Assessing Diffusion of Spatial Features in Deep Belief Networks

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

Ben Mitchell
Department of Computer Science
Johns Hopkins University
Email: ben@cs.jhu.edu

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

***Abstract*—Deep learning has recently gained popularity in many machine learning applications, but a theoretical grounding for the strengths, weaknesses, and implicit biases of various deep learning methods is still a work in progress. Here, we analyze the role of spatial locality in Deep Belief Networks (DBN) and show that spatially local information is lost through diffusion as the network becomes deeper. We then analyze an approach we developed previously, based on partitioning of Restricted Boltzmann Machines (RBMs), to demonstrate that our method is capable of retaining spatially local information when training DBNs. Specifically, we find that spatially local features are completely lost in DBNs trained using the "standard" RBM method, but are largely preserved using our partitioned training method. In addition, reconstruction accuracy of the model is improved using our *Partitioned-RBM* training method.**

I. Introduction

The technique of *Deep Learning* has gained tremendous popularity in recent years as it has been demonstrated to provide state-of-the-art performance on a wide range of machine learning tasks. In spite of these successful applications, however, there are still deep theoretical questions about why and how deep learning techniques work. The fact that features represented by hidden nodes in Convolutional Networks (ConvNets) explicitly represent localized regions suggests that deep learning methods make use of *spatially local* statistical information. However, after applying spatial statistics analysis tools to RBMs (a standard training algorithm for Deep Belief Nets), we showed that traditional RBMs are insensitive to spatially local structure in the input [1]. In order to preserve these spatial features, we developed Partitioned-RBMs. A Partitioned-RBM partitions the traditional Restricted Boltzmann Machine (RBM) into multiple smaller RBMs, which are trained independently before being re-joined in the final model. We found that such Partitioned-RBMs exhibited high reconstruction accuracy compared to traditional RBMs, so long as the initial input data contained spatially local structure [2]. We have also found that classification accuracy remains competitive with traditional RBMs [3]. Moreover, as the number of dimensions increases, the number of partitions can be increased to significantly reduce computational resource requirements. Other recently developed methods do not scale well; one is forced to use only some of the available data (sampling), run fewer iterations (stopping prior to convergence), or both. Our Partitioned-RBM method provides an innovative scheme to alleviate this problem.

In this paper, we extend our analysis to Deep Belief Networks (DBNs) with two and three hidden layers. We used the same spatial statistics tools of spatial variance and correlation to examine the diffusion of spatially local statistical information in DBNs trained using both traditional RBMs and Partitioned-RBMs. We used t-SNE [4] as a complementary tool in this analysis. We also studied the reconstruction performance of these DBNs, in both the presence and absence of spatially local structure.

A. Deep Learning

Modern deep learning techniques can be traced back to work started by Fukushima [5], and later extended by Hinton, Bengio, and LeCun (see [6]–[8]). In recent years, these techniques have truly matured and are now used in a wide range of state-of-the-art machine learning systems.

This rapid expansion of interest and application was enabled by a combination of improved learning algorithms and improved computational resources, which allowed much more complex models to be learned than had previously been possible. While there are now a variety of models that can be categorized as *deep*, they still tend to share a set of common features that can be traced back to the earliest work in this field. In particular, they tend to be connectionist networks consisting of multiple hidden layers, arranged in a basically feed-forward architecture.

It is the multiple hidden layers that gave rise to the label "deep." The mathematical power that comes from stacked non-linear units has long been recognized, but historically deep networks were difficult to train and scaled poorly. Using standard error-backpropagation on a deep network leads to gradient diffusion, resulting in a high probability of becoming stuck in a shallow local optimum and generally poor performance [9]. One answer to this problem is a technique called *pre-training*, in which the weights in a network are initially trained in a bottom-up fashion (i.e., one layer at a time); these weights are then treated as the initial values when performing standard gradient descent. The pre-training step is frequently done in an "unsupervised" fashion, using some form of compression-encoding as the optimization criterion

(e.g. using Autoencoders or RBMs to generate the weights for each layer) [10].

Deep networks of this type have produced impressive results on a number of problems that have historically been considered difficult, particularly in the field of computer vision. Some examples include handwritten character recognition [6], object recognition [11], de-noising [12], and re-construction of missing or obscured information [12].

There has been less theoretical work done on deep learning than work on practical applications, but a few things seem clear. Deep networks have a built-in regularization effect that does not appear to weaken even in the face of large amounts of data [13]; this gives them an advantage on data with noise (or complexities difficult to distinguish from noise). In addition, for *partitioned* deep models like ConvNets, exploiting spatially local information is important to performance [14].

Partitioned deep models work by partitioning nodes within layers of a network. This can be visualized as a feed-forward network in which layers are not fully connected; instead the nodes of each layer are broken into subsets, and each subset of nodes is fully connected only with a subset of nodes in the layer above. For instance, in a classic ConvNet a data vector represents an input (e.g. an image), and is effectively split into several small overlapping regions (image patches), each of which is checked for similarity to a set of templates (filters). In addition to ConvNets, several other deep learning techniques fall under the classification of *partitioned* deep learning, including [5], [15] and [16].

