Proceedings of Machine Learning Research 77:498–513, 2017 ACML 2017

Learning RBM with a DC programming Approach

Vidyadhar Upadhya VIDYADHAR@EE.IISC.ERNET.IN
Electrical Engineering Dept, Indian Institute of Science, Bangalore

P. S. Sastry SASTRY@EE.IISC.ERNET.IN
Electrical Engineering. Dept, Indian Institute of Science, Bangalore

Editors: Yung-Kyun Noh and Min-Ling Zhang

Abstract

By exploiting the property that the RBM log-likelihood function is the difference of convex functions, we formulate a stochastic variant of the difference of convex functions (DC) programming to minimize the negative log-likelihood. Interestingly, the traditional contrastive divergence algorithm is a special case of the above formulation and the hyperparameters of the two algorithms can be chosen such that the amount of computation per mini-batch is identical. We show that for a given computational budget the proposed algorithm almost always reaches a higher log-likelihood more rapidly, compared to the standard contrastive divergence algorithm. Further, we modify this algorithm to use the centered gradients and show that it is more efficient and effective compared to the standard centered gradient algorithm on benchmark datasets.

Keywords: RBM, Maximum likelihood learning, contrastive divergence, DC programming

1. Introduction

The Restricted Boltzmann Machines (RBM) (Smolensky, 1986; Freund and Haussler, 1994; Hinton, 2002) are among the basic building blocks of several deep learning models including Deep Boltzmann Machine (DBM) (Salakhutdinov and Hinton, 2009) and Deep Belief Networks (DBN) (Hinton et al., 2006). Even though they are used mainly as generative models, RBMs can be suitably modified to perform classification tasks also.

The model parameters in an RBM are learnt by maximizing the log-likelihood. However, the gradient (*w.r.t.* the parameters of the model) of the log-likelihood is intractable since it contains an expectation term *w.r.t.* the model distribution. This expectation is computationally expensive: exponential in (minimum of) the number of visible/hidden units in the model. Therefore, this expectation is approximated by taking an average over the samples from the model distribution. The samples are obtained using Markov Chain Monte Carlo (MCMC) methods which can be implemented efficiently by exploiting the bipartite connectivity structure of the RBM. The popular Contrastive Divergence (CD) algorithm is a stochastic gradient ascent on log-likelihood using the estimate of gradient obtained through MCMC procedure. However, the gradient estimate based on MCMC methods may be poor when the underlying density is high dimensional and can make the simple stochastic gradient descent (SGD) based algorithms to even diverge in some cases (Fischer and Igel, 2010).

There are two approaches to alleviate these issues. The first is to design an efficient MCMC method to get good representative samples from the model distribution and thus to get a reasonably accurate estimate of the gradient (Desjardins et al., 2010; Tieleman and Hinton, 2009). However, sophisticated MCMC methods are computationally intensive, in general. The second approach is to use the noisy estimate based on simple MCMC but employing sophisticated optimization strategies like second-order gradient descent (Martens, 2010), natural gradient (Desjardins et al., 2013), stochastic spectral descent (SSD) (Carlson et al., 2015),etc. These sophisticated optimization methods often result in additional computational costs.

In this paper, we follow the second approach and propose an efficient optimization procedure based on the so called difference of convex functions programming (or concave-convex procedure), by exploiting the fact that the RBM log-likelihood is the difference of two convex functions. Even for such a formulation we still need the intractable gradient which is approximated with the usual MCMC based sampling. We refer to the proposed algorithm as the *stochastic- difference of convex functions programming* (S-DCP) since the optimization involves stochastic approximation over the existing difference of convex functions programming approach.

What is interesting is that the resulting learning algorithm that we derive turns out to be an interesting and simple modification of the standard Contrastive Divergence, CD, which is the most popular algorithm for training RBMs. As a matter of fact, the CD can be seen to be a special case of our proposed algorithm. Thus our algorithm can be viewed as a small modification of CD which is theoretically motivated by looking at the problem as that of optimizing difference of convex functions. Although small, this modification to CD turns out to be important because our algorithm exhibits a much superior rate of convergence as we show through extensive simulations on benchmark data sets. Due to the similarity of our method with CD, it is possible to choose hyperparameters of the two algorithms such that the amount of computation per mini-batch is identical. Hence, we show that, for a fixed computational budget, our algorithm reaches a higher log-likelihood more rapidly compared to CD. Our empirical results provide a strong justification for preferring this method over the traditional SGD approaches.

Further, we modify the S-DCP algorithm to use the centered gradients (CG) as in Melchior et al. (2016), motivated by the principle that by removing the mean of the training data and the mean of the hidden activations from the visible and the hidden variables respectively the conditioning of the underlying optimizing problem can be improved (Montavon and Müller, 2012). The simulation results on benchmark data sets indicate that models learnt by S-DCP algorithm with centered gradients achieve better log-likelihood compared to the other standard methods.

It is important to note that the proposed method is a minorization-maximization algorithm and can be viewed as an instance of expectation maximization (EM) method. Since RBM is a graphical model with latent variables, many algorithms based on ML estimation, including CD, can be cast as a (generalized) expectation maximization (EM) algorithm. In itself, this EM view does not give any extra insight.

In fact, learning RBM using EM, alternate minimization and maximum likelihood approach are all similar (van de Laar and Kappen, 1994; Amari et al., 1992). Therefore, the proposed algorithm can be interpreted as a tool which provides better optimization dynamics to learn RBM with any of these approaches.

The rest of the paper is organized as follows. In section 2, we first briefly describe the RBM model and the maximum likelihood (ML) learning approach for RBM. We explain the proposed algorithm, the S-DCP, in section 3. In section 4, we describe the simulation setting and then present the results of our study. Finally, we conclude the paper in section 6.

