Noname manuscript No.
(will be inserted by the editor)

Learning Sparse Feature Representations using Probabilistic Quadtrees and Deep Belief Nets*

Saikat Basu · Manohar Karki · Sangram Ganguly · Robert DiBiano · Supratik Mukhopadhyay · Shreekant Gayaka · Rajgopal Kannan · Ramakrishna Nemani

Received: date / Accepted: date

Abstract Learning sparse feature representations is a useful instrument for solving an unsupervised learning problem. In this paper, we present three labeled handwritten digit datasets, collectively called n-MNIST by adding noise to the MNIST dataset, and three labeled datasets formed by adding noise to the offline Bangla numeral database. Then we propose a novel framework for the classification of handwritten digits that learns sparse representations using probabilistic quadtrees and Deep Belief Nets. On the MNIST, n-MNIST and noisy Bangla datasets, our framework shows promising results and outperforms traditional Deep Belief Networks.

* This is an extended version of the paper published in ESANN 2015.[1]

Saikat Basu, Manohar Karki, Supratik Mukhopadhyay, Rajgopal Kannan
Louisiana State University - Department of Computer Science
Baton Rouge, Louisiana 70803 - USA
E-mail: sbasu8@lsu.edu, mkarki6@lsu.edu, supratik@csc.lsu.edu, rkannan@bit.csc.lsu.edu

Robert DiBiano
Autopredictive Coding LLC
E-mail: robertdibiano@gmail.com

Sangram Ganguly
Bay Area Environmental Research Institute (BAERI)
/NASA Ames Research Center, Moffett Field, California 94035 - USA
E-mail: sangram.ganguly@nasa.gov

Shreekant Gayaka
Applied Materials
E-mail: gshreekant@gmail.com

Ramakrishna Nemani
NASA Advanced Supercomputing Division,
NASA Ames Research Center, Moffett Field, California 94035 - USA
E-mail: rama.nemani@nasa.gov

1 Introduction

Deep Learning has gained popularity over the last decade due to its ability to learn data representations in an unsupervised manner and generalize to unseen data samples using hierarchical representations. One of the popular unsupervised Deep learning models is the *Deep Belief Network*[2]. In [3], Deep Belief Networks have been used for modeling acoustic signals and have been shown to outperform traditional approaches using Gaussian Mixture Models for Automatic Speech Recognition (ASR). A greedy layerwise pre-training using Restricted Boltzmann Machine (RBM) is used to train Deep Belief Network. A sparse feature learning algorithm for Deep Belief Networks was proposed in [4]. However, their work was focused on maximization of information content in the learned representations. Restricted Boltzmann Machines, on the other hand, are trained by minimizing a contrastive term in the loss function. Deep Belief Networks have also been used for analyzing satellite imagery [5] as well as texture datasets [6].

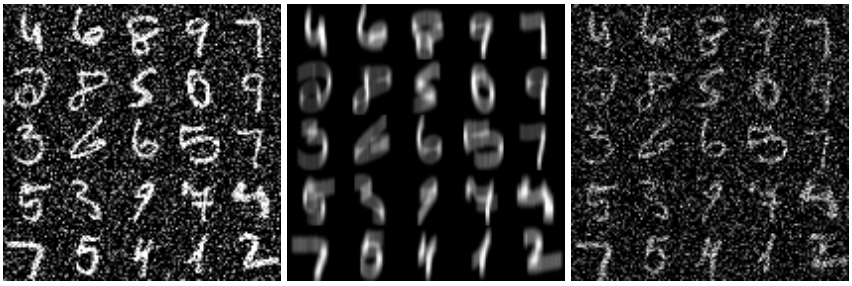
The main contributions of our work are twofold – (1) We first present three labeled handwritten digit datasets, collectively called n-MNIST, and three labeled Bangla numeral datasets, created by adding white gaussian noise, motion blur and reduced contrast to the original MNIST dataset[7] and the offline handwritten Bangla numeral dataset [8]. (2) Then, we present a framework for the classification of handwritten digits that a) learns probabilistic quadtrees from the dataset, b) performs a Depth First Search on the quadtrees to create sparse representations in the form of linear vectors, and c) feeds the linear vectors into a Deep Belief Network for classification. On the MNIST, n-MNIST and noisy Bangla datasets, our framework shows promising results and outperforms traditional Deep Belief Networks.