In partitioned learning models, the partitioning can be thought of as a simplifying assumption that reduces both the total number of parameters and the number of inter-parameter dependencies, thus simplifying the learning process. If the data conform to this assumption, this should provide an advantage. We have tested the validity of this assumption by training partitioned models on both "normal" images and "scrambled" images (in which spatially local structure was intentionally destroyed by permutation, but no statistical information was lost if the full vectors were considered). As expected, the performance of partitioned techniques is severely degraded if the data badly violates the assumption of spatial structure, but performance is increased on non-scrambled images [14].

One problem with these results is that they do not cover all deep learning methods; DBNs, for example, do not normally take a partitioned approach, so there may be different principles at play in their success. In this paper, we focus on the question of whether DBNs are in fact making use of spatially local statistical information, and if not whether we can modify the training procedure to incorporate this spatial locality prior and improve the performance of the network (without modifying the overall architecture or operation of the final DBN model).

B. Deep Networks

Autoencoders and RBMs are two standard methods used as components for deep learning where these components are stacked to form a deep network [10], [17]–[22]. Training is done layer-wise where each layer of the RBM or Autoencoder is trained individually.

An Autoencoder [18] is a feed-forward neural net that predicts its own input. An Autoencoder often introduces one or more hidden layers that have lower dimensionality than the inputs so that it creates a more efficient code for representation [23]. As a generative model, the Autoencoder encodes the input $\mathbf{x}$ into some representation $c(\mathbf{x})$ so that the input can be reconstructed from the resulting code. Training is done by updating model parameters (weights and biases) by minimize reconstruction error [10].

Likewise, a Restricted Boltzmann Machine is used to form a DBN where the network is trained layer-wise by compressing input while minimizing reconstruction error. The training method differs from Autoencoders (see below), but the goal is the same. 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 RBM was first proposed by Smolensky [24] 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 unconditional dependencies between hidden nodes, or between visible nodes. This model can be represented as Boltzmann energy distribution [25], 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 $p(\mathbf{x}, \mathbf{h})$ exactly is not tractable due to 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 [26]. Gibbs sampling makes use of the conditional probability, $p(\mathbf{h}|\mathbf{x}) = p(\mathbf{x}, \mathbf{h}) / \sum_{h'} p(\mathbf{x}, \mathbf{h}')$, which has a simple form. Since then, RBMs are used widely as basic components of deep learning algorithms [17], [18], [21].

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 [10], [27] including classification tasks [28]–[30] and other tasks such as Collaborative Filtering [31]. We use CD-1 in our experiments.

II. Partitioned RBMs

In order to preserve local features in the dataset and improve the performance of RBMs, Tosun and Sheppard developed a Partitioned-RBM [2]. The Partitioned-RBM partitions the traditional RBM into multiple smaller RBMs, which are trained independently (and ideally in parallel). The training

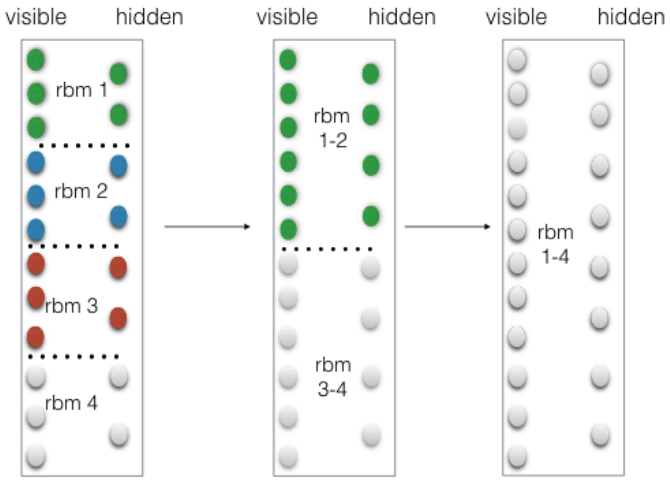


Fig. 1. Example partitioning approach for an RBM

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 the initial weights learned in the earlier stages.

Each stage of a Partitioned-RBM is trained using a set of sub-vectors "partitioned" from full-length 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 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 by 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 approximately $N_s \times N_h \times N_v$ probability calculations where N_s is number of samples, N_h number of hidden nodes and N_v number of visible nodes or features. While the results of the disjoint training will not be optimal (because they lack the full data vectors), they can go through far more training iterations in a fixed amount of time, allowing for either larger datasets or more training epochs in a given time frame. At the later stages, as the RBMs covers more links, the training requires many fewer epochs than normal to converge because the weights are closer to their optimal values than would be the case with random initialization. This enables the overall Partitioned-RBM hierarchy to achieve the same or higher performance in a fixed time period than a single RBM trained all at once.