2. Background

2.1. Restricted Boltzmann Machines

The Restricted Boltzmann Machine (RBM) is an energy based model with a two layer architecture, in which m visible stochastic units ($\mathbf{v}$) in one layer are connected to n hidden stochastic units ($\mathbf{h}$) in the other layer (Smolensky, 1986; Freund and Haussler, 1994; Hinton, 2002). Connections within a layer (i.e., visible to visible and hidden to hidden) are absent and the connections between the layers are undirected. This architecture is normally used as a generative model. The units in an RBM can take discrete or continuous values. In this paper, we consider the binary case , i.e., $\mathbf{v} \in \{0,1\}^m$ and $\mathbf{h} \in \{0,1\}^n$. The probability distribution represented by the model with parameters, θ, is

$$p(\mathbf{v},\mathbf{h}|\theta) = e^{-E(\mathbf{v},\mathbf{h};\theta)}/Z(\theta) \tag{1}$$

where, $Z(\theta) = \sum_{\mathbf{v},\mathbf{h}} e^{-E(\mathbf{v},\mathbf{h};\theta)}$ is the so called partition function and $E(\mathbf{v},\mathbf{h};\theta)$ is the energy function defined by

$$E(\mathbf{v},\mathbf{h};\theta) = -\sum_{i,j} w_{ij} h_i \, v_j - \sum_{j=1}^{m} b_j \, v_j - \sum_{i=1}^{n} c_i \, h_i$$

where $\theta = \{\mathbf{w} \in \mathbb{R}^{n\times m}, \mathbf{b} \in \mathbb{R}^m, \mathbf{c} \in \mathbb{R}^n\}$ is the set of model parameters. The w_{ij}, the $(i,j)^{\text{th}}$ element of $\mathbf{w}$, is the weight of the connection between the i^{th} hidden unit and the j^{th} visible unit. The bias for the i^{th} hidden unit and the j^{th} visible unit are denoted as c_i and b_j, respectively.

2.2. Maximum Likelihood Learning

The RBM parameters, θ, are learnt through the maximization of the log-likelihood over the training samples. The log-likelihood, given one training sample ($\mathbf{v}$), is given by,

$$\begin{aligned} \mathcal{L}(\theta|\mathbf{v}) &= \log\, p(\mathbf{v}|\theta) \\ &= \log \sum_{\mathbf{h}} p(\mathbf{v},\mathbf{h}|\theta) \\ &= \log \sum_{\mathbf{h}} e^{-E(\mathbf{v},\mathbf{h}:\theta)} - \log\, Z(\theta) \\ &\triangleq (g(\theta,\mathbf{v}) - f(\theta)) \end{aligned} \tag{2}$$

where we define

$$\begin{aligned} g(\theta, \mathbf{v}) &= \log \sum_{\mathbf{h}} e^{-E(\mathbf{v},\mathbf{h}:\theta)} \\ f(\theta) &= \log Z(\theta) = \log \sum_{\mathbf{v}',\mathbf{h}} e^{-E(\mathbf{v}',\mathbf{h};\theta)} \end{aligned} \tag{3}$$

The optimal RBM parameters are to be found by solving the following optimization problem.

$$\theta^* = \text{argmax}_\theta\, \mathcal{L}(\theta|\mathbf{v}) = \text{argmax}_\theta\, (g(\theta, \mathbf{v}) - f(\theta)) \tag{4}$$

The stochastic gradient descent iteratively updates the parameters as,

$$\theta^{t+1} = \theta^t + \eta\, \nabla_\theta \mathcal{L}(\theta|\mathbf{v})|_{\theta=\theta^t}$$

One can show that (Hinton, 2002; Fischer and Igel, 2012),

$$\begin{aligned} \nabla_\theta\, g(\theta, \mathbf{v}) &= -\frac{\sum_{\mathbf{h}} e^{-E(\mathbf{v},\mathbf{h}:\theta)} \nabla_\theta\, E(\mathbf{v}, \mathbf{h}; \theta)}{\sum_{\mathbf{h}} e^{-E(\mathbf{v},\mathbf{h}:\theta)}} = -\mathbb{E}_{p(\mathbf{h}|\mathbf{v};\theta)} \left[\nabla_\theta\, E(\mathbf{v}, \mathbf{h}; \theta)\right] \\ \nabla_\theta\, f(\theta) &= -\frac{\sum_{\mathbf{v}',\mathbf{h}} e^{-E(\mathbf{v}',\mathbf{h};\theta)} \nabla_\theta\, E(\mathbf{v}', \mathbf{h}; \theta)}{\sum_{\mathbf{v}',\mathbf{h}} e^{-E(\mathbf{v}',\mathbf{h};\theta)}} = -\mathbb{E}_{p(\mathbf{v}',\mathbf{h};\theta)} \left[\nabla_\theta\, E(\mathbf{v}', \mathbf{h}; \theta)\right] \end{aligned} \tag{5}$$

where $\mathbb{E}_q$ denotes the expectation *w.r.t.* the distribution q. The expectation under the conditional distribution, $p(\mathbf{h}|\mathbf{v};\theta)$, for a given $\mathbf{v}$, has a closed form expression and hence, $\nabla_\theta\, g$ is easily evaluated analytically. However, expectation under the joint density, $p(\mathbf{v}, \mathbf{h}; \theta)$, is computationally intractable since the number of terms in the expectation summation grows exponentially with the (minimum of) the number of hidden units/visible units present in the model. Hence, sampling methods are used to obtain $\nabla_\theta\, f$.

2.3. Contrastive Divergence

The contrastive divergence (Hinton, 2002), a popular algorithm to learn RBM, is based on Hastings-Metropolis-Gibbs sampling. In this algorithm, a single sample, obtained after running a Markov chain for K steps, is used to approximate the expectation as,

$$\begin{aligned} \nabla_\theta\, f(\theta) &= -\mathbb{E}_{p(\mathbf{v},\mathbf{h};\theta)} \left[\nabla_\theta\, E(\mathbf{v}, \mathbf{h}; \theta)\right] \\ &= -\mathbb{E}_{p(\mathbf{v};\theta)} \mathbb{E}_{p(\mathbf{h}|\mathbf{v};\theta)} \left[\nabla_\theta\, E(\mathbf{v}, \mathbf{h}; \theta)\right] \\ &\approx -\mathbb{E}_{p(\mathbf{h}|\tilde{\mathbf{v}}^{(K)};\theta)} \left[\nabla_\theta\, E(\tilde{\mathbf{v}}^{(K)}, \mathbf{h}; \theta)\right] \\ &\triangleq \hat{f}'(\theta, \tilde{\mathbf{v}}^{(K)}) \end{aligned} \tag{6}$$

Here $\tilde{\mathbf{v}}^{(K)}$ is the sample obtained after K transitions of the Markov chain (defined by the current parameter values θ) initialized with the training sample $\mathbf{v}$. A detailed description of the method is given as Algorithm 1. In practice, the mini-batch version of this algorithm is used. There exist many variations of this CD algorithm in the literature, such as persistent (PCD) (Tieleman, 2008), fast persistent (FPCD) (Tieleman and Hinton, 2009), population (pop-CD) (Oswin Krause, 2015), and average contrastive divergence (ACD) (Ma and Wang, 2016). Another popular algorithm, parallel tempering (PT) (Desjardins et al., 2010), is also based on MCMC.

