A Tutorial on Stochastic Approximation Algorithms for Training Restricted Boltzmann Machines and Deep Belief Nets

Kevin Swersky, Bo Chen, Ben Marlin and Nando de Freitas
Department of Computer science
University of British Columbia
BC, Canada
Email: {kswersky,bochen,bmarlin,nando}@cs.ubc.ca

***Abstract*—In this study, we provide a direct comparison of the Stochastic Maximum Likelihood algorithm and Contrastive Divergence for training Restricted Boltzmann Machines using the MNIST data set. We demonstrate that Stochastic Maximum Likelihood is superior when using the Restricted Boltzmann Machine as a classifier, and that the algorithm can be greatly improved using the technique of iterate averaging from the field of stochastic approximation. We further show that training with optimal parameters for classification does not necessarily lead to optimal results when Restricted Boltzmann Machines are stacked to form a Deep Belief Network. In our experiments we observe that fine tuning a Deep Belief Network significantly changes the distribution of the latent data, even though the parameter changes are negligible.**

I. Introduction

Deep belief networks have recently been successfully applied to many problems in the areas of classification, dimensionality reduction [1], collaborative filtering [2], semantic hashing [3], and more. In the past they were generally trained from random weight initializations using backpropagation, however this suffered from the tendency to get stuck in poor local minima, making such deep networks inferior compared to more shallow architectures with only one or two hidden layers [4]. Deep architectures have the potential to be extremely useful, as it has been shown that in some domains such as representing logic circuits, deep architectures can represent functions much more efficiently (i.e. with exponentially fewer parameters) than shallow ones [5]. Invariance is also an important motivating factor for considering hierarchical structures [6], [7], [8], [9].

In [10] it was shown that it is possible to train a deep network by learning each layer greedily in an unsupervised manner as a Restricted Boltzmann Machine (RBM). The mean-field hidden unit activations in one RBM would be used as the features for the next RBM, and so forth. In order to train the RBMs with Maximum Likelihood, they run a blocked Gibbs sampler for many iterations in order to draw unbiased samples from the model distribution to generate a good gradient approximation. This is slow and impractical, however they found that they could train an RBM efficiently by always starting from the data, and truncating the Gibbs chain at a single iteration. They called this procedure Contrastive Divergence (CD). In [11], Laurent Younes proposed a method for training general Boltzmann Machines (of which an RBM is a special case) by using a persistent chain of samples to represent the model distribution, from which only one iteration of Gibbs sampling needs to be run per iteration. We will refer to this algorithm as Stochastic Maximum Likelihood (SML). In [12] it was shown that for classification with an RBM, SML outperformed CD. However, it required a much smaller learning rate to work effectively, making CD much faster for the initial stages of learning. As we will show, this is not necessarily true.

This paper addresses several issues regarding the training of RBMs and Deep Belief Networks. It discusses, briefly, the asymptotic theory of stochastic approximation that governs many of the existing algorithms. This theory provides some guidelines for understanding the behaviour of the algorithms. Yet, the field is so immature, that it is useful to analyze carefully all the empirical choices (parameters, training procedures) that researchers are currently adopting to train these feature learning architectures. Some of the tricks, often not mentioned in papers, end up playing a crucial role. For these reasons, this paper embarks on an empirical study of these choices in the hope of providing useful practical guidelines for people entering this area of research.

We run a series of experiments which explore various implementation issues and strategies for training RBMs. We provide a direct comparison between SML and CD for training both RBMs and Deep Belief Networks, and we show that using the appropriate techniques, it is possible to make SML efficient even with a high learning rate. Next, we explore the issues related to training Deep Belief Networks, including the effects of fine tuning on the weights in the network as well as the performance of the features learned from the RBMs before and after backpropagation.

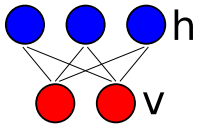


Fig. 1. An RBM with visible units v and hidden hints h. The parameters W correspond to the strengths of the edge connections.

II. RBMs

An RBM, shown in Figure 1, is a bipartite probabilistic graphical model corresponding to the following distribution:

$$p(v,h|W) = \frac{1}{Z(W)} \exp\left(-E(v,h,W)\right),$$

where $v \in V$ denotes the observation nodes, $h \in H$ the latent random variables and $W \in \mathbb{R}^{D\times K}$ the parameters governing the interactions between the D visible units and the K hidden units. The term $E(v,h,W)$ is called the energy function and $Z(W)$ is a normalizing term given by:

$$Z(W) = \sum_{v'\in V}\sum_{h'\in H} p(v',h'|W).$$

When following the maximum likelihood principle, the goal of learning in an RBM is to maximize the marginal probability of the data $p(v|W)$ given by:

$$p(v|W) = \frac{1}{Z(W)} \sum_{h\in H} \exp\left(-E(v,h,W)\right).$$

In the binary case where $V = \{0,1\}^D$ and $H = \{0,1\}^K$ the energy function can be expressed as:

$$E(v,h,W) = -\sum_{i=1}^{D}\sum_{j=1}^{K} v_i W_{ij} h_j - \sum_{i=1}^{D} v_i c_i - \sum_{j=1}^{K} h_j b_j.$$

Here we have added a bias terms for the visible units and the hidden units, which are parameterized by c and b respectively. We can absorb these terms into W by simply adding a visible unit and a hidden unit which are always set to 1.

In general, RBMs can be used to model many types of data, and a procedure for derving RBMs for general exponential family models can be found in [13]. For simplicity, we restrict our description to the binary case.

RBMs have the property that given the hidden units, all of the visible units become independent and given the visible units, all of the hidden units become independent. Thus we can sample from the conditionals $p(v_i|h)$ and $p(h_j|v)$, $i \in \{1,...,D\}$, $j \in \{1,...,K\}$ exactly since they are logistic functions taking the form:

$$p(v_i = 1|h,W) = \sigma\left(\sum_{j=1}^{K} W_{ij} h_j\right)$$

$$p(h_j = 1|v,W) = \sigma\left(\sum_{i=1}^{D} W_{ij} v_i\right),$$

where $\sigma(a) = \frac{1}{1+exp(-a)}$.