2 Datasets[1]

We evaluate our framework on the MNIST dataset[7] of handwritten digits as well as three artificial datasets collectively called n-MNIST (noisy MNIST) created by adding – (1) additive white gaussian noise, (2) motion blur and (3) a combination of additive white gaussian noise and reduced contrast to the MNIST dataset. A second set of experiments were performed on the offline Bangla numeral database [8]. The additive white gaussian noise was created by adding a noise with signal to noise ratio (SNR) of 9.5. It is a standard kind of noise. It is similar to noise caused by electrical variations in camera, television static, etc. This also emulates background clutter. The Motion Blur filter is a very common type of noise in images and videos. It is made to emulate a linear motion of the camera by τ pixels, with an angle of θ degrees. The filter becomes a vector for horizontal and vertical motions. We use a value of 5 pixels for τ and a value of 15 degrees in the counterclockwise direction for θ. For the noisy dataset with reduced contrast and AWGN, the contrast range was scaled down to half and

[1] The datasets are available at the web link [9] and [10] along with a detailed description of the methods and parameters used to create them

was applied with an Additive White Gaussian Noise with signal-to-noise ratio of 12. This emulates background clutter along with significant change in lighting conditions. Some of the images from the noisy MNIST datasets are shown in Figure 1 and those from the noisy Bangla dataset are shown in Figure 2.



(a) MNIST with Additive White Gaussian Noise (b) MNIST with Motion Blur (c) MNIST with AWGN and reduced contrast

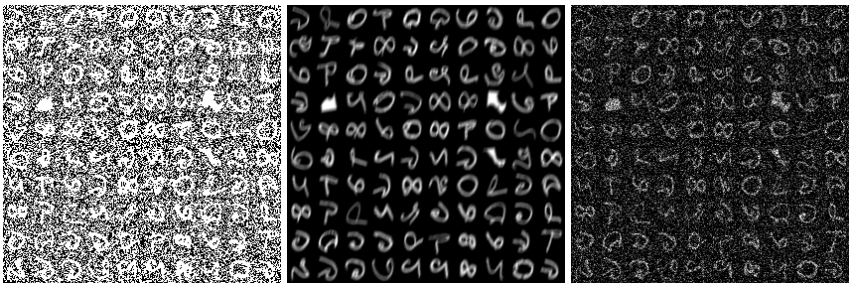
Fig. 1: Example images from the n-MNIST dataset created as part of the experiments

2.1 Pre-processing for the Bangla Dataset

The numerals in the Bangla dataset are first thresholded using a local adaptive mean filter. The thresholded images are then complemented and we extract the largest connected component. Then we find the center of mass of the largest connected component and the corresponding bounding box of the numeral and then use this information to center the image. The centered image is then padded with 10 pixels on all sides. Finally, the images are resized to 28×28 pixels.

2.1.1 Data Augmentation

Following the procedure defined in [8], we create a synthetic dataset by using rotation and blurring on the original Bangla dataset. For the rotation transformation, each sample is randomly rotated by an angle which lies between 5° and 10° and another between -10° and -5°. All the original and rotated training samples generated above are blurred by applying a Gaussian blurring kernel with mean, $\mu = 0.75$ and standard deviation $\sigma = 0.33$. So, the original images along with the rotated and blurred images form the final training dataset. These images are in turn corrupted with noise and form our noisy Bangla dataset.



(a) Bangla with Additive White Gaussian Noise (b) Bangla with Motion Blur (c) Bangla with AWGN and reduced contrast

Fig. 2: Example images from the noisy Bangla dataset created as part of the experiments

3 Probabilistic Quadtrees for Learning Sparse Representations

A quadtree takes the form of a tree in which all the internal nodes have four child nodes. Quadtrees were first proposed in [11] as a technique for storing key-value pairs indexed with composite keys. We propose a novel technique based on probabilistic quadtrees that can reduce the dimensionality of a dataset in a probabilistically sound way. We learn the structure of the quadtree from the samples of a dataset. A quadtree splits each image into four equal sized windows, and then performs a test of homogeneity on each image window. If a block satisfies the homogeneity criterion, it is not split any further into sub-windows. If otherwise, it does not satisfy the criterion, it is subdivided into four sub-windows, and the test condition is in turn tested on those smaller windows. This process is continued on all the sub-windows until each satisfies the homogeneity criterion. The resulting data structure can have windows of several different sizes. The homogeneity criterion can be defined as follows - Split a block if the difference beween the greatest and least values of the elements in a block exceeds a threshold τ. Threshold τ is set as a value between 0 and 1 (chosen here as 0.27 by experiments). Denoting the homogeneity criterion for sample d as H_d, this can be formally presented as follows:

$$H_d = \begin{cases} true, & \text{if} \max_{i \in d}(i) - \min_{i \in d}(i) \leq \tau \mid \tau \in [0,1] \\ false, & \text{if} \max_{i \in d}(i) - \min_{i \in d}(i) > \tau \mid \tau \in [0,1] \end{cases} \tag{1}$$

Alternatively, the homogeneity criterion can be considered proportionate to standard deviation of the probability distribution of the dataset. So, higher the standard deviation, higher the average texture of a block and higher is the probability of the block being divided into sub-blocks.

