Efficient Learning of Restricted Boltzmann Machines Using Covariance Estimates

Vidyadhar Upadhya VIDYADHARU@IISC.AC.IN and
P S Sastry SASTRY@IISC.AC.IN
Indian Institute of Science Bangalore, India

Editors: Wee Sun Lee and Taiji Suzuki

Abstract

Learning RBMs using standard algorithms such as CD(k) involves gradient descent on the negative log-likelihood. One of the terms in the gradient, which involves expectation w.r.t. the model distribution, is intractable and is obtained through an MCMC estimate. In this work we show that the Hessian of the log-likelihood can be written in terms of covariances of hidden and visible units and hence, all elements of the Hessian can also be estimated using the same MCMC samples with small extra computational costs. Since inverting the Hessian may be computationally expensive, we propose an algorithm that uses inverse of the diagonal approximation of the Hessian, instead. This essentially results in parameter-specific adaptive learning rates for the gradient descent process and improves the efficiency of learning RBMs compared to the standard methods. Specifically we show that using the inverse of diagonal approximation of Hessian in the stochastic DC (difference of convex functions) program approach results in very efficient learning of RBMs.

Keywords: RBM, Maximum likelihood learning, Difference of Convex (DC) algorithm, Contrastive divergence

1. Introduction

The Restricted Boltzmann Machine (RBM), an energy based generative model (Smolensky, 1986; Freund and Haussler, 1994; Hinton, 2002), is among the basic building blocks of several deep learning models including Deep Boltzmann Machine (DBM) and Deep Belief Networks (DBN) (Salakhutdinov and Hinton, 2009; Hinton et al., 2006; Montúfar, 2018). It can also be used as a discriminative model with suitable modifications.

The traditional method of learning the parameters of an RBM involves minimizing the KL divergence between the data and the model distribution. This is equivalent to the maximum likelihood estimation and is implemented as a gradient ascent on the log-likelihood. However, evaluating the gradient (*w.r.t.* the parameters of the model) of the log-likelihood is computationally expensive (exponential in minimum of the number of visible/hidden units in the model) since it contains an expectation term *w.r.t.* the model distribution. Therefore, in the iterative stochastic gradient methods this term is approximated using samples from the model distribution. The samples are obtaining using Markov Chain Monte Carlo (MCMC) methods which are efficient in this regard due to RBM's bipartite connectivity structure. The popular Contrastive Divergence (CD) algorithm uses samples obtained through an MCMC procedure with a specific initialization strategy. However, the resulting estimated gradient may be poor when the RBM model is high dimensional. The poor

estimate can make the stochastic gradient descent (SGD) based algorithms such as CD to even diverge in some cases (Fischer and Igel, 2010).

There are two general approaches to make the learning of RBMs more efficient. The first is to design an efficient MCMC method to get good representative samples from the model distribution and thereby reduce the variance of the estimated gradient (Desjardins et al., 2010; Tieleman and Hinton, 2009). However, advanced MCMC methods are computationally intensive, in general. The second approach is to design better optimization strategies which are robust to the noise in the estimated gradient (Martens, 2010; Desjardins et al., 2013; Carlson et al., 2015). Most approaches to design better optimization algorithms for learning RBMs are second order optimization techniques that either need approximate Hessian inverse or an estimate of the inverse Fisher matrix (The two approaches differ for the RBM since it contains hidden units). The Hessian-Free (H-F) algorithm (Martens, 2010) is an iterative procedure which approximately solves a linear system to obtain the curvature through matrix-vector product. In (Desjardins et al., 2013) H-F algorithm is used to design natural gradient descent for learning Boltzmann machines. A sparse Gaussian graphical model is proposed in (Grosse and Salakhutdinov, 2015) to estimate the inverse Fisher matrix in order to devise factorized natural gradient descent procedure. All these algorithms either need additional computations to solve an auxiliary linear system or are computationally intensive algorithms to directly estimate the inverse Fisher matrix.

There have been attempts to exploit the fact that the RBM log-likelihood function is a difference of convex functions by modifying the standard difference of convex programming (DCP) approach to handle the stochasticity (Upadhya and Sastry, 2017; Nitanda and Suzuki, 2017). The *stochastic- difference of convex functions programming* (S-DCP) algorithm (Upadhya and Sastry, 2017) uses only the first order derivatives of the log-likelihood and solves a series of convex optimization problems using constant step-size gradient descent method for a fixed number of iterations. The *Stochastic proximal DC* (SPD) algorithm (Nitanda and Suzuki, 2017) uses an additional proximal term along with the DC objective function and solves series of convex optimization problems. Unlike S-DCP, the SPD solves each subproblem to a certain level of accuracy (predefined). In order to achieve the required accuracy level large minibatch is used which significantly increases the computational cost (Xu et al., 2019). However, the computational cost of S-DCP algorithm can be made identical to that of CD based algorithms with a proper choice of hyperparameters and is shown to perform well compared to other algorithms (Upadhya and Sastry, 2017).

Motivated by the simplicity and the efficiency of the S-DCP algorithm, in this work, we modify the S-DCP algorithm using the diagonal approximation of the Hessian of the log-likelihood and propose a diagonally scaled S-DCP, denoted as S-DCP-D. Use of a diagonal approximation of Hessian essentially amounts to having an adaptive stepsize which is different for different parameters.

We show that the diagonal terms of the Hessian can be expressed in terms of the covariances of the visible and hidden units and can be estimated using the same MCMC samples that are used to get the gradient estimates. Therefore, the additional computational cost incurred is small. Thus, the main contribution of the paper is a well-motivated algorithm (with small additional computational costs) that can automatically adopt the step-size (through the inverse of the diagonal approximation of the Hessian) to improve the

efficiency of learning an RBM. Through empirical investigations we show the effectiveness of the proposed algorithm.

