Deep Learning Using Partitioned Data Vectors

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

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@cs.montana.edu

***Abstract*—Deep learning is a popular field that encompasses a range of multi-layer connectionist techniques. While these techniques have achieved great success on a number of difficult computer vision problems, the representation biases that allow this success have not been thoroughly explored. In this paper, we examine the hypothesis that one strength of many deep learning algorithms is their ability to exploit spatially local statistical information.**

We present a formal description of how data vectors can be partitioned into sub-vectors that preserve spatially local information. As a test case, we then use statistical models to examine how much of such structure exists in the MNIST dataset. Finally, we present experimental results from training RBMs using partitioned data, and demonstrate the advantages they have over non-partitioned RBMs. Through these results, we show how the performance advantage is reliant on spatially local structure, by demonstrating the performance impact of randomly permuting the input data to destroy local structure. Overall, our results support the hypothesis that a representation bias reliant upon spatially local statistical information can improve performance, so long as this bias is a good match for the data. We also suggest statistical tools for determining *a priori* whether a dataset is a good match for this bias or not.

I. Introduction

In the past few years, the area of exploration known as *Deep Learning* has demonstrated the ability of multilayer connectionist networks to achieve good performance on a range of difficult machine learning problems, and has been gaining increasing prominence and attention in the machine learning and neural network communities. The theoretical basis for understanding *why* these techniques perform well on particular problems, however, has only begun to be explored. A deeper understanding of how and why deep learning algorithms work will not only help us to decide what problems are good candidates for their application, but may also suggest ways of improving existing techniques or harnessing their strengths in novel contexts.

The hypothesis we seek to examine is the hypothesis that many deep learning algorithms make use of spatially local structure in their input data to help them achieve good performance. There has long been an intuition in the deep learning community that this hypothesis is likely to hold, based partly on everyday experience, and partly on the structure of the human visual cortex. To our knowledge, however, there has been no previous attempt to formalize and test this hypothesis specifically.

In this paper, we take the example of the Restricted Boltzman Machine (RBM) and show how its performance can be improved while simultaneously decreasing total training time. We do this by introducing a vector-partitioning scheme borrowed from deep learning algorithms like Convolutional Networks. We then analyze the input data to see if we can identify the statistical properties that are being exploited by the Partitioned-RBM to achieve this performance. In so doing, we hope to expose part of the representation bias of deep, vector-partitioning approaches in general. As Tom Mitchell pointed out in his seminal paper [1], not only is a bias necessary for learning, but examination of that bias is critical to understanding and improving our learning algorithms.

A. Deep Learning

While techniques similar to modern deep learning algorithms have been proposed previously (see Fukushima [2], for example), it has only been the past few years that have seen deep learning come into its own. Works begun by Hinton, Bengio, and LeCun (for example, [3]–[5]) have since been extended by many others, and a set of common characteristics defines an overall framework for deep learning. In particular, the word "deep" in this context refers to the fact that there are several levels of functional abstraction between inputs and outputs; generally, this means a connectionist network consisting of multiple hidden layers in a feed-forward architecture.

The innovation in deep learning has not been the proposal to use such networks, but rather the algorithms required to train the weight vectors. Historically, deep networks were difficult to train, because of a credit assignment problem; standard error-backpropagation suffers from gradient diffusion if applied to a deep network, resulting in generally poor performance [6]. Most deep learning techniques now get around this problem by performing some form of "unsupervised pre-training," which involves learning the weights to minimise reconstruction error for (unlabeled) training data, one layer at a time in a bottom up fashion. This is then often followed by a supervised learning algorithm which, it is hoped, has been initialized in a good part of the search space.

Deep networks of this type have demonstrated good performance for a number of traditionally difficult tasks, many in the domain of computer vision. Some examples are handwritten character recognition [3], object recognition [7], denoising [8], and re-construction of missing or obscured information [8].

There has also been some work attempting to derive a theory to explain the success of deep learning. Erhan and Bengio [9]

suggest that unsupervised pre-training acts as a regularizer, and we have suggested in previous work [10] that it also takes advantage of spatially local statistical information in the training data.

While some deep learning techniques (such as Deep Belief Networks) treat all the elements of an input vector symmetrically, many take an approach that partitions the data vectors into shorter sub-vectors, and performs analysis on the sub-vectors before re-combining the analyzed data to form a representation of the full data vector. Convolutional Networks are the most well known of these, though there are several others, including [2], [11] and [12].

From a statistical and information-theoretic standpoint, this type of analysis seems like it *should* be highly detrimental to performance. After all, any statistical information relating two features that are not contained in the same partition is completely hidden from the model. Unless we make the rather absurd assumption that no relevant statistical patterns exist in any of this data, we are left with less data upon which to base our model. It is fundamental to all statistical learning that the performance of a model is a function of the amount of information available; so long as it is sampled from the same underlying distribution, more information will always lead to a better model.