In the learned quadtree structure for a given dataset, a node is divided into smaller windows if the homogeneity criterion is not met for any sample in the

dataset. The node is not divided into smaller windows only if the homogeneity criterion is met by all samples in the dataset.

We can consider each node of the quadtree as a binary random variable X, which can take one of two values 1 or 0 based on whether it is divided into smaller windows or not. So, for a total of N samples in dataset D, the random variable X may assume one of $N+1$ split state values - one value for each sample not meeting the homogeneity criterion, and one value indicating that all samples meet the homogeneity criterion. This can be formally presented as follows:

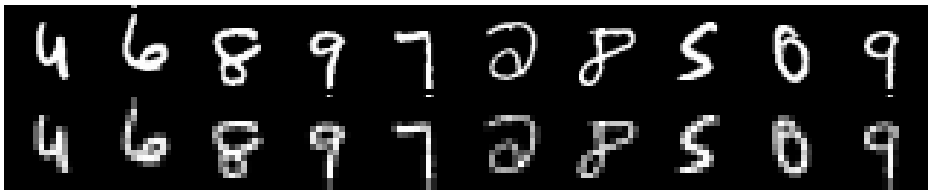
$$X = \begin{cases} 1, & \text{if } \exists d \in D \mid D = \{d_0, d_1, d_2, ..., d_N\} \cap \{H_d = false\} \\ 0, & \text{if } \forall d \in D \mid D = \{d_0, d_1, d_2, ..., d_N\} \cap \{H_d = true\} \end{cases} \tag{2}$$

Once learned, the probabilistic quadtree helps in reduction of the data dimensionality, which captures the statistics of the training samples in the dataset. A depth first search on the learned tree yields a linear vector that is then fed into an unsupervised learning framework. Some sample images from the MNIST and n-MNIST databases along with the images learnt by the probabilistic quadtrees are shown in Figure 3. A quadtree seems to be a natural choice for denoising image datasets like handwritten digits or letters which are inherrently sparse. The probabilistic quadtree creates a lower dimensional representation where uniform patches from the input image are eliminated while feeding it into the neural network thereby creating a sparse representation.

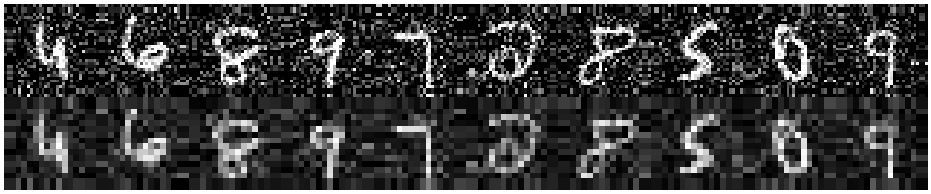
4 Deep Belief Network for Feature Learning

Deep Belief Network (DBN) consists of multiple layers of stochastic, latent variables trained using an unsupervised learning algorithm followed by a supervised learning phase using Feedforward Backpropagation Neural Networks. In the unsupervised pre-training stage, each layer is trained using a *Restricted Boltzmann Machine* (RBM). Once trained, the weights and biases of the Deep Belief Net are used to initialize the corresponding weights and biases of a Neural Network [12]. A Neural Network which has been initialized in this manner converges much faster to the optimal solution as compared to an uninitialized one. A DBN is a graphical model [13] where neurons of the hidden layer are conditionally independent of each other given a particular configuration of the visible layer and vice versa. A DBN can be trained layer-wise by iteratively maximizing the conditional probability of the input vectors or visible vectors given the hidden vectors and a particular set of layer weights. As shown in [2], this layer-wise training can help in improving the variational lower bound on the probability of the input training data, which in turn leads to an improvement of the overall generative model. We first provide a formal introduction to the Restricted Boltzmann Machine. The RBM can be denoted by the energy function:

