LCD: A Fast Contrastive Divergence Based Training Algorithm for RBM

Lin Ning LNING@NCSU.EDU
Computer Science Department, NCSU

Xipeng Shen XSHEN5@NCSU.EDU
Computer Science Department, NCSU

Abstract

This paper proposes Lean Contrastive Divergence (LCD), a modified Contrastive Divergence (CD) algorithm to accelerate the training process of Restricted Boltzmann Machine (RBM) while maintaining the same training result achieved by the original CD algorithm. It efficiently recognizes and removes redundant computations from two aspects: one focuses on the sampling process, giving upper and lower bounds of the possible conditional probability of each sampled unit; the other one focuses on the calculation of the conditional probability, reusing history results to speed up the computation. Experiments show that LCD could achieve significant speedups comparing to the original algorithm.

1. Introduction

As a generative model , RBM has been used for extracting meaningful high-level representations (e.g., hidden features in images) from many different types of data input including labeled or unla- beled images (Hinton. et al., 2006; Ranzato et al., 2010), sequence of speech signals (Mohamed & Hinton , 2010), word observations (Dahl et al., 2012) and movie ratings (Salakhutdinov , 2010). It is originally developed as Binary RBM in which both the visible and hidden layers are binary units. Variations (e.g., Gaussian -Bernoulli RBM) are introduced to deal with real value observations and other types of input. RBMs can also be stacked to form deep learning networks like Deep Belief Network and Deep BMs (Hinton . et al., 2006 ; Bengio et al., 2007 ; Salakhutdinov & Hinton , 2009 ; Nair & Hinton , 2009 ; Sri - vastava & Salakhutdinov, 2012).

For many machine learning algorithms, the training is based on gradient descent. However, the loss function of RBM is intractable so it is difficult to use gradient descent directly. Therefore, Contrastive Divergence (CD) learning procedure (Hinton, 2002; 2010), an approximation to gradient descent, is used to train RBM. It creates a Markov Chain for each data point and runs Gibbs Sampling for several iterations to get a low variance estimation of the equilibrium states distribution under the RBM model. CD-k represents a CD algorithm with k Gibbs Sampling iterations for each data point. It is shown that CD-10 always performs better than CD-1 (Tieleman, 2008) but consumes much more time due to more Gibbs Sampling steps. Persistent Contrastive Divergence (PCD) (Tieleman, 2008) is introduced as a faster and simple alternative of the original CD algorithm. Other Efforts (Tieleman & Hinton, 2009; Nair & Hinton, 2010; Cho et al., 2011; Tang & Sutskever, 2011; Courville et al., 2011; Tran et al., 2013; Yamashita et al., 2014; Wang et al., 2014) have also been taken to find efficient and better ways to train the model.

In this paper, we present Lean Contrastive Divergence (LCD), an enhanced training algorithm that accelerates the training process from a different perspective. Instead of introducing new approximations as previous studies did, LCD recognizes and removes unnecessary computations existing in the original algorithm. It could speed up not only the original CD but also those CD based algorithms developed in previous studies. Taking CD as an example, LCD optimizes it without changing the training results from two aspects. First, by calculating upper and lower bounds of vector dot product result, it saves some dot product calculations in the Gibbs Sampling process. Second, it reuses conditional probabilities calculated from the previous iteration and only computes the changes contributed by vector units that are different between two consecutive iterations. LCD combines these two approaches and uses either one of them or the combined version adaptively for each training epoch. This paper uses Binary RBM trained with CD as an example to illustrate the main idea of the op-

timization. Section 3.4 shows that LCD could be applied to variations of RBM, RBM based Deep Learning Networks and other CD based training algorithms with only small modifications. We test the performance of LCD on Binary RBMs and Gaussian-Bernoulli RBMs. We also apply our optimization to PCD training algorithm (LPCD) and use it to train Binary RBMs. The experiment results are included in section 4. In most of the cases, LCD achieves substantial speedups over the original CD based training algorithms.

In the rest of this paper, we first give a brief introduction of the RBM training algorithm, emphasizing the CD algorithm and the Gibbs Sampling process, in section 2. In section 3, we describe the optimization approaches used in LCD and its applicability to variations of RBM, Deep Neural Networks and other Training Algorithms. Section 4 presents setups and results of the experiments. Finally, we summarize the work in section 5.

2. RBM Training Algorithm

2.1. RBM with Binary Units

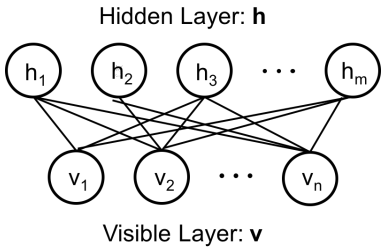

Figure 1. A Binary RBM with n visible units and m hidden units. $\mathbf{h}$ represents the hidden unit vector and $\mathbf{v}$ represents the visible unit vector.

