COGS 260: Assignment 2

Kriti Aggarwal (PID: A53214465)
Department of Computer Science and Engineering
University of California, San Diego
San Diego, CA 92093
kriti@eng.ucsd.edu

Abstract

Image classification is the task of assigning an input image one label from a fixed set of categories. Though this task is trivial for humans, it's quite challenging for computers to solve. Through this assignment we attempt to classify MNIST digits using various supervised and unsupervised learning techniques for image classification including nearest neighbor, SVM, convolutional neural networks, deep belief networks and spatial pyramid matching. We perform several experiments with the parameters, optimization techniques, network architecture, etc. of these methods to analyze and critique their performance.

1 Description of the methods and Experimental Results

1.1 Instance based learning: Nearest Neighbor

K-nearest neighbors algorithm (k-NN) belongs to the category of non-parametric classifiers which assume that data which are close together based upon some metrics, such as Euclidean distance etc., more likely belong to the same category. Therefore, for classifying a test image to one of the data labels, distance of the test image is computed from all the training images. The k closest neighbors are then selected and the label of the training label with maximum votes is predicted. Nearest neighbor is the simplest case of k-nearest neighbor algorithm in which we only take one closest neighbor.

Nearest neighbor method has several key advantages, as easy implementation, competitive performance, and a nonparametric computational basis which is independent of the underlying data distribution. However, the computational cost at test time is high, since classifying a test example requires a comparison to every single training example. Also, k-nearest neighbor is rarely appropriate for use in practical high-dimensional image classification settings, since distances over high-dimensional spaces can be very counter-intuitive.

For the experiments below, we used a normalized feature vector of size 784 for every image. Here are the experiments, we performed.

1.1.1 Distance metric

The accuracy of nearest neighbor classification depends significantly on the metric used to compute distances between different examples. We experimented with various distance metrics viz a viz L1, L2 and L3 norm in this assignment and found that L2 norm gave the maximum accuracy in test dataset. Figure 1 shows how the accuracy varies across different distance metrics as the number of training samples is increased from 10000 to 20000, while the test set is constant of 1000 images. Accuracy increases as the number of training samples is increased. L2 norm performs slightly worse initially but after 16000 training samples performs the best, L1 norm just below it and L3 norm performing the worst.

We found that the L2 norm performs the best for this case which can be attributed to the fact that L2 norm tries to get the contribution of all the features while L1 norm acts as a feature selector. Since, we used PCA to find the features capturing the most variance, L1 norm would further reduce the features making the overall accuracy lower while L2 gives every feature equal importance and hence performs better than L1 norm.

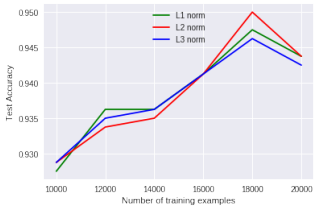

Figure 1: Accuracy vs various distance metrics

1.1.2 Dimensionality reduction using PCA

The complexity of computing a nearest neighbor for one test image is O(N*d) where N is the size of training set and d is the dimension of one test image feature vector. For MNIST dataset, the dimension of a feature vector is 784 and N is 60000. Hence, the process of testing is highly computation intensive. To combat with this challenge, we experimented with reducing the size of feature vector by doing principal component analysis. We experimented with the number of principal components in the image data from 1-100 as shown in figure 2. We used 60000 images for training, 1000 images for testing and L2 norm as the distance metric.

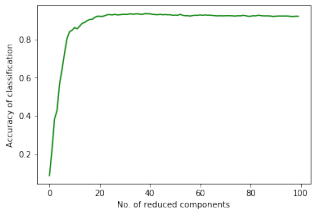

Figure 2: Accuracy vs number of principal components

The accuracy on test set starts to converge as the number of principal components reaches 16. Though we achieved a maximum accuracy somewhere when number of principal components is between 20-50 and the accuracy starts to dip after that. To find the exact number of principal components where we achieved maximum accuracy, we chose a fine range of principal components from 16-60 as shown in figure 3.