A. Training RBMs

For a given *i.i.d.* data set $\mathbf{e} = \{d_1, d_2, ...d_N\}$ the goal of training an RBM is to find the set of weights that maximize the average log-likelihood of the data:

$$W^* = \arg\max_W \frac{1}{N}\sum_{n=1}^{N} log\left(\sum_{h\in H} exp(-E((v=d_n,h,W))\right) - log\left(Z(W)\right).$$

The partial derivative with respect to W_{ij} of the objective above is given by:

$$\frac{1}{N}\sum_{n=1}^{N}\sum_{h\in H} d_{in} h_j p(h|W, v=d) - \sum_{v'\in V}\sum_{h'\in H} v'_i h'_j p(v',h'|W),$$

where d_{in} refers to the i^{th} unit of the n^{th} data instance. The sum on the left hand side can be computed exactly, however the expectation on the right hand side (also called the expectation under the model distribution) is intractable. In order to calculate it, one can do alternating blocked-Gibbs sampling from the conditionals $p(h|v)$ and $p(v|h)$. This leads to a Maximum Likelihood algorithm which requires an infinite number of Gibbs transitions per update to fully characterize the expectation. In practice one would not be able to run the Gibbs chain forever, but it would be run for a large number of transitions to generate approximate samples from the model distribution. Unfortunately, for RBMs with many visible and hidden units this algorithm is prohibitively slow. Geoff Hinton [14] proposed a modification of the maximum likelihood updates known as contrastive divergence. The key difference is that the Gibbs chain is started at the data, and run for only a few steps. The algorithm for taking one step, CD-1, proceeds as follows:

1) *Sample* $\widehat{h}^{(t)}$ *from* $\prod_j p(h_j|v=d)$, *then sample* $\widetilde{v}^{(t)}$ *from the conditional* $\prod_i p(v_i|\widehat{h}^{(t)})$ *and finally* $\widetilde{h}^{(t)}$ *from* $\prod_j p(h_j|\widetilde{v}^{(t)})$.
2) *Perform a CD-1 update:*

$$W_{ij}^{(t)} = W_{ij}^{(t-1)} + \eta_t\left[d_i\widehat{h}_j^{(t)} - \widetilde{v}_i^{(t)}\widetilde{h}_j^{(t)}\right]$$

3) *Increase* t *to* $t+1$ *and iterate again.*

One important consideration with these algorithms is that during the parameter updates, we can replace the sampled $\widetilde{h}$ with their expected values $E[h|v]$. This process is called Rao-Blackwellisation and the resulting estimator has lower variance than the versions given above:

Proposition 1. *The variance of the Rao-Blackwellised estimate* $d\mathbb{E}(h|d)$ *is lower than the variance of the Monte Carlo estimate with* dh.

Proof:

$$\begin{aligned}\mathbb{V}(dh) &= \mathbb{E}(d^2h^2) - \mathbb{E}(dh)^2 \\ &= \mathbb{E}[\mathbb{E}(d^2h^2|d)] - \mathbb{E}[\mathbb{E}(dh|d)]^2 \\ &= \mathbb{E}[\mathbb{V}(dh|d) + \mathbb{E}(dh|d)^2] - \mathbb{E}[\mathbb{E}(dh|d)]^2 \\ &= \mathbb{E}[\mathbb{V}(dh|d)] + \mathbb{V}[\mathbb{E}(dh|d)] \\ &\geq \mathbb{V}[\mathbb{E}(dh|d)] \\ &= \mathbb{V}[d\mathbb{E}(h|d)]\square\end{aligned}$$

The Rao-Blackwellised CD update is given by:

$$W_{ij}^{(t)} = W_{ij}^{(t-1)} + \eta_t \left[d_i \sigma\left(\sum_{i=1}^{D} W_{ij} d_i \right) - \widetilde{v}_i^{(t)} \sigma\left(\sum_{i=1}^{D} W_{ij} \widetilde{v}_i^{(t)} \right) \right]$$

It has been shown in [15] that in general, the fixed points of CD will differ from those of Maximum Likelihood, but assuming the data is generated from an RBM it can be shown that asymptotically they both share the Maximum Likelihood solution as a fixed point [16]. In [17] conditions were given to guarantee convergence of CD, however they are difficult to satisfy in practice.

Prior to the advent of CD-1, Laurent Younes devised a Stochastic Approximation algorithm for Maximum Likelihood training of general Boltzmann Machines, however we focus on the RBM variant. Instead of resetting the chain to the data after each parameter update, the previous state of the chain was kept and used for the next iteration of the algorithm. It can be shown that this algorithm generates a consistent estimator, even with one Gibbs transition per iteration. The algorithm is given as follows (for one step of Gibbs sampling):

1) *If* $t = 1$ *then generate a random sample* $\widetilde{h}^{(t-1)}$ *from* $\prod_j p(h_j|v = d)$.
2) *Sample* $\widetilde{v}^{(t)}$ *from* $\prod_i p(v_i|\widetilde{h}^{(t-1)})$ *and* $\widetilde{h}^{(t)}$ *from* $\prod_j p(h_j|\widetilde{v}^{(t)})$.
3) *Perform a Robins-Monro parameter update (with Rao-Blackwellisation):*

$$W_{ij}^{(t)} = W_{ij}^{(t-1)} + \eta_t \left[d_i \sigma\left(\sum_{i=1}^{D} W_{ij} d_i \right) - \widetilde{v}_i^{(t)} \sigma\left(\sum_{i=1}^{D} W_{ij} \widetilde{v}_i^{(t)} \right) \right]$$

4) *Increase* t *to* $t+1$ *and iterate again.*

The theory of stochastic approximation (SA) provides us with tools for analysing the asymptotic behaviour of these algorithms. The standard proofs apply directly to the stochastic maximum likelihood (SML) method proposed by Younes. It is trickier to analyse the convergence of CD, but the theory still says a lot about the tricks (*e.g.* momentum, constant learning rates) used to make CD work in practice. For this reason, and to guide our experiments later on, we review briefly some ideas of stochastic approximation that are of relevance to our problem. Of course, the theory is mostly asymptotic. We will therefore conduct a comprehensive empirical evaluation of the many routinely used training tricks.

B. Stochastic Approximation

Stochastic approximation algorithms are designed to minimize the expected loss $l(W)$, with $W \in \mathbb{R}^{D\times K}$, over the distribution of the data $P(v)$:

$$l(W) = \int L(W, v) p(v) dv.$$

(For simplicity of presentation only, we have marginalized out the hidden units.) The goal is to find a stationary point $W^\star$ of the gradient of this loss function, which under mild continuity conditions is given by:

$$g(W) = \int \nabla L(W, v) p(v) dv = 0.$$

Multiplying both sides of this equation by $-\frac{1}{t}\Gamma$, where Γ is a $(D \times K) \times (D \times K)$ matrix, adding W to both sides of the equation, and finally discretizing by simulating $v^{(t)} \sim P(v)$ so that $\nabla L(W^{(t)}, v^{(t)})$ is an point-estimate of the integral, we obtain the standard SA algorithm:

$$W^{(t+1)} = W^{(t)} - \frac{1}{t}\Gamma \nabla L(W^{(t)}, v^{(t)}).$$

Adding and subtracting $g(W^{(t)})$ gives the celebrated Robins-Monro update:

$$W^{(t+1)} = W^{(t)} - \frac{1}{t}\Gamma g(W^{(t)}) + \frac{1}{t}\Gamma \left[g(W^{(t)}) - \nabla L(W^{(t)}, v^{(t)}) \right]$$

where $M^{(t)} = g(W^{(t)}) - \nabla L(W^{(t)}, v^{(t)})$ is typically a bounded martingale difference. It can be interpreted as the "noise" in the estimate of the gradient as a result of partial observability of the entire dataset.