Since this is a technique for training an RBM, it can be used to train a DBN without significant modification. Like a normal DBN, a Partitioned-RBM trained DBN is trained one layer at a time, using Partitioned-RBM training for each layer. Unlike the Convolutional Deep Belief Network (CDBN) model [32], the resulting network is still a traditional DBN in its architecture and operation. Like Partitioned-RBM, a CDBN uses *partitioning* for its filter-response units, but the partitions are permanent and persist even in the final trained model. The outputs are then combined with probabilistic max-pooling, resulting in an architecture that looks more like a CNN than a DBN. A CDBN is trained using standard sampling techniques. What differs is the architecture and the novel probabilistic max-pooling operation. Partitioned-RBM, on the other hand, is not a new architecture, but rather a new method of training a standard DBN. Thus, a Partitioned-RBM trained DBN can be compared directly to a conventionally trained DBN. Doing the same comparison with a CDBN is much less informative, as the two models have different representational power.

III. Analysis of Spatial Features

To analyze the spatial behavior of different learning algorithms, we need tools to measure and describe spatial structure in data. For this, we look primarily to the field of spatial statistics. In this work, we apply two different methods for detecting spatial structure. The first is to measure how inter-feature variance and correlation changes with spatial distance between features. This gives us a quantitative measure, but tells us only about pair-wise relationships, and is only meaningful as an average. The second is a more holistic qualitative analysis that can be done by using dimensionality reduction to embed the data in a plane so it can be visualized. This lets us see what kind of clusters or structure exists in the data, and compare the amount of structure present in different datasets.

A. Quantitative analysis

For quantitative analysis, we use a tool from spatial statistics called a *variogram*, which lets us examine the relationship between spatial distance and statistical correlation. Note again that we use *spatial* in the sense of spatial statistics; the distances here are between *feature locations*, not between datapoints in $\mathbb{R}^n$.

Given a spatial distribution Z, if we take two samples at locations s_1 and s_2, we can compare them for similarity. $s_1 - s_2$ is the spatial distance between the locations where the samples were taken, and $Z(s_1) - Z(s_2)$ is the difference between the sampled values. A variogram is a mapping between these two quantities:

$$var(Z(s_1) - Z(s_2)) = 2\gamma(s_1 - s_2), \forall s_1, s_2 \in D,$$

where D is the set of all possible sampling locations. Note that this is a static, deterministic function of the distance between two points. For $\gamma(\cdot)$ to be a well-defined function, some fairly stringent assumptions must be met (the distribution must be static and isotropic, among other things). If such a $\gamma(\cdot)$ exists, then $2\gamma(\cdot)$ is defined as the *variogram* of the distribution. Generally, this function will be plotted

and examined to observe the relationship between distance and difference. In geostatistics, where variograms are used for predictive modeling, these assumptions are often closely matched by the underlying problems.

Here, we use an empirical estimation of the variogram as a descriptive analytic tool. We acknowledge that the assumptions are violated by natural image data, so a "true" variogram is not well defined. Instead, we calculate an "average" empirical variogram. A "sample" in the context of image analysis is basically a pixel, so the location of the sample is merely the pixel location, and the value of the sample is the pixel value. In this case, difference-as-a-function-of-distance will not be static across a set of images, or even within a single image (there are many pairs of pixels that are the same spatial distance apart), so to get a static function we average across equidistant pixel pairs in all the images in our dataset.

To generate the variogram plots in this paper (e.g. Fig. 3), we compute the variance of each pair of features (computed across all images in the set). For each pair of features (i.e. pixels) (i, j), we have n samples (one per image in the dataset); this can be thought of as two vectors of values, where the length of the vector is n. We take the difference between these vectors, and then compute the variance of the resulting vector, $var(X^i - X^j)$. This gives us one scalar term for each pair of features. We then average together all feature pairs with equal inter-feature distances.

We can make similar plots for any other pairwise statistical measure. For example, we have done something similar using correlation in place of variance-of-differences. In this paper, we omit these *mean correlation plots* for the sake of space, since they show results consistent with the variograms. Copies of the mean correlation plots are available upon request. For more details about variograms, correlograms, and spatial statistics, the reader is directed to Cressie's book [33].

B. Qualitative analysis

To embed our data in a 2-dimensional space for visualization, we used *t-Distributed Stochastic Neighbor Embedding* (t-SNE) [4]. t-SNE was chosen because it is a dimensionality reduction technique that is particularly good for visualizing high dimensional data. The method builds a map in which distances between points reflect similarities in the data. It embeds high-dimensional data in lower dimensional space by minimizing the discrepancy between pairwise statistical relationships in the high and low dimensional spaces.