In practice, however, things are more complicated. Aside from the impossibility of truly unbiased learning, the main reason for this is that for most real-world problems, we have neither enough samples nor the computational capacity to generate a fully optimal statistical model of our data. Most of the models we are interested in yield exponential complexities, and so we must rely on approximations to estimate them. Effectively all of the problems experienced over the history of connectionist networks, going all the way back to McCulloch and Pitts [13], have been caused by the fact that finding the optimal set of connection weights is intractable. The history of the field has largely been a progression of ever improving gradient descent-based approximation algorithms, which have enabled more and more complex network architectures to be capable of converging on an acceptable solution.

In this context, deep learning can be seen as the most recent step in this long progression, with new approximation learning algorithms allowing for deeper architectures than could be successfully used previously. This brings us back to the question of why they work; simplifying assumptions often allow much more tractable approaches to problem solving (since only the simplified version of the problem needs to be solved), but the results are only as good as the match between the assumptions made and the properties that actually hold for the original problem.

We suggest that deep learning algorithms, and in particular the ones that partition data vectors, are taking advantage of spatially local statistical structure for their performance. This requires not only that such structure exist, but that it be highly relevant to the performance criterion being optimized. If this hypothesis is correct, however, it means we may be able to exploit this structure in other contexts as well.

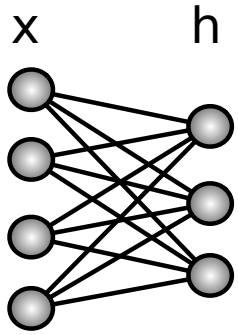

Fig. 1. Basic architecture of a Restricted Boltzmann Machine

To test this hypothesis, we examine the application of this type of partitioning scheme to an RBM, which is not inherently a deep learning model. In particular, because RBM network architectures form bipartite graphs, there is no deeply connected layering in place. In spite of this fact, we can apply a vector-partitioning approach as a "deep feature extraction" method to help us learn better RBMs and to learn them faster as well.

B. Restricted Boltzman Machines

The RBM model 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 shown in Figure 1. There are no dependencies between hidden nodes, or between visible nodes; thus, an RBM forms a complete bipartite graph. This model can be represented as a Boltzmann energy distribution [15], in which the probability distribution of the RBM is given as follows:

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

where the partition function defines configurations over all possible $\mathbf{x}$ and $\mathbf{h}$ vectors

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

The conditional probability can be written in terms of the energy function as follows:

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

The probability of data $p(\mathbf{x})$ is then obtained by marginalizing over the hidden vector $\mathbf{h}$.

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

Calculating $p(\mathbf{x}, \mathbf{h})$ is not tractable due to the partition function, Z; however, the conditional probability, $p(\mathbf{h}|\mathbf{x}) = p(\mathbf{x}, \mathbf{h}) / \sum_{h'} p(\mathbf{x}, \mathbf{h}')$, has a rather simple form. To differentiate from the current vector $\mathbf{h}$, we use $\mathbf{h}'$ to represent all hidden vectors (configurations) of size n. Then given our definition of the Boltzmann distribution, we obtain

Algorithm 1 CD-1($\mathbf{x}_i$, α)

1: **Input**: $\mathbf{x}_i$: data sample, α: learning rate.
2: **Model Parameters**: $\mathbf{W}$: weight vector, $\mathbf{b}$: bias vector on visible nodes, $\mathbf{c}$: bias vector on hidden nodes
Notation: $\mathbf{x} \sim p$ means $\mathbf{x}$ is sampled from p
Positive Phase:
3: $\mathbf{x^0} \leftarrow \mathbf{x_i}$
4: $\mathbf{h^0} \leftarrow \sigma(\mathbf{c} + \mathbf{W}\mathbf{x})$
Negative Phase:
5: $\tilde{\mathbf{h}}\mathbf{^0} \sim \mathbf{p}(\mathbf{h}|\mathbf{x^0})$
6: $\tilde{\mathbf{x}} \sim \mathbf{p}(\mathbf{x}|\tilde{\mathbf{h}}\mathbf{^0})$
7: $\mathbf{h^1} \leftarrow \sigma(\mathbf{c} + \mathbf{W}\tilde{\mathbf{x}})$
Update parameters:
8: $\mathbf{b} \leftarrow \mathbf{b} + \alpha(\mathbf{x^0} - \tilde{\mathbf{x}})$
9: $\mathbf{c} \leftarrow \mathbf{c} + \alpha(\mathbf{h^0} - \mathbf{h^1})$
10: $\mathbf{W} \leftarrow \mathbf{W} + \alpha(\mathbf{h^0}\mathbf{x^0} - \mathbf{h^1}\tilde{\mathbf{x}})$

$$p(\mathbf{h}|\mathbf{x}) = \frac{\exp(\mathbf{h}^\top \mathbf{W}\mathbf{x} + \mathbf{b}^\top \mathbf{x} + \mathbf{c}^\top \mathbf{h})/Z}{\sum_{h' \in \{0,1\}^n} \exp(\mathbf{h'}^\top \mathbf{W}\mathbf{x} + \mathbf{b}^\top \mathbf{x} + \mathbf{c}^\top \mathbf{h'})/Z}$$

