prior work considered an online approximate Bayesian learning procedure for this problem, and the contribution of [13] is to show that this algorithm is consistent, meaning the sequence of \( \gamma_n \) generated by the algorithm satisfies

\[ \gamma_n \to \gamma \text{ w.p. 1} \]

The algorithm has weak requirements on the sequence of prices \( q_n \) offered to the customers: Only that \( \sup q_n < 1 \) and \( \inf q_n > 0 \).

### 3.3.2 Learning from censored observations

Another example concerns learning the mean of a distribution via censored observations. Alternatively, one can view it as the problem of learning a threshold via noisy queries. There is a sequence of random variables \( Y_n \) which are i.i.d and distributed according to a normal distribution \( N(\theta, \lambda) \). The variance \( \lambda \) is known and one wants to determine \( \theta \). The \( Y_n \) are not directly observable, but at each time \( n \) one can set a threshold \( b_n \) and observe the variables \( B_{n+1} = 1_{Y_n < b_n} \). The work of [13] is not concerned with how to set the \( b_n \), but rather with showing that the mean \( \theta \) can be recovered under very mild conditions on the \( b_n \). The initial belief is expressed by a normal distribution \( N(\mu_0, \sigma_0^2) \), for some estimate \( \mu_0 \) and a level of uncertainty \( \sigma_0 \). Given the sequence \( B_n \), at each step the approximate update (14) is performed, where \( \Pi \) is a projection into the space of normal distributions. Thus, at each time step the current belief is represented as a pair \((\mu_n, \sigma_n)\). Explicitly, the update they use is

\[
\begin{align*}
\mu_{n+1} &= \mu_n - \sigma_n \left( B_{n+1} \frac{1}{\sqrt{\lambda + \sigma_n^2}} \phi(p_n) - (1 - B_{n+1}) \frac{1}{\sqrt{\lambda + \sigma}} (1 - \Phi(p_n)) - \Phi(p_n) \right) \\
\sigma_{n+1} &= \sigma_n \left( 1 - B_{n+1} \frac{\sigma_n}{\lambda + \sigma_n} p_n \phi(p_n) \Phi(p_n) + \phi^2(p_n) \Phi(p_n)^2 - (1 - B_{n+1}) \frac{\sigma}{\lambda + \sigma} \phi^2(p_n) - p_n \phi(p_n)(1 - \Phi(p_n)) \right) \\
&\qquad - \frac{\sigma}{\lambda + \sigma} \frac{\phi^2(p_n) - p_n \phi(p_n)(1 - \Phi(p_n))}{(1 - \Phi(p_n))^2}
\end{align*}
\]

where \( p_n = b_n - \frac{\mu_n}{\sqrt{\lambda + \sigma_n}} \), \( \phi \) is the normal density and \( \Phi \) is the normal cumulative distribution function. Note that we can write this as

\[
(\mu_{n+1}, \sigma_{n+1}) = g(\mu_n, \sigma_n, B_{n+1}, b_n)
\]

where \( b_n \) is the control (threshold) used to generate the observation \( B_{n+1} \), and the pair \((\mu, \sigma)\) summarize the belief from the previous time step.
4 Sigmoid belief networks

We now consider another network of stochastic binary units, similar to the Boltzmann machine except that (1) the connections are directed, rather then undirected, and (2) the connectivity graph is acyclic. This model, introduced in [7], is the Sigmoid Belief Network (SBN). The use of “sigmoid” refers to the $\sigma$ function, which is used to determine update probabilities just as in the Boltzmann machine. The model was also inspired by the idea of a Bayesian network.

Sigmoid belief networks are attractive for several reasons. First, their directed nature means that some tasks are easier for the SBN compared with their counterparts in the Boltzmann machine. For instance, a sample from the joint distribution over length $n$-binary vectors takes at most $n$-steps in the sigmoid belief network, while the distribution is only available asymptotically for the Boltzmann machine. Second, the parameters of the sigmoid belief network are somewhat easier to interpret. For instance, increasing the bias parameter at a certain node directly increases the probability of the corresponding node being found in the “on” position. Increasing a weight $w_{i,j}$ results in a model where node $i$ is more likely to be on when node $j$ is also on. However, the sigmoid belief network also comes with computational difficulties, as the gradients involve difficult conditional expectations and so one must resort to Gibbs sampling techniques during optimization.

4.1 Model

The connectivity in an $n$-node SBN is determined by a directed acyclic graph. The parameters of the model are an $n \times n$ matrix $w$ of weights on the connections between nodes, and a bias vector $b$. Let the nodes, labeled 1, 2, ..., $n$, be ordered in such a way that there is never a connection to $i$ from $j$ if $j > i$. The function $u_i(x)$, that determines the input to each node at the state $x$ is defined as

$$ u_i(x) = \sum_{0 < j < i} w_{i,j}x_j + b_i $$

To produce an output from the network, one visits the nodes 1, 2, ... in order. The value of node $i$ is determined probabilistically by the states of its predecessors as

$$ P(X_i = x \mid X_1, \ldots, X_{i-1}) = \sigma(u_i(x))^x_i + (1 - \sigma(u_i(x)))^{1-x_i} $$

The probability distribution over $\{0, 1\}^n$ determined by the update rule takes the following form.

$$ \pi(X = x) = \prod_{i=1}^{n} \sigma(u_i(x))^{x_i} (1 - \sigma(u_i(x)))^{1-x_i} $$

(17)

Alternatively, we can use the following notation of [7]: for $x \in \{0, 1\}$, define

$$ x^* = 2x - 1 $$

Using this with the identity $1 - \sigma(x) = \sigma(-x)$, we get

$$ \pi(X = x) = \prod_{i=1}^{n} \sigma(x_i^* u_i(x)) $$

(18)
4.2 Optimization problem

The typical optimization problem for the SBN is to find the parameters for the belief network that most closely capture the distribution of interest in a given domain. Also like the Boltzmann machine, the network is usually partitioned into visible and hidden units. Let \( V \) be a subset \( \{1, 2, \ldots, n\} \) of the nodes, which we call the visible group. We let \( n_V = |V| \) be the size of the visible group. After acquiring \( m \) samples \( v_1, \ldots, v_m \) of our probability distribution, we obtain the random optimization problem of minimizing the empirical KL divergence

\[
\hat{J}(w, b) = \sum_{v \in \{0, 1\}^n} \hat{Q}(v) \log \frac{\hat{Q}(v)}{\pi(v; w, b)}
\]  

(19)

