Fast Classifier Learning under Bounded Computational Resources using Partitioned Restricted Boltzmann Machines

Hasari Tosun
Department of Computer Science
Montana State University
Email: hasari@gmail.com

John Sheppard
Department of Computer Science
Montana State University
Email: john.sheppard@montana.edu

***Abstract*—We develop a Partitioned Restricted Boltzmann Machine (PRBM) for classification. We demonstrate that this method provides both speed and accuracy. Specifically, because it is partitioned into smaller RBMs, all available data can be used for training, and individual RBMs can be trained in parallel. Moreover, as the number of dimensions increases, the number of partitions can be increased to significantly reduce runtime computational resource requirements. All other recently developed methods using RBMs for classification suffer from some serious disadvantage under bounded computational resources; one is forced to either use a subsample of the whole data, run fewer iterations (early stop criterion), or both. Our Partitioned-RBM method provides an innovative scheme to overcome this shortcoming.**

I. Introduction

The Boltzmann Machine (BM) was initially designed as a parallel network for constraint satisfaction [1]. A Restricted Boltzmann Machine (RBM) is a variant of a Boltzmann machine in which hidden and visible nodes constitute a bipartite graph: there are no probabilistic dependencies among visible nodes or among hidden nodes. Though this restriction allows more efficient computation still RBMs were not practical for a long time. Increased computational power and the development of an efficient training technique Contrastive Divergence (CD), based on Gibbs Sampling [2], have made RBMs applicable to many machine learning tasks. After being proposed as building blocks of Deep Belief Networks (DBNs), they attracted much attention from the machine learning community [3]–[5].

Restricted Boltzmann Machines are used to form a DBN where the network is trained layer-wise while minimizing reconstruction error. This is a form of unsupervised pre-training. Once all layers have been trained, the resultant network can then be used in different ways, including adding a new output layer and running a standard gradient descent algorithm to learn a supervised task such as classification. The aim of stacking RBMs in this way is to learn features in order to obtain a high level representation.

Recent developments in the field of DBNs have led to a renewed interest in applying RBMs as a standalone learning technique in addition to their wide usage as powerful feature extractors. For example, RBMs have successfully been applied to Collaborative Filtering [6]. These authors developed efficient learning and inference procedures using RBMs and successfully applied them to the Netflix data set containing over 100 million user/movie ratings. Because the user-movie matrix has lot of missing entries, the authors designed one RBM per user. However, the RBMs share weights and biases for common movies.

One of the most significant current research achievements with RBMs is to apply them as standalone classifier by Larochelle *et al.* [7]–[9]. In combination with a generative objective function, Larochelle *et al.* developed a discriminative objective function to train RBMs as classifiers. It was demonstrated that when a hybrid objective function is used, the classification accuracy can be increased significantly. However, interestingly, as demonstrated in Section III, the results could not be replicated due to an overflow when computing the gradient of the discriminative objective function. We will discuss this in more detail in Section III.

Schmah *et al.* took a different approach in applying RBMs to classification tasks: instead of using a monolithic RBM to represent all classes, the authors trained one RBM per class label [10]. However, a major drawback of this approach is that it cannot model latent similarities between classes. Nonetheless, a most interesting result of their study is the demonstration that the generative training can improve discriminative performance even if all data are labeled. Studying two methods of training, one almost entirely generative and one discriminative, the authors found that a generatively trained RBM yielded better discriminative performance for one of the two tasks studied.

We previously proposed a novel algorithm, Partitioned-RBM, for training RBMs that splits a single RBM into multiple partitions. Each partition is then trained on a subsection of the data instances. In our experiments, we found that this training process improves the performance in terms of both generative power and speed [11]. After applying spatial statistical analysis tools, we found that traditional RBMs are insensitive to spatially local structure in the input [12], whereas

Partitioned-RBMs are sensitive to local structure. We further evaluated statistical features in the context of DBNs. The experimental results showed that spatial statistics are completely lost when traditional RBMs are used [13]. However, in the case of Partitioned-RBMs, spatial features are preserved in higher layers of the network while maintaining high reconstruction accuracy. This conclusion led us to believe that a Partitioned-RBM may be used as an efficient standalone learning method for classifications tasks as well.