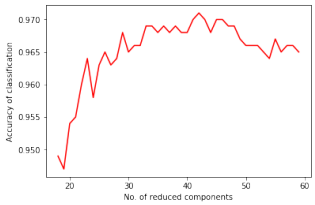

Figure 3: Accuracy vs number of principal components

We found that the accuracy suffered slight fluctuations and achieved the maximum accuracy of 97% when the number of principal components is 43. This shows that the the initial feature vector of size 784 contained highly correlated features with less than 43 features capturing most of the variance in

data. As the dimensions are increased after that, the noise/anomalies unique to training data are also captured by PCA resulting in overfitting.

1.1.3 Increasing the number of neighbors

Nearest neighbor is highly prone to noise as we predict only according to the highest voted neighbor. Figure 4 shows the confusion matrix when the number of components taken is 43. The confusion

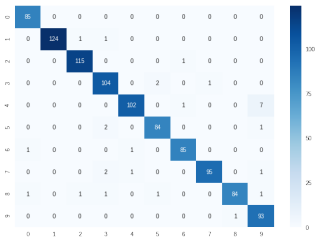

Figure 4: confusion matrix with number of components 43

matrix shows many outliers such as 8 being confused as 0, 2,3,5 or 9, 7 being confused as 3, 4 or 9. 0 is correctly predicted while 8 is the most confused digit. The confusion can be because of the similarity in the structure of the digits.

Nearest neighbor might be made resilient to such outliers by using k-Nearest neighbor where k¿1 which would help in predicting the correct class since more than 1 nearest neighbor vote for the predicted label. Though, when we increased the number of nearest neighbors, we failed to get any improvement in the test accuracy above 97% probably because of the high number of classes.

1.2 Support Vector Machines(SVMs)

Support Vector Machine (SVM) is a supervised machine learning algorithm which maximizes the margin around the separating hyperplane of the data points. The name of support vector machines comes from 'Support vectors' which are the data points that lie closest to the decision surface, making them the most difficult to classify. Shifting the support vectors shifts the decision boundary, hence support vectors can be viewed as the prototype of the data points. These support vector prototype selection can also combined with nearest neighbor algorithm. SVM can thus be compared to Nearest Neighbor where every test example is compared to every test set example, while in SVM a test example is only compared to the support vectors.

This also presents a possibility of combining the two techniques, by selecting the prototype of data points using Support vectors and then using nearest neighbor for classification.

In the following sections, we list the various experiments performed by us. The computation complexity of SVM increases with the number of training set and the number of features. Hence, in order to run all the experiments in due time, we used PCA to project the data samples in a 43 dimensional space(taking the results from the Nearest Neighbor experiments).

When trained on 60,000 training set and 10,000 test set, we got an accuracy of 98.41%. We used rbf kernel, C=10^6 and gamma = $6x10^{-3}$ for this experiment. Figure 5 shows the confusion matrix we got using SVM.

Comparing the confusion matrix of SVM to that of Nearest neighbor we find that the network has learnt to distinguish between many similar digits. For eg. the network can easily distinguish between the digits 4 and 9. The network has also learnt to distinguish 8 from other similar digits like 0,3,5 etc. Though the model still fails to distinguish very similar digit pairs like 8 and 9, 9 and 7 etc.

1.2.1 Kernel functions

A powerful insight using SVM is that the linear SVM can be rephrased using the inner product of any two given observations, rather than the observations themselves. This is leveraged by using a kernel

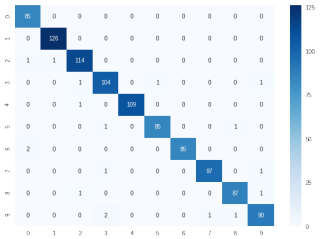

Figure 5: Confusion matrix for SVM with rbf kernel

in SVM which is essentially a similarity function. For points that are not linear separable , SVM projects them in a higher dimension to gain linear separation in the new dimension. To transform the data points in higher dimension we use a non linear kernels like radial basis function(gaussian), polynomial kernels etc.