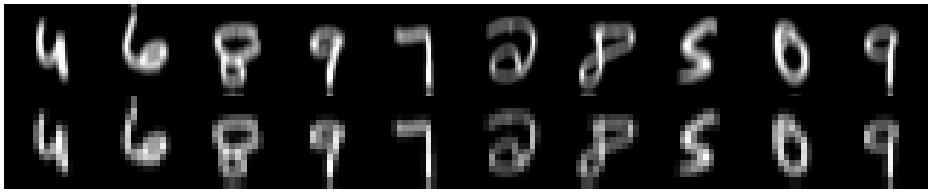
$$E(u, v) = -\sum_i a_i u_i - \sum_j b_j v_j - \sum_i \sum_j v_j w_{i,j} u_i \tag{3}$$



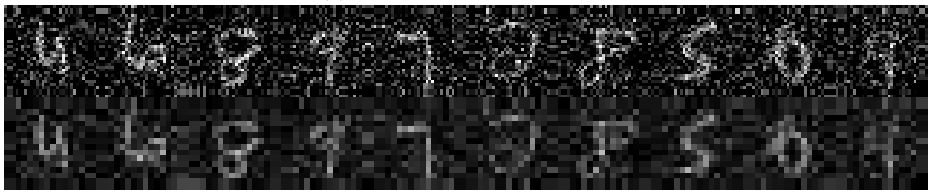
(a) Images from MNIST



(b) MNIST with Additive White Gaussian Noise



(c) MNIST with Motion Blur



(d) MNIST with AWGN and reduced contrast

Fig. 3: Example images from the MNIST and n-MNIST dataset and the representations learnt using the probabilistic quadtrees. The images in the top rows show the original images from the corresponding datasets and the bottom row shows representations learnt using the probabilistic quadtrees.

where, the RBM consists of a matrix of layer weights $W = (w_{i,j})$ between the hidden units v_j and the visible units u_i. The a_i and b_j are the bias weights for the visible units and the hidden units respectively. The RBM takes the structure of a bipartite graph and hence it only has inter-layer connections between the hidden or visible layer neurons but no intra-layer connections within the hidden or visible layers. So, the visible unit activations are mutually independent given a particular set of hidden unit activations and vice versa [14]. Hence, by setting either v or u constant, we can compute the conditional distribution of the other as follows:

$$P(v_j = 1|u) = \sigma(\sum_{i=1}^{p} w_{i,j}u_i + b_j) \tag{4}$$

$$P(u_i = 1|v) = \sigma(\sum_{j=1}^{q} w_{i,j}v_j + a_i) \tag{5}$$

where, σ denotes the log sigmoid function:

$$f(x) = \frac{1}{1 + e^{-x}} \tag{6}$$

The training algorithm maximizes the expected log probability assigned to the training dataset D. So if the training dataset D consists of the visible vectors u, then the objective function is as follows:

$$\underset{W}{\operatorname{argmax}} E\Big[\sum_{u \in U} \log P(u)\Big] \tag{7}$$

A Restricted Boltzmann Machine is trained using a *Contrastive Divergence* algorithm [14]. Once trained the DBN is used to initialize the weights of a feed-forward backpropagation neural network that is then used for classification. The neural network gives an estimate of the posterior probabilities of the class labels, given the input vectors. As illustrated in [15], the output of a neural network trained by minimizing the sum of squares error function approximates the conditional averages of the target data

$$y_k(x) = \langle t_k|x\rangle = \int t_k p(t_k|x)dt_k \tag{8}$$

Here, t_k are the set of target values that represent the class membership of the input vector x_k. For a binary classification problem, in order to map the outputs of the neural network to the posterior probabilities of the labeling, we use a single output y and a target coding that sets $t^n = 1$ if x^n is from class C_1 and $t^n = 0$ if x^n is from class C_2. The target distribution would then take the form

$$p(t_k|x) = \delta(t-1)P(C_1|x) + \delta(t)P(C_2|x) \tag{9}$$

Here, δ represents the Dirac delta function that satisfies the following conditions [16] $\delta(x) = 0$ if $x \neq 0$ and

$$\int_{-\infty}^{\infty} \delta(x)\,\mathrm{d}x = 1 \tag{10}$$

From 8 and 9, we get

$$y(x) = P(C_1|x) \tag{11}$$

So, the network output $y(x)$ represents the posterior probability of the input vector x having the class membership C_1 and the probability of the class membership C_2 is given by $P(C_2|x) = 1-y(x)$. This argument can easily be extended to multiple class labels for a generalized multi-class classification problem like MNIST.

The dimensionality reduction using probabilistic quadtrees helps improve the discriminative power of the DBN based classifier significantly.

4.0.2 Gibbs' Sampling in RBM

Gibbs' Sampling is a Markov Chain Monte Carlo method that is used to sample from a distribution when it is difficult to do direct sampling. Once sampled, these observations can be used to estimate the joint probability distribution or the marginal probability of a variable. In the context of an RBM, Gibbs' sampling is useful to estimate the log-likelihood gradient of the data. Generating samples from the model is equivalent to sampling from the DBN since the DBN is formed by stacking multiple layers of RBMs.

