Learning Deep Boltzmann Machines using Adaptive MCMC

Ruslan Salakhutdinov
Brain and Cognitive Sciences and CSAIL, MIT
77 Massachusetts Avenue, Cambridge, MA 02139 RSALAKHU@MIT.EDU

Abstract

When modeling high-dimensional richly structured data, it is often the case that the distribution defined by the Deep Boltzmann Machine (DBM) has a rough energy landscape with many local minima separated by high energy barriers. The commonly used Gibbs sampler tends to get trapped in one local mode, which often results in unstable learning dynamics and leads to poor parameter estimates. In this paper, we concentrate on learning DBM's using adaptive MCMC algorithms. We first show a close connection between Fast PCD and adaptive MCMC. We then develop a Coupled Adaptive Simulated Tempering algorithm that can be used to better explore a highly multimodal energy landscape. Finally, we demonstrate that the proposed algorithm considerably improves parameter estimates, particularly when learning large-scale DBM's.

1. Introduction

A Deep Boltzmann Machine (DBM) is a type of binary pairwise Markov Random Field (MRF) with multiple layers of hidden random variables. Maximum likelihood learning in DBM's is very difficult because of the hard inference problem induced by the partition function. Moreover, multiple layers of hidden units make learning in DBM's far more difficult. Recently, (Salakhutdinov & Hinton, 2009) proposed a variational approach, where inference over the states of the hidden variables is performed using variational approaches, such as mean-field. Learning can then be carried out by applying a stochastic approximation procedure that uses Markov chain Monte Carlo (MCMC) in order to approximate the gradients of the intractable partition function.

When modeling high-dimensional structured data, the distribution we need to infer is likely to be highly multimodal. This is common when modeling real-world distributions, in which exponentially large number of possible input configurations have extremely low probability, but there are many very different inputs that occur with similar probabilities. When learning DBM's, or general MRF's, canonical MCMC algorithms, such as the plain Gibbs sampler or Metropolis-Hasting algorithm, often get trapped in one local mode. The inability of the Markov chains to efficiently move around the energy landscape often results in unstable learning dynamics and consequently leads to poor parameter estimates (Hinton et al., 2003; Tieleman & Hinton, 2009; Salakhutdinov, 2010).

In this paper we propose a novel learning algorithm, Coupled Adaptive Simulated Tempering, that is based on the well-studied class of adaptive MCMC algorithms, and show that it is able to better traverse a highly multimodal energy landscape. The proposed algorithm is only twice as slow as the original learning algorithm that uses the plain Gibbs sampler, yet it considerably improves parameter estimates, particularly when learning in large-scale DBM's. Finally, we also provide a conceptual connection between the recent "Fast PCD" algorithm (Tieleman & Hinton, 2009), aimed at alleviating the problem of poor mixing, and adaptive MCMC algorithms.

2. Deep Boltzmann Machines

A Deep Boltzmann Machine is a network of symmetrically coupled stochastic binary units. It contains a set of visible units $\mathbf{v} \in \{0,1\}^D$, and a set of hidden units forming multiple layers $\mathbf{h}^1 \in \{0,1\}^{F_1}$, $\mathbf{h}^2 \in \{0,1\}^{F_2}$,..., $\mathbf{h}^L \in \{0,1\}^{F_L}$.

Consider a Deep Boltzmann Machine with two hidden layers[1] (see Fig. 1), with no within-layer connections.

Appearing in *Proceedings of the 27th International Conference on Machine Learning*, Haifa, Israel, 2010. Copyright 2010 by the author(s)/owner(s).

[1]Extensions to models with more than two layers is

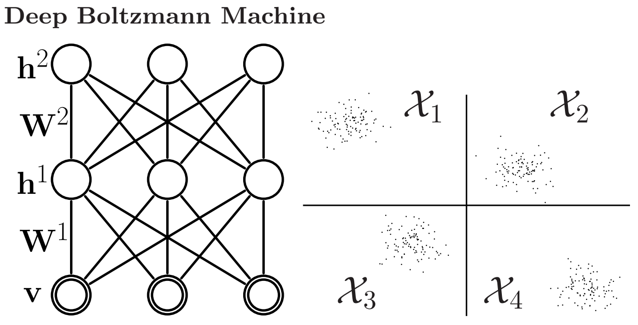

Figure 1. **Left:** Deep Boltzmann Machine: All connections between layers are undirected. **Right:** An example of the state space partition. The goal is to construct a Markov chain that would spend an equal amount of time in each partition.