The rest of the paper is organized as follows. In section 2, we briefly describe the RBM model and the maximum likelihood (ML) learning approach for RBM. We explain the proposed algorithm, S-DCP-D, in section 3. In section 4, we present simulation results on some benchmark datasets to show the efficiency of S-DCP-D. Finally, we conclude the paper in section 5.

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). There are no connections from visible to visible and hidden to hidden nodes and the connections between the layers are undirected. An RBM with parameters θ represents a probability distribution

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

where, $Z_\theta = \sum_{\mathbf{v},\mathbf{h}} e^{-E(\mathbf{v},\mathbf{h};\theta)}$ is the normalizing constant which is called the partition function and $E(\mathbf{v}, \mathbf{h}; \theta)$ is the energy function. The energy function is defined based on the type of units, discrete or continuous. In this work, we consider binary units, i.e., $\mathbf{v} \in \{0, 1\}^m$ and $\mathbf{h} \in \{0, 1\}^n$ for which the energy function is defined as

$$E(\mathbf{v}, \mathbf{h}; \theta) = -\sum_{i,j} w_{ij} h_i \, v_j - \sum_{j=1}^{m} b_j \, v_j - \sum_{i=1}^{n} c_i \, h_i \tag{2}$$

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. Here, w_{ij} is the weight of the connection between the i^{th} hidden unit and the j^{th} visible unit. The c_i and b_j denote the bias for the i^{th} hidden unit and the j^{th} visible unit, respectively.

2.2. Maximum Likelihood Learning

One of the methods to learn the RBM parameters, θ, is through the maximization of the log-likelihood over the training samples. The log-likelihood, for a given training sample ($\mathbf{v}$), is given by,

$$\begin{aligned} \mathcal{L}(\theta|\mathbf{v}) &= \log p(\mathbf{v}|\theta) = \log \sum_{\mathbf{h}} p(\mathbf{v}, \mathbf{h}|\theta) \\ &\triangleq g(\theta, \mathbf{v}) - f(\theta) \end{aligned} \tag{3}$$

where,

$$\begin{aligned} g(\theta, \mathbf{v}) &= \log \sum_{\mathbf{h}} e^{-E(\mathbf{v},\mathbf{h};\theta)} \\ f(\theta) &= \log Z_\theta = \log \sum_{\mathbf{v}',\mathbf{h}} e^{-E(\mathbf{v}',\mathbf{h};\theta)}. \end{aligned} \tag{4}$$

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

$$\theta^* = \arg\max_{\theta} \mathcal{L}(\theta|\mathbf{v}) = \arg\max_{\theta} \left(g(\theta, \mathbf{v}) - f(\theta)\right) \tag{5}$$

The above optimization problem is solved using an iterative gradient ascent procedure:

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

The gradient of g and f are given by (Hinton, 2002; Fischer and Igel, 2012),

$$\begin{aligned}
\nabla_\theta \, g(\theta, \mathbf{v}) &= -\frac{\sum_{\mathbf{h}} e^{-E(\mathbf{v},\mathbf{h};\theta)} \nabla_\theta \, E(\mathbf{v}, \mathbf{h}; \theta)}{\sum_{\mathbf{h}} e^{-E(\mathbf{v},\mathbf{h};\theta)}} \\
&= -\mathbb{E}_{p(\mathbf{h}|\mathbf{v};\theta)} \left[\nabla_\theta \, E(\mathbf{v}, \mathbf{h}; \theta)\right] \\
\nabla_\theta \, f(\theta) &= -\frac{\sum_{\mathbf{v}',\mathbf{h}} e^{-E(\mathbf{v}',\mathbf{h};\theta)} \nabla_\theta \, E(\mathbf{v}', \mathbf{h}; \theta)}{\sum_{\mathbf{v}',\mathbf{h}} e^{-E(\mathbf{v}',\mathbf{h};\theta)}} \\
&= -\mathbb{E}_{p(\mathbf{v}',\mathbf{h};\theta)} \left[\nabla_\theta \, E(\mathbf{v}', \mathbf{h}; \theta)\right]
\end{aligned} \tag{6}$$

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 (minimum of) the number of hidden units/visible units present in the model. Hence, sampling methods are used to obtain $\nabla_\theta \, f$.

The contrastive divergence (Hinton, 2002), a popular algorithm to learn RBMs, uses a single sample (obtained after running a Markov chain for K steps) to approximate the expectation as,

$$\begin{aligned}
\nabla_\theta \, f(\theta) &= -\mathbb{E}_{p(\mathbf{v},\mathbf{h};\theta)} \left[\nabla_\theta \, E(\mathbf{v}, \mathbf{h}; \theta)\right] \\
&= -\mathbb{E}_{p(\mathbf{v};\theta)} \mathbb{E}_{p(\mathbf{h}|\mathbf{v};\theta)} \left[\nabla_\theta \, E(\mathbf{v}, \mathbf{h}; \theta)\right] \\
&\approx -\mathbb{E}_{p(\mathbf{h}|\tilde{\mathbf{v}}^{(K)};\theta)} \left[\nabla_\theta \, E(\tilde{\mathbf{v}}^{(K)}, \mathbf{h}; \theta)\right] \\
&\triangleq \hat{f}'(\theta, \tilde{\mathbf{v}}^{(K)})
\end{aligned} \tag{7}$$

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 $\mathbf{v}$. 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), average contrastive divergence (ACD) (Ma and Wang, 2016) and weighted contrastive divergence (WCD) (Merino et al., 2018). Another popular algorithm, parallel tempering (PT) (Desjardins et al., 2010), is also based on MCMC. All these algorithms differ in the way they obtain representative samples from the model distribution for estimating the gradient. The centered gradient (CG) (Montavon and Müller, 2012) algorithm also uses the same principle as that of CD algorithm to obtain the samples; however, while estimating the gradient it removes the mean of the training data and the mean of the hidden activations from the visible and the hidden variables respectively. This approach has been seen to

Algorithm 1 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: $\theta^{(t+1)} = \tilde{\theta}^{(d)}$

improve the conditioning of the underlying optimizing problem (Montavon and Müller, 2012).

