Using Fast Weights to Improve Persistent Contrastive Divergence

Tijmen Tieleman TIJMEN@CS.TORONTO.EDU
Geoffrey Hinton HINTON@CS.TORONTO.EDU
Department of Computer Science, University of Toronto, Toronto, Ontario M5S 3G4, Canada

Abstract

The most commonly used learning algorithm for restricted Boltzmann machines is contrastive divergence which starts a Markov chain at a data point and runs the chain for only a few iterations to get a cheap, low variance estimate of the sufficient statistics under the model. Tieleman (2008) showed that better learning can be achieved by estimating the model's statistics using a small set of persistent "fantasy particles" that are not reinitialized to data points after each weight update. With sufficiently small weight updates, the fantasy particles represent the equilibrium distribution accurately but to explain why the method works with much larger weight updates it is necessary to consider the interaction between the weight updates and the Markov chain. We show that the weight updates force the Markov chain to mix fast, and using this insight we develop an even faster mixing chain that uses an auxiliary set of "fast weights" to implement a temporary overlay on the energy landscape. The fast weights learn rapidly but also decay rapidly and do not contribute to the normal energy landscape that defines the model.

1. Introduction

RBMis a EBM for unsupervised learning (Hinton, 2002; Smolensky, 1986).

Appearing in *Proceedings of the 26th International Conference on Machine Learning*, Montreal, Canada, 2009.

Standard notation is to use i for indices of visible units, j for indices of hidden units, and w_{ij} for the strength of the connection between the i^{th} visible unit and the j^{th} hidden unit. If v_i denotes the state of the i^{th} visible unit, and h_j denotes the state of the j^{th} hidden unit, an *energy* function is defined on states:

$$E(v,h) = -\sum_{i,j} v_i h_j w_{ij} - \sum_i v_i b_i - \sum_j h_j b_j \quad (1)$$

where b stands for the biases. Through these energies, probabilities are defined as

$$P(v,h) = \frac{e^{-E(v,h)}}{Z} \quad (2)$$

where Z is the normalizing constant

$$Z = \sum_{x,y} e^{-E(x,y)} \quad (3)$$

The probability of a data point (represented by the state v of the visible layer) is defined as the marginal:

$$P(v) = \sum_h P(v,h) = \frac{\sum_h e^{-E(v,h)}}{Z} \quad (4)$$

Thus, the training data likelihood, using just one training point for simplicity, is

$$\phi = \log P(v) = \phi^+ - \phi^- \quad (5)$$

where

$$\phi^+ = \log \sum_h e^{-E(v,h)} \quad (6)$$

and

$$\phi^- = \log Z = \log \sum_{x,y} e^{-E(x,y)} \quad (7)$$

The *positive* gradient $\frac{\partial \phi^+}{\partial w_{ij}}$ is simple:

$$\frac{\partial \phi^+}{\partial w_{ij}} = v_i \cdot P(h_j = 1|v) \quad (8)$$

The *negative* gradient

$$\frac{\partial \phi^-}{\partial w_{ij}} = P(v_i = 1, h_j = 1) \tag{9}$$

however, is intractable. If we could get samples from the model, we could Monte Carlo approximate it, but even getting those samples is intractable.

To get a tractable approximation of $\frac{\partial \phi^-}{\partial w_{ij}}$, one uses some algorithm to *approximately* sample from the model. The Contrastive Divergence (CD) algorithm (Hinton, 2002) is one way to do this. It is designed in such a way that at least the *direction* of the gradient estimate is somewhat accurate, even when the size is not. To estimate $\frac{\partial \phi^-}{\partial w_{ij}}$, the algorithm starts a Markov Chain at one of the training data points used to estimate $\frac{\partial \phi^+}{\partial w_{ij}}$, performs one full Gibbs update, and treats the resulting configuration as a sample from the model.