The energy of the state $\{\mathbf{v}, \mathbf{h}\}$ is defined as:

$$E(\mathbf{v}, \mathbf{h}; \theta) = -\mathbf{v}^\top \mathbf{W}^1 \mathbf{h}^1 - \mathbf{h}^{1\top} \mathbf{W}^2 \mathbf{h}^2, \qquad (1)$$

where $\mathbf{h} = \{\mathbf{h}^1, \mathbf{h}^2\}$ are the set of hidden units, and $\theta = \{\mathbf{W}^1, \mathbf{W}^2\}$ are the model parameters, representing visible-to-hidden and hidden-to-hidden symmetric interaction terms[2]. The probability that the model assigns to a visible vector $\mathbf{v}$ is:

$$P(\mathbf{v}; \theta) = \frac{P^*(\mathbf{v}; \theta)}{\mathcal{Z}(\theta)} = \frac{1}{\mathcal{Z}(\theta)} \sum_{\mathbf{h}} \exp\left(-E(\mathbf{v}, \mathbf{h}^1, \mathbf{h}^2; \theta)\right).$$

The derivative of the log-likelihood with respect to parameter vector W^1 takes the following form:

$$\frac{\partial \log P(\mathbf{v}; \theta)}{\partial W^1} = \mathrm{E}_{P_{\text{data}}}[\mathbf{v}\mathbf{h}^{1\top}] - \mathrm{E}_{P_{\text{model}}}[\mathbf{v}\mathbf{h}^{1\top}], \quad (2)$$

where $\mathrm{E}_{P_{\text{data}}}[\cdot]$ denotes an expectation with respect to the completed data distribution $P_{\text{data}}(\mathbf{h}, \mathbf{v}; \theta) = P(\mathbf{h}|\mathbf{v}; \theta) P_{\text{data}}(\mathbf{v})$, with $P_{\text{data}}(\mathbf{v}) = \frac{1}{N}\sum_n \delta(\mathbf{v} - \mathbf{v}_n)$ representing the empirical distribution, and $\mathrm{E}_{P_{\text{model}}}[\cdot]$ is an expectation with respect to the distribution defined by the model. The derivatives with respect to parameters W^2 take similar form.

Exact maximum likelihood learning in this model is intractable, because both data-dependent and model-dependent expectations cannot be computed analytically in less than exponential time. We now briefly review the learning algorithm for DBM's. For more details refer to (Salakhutdinov & Hinton, 2009).

2.1. Variational Learning

Consider any approximating distribution $Q(\mathbf{h}|\mathbf{v}; \boldsymbol{\mu})$ for the posterior $P(\mathbf{h}|\mathbf{v}; \theta)$. The log-likelihood of our DBM model then has the following variational lower bound:

$$\log P(\mathbf{v}; \theta) \geq \sum_{\mathbf{h}} Q(\mathbf{h}|\mathbf{v}; \boldsymbol{\mu}) \log P(\mathbf{v}, \mathbf{h}; \theta) + \mathcal{H}(Q), (3)$$

where $\mathcal{H}(\cdot)$ is the entropy functional. In the mean-field approach, a fully factorized distribution is chosen in order to approximate the true posterior. $Q(\mathbf{h}; \boldsymbol{\mu}) = \prod_i q(h_i)$, with $q(h_i = 1) = \mu_i$. The learning proceeds by first maximizing the lower bound with respect to the variational parameters $\boldsymbol{\mu}$ for fixed θ. Given the variational parameters $\boldsymbol{\mu}$, the model parameters θ are then updated using stochastic approximation.

2.2. Stochastic Approximation Algorithm

Stochastic approximation procedure (SAP), also called Persistent CD (PCD) (Tieleman, 2008), belongs to the general class of well-studied stochastic approximation algorithms of the Robbins-Monro type (Younes, 1988; Robbins & Monro, 1951). Let θ^t and $\mathbf{x}^t$ be the current setting of the parameters and the state. Then the state and the parameters are updated sequentially: Given $\mathbf{x}^t$, we sample a new state $\mathbf{x}^{t+1}$ using the transition operator $T_{\theta^t}(\mathbf{x}^{t+1} \leftarrow \mathbf{x}^t)$ that leaves $p(\cdot; \theta^t)$ invariant. For DBM's we use Gibbs sampler. A new parameter θ_{t+1} is then obtained by making a gradient step, where the intractable model's expectation $\mathrm{E}_{P_{\text{model}}}[\cdot]$ is replaced by a point estimate at sample $\mathbf{x}_{t+1}$. In practice, we typically keep several Markov chains, which we will sometimes refer to as persistent chains. We will also refer to the current state in each of these chains as a sample particle.

Almost sure convergence guarantees of SAP to an asymptotically stable point are given in (Younes, 2000; Yuille, 2004). One necessary condition requires the learning rate to decrease with time, i.e. $\sum_{t=0}^{\infty} \alpha_t = \infty$ and $\sum_{t=0}^{\infty} \alpha_t^2 < \infty$, which can be easily satisfied by setting $\alpha_t = 1/(t_0 + t)$. The proof of convergence relies on the following basic observation: As the learning rate becomes sufficiently small compared to the mixing rate of the Markov chain, the chain will stay close to its stationary distribution, even if it is only run for a few MCMC steps per parameter update.

3. Learning vs. Mixing

The proof of convergence of stochastic approximation, however, does not tell us why the algorithm works well when using much larger learning rates. Indeed, when looking at the behavior of SAP in practice, the algorithm makes very rapid progress towards finding a good region in the parameter space, even though the Markov chain stays far from its stationary distribution.

straightforward.

[2]We omit the bias terms for clarity of presentation.