As illustrated in Figure 1, Binary RBM, the simplest RBM, is composed of two layers of binary units: a visible layer ($\mathbf{v}$) with n visible units and a hidden layer ($\mathbf{h}$) with m hidden units. These two layers are denoted by a visible unit vector $\mathbf{v}$ and a hidden unit vector $\mathbf{h}$. The state of each unit can only be 0 or 1. $\theta = (\mathbf{a}, \mathbf{b}, W)$ are the model parameters: the bias vectors $\mathbf{a} \in \mathbb{R}^n$ and $\mathbf{b} \in \mathbb{R}^m$ for the visible and hidden layers and the weight matrix $W \in \mathbb{R}^{n \times m}$ that contains the weights on edges between pairwise visible-hidden units. The conditional distributions are:

$$P(h_j = 1|\mathbf{v}) = \sigma(b_j + \mathbf{v}^T W_{(:,j)}) \tag{1}$$

$$P(v_i = 1|\mathbf{h}) = \sigma(a_i + W_{(i,:)}\mathbf{h}) \tag{2}$$

where $\sigma(\cdot)$ is the sigmoid activation function. Previous studies (Hinton, 2010; Tieleman & Hinton, 2009) showed that the updating rules for θ during the training process with a learning rate ϵ are the following:

$$\Delta W_{ij} = \epsilon(\langle v_i h_j \rangle_d - \langle v_i h_j \rangle_m) \tag{3}$$

$$\Delta a_i = \epsilon(\langle v_i \rangle_d - \langle v_i \rangle_m); \quad \Delta b_j = \epsilon(\langle h_j \rangle_d - \langle h_j \rangle_m) \tag{4}$$

where $\langle \cdot \rangle_d$ and $\langle \cdot \rangle_m$ are the expectations under the distribution specified by the training input data and the theoretical RBM model. Although computing $\langle v_i h_j \rangle_d$ is straightforward, $\langle v_i h_j \rangle_m$ is intractable due to the large number of possible joint $(\mathbf{v}, \mathbf{h})$ configurations.

2.2. Contrastive Divergence and Gibbs Sampling Process

Contrastive Divergence (CD) algorithm (Hinton, 2002) is a learning procedure being used to approximate $\langle v_i h_j \rangle_m$. For every input, it starts a Markov Chain by assigning an input vector to the states of the visible units and performs a small number of full Gibbs Sampling steps. Resulting reconstructed visible units are used to approximate the expectation of the model distribution (Hinton, 2010; Bengio, 2009). The detailed algorithm for processing a whole dataset is shown in Alg. 1.

The main part of CD is the Gibbs Sampling process. As shown in Alg. 1 lines 11 - 18, a full Gibbs Sampling step includes sampling visible units given the hidden layer and then sampling the hidden units given the reconstructed visible layer. Based on how many full Gibbs Sampling steps are performed, the algorithm is named as CD-k with k standing for the number of steps.

Tieleman (2008) observed that CD-10 outperformed CD-1 for almost all tested cases. Although it takes longer for CD-10 to finish a same number of epochs, CD-10 always achieves a larger log-likelihood with a same time of training. After a close study of CD, we find some redundant computations in the Gibbs Sampling process. By eliminating those computations, there are opportunities to speed up the training algorithm. CD-k with $k > 1$ then costs much less and is preferable due to better learning result.

3. Optimizing the CD Algorithm

The main computation of the original CD algorithm comes from the k-steps Gibbs Sampling when $k > 1$. Therefore, optimizing this part could largely reduce the execution time. We utilize two different approaches to remove redundant computations in this part.

The first one is Bound optimization. As shown in lines 12-13 and lines 16-17 of Alg. 1, to sample a visible or hidden unit, CD calculates the probability of turning on the unit and set it to 1 if the probability is greater than a generated random number. The computation of the conditional probability includes a dot product of two vectors ($W_{(i,:)}\mathbf{h}$

Algorithm 1 original CD-RBM

1: **Input:** input dataset, number of inputs N, batch size N_b, number of training epochs N_e, number of gibbs sampling steps k, input vector dimension n, size of hidden layer m
2: **for** $e = 1$ **to** N_e **do**
3: **for** $batch = 1$ **to** N/N_b **do**
4: **for** $q = 1$ **to** N_b **do**
5: Let the q^{th} input be vector $\mathbf{v}_d$
6: **for** $j = 1$ **to** m **do** for positive term
7: $P_+(h_j = 1|\mathbf{v}_d) = \sigma(b_j + \mathbf{v}_d^T W_{(:,j)})$
8: $h_j \sim P_+(h_j = 1|\mathbf{v}_d)$
9: **end for**
10: **for** $step = 1$ **to** k **do**
11: **for** $i = 1$ **to** n **do** for negative term
12:
13: $v_i \sim P(v_i = 1|\mathbf{h}) = \sigma(a_i + W_{(i,:)}\mathbf{h})$
14: **end for**
15: **for** $j = 1$ **to** m **do**
16: $P(h_j = 1|\mathbf{v}) = \sigma(b_j + \mathbf{v}^T W_{(:,j)})$
17: $h_j \sim P(h_j = 1|\mathbf{v})$
18: **end for**
19: **end for**
20: **for** $i = 1$ **to** n **do**
21: **for** $j = 1$ **to** m **do**
22: $< v_i h_j >_d = v_{d,i} \cdot P_+(h_j = 1|\mathbf{v}_d)$
23: $< v_i h_j >_m = v_i \cdot P(h_j = 1|\mathbf{v})$
24: $\Delta W_{ij} \mathrel{+}= \epsilon(\langle v_i h_j \rangle_d - \langle v_i h_j \rangle_m)$
25: $\Delta a_i \mathrel{+}= \epsilon(\langle v_i \rangle_d - \langle v_i \rangle_m)$
26: $\Delta b_j \mathrel{+}= \epsilon(\langle h_j \rangle_d - \langle h_j \rangle_m)$
27: **end for**
28: **end for**
29: **end for**
30: update parameters $\theta = (\mathbf{a}, \mathbf{b}, W)$
31: **end for**
32: **end for**

in line 12 or $\mathbf{v}^T W_{(:,j)}$ in line 16), which is time consuming. Also, the memory access is expensive here because the Gibbs Sampling process goes though the whole W matrix in order to sample a hidden or visible layer once. The main idea of Bound optimization is to give upper and lower bounds of the dot product result instead of calculating the exact value. In this way, we may be able to avoid some of the expensive dot-product computations as well as saving memory access time.

