Adaptive Parallel Tempering for Stochastic Maximum Likelihood Learning of RBMs

Guillaume Desjardins, Aaron Courville and Yoshua Bengio
Dept. IRO
Université de Montréal
Montréal, QC
{desjagui,courvila,bengioy}@iro.umontreal.ca

Abstract

Restricted Boltzmann Machines (RBM) have attracted a lot of attention of late, as one the principle building blocks of deep networks. Training RBMs remains problematic however, because of the intractibility of their partition function. The maximum likelihood gradient requires a very robust sampler which can accurately sample from the model despite the loss of ergodicity often incurred during learning. While using Parallel Tempering in the negative phase of Stochastic Maximum Likelihood (SML-PT) helps address the issue, it imposes a trade-off between computational complexity and high ergodicity, and requires careful hand-tuning of the temperatures. In this paper, we show that this trade-off is unnecessary. The choice of optimal temperatures can be automated by minimizing average return time (a concept first proposed by (Katzgraber et al., 2006)) while chains can be spawned dynamically, as needed, thus minimizing the computational overhead. We show on a synthetic dataset, that this results in better likelihood scores.

1 Introduction

Restricted Boltzmann Machines (RBM) (Freund & Haussler, 1994; Welling et al., 2005; Hinton et al., 2006) have become a model of choice for learning unsupervised features for use in deep feed-forward architectures (Hinton et al., 2006; Bengio, 2009) as well as for modeling complex, high-dimensional distributions (Welling et al., 2005; Taylor & Hinton, 2009; Larochelle et al., 2010). Their success can be explained in part through the bi-partite structure of their graphical model. Units are grouped into a visible layer \( v \) and a hidden layer \( h \), prohibiting connections within the same layer. The use of latent variables affords RBMs a rich modeling capacity, while the conditional independence property yields a trivial inference procedure.

RBM are parametrized by an energy function \( E(v, h) \) which is converted to probability through the Boltzmann distribution, after marginalizing out the hidden units. The probability of a given configuration \( p(v) \) is thus given by \( p(v) = \frac{1}{Z} \sum_h \exp(-E(v, h)) \), where \( Z \) is the partition function defined as \( Z = \sum_{v, h} \exp(-E(v, h)) \).

Despite their popularity, direct learning of these models through maximum likelihood remains problematic. The maximum likelihood gradient with respect to the parameters \( \theta \) of the model is:

\[
\frac{\partial \log p(v)}{\partial \theta} = - \sum_h p(h|v) \frac{\partial E(v, h)}{\partial \theta} + \sum_{v^-, h^-} p(v^-, h^-) \frac{\partial E(v^-, h^-)}{\partial \theta} \quad (1)
\]

The first term is trivial to calculate and is referred to as the positive phase, as it raises the probability of training data. The second term or negative phase is intractable in most applications of interest,
as it involves an expectation over $p(v, h)$. Many learning algorithms have been proposed in the literature to address this issue:

- **Contrastive Divergence (CD)** (Hinton, 1999; Hinton, 2002) replaces the expectation with a finite set of negative samples, which are obtained by running a short Markov chain initialized at positive training examples. This yields a biased, but low-variance gradient which has been shown to work well as a feature extractor for deep networks such as the Deep Belief Network (Hinton et al., 2006).

- **Stochastic Maximum Likelihood (SML) or Persistent Contrastive Divergence (PCD)** (Younes, 1998; Tieleman, 2008) on the other hand, relies on a persistent Markov chain to sample the negative particles. The chain is run for a small number of steps between consecutive model updates, with the assumption that the Markov chain will stay close to its equilibrium distribution as the parameters evolve. Learning actually encourages this process, in what is called the “fast-weight effect” (Tieleman & Hinton, 2009).

- **Ratio Matching and Score Matching** (Hyvärinen, 2005; Hyvärinen, 2007) avoid the issue of the partition function altogether by replacing maximum likelihood by another learning principle, based on matching the change in likelihood to that implied by the empirical distribution.