4.3 Optimization algorithm

First we derive an expression for the derivative of the objective function (19), following [7]. It is obtained using basic calculus and properties of the function \( \sigma \). Let \( j < i \). Then

\[
\frac{\partial}{\partial w_{i,j}} \log \pi(v; w, b) = \sum_{s \in \{0, 1\}^n} \pi(X = s \mid X \#V = v) (2s_i - 1)s_j \sigma((1 - 2s_i)u_i(s))
\]

(20)

Summing over the \( m \) training examples, we obtain

\[
\frac{\partial \hat{J}}{\partial w_{i,j}} (w) = -\frac{1}{m} \sum_{k=1}^{m} \mathbb{E}_{\pi} [(2s_i - 1)s_j \sigma((1 - 2s_i)u_i(s)) \mid X \#V = v^k]
\]

(21)

A similar formula is available for the gradients with respect to the biases. We describe the optimization procedure suggested in [7]. One can see in equation (21) that the derivative involves conditional expectations. Calculating such conditional expectations is the computational bottleneck for the sigmoid belief network. There are certain cases that are easy, depending on the structure of the network and which nodes one is conditioning on, but, in general, one must resort to a simulation approach. An immediate choice is to do a Gibbs sampling procedure, as we describe in the next section. This is practical as long as computing the conditional probability of one node given the others is easy.

4.3.1 General Gibbs sampling

Consider the general problem of estimating a conditional expectation

\[
\mathbb{E}_{\pi} [f(X) \mid X \#V = v]
\]
We define a sequence of random variables $X(1), X(2), \ldots$ taking values in $\{0, 1\}^n$. These variables have the components corresponding to the visible units clamped to the vector $v$, meaning $X(t)_i = v_i$ for all $t$ and $i \in V$. The other components of these vectors are random, determined in the following way. The variable $X(t+1)$ is obtained from $X(t)$ by choosing a random node $I_{t+1}$ among the hidden units $\{1, \ldots, N\} \setminus V$, and updating node $I_{t+1}$ based on the probability distribution

$$P(X(t+1)_i = k \mid X(t) = x \text{ and } I_{t+1} = i) = \pi(X_i = k \mid X_j = x_j, j \in \{1, \ldots, N\} \setminus \{i\})$$

which is the distribution of unit $I_{t+1}$ conditioned on the values of all the other components. A standard property of the Gibbs sampling procedure is that the distributions of the $X(t)$ defined in this way converge to the conditional distribution $\pi(X \mid X \#V = v)$ as $t \to \infty$. This means

$$\mathbb{E}[f(X(t))] \to \mathbb{E}_\pi[f(X) \mid X \#V = v]$$

as $t \to \infty$, for any function $f : \{0, 1\}^n \to \mathbb{R}$.

### 4.3.2 Gibbs sampling in the SBN

Applying the above to the sigmoid belief network, we can compute the conditional probabilities showing up in the right side of equation (22) as follows. Let $g$ be the function

$$g(s_i; s_1, \ldots, s_{i-1}, s_{i+1}, \ldots, s_n) = \sigma((2s_i - 1)u_i(s)) \prod_{j > i} \sigma \left( (2s_j - 1) \left( s_{i}w_{j,i} + \sum_{k < j, k \neq i} s_{k}w_{j,k} \right) \right)$$

Then the conditional probability is

$$\pi(X_i = k \mid X_j = x_j, j \in \{1, \ldots, N\} \setminus \{i\}) = \frac{g(k; x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n)}{g(1-k, x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n) + g(k; x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n)}$$

The equation (23) shows something interesting: When running the sigmoid belief network, each node receives information from its predecessors and sends information to its successors. Inspecting the form of $g$, we see that running the Gibbs sampler involves communicating in both directions. A node $i$ must gather information not only from its predecessors, but there must also be some communication with its immediate successors, and the predecessors of its successors. This collection is known as the Markov blanket of the node $i$. We can see that iterating the Gibbs sampling procedure to approximate the conditional expectations needed for (20) is straightforward, although this does not mean that obtaining good approximations to the gradient is easy, as the quality of approximation depends on how many iterations are used, and the ergodicity properties of the chain.
Based on the above discussion, we can approximate the gradient (21) by introducing \( n \) parallel Gibbs samplers, each with a different training example clamped onto its visible units. Based on Equation 21, the function \( f \) that we evaluate at these samples to build the gradient estimate is

\[
f(x) = (2x_i - 1)x_j\sigma((1 - 2x_i)u_i(x_{1,i-1}))
\]

When we combine this gradient estimation procedure with a parameter update step, we end up with an algorithm that fits in the framework of stochastic approximation with state dependent noise, like the Boltzmann machine. We now write this a little more formally.

Let the state space of the auxiliary system be \( \mathcal{Z} = \mathcal{X}^m \). The auxiliary variables are \( Z(1), Z(2), \ldots \) where \( Z(k) = (X^1(k), \ldots, X^m(k)) \). Let \( P_C(\cdot \mid X; v, (w, b)) \) be the Markov kernel governing the Gibbs sampling procedure used to get a sample of \( \pi(\cdot \mid w, v) \), as defined by equations (22) and (24). Then

\[
P(Z(t + 1) \mid w(0), Z(1), \ldots, Z(t)) = \prod_{i=1}^m P_C(X^i(t+1) \mid X^i(k); v^i, w(k))
\]

and the \( w(t) \) are defined as

\[
w(t + 1)_{i,j} = w(t)_{i,j} + \epsilon \left[ \sum_{k=1}^m (2X^k_i(t) - 1)X^k_j(t)\sigma((1 - 2X^k_i(t))u_i(X^k(t))) \right]
\]

Interestingly, this shows that even beginning from a feed-forward model such as the sigmoid belief network, one can end up with a very non-trivial gradient estimation procedure, involving more complex dynamics on the underlying graph.

It may be possible to investigate the convergence of the procedure, using methods from stochastic approximation, yet this seems to have not been explored in the literature.

### 4.4 Numerical experiment

We replicated an experiment from [7]. The problem is the same as that in Section 2.5, where the Boltzmann machine was used to model a simple mixture distribution over binary vectors. Figure 7 shows the structure of the network used: 6 hidden units feed into the 9 visible units. The distribution of interest was the empirical distribution specified by the same 250 binary vectors used in the Boltzmann machine experiment. The optimization procedure is defined by Equations (25) and (26). That is, 250 copies of the network are instantiated in order to compute the conditional expectations in Equation 21, and these auxiliary chains evolve according to the Gibbs sampling procedure of Section 4.3.2.