The second one, Partial Dot Production, is inspired by the fact that only a few units change their states between two consecutive Gibbs Sampling of either the visible layer or the hidden layer after training for a while. Therefore, to compute $W_{(i,:)}\mathbf{h}$ or $\mathbf{v}^T W_{(:,j)}$, we could keep track of the changing units and update the result from the previous iteration by adding or subtracting the weights corresponding to the changing units.

If not mentioned otherwise, the remaining of this section uses the sampling process of hidden units as an example.

3.1. Bound Optimization

In the Gibbs Sampling , $h_j \sim P(h_j=1|\mathbf{v})$ is used to sample the j^{th} hidden unit . Since there is a comparison between the conditional probability and a random number , the information we actually need for later computation is the comparison result , not the exact value of the probability . Therefore , it is possible that some of the calculations of the probability are unnecessary . A method of obtaining the comparison result without calculating the probability could then save us a lot of work. The method we propose is using the bounds of $P(h_j=1|\mathbf{v})$ as a filter to avoid some computations . Given Eq . 1, since the sigmoid function is monotonically increasing and b_j is a constant , the lower and upper bounds of the probability $P(h_j=1|\mathbf{v})$ are

$$lb(P(h_j = 1|\mathbf{v})) = \sigma(b_j + lb(\mathbf{v}^T W_{(:,j)})) \quad (5)$$

$$ub(P(h_j = 1|\mathbf{v})) = \sigma(b_j + ub(\mathbf{v}^T W_{(:,j)})) \quad (6)$$

Then if $r > ub(P(h_j = 1|\mathbf{v}))$ or $r < lb(P(h_j = 1|\mathbf{v}))$, we set $h_j = 0$ or $h_j = 1$ accordingly. Only when r falls between $lb(P(h_j = 1|\mathbf{v}))$ and $ub(P(h_j = 1|\mathbf{v}))$, we compute the exact value of the probability using Eq. 1.

To define the bounds of $\mathbf{v}^T W_{(:,j)}$, we explore two different ways in this paper: one simply adds the weights and the other makes use of the triangular inequality.

3.1.1. Bound with Weight Summation (WS)

Let $W^+_{min,j}$, $W^-_{max,j}$ be the minimum positive weight and the maximum negative weight in $W_{:,j}$. $N^+_{W_{:,j}}$ and $N^-_{W_{:,j}}$ are the number of non-negative and non-positive weights in $W_{:,j}$. $N^1_{\mathbf{v}}$ is the number of units with a state of *1* in the visible unit vector $\mathbf{v}$. We use the following formulas to estimate the bounds of the dot product:

$$lb(\mathbf{v}^T W_{(:,j)}) = \sum_i W^-_{ij} + max(0, N^1_{\mathbf{v}} - N^-_{W_{:,j}}) \cdot W^+_{min,j} \quad (7)$$

$$ub(\mathbf{v}^T W_{(:,j)}) = \sum_i W^+_{ij} + max(0, N^1_{\mathbf{v}} - N^+_{W_{:,j}}) \cdot W^-_{max,j} \quad (8)$$

Taking Eq. 7 as an example, we first add the negative weights to get a lower bound. If $N^1_{\mathbf{v}}$ is larger than $N^-_{W_{:,j}}$, it means there are visible units with a state of 1 corresponding to positive weights. In that case, we multiply the difference between $N^1_{\mathbf{v}}$ and $N^-_{W_{:,j}}$ with the minimum positive weight

to get a tighter lower bound. In a similar way, we could calculate the upper bound as shown in Eq. 8.

Overhead Analysis: To get the summations of positive and negative weights in each weight vector $W_{:,j}$, the algorithm need to go through the entire weight matrix once, resulting in a computation overhead of $O(mn)$. It seems to be a large overhead since it is comparable to the computation complexity of a full Gibbs Sampling step. However, the summations could be reused as long as the weight matrix doesn't get updated. Therefore, when training RBM with the CD-k algorithm, we could reuse the calculated bounds across k Gibbs Sampling steps for processing a training input. Moreover, the training inputs set are usually divided into batches with a batch size of N_b (which is usually 100 or more). The weights as well as the biases are updated after processing a batch of inputs. Considering this, we only need to update the summations once per batch and the computation overhead remains $O(mn)$ for processing a whole input batch.

The remaining part of bounds calculation are about finding the maximum and minimum weights in $W_{:,j}$, counting the number of positive and negative weights in $W_{:,j}$ and counting the number of 1s in the visible unit vector $\mathbf{v}$. Information about the weights could be gathered while calculating the summations of positive and negative weights. $N_{\mathbf{v}}^1$ could be counted by adding a counter when sampling the visible units (lines 11-14 in Alg. 1). Overall, the overhead brought by WS is negligible.