For a dataset of n points, let $i, j \in [1, n]$ be indices, and let $x_i \in \mathbf{X}$ and $y_i \in \mathbf{Y}$ refer to the ith datapoint of the original dataset and the low-dimensional equivalent respectively. Given a candidate embedding, t-SNE first calculates all pairwise Euclidean distances between data points in each space. The pairwise Euclidean distance between x_i and x_j is used to calculate a conditional probability, $p_{j|i}$, which is the probability that x_i would pick x_j as its neighbor. This probability is based on a Gaussian centered at x_i, with a variance based on sampling density (in densely sampled regions, the variance will be smaller than in more sparsely sampled regions). Similarly, pairwise conditional probabilities $q_{j|i}$ are calculated for each pair (y_i, y_j) in the low-dimensional embedding. As an objective function, t-SNE tries to minimize the discrepancies between the conditional probabilities for corresponding pairs in the high dimensional and low dimensional spaces by using *Kullback-Leibler* divergence (KL divergence). This is an intractable global optimization problem, so gradient descent is used to find a local optimum.

One drawback of t-SNE is that for large, high-dimensional datasets, even the local search can be quite slow. In such cases, PCA is sometimes used as a pre-processing step to speed up the computation and suppresses high-frequency noise. A typical example might retain the top 30 eigenvectors, and project the original data into the eigenbasis. t-SNE would then be applied to this 30-dimensional dataset to reduce it to a 2-dimensional set for visualization.

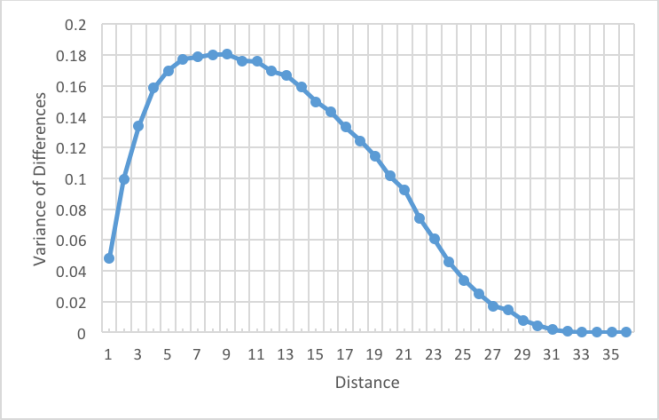
The resulting 2D plots make the structure (or lack thereof) readily apparent. Since the optimization is done on pair-wise vector distances, feature ordering (i.e. spatially local structure) in the high-dimensional data does not significantly change the qualitative properties of the low-dimensional data. Moreover, since the mapping is non-linear and non-parametric, it is relatively insensitive to whether information is encoded using sparse or distributed representations. As a result, t-SNE allows us to examine the presence of structure without having to worry about the form of that structure impacting our analysis.

IV. Experiments and 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-3 and SD-1 databases. MNIST has 60,000 training instances; we repeatedly split this 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 [6], and can be obtained from [34].

We measured the performance of the DBNs using reconstruction error, which is defined to be the mean difference between the original and reconstructed images. We used a binary reconstruction error, with a fixed threshold value of 30 to map pixels in the range $[0 - 255]$ to a binary 1 or 0 for the original images. To get the reconstructed image from the DBN, we propagate the image all the way to the last hidden layer of Deep Belief Network, then reconstruct it by reverse propagation to the visible layer. The resulting vector is binarized and compared with the original vector to calculate a reconstruction error, E. If x is the original vector, x' is the reconstruction, and both are of length n, then E is defined as:

$$E(\mathbf{x}, \mathbf{x}') = \frac{1}{n} \sum_{i=1}^{n} I(x_i \neq x_i')$$



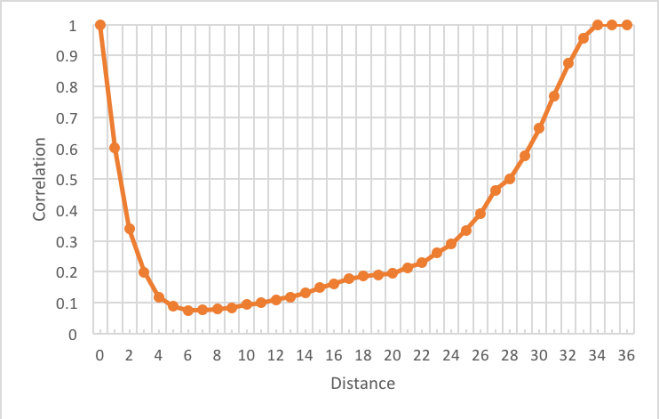


Fig. 2. Variogram and mean-correlation plots for the MNIST training set.

TABLE I
RECONSTRUCTION ERRORS FOR 2-LAYER DBN

Configuration	Samples (10^3)	Error (%)
Single RBM	60	2.59
Partitioned-RBM-(16-4-1)	60-50-30	1.05

To examine whether spatially local features are being disrupted, we constructed a 2-layer deep belief network, where each layer is composed of 784 hidden nodes. We calculated the variogram for the output of hidden nodes at each layer.

Table I shows the results of our DBN with two layers. The configuration column specifies training method used; "Single RBM" indicates the traditional RBM and "Partitioned-RBM" indicates a partitioned RBM. The number of partitions in each training stage is defined in parentheses; (16-4-1) indicates that we trained a Partitioned-RBM first with 16 splits, then 4, and finally trained 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 DBN. 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 15 iterations, and the error rates reported are the mean values from 10-fold cross-validation (not using MNIST's predefined training and test sets).