Gibbs' sampling in unrestricted Boltzmann Machines are computationally intensive because we need to perform sampling both for the input neurons and the joint distributions of the input and hidden neurons. Also for unrestricted Boltzmann machines, the number of Gibbs' sampler steps is equal to the number of units in the neural network. On the other hand, in an RBM, the hidden layer units are conditionally independent for a given configuration of the visible layer units and vice versa. So, the neurons can be updated in parallel and hence a Gibbs sampling in an RBM consists of only two sub-steps - 1) Sample hidden vectors given the visible vectors. Also, in RBM, the free energy state of the input neurons doesn't need to be sampled but can be directly calculated analytically.

4.0.3 Contrastive Divergence

The contrstive divergence algorithm approximates the the log-likelihood gradient of the data. This has been shown to be very useful for the training of Restricted Boltzmann machines. To obtain this approximation, the average over all possible samples is replaced with a single sample. With frequent updates in parameter values (after seeing one or a few training samples), there is automatic averaging going on which and this partially nullifies the effect of the increased variance introduced into the model due to the use of one or a few MCMC samples instead of using all the samples from the chain. However, this cancellation is only partial and hence there is still some extra variance introduced during the approximation. Running the full MCMC chain is computationally infeasible and hence we resort to k-step Contastive Divergence (CD-k). The CD-k algorithm performs another approximation. It runs the MCMC chain for only k steps. This reduces the computational cost greatly however leads to a higher bias in the model. The CD-k update rule can be noted as follows:

$$\triangle\theta = \frac{\partial E(v)}{\delta\theta} + \frac{\partial E(\tilde{v})}{\delta\theta} \tag{12}$$

where, E is the free energy and v is the observed sample and $\tilde{v}$ is the sample obtained after k-steps of MCMC. The bias tends to 0 as $k \to \infty$, since at $k = \infty$ the model distribution converges to the equilibrium distribution.

Another interesting observation about the CD-k algorithm is that when the distribution of the model approaches the equilibrium distribution, i.e., when $P \approx \tilde{P}$, then when we start the MCMC chain from a sample x, belonging to $\tilde{P}$ then the model is already good at approximating the underlying distribution and we need only one step to generate a sample of the unbiased estimator with underlying distribution P.

It has been shown that in the CD-k algorithm even $k = 1$ leads to good results. This has been shown both theoretically and empirically. The theoretical results are supported by the argument that the contrastive divergence algorithm approximates the first k terms of a series which provides an approximation of the log-likelihood gradient of the data. The contrastive divergence algorithm can be seen as approximating the gradient of the log-likelihood of the data centered around the training sample x_1. As we increase k, the underlying distribution of the reconstructed sample x_{k+1} moves further from X and closer towards the model distribution.

	MNIST		**n-MNIST with AWGN**	
Architecture (Neurons)	**Error DBN(%)**	**Error Ours(%)**	**Error DBN(%)**	**Error Ours(%)**
50-50	4.55	3.78	36.19	20.15
100-100	3.06	2.85	38.82	15.53
150-150	2.85	2.19	35.36	14.73
200-200	2.24	2.06	31.83	24.97
250-250	2.27	1.79	30.59	**12.77**
300-300	1.9	1.78	**27.42**	26.33
350-350	1.89	1.7	31.13	22.48
400-400	1.79	1.63	37.38	32.84
450-450	1.85	**1.43**	35.18	30.6
500-500	**1.78**	1.61	32.49	22.12

Table 1: Validation Error of a traditional DBN and our framework with various architectures on MNIST and n-MNIST with AWGN on a validation set sampled from the training set of MNIST

5 Results and Comparative Studies

Our quadtree based DBN framework was trained using various network architectures. The training details are listed in the following section.

	n-MNIST with Motion Blur		n-MNIST with AWGN and Reduced Contrast	
Architecture (Neurons)	**Error DBN(%)**	**Error Ours(%)**	**Error DBN(%)**	**Error Ours(%)**
50-50	14.55	8.39	30.78	15.57
100-100	12.72	6.66	26.1	12.29
150-150	10.32	5.86	26.4	11.52
200-200	11.13	5.21	25.23	11.39
250-250	**9.49**	5.03	25.16	11.08
300-300	11.47	4.56	24.83	10.72
350-350	12.82	4.69	25.42	10.73
400-400	12.74	4.35	23.77	**10.5**
450-450	12.28	4.61	24.28	10.98
500-500	13.57	**4.29**	**20.03**	10.71