The relatively small size of the network enables us to exactly compute the value of the objective function at each stage of optimization. A graph of the value of the objective function (the function \( J \) of Equation 19) is shown in Figure 8. One can see
that there are very little probabilistic effects in the procedure. This is likely due to the fact that the Gibbs sampling procedure converges very quickly in this setting, as the network is small and the only free variables to update are the 6 hidden units.

We also plotted the norm of the weights at each step, as a way of tracking the stability of the procedure. This is shown in Figure 9. This figure suggests that the parameters are diverging. This is one of the typical outcomes in optimization in connectionist models - instead of terminating in a local minima, the parameters end up in a region where the error function displays an asymptote, and the weights can go off to infinity while the overall error moves only a little. One way to interpret it is that at this stage of optimization, the cost per unit of decrease in the error function becomes higher and higher. To verify that this phenomena was not related to probabilistic effects or to an error in the implementation, the program was validated in a number of ways:

i. First, a procedure to exactly compute the gradients was implemented. This is feasible in a small network. Running the optimization with exact gradients produced the same results. However, as one can see from the formula for the derivative (21), calculating the exact gradients is a bit more complicated than, for example, in the Boltzmann machine.

ii. To validate the gradient computation we also implemented gradient checking routines that compared our supposed exact gradients with a finite difference approximation. This is an often used technique to validate complex derivative calculation routines. The check showed agreement between the finite difference and exact gradients on the order of $10^{-3}$, and the signs of gradient estimation agreed 100% of the time.

iii. Finally, the experiment was replicated across a range of iteration numbers and step sizes, and it was observed that the behavior reappeared in each case.

4.5 Extensions

The directed nature of the sigmoid belief network lends a certain ease of interpretability to the model: A large positive weight from node $i$ into node $j$ means that node $i$ is one possible cause of node $j$. Generating a sample from the model is easy, requiring a single
feed-forward pass through the network. It is able to represent a one-to-many mapping from causes to effects, which a deterministic feed-forward network is unable to do. However, the optimization problem is very difficult. The Gibbs sampling procedure described above has been observed to have poor mixing properties. This has inspired researchers to look at other approaches to optimization of the SBN, or to look at nearby models that have similar features but are easier to work with. The next section considers one example of this.

4.5.1 Choice of norm used to compute gradient

When obtaining a derivative estimate is expensive, it becomes more important to make sure that each parameter update extracts the maximum benefit from the estimate. This suggests looking at how derivative estimates are turned into parameter updates. In gradient descent for function minimization, it is common for the update to take the form

$$w_{n+1} = w_n - \epsilon_n G(\Delta_n)$$

where $\epsilon_n > 0$ is a step size, $\Delta_n$ is an approximation to $\frac{\partial L}{\partial w}(w_n)$ and $G$ is a function that maps derivatives to search directions. Formally, the domain of such a function is $L(\mathbb{R}^n, \mathbb{R})$, the space of linear functionals from $\mathbb{R}^n$ to $\mathbb{R}$, and the range is $\mathbb{R}^n$. The function $G$ should at least have the property that $\Delta(G(\Delta)) > 0$. This guarantees that $\frac{\partial L}{\partial w}(w_n)(G(\frac{\partial J}{\partial w_n}(w_n))) > 0$ for all $n$, which in turn guarantees that at each time $n$ there is a step size $\epsilon_n$ that will result in a function decrease. For instance, since $(\mathbb{R}^n)'$ has a natural isomorphism with $\mathbb{R}^n$, one can set $G(\Delta_n)$ to the vector $G(\Delta_n)_i = \Delta_n(e_i)$. 

![Figure 8: Evolution of the KL-divergence between the distribution $\hat{Q}$ and the distribution of the sigmoid belief network, as optimization progresses. The blue curve shows the progress of the optimization algorithm using Gibbs sampling to estimate the gradients, as described above. The Green curve shows the trajectory of a procedure that uses exact gradients at each step.](image-url)
This is the usual gradient used in gradient descent. But there are other choices of $G$ that are possible.

One way to motivate the gradient descent algorithm is through a majorization argument. One uses a local quadratic approximation that “majorizes” (dominates) the objective function in a local neighborhood of the current point, and then obtains a new point by minimizing the majorizer. The vanilla gradient descent is based on a majorization using the Euclidean norm, but other choices are possible, each choice of norm yielding a different majorizing function. Let $\| \cdot \|_A$ be any norm on $\mathbb{R}^n$ and let $\| \cdot \|_{A^*}$ be the corresponding norm on $(\mathbb{R}^n)'$. Let $L_A$ be a Lipschitz constant for the function $\frac{\partial J}{\partial w}$, as a function from $(\mathbb{R}^n, \| \cdot \|_A)$ to $(\mathbb{R}^n, \| \cdot \|_{A^*})$. Then using a quadratic Taylor expansion the following bound holds

$$J(w_{n+1}) \leq J(w_n) - \epsilon_n \frac{\partial J}{\partial w}(w_n)G(\frac{\partial J}{\partial w}(w_n)) + \frac{1}{2} L_A \epsilon_n^2 \|G(\frac{\partial J}{\partial w}(w_n))\|^2_{A^*}$$  \hspace{1cm} (27)$$

For each norm $\| \cdot \|_A$ there is a function known as a duality mapping. This maps a vector $\Delta \in L(\mathbb{R}^n, \mathbb{R})$ to a vector $\rho_A(\Delta) \in \mathbb{R}^n$ such that $\Delta(\rho_A(\Delta)) = \| \Delta \|^2_{A^*}$ and $\| \rho_A(\Delta) \|_A = \| \Delta \|_{A^*}$. Using the $G = \rho_A$, the inequality (27) becomes

$$J(w_{n+1}) \leq J(w_n) - \epsilon_n \| \frac{\partial J}{\partial w}(w_n) \|^2_{A^*} + \frac{L_A}{2} \epsilon_n^2 \| \frac{\partial J}{\partial w}(w_n) \|^2_{A^*} = J(w_n) - \| \frac{\partial J}{\partial w}(w_n) \|_{A^*} [\epsilon_n - \frac{L_A}{2} \epsilon_n^2]$$

Minimizing this quadratic function is a trivial matter. Set

$$F_w(\epsilon) = J(w) - \| \frac{\partial J}{\partial w}(w) \|^2_{A^*} [\epsilon - \frac{L_A}{2} \epsilon^2]$$

The minimum occurs at $\epsilon = \frac{1}{L_A}$. The amount of function decrease is at least

$$J(w_{n+1}) - J(w_n) \leq -\| \frac{\partial J}{\partial w}(w_n) \|^2_{A^*} \frac{1}{2L_A}$$
Inspecting this last quality we can see what to look for when finding a choice of norm to use during optimization: The products $\frac{\partial J}{\partial w} \|_{2}^{2} A^{*} A^{2} L A$ should be large at all points $w$.