Recently, (Tieleman & Hinton, 2009) made a notable observation that there is a delicate interplay between the mixing rate of the persistent Markov chain and parameter learning. In particular, it was empirically demonstrated that the energy surface is being changed during learning in a way that improves the mixing rate of the Markov chain. Note that the learning rule of Eq. 2 attempts to lower the energy in the vicinity of the data while raising the energy at the places where the sample particles reside. Therefore large changes in energy landscape cause sample particles to quickly move away from their currently residing modes[3]. This in turn allows us to do learning using much larger learning rates than would be possible if the persistent chain needed to stay close to its stationary distribution. This key observation emphasizes the fact that inference strongly interacts with learning.

As the learning proceeds, it is necessary to decrease the learning rate in order to avoid strong oscillations in the parameter estimates. This, however, reduces the changes made to the energy landscape, which causes persistent chains to mix poorly. When learning Restricted Boltzmann Machines (RBM's), (Desjardins et al., 2010; Salakhutdinov, 2010) show that this can often lead to poor generative models. The problem of poor mixing becomes considerably worse when learning Deep Boltzmann Machines.

One way to encourage the sample particles to mix would be to use Fast PCD (FPCD) algorithm (Tieleman & Hinton, 2009). In addition to the regular parameters $\theta = \{\mathbf{W}^1, \mathbf{W}^2\}$, also termed "slow weights", FPCD uses a set of "fast weights" $\theta_f = \{\mathbf{W}^1_f, \mathbf{W}^2_f\}$. Slow weights represent our standard generative model and they are used to estimate data-dependent expectations. The role of the fast weights is to improve mixing. In particular, sample particles are updated using a persistent Markov chain with parameters $\theta + \theta_f$. To facilitate mixing, fast weights have a large fixed learning rate, which is different from the learning rate used by the slow weights[4]. As training progresses, slow weights can be fine-tuned by decreasing its learning rate, whereas fast weights will still ensure that the particles move around the energy landscape quickly.

While it has been shown empirically that fast weights tend to improve mixing of the sample particles, it is not clear what distribution FPCD is sampling from and what kind of biases fast weights introduce into the learning algorithm. We next describe the Wang-Landau algorithm, which belongs to the class of adaptive MCMC methods, and show that it is very similar in spirit to the idea of introducing fast weights in order to improve mixing.

4. Wang-Landau (WL) Algorithm

Suppose that our target distribution, given by $p(\mathbf{x};\theta) = {}^1\!/\!_\mathcal{Z} \exp(-E(\mathbf{x};\theta))$, is defined over some state space $\mathcal{X}$. For the case of a two-layer DBM, we have $\mathcal{X} = \{\mathbf{v}, \mathbf{h}^1, \mathbf{h}^2\}$. Consider partitioning the state space into K disjointed sets $\{\mathcal{X}_k\}_{k=1}^K$. The goal is to construct a Markov chain that would spend an equal amount of time in each partition and where the moves within each partition are carried out using standard Metropolis-Hastings or Gibbs updates (see Fig 1, right panel). The main difficulty here lies in estimating a set of weights that would properly reweight the probability in each partition to ensure that the chain spends the same amount of time in each set $\mathcal{X}_k$. A recent breakthrough, introduced by (Wang & Landau, 2001), solves both the weight estimation and sampling problem in a single run. Their proposed algorithm is very reminiscent of the stochastic approximation algorithm.

Let $\mathbf{g}$ be a vector of length K, that contains our current unnormalized weight estimates. At time $t = 0$, all elements of $\mathbf{g}^t$ are initialized to 1. The algorithm proceeds as follows:

- Given $\mathbf{x}^t$, sample a new state $\mathbf{x}^{t+1}$ from transition operator $T_{\mathbf{g}^t}(\mathbf{x}^{t+1} \leftarrow \mathbf{x}^t)$, whose invariant probability distribution is proportional to:

$$p(\mathbf{x};\theta,\mathbf{g}^t) \propto \sum_{k=1}^{K} \frac{p^*(\mathbf{x};\theta)}{g_k^t} \mathrm{I}(\mathbf{x} \in \mathcal{X}_k). \tag{4}$$

- Update weights:

$$g_k^{t+1} = g_k^t(1 + \gamma_t \mathrm{I}(\mathbf{x}^{t+1} \in \mathcal{X}_k)), \tag{5}$$

where I is the indicator function, and $\gamma_t > 0$ is known as the weight adapting factor. As the chain stays in the set $\mathcal{X}_k$, the weight g_k will increase, exponentially increasing the probability of moving outside $\mathcal{X}_k$.

The algorithm was further generalized to general state spaces (Atchade & Liu, 2004) and can be viewed and analyzed within the framework of adaptive MCMC and stochastic approximation algorithms. In particular, under certain regularity conditions, as $t \to \infty$ and $\gamma_t \to 0$, for all $k \in \{1, 2, ..., K\}$ and any measurable

[3]Unless sample particles are perfectly balanced by the presence of the training data at those modes, in which case the gradient of Eq. 2 is zero.