In this paper, we introduce a variant of the Partitioned-RBM for classification tasks. We believe that far too little attention has been paid to runtime performance and far too much attention has been paid to obtaining higher classification accuracy rates or lower classification error rates. To the best of our knowledge, using RBMs with bounded computational resources, namely CPU time, has not been considered previously. Hence, the goal of this paper is to find out if there exists a method for improving the runtime performance of RBMs without significantly degrading classification accuracy. When we developed the Partitioned-RBM to obtain better reconstruction accuracy, we noticed that runtime performance was also superior to regular RBMs [11]. Thus, our previous results suggest that Partitioned-RBMs can perform well under bounded computational resources.

Another key aspect of our Partitioned-RBM is that the training process introduces an intrinsic sparsity by design. At each training stage, the weights only inside individual partitions are optimized; the weights connecting the visible layer of one partition to the hidden layer of another are untouched. Thus, the Partitioned-RBM weights are inherently sparse. This is important because Larochelle *et al.* showed that a sparse version of the hybrid RBM significantly outperforms all other techniques in terms of classification error rates; the results were even better than Deep Belief Network results reported in the literature [8]. Larochelle *et al.* introduced sparsity by subtracting a small constant δ value, a hyper-parameter, from biases after each parameter update. In our case, the training of smaller partitions and gradually piecing them together accomplishes a similar result.

The rest of this paper is organized as follows: In Section II, we briefly introduce the Boltzmann Distribution and describe our training method as a generative model. In Section III, we describe the extension of Partitioned-RBMs for classification and show experimental results in Section IV. Finally, we discuss future work in Section V.

II. Partitioned Restricted Boltzmann Machines

The RBM network was first proposed by Smolensky [14] in 1986. As a type of Hopfield Network, an RBM is a generative model with visible nodes ($\mathbf{x}$) and hidden nodes ($\mathbf{h}$). As a complete bipartite graph, there are no dependencies among hidden nodes, nor among visible nodes. This model can be represented as a Boltzmann energy distribution [15], in which the joint probability distribution is given as follows:

$$p(\mathbf{x}, \mathbf{h}) = \frac{1}{Z} \exp(-E(\mathbf{x}, \mathbf{h}))$$

where the partition function, Z defines configurations over all possible states of $\mathbf{x}$ and $\mathbf{h}$.

$$Z = \sum_{\mathbf{x},\mathbf{h}} \exp(-E(\mathbf{x}, \mathbf{h}))$$

Calculating $p(\mathbf{x}, \mathbf{h})$ is not tractable due to the presence of the partition function. Thus, training algorithms were initially very inefficient. RBMs did not gain popularity for many years until Hinton *et al.* developed Contrastive Divergence (CD), a method based on Gibbs Sampling [2]. Gibbs sampling makes use of the conditional probability, $p(\mathbf{h}|\mathbf{x})$, which has a simple form as presented in Equation 1. Thus, with development of the CD technique, RBMs became widely used as basic components of deep learning algorithms [3]–[5].

$$p(\mathbf{h}|\mathbf{x}) = p(\mathbf{x}, \mathbf{h}) / \sum_{h'} p(\mathbf{x}, \mathbf{h}') \tag{1}$$

The CD method provides a reasonable approximation to the likelihood gradient of the energy function, and the CD-1 algorithm (i.e, Contrastive Divergence with one step) has been shown to be sufficient for many learning applications [16], [17], including classification tasks [7]–[9], [18] and other tasks such as Collaborative Filtering [6].

In order to preserve spatially local features in the dataset and improve the performance of RBMs, Tosun and Sheppard developed a Partitioned-RBM [11]. The Partitioned-RBM partitions a traditional RBM into multiple smaller RBMs, which are trained independently (and, ideally, in parallel). The training involves multiple stages, where each stage has progressively fewer partitions until one all-inclusive partition is left. In other words, partitions at successive stages cover a progressively larger portion of the RBM weights and biases, until the final stage looks just like a traditional RBM, but uses as initial weights the ones learned in the earlier stages.

Each stage of a Partitioned-RBM is trained using a set of sub-vectors partitioned from the training instances; in effect, each "sub-RBM" is trained on instances that contain only the features that correspond to the input nodes assigned to that RBM. Once they have been trained, a new partitioning with fewer splits is created, which forms the basis for the next stage. Figure 1 shows an example where the first stage has 4 distinct RBMs, the second stage has 2, and the final stage has 1.