The work of [14] uses the fact that the parameter in a sigmoid belief network is a matrix, and there are several natural norms one can put on a space of matrices. Each of these yields a different class of majorizers. The authors did not give a theoretical proof that such majorizers would provide better guarantees on optimization, but they report results from a number of empirical experiments. Let $\mathbb{R}^{n \times m}$ be the space of matrices with $n$ rows and $m$ columns. A matrix $W$ in this space has $K$ non-negative singular values $\lambda_1(W), \ldots, \lambda_K(W)$ where $K = \min\{n, m\}$. Then the Schatten $\infty$-norm is defined as

$$\|W\|_{S_{\infty}} = \max_{1 \leq i \leq K} \{\lambda_i(W)\}$$

This is the norm used to construct the majorizers in [14]. To use this approach one also needs to compute the duality mapping $\rho$; they give a formula for computing it involving the singular value decomposition of the matrix.

The experiments of [14] involve using relatively shallow networks to model the distribution of images of handwritten digits. The database is a binary version of the MNIST dataset, consisting of 60,000 binary images of handwritten digits of size $28 \times 28$. They experiment with several networks of varying size. All the networks have a two layer, bipartite structure, similar to that shown in Figure 7. The visible layer had $28 \times 28 = 740$ nodes, and the number of units in the hidden layer, denoted $n_H$, varied across experiments. Their smallest network has $n_H = 25$ nodes while their largest network has $n_H = 100$ nodes. Since computing the marginal probability of a given visible vector is intractable (it would involve summing over $2^{n_H}$ possibilities), the probabilities are estimated. Their results show that the model converges to a good parameter much faster than existing optimization approaches.

5 Attractor networks

We now turn to models operating on a continuous state space. Attractor networks are deterministic neural networks whose connectivity graphs may have cycles. In this general configuration, the output of the network generally will not stabilize after a fixed number of steps. There are a number of possibilities for the dynamical behavior of the network. In some situations one can guarantee that the network has a unique attractive fixed-point, and in this case one can apply variants of the usual back-propagation type of procedures to calculate gradients.

5.1 Model

The state space is $\mathcal{X} = \mathbb{R}^{n}$ and there is a set of edges $E$ with that determines the connectivity between nodes. We let $x = (x_1, \ldots, x_n)$ denote a state of the neural network. The weights $w \in \mathbb{R}^{n \times n}$ and biases $b \in \mathbb{R}^{n}$ are the parameters of the network. We let $\Theta = \mathbb{R}^{n} \times \mathbb{R}^{n \times n}$ represent the joint parameter space. The graph may have cycles, and $w$ is not necessarily symmetric. These networks may operate in discrete time or continuous time. Also, we will be concerned with networks that process a fixed
input, although a separate class of interesting optimization problems can be associated with a network that processes time varying input.

In the discrete time setting the evolution of the network is determined by a function $f : X \times \Theta \times X \rightarrow X$, where

$$f_i(x, \theta, u)_i = \sigma \left( \sum_{j: (i,j) \in E} w_{i,j} x_j + b_i + u_i \right), \quad i = 1, \ldots, n \quad (28)$$

That is, the state of node $i$ at time $t + 1$ is determined by the input $u_i$ at that node, and the states of its neighbors at time $t$. We use the same sigmoid function $\sigma$ as above, but other choices are possible.

Depending on the values of the parameters $(w, b)$, the network can exhibit a range of dynamical behaviors. For instance, a feed-forward network is one where $w$ has triangular structure and there are no self-connections ($w_{i,i} = 0$ for all $i$). Such a network reaches a stable state after a finite number of steps. Of course, the same holds if $w$ is related to a triangular matrix by permutation. Other possibilities include periodic trajectories, unstable equilibria, and globally attractive fixed-points. In [15] the author analyzed a single neuron model with a self-connection, and found that all three of the just mentioned phenomena can occur for appropriate choices of $(w, b)$. In this review we will be concerned with networks that admit globally attractive fixed-points. This includes feed-forward networks, and also networks with feed-back that is sufficiently “weak”, as we describe below.

By iterating (28) starting from an initial point $x(0)$, one obtains a sequence of network states $x(1), x(2), \ldots$ where $x(t + 1) = f(x(t), w, u)$. Alternatively, we may write $x(t + 1) = f^{t+1}(x(0), w, u)$. A network has a globally attractive fixed-point when there is a state $x^*$ that is a fixed-point for $f$, meaning $f(x^*, w, u) = x^*$, and, this fixed-point is globally attractive, meaning $\lim_{t \to \infty} f^t(x(0), w, u) = x^*$ for any initial point $x_0$. In general, this fixed-point will depend on the parameters $w$ and $u$, and to make this explicit we will write $x^*(w, u)$. The question of whether a network admits a globally attractive fixed-point for a given value of the parameters $w, b$ seems to be difficult; indeed, for one type of attractor network known as the Hopfield network, various hardness results have been obtained for this question [16].

One class of conditions that guarantee a globally attractive fixed-point can be obtained from the contraction mapping theorem. Fixing a norm $\| \cdot \|$ on $\mathbb{R}^n$, the system defined by (28) will possess a globally attractive fixed-point $x^*(w, u)$ if there is an $\alpha \in [0, 1)$ such that

$$\sup_x \| \frac{\partial f}{\partial x} (x, w, u) \| \leq \alpha \quad (29)$$

We call such an $\alpha$ a contraction coefficient for the system. Combining this with the particular form of $f$ and $\sigma$, we get the following sufficient condition on $w$: If $\| \cdot \|$ is an absolute norm, meaning $\|(x_1, \ldots, x_n)\| = \|(|x_1|, \ldots, |x_n|)\|$, then it suffices that

$$\|w\| < 4 \quad (30)$$

The condition (29) represents one situation when the network has regular long-term behavior. To “run” the network, one fixes an input $u$ and iterates the update rule
for a large, fixed, number of steps or until some convergence threshold is reached. For such dynamic approaches to reaching equilibrium, one can measure the distance $d(x(n), x(n+1))$ between successive iterates, and can use this to bound the distance to equilibrium $d(x(n), x^*)$. This follows from a basic inequality that holds for contraction mappings:

$$d(x^*, x_n) \leq \frac{1}{1-\alpha} d(x_n, x_{n+1})$$

One can use estimate to decide when to stop iterating.