[4]Fast weights also have a strong weight decay.

function f, we have convergence in probability:

$$\frac{g_k^t}{\sum_i g_i^t} \rightarrow p(\mathbf{x} \in \mathcal{X}_k), \qquad (6)$$

$$\frac{1}{t_k}\sum_{i=1}^{t} f(\mathbf{x}_i)\mathrm{I}(\mathbf{x}_i \in \mathcal{X}_k) \rightarrow \sum_{\mathbf{x}\in\mathcal{X}_k} f(\mathbf{x})\frac{p(\mathbf{x})}{p(\mathbf{x}\in\mathcal{X}_k)},$$

where $t_k = \sum_i \mathrm{I}(\mathbf{x}_i \in \mathcal{X}_k)$. This result states that asymptotically, the sequence of weights converges to the set of optimal weights, so that the Markov chain spends an equal amount of time in each partition. More importantly, the Monte Carlo estimates converge to the correct limits as well.

The idea behind the WL algorithm is very similar in spirit to the idea of FPCD. The log of the weights $\log \mathbf{g}$ effectively raise the energy of the partition $\mathcal{X}_k$, so as to encourage the sample particle $\mathbf{x}^t$ to move out of the current partition. We now show that connection more explicitly.

4.1. Connection to FPCD

Consider a binary Markov random field with only two nodes $\mathbf{x} = \{x_1.x_2\}$, each of which can take on the value of 1 or -1. Let us partition the state space into 4 sets, each containing only a single point: $\mathcal{X}_1 = \{1, 1\}$, $\mathcal{X}_2 = \{-1, 1\}$, $\mathcal{X}_3 = \{-1, 1\}$, and $\mathcal{X}_4 = \{-1, -1\}$. The probability that the model assigns to a vector $\mathbf{x}$ is:

$$p(\mathbf{x}; \theta) = \frac{1}{\mathcal{Z}}\sum_{k=1}^{4} \exp\left(\theta(x_1, x_2)\right)\mathrm{I}(\mathbf{x} \in \mathcal{X}_k). \qquad (7)$$

Given unnormalized weights $\mathbf{g}$, the WL algorithm samples the next state $\mathbf{x}$ from (see Eq. 4):

$$p(\mathbf{x}; \theta, \mathbf{g}) \propto \sum_{k=1}^{4} \exp\left(\theta(x_1, x_2) - \log g_k\right)\mathrm{I}(\mathbf{x} \in \mathcal{X}_k). \quad (8)$$

In this formulation, parameters $\log g_k$ can be viewed as a corresponding set of fast weights, defined by the FPCD algorithm. Given an observation $\mathbf{x}_0$, the update rule for these fast weights in FPCD algorithm can be easily derived from Eq. 2:

$$\log g_k^{t+1} = \log g_k^t + \alpha_t(\mathrm{I}(\mathbf{x}^{t+1} \in X_k) - \mathrm{I}(\mathbf{x}_0 \in \mathcal{X}_k)).$$

Note that if the sample particle $\mathbf{x}^{t+1}$ is in the same partition as the data point, the weight update will be zero and the chain will not attempt to explore other partitions. If $\mathbf{x}^{t+1}$ is in a different partition, the fast weights will adjust so as to push the particle into partition where the data resides. Observe that the update rule of FPCD is very similar to the corresponding update rule of the WL algorithm (Eq. 5):

$$\log g_k^{t+1} = \log g_k^t + \alpha_t(\mathrm{I}(\mathbf{x}^{t+1} \in X_k)), \qquad (9)$$

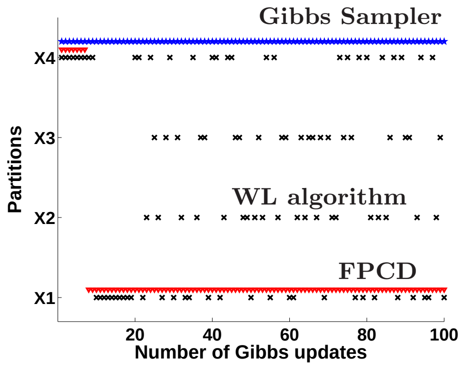

Figure 2. Mixing behavior of different algorithms for a simple binary MRF. Gibbs sampler (blue) and FPCD (red) get stuck in local modes. The WL algorithm (black) spends roughly the same amount of time in each of four partitions.

where $\alpha_t = \log(1 + \gamma_t)$. The key difference is that the data-dependent term is missing, so there is no "data-pulling" bias. The WL algorithm will keep traversing the energy landscape, asymptotically converging to the correct limits. This small, yet significant difference highlights the fact that FPCD cannot be viewed as a valid Monte Carlo algorithm. This connection also suggests that FPCD is unlikely to uncover spurious modes, located far away from the data, because the data-pulling term will always drive sample particles into the modes where the data reside.

Figure 2 precisely illustrates this phenomenon on a simulation experiment. For this toy example we set $\theta(1, 1) = \theta(-1, -1) = 5$, $\theta(-1, 1) = \theta(1, -1) = -5$, and the initial state is $\mathbf{x} = \{-1, -1\} \in \mathcal{X}_4$. As expected, the Gibbs sampler gets stuck in a local mode. The FPCD algorithm quickly jumps into the mode where the data point $\mathbf{x}_0 = \{1, 1\} \in \mathcal{X}_1$ resides, and then stops mixing. The WL algorithm, on the other hand, keeps moving between all four partitions.