Performance gains come from all training stages sharing the same weight matrix. As each RBM covers its own part of the globally shared weights and biases, this method enables data-independent parallelization of earlier stages. The later stages, while allowing less parallelization, begin their training with weights that have been pre-trained in the earlier stages. This has several advantages. First, when RBMs have fewer nodes and weights to be updated, they can be trained more quickly; for each Gibbs step, CD-1 involves $O(S \times H \times V)$ probability calculations per iteration where S, H and V are number of samples, hidden nodes, and visible nodes respectively. Thus, we estimate that the total number of Markov chain calculations for a regular RBM is approximately

$$ChainOps \simeq O(I \cdot S \cdot H \cdot V)$$

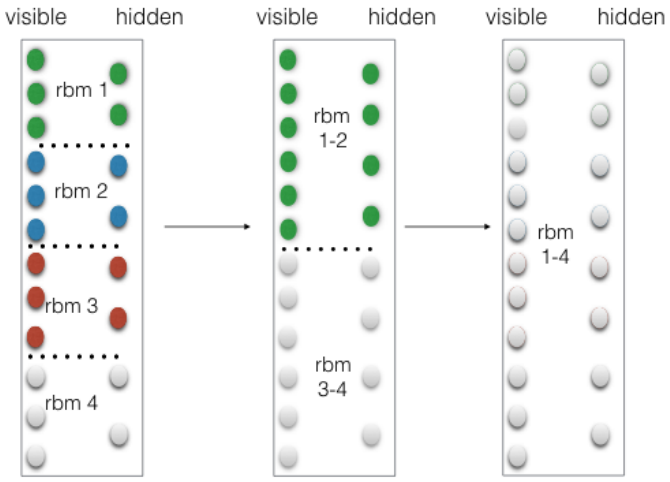

Fig. 1. Example partitioning approach for an RBM

where I is the number of iterations for the complete training. Fewer chain operations translate into less CPU time. Since Partitioned-RBM is split in each stage and run in parallel, the number of samples for a stage can be configured to improve runtime performance. Therefore, we estimate that the total number of Markov chain calculations for a Partitioned-RBM is

$$ChainOps \simeq O\left(I \cdot H \cdot V \cdot \sum_{\substack{n\in\{n_0,n_1,\ldots,1\}\\ s\in\{s_0,s_1,\ldots,\}}} \frac{s_i}{n_i^2}\right) \quad (2)$$

where n_i represents number of splits in stage i and s_i represents number of samples used at each stage for training. If we keep I, H, and V constant, we can vary the number of samples used in each stage and/or the number of partitions to improve runtime performance. We show that such a scheme works without degradation of the classification accuracy of the model.

III. Class Partitioned RBMs

Larochelle *et al.* carried out a comprehensive study of RBMs in classification tasks and successfully applied RBMs as a standalone learning technique, $classRBM$ [18]. In addition to a generative training objective function for the CD algorithm, the authors developed a discriminative training objective function as well. Furthermore, combining two objective functions, they created a hybrid objective function that was then used for the gradient calculations. The authors reported that the hybrid method produced excellent results. However, we could not replicate their results because the discriminative training objective function calculations resulted in an overflow for reasonably large networks. The discriminative objective function involves the following equation

$$p(\mathbf{y}|\mathbf{x}) = \frac{\exp(d_y+\sum_j^N s(c_j+U_{jy}+\sum_i W_{ji}x_i))}{\sum_{y^*\in\{1,\ldots,C\}}\exp(d_y+\sum_j^N s(c_j+U_{jy^*}+\sum_i W_{ji}x_i))}$$

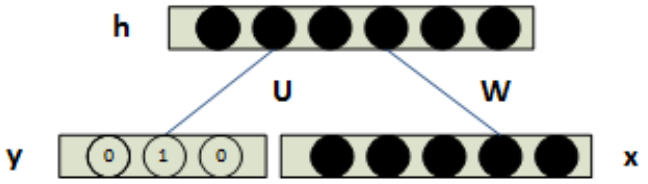

Fig. 2. Class RBM Network

where $\mathbf{c}$ is the weight vector for the bias on the hidden nodes, and $\mathbf{U}$ is the weight vector between hidden nodes and class nodes. Finally, $s(x) = \log(1 + \exp(\mathbf{x}))$ is the $softplus$ function. An example network is shown in Figure 3. Here, $\mathbf{x}$, $\mathbf{h}$, and $\mathbf{y}$ represent visible, hidden, and class label nodes (output nodes), respectively. Class nodes are all set to 0 except the for the node corresponding to the target class label, which is set to 1.0. For instance, when there are 10 classes and a specific data instance has the third label, the corresponding output vector, $\mathbf{y}$, is $[0, 0, 1.0, 0, 0, 0, 0, 0, 0, 0]$.