3.1.2. Bound with Triangular Inequality (TI)

Another way to calculate the bounds of $\mathbf{v}^T W_{(:,j)}$ can be obtained by representing the vector dot product as

$$\mathbf{v}^T W_{:,j} = \frac{1}{2}(\|\mathbf{v}\|^2 + \|W_{:,j}\|^2 - d_j^2) \tag{9}$$

with $d_j = \|\mathbf{v} - W_{:,j}\|$. As illustrated in a 2d space in Fig. 2, given $\mathbf{v}$, $W_{:,1}$, $W_{:,j'}$ with $j' \neq 1$ and $\Delta = \|W_{:,j'} - W_{:,1}\|$, we have

$$|d_1 - \Delta| \leq d_{j'} \leq d_1 + \Delta \tag{10}$$

Combining Eq. 9 and Eq. 10, we calculate the bounds as

$$lb(\mathbf{v}^T W_{(:,j)}) = \frac{1}{2}(\|\mathbf{v}\|^2 + \|W_{:,j'}\|^2 - ub(d_{j'})^2) \tag{11}$$

$$ub(\mathbf{v}^T W_{(:,j)}) = \frac{1}{2}(\|\mathbf{v}\|^2 + \|W_{:,j'}\|^2 - lb(d_{j'})^2) \tag{12}$$

An advantage of this method comparing to the previous one is that it could be applied directly to Gaussian-Bernoulli RBM which takes real number inputs.

Overhead Analysis: Calculating the bounds in this way introducing more overheads since we need to maintain

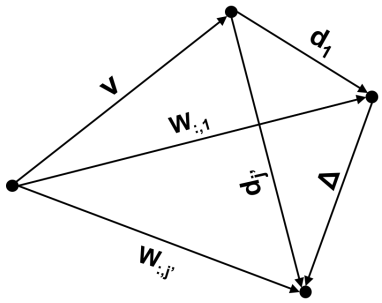

Figure 2. A 2d illustration of vector representations of visible vector $\mathbf{v}$, $W_{:,1}$, $W_{:,j'}$, $\mathbf{d_1}$, $\mathbf{d_{j'}}$ and$\mathbf{\Delta}$

$\|W_{:,j}\|$, Δ and $\|\mathbf{v}\|$. We use $\Delta = \|W_{:,j'} - W_{:,1}\|$ all the time in order to simplify the algorithm and to reduce the overhead. The computation complexity of $\|W_{:,j}\|$ and Δ are both $O(mn)$. However, we only need to update them after processing a batch of inputs. If the batch size if large, this overhead is relatively small. Similarly, we only need to calculate $\|\mathbf{v}\|$ once when sampling the hidden unit vector in one Gibbs Sampling process. Then the overhead $O(n)$ is small compared to $O(mn)$ when m is large.

3.1.3. Benefit Brought by Bound Optimization

Using the sampling process of hidden units as an example, the original CD goes through the entire weight matrix in order to sample the whole hidden unit vector. The computations required for sampling each hidden unit is $O(n)$ where n is the number of visible units. Therefore, the computation complexity of calculating the conditional probability in each Gibbs sampling step is $O(mn)$ with m denoting the number of hidden units. The total computations needed for the k-steps Gibbs Sampling is $O(kmn)$.

In the Bound optimization, we first compute the bounds. If the randomly generated number falls out of the bounded range, the computations needed to generate a sample is $O(1)$ without counting the overhead. Otherwise, it is $O(n)$, which is the same as the one needed by the original CD. The overall speedup we could achieve depends on the amount of the removed computations and the overhead introduced by calculating the bounds of $\mathbf{v}^T W_{(:,j)}$.

Let f be the fraction of hidden units sampling calculations on which our bounds filter works successfully. Then the computation complexity of our optimization for processing a batch of input comes from three main parts: the bounds maintenance ($O(mn)$ for WS based approach or $O(mn + N_b kn)$ for TI based approach), the probability calculation when the bounds filter works successfully ($O(fN_b km)$ for both of the approaches) and the probability calculation when the bound filter fails ($O((1-f)N_b kmn)$

for both of the approaches). Comparing to the original CD-k algorithm which requires $O(N_b kmn)$ computations, our method is more efficient than the original algorithm when f is large.

Experiments show that the amount of computations removed by the TI based approach is similar to the amount achieved by the WS based approach. Since the WS based approach has a smaller overhead, we use it to calculate the bounds in all the experiments other than those with Gaussian-Bernoulli RBM. For Gaussian-Bernoulli RBM, since TI based approach could be used without modification, we implement it to calculate the bounds.

3.2. Partial Dot Production (PDP)

In either the visible layer or the hidden layer, we observed in the experiments that differences only exist in a few units between any two consecutive Gibbs Sampling steps, which means only a few units flip from 0 to 1 or from 1 to 0, after training RBM for a while. Fig. 4 illustrates how the fraction of flipping units changes across epochs for both the hidden layer and the visible layer with a learning rate of 0.01 on the MNIST dataset. The fraction is not very large at the beginning and drops dramatically at later epochs. We will discuss this experiment result more in section 4. Based on this result, we could assume that after training the RBM for a while, a large fraction of visible units and hidden units has a high probability to stay in the same state in two successive Gibbs Sampling steps for a training input under most situations.