Although training of the RBM was found to be tractable, training was initially inefficient, and RBMs did not gain popularity for several years until Hinton *et al.* developed Contrastive Divergence, a method based on Gibbs Sampling [16]. Since then, RBMs are used widely as basic components of deep learning algorithms [17]–[19]. RBMs have also been successfully applied to classification tasks [20]–[22]. Moreover, RBMs have been applied to many other learning tasks such as Collaborative Filtering [23].

The Contrastive Divergence (CD) method provides a reasonable approximation to the likelihood gradient of the energy function. Algorithm 1 shows pseudocode for training RBMs using a one step Contrastive Divergence method. The algorithm accepts a sample data instance and a set of model parameters: weight vector ($\mathbf{W}$), visible layer bias vector ($\mathbf{b}$), hidden layer bias vector ($\mathbf{c}$), and learning rate (α). It updates the model parameters in two phases, referred to as the *positive* phase and the *negative* phase respectively.

First, in the positive phase (lines 3-4), the probability of the hidden node is calculated for all hidden nodes, given the visible vector. In the negative phase (lines 5-7), the probability of each hidden node is determined by sampling from the model. First a sample of points for the hidden nodes is drawn based on the current estimate of the distribution $\mathbf{p}(\mathbf{h}|\mathbf{x^0})$ (line 5). Using these sampled points $\mathbf{h^0}$, the current estimate of $\mathbf{p}(\mathbf{x}|\mathbf{h})$ is used to sample points for the visible nodes $\mathbf{x}$ (line 6). Finally, on line 7, the probabilities of the hidden nodes are updated based on the sampled vector for the visible nodes. The parameters of the network are updated on lines 8-10. Contrastive Divergence need not be limited to one forward and backward pass in this matter, and Algorithm 1 can be extended by creating a loop around lines 3–7. Then for $k > 1$, the positive and negative phases are repeated k times before the parameters are updated.

The CD-1 algorithm (i.e, Contrastive Divergence with one step) has proven to be sufficient for many applications [24], [25]. CD-k is rarely used, because resetting the Markov chain after each parameter update is inefficient (as the model has already changed [24]). As an alternative, Tieleman modified the Contrastive Divergence method by making Markov chains persistent [24]; one or more persistent chains are kept during training, leading to greater efficiency for the multi-step case. Even so, many applications have demonstrated minimal (if any) improvement in performance using persistent Markov chains.

II. Partitioned RBMs

To improve the performance of RBMs, Tosun and Sheppard proposed a training method for RBMs that partitions the network into several subnetworks [26] that are trained independently, and incrementally re-combined until only a single, all-inclusive partition is left. With the partitioned RBM method, training involves several levels of partitioning and training. At each level, the RBM is partitioned into multiple RBMs as shown in Figure 2. In this figure, the partitions do not overlap, although [26] also demonstrates additional improvement in training when some amount of overlap is permitted. Each partitioned RBM is then trained using a set of sub-vectors partitioned from the training instances; in effect, each RBM is trained on instances that contain only the features that correspond to the input nodes assigned to that RBM.

All the RBMs at a given level can be trained simultaneously and in parallel. Once they have been trained, a new partitioning with fewer splits is created, which forms the basis for the next level up. Figure 2 shows an example in which the first layer has 4 distinct RBMs, the second layer has 2, and the final layer has 1. The power of the method comes from the fact that all levels share the same weight matrix. This means that during training, each RBM updates its own part of the globally shared weights; because the partitions do not overlap, there is no data dependency (which would prevent easy parallelization), but there is only one set of weights. The result is that the later levels begin their training with weights that have been pre-initialized by the earlier levels.[1]

The reason this is useful is that, when RBMs are small (in terms of number of nodes/weights being updated), they can be trained more quickly. While the results of the disjoint training will not be perfect (because they lack the full data vectors, they will be missing some relationships), they can be allowed to run for many epochs. At the higher levels, the RBMs become larger from recombining, but training requires many fewer epochs than normal to converge because the weights are much closer to their optimal values than would be the case with random initialization. This enables the overall Partitioned-RBM hierarchy to achieve higher performance over a given training interval than a single traditional RBM.