Several authors, see for example [18], have shown that the sequence $\{W^{(t)}\}$ converges to $W^\star$ and proved central limits showing that $\sqrt{t}(W^{(t)} - W^\star)$ converges in law to a zero mean Gaussian distribution with variance Σ. They have also shown that the lowest asymptotic variance $\Sigma^\star$ is obtained for Newton's method, which corresponds to the choice $\Gamma^{-1} = \nabla g(W^\star)$. This observation has motivated the proposal of stochastic Newton algorithms [19], [20]. Unfortunately, among other drawbacks, the Hessian matrix $\nabla g(W^\star)$ is typically unknown a priori and can be difficult and expensive to estimate. There is a remarkably simple way around this difficulty: *averaging*.

The averaging method, proposed in [21], uses the following two-time-scale approximation:

$$\begin{aligned} W^{(t+1)} &= W^{(t)} - \gamma^{(t)} \nabla L(W^{(t)}, v^{(t)}) \\ \overline{W}^{(t+1)} &= \overline{W}^{(t)} - \frac{1}{t}(\overline{W}^{(t)} - W^{(t)}), \end{aligned} \quad (1)$$

where $\gamma^{(t)}$ is a scalar learning rate and, clearly, $\overline{W}^{(t)} = \frac{1}{t}\sum_{m=1}^{t} W^{(m)}$. Several authors, see for instance [22], have used martingale tools to show that this averaging procedure attains the optimal rate when $\gamma^{(t)} = 1/t^\alpha$, with $\alpha \in (1/2, 1)$. That is, the distribution of the normalized error $\sqrt{t}(W^{(t)} - W^\star)$ is asymptotically normal with zero mean and covariance $\Sigma^\star$, as defined previously for Newton's method — *both methods achieve the optimal rate.*

The slower decaying learning rate $1/t^{\alpha}$ allows the algorithms to take bigger steps (an important practical requirement), while the averaging "squezes" the estimates in the vicinity of the optimum $W^{\star}$. In fact, one can even choose a constant learning rate $\gamma^{(t)} = \gamma$ and still be able prove this theoretical result [23], [24], [25]. This is typically done with the ordinary differential equation (ODE) method of proof [26]. For a recent treatment of the ODE method for multiple timescales and fixed (or bounded) learning rates see [27].

Recall the typical SA algorithm with learning rate γ:

$$W^{(t+1)} = W^{(t)} - \gamma g(W^{(t)}) + \gamma \left[g(W^{(t)}) - \nabla L(W^{(t)}, v^{(t)}) \right]$$

Since the "noise" term, $g(W^{(t)}) - \nabla L(W^{(t)}, v^{(t)})$, vanishes asymptotically, one can interpret a SA algorithm as an Euler discretization of the following multivariate continuous-time ODE:

$$\dot{W}(\tau) = -g(W(\tau)).$$

The SA fixed points in the limit correspond to the attractors of the ODE. For a constant learning rate, the weights W will follow the ODE trajectories, instead of converging to a unique value. This has been exploited in tracking applications in the past [26].

While this simple ODE maps to the stochastic gradient update, higher order ODEs can be shown to map to more sophisticated updates, such as gradient descent with momentum [28]. These SA algorithms with momentum have also been analyzed to some extent in [29].

There is a map between dynamical systems governed by ODEs and learning in Boltzmann machines. While some people think about dynamics others think about algorithms. The two are the same. This connection has been explored to a limited extend in the learning field [30], [31], but we believe much more needs to be done.

Iterate averaging has also been considered by theoreticians working with convex problems and aiming to get finite time bounds, see for example [32] and the many references therein.

In the case of the SML algorithm with decaying learning rates, it is straightforward to prove convergence in the sense that it is possible to specify a potential (Lyapunov) function so that the maximum of this function coincides with one of the stationary points of the likelihood function. The Lyapunov function needed is, obviously, the log-likelihood function itself in this case. The algorithm iterates are then shown to be a super-martingale, which is guaranteed to ascend on average toward a maximum of the likelihood-function. In the proof, one also needs to use the Poisson equation to deal with the fact that the states are not independent but are samples from a Markov chain; see for example [33]. The treatment of convergence of these algorithms has parallels with what is done in the study and design of Monte Carlo EM and adaptive MCMC algorithms [34]. To a lesser extent, the treatment is also related to the family of stochastic algorithms used in reinforcement learning [35].

In the case of CD, on the other hand, the Lyapunov function is not obvious. Moreover, the samples are not drawn from a Markov chain, but are biased to be near the data. The latter should make the analysis simpler, but the lack of an obvious Lyapunov function complicates matters.

III. Experiments with Shallow RBMs

In order to use RBMs effectively, it is essential that one chooses appropriate parameters. There are also many other factors involved in training RBMs that can have a significant impact on their performance. This section of the paper explores various strategies that can be applied during the training of RBMs which can be used to improve classification performance.

There are many ways to assess the performance of an RBM, these include log-likelihood, train misclassification error, test misclassification error, reconstruction error, and samples generated from the model. While ideally one would choose log-likelihood as well as test error for assessing the performance of training an RBM for classification, calculating the log-likelihood is intractable for all but trivially small RBMs since the number of possible states in the model grows exponentially with the number of units. We choose test error as our performance measure since in addition to showing approximately how well the model is being trained, it also gives an indicator of how well the model is able to generalize to new data.

In order to train an RBM for classification, we append the label unit to the visible vector v to form the vector $v =< c, v_u >$ where v_u correspond to the input feature vector (pixels in the case of a raw image) and $c \in C$ corresponds to the class label of v. This is trained as an ordinary RBM and the most likely class is given by:

$$class(v) = \arg\max_{c' \in C} \log(\sum_{h \in H} \exp(-E(v =< c', v_u >, h, W))).$$

It was shown in [2] that the time complexity to find the most likely class for a single data point is $O(K \times |C|)$ where $|C|$ is the number of possible classes. The error reported in our experiments corresponds to the proportion of properly classified points on the test set.

For each experiment, we use 500 hidden units. It is expected that more hidden units will improve classification performance, but then it becomes prohibitive to run many experiments; in general the model is computationally impractical when the number of hidden units is too large. We also use mini-batches of 100 points, and a justification for this is given in a later section. We use the MNIST [36] training set and evaluate error on the provided test set since this allows us to directly ensure that our results are comparable to current published works. MNIST has the advantage of being a well studied data set, and there are many papers providing benchmark results. However, in the future these experiments should be repeated on a variety of data sets from different tasks in order to gain a more general insight on the behavior of RBMs.