Utilizing this feature could help us remove a lot of redundant computations. We still use the sampling of hidden unit vector as an example here. Let $\mathbf{v}^q$ be the visible unit vector used to sample the hidden unit vector $\mathbf{h}^{q+1}$ during the $(q+1)^{th}$ Gibbs Sampling step. Use c_j^q to represent the input of the sigmoid function such that

$$c_j^q = b_j + (\mathbf{v}^q)^T W_{(:,j)} \tag{13}$$

Hence the formula for calculating the conditional probability of turning on the j^{th} hidden unit becomes $P(h_j^{q+1} = 1|\mathbf{v^q}) = \sigma(c_j^q)$. Let $S_{0\to 1}$ and $S_{1\to 0}$ be the sets of visible units which change their states ($0 \to 1$ and $1 \to 0$) during the sampling of $\mathbf{v}^{q+1}$. For example, if $\mathbf{v}^q = \{0,1,1,0\}$ and $\mathbf{v}^{q+1} = \{0,0,1,1\}$, we construct the sets as $S_{0\to 1} = \{v_4\}$ and $S_{1\to 0} = \{v_2\}$. In the $(q+2)^{th}$ Gibbs Sampling step, when calculating the probability using $P(h_j^{q+2} = 1|\mathbf{v^{q+1}}) = \sigma(c_j^{q+1})$, instead of computing $c_j^{q+1} = b_j + (\mathbf{v}^{q+1})^T W_{(:,j)}$, we calculate

$$c_j^{q+1} = c_j^q + \sum_{v_t \in S_{0\to 1}} v_t^{q+1} w_{tj} - \sum_{v_t \in S_{1\to 0}} w_{tj} \tag{14}$$

In this way, we reuse the result computed from the previous iteration and only calculate the partial dot product $\sum_{v_t \in S_{0\to 1}} v_t^{q+1} w_{tj}$ instead of the full dot product $\sum_i v_i^{q+1} w_{ij}$. This optimization could speed up both the training and the predicting processes.

Benefit and Overhead Analysis: The overhead of PDP mainly comes from the maintain of the two sets $S_{0\to 1}$ and $S_{1\to 0}$, which is $O(n)$ for sampling the hidden unit vector. Let f_v be the fraction of flipping units in the visible unit vector. The computation complexity of using PDP for a Gibbs Sampling step is $O(2f_v mn)$. When f_v is small, PDP could bring significant speedup over the original CD.

3.3. Combing the Two Approaches

We observe that the Bound optimization works better at the beginning of the training while PDP works well in the later stage. Therefore, it could be a good idea to combine these two approaches together to achieve an even better performance for the whole training process.

There are two major factors affect the design of the combined version. First, PDP may be able to remove a lot of computations when the Bound optimization fails in the early stage of training. Second, if we apply the Bound optimization for every unit and change to PDP if the bounds filter fails, the choice of the current iteration will affect the computations needed in the next iteration if PDP is chosen in the next iteration. As shown in Eq. 14, we need to know c_j^q in order to calculate c_j^{q+1} using PDP. However, if the Bound optimization works in the current iteration, the unit h_j is set to 0 or 1 directly without calculating the exact value of c_j^q. Then in the next iteration, if the Bound optimization fails, we cannot benefit from PDP because we don't know c_j^q.

Taking these two factors into consideration, we design the Combined version in a way such that each of them has a chance to be chosen for each Gibbs Sampling step during the entire training process. For each Gibbs Sampling step, we compare the expected computations ($\langle comp \rangle$) required by both of the approaches and choose the one with a small $\langle comp \rangle$. If choosing the Bound optimization, for the units on which the Bound optimization fails, we apply PDP if c_j^q is calculated in the previous iteration.

During the training, we choose among the Bound optimization, PDP and the combined version adaptively. At the beginning, we start the training with the combined version. Then before the starting of each epoch, based on the statistics collected in the previous epoch, we calculate the overheads and benefits brought by each of the three choices and use the one with largest net benefit for the coming epoch.

3.4. Application to Variations of RBM, Deep Neural Networks and Other Training Algorithms

Binary RBM is a special class of RBM. Gaussian-Bernoulli RBM and other kinds of RBMs are proposed to learn from non-binary data images or other types of input. These variations of RBM could also benefit from our optimization. Taking Gaussian-Bernoulli RBM as an example, the visible units are real values while the hidden units are binaries. Therefore, the Bound optimization could be used for sampling the hidden units while PDP could be used for sampling the visible units. For deep networks such as Deep Belief Network and Deep Boltzmann Machine, the first layer could be either a binary RBM or a Gaussian-Bernoulli RBM, and the stacked following layers are binary RBMs, so our optimization could also be applied to them directly.

As mentioned in section 1, some algorithms introduce approximations to the original CD in order to optimizing the RBM training process. As long as these algorithms have the Gibbs Sampling process, or perform the calculation of the conditional probability of turning on a unit in the same way, our optimization could be used. For example, PCD is an approximation of the CD algorithm. Instead of starting a Markov Chain for every input, it starts a chain for every input in a batch. The total number of the Markov Chain is equal to the batch size. Then when a batch of new inputs come into the model, the Markov Chains start at the points where they stop and continue the sampling process. This method is shown to have better performance than CD-1, even CD-10 in most of the experiments (Tieleman, 2008). Our optimization could be applied directly to PCD since it has no difference from CD considering the Gibbs Sampling steps.

4. Experiments

We evaluate the efficiency of our optimization on both of Binary RBM and Gaussian-Bernoulli RBM over a variety of real world datasets. For Binary RBM, we study the performance of LCD and LPCD over six datasets: the MNIST handwritten digits dataset (LeCun et al., 1998), the CalTech 101 Silhouettes dataset (Marlin et al., 2010), the MICRO-NORB dataset (Tieleman & Hinton, 2009), the small 20-Newsgroup dataset (Marlin et al., 2010) and the UCI repository Abalone dataset (Bengio et al., 2007). Experiments are also preformed on the transformed MNIST (f-MNIST) dataset in which each pixel flips its value (Cho et al., 2011). For each of the datasets, the RBM is trained using different versions of CD-k and PCD-k algorithm with different k values (1, 5 and 10). For the Gaussian-Bernoulli RBM, we compare the performance of LCD over the original CD with three different k values. k value starts from 2. For $k = 1$ case, our optimization cannot be used because of the real value visible units. The datasets used for this part are MNIST, f-MNIST, Abalone, a Financial dataset (Bengio et al., 2007), face image dataset CBCL (Yamashita et al., 2014) and Olivetti face dataset (Cho et al., 2013).