2. Using a Persistent Markov Chain to Estimate the Model's Expectations

Instead of starting the Gibbs sampling at a random state or at a data point, it is possible to use a "persistent" Markov chain that is not reinitialized each time the parameters are changed. If the learning rate is sufficiently small compared with the mixing rate of the Markov chain, this persistent chain will always stay very close to the stationary distribution even if it is only run for a few Gibbs updates per weight update. Samples from the persistent chain will be highly correlated for successive weight updates, but again, if the learning rate is sufficiently small the chain will mix before the weights have changed enough to significantly alter the unconditional expectations. Many persistent chains can be run in parallel and we shall refer to the current joint state of the hidden and visible units in each of these chains as a "fantasy particle". The effectiveness of this technique for learning general Boltzmann machines was demonstrated by (Neal, 1992). (Tieleman, 2008) showed that, given a fixed amount of computation, RBMs can learn better models using this "Persistent Contrastive Divergence " algorithm , i.e. using persistent Markov chains , compared to using the " standard " CD method (Hinton , 2002) in which each chain is reinitialized to a data point after each weight update.

In this paper, we show two things, one theoretical and one practical. First, there is an important interaction between the mixing rate of a persistent Markov chain and the weight updates. If the persistent chain is sampling from the current stationary distribution of the model, Q_θ, the expected weight updates follow the gradient of the log likelihood[1], but if the chain is sampling from a different distribution, R, there is an additional term in the expected weight updates. This additional term is the gradient of $KL(R||Q_\theta)$ and it moves the model's distribution *away* from R which is bad[2], but this has the effect of making the persistent chain mix much faster which is good. The fact that the energy landscape is being manipulated to improve the mixing rate means that the effectiveness of persistent chains cannot be understood by using any type of analysis that ignores this strong interaction between learning and mixing.

Our practical contribution is to show that we can use an additional set of "fast weights" that learn rapidly, but also decay rapidly, to implement a temporary overlay on the energy landscape. This partially decouples the two uses of the energy landscape: the rapid mixing can be maintained by manipulating the fast weights even when the learning rate of the regular weights that represent the model is decreased towards zero. The introduction of fast weights gives large improvements in the speed with which the regular weights can be learned. This corroborates our theoretical analysis and leads to the most effective learning algorithm so far for RBM's.

3. How Learning Improves the Mixing Rate of Persistent Markov Chains

Consider an RBM that is learning by using a set of persistent Markov chains to provide estimates of the model's expectations. The current states of the M persistent chains define a distribution R which only places probability mass on at most M points and this creates sampling noise. To begin with, suppose that we have a very large number of fantasy particles so that the sampling noise can be ignored. The weight updates exactly[3] follow the negative gradient of the difference of two divergences:

$$C = KL(P||Q_\theta) - KL(R||Q_\theta) \tag{10}$$

[1] This assumes that the data-dependent expectations are estimated correctly, which is easy in a restricted Boltzmann machine.

[2] By contrast, in variational learning (Neal & Hinton, 1998; Jordan et al., 1999), the parameter updates try to move the true posterior distribution over latent variables *towards* the variational approximation.

[3] This is because, unlike in "standard" contrastive divergence learning, the weights do not change R. In standard CD, they do, and CD ignores that effect.

where P is the distribution of the training data and Q_θ is the model's distribution. The first term in Eq. 10 is the negative log likelihood (minus the fixed entropy of P). The second divergence, which is being *maximized w.r.t.* the parameters, measures the departure of the model's distribution from the distribution that the fantasy particles are sampled from.

Provided the gradient of C in equation 10 *w.r.t.* the parameters has a positive cosine with the gradient of $KL(P||Q_\theta)$ the learning will increase the log likelihood of the model. It is not necessary for R to be close to Q_θ. All that is required is that the gradient of $KL(R||Q_\theta)$ is not so large that its projection onto the gradient of $KL(P||Q_\theta)$ is larger than this gradient and in the opposite direction. Of course, this can be ensured by making R extremely close to Q_θ, but in practice this is generally not necessary. This allows much bigger learning rates than would be possible if the persistent chains needed to stay close to the current stationary distribution.