Partitioned-RBM significantly outperforms the Single RBM ($p > 99.99\%$ using a paired t-test). By design, the computational complexity of the full stack of Partitioned-RBMs is comparable to or faster than that of the Single RBM trained on the entire dataset; however, it is evident that less computation would have been necessary for the Partitioned-RBM to yield superior performance.

We generated variogram plots as described in Section III-A. Figure 2 shows the variogram and mean-correlation of the original MNIST dataset; note that the latter part of the plots is caused by the white "border pixels" that are an artifact of MNIST. Figure 3 shows the variogram plots for subsets of the data corresponding to digits 0, 5 and 9 (other digits are omitted for brevity, but look similar). The first column shows variograms of the raw input vectors for each subset, the second column shows results of the Single RBM, and the third shows results of the Partitioned-RBM. The y-axis represents the mean variance of differences, and the x-axis represents Euclidean pixel distance between points. For all digits, Partitioned-RBM produces an "arch" pattern consistent with the original digit plot. In comparison, the hidden layers of the traditional RBM do not preserve the relationship between distance and difference.

We also used t-SNE (as described in Sec. III-B) to visualize the activations at the hidden nodes. To apply t-SNE to hidden nodes, we generated sample points by setting a selected hidden neuron to 1.0 and all other hidden nodes to 0.0, and then computing corresponding input node activations. Thus, the weights between that hidden node to all visible nodes captures a "feature" (this can also be referred to as a filter, or template, depending on context). For this experiment, we constructed a 3-layer Deep Belief Network where each layer has 784 nodes. Results are shown in Figure 4. The first row shows the results for the Partitioned-RBM and the second row for a Single RBM. Columns corresponds to network layers 1–3. The scatter plot of activations shows that the Partitioned-RBM has some natural clusters, whereas the Single RBM output closely approximates a zero-mean Gaussian.

To ensure that the t-SNE method is correctly accounting for possible re-ordering of features, we applied t-SNE to both permuted and non-permuted MNIST data. Figure 5 shows results of these experiments. The left figure is for original data and right figure corresponds to permuted data. The permuted data generates a t-SNE plot with qualitatively similar structure to that generated from the original data; importantly, this resembles the output generated from the Partitioned-RBM, but it does not resemble the Gaussian-like output generated from the traditional RBM.

To explore how diffusion progresses across layers in the Partitioned-RBM, we paused the training between stages (i.e.