As mentioned earlier, here we propose S-DCP-D which is a modification of the S-DCP algorithm (Upadhya and Sastry, 2017). The S-DCP approach is advantageous since a non-convex problem is solved by iteratively solving a sequence of convex optimization problems.

3. Diagonally scaled S-DCP (S-DCP-D)

The DCP (Yuille et al., 2002; An and Tao, 2005) is an algorithm useful for solving optimization problems of the form,

$$\theta^* = \arg\min_{\theta} F(\theta) = \arg\min_{\theta} \left(f(\theta) - g(\theta)\right) \tag{8}$$

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

$$\theta^{(t+1)} = \arg\min_{\theta} \left(f(\theta) - \theta^T \nabla g(\theta^{(t)})\right). \tag{9}$$

In the RBM setting, F corresponds to the negative log-likelihood function and the functions f, g are as defined in eq. (4).

In the S-DCP algorithm, the convex optimization problem given by RHS of eq. (9) is (approximately) solved using a few iterations of gradient descent on $f(\theta) - \theta^T \nabla g(\theta^{(t)}, \mathbf{v})$ for which the ∇f is estimated using samples obtained though MCMC (as in Contrastive Divergence). Thus, it is a stochastic gradient descent for the (convex) objective function $f(\theta) - \theta^T \nabla g(\theta^{(t)}, \mathbf{v})$ for a fixed number of iterations (denoted as d). A description of this S-DCP algorithm is given as Algorithm 1. Note that, it is possible to choose the hyperparameters d and K' such that the amount of computation required is identical to CD(K) algorithm (Upadhya and Sastry, 2017).

The S-DCP algorithm can be viewed as two loops. The outer loop is the iteration given by eq. (9). Each iteration here involves a convex optimization which is (approximately) solved by the inner loop of S-DCP through stochastic gradient descent (w.r.t. θ) on the

convex function, $f(\theta) - \theta^T \nabla g(\theta^{(t)})$. The proposed S-DCP-D is a scaling of this stochastic gradient descent by using the diagonal elements of the Hessian of this convex function.

The Hessian of the objective function $f(\theta) - \theta^T \nabla g(\theta^{(t)})$ can be obtained as,

$$\begin{aligned}
&\nabla_\theta^2 f(\theta) = -\nabla_\theta \mathbb{E}_{p(\mathbf{v},\mathbf{h};\theta)} \left[\nabla_\theta E(\mathbf{v},\mathbf{h};\theta)\right] \\
&= -\sum_{\mathbf{v},\mathbf{h}} \nabla_\theta E(\mathbf{v},\mathbf{h})\ \nabla_\theta p(\mathbf{v},\mathbf{h};\theta)^T \text{ (Since } \nabla_\theta^2 E(\mathbf{v},\mathbf{h})=0) \\
&= -\sum_{\mathbf{v},\mathbf{h}} \frac{\nabla_\theta E(\mathbf{v},\mathbf{h})}{Z_\theta^2} e^{-E(\mathbf{v},\mathbf{h})} (-Z_\theta\, \nabla_\theta E(\mathbf{v},\mathbf{h})^T - \nabla_\theta Z_\theta^T) \\
&= -\sum_{\mathbf{v},\mathbf{h}} \nabla_\theta E(\mathbf{v},\mathbf{h}) \left[-\nabla_\theta E(\mathbf{v},\mathbf{h})^T - \nabla_\theta \log Z_\theta^T\right] p(\mathbf{v},\mathbf{h})
\end{aligned}$$

By substituting $\nabla_\theta \log Z_\theta = -\mathbb{E}_{p(\mathbf{v}',\mathbf{h}';\theta)} \left[\nabla_\theta E(\mathbf{v}',\mathbf{h}';\theta)\right]$ from eq. (6) in the above equation, we get,

$$\nabla_\theta^2 f(\theta) = Cov_{p(\mathbf{v},\mathbf{h})} \left[\nabla_\theta E(\mathbf{v},\mathbf{h}), \nabla_\theta E(\mathbf{v},\mathbf{h})\right] \tag{10}$$

where, $Cov_q(X,X) = \mathbb{E}_q(XX^T) - \mathbb{E}_q(X)\mathbb{E}_q(X^T)$.