[1]Note that Fortier *et al.* [27]–[29] suggest an approach to enabling and exploiting overlap that would also permit parallelization and distribution of the networks being optimized.

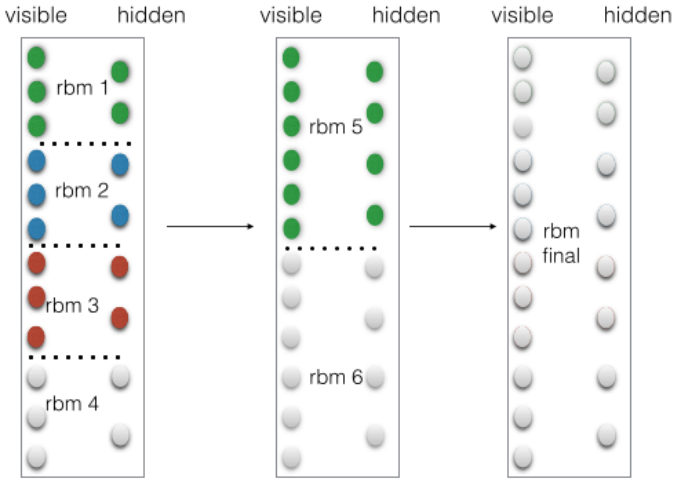

Fig. 2. Example partitioning approach for an RBM

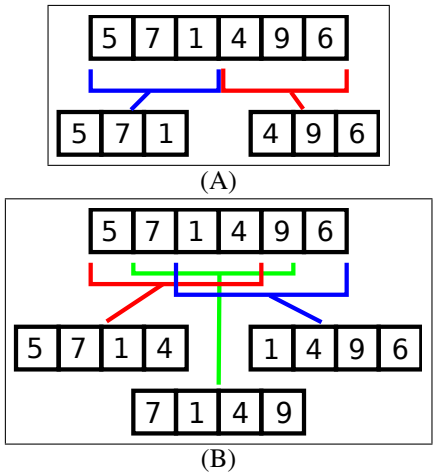

Fig. 3. Simple examples of vector partitioning. (A) breaking a vector into two disjoint sub-vectors, and (B) breaking it into three overlapping sub-vectors.

III. Partitioning Theory

To describe the statistical effects of partitioning on data vectors more formally, we will first introduce a notation to describe different partitioning schemes. Given data vectors in $\mathbb{R}^n$, we define a partitioning function π (not to be confused with the "partition function" Z used in defining the Boltzmann distribution) as

$$\pi : \mathbb{R}^n \rightarrow \{\mathbf{u}_0, \ldots, \mathbf{u}_k\}, \mathbf{u}_i \in \mathbb{R}^s$$

which takes a single input vector in $\mathbb{R}^n$ and produces a set of k output vectors in $\mathbb{R}^s$, where $s < n$. Notice that, while we will be restricting our attention to non-overlapping partitions, nothing in this definition requires the partitions to be disjoint.

As a simple example, we can consider a function that simply splits length 6 vectors into two equal length halves; in this case, $n = 6$, $s = 3$, and $k = 2$; see Figure 3-A. As mentioned, partition membership need not be mutually exclusive, and partitions may overlap. So, we might have another partition function that takes our length 6 vectors and produces 3 outputs sub-vectors of length 4 by having a size-4 partition window placed at the front, middle, and end of the original vector. This would give us $n = 6$, $s = 4$, $k = 3$; see Figure 3-B.

In general, all partition functions must have $1 \leq s \leq n$ and $1 \leq k \leq n$. There are further constraints between s and k, but the precise formula will vary depending on the nature of the partitioning being done. In the simple case given above, $s + k \leq n+1$, because the longer the sub-vector is, the fewer size-s windows can be fit into the original vector without repetition.

Partitioning functions can also be constructed that respect structural properties of the original data vectors. For example, in the case of vectors representing images, there is a natural 2-dimensional layout to the vector elements (pixels). While it is possible to treat an image as a single (long) vector and partition it in the way described above, it would make more sense to treat the image as a 2D grid and partition it into smaller 2D patches. For example, an 8×8 pixel image might be split into a set of 4×4 pixel sub-vectors. In general, it is desirable for adjacencies and distances between elements of the original data vector to be preserved in the partitioned sub-vector, especially given our hypothesis that we can exploit spatially local characteristics of the data.