Table 2: Validation Error of a traditional DBN and our framework with various architectures on n-MNIST with Motion Blur; and with AWGN and Reduced Contrast on a validation set sampled from the training set of MNIST

MNIST		n-MNIST with AWGN	
Error DBN(%)	**Error Ours(%)**	**Error DBN(%)**	**Error Ours(%)**
1.86	1.38	12.79	9.93

Table 3: Test Error of a traditional DBN and our framework with various architectures on MNIST and n-MNIST with AWGN

n-MNIST with Motion Blur		n-MNIST with AWGN and Reduced Contrast	
Error DBN(%)	**Error Ours(%)**	**Error DBN(%)**	**Error Ours(%)**
3.50	2.60	9.43	7.84

Table 4: Test Error of a traditional DBN and our framework with various architectures on n-MNIST with Motion Blur; and with AWGN and Reduced Contrast

5.1 Training Details

While training the DBN models, the learning rate was set as 0.1 while the mini batch size was set as 100. The momentum was set as 0. The learning rate was updated according to an inverse function with gamma parameter set as 0.998. We also used a Dropout fraction of 0.2 for regularization of the networks. Dropout is a useful technique proposed in [17], [18] for the regularization of Deep Neural Networks. Once trained, the DBN weights are used to initialize the weights of the Neural Network. For the supervised neural network training, we again use an initial learning rate of 0.1 and an inverse learning rate update policy with gamma value as 0.998 with the update applied in each iteration. The mini batch size is again chosen as 100.

	Noisy Bangla with AWGN	
Architecture (Neurons)	**Error DBN(%)**	**Error Ours(%)**
50-50	12.03	11.89
100-100	8.43	8.17
150-150	7.24	7.08
200-200	7.67	7.29
250-250	7.21	7.19
300-300	7.41	7.18
350-350	7.57	7.31
400-400	**7.11**	**7.02**
450-450	7.71	7.32
500-500	7.5	7.46

Table 5: Validation Error of a traditional DBN and our framework with various architectures on the noisy Bangla dataset with AWGN on a validation set sampled from the training set of the Bangla numeral dataset

	Noisy Bangla with Motion Blur		Noisy Bangla with AWGN and Reduced Contrast	
Architecture (Neurons)	**Error DBN(%)**	**Error Ours(%)**	**Error DBN(%)**	**Error Ours(%)**
50-50	7.2	7.03	25.24	16.17
100-100	3.78	3.37	21.71	13.48
150-150	2.91	2.53	19.99	**12.6**
200-200	2.77	2.6	19.53	12.97
250-250	2.6	2.15	19.12	12.82
300-300	2.53	2.39	19.65	12.78
350-350	2.24	2.22	**18.85**	12.91
400-400	2.2	2.13	19.8	13.34
450-450	**2.14**	2.1	19.18	13.29
500-500	2.35	**1.97**	19.21	13.17

Table 6: Validation Error of a traditional DBN and our framework with various architectures on noisy Bangla dataset with Motion Blur; and with AWGN and Reduced Contrast on a validation set sampled from the training set of the Bangla numeral dataset

Noisy Bangla with AWGN	
Error DBN(%)	**Error Ours(%)**
8.91	8.66

Table 7: Test Error of a traditional DBN and our framework with various architectures on the noisy Bangla dataset with AWGN

5.2 Experimental Results

Various network architectures along with the validation set error for the traditional DBN framework and the probabilistic quadtree based framework on the MNIST and the three n-MNIST datasets are listed in Tables 1 and 2. The validation set was sampled from the training set by randomly shuffling the samples

Noisy Bangla with Motion Blur		Noisy Bangla with AWGN and Reduced Contrast	
Error DBN(%)	**Error Ours(%)**	**Error DBN(%)**	**Error Ours(%)**
7.91	7.34	16.92	12.69

Table 8: Test Error of a traditional DBN and our framework with various architectures on noisy Bangla dataset with Motion Blur; and with AWGN and Reduced Contrast

Dataset	Mean Normalized Contrast
MNIST	1.0
Offline Bangla	0.87

Table 9: Mean contrast of the MNIST and Bangla numeral datasets.