We explored linear, polynomial and RBF kernel support vector machines (SVM) classifiers, to find out what works best on the MNIST dataset. The following table summarizes the results we obtained:

Kernel fun6tion	Test accuracy
Linear	96.6%
RBF with gamma10^{-4}	98.31%
RBF with gamma10^{-3}	98.41
RBF with gamma10^{-2}	98.2
Poly with degree 3	98.21%
Poly with degree 4	96.9%
Sigmoid	98.01%

- *Influence of degree in poly kernel* The accuracy on testset of poly kernel decreased with increase in degree of the polynomial which might be due to overfitting.

- *Influence of gamma parameter on rbf:* A small gamma corresponds to a Gaussian with a large variance i.e. if x_j is a support vector, a small gamma implies the class of this support vector will have influence on deciding the class of the vector x_i even if the distance between them is large. If gamma is large, then variance is small implying the support vector does not have wide-spread influence. Hence, large gamma leads to high bias and low variance models, and vice-versa.

  We ran grid search for gamma parameter with the range 10^{-7}, 100 and found that we obtained the optimal results for gamma = 10^{-3}. We observed that test accuracy fails when the gamma value is decreased or increased from 10^{-3}.

We found that the linear SVM was the most inefficient in terms of both time computation and accuracy. The accuracy on testset of poly kernel decreased with increase in degree of the polynomial which might be due to overfitting. Out of the non linear kernels, rbf outperformed all others.

1.2.2 The parameter C

A standard SVM seeks to find a margin that separates all positive and negative examples. However, this can lead to poorly fit models if any examples are mislabeled or extremely unusual. To account for this, we use "soft margin" SVM that allows some examples to be ignored or placed on the wrong side of the margin; leading to a better overall fit. C is the parameter for the soft margin cost function, which controls the influence of each individual support vector; this process involves trading error penalty for stability.

A high value of C corresponds to low error by giving the model an opportunity to consider more support vectors. A low value of C corresponds to a smoother decision boundary but more misclassifications.

We experimented with various values of C by using grid size search on the range [10,10^9] and found that C = 10^6 gave the maximum accuracy of 98.3% for 60000/10000 train and test split.

1.3 Convolution Neural Networks

ConvNets derive their name from the convolution operator. The primary purpose of Convolution in case of a ConvNet is to extract features from the input image. Convolution preserves the spatial relationship between pixels by learning image features using small squares of input data. Here are the basic layers that comprise a convolution neural network:

- *Convolution layer*: Convolutional layers consist of a rectangular grid of neurons. It requires that the previous layer also be a rectangular grid of neurons. Each neuron takes inputs from a rectangular section of the previous layer; the weights for this rectangular section are the same for each neuron in the convolutional layer.
- *Pooling layer*: Pooling layer is to progressively reduce the spatial size of the representation to reduce the amount of parameters and computation in the network, and hence to also control overfitting. In case of Max Pooling, we define a spatial neighborhood (for example, a 2x2 window) and take the largest element/average of all elements from the rectified feature map within that window. Pooling layer also makes the network invariant to small transformations, distortions and translations in the input image by representing a window of pixel by only a single value.
- *Fully connected Layer*: A fully connected layer takes all neurons in the previous layer and connects it to every single neuron it has.

1.3.1 The structure of the Convolutional Neural Network adopted in this assignment

- Convolutional layer with 30 feature maps of size 5x5.
- Pooling layer taking the max over 2*2 patches.
- Convolutional layer with 15 feature maps of size 3x3.
- Pooling layer taking the max over 2*2 patches.
- Dropout layer with a probability of 20%.
- Flatten layer.
- Fully connected layer with 128 neurons and rectifier activation.
- Fully connected layer with 50 neurons and rectifier activation.
- Output layer.