If we want to talk about what statistical information is destroyed by the partitioning and what is preserved, we need to be able to reference more than one vector at a time (since a single sample contains no statistical information in isolation). Therefore, we will assume we have a dataset of vectors

$$\mathbf{D} = \{\mathbf{x}_0, \ldots, \mathbf{x}_m\}, \mathbf{x}_i \in \mathbb{R}^n,$$

where each $\mathbf{x}_i$ is composed of n features. We will treat each feature as a random variable X^j, meaning a vector is interpreted as $\mathbf{x}_i = \{X_i^0, \ldots, X_i^n\}$. We will use $\mathbf{X^j}$ in the absence of any subscript to denote the set of all the jth vector features in the dataset,

$$\mathbf{X^j} = \{X_0^j, \ldots X_m^j\}.$$

We can now describe the statistical properties of these random variables across the dataset. For individual features, we can compute statistical measures like mean and variance. For any given pair of features (i, j), we can compute statistical measures like correlation, covariance, mean-deltas, and so forth; each vector in $\mathbf{D}$ provides one sample for each feature, so the number of samples will be the cardinality of $\mathbf{D}$.

By applying a partition function π to every element of a dataset $\mathbf{D}$, we can generate a new dataset, which will be a set of the sets created by applying π to the $\mathbf{x}_i$:

$$\mathbf{D}_\pi = \{\{\pi(\mathbf{x}_0)\}, \ldots, \{\pi(\mathbf{x}_m)\}\}$$

In some instances, we may be able to treat the partitioned sub-vectors uniformly, in which case we may want to combine them into a single set by taking the union of the subsets created by the partitioning,

$$\mathbf{D}_\pi = \{\pi(\mathbf{x}_0) \cup \ldots \cup \pi(\mathbf{x}_m)\},$$

but this will not be appropriate for all types of data. In cases where all features are interpreted the same way, this is generally fine, but if some features need to be interpreted differently, we cannot take the union this way. It also helps if the data is isotropic (i.e., changes are independent of direction), and spatial invariance is expected (i.e., a given pattern is equally likely to occur anywhere in the data vector). Natural image data tends to fit these assumptions fairly well, as do some types of time-series data, geological and meteorological data (for which this type of analysis is called *geostatistics*), and location-based medical, social, and economic datasets (e.g. for the analysis of cancer clusters).

The main advantage of taking the union is that we get a larger number of samples, which allows for better estimation. As an example, for natural image data, taking the union is generally safe, since we frequently want to create spatially invariant models anyway. In a more structured dataset, such as the MNIST dataset used in this paper, we do not expect spatial invariance in the data (because the images are always centered and surrounded by a white border, the location of the digit does not vary), so taking the union would be a tradeoff between having more samples, but making an (incorrect) assumption that they were all samples from a uniform distribution.

Within the partitioned dataset, we will need to keep track of which random variables in the original dataset map to which element(s) of the partitioned dataset. If we do so, we will then be in a position to describe which statistical quantities can still be calculated, and which cannot (because the variables involved are not in the same partition, and so will not be available to a learning algorithm at the same time).

So long as the partitioning function preserves locally adjacent blocks of the original vectors, it will be the case that we can still measure statistical properties of feature pairs that are "near" each other in the original vectors, but we will be unable to measure statistical properties of features that are "far apart" in the original vectors (here, we use Euclidean distance as the measure of how far apart two features are). In other words, spatially local statistical information is preserved, even while much of the non-local information is lost.

This emphasis on spatial locality means that we would be wise to draw from the field of spatial statistics. While some of the techniques used in geostatistics, for example, will not be relevant to all types of data, there are a number of principles and techniques that can productively be borrowed and used in our analysis of the impact of partitioning.

Probably the simplest, and most central, of these techniques is the *variogram*, which provides a way of measuring how important spatial distance is to statistical correlation. Given two samples from a spatial distribution Z, sampled at locations s_1 and s_2, $Z(s_1) - Z(s_2)$ is the difference between the *value* of those samples, and $s_1 - s_2$ is the *distance* between spatial locations at which those samples were taken. If there exists a $\gamma(\cdot)$ such that

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

where D is the set of all possible sampling locations, then $2\gamma(\cdot)$ is called a *variogram*; note that it is a function of the *distance* between sampling locations. For such a function to be properly defined, some fairly strong assumptions about the data must be made (including but not limited to it being sampled from a static, isotropic spatial distribution).

In geostatistics, variograms are often used for predictive modeling, which means that the assumptions must be fairly closely held. For our purposes, we will mainly use an empirical estimation of a variogram as a descriptive tool, so the fact that it would make a poor predictive model for image data, for example, is not a problem. Even if the assumptions are not perfectly met, an estimated variogram can still be a useful tool for testing the presence and extent of local structure in data, and it is in this capacity we will use them in our experiments.

A variogram can almost be thought of as a kind of histogram, in which the bins are inter-feature (i.e. sampling location) distances, and the height of each column is the mean of the variance of the (sample value) differences between all feature pairs that are that distance apart.