Algorithm 1 CD-K update for a single training sample $\mathbf{v}$

```
**Input:** $\mathbf{v}, K, \theta^{(t)}, \eta$
$\tilde{\mathbf{v}}^{(0)} = \mathbf{v}$
**for** $k = 0$ **to** $K-1$ **do**
  sample $h_i^{(k)} \sim p(h_i|\tilde{\mathbf{v}}^{(k)}, \theta), \forall i$
  sample $\tilde{v}_j^{(k+1)} \sim p(v_j|\mathbf{h}^{(k)}, \theta), \forall j$
**end for**
**Output:** $\theta^{(t+1)} = \theta^{(t)} - \eta \left[\hat{f}'(\theta, \tilde{\mathbf{v}}^{(K)}) - \nabla g(\theta, \mathbf{v})\right]$
```

3. DC Programming Approach

The RBM log-likelihood is the difference between the functions f and g, both of which are log sum exponential functions and hence are convex. We can exploit this property to solve the optimization problem in (4) more efficiently by using the method known as the concave-convex procedure (CCCP) (Yuille et al., 2002) or the difference of convex functions programming (DCP) (An and Tao, 2005).

The DCP is an algorithm to solve optimization problems of the form,

$$\theta^* = \operatorname{argmin}_\theta F(\theta) = \operatorname{argmin}_\theta \left(f(\theta) - g(\theta)\right) \tag{7}$$

where, both the functions f and g are convex and F is smooth but non-convex. The DCP algorithm is an iterative procedure defined by

$$\theta^{(t+1)} = \operatorname{argmin}_\theta \left(f(\theta) - \theta^T \nabla g(\theta^{(t)})\right) \tag{8}$$

Note that each iteration as given above, solves a convex optimization problem because the objective function on the RHS of (8) is convex in θ. By differentiating this convex function and equating it to zero, we see that the iterates satisfy $\nabla f(\theta^{t+1}) = \nabla g(\theta^t)$. This gives an interesting geometric insight into this procedure. Given the current θ^t, if we can directly solve $\nabla f(\theta^{t+1}) = \nabla g(\theta^t)$ for θ^{t+1}, then we can use it. Otherwise we solve the convex optimization problem (as specified in eq. (8)) at each iteration through some numerical procedure.

In the RBM setting, F corresponds to the negative log-likelihood function and the functions f, g are as defined in (3). In our method, we propose to solve the convex optimization problem given by (8) by using gradient descent on $f(\theta) - \theta^T \nabla g(\theta^{(t)}, \mathbf{v})$. For this, we still need ∇f which is computationally intractable. We propose to use the sample based estimate (as in Contrastive Divergence) for this and do a fixed number (denoted as d) of SGD iterations to minimize $f(\theta) - \theta^T \nabla g(\theta^{(t)}, \mathbf{v})$. A detailed description of this proposed algorithm is given as Algorithm 2. In practice, the mini-batch version of this algorithm is used, which is given as Algorithm 3. We refer to this algorithm as the *stochastic-DCP* (S-DCP) algorithm.

By comparing Algorithm 2 with Algorithm 1 it is easily seen that S-DCP and CD are very similar when viewed as iterative procedures. As a matter of fact if we choose the hyperparameter d in S-DCP as 1, then they are identical. Therefore CD algorithm turns out to be a special case of S-DCP. Further, there is a similarity between the initialization

Algorithm 2 S-DCP update for a single training sample $\mathbf{v}$

```
**Input:** $\mathbf{v}, \theta^{(t)}, \eta, d, K'$
Initialize $\tilde{\theta}^{(0)} = \theta^{(t)}, \tilde{\mathbf{v}}^{(0)} = \mathbf{v}$
for $l = 0$ to $d-1$ do
  for $k = 0$ to $K'-1$ do
    sample $h_i^{(k)} \sim p(h_i|\tilde{\mathbf{v}}^{(k)}, \tilde{\theta}^{(l)}), \forall i$
    sample $\tilde{v}_j^{(k+1)} \sim p(v_j|\mathbf{h}^{(k)}, \tilde{\theta}^{(l)}), \forall j$
  end for
  $\tilde{\theta}^{(l+1)} = \tilde{\theta}^{(l)} - \eta \left[\hat{f}'(\tilde{\theta}^{(l)}, \tilde{\mathbf{v}}^{(K')}) - \nabla g(\theta^{(t)}, \mathbf{v})\right]$
  $\tilde{\mathbf{v}}^{(0)} = \tilde{\mathbf{v}}^{(K')}$
end for
**Output:** $\theta^{(t+1)} = \tilde{\theta}^{(d)}$
```

steps of S-DCP and PCD. Specifically, the step $\tilde{\mathbf{v}}^{(0)} = \tilde{\mathbf{v}}^{(K')}$ in Algorithm 2, retains the last state of the chain, to use it as the initial state in the next iteration. It is important to note that for each gradient descent step, unlike in PCD, in S-DCP we start the Gibbs chain on a given example. In other words, for each gradient descent step on negative log-likelihood, we have an inner loop that is several gradient descent steps on an auxiliary convex function. It is only during this inner loop that a single Gibbs chain is maintained.

The hyperparameter d in S-DCP, controls the number of descent steps in the inner loop that optimizes the convex function $f(\theta) - \theta^T \nabla g(\theta^{(t)}, \mathbf{v})$. Since we are using the SGD on a convex function it is likely to be better behaved and thus, S-DCP may find better descent directions on the log likelihood of RBM as compared to CD. This may be the case even when the gradient ∇f obtained through MCMC is noisy. This also means we may be able to trade more steps on this gradient descent with fewer number of iterations of the MCMC. This allows us more flexibility in using a fixed computational budget.

We further modify the proposed S-DCP to use the centered gradients. We follow the Algorithm 1 given in Melchior et al. (2016). The detailed description of the CS-DCP algorithm is given as Algorithm 4.

In standard DCP or CCCP, one assumes that the optimization problem on the RHS of (8) is solved exactly to show that the algorithm given by (8) is a proper descent procedure for the problem defined by (7). However, all that we need to show this is to ensure that we get descent on $f(\theta) - \theta^T \nabla g(\theta^{(t)})$ in each iteration. In our case, each component of the gradient of f and g is an expectation of a binary random variable (as can be seen from eq. (5)) and hence is bounded by one. Thus, euclidean norms of gradients of both f and g are bounded by $\sqrt{(mn + m + n)}$, which is the square root of the dimension of the parameter vector, θ. Since both f and g are convex, this implies that gradients of f and g are globally Lipschitz (chapter 3, Bubeck (2015)) with known Lipschitz constant. Hence, we can have a constant step-size gradient descent on $f(\theta) - \theta^T \nabla g(\theta^{(t)})$ that ensures descent on each iteration.