The network architecture is inspired by Lenet in which every convolutional layer is followed by a max pooling layer with 2 fully connected layers at the end. Though we made various modifications in our architecture by using ReLU units instead of tanh, additional dropout layer, different number of covolutional layers, size of filters etc.

We plotted the test and training loss on the dataset and got the results as shown in figure 6. The loss on training set keeps on decreasing while for test set the loss decreases initially but after 23 epochs, the loss on testing set consistently increases. This shows that initially the model is underfit but after 23 epochs, the model starts to overfit on the training data.

The following plot 7 shows the test vs training accuracy at various epochs. The training accuracy keeps on increasing with more epochs, while the test accuracy increases initially and then decreases as the model starts to overfits. With the tuned parameters, we get a maximum accuracy of 99.39% in 23 epochs.

Below are the experiments that we performed to fiddle with the network architecture, hyperparameters and various optimization techniques. We used 60,000 images for training and 10,000 images for testing set.

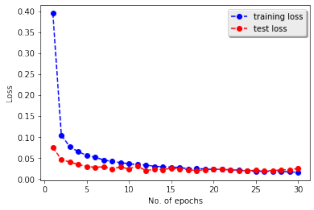

Figure 6: epochs vs loss

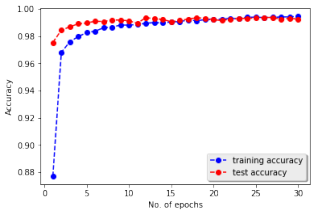

Figure 7: epochs vs accuracy

1.3.2 Network architectures

- *LeNets* LeNet introduced the convolutional networks We implemented LeNet is reaching 98.49% classification accuracy on the MNIST dataset. In general they consist of a convolutional layer followed by a pooling layer, another convolution layer followed by a pooling layer, and then two fully connected layers similar to the conventional multilayer perceptrons.

  We made various modifications to LeNet to achieve higher as described in the following sections.

- *ResNets* A Residual Network, or ResNet is a neural network architecture which solves the problem of vanishing gradients by providing the network with a shortcut at each layer to send the gradient to back layers. This is made possible by changing the activation at a layer as follows: y = f(x) + x

  We trained a 8-layer Resnet with 2 convolution layers on MNIST dataset and got an accuracy of 99.49%

1.3.3 Hyper-parameters

- *Learning rate:* The performance of stochastic gradient descent (SGD) depends critically on how learning rates are tuned and decreased over time. We experimented with various learning rates and found that 10^{-2} gave the most efficient results. Learning rate more than 10^{-2} causes SGD to diverge and less than 10^{-2} leads to very slow convergence.

- *Dropout rate:* Deep neural nets with a large number of parameters are very powerful machine learning systems. However, overfitting is a serious problem in such networks. Dropout is one of the techniques for addressing this problem. The key idea is to randomly drop units (along with their connections) from the neural network during training. This prevents units from co-adapting too much.

  The convolutional neural network we created had only 2 convolutional layers, still we found that dropping some units affected the test accuracy. This shows that the network was able to overfit the training set in a 2 layer convolutional network and dropout is a powerful regularization tool.

  We experimented with 3 dropout rates, 0.0 dropout, 0.3 dropout and 0.5 dropout rate and found that the system performed the best with a dropout rate of 0.3.

  The following table shows how the test accuracy changed with dropout rates:

Dropout rate	Test accuracy
0.0	99%
0.3	99.39%
0.5	99.11%

- *Depth of the Network:* To understand the importance of depth or number of layers in convolutional neural network, we conducted several experiments. We observed that removing one of the fully connected layer only slightly affects the accuracy and loss. Removing one of the convolutional layers also makes only a small change in the test accuracy and loss. Although, removing both the convolutional layer and the fully connected layer, causes the accuracy to drop significantly. We conclude that the depth of the network is paramount for optimal performance of the network. Figure 8 shows the accuracy plots with various layers removed.