To generate a variogram plot like that in Figure 5, first, for every pair of features i, j, take the element-wise difference $X^i - X^j$, and then compute the variance of the resultant set $var(X^i - X^j)$. Next, for each distance d in the range $[0, d_{max}]$, compute the mean of all the feature-pair-variances between feature-pairs that are separated by distance d. These averages are then plotted against the distances.

We can make a similar histogram using correlations (or any other pair-wise statistical measure) between feature-pairs of a given distance. We must be careful, however, not to call such a histogram a *correlogram*, since that is another term from spatial statistics with a specialized meaning (which is close to, but not the same as, the correlation plots we use here).

For a more in depth treatment of variograms, correlograms, and spatial statistics in general, the reader is directed to Cressie's book on the subject [30].

IV. Experiments

We used the MNIST dataset for our experiments due to its wide use in evaluating RBMs as well as a variety of 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 samples; 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. See Figure 4 for some sample images. The MNIST dataset was introduced in [3], and can be obtained from [31].

We measure the performance of the RBMs using reconstruction error, which is defined to be the mean difference between the original and reconstructed images. We used a binary reconstruction error, using a fixed threshold value of 30 to map pixels in the range $[0 - 255]$ to a binary 1 or 0

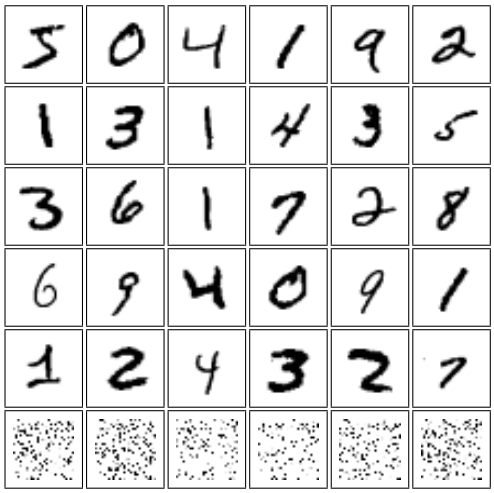

Fig. 4. Sample images from the MNIST dataset. The last row shows the randomly permuted images corresponding with the original images in the first row.

for the original images. To get the reconstructed image, we first sampled a hidden node activation vector from the RBM model for a given input image, and then sampled a visible node activation vector based on the sampled hidden activations. The resulting vector is also binarized and used in conjunction with the original vector to calculate reconstruction error for length n vectors:

$$E(\mathbf{x}, \mathbf{x}') = \frac{\sum_{i=0}^{n}(x_i \neq x_i')}{n}$$

To examine what statistical information is being exploited by the Partitioned RBMs, we performed a set of experiments on a randomly permuted version of the data set. To generate this new data set, a mapping was defined that assigned each element of an input vector to each element of the output vector with equal probability. The resultant mapping is 1-to-1 and onto, and is referred to as a random permutation. While the generation of the mapping is randomized, once a permutation has been generated its operation on input vectors is deterministic.

The permutation experiments were performed by generating a random permutation, applying it to each vector in the original dataset to generate a permuted dataset, and then training the RBMs on this permuted dataset. The result is that we can compare the performance of the RBMs using raw and permuted data to see whether or not the RBMs make use of any statistical information that is disrupted by the permutation. See the last row of Figure 4 for example permuted images.

We also generated a variogram and a mean-correlation plot for both the original dataset and for the permuted dataset to determine the presence and strength of local statistical structure in the two versions of the dataset.

V. Results

Table I shows the results of our RBM experiments. In the configuration column, Single RBM indicates the RBM was trained on the raw data vectors (i.e. no partitioning); RBM-n indicates a Partitioned-RBM with n partitions. RBM-1 is equivalent to Single RBM in terms of its configuration, but the RBM-1 is trained on fewer samples. Note that each successive RBM-n configuration starts with the output of the previous configuration, as described in Section II. The Samples column gives the number of training instances that were used to train the given RBM, selected at random from the total training set. As number of partitions decreases, we decrease training set size to match the time complexity of 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 split between training and test data).

TABLE I
Reconstruction Errors

Configuration	Samples	Original data: Error (%)	Permuted Data: Error (%)
Single RBM	60000	2.46	2.44
Single RBM	30000	2.55	2.55
RBM-28	60000	3.32	7.00
RBM-20	50000	2.20	6.42
RBM-15	40000	1.87	6.13
RBM-10	30000	1.64	5.00
RBM-5	25000	1.49	3.88
RBM-2	20000	1.44	2.89
RBM-1	30000	1.42	2.24

For the original MNIST dataset, the Partitioned-RBM outperforms the Single RBM not only for the RBM-1, but in all configurations except RBM-28. Additionally, when Single RBM is trained using the same reduced-size dataset as the final level of the Partition-RBM, its performance decreases even further. By design, the computational complexity of the full stack of Partitioned-RBMs is comparable to 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.