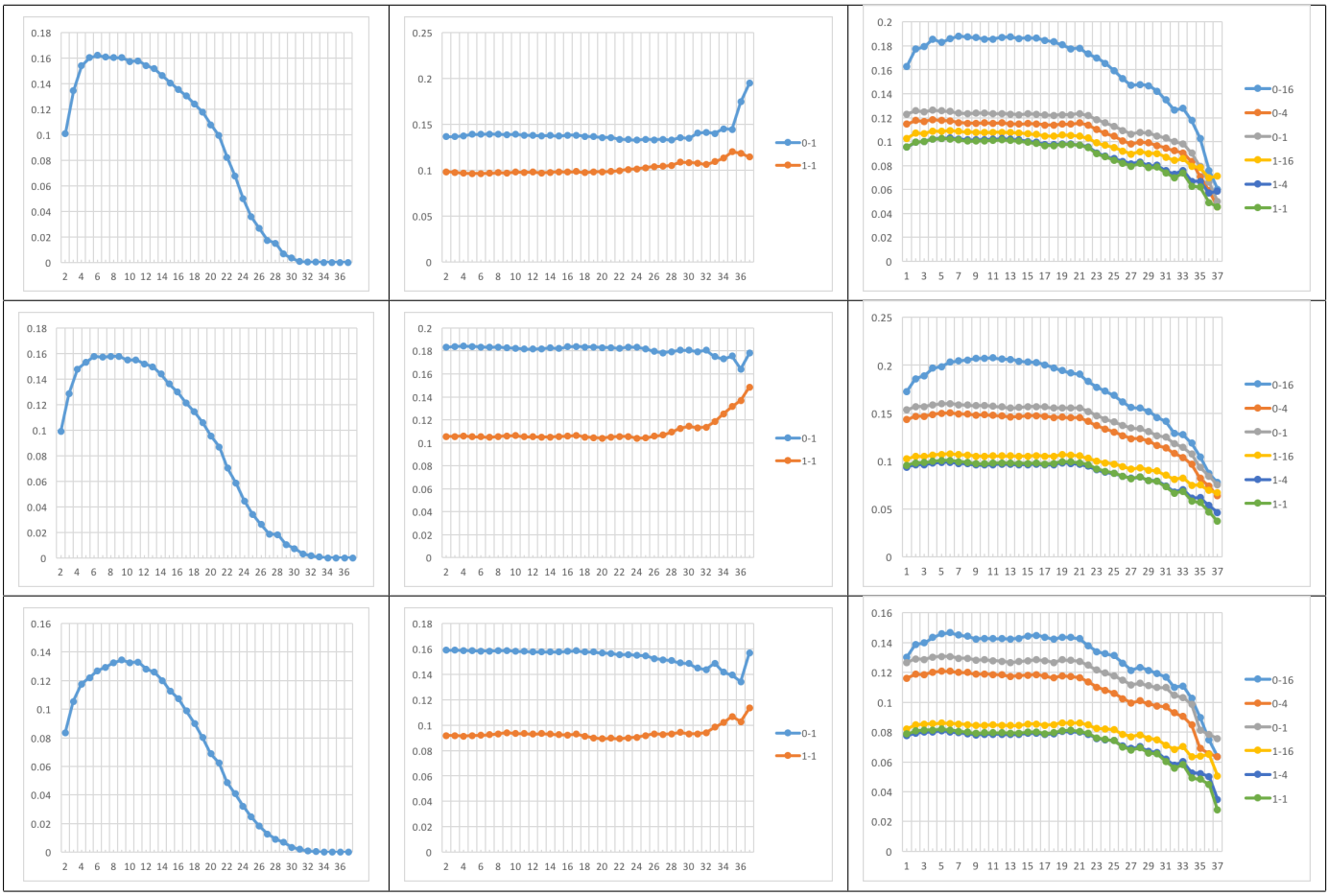


Fig. 3. Variograms: labels 0, 5 and 9 (from top to bottom). Labels of the form $N-P$ indicate data for hidden layer N of a Deep Belief Network based on Partitioned-RBMs with P partitions. First column corresponds result of original digits. Second column corresponds to Single RBM and last column corresponds to Partitioned-RBM