The parameter updates alternate with Gibbs updates of the persistent chains which have the effect of minimizing $KL(R||Q_\theta)$ *w.r.t.* R and this is what stops $KL(R||Q_\theta)$ getting out of hand. The undesirable fact that the weight updates increase $KL(R||Q_\theta)$ is countered by the fact that when $KL(R||Q_\theta)$ is large, the persistent chains will mix fast, thus reducing $KL(R||Q_\theta)$ rapidly.

To better understand the effect of the learning on the mixing, it is helpful to consider how the energy landscape is changed by a set of fantasy particles many of which have become trapped in the same low energy mode. If the fraction of the fantasy particles in that mode exceeds the fraction of the training data in the mode, the net effect of following the negative gradient of C in equation 10 will be to raise the energy of the mode. So the mode will continue to rise until enough of the fantasy particles can escape. By the time enough particles have escaped, the energy of the mode will be too high, thus preventing the fantasy particles from returning in the near future[4]. Fantasy particles develop an aversion to wherever they are and unless this aversion is balanced by the energy-lowering presence of training data, they continually seek out places they have not recently visited.

[4]This has some resemblance to "taboo search" (Cvijovi & Klinowski, 1995), but it is more effective in high-dimensional spaces because it uses the parameters of the energy function to remember where it has been rather than storing points in the high-dimensional state space. For the same reason, it should be a more effective way to balance the heights of widely separated modes in a high-dimensional and highly multi-modal space than the "darting" method described in (Sminchisescu & Welling, 2007)

4. Fast Weights

In addition to the regular weights that parameterize an RBM, we introduce an additional set of "fast weights" that are only used for updating the fantasy particles. For these updates, the effective weights are the sum of the fast and regular weights. Like the regular weights, the fast weights are driven by the difference between the data-dependent expectations and the model's expectations (as estimated by the fantasy particles), but they have much stronger weight-decay and a faster learning rate. The role of the fast weights is to increase the rate at which the combined energy landscape rises in the vicinity of the fantasy particles in order to make them mix faster. As learning progresses, it is helpful to reduce the learning rate of the regular weights (Robbins & Monro, 1951), but this has the unfortunate side effect of reducing the changes in the energy landscape that cause fast mixing. Fast weights overcome this problem. Their learning rate does not decrease as learning progresses, but their fast decay ensures that their effects are only temporary.

Those readers who wish to see a more rigorous analysis of this method can read the paper "Stochastic approximation algorithms: Overview and recent trends" (Bharath & Borkar, 1999). The method has been known in the field of Statistics for some time, and things such as convergence properties have been carefully analyzed.

5. Partially Smoothed Gradient Estimates

Without fast weights, i.e. using PCD, the regular weights have to aid the exploration of the state space. For that reason, it is important that they change rapidly. That is why momentum, or, almost equivalently, averaging several previous gradient estimates when choosing the weight update, hurts performance. However, such averaging does help smooth the noisy gradient estimates. One alternative to PCD is to use such smoothing for the estimates of the positive gradient, to reduce noise, but not use smoothing for the estimates of the negative gradient, to still enable rapid exploration of the state space. This does indeed increase performance over that of PCD, but does not perform as well as FPCD. We call this method PCD PS, for "Persistent Contrastive Divergence, with Partial Smoothing".

6. Pseudocode

Below is a pseudocode description of the PCD algorithm. For simplicity, considerations such as momentum and weight decay have been left out, but these are easy to incorporate.

Program parameters:

- Schedule of regular learning rates (linearly from the initial value to zero, in our experiments)
- Schedule of fast learning rates (constant and equal to the initial regular learning rate, in our experiments)

Initialization:

- Initialize θ-regular to small random values.
- Initialize θ-fast to all zeros.
- Initialize the 100 Markov Chains v^- to all zero states.