A. Learning Rate, Momentum, and Weight Decay

To begin, we tested CD-1 and SML over a range of parameter settings in order to determine reasonable values

and to get a sense of their robustness. We randomly selected 500 training instances, and 100 testing instances per class from the MNIST training set and ran each algorithm for 250 iterations using a grid search over the parameter settings. We ran experiments with several values for the learning rate parameter η: $\{0.1, 0.01, 0.001\}$. We also included momentum terms [37] to smooth the trajectory of the gradient descent rule as follows:

$$\begin{aligned} I_{ij}^{(0)} &= 0 \\ I_{ij}^{(t)} &= \beta * I_{ij}^{(t-1)} + \eta * \nabla_{W_{ij}} log(p(v|W^{(t-1)})) \\ W_{ij}^{(t)} &= W_{ij}^{(t-1)} + I_{ij}^{(t)}. \end{aligned}$$

Momentum is a way to average the gradient of the current iteration with the gradients of previous iterations, thus smoothing the trajectory. A similar method that does this on a separate time scale is Polyak Averaging, which is discussed later in Section 3.4. We test β with the values $\{0, 0.3, 0.5, 0.8\}$.

Finally, we experimented with weight decay, where an L2 penalty is added to the objective function to encourage smaller weights. The update with weight decay (not including momentum) is:

$$W_{ij}^{(t)} = W_{ij}^{(t-1)} + \eta^{(t)} \nabla_{W_{ij}} log(p(v|W^{(t-1)})) - \lambda W_{ij}^{(t-1)},$$

where λ regulates the strength of the weight decay term. We test with the following values for the weight decay: $\{0.001, 0.0001, 0.00001\}$.

We found that CD-1 works best with a learning rate of 0.1 and a weight decay of 0.001 and momentum of 0.8 while SML works best with a learning rate of 0.1, a weight decay of 0.001 and a momentum of 0.3.

B. Mini-Batches

Stochastic gradient descent (SGD) is a procedure for optimizing a function with gradient descent in the presence of noise where only approximations to the gradient can be obtained. In the case where the objective function relies on data, noise is injected into the process when selecting subsets of data for each evaluation of the gradient. SGD has been shown to be an effective optimization procedure for many machine learning problems, often converging to the optimal test error much faster than deterministic procedures [38]. Given a data set **e**, the ordinary gradient descent update for an RBM is given by:

$$W_{ij}^{(t)} = W_{ij}^{(t-1)} + \eta \frac{1}{N} \sum_{n=1}^{N} \nabla_{W_{ij}} log(p(d_n|W^{(t-1)}))$$

With mini-batches, we would split this data set into K disjoint (usually equally sized) sets, denoted K_l. Starting with $l = 1$ each update is given by:

$$W_{ij}^{(t)} = W_{ij}^{(t-1)} + \eta \frac{1}{|K_l|} \sum_{d' \in K_l} \nabla_{W_{ij}} log(p(d'|W^{(t-1)}))$$

Where after each update we choose the next mini batch by incrementing l, and $|K_l|$ denotes the size of the l^{th} mini-batch. Once we have gone through the each K_l we set $l = 1$ and cycle through the data again for the next set of updates.

We test the effectiveness of this procedure by splitting the data into several smaller data sets called mini-batches, and perform SGD by cycling through each batch; selecting one each time we perform a gradient evaluation. We experimeriment with CD-1 with batch sizes $\{100, 1000, 60000\}$. Note that 60000 corresponds to the full data set, where the only stochasticity comes from the sampling involved in CD-1. We did not obtain results for batch sizes smaller than 100 since there is a computational cost from memory accesses and for loops in Matlab which makes SGD on batches this small impractical. Note that each iteration in the plot corresponds to one pass through the full data set.

We found empirically that using mini-batches provides an improvement in the convergence of the optimization. What is more surprising is that smaller mini-batches seem to converge to a lower test error than the methods that use a higher batch size. For the remainder of this paper we continue using the mini-batch procedure with batches of 100 points.

C. Binary v.s. Soft Data

While the RBMs presented in this paper are designed for binary data, [39] use several tricks that violate this principle, yet still yield good results. The first trick is to normalize the data so that each pixel falls in $[0, 1]$, which we call "soft data". This can be interpreted as the probability that the pixel in a given image will be turned on. The second trick is used in their implementation of CD, where instead of sampling $\widetilde{v}^{(1)} \sim p(v|\widetilde{h}^{(0)})$, and $\widetilde{h}^{(1)} \sim p(h|\widetilde{v}^{(1)})$, they use $p(v|\widetilde{h}^{(0)})$ in the place of $\widetilde{v}^{(1)}$. This means that they sample the second set of hidden units by $\widetilde{h}^{(1)} \sim p(h|p(v|\widetilde{h}^{(0)}))$. We will call this using "soft samples". We experiment with all four combinations of using binary v.s. soft values with CD-1. In order to binarize the data, we threshold the pixel values at 0.25.

We found that sampling the second set of visible units properly is an important part of the algorithm, and that this trick should not be employed when seeking good classification performance. Normalizing the data appears to give better performance, perhaps because it retains more information about the original images which is lost in the binarization. Other distributions for visible units have been proposed including Gaussian and the Truncated Exponential [40] and these may be more appropriate for this kind of data.

D. Polyak Averaging

Generally when learning with Stochastic Approximation, a $\frac{1}{t}$ schedule is used to ensure convergence. This tends to make learning quite slow, and if the noise is sufficiently small then a constant learning rate can be used instead. This is what is usually done for training RBMs. As we will show, adopting the Polyak averaging method described in II-B can also help improve results. The updates for the parameters in

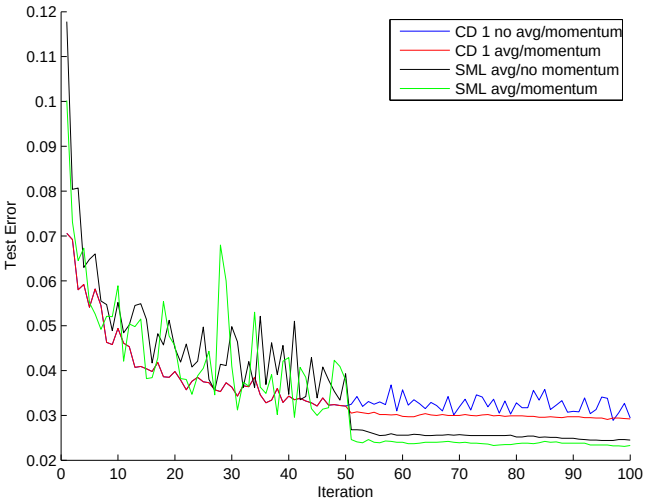


Fig. 2. Error versus iterations for CD with momentum and averaging, CD with momentum and no averaging, and averaged SML with and without momentum. (Best viewed in colour.)