### 5.2 Optimization problem

Typically, a deterministic neural network is used for classification or regression tasks. Recall that $\mathcal{X} = \mathbb{R}^n$ is the state space of the neural network. Some of these nodes will be input nodes, and some will be called output nodes. Let $n_I$ be the number of input nodes, and let $n_O$ be the number of output nodes. Let $\mathcal{I} = \mathbb{R}^{n_I}$ and $\mathcal{O} = \mathbb{R}^{n_O}$ be the input and output spaces respectively. Let $g : \mathcal{I} \rightarrow \mathcal{O}$ be any function that assigns inputs to output. This could be the function that assigns an image to the output indicating the class of the image, for example. Let $c : \mathcal{O} \times \mathcal{O} \rightarrow \mathbb{R}$ be a cost function. This means that $c(x, y)$ gives the cost incurred when a neural network outputs the state $x$ when the output should have been $y$. Let $Q$ be a distribution over $\mathcal{I}$. Then the overall cost is

$$\int_{\mathcal{I}} c(x^*(w, u), g(u))dQ(u)$$

This is the function one would want to minimize during optimization. For simplicity, however, we are just going to consider a “single sample” problem. Let $e : X \rightarrow \mathbb{R}$ be a cost function. For example, we could have the Euclidean error function $e(x) = \|x - t\|^2$ for some target vector $t$. Then the overall objective for optimization is to minimize the function $w \mapsto e(x^*(w, u))$, which maps the parameter vector to the error at the fixed-point $x^*(w, u)$.

$$\min_w e(x^*(w, u))$$

We can call the overall function $J$, that is, $J(w) = e(x^*(w, u))$. From now on we will generally consider the input $u$ as fixed and will drop it from the notation. The differentiability of $J$ follows by the implicit function theorem and the chain rule. That is, starting from the equation

$$x^*(w) = f(x^*(w), w)$$

and using the contractivity and differentiability properties of $f$, one can conclude that $x^*(w)$ is differentiable. Then as long as $e$ is differentiable we can apply the chain rule to get that $J$ is differentiable. Using this and the formulas provided by the implicit function theorem, one obtains that

$$\frac{\partial J}{\partial w} (w) = ABC$$

31
where
\[ A = \frac{\partial e}{\partial x}(x^*(w)), \quad B = \left( I - \frac{\partial f}{\partial x}(x^*(w), w) \right)^{-1}, \]
\[ C = \frac{\partial f}{\partial w}(x^*(w), w) \]

From this formula, we can see two challenges to computing, or even approximating, the derivative. The first is that these terms involve \( x^*(w) \), which can only be approximated by iteration. Secondly, they involve the solution of linear systems (one can choose between either \( AB \) or \( BC \)). We will refer to these challenges as the issues of locality in time and locality and space, respectively. We will see below how a multiple time-scale approach, along the lines of what was described for the Boltzmann machine and sigmoid belief networks, can be applied in this setting.

Attractor networks and their optimization problem (31) began to receive attention around the same time as the backpropagation algorithm became popular. The first works to consider this problem in the neural network context seem to be [17],[18],[19], which appeared at nearly the same time. These works roughly had the same innovations and limitations. Firstly, they introduced the attractor networks as the generalization of feed-forward networks that is obtained when feed-back is allowed. Next, they gave some criteria for global attractivity, including conditions like inequality (30). They suggested that the networks could be used by allowing them to reach equilibrium. They discussed differentiability and gave some ad-hoc methods of computing the derivatives. However, they did not consider the convergence of optimization nor did they articulate a multiple time-scale approach to performing the optimization.

For instance, let us consider the work of [17]. The author considers a continuous time system of \( n \) units, where the state of the \( i \)th unit evolves according to
\[ \frac{d}{dt} x_i(t) = -x_i(t) + \sum_{j: (i,j) \in E} w_{i,j} a(x_j(t)) + u_i \]
where the activation function \( a \) is any bounded, differentiable, function. For instance, the function \( \sigma \) would suffice. Written more compactly, these equations are
\[ \dot{x} = -x + Wa(x) + u \]
where \( a \) applied to a vector means to apply the scalar function to each component: \( a(x_1, \ldots, x_n) = (a(x_1), \ldots, a(x_n)) \).

The author derived a global attractivity criteria for this system in the form of a bound on the weights. Specifically, they showed that global attractivity holds when
\[ \sum_{i=1}^{n} \sum_{j=1}^{n} a_{i,j}^2 < \frac{1}{\|a\|_{Lip}^2} \]  
(33)
(for this criteria to make sense, we should have that \( a \) is non-constant. If \( g \) is constant, the attractivity is even simpler). For instance if \( a \) is the function \( \sigma \) then \( \|a\|_{Lip} = 1/4 \), and the criteria is that the sum-of-squares of the weights is less than 16. Also, the result
shows that the attractivity is exponential: When inequality (33) holds there is an \( \alpha > 0 \), depending on \( w \), so that

\[
\|x(t) - x^*(w)\|^2 \leq \|x(0) - x^*(w)\|^2 e^{-\alpha t}
\]

That work also mentioned the important issue of bifurcations in attractor networks. Even beginning with a parameter that yields global attractivity, during optimization, the weights may drift into a region for which the network does not possess a globally attractive fixed-point. At this point the value of our objective function in (31) is no longer well-defined. The authors proposed solution is simply to scale the weights back down if they get to big. Of course, this cannot happen in feed-forward networks, since optimization is constrained to the subset consisting of triangular matrices. As one solution to this problem, the author of [17] suggested a novel architecture involving a hierarchy of attractor networks. To overcome the limitation that the weights in an attractor network must be small, one can link many attractor networks together, in a directed acyclic manner. As long as each subnetwork satisfies the contractivity property uniformly for all inputs, then so will the whole network.

5.3 Optimization algorithm

The work of [20] proposed such an approach to optimization that involves running gradient estimation and parameter updates in parallel. As we shall see, the overall optimization procedure consists of a copy of the network, and an auxiliary system running alongside it, using the same connectivity pattern, and which is used to approximate the gradients. The optimization consists in periodically sampling from this auxiliary system to make a parameter update, and then allowing some time to pass so the system can settle back down to equilibrium.