(Marlin et al., 2010) recently compared these algorithms on a variety of tasks and found SML to be the most attractive method when taking computational complexity into account. Unfortunately, these results fail to address the main shortcomings of SML. First, it relies on Gibbs sampling to extract negative samples: a poor choice when sampling from multi-modal distributions. Second, to guarantee convergence, the learning rate must be annealed throughout learning in order to offset the loss of ergodicity incurred by the Markov chain due to parameter updates (Younes, 1998; Desjardins et al., 2010). Using tempering in the negative phase of SML (Desjardins et al., 2010; Cho et al., 2010; Salakhutdinov, 2010b; Salakhutdinov, 2010a) appears to address these issues to some extent. By performing a random walk in the joint (configuration, temperature) space, negative particles can escape regions of high probability and travel between disconnected modes. Also, since high temperature chains are inherently more ergodic, the sampler as a whole exhibits better mixing and results in better convergence properties than traditional SML.

Tempering is still no panacea however. Great care must be taken to select the set of temperatures $T = \{T_1, \ldots, T_M; T_1 < T_i < T_M \forall i \in [1, M], M \in \mathbb{N}\}$ over which to run the simulation. Having too few or incorrectly spaced chains can result in high rejection ratios (tempered transition), low return rates (simulated tempering) or low swap rates between neighboring chains (parallel tempering), which all undermine the usefulness of the method. In this work, we show that the choice of $T$ can be automated for parallel tempering, both in terms of optimal temperature spacing, as well as the number of chains to simulate. Our algorithm relies heavily on the work of (Katzgraber et al., 2006), who were the first to show that optimal temperature spacing can be obtained by minimizing the average return time of particles under simulation.

The paper is organized as follows. We start with a brief review of SML, then explore the details behind SML with Parallel Tempering (SML-PT) as described in (Desjardins et al., 2010). Following this, we show how the algorithm of Katzgraber et al. can be adapted to the online gradient setting for use with SML-PT and show how chains can be created dynamically, so as to maintain a given level of ergodicity throughout training. We then proceed to show various results on a complex synthetic dataset.

## 2 SML with Optimized Parallel Tempering

### 2.1 Parallel Tempered SML (SML-PT)

We start with a very brief review of SML, which will serve mostly to anchor our notation. For details on the actual algorithm, we refer the interested reader to (Tieleman & Hinton, 2009; Marlin et al., 2010). RBMs are parametrized by $\theta = \{W, b, c\}$, where $b_i$ is the $i$-th hidden bias, $c_j$ the

\footnote{We use the term “ergodicity” rather loosely, to reflect the amount of time required for the states sampled by the Markov chain, to reflect the true expectation we wish to measure.}
The algorithm works as follows. For $N_s$ sampling updates:

- assign a label to each particle: those swapped into $\beta_1$ are labeled as “up” particles. Similarly, any “up” particle swapped into $\beta_M$ becomes a “down” particle.
• after each swap proposal, update the histograms \( n_u(i), n_d(i) \), counting the number of “up” and “down” particles for the Markov chain associated with \( \beta_i \).
• define \( f_{up}(i) = \frac{n_u(i)}{n_u(i) + n_d(i)} \), the fraction of “up”-moving particles at \( \beta_i \). By construction, notice that \( f_{up}(\beta_1) = 1 \) and \( f_{up}(\beta_M) = 0 \). \( f_{up} \) thus defines a probability distribution of “up” particles in the range \([\beta_1, \beta_M]\).
• The new inverse temperature parameters \( \beta' \) are chosen as the ordered set which assigns equal probability mass to each chain. This yields an \( f_{up} \) curve which is linear in the chain index.