Since we are using a noisy estimate for the gradient in the inner loop as explained above, the standard convergence proof for CCCP is not really applicable for the S-DCP. At present we do not have a full convergence proof for the algorithm because it is not easy to obtain

Algorithm 3 S-DCP update for a mini-batch of size N_B

```
**Input:** $V = [\mathbf{v}^{(0)}, \mathbf{v}^{(1)}, \ldots, \mathbf{v}^{(N_B-1)}], \theta^{(t)}, \eta, d, K'$
Initialize $\tilde{\theta}^{(0)} = \theta^{(t)}, V_T = V, \Delta\theta = 0$
for $l = 0$ to $d - 1$ do
  for $i = 0$ to $N_B - 1$ do
    $\tilde{\mathbf{v}}^{(0)} = V_T[:, i] \quad \rightarrow [i^{\text{th}}$ column of $V_T]$
    for $k = 0$ to $K' - 1$ do
      sample $h_i^{(k)} \sim p(h_i|\tilde{\mathbf{v}}^{(k)}, \tilde{\theta}^{(l)}), \forall i$
      sample $\tilde{v}_j^{(k+1)} \sim p(v_j|\mathbf{h}^{(k)}, \tilde{\theta}^{(l)}), \forall j$
    end for
    $\Delta\theta = \Delta\theta + \left[\hat{f}'(\tilde{\theta}^{(l)}, \tilde{\mathbf{v}}^{(K')}) - \nabla g(\theta^{(t)}, \mathbf{v}^{(i)})\right]$
    $V_T[:, i] = \tilde{\mathbf{v}}^{(K')}$
  end for
  $\tilde{\theta}^{(l+1)} = \tilde{\theta}^{(l)} - \eta \frac{\Delta\theta}{N_B}$
end for
**Output:** $\theta^{(t+1)} = \tilde{\theta}^{(d)}$
```

good bounds on the error in estimating ∇f through MCMC. However, the simulation results that we present later show that S-DCP is effective and efficient.

3.1. Computational Complexity

Let us suppose that the computational cost of one Gibbs transition is T and that of evaluating g (and also $\hat{f}'$) is L. The computational cost of the CD-K algorithm for a mini-batch of size N_B is $(N_B(KT+2L))$. The S-DCP algorithm with K' MCMC steps and d inner loop iteration has cost $(d\,N_B(K'T + L) + N_B L)$. For the S-DCP algorithm, $N_B L$ is not multiplied by d because $\nabla g(\theta^{(t)}, \mathbf{v}^{(i)})$ is evaluated only once for all the samples in the mini-batch. The computational cost of both CD and S-DCP algorithms can be made equal by choosing $dK' = K$, if we neglect the term $(d-1)L$. This is acceptable since, L is much smaller than T, and d is also small (Otherwise we can choose K' and d to satisfy $KT = dK'T + (d-1)L$ to make the computational cost of both the algorithms identical).

4. Experiments and Discussions

In this section, we give a detailed comparison between the S-DCP/CS-DCP and other standard algorithms like CD, PCD, centered gradient (CG)(Melchior et al., 2016) and stochastic spectral descent (SSD)(Carlson et al., 2015). In order to see the advantages of the S-DCP over these CD based algorithms, we compare them by keeping the computational complexity same (for each mini-batch). Due to this computational architecture, the learning speed in terms of actual time is proportional to speed in terms of iterations. We analyse the learning behavior by varying the hyperparameters, namely, the learning rate and the batch size. We also provide simulation results showing the effect of hyperparameters, d and K' on S-DCP learning. Further, the sensitivity to initialization is also discussed.

Algorithm 4 CS-DCP update for a mini-batch of size N_B

```
**Input:** $V = [\mathbf{v}^{(0)}, \mathbf{v}^{(1)}, \ldots, \mathbf{v}^{(N_B-1)}], \theta^{(t)}, \eta, d, K', \boldsymbol{\mu}, \boldsymbol{\lambda}, \nu_\mu, \nu_\lambda$
Initialize $\tilde{\theta}^{(0)} = \theta^{(t)}, V_T = V, \Delta\theta = 0, V_n = V_T$
Calculate $H_p[t,i] = p(h_t = 1|V_T[:,i])\ \forall t, i$    /* $p(h_t = 1|\mathbf{v}) = \sigma\left((\mathbf{v} - \boldsymbol{\mu})^T w_{*t} + c_t\right)$/*
Calculate $\boldsymbol{\mu}_{batch} = \text{Column\_mean}(V_T)$
          $\boldsymbol{\lambda}_{batch} = \text{Column\_mean}(H_p)$
**for** $l = 0$ **to** $d - 1$ **do**
  **for** $i = 0$ **to** $N_B - 1$ **do**
    $\tilde{\mathbf{v}}^{(0)} = V_n[:,i] \quad \rightarrow [i^{\text{th}}$ column of $V_n]$
    **for** $k = 0$ **to** $K' - 1$ **do**
      sample $h_i^{(k)} \sim p(h_i|\tilde{\mathbf{v}}^{(k)}, \tilde{\theta}^{(l)}), \forall i$    /* $p(h_i = 1|\mathbf{v}) = \sigma\left((\mathbf{v} - \boldsymbol{\mu})^T w_{*j} + c_j\right)$/*
      sample $\tilde{v}_j^{(k+1)} \sim p(v_j|\mathbf{h}^{(k)}, \tilde{\theta}^{(l)}), \forall j$    /* $p(v_j = 1|\mathbf{h}) = \sigma\left(w_{i*}(\mathbf{h} - \boldsymbol{\lambda}) + b_i\right)$/*
    **end for**
    $V_n[:,i] = \tilde{\mathbf{v}}^{(K')},\ H_n[t,i] = p(h_t = 1|\tilde{\mathbf{v}}^{(K')})\ \forall t$
  **end for**
  $\mathbf{b} = \mathbf{b} + \nu_\lambda \mathbf{W}^{(l)}(\boldsymbol{\lambda}_{batch} - \boldsymbol{\lambda})$    /* Re-parameterization */
  $\mathbf{c} = \mathbf{c} + \nu_\mu \mathbf{W}^{(l)}(\boldsymbol{\mu}_{batch} - \boldsymbol{\mu})$
  $\boldsymbol{\mu} = (1 - \nu_\mu)\boldsymbol{\mu} + \nu_\mu \boldsymbol{\mu}_{batch}$    /* moving average with sliding factor*/
  $\boldsymbol{\lambda} = (1 - \nu_\lambda)\boldsymbol{\lambda} + \nu_\lambda \boldsymbol{\lambda}_{batch}$
  **if** $l == 0$ **then**
    $\nabla_w \mathbf{g} = \frac{(V_T - \boldsymbol{\mu})(H_p - \boldsymbol{\lambda})^T}{N_B}$ /* Gradient of $\mathbf{g}$ $w.r.t.$ $w$ is evaluated only once*/
  **end if**
  $\nabla\mathbf{W} = \nabla_w \mathbf{g} - \frac{(V_n - \boldsymbol{\mu})(H_n - \boldsymbol{\lambda})^T}{N_B}$
  $\nabla\mathbf{b} = \text{Column\_mean}(V_T) - \text{Column\_mean}(V_n)$
  $\nabla\mathbf{c} = \text{Column\_mean}(H_p) - \text{Column\_mean}(H_n)$
  $\mathbf{W}^{(l+1)} = \mathbf{W}^{(l)} + \eta\nabla\mathbf{W}$
  $\mathbf{b}^{(l+1)} = \mathbf{b}^{(l)} + \eta\nabla\mathbf{b}$
  $\mathbf{c}^{(l+1)} = \mathbf{c}^{(l)} + \eta\nabla\mathbf{c}$
  $\tilde{\theta}^{(l+1)} = \{\mathbf{W}^{(l+1)}, \mathbf{b}^{(l+1)}, \mathbf{c}^{(l+1)}\}$
**end for**
**Output:** $\theta^{(t+1)} = \tilde{\theta}^{(d)}$
```