Let the space \( \mathcal{Z} \) be \( \mathbb{R}^n \times \mathbb{R}^n \). We represent elements of \( \mathcal{Z} \) by pairs \( z = (x, y) \). It is straightforward to see that the gradient (32) can be represented as

\[
\frac{\partial J}{\partial w}(w) = G(z^*(w), w)
\]

where

\[
G((x, y), w) = y \frac{\partial f}{\partial \theta}(x, w)
\]

and \( z^*(w) = x^*(w), y^*(w) \) is the fixed-point of the map

\[
T((x, y), w) = \left( \frac{\partial f}{\partial x}(x, w) \right)^T y - \frac{\partial e}{\partial x}(x)
\]

This follows from basic linear algebra and the definitions of \( A, B, \) and \( C \) above. If \( T \) defines a globally attractive process on \( \mathcal{Z} \), this gives an iterative method to estimate the gradient: Iterate \( T \) enough times starting from an arbitrary point \( (x_0, y_0) \) to obtain point \( (x_m, y_m) \) close to \( (x^*(w), y^*(w)) \), and then form the estimate \( G((x_m, y_m), w) \).

By continuity properties of \( f \) it should be that

\[
G((x_m, y_m), w) \approx G(z^*(w), w)
\]
which is the true gradient. For a wide class of stability conditions on \( f \), the auxiliary system \( T \) inherits the global attractivity of \( f \). This includes the conditions on the Jacobian of \( f \) such as inequality (29). This is discussed in [20] and other works concerning attractor networks.

To turn this into an optimization procedure, one can alternate between gradient estimation and parameter updates, and indeed can be shown that [20] did just this. The algorithm consists in continually iterating \( T \) to obtain the sequence \( z_1 = (x_1, y_1), z_2 = (x_2, y_2), \ldots \) evolving in the auxiliary space \( Z \), and sampling these \( z_n \) to perform an approximate gradient update, using the function \( G \) of Equation 34. Formally, this goes as follows:

\[
\begin{align*}
Z(t + 1) &= T(Z(t); w(t)) \\
\Delta(t + 1) &= G(Z(t + 1); w(t), b(t)) \\
w(t + 1) &= w(t) - \epsilon \Delta_{i,j}(t + 1)
\end{align*}
\]

Clearly this fits into the general stochastic approximation framework described above. Here the stochastic feature is absent, only leaving the two timescale aspect. We would also like to note that feed-forward networks form a special case of attractor networks, and therefore this procedure can be viewed as a generalization of back-propagation, which is indeed how the author of [20] viewed it.

In the above algorithm, we can think of \( \epsilon \) as determining the relative rates of parameter adaptation and gradient estimation. In [20], the author notes how setting these parameters is important to guarantee optimization works. Essentially, \( \epsilon \) must be so small that \( x(t) \) and \( y(t) \) are always near their equilibrium points \( x^*(w(t)), y^*(w(t)) \), which by continuity guarantees that the approximate gradients \( \Delta(t), t = 1, 2, \ldots \) are accurate. However, the author did not rigorously investigate how to set \( \epsilon \) to guarantee convergence of the procedure.

The work of [21] showed how some results on perturbed systems could be applied to obtain local convergence of the optimization procedure. Working in continuous time, they compare the system using approximations with the system in which exact gradient information is used at each step. Under the assumption that the true optimization problem is started near a local minimum, and that the parameters of the algorithm are set appropriately (e.g. using small enough step sizes, and initializing the adjoint system near equilibrium), they can conclude convergence of the approximate optimization algorithm to the same equilibrium. However it would be more useful to make this explicit, by finding out exactly how the parameters should be set, in terms of the given model, to achieve the convergence. Additionally, the results are only applicable near a local optimum, so they are not as useful in the general situation when one is starting at a random state of the network.

Recent work in machine learning shows interest in models that have some form of feedback. For instance, while the standard framework for image classification uses a feed-forward architecture, several works have investigated features such as lateral inhibition, intralayer feedback, or top-down feedback. The recent work of [22] considered a type of “bounded” intralayer feedback added on to a traditional convolutional network architecture. Essentially this allows units within the same layer of the hierarchy to influence each other, thus allowing some consideration of context. This feedback also
enables the nodes to have a much wider support on the original image, while keeping
the number of parameters small. The bounded feedback they employ is equivalent to
a deeper feed-forward neural network, so it is not really relying on attractor dynamics.
An interesting experiment would be to extend this by using the full feedback of an
attractor network.

5.4 Numerical experiment

A small program was written to test the adjoint-based optimization algorithm described
in this section. A network with a ring structure was generated (see Figure 10) and
initialized with random weights and biases. The parameters were chosen in such a way
that the network was guaranteed to begin at a stable point. The optimization problem
was to “invert” the initial fixed-point of the network, as we describe below.

The parameters (weights and biases) of the network were initialized as follows.
The bias at each node was drawn from a normal distribution with mean 0. The weight
along each edge was either $-1$ or 1, with equal probability assigned to each. Instead of
the sigmoid function, the activation function used was $\phi(x) = \frac{3}{4} \sin(x)$. This choice
was based on the hypothesis that using a periodic function would require more accurate
gradients, and would lead to more contrasts in experiments that varied the settings of
the gradient estimation process.

We now describe the optimization objective that was placed on the model. The
initial parameters $(w_0, b_0)$ determine a fixed-point $x^*(0)$. Set $t = -x^*(0)$. We set the
function $e$ to be the squared distance of a vector to $t$, i.e. $e(x) = \|x - t\|^2$, and we
define the overall optimization problem as

$$\min_w e(x^*(w))$$
From this we can see that a perfect solution to this problem is a $w$ that results in a fixed-point where each component is the negative of the initial fixed point, hence we call it “inverting the stable point”.

Using this objective, optimization proceeded as described in Section 5.3. We found that optimization worked well for small values of $\epsilon$ (see Figure 11.) Figure 12 shows how the quality of gradient estimates deteriorates as the step sizes increases. Each row of that figure shows data gathered from an experiment using successively higher step sizes beginning from $\epsilon = 0.005$ (top row) and going up to $\epsilon = 0.2$ (bottom row). At each iteration of optimization the angle between the true adjoint and approximate adjoint, as estimated by the auxiliary process, was calculated and plotted in the figure. We can see that angles are very bad after experiment number 15, which is near $\epsilon = 0.075$.