Then repeat:

1. Get the next batch of training data, v^+.
2. Calculate $h^+ = P(h|v^+, \theta\text{-regular})$, i.e. inference using the regular parameters. Calculate the positive gradient $g^+ = {v^+}^T h^+$.
3. Calculate $h^- = P(h|v^-, \theta\text{-regular} + \theta\text{-fast})$, i.e. inference using the regular parameters plus the fast weights overlay on the energy surface. Calculate the negative gradient $g^- = {v^-}^T h^-$.
4. Update v^- = sample from $P(v|h^-, \theta\text{-regular} + \theta\text{-fast})$, i.e. one full Gibbs update on the negative data, using the fast weights.
5. Calculate the full gradient $g = g^+ - g^-$.
6. Update θ-regular = θ-regular + $g \cdot$ regular learning rate for this iteration.
7. Update θ-fast = θ-fast $\cdot \frac{19}{20} + g \cdot$ fast learning rate for this iteration

7. Experiments

First, we ran some experiments on small tasks, to get some idea of the performance, and to find out what values work well for the algorithm parameters (weight decay, learning rate, etcetera). After that, we ran a larger experiment, with algorithm parameters that appeared to work reasonably well.

7.1. Initial Experiments on Small Tasks

7.1.1. A General Performance Comparison

We used two tasks, one requiring a bit more training time than the other. For both tasks, we used the MNIST (LeCun & Cortes,) data set. The smaller task was to train an RBM density model of the MNIST images, with a small RBM (25 hidden units). The larger task was classifying the MNIST images, using an RBM with 500 hidden units that learns a joint density model of images and labels, also known as a *classification RBM* (Hinton et al., 2006; Larochelle & Bengio, 2008). These same tasks were used in (Tieleman, 2008), which compared the performance of PCD to that of other, more commonly used methods. We include the data from that paper in our plots, so we can show plots of the performance of older methods, of PCD, and of PCD with fast and regular parameters (Fast PCD, or FPCD) - see Figure 1.

To have a fairly principled comparison, we ran the experiments as follows. We ran each algorithm many times, allowing it different amounts of time for learning. On each run, the amount of training time was decided in advance, and the learning rate was decayed linearly to zero over the course of the run[5]. Thus, these plots do not display the performance of one run of the algorithm at different stages in the learning process, but rather the performance obtained given various amounts of total training time. For each algorithm and for each of the different amounts of total training time, we ran 30 experiments with different settings of the algorithm parameters (such as *initial* learning rate and weight decay), evaluating performance on a held-out validation data set. Using the results of those validation runs, we picked the settings that worked best, and ran the experiment 10 times with those, evaluating on a held-out test data set. From the 10 performance numbers that this gave, we calculate mean and standard error of the mean to get error bars. After doing that for each of the algorithms, and for each of the amounts of total training time, the resulting error bars can be shown in one plot, and that is what Figure 1 shows.

As can be seen in these plots, we ran the experiments only for a short time. Nonetheless, there are some important observations to be made. Except in the

[5]We used linear decay of the learning rate in order to be directly comparable with the results in (Tieleman, 2008), but we believe that for FPCD it would be better to decay the learning rate more slowly at the end, since the rapid mixing caused by the fast weights will allow the regular weights to learn efficiently when the regular weights do not themselves cause rapid mixing.