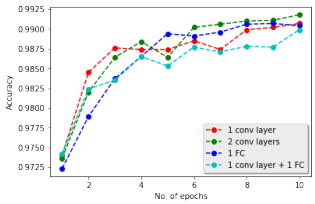

Figure 8: Layers vs test accuracies

We also observed that adding an additional convolutional layer with a bigger filter size and adding more hidden units in the fully connected layer degraded the accuracy due to overfitting. Figure 9 shows how adding an additional convolutional layer decreases the test accuracy.

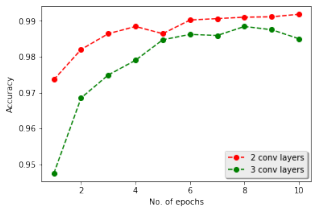

Figure 9: Adding additional convolutional layer vs test accuracies

- *Convolutional layer filter size:* For the experiments on filter sizes, we gradually increased the filter sizes and monitored the error rate to see how it is varying.

  We started experimenting with a small filter size of 3x3 and then gradually increased its size and monitored the error rate to see how it is varying. We found that the optimal values of filter sizes which gave the optimal results is a combination of 7x7 and 5x5 filter for the 2 convolutional layers. The accuracies obtained with various filter sizes is shown in the figure 10.

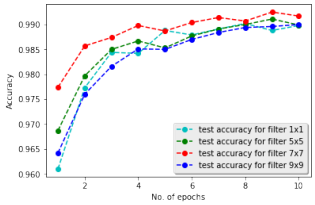

Figure 10: Various filter sizes vs test accuracies

Very small and very big filter sizes, both gave worse results. This might be because very small filter sizes capture very fine details of the image. On the other hand a bigger filter size leaves out minute details in the image. Hence, the actual filter sizes depend on the type of data classified and can be empirically determined.

- *Activation function:* We experimented with using various nonlinear functions like tanh, sigmoid and ReLU for activations, and found out that ReLU layers work far better. Using ReLu layers, we are able to train the network a lot faster with a slightly improved accuracy(~04%). This can be attributed to the fact that ReLU units helps to alleviate the vanishing gradient problem, which is the issue where the lower layers of the network train very slowly because the gradient decreases exponentially through the layers.

1.3.4 Optimization techniques

- *Nesterov Momentum:* Momentum is a method that helps accelerate SGD in the relevant direction and dampens oscillations. Nesterov Momentum is a slightly different version of the momentum update in which we maintain a running average of the past momentums.
- *AdaGrad:* Adagrad is an algoithm for gradient-based optimization that adapts the learning rate to the parameters, performing larger updates for infrequent and smaller updates for frequent parameters. But Adagrad's main weakness is its accumulation of the squared gradients in the denominator, Since every added term is positive, the accumulated sum keeps growing during training. This in turn causes the learning rate to shrink and eventually become infinitesimally small.
- *RMSProp and AdaDelta:* AdaDelta improves upon AdaGrad by restricting the past accumulated gradients to a fixed sized window. RMSProp is quite similar to AdaDelta except that it also divides the learning rate by a decaying average of squared gradients.
- *Adam:* Adaptive Moment Estimation (Adam) is another method that computes adaptive learning rates for each parameter. In addition to storing an exponentially decaying average of past squared gradients like Adadelta and RMSprop, Adam also keeps an exponentially decaying average of past gradients mt, similar to momentum:

Adding any of the above optimization technique aided in both speeding up the training process and improving the testing accuracy if compared to vanilla SGD. Adam update led to the fastest convergence, followed by Nesterov accelerated gradient, followed by RMS prop and vanilla SGD.

Figure 11 shows the test accuracy plots as using each of these techniques.