6 Chemical reaction networks

In this section we describe a hypothetical application of a multiple timescale optimization algorithm to chemical reaction networks. Say a scientist has a reproducible chemical reaction, and they know how to measure the equilibrium concentrations and set the initial concentrations. They also have some idea of the structure of the reactions, meaning they know what chemical species are involved and what interactions occur, but they don’t know what the rates of the reactions are. One approach to recovering these parameter can be to run many experiments, each time varying the initial concentrations, to generate lots of data, and then attempting to find a reaction matrix $w$ that is consistent with those measurements using gradient based optimization. This could be used for personalized medicine. For example, the metabolic processes tak-
Figure 12: Plot indicating how performance of gradient estimation deteriorates as the step size becomes longer. Best viewed in color. Each row corresponds to a different setting of the step size, from $\epsilon = 0.005$ (top row) to $\epsilon = 0.2$ (bottom row). The columns correspond to iteration numbers, and the color of a cell reflects the (cosine of the) angle between the approximate and true gradients, according to the legend on the right.

ing place in each individual likely involves the same chemicals and reactions but there may be slightly different coefficients from one person to the next, and being able to recover these coefficients could be very useful for a doctor. This would allow a more personalized approach to treatment.

6.1 Model

The dynamic system that we consider optimizing is related to an algorithm for computing equilibrium concentrations in a certain class of chemical reaction networks. There is a one-to-one correspondence between the parameters and a class of chemical reaction networks, and the fixed-point of the algorithm can be identified with the equilibrium concentration of a chemical reaction network. Then certain problems of optimizing the equilibrium of a chemical reaction network can be translated into problems of optimizing the fixed-point of this algorithm.

We now describe the type of chemical reaction network under consideration, the heterodimerization networks. In this class, the two types of reactions that can occur
are (1) pairs of “simple” species $X_i, X_j$ combine to form the corresponding complex species $X_{\{i,j\}}$ at rate $e^{w_{i,j}}$:

$$X_i + X_j \xrightarrow{e^{w_{i,j}}} X_{\{i,j\}}$$ (39)

and (2) the complex species degrade at unit rate into their constituents

$$X_{\{i,j\}} \xrightarrow{1} X_i + X_j$$ (40)

The theory of continuous-time mass-action kinetics specifies a flow on the concentration variables $\{x_i\}, \{x_{\{i,j\}}\}$, from the formal equations (39, 40). According to the resulting ODE’s, the equilibrium concentration must satisfy

$$\sum_{j:\{i,j\} \in D} x_i x_j = \sum_{j:\{i,j\} \in D} e^{w_{i,j}} x_{\{i,j\}}$$ (41)

$$x_{\{i,j\}} = e^{w_{i,j}} x_i x_j$$ (42)

where $D$ is the set pairs of simple species which react with each other. Furthermore, there is a conservation law that says for each simple species $i$ the quantity

$$\hat{b}_i(t) := x_i(t) + \sum_{j:\{i,j\} \in D} x_{\{i,j\}}(t)$$ (43)

remains constant for all $t > 0$. The $\hat{b}_i$ is referred to as the total concentration for species $i$. Using the above equations (41), (42), and (43), one can derive an iterative method with very nice properties to compute the equilibrium concentrations. We now describe this algorithm. Denote by $\text{Sym}_n(\mathbb{R})$ the set of $n \times n$ symmetric matrices with real-valued entries. For a vector $b \in \mathbb{R}^n$ define $\mathbb{R}^n_{\leq b} = \{x \in \mathbb{R}^n | x_i \leq b_i, 1 \leq i \leq n\}$. Remarkably, it was shown in [23] (see also [24]) that the components of the equilibrium for the simple species may be calculated as the fixed-point of the map $F : \mathbb{R}^N_{>0} \times \text{Sym}_n(\mathbb{R}) \times \mathbb{R}^N_{>0} \to \mathbb{R}^N_{>0}$ given by

$$F_i(x; w, \hat{b}) = \frac{\hat{b}_i}{1 + \sum_{j:\{i,j\} \in D} x_j e^{w_{i,j}}}$$ (44)

Note that the concentrations of the complex species can be recovered, if needed, by equation (42). More specifically, [23] showed that the map $F$ is a contraction in the Thompson metric $d(u, v) = \| \log u - \log v \|_{\infty}$. We let $x^*(w, \hat{b})$ denote the fixed-point of (44):

$$x^*(w, \hat{b}) = F(x^*(w, \hat{b}); w, \hat{b})$$

To get a more convenient representation of the function $F$, we change coordinates and work with image of $F$ under the log map. This is $f = \log \circ F \circ \exp$. Explicitly, the function is $f : X \times W \times X \to X$ where $X = \mathbb{R}^{n}_{\leq b}$ and $W = \text{Sym}_n(\mathbb{R})$ and the $n$ component functions $f_i : X \times W \times X \to \mathbb{R}_{\leq b_i}$ are given by

$$f_i(x; w, b) = b_i - \log \left( 1 + \sum_{j \neq i} e^{w_{i,j} + x_j} \right)$$ (45)
Let \( c^* (w, b) \) denote the fixed-point of (45), meaning
\[
c^* (w, b) = f (c^* (w, b); w, b)
\]
and recall that \( x^* (\hat{w}, \hat{b}) \) denotes the fixed-point of (44). The two are then related by
\[
\exp(c^* (w, \log \hat{b})) = x^* (w, b)
\] (46)
The next result formally states the contraction property of \( f \).

**Proposition 6.1.** For any \( n \times n \) matrix \( w \) and any \( b \in \mathbb{R}^n \) the map \( f \) is contraction in the norm \( \| \cdot \|_\infty \) on the set \( \mathbb{R}^n_{\leq b} = \{ x \in \mathbb{R}^n | x_i \leq b_i, 1 \leq i \leq n \} \).