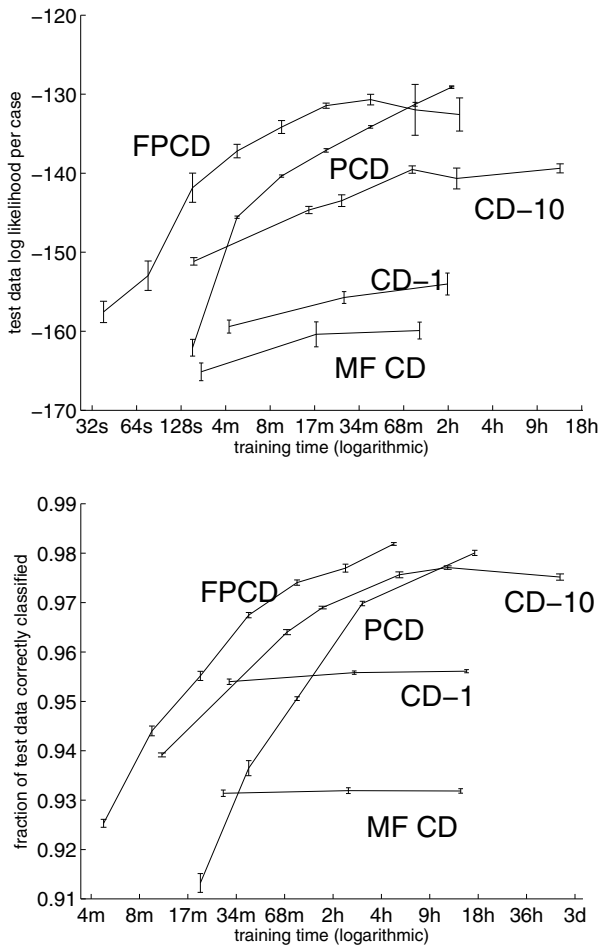

Figure 1. Modeling MNIST data using an RBM with 25 hidden units (top), and classifying MNIST data using a classification RBM with 500 hidden units (bottom). CD-1 and CD-10 refer to the Contrastive Divergence algorithm (Hinton, 2002); MF CD stands for Mean Field Contrastive Divergence (Welling & Hinton, 2002). For more details of these experiments see (Tieleman, 2008). One can see that using fast weights in situations with much training time and a small model causes a slight decrease in performance. Note that the error bars indicate the standard error of the mean - i.e. they are not the estimated standard deviation of the experiment outcomes, but, instead, that divided by $\sqrt{N}$.

part with the large error bars, FPCD outperforms the other algorithms. Comparing the amount of training time required for FPCD to get the same performance as PCD, we see that, on average, FPCD is about four times as fast as PCD. This difference is largest when there is little training time. We hope, therefore, that for larger tasks (e.g. learning larger data sets, with more hidden units), the difference will be even more significant.

7.1.2. Investigating Various Parameter Values

Fast PCD has various algorithm parameters. Not only are there the usual parameters of learning rate and weight decay for the regular model parameters, but there is also the question of how rapidly the fast model parameters should decay, whether to use momentum on the fast parameters, and what learning rate to use for the fast parameters.

In the experiments described in Figure 1, these algorithm parameters were set in a simple, but probably not optimal way. While the learning rate on the regular parameters was set with a decaying schedule, the learning rate on the fast parameters was kept constant at the *initial* learning rate for the regular parameters. Momentum was not used at all for the fast parameters; and the decaying of the fast parameters was done by multiplying them by $\frac{19}{20}$ after each parameter update.

To investigate what settings for those algorithm parameters may be better, we ran various experiments with a training time of 150 seconds - with that amount of training time, there is a significant but not extreme difference between the performance of PCD and FPCD. We hope that this is somewhat representative of the behavior on bigger tasks, where the ratio of training time to problem size is similar, while the short training time on these toy problems allows us to run many experiments.

For the classification task (500 hidden units), we found the following:

- Performance with 150 seconds of training time with the aforementioned heuristically chosen settings, and learning rate for the regular model parameters chosen using the validation set, is 10.36% misclassification. We ran that experiment 1,000 times so we have a small standard error of 0.03 percentage points on that 10.36%.
- Trying out various weight decay settings, we found that $6 \cdot 10^{-6}$ (as opposed to zero) seemed to work best, but performance was 10.33% misclassification with standard error of 0.06%, i.e. a minor difference, less than the standard error.
- The learning rate that we used on the fast weights ("fast learning rate") turned out to be a bit larger than optimal. We tried several alternative "fast learning rate" schedules, and the one that worked best was a linear *increasing* schedule, starting at about a third of the *initial* "regular" learning rate, and ending at the initial regular learning rate. This way, interestingly, the sum of the

two learning rates is approximately constant, so that the speed with which the energy surface rises under the fantasy particles is approximately constant. This keeps the fantasy particles moving rapidly, when the regular learning rate becomes small to allow fine-tuning of the regular parameters. With that schedule, the classification error rate was about 8.8% (the difference well exceeds the standard error).