4.1. The Experimental Set-up

We consider four benchmark datasets in our analysis namely Bars & Stripes (MacKay, 2003), Shifting Bar (Melchior et al., 2016), MNIST[1] (LeCun et al., 1998) and Omniglot[2] (Lake et al., 2015). The Bars & Stripes dataset consists of patterns of size $D \times D$ which are generated as follows. First, all the pixels in each row are set to zero or one with equal probability. Second, the pattern is rotated by 90 degrees with a probability of 0.5. We have used $D = 3$, for which we get 14 distinct patterns. The Shifting Bar dataset consists of patterns of size N where a set of B consecutive pixels with cyclic boundary conditions are set to one and the others are set to zero. We have used $N = 9$ and $B = 1$, for which we get

1. statistically binarized as in (Salakhutdinov and Murray, 2008)
2. https://github.com/yburda/iwae/tree/master/datasets

9 distinct patterns. We refer to these as small datasets. The other two datasets, MNIST and Omniglot have data dimension of 784 and we refer to these as large datasets. All these datasets are used in the literature to benchmark the RBM learning.

For small datasets, we consider RBMs with 4 hidden units and for large datasets, we consider RBMs with 500 hidden units. The biases of hidden units are initialized to zero and the weights are initialized to samples drawn from a Gaussian distribution with mean zero and standard deviation 0.01. The biases of visible units are initialized to the inverse sigmoid of the training sample mean.

We use multiple trials, where each trial starts with a particular initial configuration for weights and biases. For small datasets, we use 25 trials and for large datasets we use 10 trials. In order to have a fair comparison, we make sure that all the algorithms start with the same initial setting. We did not use any stopping criterion. Instead we learn the RBM for a fixed number of epochs. The training is performed for 50000 epochs for the small datasets and 200 epochs for the large datasets. The mini-batch learning procedure is used and the training dataset is shuffled after every epoch. However, for small datasets, i.e., Bars & Stripes and Shifting Bar, full batch training procedure is used. We use batch size of 200 for large data sets, unless otherwise stated .

We compare the performance of S-DCP and CS-DCP with CD, PCD and CD with centered gradient (CG). We also compare it with the SSD algorithm. We keep the computational complexity of S-DCP and CS-DCP same as that of CD based algorithms by choosing K, d and K' as prescribed in Section 3.1. Since previous works stressed on the necessity using large K for CD to get a sensible generative model (Salakhutdinov and Murray, 2008; Carlson et al., 2015), we use $K = 24$ (with $d = 6, K' = 4$ for DCP) for large datasets and $K = 12$ (with $d = 3, K' = 4$ for DCP) for small datasets. In order to get an unbiased comparison, we did not use momentum and weight decay for any of the algorithms.

For comparison with the centered gradient method, we use the Algorithm 1 given in Melchior et al. (2016) which corresponds to dd^b_s in their notation. However we use CD step size $K = 24$ compared to single step CD used in Algorithm 1 given in Melchior et al. (2016). The hyperparameters ν_μ and ν_λ are set to 0.01. The initial value of μ is set to mean of the training data and $\boldsymbol{\lambda}$ is set to $\mathbf{0.5}$. The CS-DCP algorithm also uses the same hyperparameter settings.

The SSD algorithm (Carlson et al., 2015) is a bound optimization algorithm which exploits the convexity of functions, f and g. Since, the proposed algorithm in this paper is also designed to exploit the convexity of f and g, we compare our results with that of the SSD algorithm. It is important to note that SSD has an extra cost of computing the singular value decomposition of a matrix of size $m \times n$ which costs $\mathcal{O}(mn \min(m, n))$ for each gradient update. Here, m and n are the number of hidden and visible units, respectively. We have observed that for the small data sets the SSD algorithm results in divergence of the log-likelihood. We also observed similar behavior for large data sets when small batch size is used. Therefore performance of the SSD algorithm is not shown for those cases. Since SSD is shown to work well when the batch size is of the order 1000, we compare S-DCP and CS-DCP under this setting. We use exactly the same settings as used in Carlson et al. (2015) and report the performance on the statistically binarized MNIST dataset. For the