In practice, it is often hard to choose a good partition of the state space. For the partition to be efficient, we need to make sure that $p(\mathcal{X}_k)$ are all approximately equal. For the case of tempered MCMC, however, there are natural ways of defining appropriate partitions (Atchade & Liu, 2004; Liang, 2005), which will lead us to the efficient learning algorithm.

5. Adaptive Simulated Tempering

Simulated tempering (Marinari & Parisi, 1992), is a single chain MCMC algorithm, that samples from the joint distribution:

$$p(\mathbf{x}, k) \propto w_k \exp(-\beta_k E(\mathbf{x})), \qquad (10)$$

where w_i are constants, and $0 < \beta_K < \beta_{K-1} < ... < \beta_1 = 1$ are the K "inverse temperatures". The state space is defined as $\cup_{k=1}^{K}\{k\} \times \mathcal{X}$. Conditioned on k, the distribution for $\mathbf{x}$ takes form:

$$p(\mathbf{x}|k) = \frac{1}{\mathcal{Z}_k}\exp(-\beta_k E(\mathbf{x})). \quad (11)$$

A sample from our target distribution $p(\mathbf{x})$ can therefore be obtained by simulating a Markov chain, whose stationary distribution is $p(\mathbf{x}, k)$, and then only keeping the states for which $k = 1$. We may also hope that the high-temperature distributions are more spread out and easier to sample from than our target distribution.

Simulating from the joint $p(\mathbf{x}, k)$ is done by iterating through two transition operators alternately. First, conditioned on k, the state $\mathbf{x}$ is updated according to some transition operator that leaves $p(\mathbf{x}|k)$ invariant (e.g. the Gibbs sampler). Second, conditioned on $\mathbf{x}$, we sample k using Metropolis update rule with a proposal distribution $q(k+1 \leftarrow k) = q(k-1 \leftarrow k) = 1/2$, and $q(2 \leftarrow 1) = q(K-1 \leftarrow K) = 1$.

The choice of the weights w_k greatly affects the efficiency of the algorithm. In particular, if the high-temperature distributions are to facilitate mixing between many local modes, it is necessary for the Markov chain to spend roughly equal amount of time at each temperature value. This can be achieved by letting w_k be proportional to $1/\mathcal{Z}_k$, which are of course unknown.

The WL algorithm solves this problem by partitioning the state space into K sets $\{k\} \cup \mathcal{X}$, each corresponding to a different temperature value. The algorithm is summarized in Algorithm 1. Note that if the move into a different partition (or temperature value) is rejected, the adaptive weight for the current partition will increase, thus exponentially increasing the probability of accepting the next move. As $\gamma_t \to 0$, the ratio of adaptive weights convergence in probability to the ratio of partition functions (see Eq. 6), which guarantees that algorithm will roughly spend the same amount of time at each temperature value.

Algorithm 1 The Wang-Landau algorithm for adaptive simulated tempering.

```
1: Given adaptive weights $\{\mathbf{g}_k\}_{k=1}^{K}$ and the initial configuration of the state $\mathbf{x}^1$ at temperature 1, $k = 1$:
2: **for** $n = 1 : N$ (number of iterations) **do**
3:     Given $\mathbf{x}^n$, sample a new state $\mathbf{x}^{n+1}$ from transition operator that leaves $p(\mathbf{x}|k^n)$ invariant.
4:     Given $k^n$, sample $k^{n+1}$ from proposal distribution $q(k^{n+1} \leftarrow k^n)$. Accept with probability:
           $\min\left(1, \frac{p(\mathbf{x}^{n+1}, k^{n+1})q(k^n \leftarrow k^{n+1})g_{k^n}}{p(\mathbf{x}^{n+1}, k^n)q(k^{n+1} \leftarrow k^n)g_{k^{n+1}}}\right)$
5:     Update adaptive weights:
           $g_i^{n+1} = g_i^n(1 + \gamma_n \mathrm{I}(k^{n+1} \in \{i\})), \quad i = 1, ..., K.$
6: **end for**
7: Obtain (dependent) samples from target distribution $p(x)$ by keeping states for which $k = 1$.
```

6. Learning using Adaptive Simulated Tempering

We can now use adaptive simulated tempering (adaptive ST) algorithm for learning. Given the current state $\mathbf{x}$ of the Markov chain at temperature 1, we can obtain a new state by applying Algorithm 1 for some fixed period of time N, chosen by the user. However, applying adaptive ST after every parameter update within the stochastic approximation algorithm of Sec 2.2 would be expensive. Instead, we could alternate between N plain Gibbs updates and a single run of adaptive ST, where the adaptive weights for subsequent runs would be initialized to their previous values. The proposed algorithm is only twice as slow compared to the standard stochastic approximation, yet it allows the Markov chain to periodically escape from local modes.