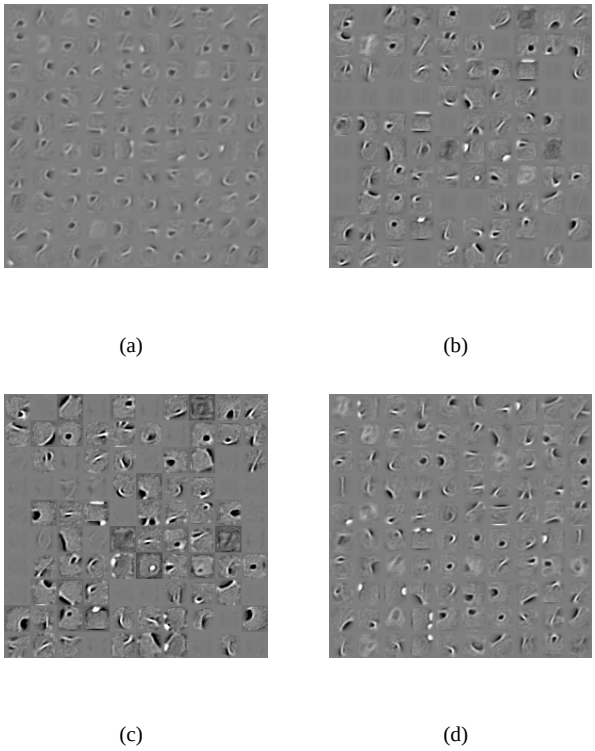


Fig. 3. Receptive fields for an RBM trained with Contrastive Divergence. (a) weight decay = 0.01, (b) weight decay = 0.001, (c) weight decay = 0.0001, (d) annealed weight decay from 0.01 to 0.001. Momentum is fixed at 0.8

this modification are given by:

$$W_{ij}^{(t)} = W_{ij}^{(t-1)} + \eta \nabla_{W_{ij}} log(p(v|W^{(t-1)}))$$
$$\overline{W_{ij}}^{(t)} = \overline{W_{ij}}^{(t-1)} - \frac{1}{t}(\overline{W_{ij}}^{(t-1)} - W_{ij}^{(t)})$$

When the algorithm terminates at time T we return $\overline{W_{ij}}^{(T)}$ instead of $W_{ij}^{(T)}$. It has been proven that this algorithm provides the optimal rate of asymptotic convergence. The intuition is that the iterates will converge quickly to a neighborhood around the solution and then oscillate. Taking the average of these oscillations will yield the true solution. Of course, intuition also tells us that if we start averaging before the oscillation begins, the averaged estimates at these times might actually be worse than the unaveraged ones. In practice, we let the algorithm run for a few iterations before we begin the averaging process. It is also possible to combine momentum with averaging, and below we show the results for using averaging starting at iteration 50 with and without momentum. To keep the graph uncluttered, we omit SML without averaging, since it did not show any real improvement after the 50^{th} iteration.

Figure III-D shows that averaging improves the test error scores of both algorithms, while providing a significant increase in stability. Indeed using averaging with a high learning rate, we were able to achieve comparable results to [22]. However, while they took approximately 18 hours to achieve their best results, Polyak Averaging allows us to achieve them after only a few minutes of training.

E. The Annealed Weight Decay/Momentum Strategy

We now take a qualitative look at the receptive fields produced by CD-1, as shown in Figure 3. Also referred to as filters, these are plots of the weights coming into each hidden unit from each visible unit. We can display these as images, and we choose a randomly selected subset of hidden units to display. The pixels corresponding to weights larger than 1 are shown in white, and weights smaller than -1 are shown in black. All intermediate values are displayed in varying shades of gray. For a momentum of 0.8 and weight decay of 0.001 some of the filters remain blank and uninteresting despite a good test error score. These blank filters correspond to dead units. They are never activated for any data points, and thus no learning occurs. It seems reasonable that each filter should contribute to the modelling of the data, otherwise they simply waste computational resources. A suprising result occurs when we change the value of the weight decay to 0.01: the previously blank filters become more interesting, though due to the higher penalty they are not quite as prominent. This suggests a new strategy where we anneal the weight decay from a high value to a low value over the course of training, in order to force the RBM to utilize as many hidden units as possible. We start with a weight decay of 0.1, and lower it to 0.01 after 5 iterations and finally 0.001 after 10 iterations.

We applied a similar procedure to mometum, testing the values $\{0, 0.5, 0.8\}$. We were similarly able to utilize more hidden units when annealing momentum from 0 to 0.8 over the course of training. The code from [39] uses a similar annealing trick, however to our knowledge this is never explicity mentioned in any papers.

We found empirically that the annealing strategy achieves a lower error, and that annealing weight decay or momentum

both yield very similar results. Annealing with SML did not improve test error. SML appears to utilize most of the hidden units.

F. Sparse RBMs

A variant of the standard RBM is the sparse RBM from Lee et al. [41]. This is obtained by adding a regularizer to the full data negative log likelihood so that it becomes:

$$-\log P(\mathbf{e}|W) = -\frac{1}{N}\sum_{n=1}^{N}\log\left(\sum_{h\in H}P(d_n,h|W)\right) + \lambda\sum_{j=1}^{K}\left(\frac{1}{N}\sum_{n=1}^{N}E[h_{jn}|d_n,W]-p\right)^2,$$

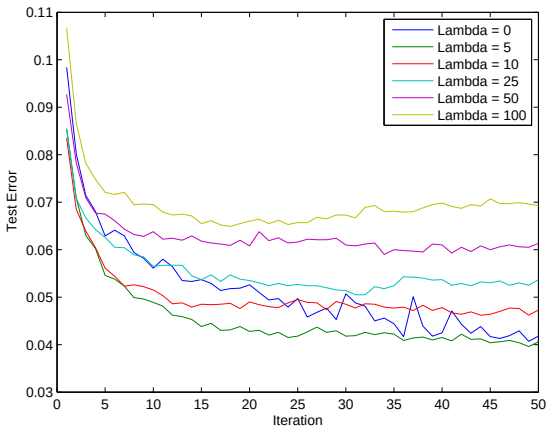
where p is a small constant corresponding to the number of times each hidden unit should be activated on average, and λ corresponds to the penalty strength. If we expand the regularization term we get:

$$\sum_{j=1}^{K}\left(\frac{1}{N}\sum_{n=1}^{N}E[h_{jn}|d_n,W]-p\right)^2 = \sum_{j=1}^{K}\frac{1}{N^2}\sum_{n=1}^{N}\sum_{n'=1}^{N}E[h_{jn}|d_n,W]E[h_{jn'}|d_{n'},W] - \frac{p}{N}\sum_{n=1}^{N}E[h_{jn}|d_n,W]-p^2.$$

If p is sufficiently small, then we this regularizer penalizes pairwise correlations between the hidden activations of different data points. There have been several attempts to model lateral inhibition in neural architctures, e.g. [42]. Generally these impose a penalty on pairwise activities between hidden units within the same data point. In our experiments this produced many dead units, and was notoriously difficult to learn. If the λ is set too high in Lee's version, the same phenomenon also occurs, however this regularizer appears to be much less sensitive to this issue. As an implementation detail, since we are training with mini-batches of data, we follow Lee's guidelines, and only update the hidden unit biases. Updating the rest of the weights according to the regularizer produces inferior results. It should be noted that in [43] an alternative approach was adopted where they simply subtracted a small constant from the hidden unit biases at each iteration. This appeared to produce qualitatively similar results.