Note that a typical element in $\nabla_\theta^2 f$ is $\frac{\partial^2 f}{\partial\theta_i \partial\theta_j}$ where θ_i refers to the parameters of the RBM, namely, all the w_{ij}, b_i, c_j. The diagonal element corresponding to w_{ij} is

$$\begin{aligned}
\frac{\partial^2 f}{\partial w_{ij} \partial w_{ij}} &= \mathbb{E}_{p(\mathbf{v},\mathbf{h})} \left[\left(\frac{\partial f}{\partial w_{ij}}\right)^2\right] - \left(\mathbb{E}_{p(\mathbf{v},\mathbf{h})} \left[\frac{\partial f}{\partial w_{ij}}\right]\right)^2 \\
&= \mathbb{E}_{p(\mathbf{v},\mathbf{h})} \left[v_j^2\, h_i^2\right] - \left(\mathbb{E}_{p(\mathbf{v},\mathbf{h})} \left[v_j\, h_i\right]\right)^2 \\
&= \mathbb{E}_{p(\mathbf{v},\mathbf{h})} \left[v_j\, h_i\right] - \left(\mathbb{E}_{p(\mathbf{v},\mathbf{h})} \left[v_j\, h_i\right]\right)^2 \\
&= \mathbb{E}_{p(\mathbf{v},\mathbf{h})} \left[v_j\, h_i\right] \left(1 - \mathbb{E}_{p(\mathbf{v},\mathbf{h})} \left[v_j\, h_i\right]\right)
\end{aligned}$$

We have used the property that $v_j^2 = v_j$ and $h_i^2 = h_i$ (since v_j, h_i are binary random variables) in the above derivation. Similarly, the diagonal terms corresponding to the bias terms are given by,

$$\begin{aligned}
\frac{\partial^2 f}{\partial b_j \partial b_j} &= \mathbb{E}_{p(\mathbf{v},\mathbf{h})} \left[v_j\right] - \left(\mathbb{E}_{p(\mathbf{v},\mathbf{h})} \left[\, v_j\right]\right)^2 \\
\frac{\partial^2 f}{\partial c_i \partial c_i} &= \mathbb{E}_{p(\mathbf{v},\mathbf{h})} \left[h_i\right] - \left(\mathbb{E}_{p(\mathbf{v},\mathbf{h})} \left[\, h_i\right]\right)^2
\end{aligned}$$

By using the above equations, the diagonal elements of the Hessian of f can be estimated simply by using the same MCMC samples used for gradient estimates. For a compact notation, the diagonal terms in $\nabla_\theta^2 f$ can be written as

$$\text{Diag}\left(\nabla_\theta^2 f\right) = -\mathbb{E}_{p(\mathbf{v},\mathbf{h})} \left[\nabla_\theta E(\mathbf{v},\mathbf{h})\right] \odot \left(\mathbf{1} + \mathbb{E}_{p(\mathbf{v},\mathbf{h})} \left[\nabla_\theta E(\mathbf{v},\mathbf{h})\right]\right)$$

where $\odot$ represents element-wise multiplication, $\mathbf{1}$ represents vector of all ones and $\text{Diag}(\boldsymbol{A})$ represents the vector consisting of the diagonal elements of matrix $\boldsymbol{A}$.

Algorithm 2 S-DCP-D update for a mini-batch of size N_B

Input: $V = [\mathbf{v}^{(0)}, \mathbf{v}^{(1)}, \ldots, \mathbf{v}^{(N_B-1)}], \theta^{(t)}, \eta, d, K', \epsilon$
Initialize $\tilde{\theta}^{(0)} = \theta^{(t)}, V_T = V$
for $l = 0$ **to** $d-1$ **do**
  $\Delta\theta = \mathbf{0}, G_f = \mathbf{0}$
  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
    $\Delta\theta = \Delta\theta + \left[\hat{f}'(\tilde{\theta}^{(l)}, \tilde{\mathbf{v}}^{(K')}) - \nabla g(\theta^{(t)}, \mathbf{v}^{(i)})\right]$
    $G_f = G_f + \hat{f}'(\tilde{\theta}^{(l)}, \tilde{\mathbf{v}}^{(K')})$
    $V_T[:, i] = \tilde{\mathbf{v}}^{(K')}$
  end for
  $H_f = \frac{G_f}{N_B} \odot \left(1 - \frac{G_f}{N_B}\right)$ /* $\odot$ represents element-wise multiplication */
  $\tilde{\theta}_s^{(l+1)} = \tilde{\theta}_s^{(l)} - \frac{\eta}{H_{f_s}+\epsilon}\frac{\Delta\theta_s}{N_B}, \ \forall s$ /* H_{f_s} is the diagonal element corresponding to θ_s */
end for
Output: $\theta^{(t+1)} = \tilde{\theta}^{(d)}$

These estimates are used in obtaining the gradient descent updates (in the inner loop S-DCP) as,

$$\theta_l^{t+1} = \theta_l^t - \eta \left.\frac{\left[\nabla_\theta \left(f(\theta) - \theta^T \nabla g(\theta^{(t)})\right)\right]_l}{[H_t + \epsilon \mathbf{I})]_l}\right|_{\theta=\theta^t} \tag{11}$$

where $[\mathbf{a}]_l$ represents l^{th} element of vector $\mathbf{a}$, $\mathbf{I}$ is the identity matrix of appropriate dimension, H_t is the estimated Hessian at iteration t and ϵ is a small constant (and the term ϵI is added for numerical stability). A detailed description of the proposed algorithm is given as Algorithm 2.

The inverse of the diagonal approximation of the Hessian essentially provides parameter-specific learning rates for the gradient ascent process. In case of S-DCP algorithm the objective function for the gradient descent is convex and the diagonal terms of the $\nabla^2 f$ are greater than or equal to zero since f is convex. Therefore, inverse of the diagonal terms of the Hessian added with a small ϵ is numerically stable.

Since the estimate of the gradient is noisy, the estimated Hessian is also noisy. Therefore, exponential averaging of the estimated Hessian is used to make the algorithm stable in terms of learning. Let $\tilde{H}_t$ denote the $\nabla^2_\theta f$ calculated at iteration t as explained earlier. Let H_t denote the Hessian that is used at iteration t for updating the weights. We calculate H_t as

$$H_t = \lambda_H H_{t-1} + (1 - \lambda_H)\tilde{H}_t \tag{12}$$

where λ_H is a parameter that decides the memory of the exponential averaging.