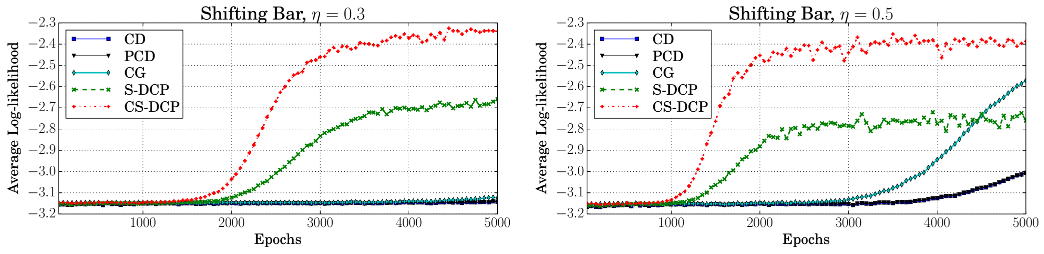


Figure 1: The performance of different algorithms on Shifting Bar dataset

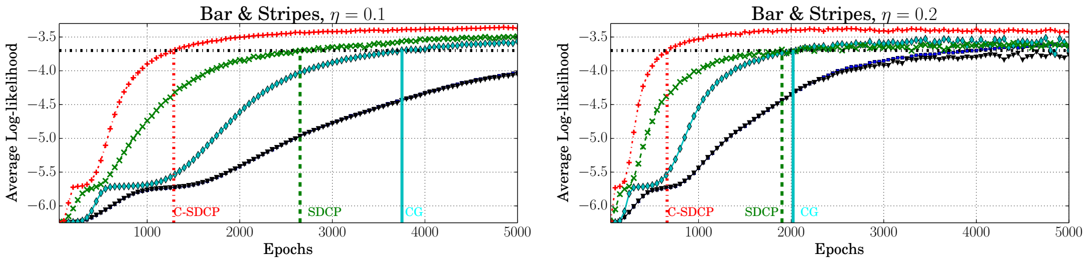


Figure 2: The performance of different algorithms on Bars & Stripes

S-DCP, we use $d = 6$ and $K' = 4$ to match the CD steps of $K = 25$ used in Carlson et al. (2015) and the batch size 1000.

The performance comparison is based on the log-likelihood achieved on the test set. We show the mean of the Average Test Log-Likelihood (denoted as ATLL) over all trials. For small RBMs, the ATLL, is evaluated exactly as,

$$ATLL = \sum_{i=1}^{N} \log\, p(\mathbf{v}_{\text{test}}^{(i)}|\theta) \tag{9}$$

However for large RBMs, we estimate the ATLL with annealed importance sampling (Neal, 2001) with 100 particles and 10000 intermediate distributions according to a linear temperature scale between 0 and 1.

4.2. Performance Comparison

In this section, we present a number of experimental results to illustrate the performance of S-DCP and CS-DCP in comparison with the other methods. In most of the experiments, we observe that the ATLL achieved by CS-DCP is greater than that by CD and other variants. Further, it achieves this relatively early in the learning stage.

4.2.1. Small data sets

Fig. 1 and Fig. 2 present results obtained with different algorithms on *Bars & Stripes* and *Shifting Bar* data sets respectively[3]. The theoretical upper bounds for the ATLL are -2.6 and -2.2 for *Bars* & *Stripes* and *Shifting Bar* datasets respectively (Melchior et al., 2016).

3. Please note that all the figures presented are best viewed in color.

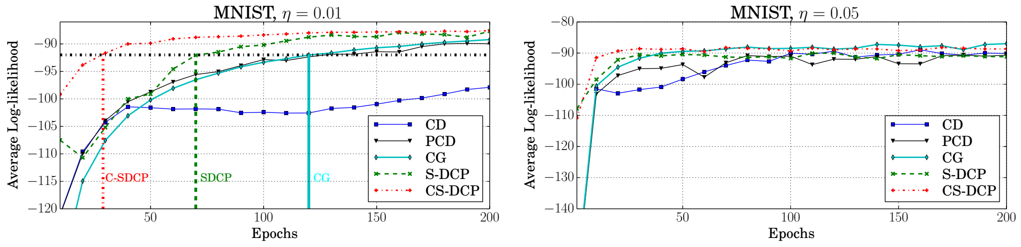


Figure 3: The performance of different algorithms on MNIST dataset .

For the *Shifting Bar* dataset we find the CD and PCD algorithms become stuck in a local minimum achieving ATLL around -3.15 for both the learning rates $\eta = 0.3$ and 0.5. We observe that the CG algorithm is able to escape from this local minima when a higher learning rate of $\eta = 0.5$ is used. However, S-DCP and CS-DCP algorithms are able to reach almost the same ATLL independent of the learning rate. We observed that in order to make the CD, PCD and CG algorithms learn a model achieving ATLL comparable to those of S-DCP or CS-DCP, more than 30000 gradient updates are required. This behavior is reported in several other experiments (Melchior et al., 2016).

For the *Bars & Stripes* dataset, the CS-DCP/S-DCP and CG algorithms perform almost the same in terms of the achieved ATLL. However CD and PCD algorithms are sensitive to the learning rate and fail to achieve the ATLL achieved by CS-DCP/S-DCP. Moreover, the speed of learning, indicated in the figure by the epoch at which the model achieve 90% of the maximum ATLL, shows the effectiveness of both CS-DCP and S-DCP algorithms. In specific, CG algorithm requires around 2500 epochs more of training compared to the CS-DCP algorithm.

The experimental results on these two data sets indicate that both CS-DCP and S-DCP algorithms are able to provide better optimization dynamics compared to the other standard algorithms and they converge faster.

4.2.2. Large data sets

Fig. 3 and Fig. 4 show the results obtained using the MNIST and OMNIGLOT data sets respectively. For the MNIST dataset, we observe that CS-DCP converges faster when a small learning rate is used. However, all the algorithms achieve almost the same ATLL at the end of 200 epochs. For the OMNIGLOT dataset, we see that both S-DCP and CS-DCP algorithms are superior to all the other algorithms both in terms of speed and the achieved ATLL.

The experimental results obtained on these two data sets indicate that both S-DCP and CS-DCP perform well even when the dimension of the model is large. Their speed of learning is also superior to that of other standard algorithms. For example, when $\eta = 0.01$ the CS-DCP algorithm achieves 90% of the maximum ATLL approximately 100 epochs before the *CG* algorithm does as indicated by the vertical lines in Fig. 3.