- Our choice of not using momentum on the fast parameters turned out to be close to optimal - although using a tiny bit of momentum resulted in 1% less misclassification (again, this difference exceeded the standard error).

- Lastly, the decay applied to the fast parameters was also close to optimal: multiplying them by $\frac{49}{50}$ after each parameter update, as opposed to our heuristic $\frac{19}{20}$, gave a an improvement of 0.2 percentage points in the misclassification rate - a minor difference, even though it is beyond the standard error (we ran the experiments many times, so the standard errors are very small).

Of course, if we were to search for optimal settings for all of these five parameters combined, performance would be better still, but our main aim was to have parameter settings that work reasonably well and require no such search.

For the density modeling task with 25 hidden units, similar results were found: choosing some of the parameters more carefully might help a bit, but the simple values we chose initially work fine: no weight decay; "fast learning rate" constant at the *initial* regular learning rate; no momentum on the fast parameters; and decaying the fast parameters using a multiplication by $\frac{19}{20}$ after each weight update. From this we conclude that for large problems, where it is very expensive to carefully tune such algorithm parameters, these choices are sensible (though perhaps it is better to keep the regular learning rate plus the fast learning rate constant, as opposed to keeping the fast learning rate constant). If one uses them, there are no more parameters to be chosen when using FPCD than when using PCD. When comparing to CD-1 or MF CD, FPCD is even more favorable, because, with a reasonable amount of training data, weight decay can safely be set to zero for FPCD, while CD-1 definitely needs weight decay to keep the weights small and thus allow significant mixing in one Gibbs update. Thus, when using FPCD, the only parameters to be chosen are those of the model architecture (number of hidden units), and the basic optimization parameters (just the learning rate for basic gradient descent). This makes FPCD easy to use, in addition to being the best existing way to learn RBM's.

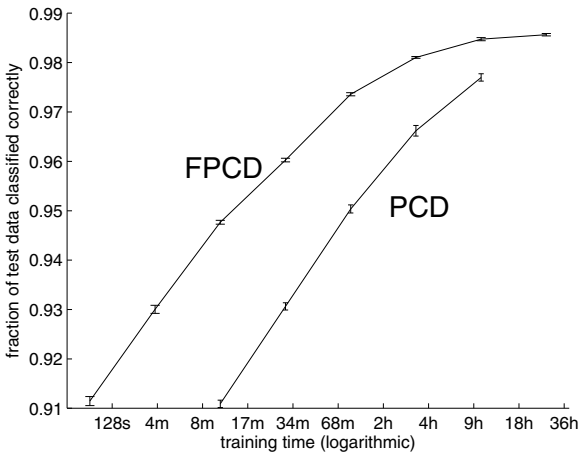

Figure 2. Classifying MNIST data (the number of hidden units is chosen using validation data).

7.2. Larger Experiments on MNIST

After verifying what algorithm parameter settings work well, we tested the algorithm on a larger task. The task was again classification of the MNIST images, but for these experiments, the number of hidden units was not fixed, but was chosen using the validation set. The initial learning rate was also chosen using the validation set. The additional FPCD parameters were set using the aforementioned heuristically chosen values. The result is displayed in Figure 2.

As was observed on the smaller tasks, FPCD takes, on average, about one quarter of the time required by PCD to get the same performance.

It is interesting to see what number of hidden units worked best for PCD versus FPCD. Given the same amount of training time, more hidden units means fewer parameter updates - the product of these two numbers has to be approximately constant. It turned out that for FPCD, the optimal number of hidden units is significantly larger than for PCD. For example, given 9 hours of training time, FPCD worked best with about 1200 hidden units, while PCD worked best with about 700 hidden units. This ratio of close to 2 was fairly consistent for different amounts of training time. The most natural explanation is that FPCD has the greatest advantage when fewer parameter updates can be done. That is why FPCD works best with a more powerful model, even though that means fewer param-

eter updates. PCD needs many parameter updates, and thus cannot afford many hidden units. Again, this confirms the hypothesis that FPCD is particularly advantageous for large tasks, where the amount of training time is a bottleneck.