MNIST		n-MNIST with AWGN	
Test Error Raw pixels(%)	**Test Error Ours(%)**	**Test Error Raw pixels(%)**	**Test Error Ours(%)**
11.03	8.21	44.71	34.97

Table 10: Test Error of a Neural Network with raw pixels and our probabilistic quadtree based framework with various architectures on MNIST and n-MNIST with AWGN

n-MNIST with Motion Blur		n-MNIST with AWGN and Reduced Contrast	
Test Error Raw pixels(%)	**Test Error Ours(%)**	**Test Error Raw pixels(%)**	**Test Error Ours(%)**
89.72	11.82	61.03	18.66

Table 11: Test Error of a Neural Network with raw pixels and our probabilistic quadtree based framework with various architectures on n-MNIST with Motion Blur; and with AWGN and Reduced Contrast

Noisy Bangla with AWGN	
Test Error Raw pixels(%)	**Test Error Ours(%)**
38.26	23.43

Table 12: Test Error of a Neural Network with raw pixels and our probabilistic quadtree based framework with various architectures on the noisy Bangla dataset with AWGN

and from the original set of 60,000 training samples, 50,000 samples were chosen as the training set and 10,000 were chosen as the validation set. Using the results from these validation sets, we obtained the final test error on the original test

Noisy Bangla with Motion Blur		Noisy Bangla with AWGN and Reduced Contrast	
Test Error Raw pixels(%)	**Test Error Ours(%)**	**Test Error Raw pixels(%)**	**Test Error Ours(%)**
60.99	23.54	57.04	48.43

Table 13: Test Error of a Neural Network with raw pixels and our probabilistic quadtree based framework with various architectures on noisy Bangla dataset with Motion Blur; and with AWGN and Reduced Contrast

MNIST	MNIST with AWGN	MNIST with Motion Blur	MNIST with AWGN and Reduced Contrast
212	244	211	244

Bangla	Bangla with AWGN	Bangla with Motion Blur	Bangla with AWGN and Reduced Contrast
203	244	202	244

Table 14: Size of the feature vectors learnt using the probabilitic quadtrees from the various datasets on an image of size 28×28.

set of 10,000 samples. The results are presented in Table 3[2] and Table 4. From the Tables, it is evident that our best performing network outperforms the best traditional Deep Belief Network on both the MNIST and n-MNIST datasets. On the MNIST dataset, our best network exhibits a relative improvement of ∼25% over the traditional DBN. For the n-MNIST dataset, it provides a relative improvement of ∼69% for Additive White Gaussian Noise (AWGN), ∼26% for Motion Blur and ∼16% for AWGN and Reduced contrast. Similar to the experiments on n-MNIST, the noisy Bangla dataset was divided into training and validation sets. The errors on the validation set are reported in Table 5 and Table 6. Table 7 and Table 8 show the test error rates of the traditional DBN based framework and the quadtree based DBN framework on the noisy Bangla datasets. As seen in the tables, the best quadtree based DBN framework provides a relative improvement of ∼3% over the best traditional DBN architecture for the noisy Bangla dataset with AWGN. For the motion blur dataset, it produces a relative improvement of ∼8% while for the noisy Bangla dataset with AWGN and reduced contrast, it produces a relative improvement of ∼33%. It is interesting to observe that the relative improvement in performance using the quadtree based framework is highest for n-MNIST with AWGN and lowest for n-MNIST with AWGN and reduced contrast. On the contrary, for the noisy Bangla dataset, the relative improvement is the highest for the dataset with AWGN and reduced contrast and lowest for the AWGN dataset. This can be attributed to the fact that the Bangla dataset generated by the pre-processing and data augmentation stages has a lower contrast than the MNIST dataset as seen in Table 9 and hence the application of reduced contrast on top of the already low contrast dataset creates a noisy dataset which is much more dif-

[2] Our test error on MNIST is different from that in [17] because the authors use a different architecture. This is true both for their Neural Network and DBN architectures.

ficult to be handled by the traditional DBN. Hence, the quadtree based DBN produces significant improvement over the traditional DBN for the noisy Bangla dataset with AWGN and reduced contrast. Due to this same reason, the mean test error rates of the various architectures of both the traditional DBN and the quadtree based DBN on the reduced contrast-AWGN dataset are higher than the AWGN dataset and the motion blur dataset. Next, we show the neural network baselines for raw pixels and the features extracted by our quadtree based framework on the various datasets. Table 10, 11, 12 and 13 show the error rates of a neural network with a) raw pixels and b) features extracted from the probabilistic quadtrees as input. From these tables we can see that the Neural Network produces higher error rates for all the datasets. This highlights the need for an unsupervised pretraining stage for reaching the global error basin.Table 14 shows the dimensionality of the feature vectors learnt using the probabilistic quadtrees on 28×28 images from the various datasets. It can be seen that on an average, the probabilistic quadtrees are able to reduce the dimensionality of the input vectors to nearly one-third of their original size.