As mentioned earlier, we use batch size 1000 to compare the S-DCP and CS-DCP with the SSD algorithm. We use exactly the same settings as used in Carlson et al. (2015) and report the performance on the statistically binarized MNIST dataset. For the S-DCP and

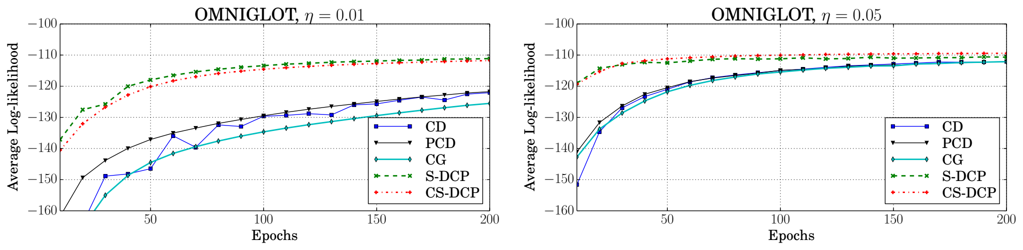


Figure 4: The performance of different algorithms on OMNIGLOT dataset

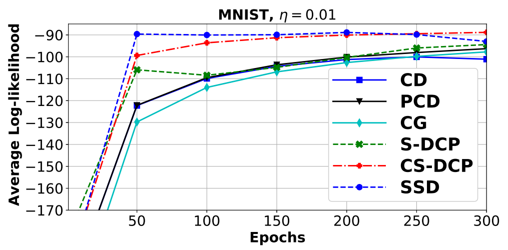


Figure 5: Comparison of SSD with CS-DCP and S-DCP with a batch size of 1000 on MNIST dataset.

CS-DCP, we use $d = 6$ and $K' = 4$ to match the CD steps of $K = 25$ used in Carlson et al. (2015). The results presented in Fig. 5 indicate that both CS-DCP and SSD have similar performance and are superior compared to all the other algorithms. The SSD is slightly faster since it exploits the specific structure in solving the optimization problem. However, as noted earlier SSD algorithm does not generalize well for small data sets and also when small batch size is used for learning large data sets. Even for a large batch size of 1000 the divergence can be noted in Fig. 5 at the end of 250 epochs. In addition, the SSD algorithm is computationally expensive.

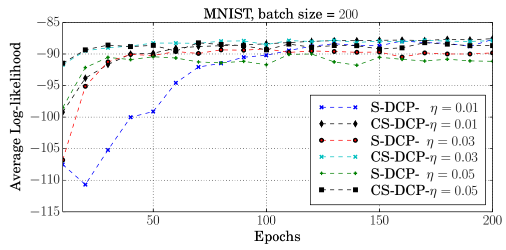
5. Sensitivity to hyperparameters

5.1. Effect of Learning Rate

The learning rate is a crucial hyperparameter in learning an RBM. If the learning rate is small we can ensure descent on each iteration. It can be seen from Fig. 6(a) that for a small learning rate $\eta = 0.01$, as epoch progress, there is an increase in the ATLL for most epochs.

5.2. Effect of d and K'

Fig. 6(b) depicts the learning behavior of S-DCP and CS-DCP algorithms when the hyperparameters d and K' are varied, keeping the computational complexity per batch fixed. We fix $dK' = 24$ and vary d and K'. As d is increased, the number of gradient updates per epoch increases leading to faster learning indicated by the sharp increase in the ATLL.

(a)


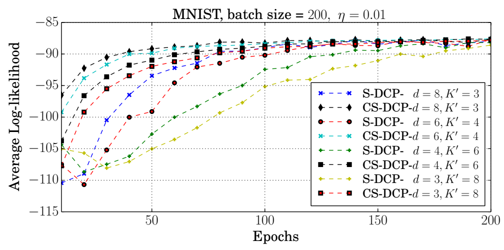
(b)


Figure 6: Comparison of S-DCP and CS-DCP algorithms with different (a) learning rates and (b) values of d and K' keeping $dK' = 24$.

However, irrespective of the values d and K' both the algorithms converge eventually to the same ATLL. We find a similar trend in the performance across other data sets as well.

In a way, the above behavior is one of the major advantages of the proposed algorithm. For example, if a longer Gibbs chain is required to obtain a good representative samples from a model, it is better to choose a smaller d and a larger K'. In such scenarios, the S-DCP and CS-DCP algorithms provide a better control on the dynamics of learning.

5.3. Effect of Batch Size

A small batch size is suggested in Hinton (2010) for better learning with a simple SGD approach. However, in Carlson et al. (2015), advantages of using a large batch size of the order of $2m$ (twice the number of hidden units) with a well designed optimization, is demonstrated. If we fix the batch size, then the learning rate can be fixed through cross-validation. However, to analyse the effect of batch size on S-DCP and CS-DCP learning, we fix the learning rate, $\eta = 0.01$, and then vary the batch size. As shown in Fig. 7(a), the S-DCP eventually reaches almost the same ATLL, after 200 epochs of training, irrespective of the batch size considered. Therefore, the proposed S-DCP algorithm is found to be less sensitive to batch size. We observed a similar behavior with the Omniglot dataset.

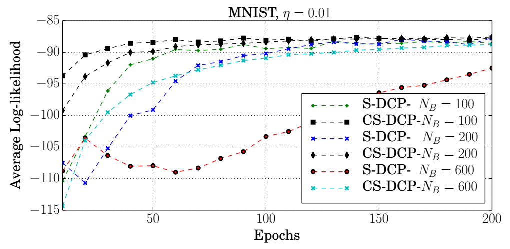
5.4. Sensitivity to Initialization

We initialize the weights with samples drawn from a Gaussian distribution having mean zero and standard deviation, σ. We consider a total of four cases: $\sigma = 0.01$ and $\sigma = 0.001$ and the biases of the visible units are initialized to zero or to the inverse sigmoid of the training sample mean (termed as base rate initialization). It is noted in Hinton (2010) that initializing the visible biases to the sample mean improves the learning behavior. The biases of the hidden units are set to zero in all the experiments.

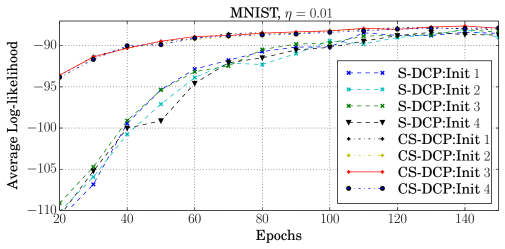
The results shown in Fig. 7(b) indicate that both S-DCP and CS-DCP algorithms are less sensitive to the initial values of weights and biases. We observed similar behavior with the other datasets.

Table 1: Legend Notation for Fig. 7(b).

	Init 1	Init 2	Init 3	Init 4
σ	0.001	0.01	0.001	0.01
b	**0**	**0**	base rate	base rate



(a)



(b)

Figure 7: MNIST dataset: Learning with different (a) batch sizes (b) Initialization

6. Conclusions and Future Work

A major issue in learning an RBM is that the log-likelihood gradient is to be obtained with MCMC sampling and hence is noisy. There are several optimization techniques proposed to obtain fast and stable learning in spite of this noisy gradient. In this paper, we proposed a new algorithm, S-DCP, that is computationally simple but achieves higher efficiency of learning compared to CD and its variants, the current popular method for learning RBMs.