3.1. Computational Complexity

The computational cost of the CD(K) algorithm for a mini-batch of size N_B is ($N_B(KT + 2L)$) where T is the cost for one Gibbs transition and L is the cost for evaluating ∇g (and also $\hat{f}'$). The S-DCP algorithm with K' MCMC transitions and d inner loop iterations has cost ($d\,N_B(K'T + L) + N_B L$). The computational cost of CD(K) and S-DCP are identical if K' and d are chosen to satisfy $KT = dK'T + (d-1)L$(Upadhya and Sastry, 2017). (By choosing $K = dK'$ we can make the computational costs of the two algorithms nearly equal). The difference between S-DCP and S-DCP-D is only in terms of estimating the diagonal terms of the Hessian. An additional $d(mn + m + n)$ elementwise multiplications are required to obtain the estimate of the the diagonal of Hessian. This represents the additional computational cost of S-DCP-D compared to that of S-DCP.

4. Experiments and Discussions

In this section, we give a detailed comparison between the S-DCP-D and other algorithms, namely, centered gradient (CG) (Melchior et al., 2016), S-DCP, CD and PCD algorithms. The CG algorithm is essentially a CD(k) algorithm with additional centering heuristic which improves learning. Further, the objective here is to compare algorithms which have similar computational complexity and hence we do not consider algorithms which are significantly computationally expensive (SPD, H-F, etc).

4.1. The Experimental Set-up

We consider four benchmark datasets in our analysis, namely, Bars & Stripes (MacKay, 2003), MNIST[1] (LeCun et al., 1998), CalTech 101 Silhouettes DataSet (Marlin, 2009) and *kannada*-MNIST (Prabhu, 2019). The Bars & Stripes dataset of data dimension $D \times D$ is generated using a two-step procedure. In the first step, all the pixels in each row are set to zero or one with equal probability and then the pattern is rotated by 90 degrees with a probability of 0.5 in the second step. We have choose $D = 3$, for which we get 16 distinct patterns. We refer to MNIST, CalTech and the *kannada*-MNIST datasets as large datasets. The MNIST , CalTech 101 Silhouettes and the *kannada*-MNIST datasets have data dimension of 784.

For the Bars & Stripes dataset, we consider three RBMs with $4, 8, 16$ hidden units and for the large datasets, we consider RBMs with 500 hidden units. We evaluate the algorithms using the performance measures obtained from multiple trials, where each trial fixes the initial configuration of the weights and biases. The biases of visible units and hidden units are initialized to the inverse sigmoid of the training sample mean and zero, respectively. The weights are initialized to samples drawn from a Gaussian distribution with mean zero and standard deviation 0.01. We use 25 trials for the Bars & Stripes dataset and 10 trials for the large datasets. The mini-batch learning procedure is used and the training dataset is shuffled after every epoch. However, for Bars & Stripes dataset full batch training procedure is used. We learn the RBM for a fixed number of epochs and avoid using any stopping criterion. The training is performed for 5000 epochs for Bars & Stripes dataset

1. statistically binarized as in (Salakhutdinov and Murray, 2008)

(corresponding to 5000 gradient updates, due to full batch training) and 200 epochs for the MNIST dataset (corresponding to $60,000$ gradient updates due to the batch size of 200).

We compare the performance of the proposed S-DCP-D with centered gradient (CG),S-DCP, CD and PCD. We keep the computational complexity (on each mini-batch) of S-DCP roughly the same as that of CD by choosing K, d and K' such that $K = dK'$ (Upadhya and Sastry, 2017). Since previous works stressed on the necessity of using a large K for CD based algorithms to get a sensible generative model (Carlson et al., 2015; Salakhutdinov and Murray, 2008), we use $K = 24$ in CD (with $d = 6, K' = 4$ for S-DCP) for large datasets and $K = 4$ in CD (with $d = 2, K' = 2$ for S-DCP) for Bars & Stripes dataset. In order to get an unbiased comparison, we did not use momentum and weight decay for any of the algorithms. For the centered gradient algorithm, we use the Algorithm 1 in (Melchior et al., 2016) which corresponds to dd_s^b in their notation. We use CD step size $K = 24$ and the hyperparameters ν_μ and ν_λ are set to 0.01. The initial value of μ is set to the mean of the training data and $\boldsymbol{\lambda}$ is set to $\mathbf{0.5}$.

The learning rate and other hyperparameters for each algorithm is set to obtain the best performance by doing a grid search over a set of values of hyperparameters.

4.2. Evaluation Criterion

The performance comparison is based on the log-likelihood achieved on the training and test samples. For comparing the speed of learning of different algorithms, the average train log-likelihood is a reasonable measure. The average test log likelihood also indicates how well the learnt model generalizes. We show the maximum (over all trials) of the average train and test log-likelihood. The average test log-likelihood (denoted as ATLL) is evaluated as,

$$ATLL = \frac{1}{N}\sum_{i=1}^{N} \log\, p(\mathbf{v}_{\text{test}}^{(i)}|\theta) \tag{13}$$

We evaluate the average train log-likelihood similarly by using the training samples rather than the test samples. For small RBMs the above expression can be evaluated exactly. 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.

The evaluation in terms of the generative ability of the learnt models is carried out by observing the samples that they generate. We randomly initialize the states of the visible units and run the alternating Gibbs Sampler for 5000 steps (for large datasets)/200 steps (for Bars & Stripes dataset) and plot the state of the visible units.

Overall, we use three evaluation criteria to show the effectiveness of the proposed S-DCP-D algorithm, specifically, i) speed of convergence ii) generalization (Average Test log-likelihood) and iii) generative ability (quality of the generated samples).