6 Discussion and Future Directions

Our learning framework based on probabilistic quadtrees outperforms traditional Deep Belief Networks on MNIST, n-MNIST and the noisy Bangla datasets. Probabilistic quadtrees help in generating sparse representations for the datasets and improve the discriminative power of the framework.

We plan to investigate the use of various pooling techniques like SPM [19] as well as certain sparse representations like sparse coding [20] to handle the noisy datasets. Hierarchical representations like Convolutional DBN [21] are other useful candidates for investigation. We believe that the noisy versions of MNIST and Bangla datasets will help researchers better apply and extend the research on understanding representations for noisy object recognition datasets.

Acknowledgment

The project is supported by Army Research Office (ARO) under Grant #W911-NF1010495 and NASA Carbon Monitoring System through Grant #NNH14ZDA001NCMS. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the ARO or the United States Government. We are thankful to the Computer Vision and Pattern Recognition unit at Indian Statistical Institute, Kolkata, India for making the offline Bangla handwritten numeral dataset available to us.

References

1. Saikat Basu, Manohar Karki, Sangram Ganguly, Robert DiBiano, Supratik Mukhopadhyay, and Ramakrishna Nemani. Learning sparse feature representations using probabilistic quadtrees and deep belief nets. In *Proceedings of the European Symposium on Artificial Neural Networks, ESANN*, 2015.
2. Geoffrey E. Hinton and Simon Osindero. A fast learning algorithm for deep belief nets. *Neural Computation*, 18:2006, 2006.
3. Abdel-rahman Mohamed, George E. Dahl, and Geoffrey E. Hinton. Acoustic modeling using deep belief networks. *IEEE Transactions on Audio, Speech & Language Processing*, 20(1):14–22, 2012.
4. Marc'Aurelio Ranzato, Y-lan Boureau, and Yann Lecun. Sparse Feature Learning for Deep Belief Networks. In *Advances in Neural Information Processing Systems*, 2008.
5. Saikat Basu, Sangram Ganguly, Supratik Mukhopadhyay, Robert DiBiano, Manohar Karki, and Ramakrishna Nemani. Deepsat: A learning framework for satellite imagery. In *Proceedings of the 23rd SIGSPATIAL International Conference on Advances in Geographic Information Systems*, GIS '15, pages 37:1–37:10, New York, NY, USA, 2015. ACM.
6. Saikat Basu, Manohar Karki, Robert DiBiano, Supratik Mukhopadhyay, Sangram Ganguly, Ramakrishna Nemani, and Shreekant Gayaka. A theoretical analysis of deep neural networks for texture classification. *arXiv preprint arXiv:1605.02699*, 2016.
7. WWW. Mnist. `http://yann.lecun.com/exdb/mnist/`.
8. Ujjwal Bhattacharya and Bidyut B Chaudhuri. Handwritten numeral databases of indian scripts and multistage recognition of mixed numerals. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 31(3):444–457, 2009.
9. WWW. n-mnist. `http://csc.lsu.edu/~saikat/n-mnist/`.
10. WWW. Noisy bangla. `http://csc.lsu.edu/~saikat/noisy-bangla/`.
11. R. A. Finkel and J. L. Bentley. Quad trees a data structure for retrieval on composite keys. *Acta Informatica*, 4(1):1–9, 1974.
12. Yoshua Bengio. Learning deep architectures for ai. *Found. Trends Mach. Learn.*, 2(1):1–127, January 2009.
13. Daphne Koller and Nir Friedman. *Probabilistic Graphical Models: Principles and Techniques - Adaptive Computation and Machine Learning*. The MIT Press, 2009.
14. Miguel A. Carreira-Perpinan and Geoffrey E. Hinton. On contrastive divergence learning. 2005.
15. Christopher M. Bishop. *Neural Networks for Pattern Recognition*. Oxford University Press, Inc., 1995.
16. Sadri Hassani. Dirac delta function. In *Mathematical Methods*, pages 139–170. Springer, 2009.
17. Nitish Srivastava, Geoffrey E Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks

from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958, 2014.
18. Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *CoRR*, abs/1207.0580, 2012.
19. Svetlana Lazebnik, Cordelia Schmid, and Jean Ponce. Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories. CVPR '06, pages 2169–2178, 2006.
20. Honglak Lee, Alexis Battle, Rajat Raina, and Andrew Y. Ng. Efficient sparse coding algorithms. In *NIPS*, pages 801–808. NIPS, 2007.
21. Honglak Lee, Roger Grosse, Rajesh Ranganath, and Andrew Y. Ng. Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. ICML '09, pages 609–616, 2009.