The above procedure is applied iteratively, each time increasing \( N \) incremented at every sampling iteration and (2) reset to 0 whenever \( \tau \) swapped into \( T \). To monitor return time, we can simply maintain a counter \( \tau \). As such, one would expect the optimal set \( T \) while Learning. When training an RBM, the distribution we are sampling from is continuously changing. While the above algorithm exhibits the right properties, it is not very well suited to the context of learning. When training an RBM, the distribution we are sampling from is continuously changing. Our solution is simple: the histograms \( n \) performing \( N \) chain, we update \( i \) as it allows us to maintain flow statistics at the proper timescale. If an “up” particle reaches the 1); \( \forall \}
\{\bar{r}<r<\bar{r}_{min}\}, \text{a hyper-parameter of the algorithm.}

Empirically, we have observed increased stability when the index \( j \) of the new chain is selected such that \( j = \arg\max_i ([f_{up}(i) - f_{up}(i+1)]), i \in [1, M-1] \). To avoid a long burn-in period, we initialize the new chain with the state of the \((j+1)\)-th chain and choose its inverse temperature as the mean \((\beta_j + \beta_{j+1})/2 \). A small but fixed burn-in period allows the system to adapt to the new configuration.
3 Results and Discussion

We evaluate our adaptive SML-PT algorithm (SML-APT) on a complex, synthetic dataset. This dataset is heavily inspired from the one used in (Desjardins et al., 2010) and was specifically crafted to push the limits of the algorithm.

It is an online dataset of 28x28 binary images, where each example is sampled from a mixture model with probability density function \( f_X(x) = \sum_{m=1}^{5} w_m f_{Y_m}(x) \). Our dataset thus consists of 5 mixture components whose weights \( w_m \) are sampled uniformly in the unit interval and normalized to one. Each mixture component \( Y_m \) is itself a random 28x28 binary image, whose pixels are independent random variables having a probability \( p_m \) of being flipped. From the point of view of a sampler performing a random walk in image space, \( p_m \) is inversely proportional to the difficulty of finding the mode in question. The complexity of our synthetic dataset comes from our particular choice of \( w_m \) and \( p_m \).

Large \( w_m \) and small \( p_m \) lead to modes which are difficult to sample and in which a Gibbs sampler would tend to get trapped. Large \( p_m \) values on the other hand will tend to intercept “down” moving particles and thus present a challenge for parallel tempering.

Figure 1(a) compares the results of training a 10 hidden unit RBM, using standard SML, SML-PT with \( \{10, 20, 50\} \) parallel chains and our new SML-APT algorithm. We performed \( 10^5 \) updates (followed by \( 2 \cdot 10^4 \) steps of sampling) with mini-batches of size 5 and tested learning rates in \( \{10^{-3}, 10^{-4}\} \), \( \beta \) learning rates in \( \{10^{-3}, 10^{-4}, 10^{-5}\} \). For each algorithm, we show the results for the best performing hyper-parameters, averaging over 5 different runs. Results are plotted with respect to computation time to show the relative computational cost of each algorithm.

As we can see, standard SML fails to learn anything meaningful: the Gibbs sampler is unable to cope with the loss in ergodicity and the model diverges. SML-PT on the other hand performs much better. Using more parallel chains in SML-PT consistently yields a better likelihood score, as well as reduced variance. This seems to confirm that using more parallel chains in SML-PT increases the ergodicity of the sampler. Finally, SML-APT outperforms all other methods. As we will see in Figure 2, it does so using only 20 parallel chains. Unfortunately, the computational cost seems similar to 50 parallel chains. We hope this can be reduced to the same cost as SML-PT with 20 parallel chains.

\( w = [0.3314, 0.2262, 0.0812, 0.0254, 0.3358] \) and \( p = [0.0001, 0.0137, 0.0215, 0.0223, 0.0544] \)
chains in the near future. Also interesting to note, while the variance of all methods increase with training time, SML-APT seems immune to this issue.

We now compare the various metrics being optimized by our adaptive algorithm. Figure 1(b) shows the average return time for each of the algorithms. We can see that SML-APT achieves a return time which is comparable to SML-PT with 10 chains, while achieving a better likelihood score than SML-PT 50.

We now select the best performing seeds for SML-PT with 50 chains and SML-APT, and show in Figure 2, the resulting $f_{up}(i)$ curves obtained at the end of training.