**Proof.** Note that the Jacobian of \( f \) is
\[
\frac{\partial f_i}{\partial x_j} (x, w) = -e^{w_i,j x_j} + 1 + \sum_{j \neq i} e^{w_i,k x_k}
\]
By the definition of \( f \), we may assume each \( x_i < b_i \). By definition of \( \| \cdot \|_\infty \), we have
\[
\left\| \frac{\partial f}{\partial x} (x, w) \right\|_\infty = \max_{1 \leq i \leq n} \frac{\sum_{j \neq i} e^{w_i,j x_j}}{1 + \sum_{j \neq i} e^{w_i,k x_k}} = M < 1
\]
where \( M = \max_i \sum_{j \neq i} e^{w_i,j x_j} \). Note that \( M \) satisfies
\[
M \leq \left( \max_i \sum_{j \neq i} e^{w_i,j} \right) \| e^b \|_\infty
\]
where the term in parenthesis is the \( \| \cdot \|_\infty \) norm of the matrix \( A \) with \( A_{i,i} = 0 \) and \( A_{i,j} = e^{w_i,j} \). \qed

### 6.2 Optimization problem

Let \( y^i, i = 1, 2, \ldots, m \) be concentration vectors. Each vector specifies concentrations for the simple species, \( y_i^j \) and for complex species \( y_{j,k}^i \). These could be obtained, for example, by running multiple experiments, each with different initial conditions. The problem is to find the reaction weights \( w \) for a heterodimerization network that has these concentration vectors as equilibria. As we discussed earlier, the equilibrium concentration reached by a heterodimerization network is determined by the reaction rates \( w \) and the total concentration vector \( b \). We can denote this concentration vector by \( x^*(w, b) \). Let \( z^i \) be the vector of total concentrations for the concentration vector \( y^i \); this is \( z^i_j = y^i_j + \sum_{k: \{j,k\} \in D} y^i_{j,k} \). Let \( e^i(x) \) be a cost incurred when the equilibrium concentration is \( x \) for a reaction that begins with total concentrations \( z^i \). A reasonable objective, then, is to minimize
\[
O_{CHEM}(w) = \sum_{i=1}^{K} e^i (x^*(w, z^i)) \] (47)
One possible choice for the \( e^i \) is \( e^i(x) = \| x - y^i \|^2 \). We describe another choice for these error functions in our experiments section below.
6.3 Optimization algorithm

According to Equation 46, we know that \( x^*(w, z^i) \) can be defined in terms of the fixed-point algorithm: If \( c^*(w, z^i) \) is the output of the fixed-point algorithm, then for the simple species one has

\[
x^*(w, z^i)_j = \exp(c^*(w, \log z^i))_j
\]

Combining this with Equation (42) we obtain a correspondence for the complex species:

\[
x^*(w, z^i)_{j,k} = \exp(c^*(w, \log z^i))_j \exp(c^*(w, \log z^i))_k e^{w_{j,k}}
\]

Then the objective (47) can be written as

\[
O_{CHEM}(w) = \sum_{i=1}^{K} e^i (\exp(c^*(w, \log z^i)), y)
\]

with the convention that \( \exp \) is applied coordinate-wise to its vector argument. Thus we have found a representation of the problem as that of optimizing the fixed-point of a globally attractive system. In addition to the contraction properties, it’s clear that the various derivatives of \( f \) are well-defined. Then one may apply the adjoint optimization procedure of Section 5.3 to this problem.

If one has some idea of the structure of the chemical reactions and a good starting point for the reaction rates (specified by \( w \)), this procedure can fine-tune the reaction rates using data gathered from experiments. Alternatively, one could use the procedure starting from almost no idea of the chemical reaction network, leading to a “virtual chemical network”, consisting of a set of species and the reaction rules between them, that can approximately realize desired equilibrium concentrations. Of course, given an arbitrary formal chemical reaction network, one must find how to realize these equations in a physical process. But as the work of [25] suggests, it may be possible to realize such virtual chemical reaction networks using reprogrammed DNA.

6.4 Numerical experiment

In this experiment, we are given a heterodimerization network with 5 simple species. All simple species may react with each other; thus the parameter \( w \) is a 5 \( \times \) 5 matrix. We consider the network behavior over 4 possible total-concentrations. Letting \( \delta \) be small positive number (we took \( \delta = e^{-5} \)), they are

\[
\begin{align*}
    z^1 &= (\delta, \delta, 1, 1, 1) \\
    z^2 &= (\delta, 1, 1, 1, 1) \\
    z^3 &= (1, \delta, 1, 1, 1) \\
    z^4 &= (1, 1, 1, 1, 1)
\end{align*}
\]

For each of these total concentrations, the objectives were as follows:

\[
e^1(x) = x_4 - x_5
\]
\[ e^2(x) = x_5 - x_4 \]
\[ e^3(x) = x_5 - x_4 \]
\[ e^4(x) = x_4 - x_5 \]

Physically this has the following interpretation: Consider that one has the ability to start the reaction from only simple species, without having any complex species present. The presence of the pair \((z^1, e^1)\) in our problem means the following. When starting from a very low concentration of simple species \((x_1, x_2)\), the network should tend toward a state where species \(x_5\) is more prevalent then species \(x_4\). The pair \((z^2, e^2)\) means that starting from a configuration where \(x_1\) is nearly absent but \(x_2\) is prevalent should drive the network to an equilibrium state where \(x_4\) is more prevalent than \(x_5\). And so on.

We applied the adjoint-based optimization method, as described in Section 5.3. Figure 13 has a visualization of the result. The figure shows how the network processes the total concentrations of interest, before and after optimization. We used a constant step size \(\epsilon = 0.4\). The matrix \(w\) was initialized by choosing the entries at random from the uniform distribution on \([-0.1, 0.1]\). The optimization procedure was run for 1575 iterations. The procedure was able to find the reaction rates that met our criteria: The first and last inputs, \(z^1\) and \(z^4\), result in concentrations of \((x_4, x_5)\) where \(x_4 < x_5\), while the inputs \(z^2\) and \(z^3\) cause equilibria with the property \(x_5 < x_4\).

7 Conclusion

In this survey we reviewed a number of network based models relevant for machine learning. We described how these models are used in practice, and the associated optimization problems. One thing all these models have in common is that computing the relevant gradients is difficult, due to feed-back effects, and one must resort to approximation methods. These approximation methods suggest a two-timescale approach to optimization, where derivative estimation and parameter adaptation happen in parallel. In addition to presenting some models well known in machine learning, including the Boltzmann machine, Sigmoid belief networks, and attractor networks, we also suggested a novel application to chemical reaction networks.

A common framework to describe and analyze these algorithms is given by Stochastic Approximation. Stochastic Approximation provides some conditions under which the trajectory of an optimization algorithm approaches that of a corresponding deterministic, continuous time, gradient system as the step size \(\epsilon\) tends to 0. We also reviewed some applications of Stochastic Approximation to online Bayesian learning.
Figure 13: Equilibrium concentrations corresponding to different total concentrations, before and after optimization. The plots (a-d) show the equilibrium concentrations of the simple species \((x_4, x_5)\), corresponding to the total concentration vectors \(z^1, \ldots, z^4\), respectively, using the initial, random, reaction rates. The second column plots the equilibrium concentrations of the simple species \((x_4, x_5)\) after optimization, for the corresponding inputs.
Bibliography