4.3. Performance Comparison

In this section, we present experimental results to illustrate the performance of S-DCP-D in comparison with the other algorithms (CG, S-DCP, CD and PCD). The algorithms are implemented using Python and CUDAMat (A CUDA-based matrix class for Python

bindings)(Mnih, 2009) on a system with Intel processor $i7-7700$ (4 CPU cores and processor base frequency 3.60 GHz), NVIDIA Titan X Pascal GPU and 16 GB RAM configuration.

In our results we show that the speed of learning, in terms of number of training epochs, exhibited by S-DCP-D is significantly higher compared to the other algorithms. As mentioned earlier, all three algorithms have comparable computational load (per minibatch) and hence comparison in terms of number of epochs would be similar to comparison in terms of actual running time. However, since the computations performed by the different algorithms are not identical, we need to understand difference in computational time per epoch of different algorithms as well. For this, we present below the actual computational time of different algorithms for a fixed number of epochs.

The mean and standard deviation(σ) of the utilized system time in seconds, for 5000 epochs of learning for Bars & Stripes dataset and for 200 epochs of learning for large datasets, for each algorithm over 10 trials are shown in the table below.

Table 1: The system time statistics for Bars & Stripes and large datasets. The mean and standard deviation of system time (in seconds) is evaluated over 10 trials.

Algorithm	Bars & Stripes		MNIST/CalTech/*kannada*-MNIST	
	Mean	σ	Mean	σ
CD/PCD	1.41	0.02	248.15	0.16
CG	1.79	0.07	309.51	0.17
S-DCP	1.73	0.06	317.87	0.39
S-DCP-D	1.92	0.02	385.65	0.17

As can be seen from table 1, the computational time for S-DCP is 3% (for large datasets) more compared to that of CG. As mentioned earlier, by taking $K = dK'$ we can make the computational time of these two algorithms nearly same. Compared to S-DCP, the time for S-DCP-D is about 8% more for Bars & Stripes and 24% more for MNIST/CalTech. The additional computation for S-DCP-D is calculating diagonal of Hessian and this grows linearly with m, n.

In all results presented here we show evolution of ATLL with number of epochs for different algorithms.[2] As would be seen from the results, the S-DCP-D is faster in terms of number of epochs by much more than 25% thus justifying the claim that it results in efficient learning. In addition, on large datasets, the ATLL achieved by S-DCP-D is also larger.

4.3.1. Bars & Stripes

Fig. 1 shows the evolution of the mean and maximum ATLL achieved by the RBM with 4 hidden units, learnt for the Bars & Stripes dataset. (Note that here all patterns are used for training and hence there is no distinction between training and test data sets). As can be seen, the S-DCP-D has significantly higher speed of learning compared to S-DCP indicating

2. Since we do not employ any stopping criterion, we cannot give 'time taken to learn' for different algorithms; we can only show how log likelihood evolves with number of training epochs.

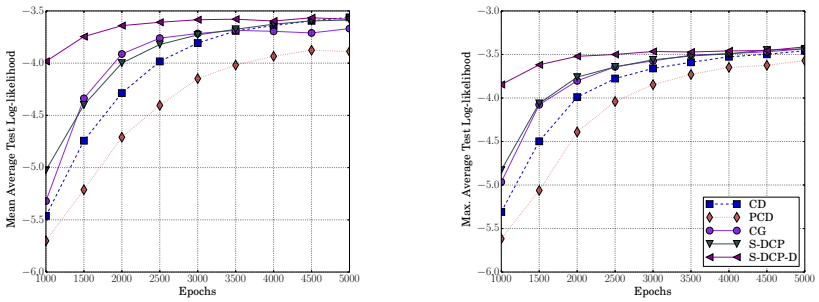


Figure 1: The evolution of mean and maximum average log-likelihood acheived on the training and the test set over all the trials on the Bars & Stripes dataset.

the effectiveness of the parameter-specific learning rate induced by the diagonal scaling. It is also faster than CG, CD and PCD. This increased speed does not come at the expense of accuracy; the final ATLL of all algorithms is roughly same though S-DCP and CG take more epochs to converge. Further, all the three learnt models generate valid samples as shown in Fig. 5. We observed similar behavior with RBM models having number of hidden units 8 and 16.

4.3.2. Large Datasets

Fig. 2, 3,4 show the evolution of the mean and maximum average log-likelihood of the test and training set for the MNIST, CalTech and *kannada*-MNIST datasets respectively. The convergence of S-DCP-D is faster compared to both S-DCP and CG. We observe in Fig. 3 that the S-DCP-D evolution is smoother compared to S-DCP which suggests that the stability of the learning algorithm is improved by the parameter-specific learning rate employed. Further, the ATLL evolution in Fig. 2 indicates that the generalization ability of the model learnt using S-DCP-D is comparable to that learnt by the other algorithms. The maximum ATLL achieved by S-DCP-D is -86.9 which is comparable to the other algorithms. The provided maximum ATLL score for S-DCP matches with that reported in an earlier study in (Upadhya and Sastry, 2017). Also, the ATLL achieved by the learnt models are comparable to that of the VAE (Variational Autoencoder) and IWAE (Importance Weighted Autoencoder) models (Burda et al., 2015). We observe a similar behaviour for the CalTech and *kannada*-MNIST datasets, as shown in Fig. 3 and 4 respectively. The performance of S-DCP-D is superior to that of S-DCP, CG, CD and PCD algorithms.

The samples generated by the models learnt using MNIST dataset are given in Fig. 5. As observed from Fig. 5, the samples generated by S-DCP-D are sharp compared to those produced by CG based model. Also, it can be observed that the samples generated by CG and S-DCP-D are more diverse compared to those produced by S-DCP. We observed a similar behaviour for the CalTech and *kannada*-MNIST dataset. While subjectively the samples produced by S-DCP-D look better, it is important to note that there exist no objective measures to evaluate a generative model based on the quality of the generated samples.