Unfortunately, this conditional distribution can only be computed exactly and efficiently when there is a reasonably small number of classes. Otherwise, the denominator will result in an exponential number of terms, as we have to sum over all classes. A more serious weakness with this method, however, is that in practice it suffers from computational overflow.

For each y_i value, ignoring $\mathbf{c}$ biases and $\mathbf{U}$, the value in $softplus$ function is a vector of $W_{H,V}x_{V,1}$. If the $\mathbf{W}$ matrix is initialized with values randomly chosen between $\frac{-1.0}{\max\{V,H\}}$ and $\frac{+1.0}{\max\{V,H\}}$ and for $H = 1500$ and $V = 784$, the resulting matrix values will be distributed uniformly between -0.00067 and 0.00067. Thus, the value for $\exp(*)$ in $softplus$ function ranges from 0.9993 to 1.0007. The $\log(1 + *)$ expression results in values around 0.69. Then, ignoring d_y, the $\sum(*)$ expression has value of $H \times 0.69$. Finally, it results in $\exp(H \times 0.69)$. Thus, with $H = 1500$, we have $\exp(1035)$. But for modern CPUs, $\exp(710.0)$ results in overflow (infinity). Therefore, the maximum number of hidden nodes one can use is around $710.0/0.69$, that is 1028 nodes.

It should be noted that we ignored many variables in this formula to simplify the analysis. Thus, even for 1028 nodes, it is not guaranteed that the overflow will not happen. Overflow can be encountered when one uses 1000 nodes or fewer. In addition to the overflow, the time complexity of an RBM increases with addition of the discriminative gradient.

As we could not use $classRBM$ for some reasonably sized experiments, we propose a model where we train partitions as described in Section II in an unsupervised fashion except for the last stage. As shown in Figure 3, class nodes are added to the visible vector of the final stage with one node per class. Thus, we train the final stage in a supervised fashion.

Algorithm 1 is a slightly modified version of the CD-1 training technique for classification tasks. On line 2-6, we calculate hidden node activations. When the training is done for the classification (when there are class nodes), we add the contribution of the class input nodes by multiplying with U

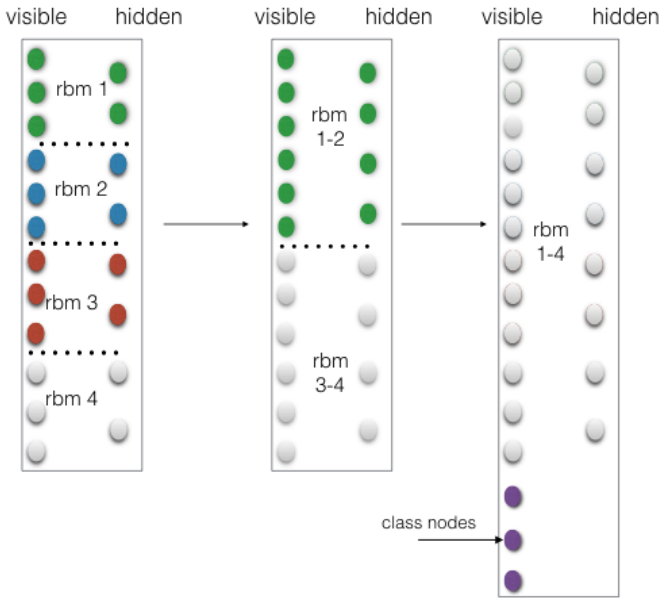

Fig. 3. Class Partitioned-RBM

matrix. In other words, the squashing function that calculates hidden node activations, in addition to $\mathbf{Wx}$, has the $\mathbf{Uy}$ component. On line 7, new probabilities are sampled from the hidden activations. Using sampled probabilities, the new values for $\mathbf{x}$ and $\mathbf{y}$ are calculated on lines 8-11. Using new predicted values of $\mathbf{x}$ and $\mathbf{y}$, new hidden activation probabilities are obtained on lines 12-16. Depending on whether the network is trained for classification or not, the contribution of the $\mathbf{Uy}$ is added to the probability calculations as described above. Finally, all network parameters are updated on line 17-23. Of course for CD-k, this process is repeated k steps before updating parameters.