We exploited the fact that the RBM log-likelihood function is a difference of convex functions and adopted the standard DC programming approach for maximizing the log-likelihood. We used SGD for solving the convex optimization problem, approximately, in each step of the DC programming. The resulting algorithm is very similar to CD and has the same computational complexity. As a matter of fact, CD can be obtained as a special case of our proposed S-DCP approach. Through extensive empirical studies, we illustrated the advantages of S-DCP over CD. We also presented a centered gradient version CS-DCP. We showed that S-DCP/CS-DCP is very resilient to the choice of hyperparameters through simulation results. The main attraction of S-DCP/CS-DCP, in our opinion, is its simplicity compared to other sophisticated optimization techniques that are proposed in literature for this problem.

The fact that the log-likelihood function of RBM is a difference of convex functions, opens up new areas to explore, further. For example, the inner loop of S-DCP solves a convex optimization problem using SGD. A proper choice of an adaptable step size could make it more efficient and robust. Also, there exist many other efficient techniques for solving such convex optimization problems. Investigating the suitability of such algorithms for the case where we have access only to noisy gradient values (obtained through MCMC samples) is also an interesting problem for future work.

Acknowledgments

We thank the NVIDIA Corporation for the donation of the Titan X Pascal GPU used for this research.

References

Shun-ichi Amari, Koji Kurata, and Hiroshi Nagaoka. Information geometry of Boltzmann machines. *IEEE Transactions on neural networks*, 3(2):260–271, 1992.

Le Thi Hoai An and Pham Dinh Tao. The DC (difference of convex functions) programming and DCA revisited with DC models of real world nonconvex optimization problems. *Annals of Operations Research*, 133(1):23–46, 2005. ISSN 1572-9338. .

Sébastien Bubeck. Convex optimization: Algorithms and complexity. *Found. Trends Mach. Learn.*, 8(3-4):231–357, November 2015. ISSN 1935-8237. .

David Carlson, Volkan Cevher, and Lawrence Carin. Stochastic spectral descent for restricted Boltzmann machines. In *Proceedings of the Eighteenth International Conference on Artificial Intelligence and Statistics*, pages 111–119, 2015.

Guillaume Desjardins, Aaron Courville, and Yoshua Bengio. Adaptive parallel tempering for stochastic maximum likelihood learning of RBMs. *arXiv preprint arXiv:1012.3476*, 2010.

Guillaume Desjardins, Razvan Pascanu, Aaron C. Courville, and Yoshua Bengio. Metric-free natural gradient for joint-training of Boltzmann machines. *CoRR*, abs/1301.3545, 2013.

Asja Fischer and Christian Igel. Empirical analysis of the divergence of Gibbs sampling based learning algorithms for restricted Boltzmann machines. In *Artificial Neural Networks–ICANN 2010*, pages 208–217. Springer, 2010.

Asja Fischer and Christian Igel. An introduction to restricted Boltzmann machines. In *Progress in Pattern Recognition, Image Analysis, Computer Vision, and Applications*, pages 14–36. Springer, 2012.

Yoav Freund and David Haussler. *Unsupervised learning of distributions of binary vectors using two layer networks*. Computer Research Laboratory [University of California, Santa Cruz], 1994.

Geoffrey Hinton. A practical guide to training restricted Boltzmann machines. *Momentum*, 9(1):926, 2010.

Geoffrey E Hinton. Training products of experts by minimizing contrastive divergence. *Neural computation*, 14(8):1771–1800, 2002.

Geoffrey E. Hinton, Simon Osindero, and Yee Whye Teh. A fast learning algorithm for deep belief nets. *Neural Computation*, 18:1527–1554, 2006.

Brenden M. Lake, Ruslan Salakhutdinov, and Joshua B. Tenenbaum. Human-level concept learning through probabilistic program induction. *Science*, 350(6266):1332–1338, 2015. ISSN 0036-8075. .

Yann LeCun, Léon Bottou, Genevieve B. Orr, and Klaus-Robert Müller. Effiicient backprop. In *Neural Networks: Tricks of the Trade, This Book is an Outgrowth of a 1996 NIPS Workshop*, pages 9–50, London, UK, UK, 1998. Springer-Verlag. ISBN 3-540-65311-2.

Xuesi Ma and Xiaojie Wang. Average contrastive divergence for training restricted Boltzmann machines. *Entropy*, 18(1):35, 2016.

David JC MacKay. *Information theory, inference, and learning algorithms*, volume 7. Cambridge university press Cambridge, 2003.

James Martens. Deep learning via hessian-free optimization. In *ICML*, 2010.

Jan Melchior, Asja Fischer, and Laurenz Wiskott. How to center deep Boltzmann machines. *Journal of Machine Learning Research*, 17(99):1–61, 2016.

Grégoire Montavon and Klaus-Robert Müller. *Deep Boltzmann Machines and the Centering Trick*, pages 621–637. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012. ISBN 978-3-642-35289-8. .

Radford M Neal. Annealed importance sampling. *Statistics and Computing*, 11(2):125–139, 2001.

Christian Igel Oswin Krause, Asja Fischer. Population-contrastive-divergence: Does consistency help with RBM training? *CoRR*, abs/1510.01624, 2015.

Ruslan Salakhutdinov and Geoffrey E Hinton. Deep Boltzmann machines. In *AISTATS*, volume 1, page 3, 2009.

Ruslan Salakhutdinov and Iain Murray. On the quantitative analysis of deep belief networks. In *Proceedings of the 25th international conference on Machine learning*, pages 872–879. ACM, 2008.

Paul Smolensky. Information processing in dynamical systems: Foundations of harmony theory. 1986.

Tijmen Tieleman. Training restricted Boltzmann machines using approximations to the likelihood gradient. In *Proceedings of the 25th international conference on Machine learning*, pages 1064–1071. ACM, 2008.

Tijmen Tieleman and Geoffrey Hinton. Using fast weights to improve persistent contrastive divergence. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 1033–1040. ACM, 2009.

Pierre van de Laar and Bert Kappen. Boltzmann machines and the em algorithm. 1994.

Alan L Yuille, Anand Rangarajan, and AL Yuille. The concave-convex procedure (CCCP). *Advances in neural information processing systems*, 2:1033–1040, 2002.