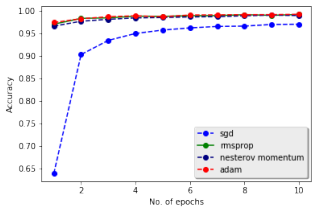

Figure 11: Optimization techniques vs test accuracies

Although, as can been seen from the plot, RMSprop,Adam and Nesterov momentum gave very similar results in terms of testing accuracy and loss. Still, Adam slightly outperformed RMSprop and Nesterov giving the fastest convergence. Hence, we chose the Adam optimization as the technique of choice for all the other experiments.

1.4 Spatial Pyramid Matching

Images classification is challenging because of variations caused by rotations, scaling, occlusion etc. in an image. Though,there are many techniques which use bag-of-features and local transformation invariant features for image representation. But most of them suffer from the lack of capturing spatial representation and creating a semantic understanding of the image.

All these problems are addressed by spatial pyramid matching technique. Spatial Pyramid is a simple and computationally efficient extension of an orderless bag-of-features image representation. In this type of image representation, image is partitioned into increasingly fine sub-regions and histograms of local features found inside each sub-region are concatenated. This makes spatial pyramid a hybrid of local features which provides clutter tolerance and the global features which helps in estimating overall perceptual similarity between images.

Pyramid matching works by placing a sequence of increasingly coarser grids over the feature space and taking a weighted sum of the number of matches that occur at each level of resolution. At any fixed resolution, two points are said to match if they fall into the same cell of the grid; matches found at finer resolutions are weighted more highly than matches found at coarser resolutions.

The process of performing classification using SPM can be summarized as below:

- Compute K visual words(codebook) using a part or complete training set using K-means.
- Generate SIFT descriptors in mxm blocks of the image for a particular level of pyramid.
- All the local features of the images are mapped to their corresponding visual code words.
- Histograms are calculated for an image at various resolutions capturing the number of visual points in a grid.
- To compute the kernel value over two images, weighted intersection of the histograms at various levels is calculated.
- Multi-class classification is then done with a support vector machine (SVM) trained using the one-versus-all rule using the kernel values computed over a test image with all training set images.

Because there are limited experimental resources, we have adopted a subset of the MNIST Database of Handwritten Digits with 5000 images for training and 200 images for testing set. For the purpose of this assignment, we found an existing implementation of SPM in python. we modified it for my use and perform experiments.

1.4.1 Image Sampling methods

The experimental results of dense, sparse and key-point sampling on natural scenes dataset shows that dense sampling works best for natural scenes. Though, we worked with MNIST dataset, we did not change the sampling method and continued with dense sampling.

1.4.2 Dense SIFT descriptors

We experimented with various DSIFT grid sizes to figure out what works best for MNIST dataset. Since, the original image is 28x28, we could only break it into grid sizes which were divisible by 28 namely 2x2, 4x4 and 7x7. We performed the experiments on 5000:200 train:test split and got the best results for 4x4 grid size.

DSIFT grid size	Test accuracy
2x2	93%
4x4	98%
7x7	85%

1.4.3 Textons

Initially, we generated the dictionary of visual code words using all the training samples. This greatly increased the time computation of the algorithm. We deal with this problem we decreased the training sample size used for generating the dictionary, which is called texton. We experimented with different texton sizes and found that if the training samples were chosen intelligently, only 50-100 training samples were sufficient to capture all the code words. The figure 12 shows the test accuracy and precision at various values texton size. We used a 5000:200 train:test split for this experiment. We found that the value of accuracy increased by increasing the texton size from 10-70 and after that it remains constant. So, we used the value 80 for the texton size for rest of the experiments.

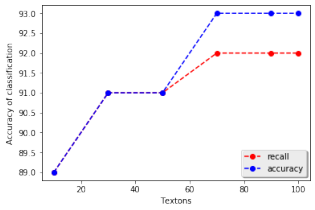

Figure 12: Number of textons vs test accuracies

1.4.4 Codeword Dictionary size

This parameter is used to cluster the visual code words in images. This parameter hence plays an important role in the correct classification of the images since if the dictionary size is very small, information can be lost. Also, in case the dictionary size is too high, it has too many clusters of the visual keywords and similar code words are classified in different clusters.