Tieleman improved the Contrastive Divergence method by making Markov chains persistent [16]. In other words, the Markov chain is not reset for each training example or batch. This has been shown to significantly outperform Contrastive Divergence with one step, CD-1, with respect to classification accuracy. However, it does not address the problem of training speed. To alleviate the runtime performance issues of the Contrastive Divergence with multiple chains, Brekal *et al.* introduced an algorithm using parallel Markov chains [19]. However, the resulting Markov chains need to share messages, and the gradient is estimated by averaging chains. Also, this method does not improve the runtime performance of CD-1.

Finally, to classify a new data sample, the input vector will be constructed with all class nodes set to 0. Then, the input vector will be propagated to the hidden layer using Equation 1. A new visible vector will be sampled from the model based on the activation of hidden nodes. Since the visible vector contains class nodes as well, the class nodes' values will be used to predict the class label. In other words, the class node with the highest activation value determines the class label.

Algorithm 1 Class Partitioned-RBM

1: W: weight matrix from hidden nodes to visible nodes, U: weight matrix from hidden nodes to output nodes. b: bias on visible nodes, c: bias on hidden nodes, d: bias on class nodes. ϵ: is the learning rate. $x \sim p$ means x is sampled from p. $\sigma(x) = \frac{1}{1+e^{-x}}$
2: **if** classification **then**
3: $\quad h_i \leftarrow \sigma(c_i + \sum_j W_{ij}x_j + \sum_j U_{ij}y_j)$
4: **else**
5: $\quad h_i \leftarrow \sigma(c_i + \sum_j W_{ij}x_j)$
6: **end if**
7: $h_{i1} \sim p(h_i)$ {sample h_{i1} from a binomial distribution given h_i}
8: $x_j \leftarrow \sigma(b_j + \sum_i W_{ij}h_{i1})$
9: $x_{j1} \sim p(x_j)$
10: $y_j \leftarrow \sigma(d_j + \sum_i U_{ij}h_{i1})$
11: $y_{j1} \sim p(y_j)$
12: **if** classification **then**
13: $\quad h_{i2} \leftarrow \sigma(c_i + \sum_j W_{ij}x_{j1} + \sum_j U_{ij}y_{j1})$
14: **else**
15: $\quad h_{i2} \leftarrow \sigma(c_i + \sum_j W_{ij}x_{j1})$
16: **end if**
Update parameters:
17: $W \leftarrow W + \epsilon(h_{i1}x_j - h_{2i}x_{j1})$
18: $b \leftarrow b + \epsilon(x_j - x_{j1})$
19: $c \leftarrow c + \epsilon(h_{i1} - h_{i2})$
20: **if** classification **then**
21: $\quad U \leftarrow U + \epsilon(h_{i1}y_j - h_{2i}y_{j1})$
22: $\quad d \leftarrow d + \epsilon(y_{i1} - y_{i2})$
23: **end if**

IV. Experimental Results

We used the MNIST dataset for our experiments due to its wide use in evaluating RBMs and deep learning algorithms. The MNIST database (Mixed National Institute of Standards and Technology database) is a database of handwritten digits, constructed from NIST's SD-1 and SD-3 databases. MNIST has 60,000 training instances; rather than using the pre-defined test set of 10,000 images, we repeatedly split the training dataset for our cross-validation experiments. Each image is 28×28 pixels, and encodes a single handwritten digit (0 to 9). The raw digit images are scaled to fit in a 20×20 region (original aspect ratio maintained), and are then centered in the final 28×28 image, resulting in a white border around every image. This dataset was introduced in [20], and can be obtained from [21].

Unless stated otherwise, for all experiments, we trained models for 20 iterations. Moreover, when we compare methods in terms of the significance of results, we compare them using a paired t-$test$ with 99% confidence intervals.