For the Partitioned-RBM, reconstruction error on the permuted dataset is significantly worse ($p > 99.99\%$ using a paired t-test) than the original dataset. Permutation has no statistically significant impact on the performance of the standard Single RBM, as we would expect.

Figure 5 shows the variograms and mean-correlation plots for both the original MNIST training data and for the randomly permuted version. To generate a variogram, we calculated $var(X^i - X^j)$ for each pair i, j, and then for each distance, plotted the mean of all the variances of feature pairs that distance apart. For the mean-correlation plots, we calculated $corr(X^i, X^j)$ for each pair i, j, and then for each distance, plotted the mean of the correlations of feature pairs that distance apart.

VI. Discussion

From the results of our experiments using the permuted dataset, we are led to the conclusion that the Partitioned-

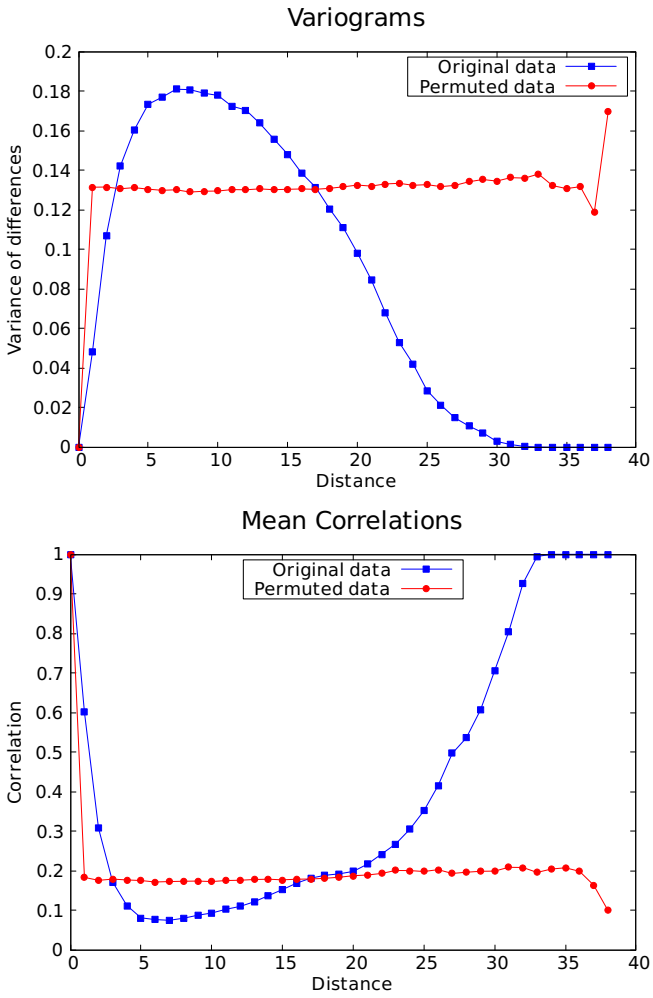

Fig. 5. Variogram and mean-correlation plots for the MNIST training set.

RBM is making use of statistical information that is spatially localized, where the Single RBM is not. The permutation results in no overall loss of information for the Single RBM; it simply re-orders the elements of the vectors. For this reason, any pair-wise correlation between a given pair of features will be unaltered by the permutation. The unchanged behavior of the Single RBM is therefore exactly what we would expect.

Things are different for the Partitioned-RBM, however, since the location of the two features in a given pair will be altered. This means that two features that would have been assigned to the same partition in the original dataset might *not* be assigned to the same partition in the permuted dataset. Since each piece of a Partitioned-RBM only has access to features in its partition, this means that whatever statistical information was contained in the correlation between this pair of features is no longer available to the Partitioned-RBM.

In fact, the Partitioned-RBM will always be cut off from a great many of the pair-wise feature correlations; the difference between the original and permuted datasets is simply in *which* correlations are lost. The fact that the Partitioned-RBM performs significantly worse on the permuted dataset implies that not all correlations are of equal value. In particular, it means that correlations between pairs of features that are spatially proximate in the original data are more important to the success of the algorithm than correlations between arbitrary pairs (which will, on average, be significantly farther apart in the original image).

In Figure 5, we can examine variogram and mean-correlation plots for the original and permuted datesets to draw much the same conclusion. At distance 0, there is of course no variance and perfect correlation, since any feature always has the same value as itself, regardless of the data. Beyond that, however, the variance and correlation plots for the permuted dataset are basically flat, indicating that after permutation there remains no significant relationship between statistical information and spatial distribution for feature pairs (the deviation from the trend of the last two points in each plot are caused by the reduced sample size available for the extreme distances; generating a new random permutation causes significant fluctuation in these last two values). This is the expected behavior for *any* dataset which has been randomly permuted, since the permutation is specifically created to make the spatial distribution uniformly random.