We experimented with a range of dictionary sizes from 50-600 and found that as the dictionary size increases, the accuracy peaks at 170 and 400 dictionary size. Figure 13 shows how dictionary size influences the test accuracy. As is shown, we get 2 peaks one at 170 and the other at 400 dictionary size. This shows that both of these dictionary sizes capture visual representations of the codewords. So, we could chose either of the 2 for further experiments and we used 170 as the dictionary size for further experiments.

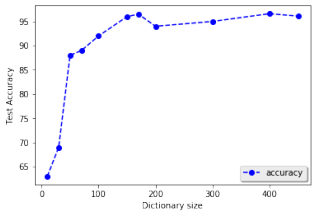

Figure 13: Dictionary size vs test accuracies

1.4.5 Pyramid levels

We experimented with various pyramid sizes

Pyramid layers	Test accuracy
1	80%
2	95%
3	72%

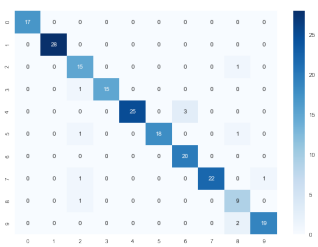

Figure 14: confusion matrix

Figure 14 shows the the confusion matrix for the SPM. We found that the model confuses between 2 and 7, 8 and 9 and 5 and 2. This is mainly because these digits are very similar spatially. Although

the image 0, 1 are almost always predicted correct. Comparing the results of SPM with NN and SVM, we find that all the 3 systems did some similar mistakes and some were peculiar to the model. NN was unable to distinguish between 1 digit and multiple other digits, every digit with a curve like 0,2,3,6,8,9,were being confused. But in SPM, the system is able to improve upon that by learning the spatial orientation of the digits.

Though when trained on a bigger dataset of 40,000 images, we were able to get an accuracy of only 92% on the test set of 1000 images which is surprising.

1.5 Deep Belief Networks

Deep Belief Networks consist of multiple layers, or more concretely, a hierarchy of unsupervised Restricted Boltzmann Machines (RBMs) where the output of each RBM is used as input to the next.

They address an important problem faced with traditional multilayer perceptrons / artificial neural networks that is backpropagation can often lead to local minima. Deep belief networks solve this problem by using an extra step called pre-training. Pre-training is done before backpropagation and can lead to an error rate not far from optimal. This puts the network in the neighborhood of the final solution. Then backpropagation is used to slowly reduce the error rate from there. There are a number of parameters involved in training DBN with which we can experiment. We list the experiments we performed on the DBNs in the following sections.

1.5.1 The structure of DBN adopted in this assignment

We found a Keras based implementation of DBN. We used it for the purpose of this assignment. The implementation we found gave us test accuracy of 99.4% with only 1400:300 training:test data split. The network we used contained 2 hidden layers with 250 hidden units each. The network was trained for 10 epochs during pre-training and for 100 epochs during backpropagation. We used mini batch of 150 samples for training.

1.5.2 Number of hidden layers in the network

We experimented with how the number of hidden layers affected the test accuracy and reconstruction error. We increased the number of layers while keeping the number of hidden units fixed 250.

The following table shows how the test accuracy changed with layers:

Number of layers	Test accuracy
1	99.16%
2	99.4%
3	99.13%
4	98.6%

We found that the test accuracy initially increases with the increase in the number of layers to 2. But after that the accuracy dropped to 99.16 % in Layer3 and 98.8 % in layer 4. Low accuracy in higher layers can be attributed to overfiting, while lower accuracy in lower layers can be due to underfitting.

1.5.3 Number of nodes in a layer

For this experiment, we kept only one hidden layer and increased the number of hidden nodes. We found that the accuracy initially increased from 99.1% to 99.4% as the number of nodes changed from 100 to 400, but after that the accuracy fluctuated but degraded to 98.6% as the nodes increased from 400 to 500.