7.3. Experiments on Another Data Set: 'Micro-NORB'

We also ran experiments on a modified version of the NORB data set (LeCun et al., 2004). NORB is an object classification data set, where the task is to tell apart previously unseen types of aircraft, trucks, cars, people, and animals, all under various lighting conditions and viewing angles. One drawback of NORB is its large input dimensionality: the data consists of pairs of 96x96 images, i.e. it is 18432-dimensional. Compared to MNIST's 784, this means that experimentation on NORB requires much more computer time.

We made a modified version of NORB, Micro-NORB (MNORB). The modification involved three steps: first, we apply a Mexican Hat filter to the images; then, we threshold the outcomes of the Mexican Hat filter; and finally, we downsample the images from 96x96 to 32x32, by averaging each 3x3 pixels into one pixel (see Figure 3). We use only the first of the two NORB images to further reduce the size of the input. This leaves us with a data set of 32x32 images - one eighteenth of the original size. This allows for less costly experimentation. The parameters of the Mexican Hat, as well as the threshold, were chosen to optimize the classification performance of the logistic regression algorithm, trained on 80% of the training set, and evaluated on the remaining 20%. Logistic regression performance on the test set (when trained on the full training set) was 26% misclassification, compared to approximately 23% on the full 18432-dimensional NORB data. In other words, the reduction in input size has made classification harder, but not impossible.

For these experiments, we tried some different learning rate schedules. As anticipated, the linearly-to-zero schedule turned out to be suboptimal. Another learning rate schedule, which allows more fine-tuning with small learning rates, worked better for all investigated algorithms, though the improvement was greatest for FPCD. The results, shown in Figure 4, are somewhat less clear due to measurement noise. However, roughly the same pattern is shown, with FPCD outperforming PCD. It appears that overfitting is a more serious concern for this data set: in the plot, one can see that more training time does not necessarily make for better performance. That is different for the MNIST data set. Nonetheless, for the MNORB data set, too, FPCD outperforms PCD and PCD PS.

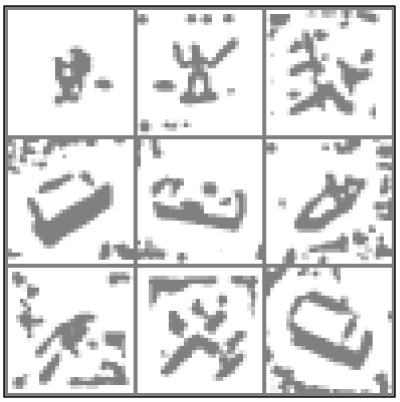

Figure 3. Some example pictures from MNORB. Some of these are easily recognizable, while others have become hard or impossible to recognize after the downsampling.

After some additional experiments on the MNIST data set, we chose a constant learning rate for the fast weights, of simply e^{-1}. That worked well, both for short and long runs. We used that same constant fast learning rate for the MNORB experiments, and on that data set, too, it seems to have worked well. We recommend setting the fast learning rate to that constant, as a first thing to try, also on other data sets.

8. Discussion and Future Work

In our MNIST experiments, we find that FPCD indeed outperforms PCD without fast weights. The difference is largest when the number of weight updates is small. With more time available for the optimization, the value of fast weights seems to diminish. However, it must be said that our choices for the algorithm parameters were chosen to match those in (Tieleman, 2008), which were optimized for PCD. A regular learning rate that decays linearly to zero may be a fine choice for PCD, but for FPCD it is likely suboptimal. The more established $\frac{1}{t}$ learning rate schedule (Robbins & Monro, 1951) fails to work well for PCD, because the mixing of the negative data Markov Chains requires a significant learning rate. Thus, very little learning would be possible for PCD after the learning rate gets small. For FPCD, however, mixing is ensured by the fast weights, so the regular weights can have a small learning rate without preventing rapid mixing. Thus, as long as the sum of the fast and the regular learning rates remains large enough, FPCD can hope