The learning rates are fixed during the training process and different learning rates are studied in the experiments. The number of visible units of RBM is determined by the size of the input image. Previous studies have shown that RBM is able to achieve a good training result on these datasets. Most of the configurations (e.g., # of visible and hidden units, k) we used in our experiments are adopted from these studies. Since our optimization doesn't change the training result of the original CD and calculating the training data log-likelihood is expensive, we use a fixed number of epochs as the termination criteria. To compare the optimized algorithm with the original one, we run both algorithms for the same number of epochs and compare the execution time. The size of the datasets and the configurations of corresponding RBM setups are shown in columns 2-8 of Table 1 and Table 2. N is the size of the dataset. n and m are the number of visible and hidden units. lr is the learning rate and k is the number of Gibbs Sampling steps.

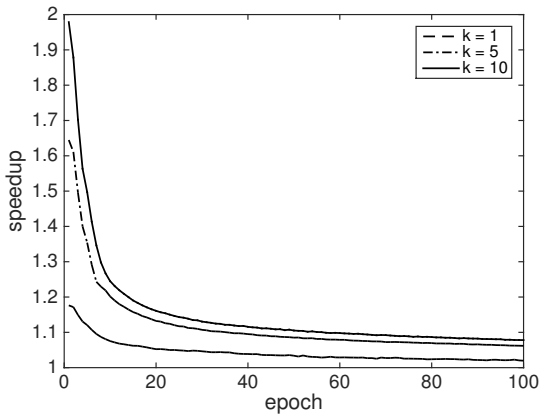

Figure 3. Speedups gained by applying the Bound optimization to the original CD-k algorithm over the MNIST dataset. The speedup is measured for every epoch and plotted as a function of epoch in this graph.

4.1. Performance of the Bound Optimization Across Epochs

Figure 3 illustrates how the speedups obtained by the Bound optimization change with training epochs. The results of training the MNIST dataset using CD-1, CD-5 and CD-10 with a learning rate of 0.01 are shown as an example here. This experiment takes 10000 training inputs and runs for 100 epochs. The speedup is significant at the first several epochs and diminishes very fast. It is because the weights are initialized with small random values

Table 1. Overall Speedups for Binary RBM with CD and PCD algorithm

dataset	N	n	m	No. Epoch	Batch Size	lr	k	overall speedup				
								LCD				LPCD
								Bound	PDP	Comb	Adapt	pdp
Abalone	2100	8	25	500	100	0.01	1	0.87	0.83	0.74	1.04	1.10
							5	0.67	1.06	0.57	1.12	1.27
							10	0.63	1.12	0.54	1.15	1.31
20Newsgroup	8500	100	500	100	100	0.01	1	1.13	1.19	1.34	1.45	1.21
							5	1.60	1.87	1.98	1.92	1.83
							10	1.73	2.14	2.17	2.16	2.11
MNIST	50000	784	500	20	100	0.01	1	1.08	1.22	1.28	1.34	1.17
							5	1.16	2.36	2.35	2.49	2.29
							10	1.15	2.72	2.55	3.29	3.02
f-MNIST	50000	784	500	20	100	0.01	1	1.24	1.17	1.51	1.22	1.15
							5	2.01	2.59	2.51	2.71	2.75
							10	2.26	3.45	3.09	3.47	4.01
CAL101	4100	784	500	300	100	0.01	1	0.98	1.22	1.19	1.22	1.08
							5	1.01	2.51	2.29	2.47	2.58
							10	0.96	3.04	2.73	3.56	3.54
MICRO-NORB	15000	1024	500	100	100	0.01	1	0.99	1.14	1.17	1.22	1.21
							5	0.98	2.33	2.25	2.34	2.34
							10	1.01	2.82	2.49	3.08	2.61

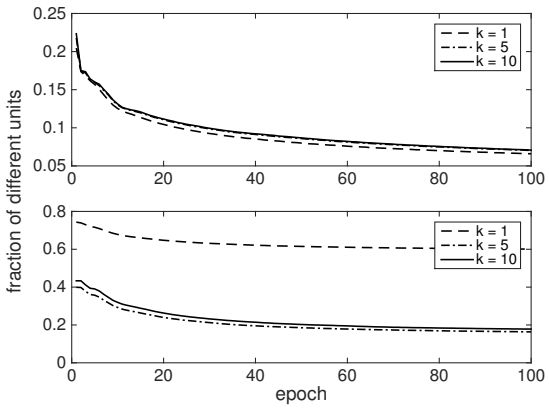

Figure 4. The fraction of flipping units in both the visible and the hidden unit vectors. The upper figure corresponds to the visible unit vector while the lower figure represents the hidden unit vector. The MNIST dataset is used to get these results.