Table I shows the results of our RBMs with 1500 hidden nodes. In the configuration column, Single RBM represents the traditional RBM and Partitioned-RBM represents a Partitioned RBM. The number of partitions in each training stage is defined in parentheses; (16-4-1) indicates that we trained the

TABLE I
CLASSIFICATION ACCURACY RATES

Configuration	Samples (10^3)	Accuracy (%)	Chain Operations (10^{10})
Single RBM	54	96.97	131.71
Partitioned-RBM-(16-4-1)	54-50-30	97.18	78.42

TABLE II
CLASSIFICATION F1 SCORES

Labels	PRBM (%)	Single RBM (%)
0	98.27	98.49
1	98.00	98.13
2	97.01	96.99
3	96.15	96.63
4	97.31	97.27
5	96.67	97.34
6	98.32	98.09
7	97.09	96.94
8	95.51	96.12
9	95.17	95.75

RBM first with 16 splits, then 4, and finally trained it as a single partition. Note that each successive Partitioned-RBM configuration starts with the output of the previous configuration, as described in Section II. The Samples column gives the number of training instances, selected at random from the total training set, that were used to train the given RBM. As the number of partitions decreases, we decrease the training set size to match the time complexity of the full Partitioned-RBM training process to that of the Single RBM. Each RBM was run for 20 iterations, and the classification accuracy rates reported are the mean values from 10-fold cross-validation (not using MNIST's predefined split between training and test data). As shown, Partitioned-RBM significantly outperforms the Single RBM.

By design, the computational complexity of the Partitioned-RBM is significantly better than that of the Single RBM trained on the entire dataset; it is evident that less computation would have been necessary for the Partitioned-RBM to yield superior performance. Based on Equation 2 we could tune the number of samples used in each stage and/or the number of partitions, in order to adjust the runtime performance. We emphasize that our aim is not to obtain the highest classification accuracy or lower error rates reported in the literature. Rather, we are seeking ways to obtain reasonable classification accuracy rates under bounded resources. Nevertheless, Partitioned-RBM significantly outperforms Single RBM as shown in Table I in terms of classification accuracy with significantly lower CPU requirements.

Table II shows $F1$ scores (which is harmonic mean of the precision and recall) for all class labels. Partitioned-RBM results are comparable to Single RBM if not better. This indicates that Partitioned-RBM has robust accuracy even though we partition the whole network into atomic partitions.

Indeed, when Partitioned-RBM has a smaller number of stages, the runtime performance will improve. However, ideally, the runtime performance gain should not be at the cost

TABLE III
PARTITIONED-RBM CLASSIFICATION ACCURACY

Configuration	Samples (10^3)	Accuracy (%)	Chain Operations (10^{10})
Partitioned-RBM-(16-4-1)	54-50-30	97.18	78.42
Partitioned-RBM-(16-1)	54-30	96.78	71.07

TABLE IV
CLASSIFICATION ACCURACY RATES WITH SAMPLES

Configuration	Samples (10^3)	Accuracy (%)	Chain Operations (10^{10})
Single RBM	20	96.43	47.04
Single RBM	10	95.47	23.52
Single RBM	5	94.15	11.76
Partitioned-RBM-(16-4-1)	54-10-20	96.90	49.02
Partitioned-RBM-(16-4-1)	54-10-10	96.25	25.50
Partitioned-RBM-(16-4-1)	54-10-5	95.44	13.74

of degradation in classification accuracies. In the following experiments, we ran Partitioned-RBM with fewer stages. Table III demonstrates that classification accuracy is still comparable but runtime is less. This is important, because training the model with many samples in the first stage will optimize the weights sufficiently; one does not have to use more samples when there are fewer splits to maintain a reasonable accuracy.

The purpose of the current study was to determine if RBM can be a viable learning technique under bounded computational resources. We can claim that increasing the number of partitions in the first stage is sufficient to optimize network weights to a degree that fewer samples or fewer training epochs are required in the last stage. In other words, the last stage with one split does not need to run on the whole dataset. Most of the computational effort is spent in the last stage when there is only one split, because the training needs to cover the whole weight matrix. We ran a series of experiments to determine the effects of sample size in last the stage and compared the results with Single RBM. As seen in Table IV, keeping runtime approximately the same, Partitioned-RBM performs significantly better than Single RBM for all experiments when the sample size decreases. This is not surprising, since Partitioned-RBM uses the full data set in the first stage. However, the result is important because the number of computations in the first stage is insignificant compared to the total number of operations. Thus, the following conclusions can be made: 1) if dimensions of the data are too high, increasing the number of partitions or splits will improve the runtime performance. 2) If the volume of data is too high, using more samples in early stages and fewer samples in later stages with fewer partitions will improve runtime performance.