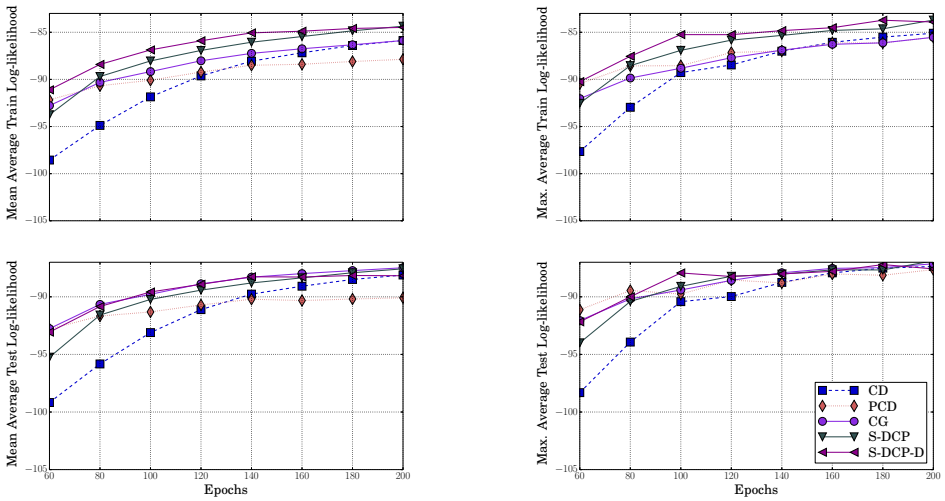


Figure 2: The mean and maximum average log-likelihood over all the trials on the training and test set for MNIST dataset. Note that the learning rate for each algorithm is set to obtain the best performance.

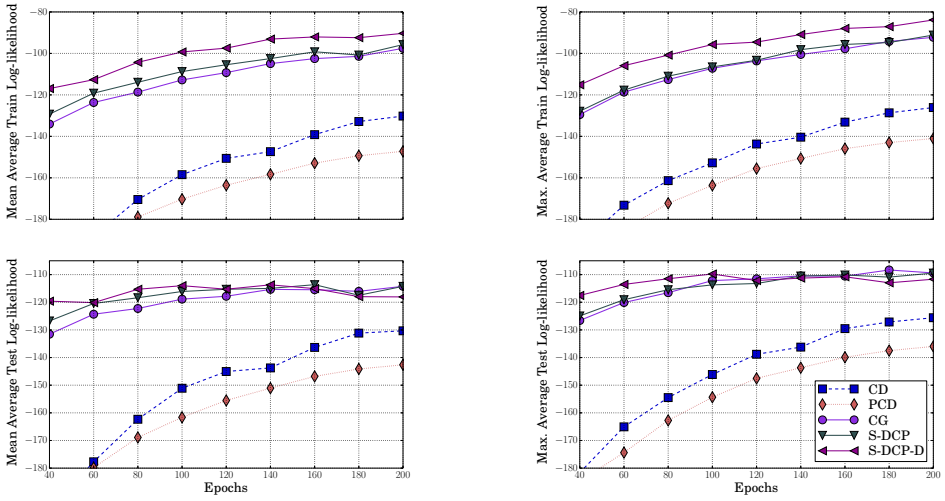


Figure 3: The mean and maximum average log-likelihood over all the trials on the training and test set for CalTech dataset.

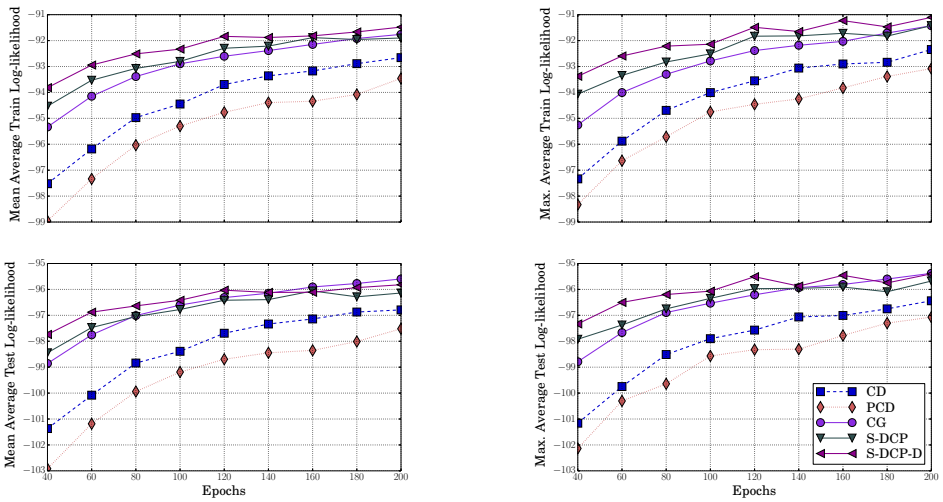
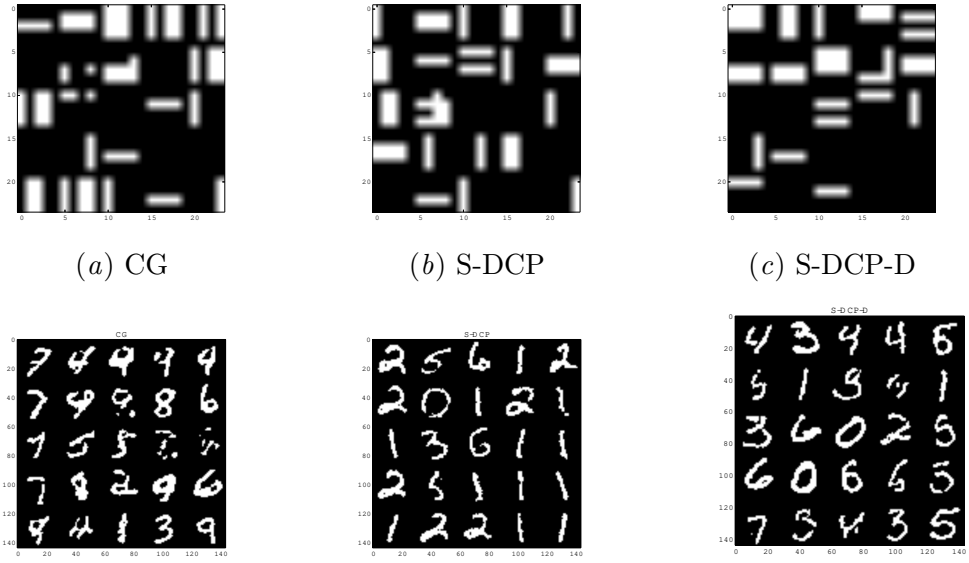