To assess the effectiveness of sparsity, we evaluated test error as the sparsity parameter λ changed as shown in Figure 4. For p we used a value of 0.02. Interestingly, momentum appears to hurt the performance of sparse RBMs, so we set it to 0. The rest of the parameters remain the same, and following Lee's procedure we use the CD learning rule.

Some degree of sparsity does appear to help test error performance, although without momentum the results are not as good as the non-sparse case. When λ is too high this seems to hurt performance. Interestingly, the majority of decrease in average code size occurs when λ goes from 0 to 5. Finally,



(a)

Fig. 4. (a) Error for different values of the sparsity parameter λ for Sparse RBMs with averaging and 0 momentum.

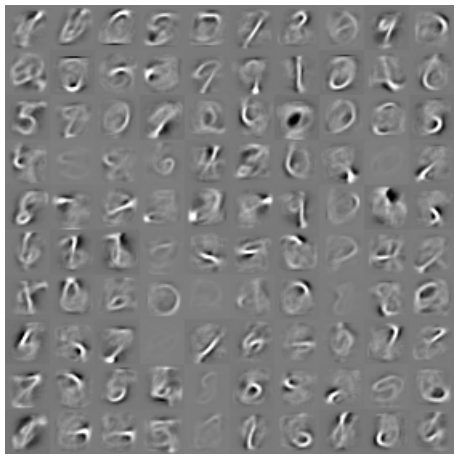


Fig. 5. Filters learned by a Sparse RBM. The majority appear to capture various strokes.

we show in Figure 5 that the Sparse RBM does indeed learn more stroke filters than a standard RBM.

We should mention that there have been successful applications of sparse RBMs in [6] but these used Gaussian visible units which may behave quite differently with this regularizer since they have negative as well as positive values.

G. Conclusions on Shallow Learning

In this limited set of experiments, an RBM with a properly tuned SML always beats an RBM with a properly tuned Contrastive Divergence. This is reasonable, since CD is most likely a biased estimator. These results are also consistent with those obtained in [12]. It seems fairly clear that for shallow training of generative binary RBMs for classification, SML is superior.

There are many tricks that can improve the performance of both algorithms, and these experiments helped to reveal insight into some of the pitfalls people might come across when trying to match published results. In the next section, we look at the issue of deep learning, and whether proper training in the shallow case leads to improved results in the deep case.

IV. Deep Belief Networks

Current state of the art training of DBNs, see Figure 6, involves training each layer greedily as an RBM, passing the mean-field approximation to the hidden unit activations as features for the next RBM, followed by a global fine tuning phase with backpropagation. Note that in the greedy phase no label information is given, and it is only provided during the fine tuning phase as the output of the network. The idea of a DBN is to automatically learn low level features in the lower layers, and higher level concepts in the higher layers in order to more accurately capture the statistical regularities present in the data. A different perspective given in [44] casts greedy training of a DBN as a regularizer for a Multilayer Perceptron.

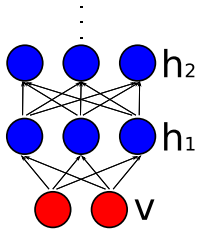


Fig. 6. A Deep Belief Network with visible units v and hidden hints h_1, h_2, etc. The edges are now directed.

A. Application of Shallow Parameters

Having discovered good parameters and strategies for training a shallow RBM for classification, we trained a DBN to see if these strategies would also perform well in a deep setting. For CD we annealed the momentum from 0 to 0.8 and for SML, we used a momentum of 0.3. For both algorithms we used a learning rate of 0.1, a weight decay of 0.001, and we used the standard 500-500-2000 architecture given in [1]. Each layer was trained for 50 iterations with Polyak Averaging beginning at the 40^{th} iteration. Fine tuning with backpropagation was applied for 30 iterations using Conjugate Gradient with 3 line searches on mini-batches of 1000 points. To determine the effect of sparsity in DBNs, we also pretrain the deep network using Sparse RBMs. The results can be seen in Table I

Algorithm	Test Error (%)
CD using best strategies from above	1.2
SML using best strategies from above	1.22
CD weight decay 0.0001	1.12
SML weight decay 0.0001	1.14
CD default settings	1.10
Sparse RBM with CD, λ = 5	1.87

TABLE I
Test Error Rates for CD and SML.

Algorithm	Test Error (%)
MLR original data	7.78
MLR augmented data	1.59
MLR top hidden layer	1.67
MLR augmented data with sparse RBM	2.83
DBN fine tuned	1.10

TABLE II
The effect of features learned by a stack of RBMs, and by fine tuning a DBN

Surprisingly, using good shallow parameters did not yield the best results in the deep network case. In fact, lowering the weight decay slightly lowers the test set error in the deep case, even though it raised test set error in the shallow case. This indicates that perhaps test error is not the ideal metric to use when choosing the parameters of an RBM to initialize a DBN. Another surprising result is that SML performs about the same, if not slightly worse than CD in the deep case. At the very least, the dominance it seemed to hold over CD in the shallow case does not seem to translate to the deep case. The best results came from using CD-1 with the default parameters provided by [39]. Presumably this is because these parameters have been tuned specifically for good performance in a Deep Belief Network.

In order to choose good RBM parameters for use in a DBN, perhaps a different error metric could be used that would be more indicative of the performance of the DBN. One possibility is to use the reconstruction error, which is the difference between the data d, and the generated data $\widetilde{v}$ caused by one iteration of sampling $\widetilde{h} \sim p(h|d)$, followed by $\widetilde{v} \sim p(v|\widetilde{h})$. This may be more appropriate for predicting the quality of the features learned by the unsupervised portion of DBN training.

B. The Effect of Fine Tuning

From this point forward, we use CD-1 with the default parameters provided by Geoff Hinton's code. We now explore the affect of fine tuning on a DBN by training a stack of RBMs greedily, and then comparing the difference in error before and after performing backpropagation.