The low accuracy in the beginning might be because the model was underfitting. While the low accuracy at higher number of nodes can be due be overfitting of the model.

The following table shows how the test accuracy changed with number of nodes in the hidden layer:

Number of hidden nodes	Test accuracy
10	95%
100	99.1%
300	99.1%
400	99.41%
500	98.6%
600	98.8%
700	98.6%

1.5.4 Learning rate

Learning rate is an important parameter and plays an important role in the learning of the model as explained in the section 1.3.3. Wee experimented with many values of learning rate in range of 10^{-5} to 0.1. and found that learning rate of 0.01 works the best for us.

1.5.5 Number of epochs

To see the effect of the number of epochs on the accuracy, we trained the model for 100 epochs for pretraining and 1000 epochs for tuning. The test accuracy degraded to 98.5% which can be attributed to overfitting. Figure 15 shows the t-SNE representation of the last layer of the DBN. The

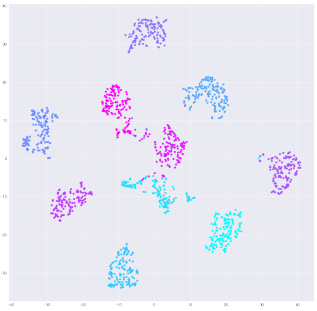

Figure 15: T-SE representation of the last layer

activations of the hidden units are projects in 2D space and can be seen in the form of 10 clusters for the ten digits.

2 Comparison of all approaches

We start the experiments with nearest neighbor classifier which is a simple algorithm based on computing the distances of test sample from the training set samples. There is essentially no training involved. It gives maximum accuracy of 97% but suffers from high computation and huge storage during testing. These problems are solved by SVM which can be seen as a generalisation of NN such that the distance of test samples is only computed from the support vectors which act as the prototype of the training set. This gives us accuracy of 98.4% on a smaller dataset. We then experimented with Spatial Pyramid which is an extension of orderless bag of features image representation. This captures the spatial representation of the image and gave us an accuracy of 97% on a smaller dataset. We then used convolutional neural networks and achieved accuracy of 99.39%. With DBN we achieved an accuracy of 99.4% on a smaller dataset.

3 Conclusion

Through this assignment we analyzed a number of techniques to classify MNIST digits. Out of all the techniques, deep convolutional neural networks gave the best accuracy 99.39%. DBNs were also able to give accuracy of 99.4% (though on a smaller dataset) and were easier to train due to the pre-training step. The astounding success of deep neural networks can be attributed to their ability

to learn the internal representation of the images helping them to outperform all the other models. SPM failed to give high accuracy when tested on bigger dataset, and performed worse than NN. This is surprising as SPM also captures spatial orientations in the image. NN gives higher accuracy with PCA which shows the importance of reducing the feature vector space in image classification. RBF filter in SVM gave the best accuracy of 98.4% which is more than the linear SVM which shows that image classification is a hard problem to solve even in a simple dataset as MNIST.

4 References

$https : //www.ki.tu - berlin.de/fileadmin/fg135/publikationen/Hebbo_2013_CDB.pdf$
$https : //chatbotslife.com/resnets - highwaynets - and - densenets - oh - my - 9bb15918ee32$ $http : //www - cvr.ai.uiuc.edu/ponce_grp/publication/paper/cvpr06b.pdf$
$http : //www - cvr.ai.uiuc.edu/ponce_grp/publication/paper/cvpr06b.pdf$ $http : //yann.lecun.com/exdb/publis/pdf/lecun - 01a.pdf$ $https : //blog.keras.io/building - autoencoders - in - keras.html$ $https : //www.cs.toronto.edu/ hinton/absps/fastnc.pdf$
$http : //sebastianruder.com/optimizing - gradient - descent/index.html$ $http : //www.cs.ucf.edu/courses/cap6412/fall2009/papers/Berwick2003.pdf$

5 Appendix