We emphasize that the adaptation plays a crucial role in "forcing" the sample particle to move between different temperature values, which greatly facilitates mixing. This allows the chain to make large changes to the current state, producing less correlated samples between successive parameter updates, which significantly improves the accuracy of the estimator. We also note that as the amount of adaptation goes to zero, the transition operator defined by the simulated tempering algorithm is a valid MCMC transition operator. This in turn ensures that the original stochastic approximation will maintain almost sure convergence guarantees.

This simple algorithm already outperforms the plain stochastic approximation procedure (SAP) and FPCD algorithms. However, it is not obvious how to choose N in order to balance between exploration, or waiting until adaptive ST escapes from the local mode, and exploitation, or learning model parameters θ. Instead we propose a Coupled Adaptive Simulated Tempering (CAST) algorithm that avoids this problem altogether.

6.1. Coupled Adaptive Simulated Tempering

Consider two Markov chains, both sampling from the same target distribution $p(\mathbf{x})$. One chain, that we refer to as the "slow" chain, evolves according to simple Gibbs or Metropolis updates. The second chain, called

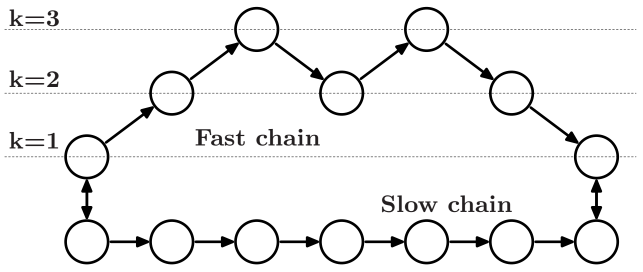

Figure 3. Coupled Adaptive Simulated Tempering.

the "fast" chain, uses adaptive ST. Parameters are updated based on the sample particles obtained from the slow chain, whereas the role of the fast chain is to facilitate mixing. In particular, once the fast chain reaches the state for which $k = 1$, (its temperature value is 1), the state is swapped with the current state of the slow chain, as illustrated in Fig. 3.

CAST, just like adaptive ST, is only twice as expensive compared to the standard stochastic approximation. Unlike adaptive ST, parameters are updated after every Gibbs update, while the fast chain runs in parallel, adaptively mixing between different modes of the energy landscape. As the learning progresses, the model parameters can be fine-tuned by decreasing the learning rate. The adaptive chain, on the other hand, will still ensure that the sample particles continue to explore the energy landscape. Unlike FPCD, the fast chain is likely to visit spurious modes that may reside far away from the data.

7. Previous Work

There has been recent work on using tempered MCMC algorithms for learning in Restricted Boltzmann Machines. (Salakhutdinov, 2010) proposed Trans-SAP algorithm that uses MCMC operators based on tempered transitions (Neal, 1996). Unlike simulated tempering, tempered transitions systematically "move" the sample particle from the original target distribution to the high-temperature distribution, and then back to the original distribution. A new state is accepted based on the Metropolis-Hasting rule. When applied to more complex models, such as DBM's, it is often necessary introduce many intermediate distributions in order to maintain a reasonable acceptance probability and allow the sample particle to move out of the local mode. (Desjardins et al., 2010) further proposed to use parallel tempering, where several coupled chains are run in parallel. The problem here is that running multiple chains, for example 50 as done in (Desjardins et al., 2010), is 50 times slower compared to the plain stochastic approximation. In addition, for large models, there are substantial memory requirements, since at each time step, one needs to store the state of every coupled chain.

The key difference between CAST and the recently proposed methods lies in its adaptive nature. CAST maintains only a single adaptive chain, unlike multiple chains of parallel tempering, or many graded temperature levels in tempered transitions. Its adaptive weights encourage the sample particle to move across multiple temperature levels, while still asymptotically converging to the correct limits.

8. Experimental Results

In our experiments we used the MNIST and NORB datasets. For both datasets, we subdivided the data into minibatches, each containing 100 training cases. For SAP, the number of persistent chains was set to 100. For the CAST algorithm, we maintained 50 slow and corresponding 50 fast chains. To speed-up learning, the number of the mean-field updates was set to 5 and all DBM models were trained using 200,000 weight updates. The initial learning rate α_t was set 0.005 and was decreased as 10/(2000+t).

In practice, for the CAST algorithm, we found it useful to swap states between fast and slow chains after fixed lag intervals (we use 50). The state of the fast chain is taken to be the last state for which $k = 1$. This avoids the problem of continuous swapping between the same states as the fast chain moves around the temperature value of 1.

8.1. MNIST dataset

The MNIST dataset contains 60,000 training and 10,000 test images of ten handwritten digits (0 to 9), with 28×28 pixels. From the training data, a random sample of 10,000 images was set aside for validation.

In our first experiment we trained a two-layer Deep Boltzmann Machine using CAST with 500 and 1000 hidden units in the first and second hidden layers respectively. Fig. 4, left panel, shows consecutive samples (by column) generated from the DBM model after learning is stopped using Gibbs sampler and adaptive ST algorithm. Both methods were randomly initialized and the first 1000 samples were discarded as burn-in. It is clear that the plain Gibbs hardly mixes at all. Adaptive ST, on the other hand, is able to move around the energy landscape far better. For adaptive ST, we used 20 $\beta's$, spaced uniformly from 1 to 0.9. The weight adapting factor was set to $\gamma_t = 10$. The middle panel of the same figure further shows that the adaptive weights push the sample particle in roughly systematic manner through all 20 temperature values, forcing the particle to move away from its local mode.