In order to assess the degree to which unsupervised learning of higher level features improves classification, and how much backpropagation helps, we augmented the original data with the higher level features obtained before backpropagation and trained a Multinomial Logistic Regression (MLR) classifier. We also trained MLR classifiers on the original data, and on the features from the top hidden layer only. The results are shown in Table II

Clearly the additional features improve classification significantly, and using more features including the original data seems to help. Supervised backpropagation changes the intermediate weights (and thus, features) to allow for greater linear separability in the top level of features, and gives the best results. Since most of the gain seems to come from

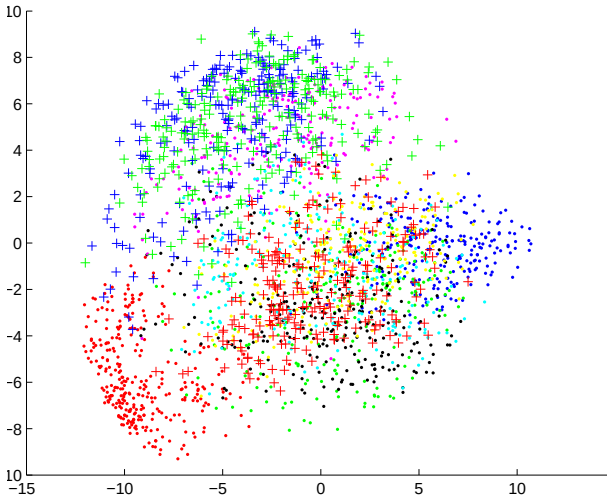
Layer	Mean Difference	Max. Difference	Initial Magnitude
1	0.0009	0.0030	0.0436
2	0.0020	0.0067	0.0564
3	0.0015	0.0138	0.0181

TABLE III
MEAN AND MAXIMUM SQUARED DISTANCE BETWEEN RECEPTIVE FIELDS OF EACH LAYER BEFORE AND AFTER BACKPROPAGATION

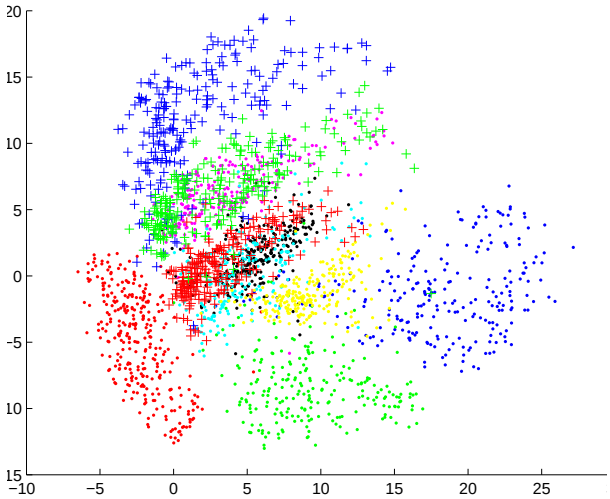
the initial greedy learning, it may be possible to use these features directly with a nonlinear classifier such as Support Vector Machines with a nonlinear Kernel, or Random Forests, instead of using backpropagation. Such procedures may yield improved results, or at least they might be faster to train.

Visually, the changes in the weights due to this fine tuning step are barely noticeable, suggesting that the original features learned by the stacked RBMs were quite good to begin with. To quantify this, we also show the average squared differences and the maximum squared differences between the weights before and after backpropagation in Table III. To get a sense of scale, we also show the averaged squared magnitude of the weights before backpropagation. It is clear that they hardly change up until the final layer, where some clearly change by a large magnitude, while on average they mostly stay fairly stable.

While it appears that the weights have been perturbed only slightly, there is certainly some degree of change which is evident in the change in classification performance. A different way of visualizing this effect is to see what happens to the actual data in the last hidden layer before and after backpropagation. To do this, we appended class labels to the data and trained a 794-1000-500-250-2 deep autoencoder with Gaussian activations in the last hidden layer [1]. This is different from a DBN classifier in that the network is unrolled after greedy training so that the last layer becomes the middle layer. The unrolled autoencoder initially has symmetric weights, but this constraint is not maintained during backpropagation. A data point is reconstructed by sending it through the autoencoder to the 794-unit output layer. After initial greedy training, we applied 30 epochs of backpropagation to minimize the cross-entropy error between the reconstructed data, and the actual data. In figure 7 we show the results before and after applying backpropagation to the projected mean-field data in the last hidden layer. A very pronounced separation of the classes takes place.



(a) Before backpropagation



(b) After backpropagation

Fig. 7. 2500 data points from MNIST in the 2-dimensional hidden layer of a deep autoencoder (a) before and (b) after backpropagation. Each class has been given a unique color/marker combination. (Best viewed in colour.)

V. CONCLUSION

This paper investigated strategies for training RBMs to achieve good classification performance in the shallow and deep learning settings. We examined many tricks that are used to to improve the performance of RBMs, as well as introduced a few previously unused ones. We also compared Stochastic Maximum Likelihood with Contrastive Divergence, and found that for training RBMs, SML consistently outperforms CD. Surprisingly, this did not seem to carry over to the deep case, where they performed somewhat equally.

It was found that tuning generative RBMs for good classification performance in the shallow setting does not necessarily translate to optimal performance in the deep learning setting. This is most likely caused by the fact that stacked RBMs and a feedforward neural network are two different models, and so optimizing one should not necessarily lead to optimal results in the other.

We also examined the training of Deep Belief Networks, such as the effects of global fine tuning through backpropagation, as well as alternative strategies for utilizing class information during the greedy stages. It was found that backpropagation hardly changes the weights learned from the

greedy stage, and yet it has a non-negligible impact on the final results. Again, this may simply be caused by using one model to initialize another. Interestingly, it was shown that most of the gains in performance are actually made during the greedy stage, and that in this context the gains made through backpropagation are fairly small.

This paper has a number of shortcomings, however. Firstly, the experiments were only performed on the MNIST data set, and at this point it seems that we have reached the limit for improvements that can be made on this data set using these models, making it incredibly difficult to determine if improvements are genuine, or simply the effect of noise. In order for these results to be conclusive, it will be necessary to carry these experiments out on a variety of data sets, over a large number of runs to factor out noise, and to truly see which tricks are the most useful across many different tasks. In addition, classification is only one type of problem, and other problems such as regression and coding should also be considered.

ACKNOWLEDGEMENTS

This work was supported by NSERC and CIFAR's Neural Computation and Adaptive Perception Program.

REFERENCES

[1] G. Hinton and R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *Science*, vol. 313, no. 5786, pp. 504–507, 2006.

[2] R. Salakhutdinov, A. Mnih, and G. Hinton, "Restricted Boltzmann machines for collaborative filtering," in *International Conference on Machine learning*, 2007, pp. 791–798.

[3] R. Salakhutdinov and G. Hinton, "Semantic hashing," *International Journal of Approximate Reasoning*, 2008.