Figure 4: The mean and maximum average log-likelihood over all the trials on the training and test set for *kannada*-MNIST dataset.



(a) CG (b) S-DCP (c) S-DCP-D

(d) CG (e) S-DCP (f) S-DCP-D

Figure 5: 25 sample images generated from the models learnt on the Bars & Stripes (first row) and MNIST dataset (second row). The visible states are randomly initialized and the Gibbs sampler is run for 200 steps (for Bars & Stripes) and 5000 steps (for MNIST). The final states of the visible units are shown.

5. Conclusions

Learning an RBM is difficult due to the noisy estimates of the gradient of the log-likelihood obtained through an MCMC procedure. In this work we proposed an algorithm where we can automatically obtain different adaptive step-sizes for gradient descent for different parameters. This is done by using the inverse of the diagonal approximation of the Hessian. We showed that the Hessian of the log likelihood is given by covariances of the model distribution and hence the Hessian can be estimated using the same MCMC samples that are used for estimating the gradient. Thus, estimating the diagonal of the Hessian has only small additional computational cost.

Through extensive simulations, we showed that the S-DCP-D results in a more efficient learning of RBMs compared to S-DCP and CG algorithms. The additional attraction in using the Hessian here is that in S-DCP-D the gradient descent in the inner loop is on a convex objective function. The diagonal scaling also seems to stabilize the learning and the resulting generative model seems to produce better samples as we showed empirically.

It is known that learning of RBMs can be more efficient if the learning rate is reduced with iterations using a heuristically devised schedule. But the schedule has to be fixed through cross validation. The proposed approach automatically provides parameter-specific learning rates which makes the learning procedure both stable and efficient. The only hyper parameters of the proposed algorithm is ϵ which does not affect the learning dynamics much and is there only to control numerical underflows. The main attraction of S-DCP-D, in our opinion, is its simplicity compared to other sophisticated second-order optimization techniques which use computationally intensive algorithms to estimate the inverse of the Hessian.

For learning an RBM, the centered gradient algorithms are shown to be better compared to CD(k) type algorithm. The reason is conjectured to be the similarity among the second order optimization algorithms and centered gradient method. We feel that the well-motivated and simple second-order algorithm proposed, namely S-DCP-D, can provide the correct platform to further explore this view of centered gradient algorithms.

References

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. doi: 10.1007/s10479-004-5022-1.

Yuri Burda, Roger B. Grosse, and Ruslan Salakhutdinov. Importance weighted autoencoders. *CoRR*, abs/1509.00519, 2015.

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.

Roger B. Grosse and Ruslan Salakhutdinov. Scaling up natural gradient by sparsely factorizing the inverse fisher matrix. In *Proceedings of the 32Nd International Conference on International Conference on Machine Learning - Volume 37*, ICML'15, pages 2304–2313. JMLR.org, 2015.

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.

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.

Benjamin M. Marlin. CalTech 101 Silhouettes Data Set, 2009.

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.

Enrique Romero Merino, Ferran Mazzanti Castrillejo, Jordi Delgado Pin, and David Buchaca Prats. Weighted contrastive divergence. *CoRR*, abs/1801.02567, 2018.

Volodymyr Mnih. Cudamat: a cuda-based matrix class for python. 2009.

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.

Guido Montúfar. Restricted boltzmann machines: Introduction and review. *CoRR*, abs/1806.07066, 2018.

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

Atsushi Nitanda and Taiji Suzuki. Stochastic Difference of Convex Algorithm and its Application to Training Deep Boltzmann Machines. In *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, volume 54 of *Proceedings of Machine Learning Research*, pages 470–478, 20–22 Apr 2017.

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

Vinay Uday Prabhu. Kannada-mnist: A new handwritten digits dataset for the kannada language. *arXiv preprint arXiv:1908.01242*, 2019.

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.

Vidyadhar Upadhya and P. S. Sastry. Learning RBM with a DC programming approach. In *Proceedings of the Ninth Asian Conference on Machine Learning*, volume 77 of *Proceedings of Machine Learning Research*, pages 498–513. PMLR, 15–17 Nov 2017.

Yi Xu, Qi Qi, Qihang Lin, Rong Jin, and Tianbao Yang. Stochastic optimization for DC functions and non-smooth non-convex regularizers with non-asymptotic convergence. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 6942–6951, 09–15 Jun 2019.

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