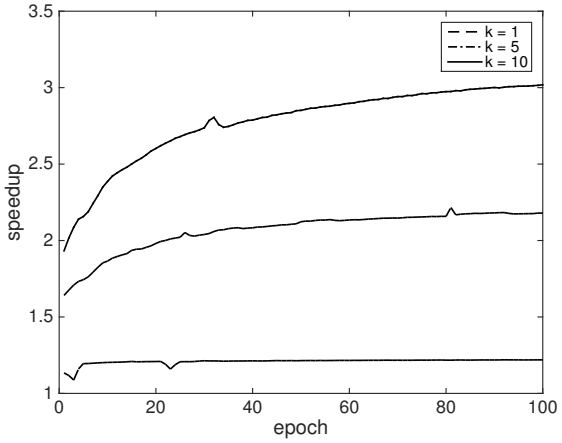

Figure 5. Speedups gained by applying the partial-dot-product optimization (PDP) to the original CD-k algorithm over the MNIST dataset. The speedup is measured for every epoch and plotted as a function of epoch.

at the beginning of the training. So the bounds calculated by summing the positive and negative weights are tight and able to remove many computations. However, as the model gets better trained, the weights get larger and more discrete. The summations of positive and negative weights also get larger. Due to the nature of the sigmoid function (the "S" shape), the resulting bounds become looser quickly and the efficiency of the bounds filter decreases fast.

The results also show that a larger k value results in a larger speedup. This is because for CD, we need to compute the exact conditional probability of turning on each hidden unit in the first Gibbs sampling step and the last Gibbs Sampling step. They are used on lines 22 and 23 in Alg. 1 for weights update. Only the sampling steps in between could get saved by the Bound optimization. As a result, the more Gibbs Sampling steps the training algorithm has for processing a single input, the more benefit it could get from the bound optimization. According to previous works (Tieleman, 2008; Hinton, 2010; LISAlab, 2016), k could be as large as 10 or 15 and a larger k value typically leads to a

Table 2. Overall Speedups for Gaussian-Bernoulli RBM with CD algorithm

dataset	N	n	m	No. Epoch	Batch Size	lr	k	overall speedup		
								PDP	Comb	Adapt
Abalone	2100	8	25	500	100	0.0005	2	1.04	1.00	1.04
							5	1.08	0.97	1.07
							10	1.08	0.97	1.09
Financial	3100	13	35	500	100	0.0005	2	1.04	1.01	1.05
							5	1.11	1.02	1.10
							10	1.11	1.00	1.11
MNIST	50000	784	500	20	100	0.0005	2	1.00	0.99	1.19
							5	1.44	1.41	1.27
							10	1.56	1.41	1.53
f-MNIST	50000	784	500	20	100	0.0005	2	1.03	0.96	1.18
							5	1.44	1.49	1.60
							10	1.59	1.67	1.53
CBCL	2000	381	400	500	100	0.0005	2	1.16	1.18	1.19
							5	1.46	1.42	1.46
							10	1.48	1.64	1.58
Olivetti	384	4096	500	300	64	0.0005	1	1.27	1.20	1.28
							5	1.59	1.75	1.76
							10	1.97	2.05	2.46

better learning result even with a same training time. This leaves some substantial room for the Bound optimizations to take effect.

4.2. Performance of PDP Across Epochs

Figures 4 and 5 show the performance of PDP on the MNIST dataset. If a unit changes its state between two consecutive sampling steps, we call it a flipping unit. We can tell from Figure 4 that the number of flipping visible and hidden units decreases during the training process, as stated in section 3.2. For the visible unit vector, the fraction of flipping units is around 20% at the beginning of the training and drops to under 10% for all three k values. For the hidden unit vector, we mark every hidden unit from unknown to 0 or 1 in the first Gibbs Sampling step. So the fraction of flipping units of the hidden unit vector is larger than that of the visible unit vector in a same epoch. When k=1, we only sample the hidden unit vector twice. Therefore at least half of the hidden units are considered to be different from that of the previous iteration. As k increases, the first Gibbs Sampling step weights less, so the fraction of flipping units is much smaller than the k=1 case.

The speedups achieved by only applying PDP is shown in Figure 5. When k=5 or k=10, the fraction of flipping units in both of the visible and the hidden unit vectors are less than 20% at around epoch 100. However, the maximum speedup achieved in our experiments is not 5. This could be explained by the memory access latency. Even though the computations are removed due to identical unit values, the data may still be loaded into the cache from memory if other data in the same block are used for calculation. This could result in the speedup not proportional to the percentage of computation being saved when applying PDP.

4.3. Overall Speedups

The overall speedups are given in Table 1 and Table 2. The rightmost 5 columns of Table 1 report the speedups of our optimized algorithms over the original ones on binary RBM with CD and PCD and a learning rate of 0.01. The performance of Bound, PDP, combined version and adaptive version are shown for LCD. For LPCD, only the speedups of adaptive version are included. According to the results, our optimization works better on dataset with a larger input vector dimension. Also, a larger k value results in a larger speedup. When k=1, the overall speedups over the original algorithm are relatively small comparing to the k=5 or 10 cases. It indicates that with only one Gibbs Sampling step, there are only a few opportunities existing for either the bound optimization or PDP. When k increases to 5 or 10, the improvement of performance given by bound optimization itself is limited. Meanwhile, PDP brings a significant speedup. The adaptive version always achieves a speedup of larger or comparable to the other three version (except for the k=1 case of f-MNIST). The speedups achieved with a learning rate of 0.1 and 0.001 are similar.

For the Gaussian-Bernoulli RBM, we study the performance of PDP, the combined verision and the adaptive version. It is clear that the speedup achieved by our method is larger when the input vector dimension is large. Also, the adaptive version works better than the other two versions for most of the cases. Only the result of learning rate 0.0005 are shown here. The training process with a learning rate of 0.001 and 0.0001 are also studied and the speedups are similar.

Overall, the experiments demonstrate that our optimizations could efficiently remove a lot of redundant computations and bring substantial speedups with respect to the original CD-k and PCD-k algorithms.