Finally, to show how fast Partitioned-RBMs optimize weights, we ran experiments with different training epochs. As seen in Table V, Partitioned-RBM performs significantly better than Single RBM when trained for fewer epochs (for all experiments). This again demonstrates that Partitioned-RBM can perform well under bounded computational resources. On the other hand, Single RBM must run many iterations in order to obtain reasonable accuracy. The evidence from this

TABLE V
CLASSIFICATION ACCURACY RATES WITH ITERATIONS

Configuration	Samples (10^3)	iterations	Accuracy (%)	Chain Operations (10^{10})
Single RBM	54	1	93.72	6.59
Single RBM	54	5	96.50	32.93
Single RBM	54	10	96.89	65.86
Partitioned-RBM-(16-4-1)	54-50-50	1	94.30	6.27
Partitioned-RBM-(16-4-1)	54-50-50	5	96.67	31.37
Partitioned-RBM-(16-4-1)	54-50-50	10	97.26	62.73

experiment suggests that if one cannot afford to train a regular RBM for many iterations, there is a need to partition to allow it to run for many iterations.

V. DISCUSSION AND FUTURE WORK

This study set out to improve the runtime performance of RBMs without significant cost to precision and recall rates. From the results of our experiments, we are led to the conclusion that the Partitioned-RBM performs as well or better than a Single RBM with less CPU time. Table I demonstrates that the Partitioned-RBM has significantly better accuracy than Single RBM, with only about half the CPU time. This is important because Partitioned-RBM can run on very high dimension and high volume datasets. A single RBM can not be run efficiently on high volume data, while Partitioned-RBM can use all samples in the first stage, with many partitions running in parallel. When weights are optimized in early stages with many samples, the final stage requires a smaller number of samples.

Under bounded computational resources, with Partitioned-RBM we can either increase the number of splits and train it on many data samples, or adjust sample size and number of iterations to obtain better runtime performance, as seen in Tables III, IV, and V. We demonstrated that regular RBM, in general, requires many iterations and many training samples. However, this is not practical when computational resources are bounded.

One might ask why we did not add class nodes in early stages. Our initial results indicated that such a scheme does not provide any advantage. It appears that when weights between hidden nodes and class nodes, $\mathbf{U}$, are updated by all partitions, the weight matrix takes a long time to be optimized; one partition changes the effect of a previous partition. However, we are planning to investigate having each partition optimize its own $\mathbf{U}$ matrix, and we need to find a better mechanism to combine all $\mathbf{U}$ matrices when we move to the next stage in the training process.

To see our method perform even better, we need to run it on a dataset with high volume and high dimensions. We speculate that a single RBM cannot be trained optimally unless it runs many iterations, which will take days to train. On the other hand, Partitioned-RBM can run many iterations in the first stage with all data samples efficiently. We are planning to use this scheme to run classification tasks on many datasets with high dimensions and high volume.

Better runtime performance also enables us to create an ensemble of Partitioned-RBM. Thus, we plan to run experiments using ensembles. We speculate that an ensemble of Partitioned-RBMs will result in even better classification accuracy rates.

The results of the present study also suggest that Partitioned-RBM will provide even better runtime improvement when used as a component of a DBN. In other words, as we stack up more layers of RBMs, the runtime performance becomes more relevant. Our preliminary experiments indicate that Partitioned-RBM has comparable classification accuracy rates with Single RBM when used in DNBs. But, the results were not significantly better. Moreover, we found that DBN does not increase accuracy rates drastically as compared a regular RBM. It is possible this is because of the relative simplicity of the MNIST data set. Hence, we plan to 1) improve classification accuracy of Partitioned-RBM when used as a component of DBN, 2) investigate why DBN does not drastically improve accuracy rates, and 3) investigate alternative data sets.