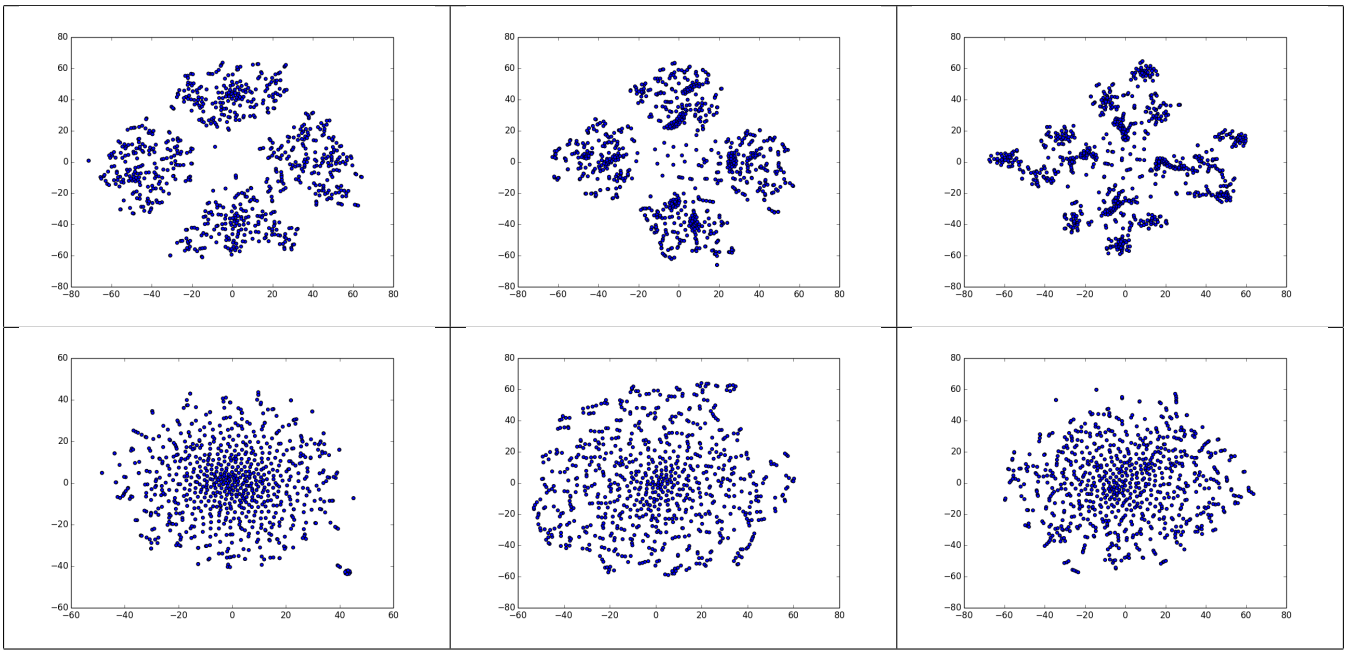


Fig. 4. MNIST t-SNE mappings for hidden node activations: 15 iterations

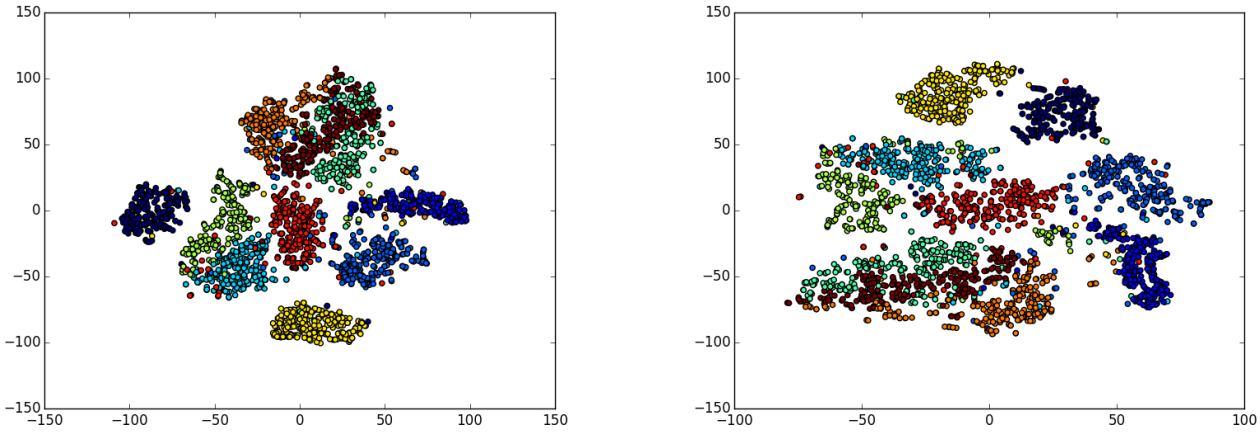


Fig. 5. MNIST t-SNE mappings for both original data and permuted data

just before the number of partitions was decreased). As the number of partitions changed, we plotted the t-SNE mapping for the first hidden layer of the DBN (first RBM). Figure 6 shows that structure continues to be present, though some consolidation does take place as partitions are joined.

V. Discussion

From the results of our experiments using spatial statistics and t-SNE, we are led to the conclusion that the Partitioned-RBM in a Deep Belief Network is making use of spatially local information, where the Single RBM is not. The variograms (Fig. 3) demonstrate that the Partitioned-RBM output has broadly similar spatial statistics as the original dataset. While the plots are not identical, similar overall trends are present. Partitioned-RBM training preserves these statistical patterns even in higher layers of the network. The same plots for Single RBM training show that spatial locality is lost in the first hidden layer.

Figure 5 shows what happens when the original data is scrambled beforehand, by generating a random permutation, and then applying it to each input vector. Despite the disruption of spatial organization of features, the transformation is lossless, and structure obtained in the t-SNE projection is similar to that for the original data. This suggests that if the Single RBM had preserved any spatial features, we should see similar structure in the t-SNE projection (even if the spatial organization of those features was not preserved). As the t-SNE results show, there is no structure in the t-SNE projection after the first layer of the Single RBM; the projected distribution closely approximates a Gaussian (i.e. it is indistinguishable from noise). This leads us to the conclusion that traditional RBMs do not retain spatially local statistical information in any recoverable form. As a result, any deep network trained using standard RBMs will lose all spatial information in the first hidden layer. On the other hand, the Partitioned-RBM training technique preserves spatially local information, meaning a deep network trained using this method can make use of spatial patterns in all layers of the network.

This result, combined with the performance edge the Partitioned-RBM has in practice, reinforces the hypothesis that the MNIST data has relevant spatially local structure, and that like other partitioned deep methods, the Partitioned-RBM achieves its performance due to an implicit model bias that assumes (and exploits) the presence of spatially local features. DBNs trained with the standard RBM method lose spatially local features, and are therefore at a disadvantage because they are attempting to solve a harder problem. Without the constraint imposed by the assumption of local structure, the standard RBM training algorithm is left with a much larger hypothesis space to search.

In spite of this, traditional DBNs achieve remarkable performance when applied to classification, image recognition and many other applications. We have begun to explore applications where we can apply partitioned-data techniques, and we are in the process of comparing a partitioned-data DBN to a traditional DBN in terms of classification performance (as opposed to the reconstruction task examined here). This is important in two ways: 1) we would like to determine whether preserving spatially local structure in higher layers of the network can improve classification accuracy, and 2) we would like to further explore how and why deep learning works, by analyzing how traditional DBNs achieve their performance even without exploiting any spatially local information.

Finally, it appears that Partitioned-RBM preserves local structures by imposing some kind of ordering on hidden nodes. As a result, it maintains similar structures in all layers. We plan to evaluate whether this feature can be productively applied to temporal applications and time-series data, as well as the spatial data used here.