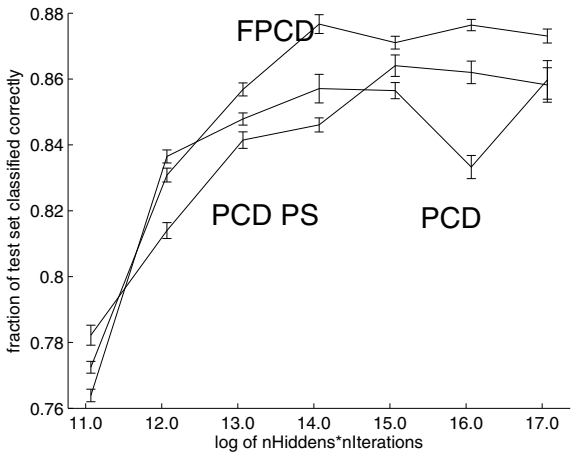

Figure 4. Classifying the MNORB data. On the x-axis is the logarithm of the product of two quantities: the number of hidden units and the number of updates. This product is approximately linearly related to the amount of time that the optimization requires. The *ratio* of these two quantities was chosen using validation data, for each plotted point. Notice that performance *decreases* at times, even though the learning rate was chosen using validation data, and thus early stopping was possible. The most plausible explanation for this is stochastic measurement noise.

to do fine-tuning with a small regular learning rate, which PCD cannot. With learning rate schedules that are more appropriate for FPCD, it is quite possible that even with much training time it will outperform PCD. Investigation into other learning rate schedules for FPCD is the main component of future work.

Our MNORB experiments suggest that FPCD can also perform well in situations where overfitting is a concern. They also suggest that the algorithm can work well with a simple, constant fast learning rate.

Other future work is to run this algorithm for a long time on an established data set, to test whether state of the art performance can be achieved or possibly exceeded, by RBMs using FPCD training.

Acknowledgements

We thank Ruslan Salakhutdinov for many useful discussions and suggestions. This research was financially supported by NSERC and Microsoft.

References

Bharath, B., & Borkar, V. (1999). Stochastic approximation algorithms: Overview and recent trends. *Sadhana*, *24*, 425–452.

Cvijovi, D., & Klinowski, J. (1995). Taboo Search: An Approach to the Multiple Minima Problem. *Science*, *267*, 664–666.

Hinton, G. (2002). Training Products of Experts by Minimizing Contrastive Divergence. *Neural Computation*, *14*, 1771–1800.

Hinton, G., Osindero, S., & Teh, Y. (2006). A fast learning algorithm for deep belief nets. *Neural Computation*, *18*, 1527–1554.

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

Larochelle, H., & Bengio, Y. (2008). Classification using discriminative restricted boltzmann machines. *Proceedings of the 25th international conference on Machine learning* (pp. 536–543).

LeCun, Y., & Cortes, C. The MNIST database of handwritten digits. `http://yann.lecun.com/exdb/mnist/`.

LeCun, Y., Huang, F., & Bottou, L. (2004). Learning methods for generic object recognition with invariance to pose and lighting. *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition* (pp. 97–104).

Neal, R. (1992). Connectionist learning of belief networks. *Artificial Intelligence*, *56*, 71–113.

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

Robbins, H., & Monro, S. (1951). A Stochastic Approximation Method. *The Annals of Mathematical Statistics*, *22*, 400–407.

Sminchisescu, C., & Welling, M. (2007). *Generalized darting monte carlo* (Technical Report CSRG-478). University of Toronto, Department of Computer Science.

Smolensky, P. (1986). *Information processing in dynamical systems: foundations of harmony theory.* MIT Press Cambridge, MA, USA.

Tieleman, T. (2008). Training restricted Boltzmann machines using approximations to the likelihood gradient. *Proceedings of the 25th international conference on Machine learning* (pp. 1064–1071).

Welling, M., & Hinton, G. (2002). A new learning algorithm for mean field Boltzmann machines. *Proceedings of the International Conference on Artificial Neural Networks* (pp. 351–357). Springer.