![Figure 2](image)

(a) SML-APT
(b) SML-PT 50

Figure 2: Return time is minimized by tagging each particle with a label: “up” if the particle visited $T_1$ more recently than $T_M$ and “down” otherwise. Histograms $n_u(i)$ and $n_d(i)$ track the number of up/down particles at each temperature $T_i$. Temperatures are modified such that the ratio $f_{up}(i) = n_u(i) / (n_u(i) + n_d(i))$ is linear in the index $i$. (a) $f_{up}$ curve obtained with SML-APT, as a function of temperature index (blue) and inverse temperature (red). SML-APT achieves a linear $f_{up}$ in the temperature index $i$. (b) Typical $f_{up}$ curve obtained with SML-PT (here using 50 chains). $f_{up}$ is not linear in the index $i$, which translates to larger return times as shown in Fig. 1(b).

The blue curve plots $f_{up}$ as a function of beta index, while the red curves plots $f_{up}$ as a function of $\beta$. We can see that SML-APT results in a more or less linear curve for $f_{up}(i)$, which is not the case for SML-PT. In Figure 3(a) we can see the effect on the pair-wise swap statistics $r_i$. As reported in (Katzgraber et al., 2006), optimizing $T$ to maintain a linear $f_{up}$ leads to temperatures pooling around the bottleneck. In comparison, SML-PT fails to capture this phenomenon regardless of whether it uses 20 or 50 parallel chains (figures 3(b)-3(c)).

![Figure 3](image)

(a) SML-APT
(b) SML-PT 20
(c) SML-PT 50

Figure 3: Pairwise swap statistics obtained after $10^5$ updates. Minimizing return time causes SML-APT to pool temperatures around bottlenecks, achieving large swap rates (0.9) around bottlenecks with relatively few chains. SML-PT on the other hand results in a much flatter distribution, requiring around 50 chains to reach swap rates close to 0.8.
Finally, Figure 4 shows the evolution of the inverse temperature parameters throughout learning. We can see that the position of the bottleneck in temperature space changes with learning. As such, a manual tuning of temperatures would be hopeless in achieving optimal return times.

![Graphical depiction of the set \{\beta_i; i \in [1, M]\}, of inverse temperature parameters used by SML-APT during learning. Temperatures pool around a bottleneck to minimize return time, while new chains are spawned to maintain a given average swap rate. Note that the last 20k updates actually correspond to a pure sampling phase (i.e. a learning rate of 0).](image)

Figure 4: Graphical depiction of the set \(\{\beta_i; i \in [1, M]\}\), of inverse temperature parameters used by SML-APT during learning. Temperatures pool around a bottleneck to minimize return time, while new chains are spawned to maintain a given average swap rate. Note that the last 20k updates actually correspond to a pure sampling phase (i.e. a learning rate of 0).

### 4 Conclusion

We have introduced a new adaptive training algorithm for RBMs, which we call Stochastic Maximum Likelihood with Adaptive Parallel Tempering (SML-APT). It leverages the benefits of PT in the negative phase of SML, but adapts and spawns new temperatures so as to minimize return time. The resulting negative phase sampler thus exhibits greater ergodicity. Using a synthetic dataset, we have shown that this can directly translate to a better and more stable likelihood score. In the process, SML-APT also greatly reduces the number of hyper-parameters to tune: temperature set selection is not only automated, but optimal. The end-user is left with very few dials: a standard learning rate on \(\beta_i\) and a minimum average swap rate \(\bar{r}_{\text{min}}\) below which to spawn.

Much work still remains. In terms of computational cost, we would like a model trained with SML-APT and resulting in \(M\) chains, to always be upper-bounded by SML-PT initialized with \(M\) chains. Obviously, the above experiments should also be repeated with larger RBMs on natural datasets, such as MNIST or Caltech Silhouettes.³.

### Acknowledgments

The authors acknowledge the support of the following agencies for research funding and computing support: NSERC, Compute Canada and CIFAR. We would also like to thank the developers of Theano⁴, for developing such a powerful tool for scientific computing, especially gradient-based learning.

### References


³http://people.cs.ubc.ca/bmarlin/data/index.shtml

⁴http://deeplearning.net/software/theano/