Acknowledgments

We would like to thank Nathan Fortier, Shane Strasser, and Logan Perreault, who all contributed to discussions about

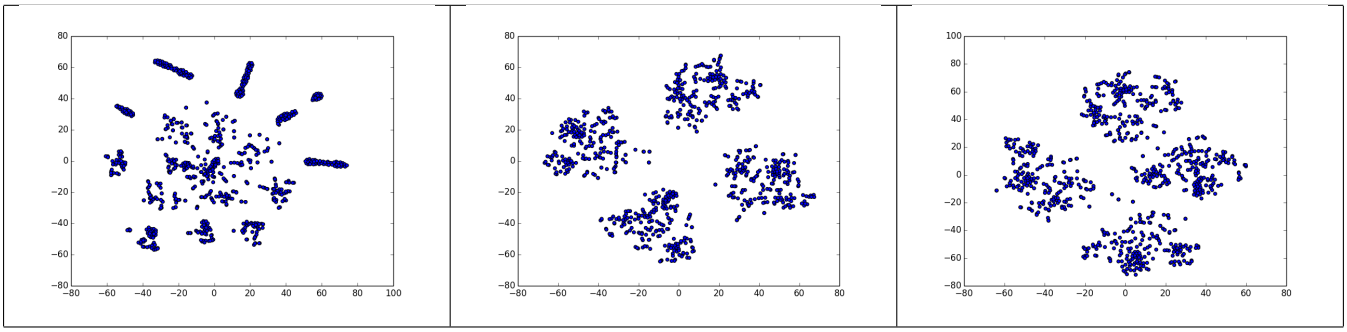


Fig. 6. MNIST t-SNE mappings for hidden node activations: Partitions

the material covered in this paper, and Carey Priebe, for his expert advice on spatial statistics. We would also like to thank the Johns Hopkins University CS department for providing financial support to Ben Mitchell while he completes his PhD.

REFERENCES

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

[2] H. Tosun and J. W. Sheppard, "Training restricted Boltzmann machines with overlapping partitions," in *Machine Learning and Knowledge Discovery in Databases,ECML PKDD*, ser. Lecture Notes in Computer Science. Springer, 2014, vol. 8726, pp. 195–208.

[3] H. Tosun and J. Sheppard, "Fast classification under bounded computational resources using partitioned-rbms," in *to appear in International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2016.

[4] L. Van der Maaten and G. Hinton, "Visualizing data using t-sne," *Journal of Machine Learning Research*, vol. 9, no. 2579-2605, p. 85, 2008.

[5] K. Fukushima, "Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position," *Biological Cybernetics*, vol. 36, no. 4, pp. 193–202, 1980.

[6] 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.

[7] G. E. Hinton, S. Osindero, and Y. W. Teh, "A fast learning algortihm for deep belief nets," *Neural Computation*, vol. 18, pp. 1527–1554, 2006.

[8] Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle, "Greedy layer-wise training of deep belief networks," *Advances in Neural Information Processing Systems 19 (NIPS '06)*, pp. 153–160, 2007.

[9] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *IEEE Transactions on Neural Networks*, vol. 5, no. 2, pp. 157–166, March 1994.

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

[11] Y. LeCun, F. Huang, and L. Bottou, "Learning methods for generic object recognition with invariance to pose and lighting," *Proceedings of IEEE CVPR '04*, vol. 2, pp. 97–104, 2004.

[12] J. Xie, L. Xu, and E. Chen, "Image denoising and inpainting with deep neural networks," in *Advances in Neural Information Processing Systems*, 2012, pp. 341–349.

[13] D. Erhan, Y. Bengio, A. Courville, P. Manzagol, and P. Vincent, "Why does unsupervised pre-training help deep learning?" *Journal of Machine Learning Research*, vol. 11, pp. 625–660, 2010.

[14] B. Mitchell and J. Sheppard, "Deep structure learning: Beyond connectionist approaches," *Proceedings of ICMLA '12*, pp. 162–167, 2012.

[15] S. Behnke and R. Rojas, "Neural abstraction pyramid: a hierarchical image understanding architecture," *International Joint Conference on Neural Networks (IJCNN)*, vol. 2, pp. 820–825, 1998.

[16] D. George and B. Jaros, "The HTM Learning Algorithm," *Numenta, Inc.* `www.numenta.com`, March 2007, March 2007. [Online]. Available: www.numenta.com

[17] 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.

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

[19] R. Salakhutdinov and G. E. Hinton, "Deep Boltzmann machines," in *International Conference on Artificial Intelligence and Statistics*, 2009, pp. 448–455.

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

[21] 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.

[22] H. Larochelle, Y. Bengio, J. Louradour, and P. Lamblin, "Exploring strategies for training deep neural networks," *The Journal of Machine Learning Research*, vol. 10, pp. 1–40, 2009.

[23] B. A. Olshausen *et al.*, "Emergence of simple-cell receptive field properties by learning a sparse code for natural images," *Nature*, vol. 381, no. 6583, pp. 607–609, 1996.

[24] 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.

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

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

[27] 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.

[28] 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.

[29] 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.

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

[31] 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.

[32] H. Lee, R. Grosse, R. Ranganath, and A. Y. Ng, "Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations," in *Proceedings of the 26th Annual International Conference on Machine Learning*. ACM, 2009, pp. 609–616.

[33] N. A. C. Cressie, *Statistics for Spatial Data*. Wiley-Interscience, 1993.

[34] 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/