5. Conclusion

This work studies the training algorithm of RBM and proposes LCD as an accelerated CD algorithm by recognizing and removing redundant computations. Choosing adaptively among the Bound optimization, PDP and the combined version, LCD efficiently cut up the RBM training time by two thirds (10 sampling steps per round) without affecting the training results. It demonstrates the promise for accelerating RBM-based deep networks.

Acknowledgement

This material is based upon work supported by DOE Early Career Award and the National Science Foundation (NSF) under Grant No. 1320796, 1525609 and CAREER Award. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of DOE or NSF.

References

Bengio, Y. *Learning Deep Architectures for AI*. Now Publishers Inc, 2009.

Bengio, Y., Lamblin, P., Popovici, D., and Larochelle, H. Greedy layer-wise training of deep networks. In *Advances in Neural Information Processing Systems 19*, pp. 153–160. MIT Press, 2007.

Cho, K., Raiko, T., and Ilin, A. Enhanced gradient and adaptive learning rate for training restricted boltzmann machines. In *Proceedings of the 28th International Conference Proceedings of the 28 th International Conference on Machine Learning*, Bellevue, WA, USA, 2011.

Cho, K., Raiko, T., and Ilin., A. Gaussian-bernoulli deep boltzmann machine. In *Proceedings of the IEEE International Joint Conference on Neural Networks (IJCNN 2013)*, pp. 1–7, Dallas, TX, 2013.

Courville, A., Bergstra, J., and Bengio, Y. A spike and slab restricted boltzmann machine. In *Proceedings of the 14th International Conference on Artificial Intelligence and Statistics (AISTATS)*, pp. 233–241, Fort Lauderdale, FL, 2011.

Dahl, G. E., Adams, R. P., and Larochelle, H. Training restricted boltzmann machines on word observations. In *Proceedings of the 29 th International Conference on Machine Learning*, Edinburgh, Scotland, UK, 2012.

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

Hinton, G. A practical guide to training restricted boltzmann machines. Technical report, 2010.

Hinton., G., Osindero, S., and Teh, Y. A fast learning algorithm for deep belief nets. *Neural Comput.*, 18(7): 1527–1554, July 2006.

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, November 1998.

LISAlab. Restricted boltzmann machines (rbm), 2016.

Marlin, B. M., Swersky, K., Chen, B., and Freitas, N. Inductive principles for restricted boltzmann machine learning. In *Proceedings of the 13th International Conference on Artificial Intelligence and Statistics (AISTATS)*, pp. 509–516, Chia Laguna Resort, Sardinia, Italy, 2010.

Mohamed, A. and Hinton, G. Phone recognition using restricted boltzmann machines. In *Acoustics Speech and Signal Processing (ICASSP), 2010 IEEE International Conference on*, pp. 4354–4357, Dallas, TX, 2010. IEEE.

Nair, V. and Hinton, G. 3d object recognition with deep belief nets. In Bengio, Y., Schuurmans, D., Lafferty, J.D., Williams, C.K.I., and Culotta, A. (eds.), *Advances in Neural Information Processing Systems 22*, pp. 1339–1347. Curran Associates, Inc., 2009.

Nair, V. and Hinton, G. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pp. 807–814, Haifa, Israel, 2010.

Ranzato, M., Krizhevsky, A., and Hinton, G. Factored 3-way restricted boltzmann machines for modeling natural images. In *Proceedings of the 13th International Conference on Artificial Intelligence and Statistics (AISTATS)*, pp. 621–628, Chia Laguna Resort, Sardinia, Italy, 2010.

Salakhutdinov, R. Learning deep boltzmann machines using adaptive mcmc. In *Proceedings of the 27th International Conference on Machine Learning*, pp. 943–950, Haifa, Israel, 2010. Omnipress.

Salakhutdinov, R. and Hinton, G. Deep boltzmann machine. In *Proceedings of the 12th International Conference on Artificial Intelligence and Statistics (AISTATS)*, pp. 448–455, Clearwater Beach, FL, 2009.

Srivastava, N. and Salakhutdinov, R. Multimodal learning with deep boltzmann machines. In Pereira, F., Burges, C.J.C., Bottou, L., and Weinberger, K.Q. (eds.), *Advances in Neural Information Processing Systems 25*, pp. 2222–2230. Curran Associates, Inc., 2012.

Tang, Y. and Sutskever, I. Data normalization in learning of restricted boltzmann machines. Technical report, Department of Computer Science, University of Toronto, November 2011.

Tieleman , T. Training RBMs using approximations to the likelihood gradient . In *Proceed - ings of the 25 th International Conference on Machine Learning* , pp. 1064 –1071, New York, NY, USA, 2008. ACM.

Tieleman, T. and Hinton, G. Using fast weights to improve persistent contrastive divergence. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pp. 1033–1040, New York, NY, USA, 2009. ACM.

Tran, T., Phung, D., and Venkatesh, S. Thurstonian boltzmann machines: Learning from multiple inequalities. In *Proceedings of the 30 th International Conference on Machine Learning*, Atlanta, Georgia, USA, 2013.

Wang, S., Frostig, R., Liang, P., and Manning, C. D. Relaxations for inference in restricted boltzmann machines. In *International Conference on Learning Representations*, Banff, Canada, 2014.

Yamashita, T., Tanaka, M., Yoshida, E., Yamauchi, Y., and Fujiyoshii, H. To be bernoulli or to be gaussian, for a restricted boltzmann machine. In *2014 22nd International Conference on Pattern Recognition (ICPR)*, pp. 1520 – 1525. IEEE, 2014.