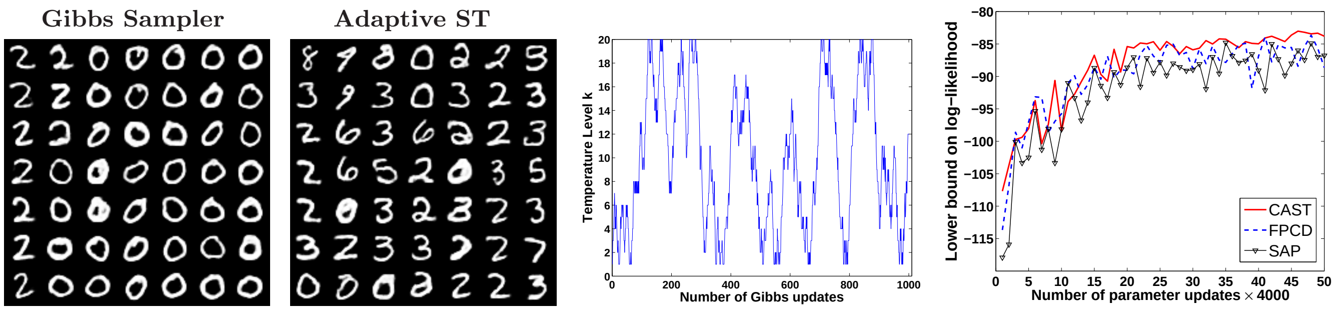

Figure 4. **Left:** Sample particles produced by the Gibbs sampler and adaptive simulated tempering, with 300 Gibbs steps between consecutive images (by column). **Middle** Temperature trajectory of a single particle. Note that adaptive weights are able to push the samples in roughly systematic manner through all 20 temperature values. **Right:** Evolution of the lower bound on the log-likelihood objective for SAP, FPCD, and CAST as learning progresses.

Table 1. The estimates of the variational lower bound on the average test log-probabilities per image for different learning algorithms.

Models	Datasets MNIST	NORB
SAP	-87.23	-596.92
FPCD	-86.72	-597.12
Trans-SAP	-85.41	-595.54
CAST	-84.12	-591.18

In our second experiment, we trained two additional two-layer DBM's using SAP and FPCD. To provide some quantitative assessment of behavior of different algorithms, we estimated the variational lower bound of Eq. 3 on the log-likelihood objective after every 4,000 parameter updates. To estimate the models' partition functions we used Annealed Importance Sampling (Neal, 2001; Salakhutdinov, 2008).

Figure 4, right panel, shows evolution of the bound on the log-likelihood of the training data, as learning progresses. Towards the end of learning, both stochastic approximation and FPCD suffer from large oscillations in parameter estimates. We attribute this behavior to the inability of the Markov chains to explore distributions with many isolated modes. Indeed, persistent chains get stuck in local modes, producing highly correlated samples for successive parameter updates, which leads to the unstable behavior of both algorithms. Strong oscillations in parameter estimates are particularly problematic when learning in RBM's or DBM's, since there does not exist a good stopping criterion. In contrast, CAST exhibits a far more stable behavior, gradually increasing the variational lower bound on the log-likelihood.

Finally, after learning is complete, we also estimated the variational lower bound on the average test log-probability. Table 1 shows results, which also include Trans-SAP, a competitor algorithm based on tempered

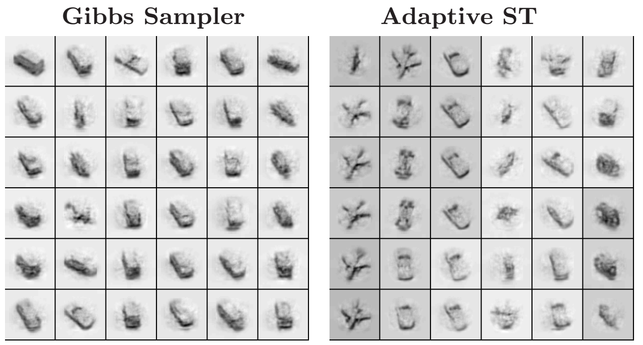

Figure 5. Sample particles produced by the Gibbs sampler and adaptive simulated tempering, with 500 Gibbs steps between consecutive images (by column).

transitions. The plain SAP achieves a lower bound of -87.23. Both FPCD and Trans-SAP improve upon SAP, achieving a lower bound of -86.72 and -85.41 respectively. CAST, on the other hand, achieves a considerably better lower bound of -84.12.

To quantify how loose the variational bound is, we randomly sampled 100 test cases, 10 of each class, and estimated the true test log-probability by running AIS for each test case. The estimate of the true test log-probability was -84.28, whereas the estimate of the variational bound was -84.76. This shows that the bound is rather tight.