The plots for the original data, on the other hand, exhibit some interesting structure. The first thing to note is that adjacent features have high correlation, and the variance of the inter-pair differences is low. As the feature-pairs get farther apart, the degree of correlation drops off rapidly; by a distance of 5 pixels, there remains on average little statistical relationship between feature pairs. This is likely related to the average line-width of the hand-written digits, which is generally 2-3 pixels. The fact that correlation improves again as distances increase may seem counterintuitive; however, this is an artifact of the construction of the MNIST dataset. In particular, the MNIST images all have a centered digit surrounded by a white border. Since background pixels have the same value in *all* images, correlation scales with the likelihood that both members of a pair are background. The greater the distance between a pair of features, the more likely that both features will be a part of the background; in fact, beyond a distance of 30 (note that maximum distance is $28 \times \sqrt{2} \approx 40$) both pixels are *guaranteed* to be background pixels, meaning they are guaranteed to always have the same value. Note that most correlation measures are actually undefined when correspondence is perfect; for the purposes of Figure 5, we have set these values to 1.

Thus, the results lead us readily to the conclusion that the Partitioned-RBM is making use of spatially local statistical information in the MNIST dataset to achieve its performance. This supports our hypothesis [10] that partitioned deep learning algorithms rely on such local information.

VII. Conclusion and Future Work

We have begun to explore one of the potential representation biases that deep learning techniques leverage to achieve their performance. Our experimental results suggest that partitioned-data techniques are able to make use of

spatially local information, and we have in the variogram and the mean-correlation plot some crude tools for analyzing how much spatially local structure exists in a dataset. While these results are promising, they offer only a first jumping off point for the work that could be done in the area of understanding and exploiting the biases that underpin deep learning.

In particular, we need to apply the idea of partitioning to other problems and other techniques before we can make any claims of generality. Our previous work [10] applied a deep partitioning approach to Principal Component Analysis and achieved results that support the results presented here. However, there is still a great deal of space to explore, both in terms of existing deep learning techniques that make use of partitioning, and techniques that do not use partitioning but might be modified to do so.

The statistical underpinnings are also in need of further expansion and exploration. While the variogram is a useful tool, its utility is somewhat limited by the assumptions it makes, which are unrealistic for many types of data. Developing more sophisticated statistical models of spatially local structure that allow us to measure both the amount and the kind of spatial structure in data would allow us to check in advance whether a given dataset is a good candidate for partitioned learning. With sufficient resolution, such a model might even be used in a predictive or generative capacity; truly used as a model, rather than merely as a descriptive measure.

Finally, not all deep learning techniques do partitioning, and even the ones that do not are able to achieve impressive performance. This suggests that there must be other representation biases that underly the success of these methods, and efforts should be made to understand and explore these biases as well.

ACKNOWLEDGMENTS

We would like to thank Nathan Fortier, Shane Strasser, and Logan Perreault, who all contributed to discussions about the material covered in this paper.

REFERENCES

[1] T. Mitchell, "The need for biases in learning generalizations," *Technical Report CBM-TR-117*, 1980.

[2] 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.

[3] 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.

[4] 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.

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

[6] 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.

[7] 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.

[8] 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.

[9] 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.

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

[11] S. Behnke and R. Rojas, "Neural abstraction pyramid: a hierarchical image understanding architecture," *Proceedings of IJCNN '98*, vol. 2, pp. 820–825, 1998.

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

[13] W. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *Bulletin of Mathematical Biophysics*, vol. 5, pp. 115–133, 1943.

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

[17] 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.

[18] 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.

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

[20] 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.

[21] 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.

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

[23] 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.

[24] 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.

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

[26] 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.

[27] N. Fortier, J. W. Sheppard, and K. G. Pillai, "DOSI: Training artificial neural networks using overlapping swarm intelligence with local credit assignment," in *Joint 6th International Conference on Soft Computing and Intelligent Systems (SCIS) and 13th International Symposium on Advanced Intelligent Systems (ISIS)*. IEEE, 2012, pp. 1420–1425.

[28] N. Fortier, J. Sheppard, and S. Strasser, "Abductive inference in bayesian networks using overlapping swarm intelligence," *Soft Computing*, pp. 1–21, May 2014.

[29] ——, "Learning bayesian classifiers using overlapping swarm intelligence," in *Proceedings of the IEEE Swarm Intelligence Symposium*, December 2014, pp. 205–212.

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

[31] 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/