[4] Y. Bengio, P. Lamblin, D. Popovici, H. Larochelle, and U. Montreal, "Greedy layer-wise training of deep networks," in *Advances in Neural Information Processing Systems*. MIT Press, 2007.

[5] Y. Bengio and Y. Le Cun, "Scaling learning algorithms towards AI," *Large-Scale Kernel Machines*, 2007.

[6] H. Lee, R. Grosse, R. Ranganath, and A. Ng, "Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations," in *International Conference on Machine Learning*, 2009.

[7] L. Wiskott and T. Sejnowski, "Slow feature analysis: Unsupervised learning of invariances," *Neural Computation*, vol. 14, no. 4, pp. 715–770, 2002.

[8] J. Hawkins and D. George, "Hierarchical temporal memory: Concepts, theory and terminology," Numenta, Tech. Rep., 2006.

[9] I. J. Goodfellow, Q. V. Le, A. M. Saxe, H. Lee, and A. Y. Ng., "Measuring invariances in deep networks," *Advances in neural information processing systems*, 2009.

[10] G. Hinton, S. Osindero, and Y. Teh, "A fast learning algorithm for deep belief nets," *Neural Computation*, vol. 18, no. 7, pp. 1527–1554, 2006.

[11] L. Younes, "Parametric inference for imperfectly observed Gibbsian fields," *Probability Theory and Related Fields*, vol. 82, no. 4, pp. 625–645, 1989.

[12] T. Tieleman, "Training restricted Boltzmann machines using approximations to the likelihood gradient," in *International conference on Machine Learning*, 2008, pp. 1064–1071.

[13] M. Welling, M. Rosen-Zvi, and G. Hinton, "Exponential family harmoniums with an application to information retrieval," *Advances in neural information processing systems*, vol. 17, pp. 1481–1488, 2005.

[14] G. E. Hinton, "Training products of experts by minimizing contrastive divergence," *Neural Computation*, vol. 14, p. 2002, 2002.

[15] M. Carreira-Perpinan and G. Hinton, "On contrastive divergence learning," in *Artificial Intelligence and Statistics*, vol. 2005, 2005.

[16] B. Marlin, "A direct proof that the true RBM parameters are a fixed point of both ML and CD1 in the asymptotic setting," 2008.

[17] A. Yuille, "The convergence of contrastive divergences," in *Advances in Neural Information Processing Systems*, 2004.

[18] A. Benveniste, M. Métivier, and P. Priouret, *Adaptive algorithms and stochastic approximations*. Springer-Verlag, 1990.

[19] D. Ruppert, "A Newton-Raphson version of the multivariate Robbins-Monro procedure," *Ann. Statist.*, vol. 13, no. 1, pp. 236–245, 1985.

[20] J. Spall, "Adaptive stochastic approximation by the simultaneous perturbation method," *IEEE Conference on Decision and Control*, pp. 3872–3879, 1998.

[21] B. T. Polyak, "A new method of stochastic approximation type," *Avtomat. i Telemekh.*, no. 7, pp. 98–107, 1990.

[22] B. Polyak and A. Juditsky, "Acceleration of stochastic approximation by averaging," *SIAM Journal on Control and Optimization*, vol. 30, p. 838, 1992.

[23] H. J. Kushner and H. Huang, "Averaging methods for the asymptotic analysis of learning and adaptive systems, with small adjustment rate," *SIAM J. Control Optim.*, vol. 19, no. 5, pp. 635–650, 1981.

[24] ——, "Asymptotic properties of stochastic approximations with constant coefficients," *SIAM J. Control Optim.*, vol. 19, no. 1, pp. 87–105, 1981.

[25] H. J. Kushner and G. G. Yin, *Stochastic Approximation Algorithms and Applications*. Springer-Verlag, 1997.

[26] L. Ljung and T. Söderström, *Theory and practice of recursive identification*. MIT Press, 1983.

[27] V. Borkar, *Stochastic Approximation: A Dynamical Systems Viewpoint*. Cambridge University Press, 2008.

[28] A. Bhaya and E. Kaszkurewicz, "Steepest descent with momentum for quadratic functions is a version of the conjugate gradient method," *Neural Networks*, vol. 17, no. 1, pp. 65 – 71, 2004.

[29] R. Sharma, W. Sethares, and J. Bucklew, "Analysis of momentum adaptive filtering algorithms," *IEEE Transactions on Signal Processing*, vol. 46, no. 5, pp. 1430–1434, May 1998.

[30] J. J. Hopfield, "Hopfield network," *Scholarpedia*, 2007.

[31] M. Welling, "Herding Dynamic Weights for Partially Observed Random Field Models," in *UAI*, 2009.

[32] A. Juditsky, A. Nazin, A. Tsybakov, and N. Vayatis, "Generalization error bounds for aggregation by mirror descent with averaging," *Advances in neural information processing systems*, 2005.

[33] B. Delyon, "General results on the convergence of stochastic algorithms," *IEEE Transactions on Automatic Control*, vol. 41, no. 9, pp. 1245–1255, 1996.

[34] C. Andrieu and J. Thoms, "A tutorial on adaptive MCMC," *Statistics and Computing*, vol. 18, no. 4, pp. 343–373, 2008.

[35] D. P. Bertsekas and J. N. Tsitsiklis, *Neuro-Dynamic Programming*. Athena Scientific, 1996.

[36] Y. LeCun and C. Cortes, "The MNIST database of handwritten digits," *NEC Research Institute, http://yann. lecun. com/exdb/mnist/index. html*.

[37] B. Polyak, "Some methods of speeding up the convergence of iterative methods," *USSR Computational Mathematics and Mathematical Physics.*, vol. 4, pp. 1–17, 1964.

[38] L. Bottou and O. Bousquet, "The tradeoffs of large scale learning," *Advances in neural information processing systems*, vol. 20, 2007.

[39] R. Salakhutdinov and G. Hinton, "Training a deep autoencoder or a classifier on MNIST digits — source code," 2006.

[40] H. Larochelle, Y. Bengio, J. Louradour, and P. Lamblin, "Exploring Strategies for Training Deep Neural Networks," *Journal of Machine Learning Research*, vol. 1, pp. 1–40, 2009.

[41] H. Lee, C. Ekanadham, and A. Ng, "Sparse deep belief net model for visual area V2," *Advances in neural information processing systems*, vol. 20, 2008.

[42] P. Garrigues and B. Olshausen, "Learning horizontal connections in a sparse coding model of natural images," *Advances in Neural Information Processing Systems*, vol. 20, pp. 505–512, 2008.

[43] H. Larochelle and Y. Bengio, "Classification using discriminative restricted Boltzmann machines," in *International Conference on Machine learning*, 2008, pp. 536–543.

[44] D. Erhan, P. Manzagol, Y. Bengio, S. Bengio, and P. Vincent, "The difficulty of training deep architectures and the effect of unsupervised pre-training," *AISTATS*, 2009.