Finally, in our previous study with DBNs, we showed that Partitioned-RBM preserves statistical spatial features in all layers of the network, while regular RBMs diffuse all spatially local features. We plan to carry out further experiments to determine whether preserving statistically local features will result in higher classification accuracy.

ACKNOWLEDGMENTS

We would like to thank Ben Mitchell who contributed to discussions about the material covered in this paper. We also like to thank Numerical Intelligent Systems Laboratory(NISL) members for their feedback.

REFERENCES

[1] D. H. Ackley, G. E. Hinton, and T. J. Sejnowski, "A learning algorithm for boltzmann machines," *Cognitive science*, vol. 9, no. 1, pp. 147–169, 1985.

[2] G. E. Hinton, "Training products of experts by minimizing contrastive divergence," *Neural computation*, vol. 14, no. 8, pp. 1771–1800, 2002.

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

[4] G. Hinton and R. Salakhutdinov, "Discovering binary codes for documents by learning deep generative models," *Topics in Cognitive Science*, vol. 3, no. 1, pp. 74–91, 2011.

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

[6] R. Salakhutdinov, A. Mnih, and G. Hinton, "Restricted Boltzmann machines for collaborative filtering," in *Proceedings of the 24th International Conference on Machine Learning*. ACM, 2007, pp. 791–798.

[7] G. E. Dahl, R. P. Adams, and H. Larochelle, "Training restricted Boltzmann machines on word observations," in *Proceedings of the 29th International Conference on Machine Learning*. ACM, 2012, pp. 679–686.

[8] H. Larochelle and Y. Bengio, "Classification using discriminative restricted Boltzmann machines," in *Proceedings of the 25th International Conference on Machine Learning*. ACM, 2008, pp. 536–543.

[9] J. Louradour and H. Larochelle, "Classification of sets using restricted Boltzmann machines," in *Uncertainty in Artificial Intelligence*. AUAI, 2011, pp. 463–470.

[10] T. Schmah, G. E. Hinton, S. L. Small, S. Strother, and R. S. Zemel, "Generative versus discriminative training of rbms for classification of fmri images," in *Advances in neural information processing systems*, 2008, pp. 1409–1416.

[11] H. Tosun and J. W. Sheppard, "Training restricted Boltzmann machines with overlapping partitions," in *Proceedings of the European Conference on Machine Learning-Principles and Practice of Knowledge Discovery in Databases (ECML-PKDD)*. Springer, 2014, vol. 8726, pp. 195–208.

[12] B. Mitchell, H. Tosun, and J. Sheppard, "Deep learning using partitioned data vectors," in *Proceedings of the International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2015, pp. 1–8.

[13] H. Tosun, B. Mitchell, and J. Sheppard, "Assessing diffusion of spatial features in deep belief networks," in *Submitted to International Joint Conference on Neural Networks (IJCNN)(IJCNN)*. IEEE, 2016.

[14] P. Smolensky, "Information processing in dynamical systems: Foundations of harmony theory," in *Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 1*, D. E. Rumelhart and J. L. McClelland, Eds. Cambridge, MA, USA: MIT Press, 1986, pp. 194–281.

[15] D. S. Lemons, *A student's guide to entropy*. Cambridge University Press, 2013.

[16] T. Tieleman, "Training restricted boltzmann machines using approximations to the likelihood gradient," in *Proceedings of the 25th International Conference on Machine Learning*. ACM, 2008, pp. 1064–1071.

[17] Y. Bengio, "Learning deep architectures for ai," *Foundations and Trends in Machine Learning*, vol. 2, no. 1, pp. 1–127, 2009.

[18] H. Larochelle, M. Mandel, R. Pascanu, and Y. Bengio, "Learning algorithms for the classification restricted boltzmann machine," *The Journal of Machine Learning Research*, vol. 13, no. 1, pp. 643–669, 2012.

[19] P. Brakel, S. Dieleman, and B. Schrauwen, "Training restricted boltzmann machines with multi-tempering: Harnessing parallelization," in *Artificial Neural Networks and Machine Learning–ICANN 2012*. Springer, 2012, pp. 92–99.

[20] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[21] Y. LeCun, C. Cortes, and C. J. Burges. The MNIST database of handwritten digits. Accessed 2014-01-15. [Online]. Available: http://yann.lecun.com/exdb/mnist/