8.2. NORB dataset

We now present results on a considerably more difficult NORB dataset. NORB (LeCun et al., 2004) contains images of 50 different 3D toy objects with 10 objects in each of five generic classes: planes, cars, trucks, animals, and humans. Each object is captured from different viewpoints and under various lighting conditions. The training set contains 24,300 stereo image pairs of 25 objects, whereas the test set contains 24,300 stereo pairs of the remaining, different 25 objects. From the training data, 4,300 cases were set aside for validation. To deal with raw pixel data, we followed the approach of (Nair & Hinton, 2009) by first learning a Gaussian-binary RBM with 4000 hidden units, and then treating the the activities of its hidden layer as "preprocessed" data.

We then proceed to training a large two-hidden-layer DBM with each layer containing 4000 hidden units. Figure 5 shows samples generated from the model, using Gibbs and adaptive ST. For adaptive ST, we used 30 $\beta's$, spaced uniformly from 1 to 0.9. Note that the Gibbs sampler simply moves around its local mode, unable to transition into different parts of the energy surface, containing different objects under different viewpoint and lighting conditions. Adaptive ST, on the other hand, is still able to systematically move from one mode to the next.

The ability of the adaptive ST algorithm to escape from local modes allows us to learn much better generative models. Table 1 shows that CAST achieves a lower bound on the average test log-probability of -592.23, improving upon its closest competitor by at least 4 nats.

9. Conclusion

We have developed a new learning algorithm, based on adaptive simulated tempering, for training Deep Boltzmann Machines and showed that it is able to more flexibly traverse a highly multimodal energy landscape. The proposed algorithm is easy to implement and is only twice as slow as the original learning algorithm that uses the plain Gibbs sampler.

Our results on the MNIST and NORB dataset further demonstrate that CAST considerably improves parameter estimates, which allows us to learn much better generative models. More importantly, CAST tends to exhibit a more stable behavior during learning, gradually increasing the variational lower bound on the log-likelihood of the training data. Finally, we believe that the connection between learning dynamics of Boltzmann machines and adaptive MCMC will allow us to further develop better learning techniques and to better understand the interesting interplay between learning and inference.

Acknowledgments

We acknowledge the financial support from NSERC, Shell, and NTT Communication Sciences Laboratory.

References

Atchade, Y. and Liu, S. The Wang-Landau algorithm for Monte Carlo computation in general state spaces. Technical report, University of Ottawa, 2004.

Desjardins, G., Courville, A., Bengio, Y., Vincent, P., and Delalleau, O. Tempered Markov chain Monte Carlo for training of restricted Boltzmann machines. In *AI and Statistics*, pp. 145–152, 2010.

Hinton, Geoffrey E., Welling, Max, and Mnih, Andriy. Wormholes improve contrastive divergence. In *NIPS*. MIT Press, 2003.

LeCun, Y., Huang, F. J., and Bottou, L. Learning methods for generic object recognition with invariance to pose and lighting. In *CVPR (2)*, pp. 97–104, 2004.

Liang, F. Determination of normalizing constants for simulated tempering. *Physica A: Statistical Mechanics and its Applications*, 356(2):468–470, 2005.

Marinari, E. and Parisi, G. Simulated tempering: A new Monte Carlo scheme. *Europhysics Letters*, 19:451–458, 1992.

Nair, V. and Hinton, G. Implicit mixtures of restricted Boltzmann machines. In *Advances in Neural Information Processing Systems*, volume 21, 2009.

Neal, R. Sampling from multimodal distributions using tempered transitions. *Statistics and Computing*, 6:353–366, 1996.

Neal, R. Annealed importance sampling. *Statistics and Computing*, 11:125–139, 2001.

Robbins, H. and Monro, S. A stochastic approximation method. *Ann. Math. Stat.*, 22:400–407, 1951.

Salakhutdinov, R. Learning and evaluating Boltzmann machines. Technical Report UTML TR 2008-002, Department of Computer Science, University of Toronto, 2008.

Salakhutdinov, R. and Hinton, G. Deep Boltzmann machines. In *Proceedings of the International Conference on Artificial Intelligence and Statistics*, volume 5, pp. 448–455, 2009.

Salakhutdinov, R. R. Learning in Markov random fields using tempered transitions. In *Advances in Neural Information Processing Systems*, volume 22, 2010.

Tieleman, T. Training restricted Boltzmann machines using approximations to the likelihood gradient. In *ICML*. ACM, 2008.

Tieleman, T. and Hinton, G.E. Using fast weights to improve persistent contrastive divergence. In *ICML*, pp. 1033–1040. ACM New York, NY, USA, 2009.

Wang, F. and Landau, D. P. Efficient, multiple-range random walk algorithm to calculate the density of states. *Physical Review Letters*, 86(10):2050–2053, 2001.

Younes, L. Estimation and annealing for Gibbsian fields. *Ann. Inst. Henri Poincaré (B)*, 24(2):269–294, 1988.

Younes, L. On the convergence of Markovian stochastic algorithms with rapidly decreasing ergodicity rates, March 17 2000.

Yuille, A. The convergence of contrastive divergences. In *NIPS*, 2004.