Computer Assignment 2: Comparing BPNN and CNN performance for Pattern Recognition on MNIST Dataset

Christopher Robbiano — Due: April 3, 2018 — ECE656

1 Introduction

The goal of this computer assignment is to design two neural networks, one using a back propagation neural network (BPNN) and one using convolutional neural network (CNN), for pattern recognition. The MNIST database of handwritten digits, containing 60,000 training and 10,000 testing samples, is used for comparing the performance of the two nets. The effect of net size was also examined for both network types.

2 Theory

2.1 Back Propagation Neural Nets

A back propagation neural network is a layered, feed-forward, and fully interconnected machine. The information processing that it carries out is approximating a bounded mapping function $f : A \subset R^N \rightarrow R^M$ by means of supervised learning using training examples $\{\mathbf{x}_p, \mathbf{d}_p\}_{p=1}^P$ where $\mathbf{x}_p$ vectors, which meet $\mathbf{d}_p = \mathbf{f}(\mathbf{x}_p)$, are drawn randomly from A in accordance with a fixed PDF.

When using a three layer network and letting μ be the learning rate, o_i be the ith layer output and r_i be the error signal, then the weight update equations for learning for time $p+1$ are

$$
\begin{aligned}
w_{kl}^{(3)}(p+1) &= w_{kl}^{(3)}(p) + \mu r_k^{(3)} o_l^{(2)}(p), \ \forall\ k \in [1, M] \\
w_{li}^{(2)}(p+1) &= w_{li}^{(2)}(p) + \mu r_l^{(2)} o_l^{(1)}(p), \ \forall\ l \in [1, L] \\
w_{ij}^{(1)}(p+1) &= w_{ij}^{(1)}(p) + \mu r_i^{(3)} x_j(p), \ \forall\ i \in [1, K].
\end{aligned}
$$

2.2 Convolutional Neural Nets

Convolutional neural nets are multi-layer feedforward nets designed to perform translation, scaling, skewing, and distortion invariant pattern recognition. Some popular examples of these nets are LeNet-5, AlexNet, and ImageNet. Several pairs of convolutional and subsampling layers are used to extract more abstract feature representations. Also, adding each pair of layers offers better invariance to input transformations. CNNs are able to extract local features which together with global features yield information about an image as whole. The final layer in a CNN is typically a softmax nonlinearity function or an SVM for classification problems. For the purpose of this assignment, the final layer is a softmax function.

A CNN works through 3 major ideas. The first being local receptive fields, where each field is non overlapping and contains different neurons. The second is weight sharing, through the use of feature maps. A feature map is an $N \times N$ filter that allows convolution of localized pixels with the same weights. The third is a subsampling via the pooling layers, allowing a lower resolution and in turn reducing the sensitivity to small shifts and other small distortions.

The feature value extracted at location (i, j) in the kth feature map of the lth layer is

$net_{i,j,k}^{(l)} = \mathbf{w}_k^{(l)^T} \mathbf{x}_{i,j}^{(l)} + b_k^{(l)}$, where $\mathbf{w}_k^{(l)}$ and $b_k^{(l)}$ are the weight vector and bias term of the kth filter of the lth layer respectively, and $\mathbf{x}_{i,j}^{(l)}$ is the input vector for the patch centered at location (i, j) of the lth layer. Passing this through an activation function gives the convolutional feature, i.e. $y_{i,j,k}^{(l)} = f(net_{i,j,k}^{(l)})$. The sub-sampling or pooling layer achieves local shift-invariance by reducing the resolution.

2.3 Levenberg-Marquet Optimization Algorithm

The Levenberg-Marquet algorithm solves damped or regularized least squares problems. The LM algorithm is able to interpolate between the gradient descent and Gauss-Newton algorithms. It is faster to convergence than either the Gauss-Newton or gradient descent, and still converge even if the initial guess is substantially far away from the optimal solution.

3 Network Configuration

Different network configurations were examined throughout the training phase to find what type of configuration would yield the best performance for both the BPNN and the CNN. Ultimately, configurations for each of the networks were decided upon that maximized the accuracy of classification on the validation set during early stages of training.

3.1 BPNN

A three layer network was used for the BPNN, with one input layer, one hidden layer and one output layer. All layers are fully connected at the beginning of the training phase. The input layer consists of 784 inputs, one for each pixel in the vectorized version of the training image. The hidden layer contains 84 neurons and each neuron used a hyperbolic tangent function as a logistic function. Each of the 84 neurons in the hidden layer are connected to 10 output neurons that represent each of the 10 classes. The output layer implements a softmax function and the ith output neuron contains the probability that the input pattern corresponds to the ith class. A decision to classify the input pattern is made by choosing the class corresponding the output neuron that is maximal.

3.2 Complexity of BPNN

The complexity C, or number of parameters, of the BPNN used in this assignment can be calculated as $C = N_{inputs} \times N_{hidden\ neurons} \times N_{outputs}$, where each N_i is the cardinality of the associated subscript. This formulation leads to $C = 658,560$.

3.3 CNN

Two different network configurations were used with the CNN. The first network used 2 convolution/pooling layer pairs (referred to as a 4 layer network in the assignment) while the second network used 3 convolution/pooling layer pairs (referred to as a 6 layer network in the assignment). Both networks use an input layer connected directly to the first convolutional layer and a fully connected output layer after the last pooling later and a series of convolutional and pooling layer pairs with a ReLU logistic function after each convolutional/pooling

pair for renormalization of the outputs from each pair. The input layer consists of a 28×28 grid input, with one input for each pixel of the training image. The output layer implements a softmax function connected to a classification layer where the ith output neuron contains the probability that the input pattern corresponds to the ith class. A decision to classify the input pattern is made by choosing the class corresponding the output neuron that is maximal.

3.4 Complexity of CNN

The first network uses 3 convolutional/pooling layer pairs. The convolutional layers in the first network all use a 3×3 filter size and have 8, 16 and 32 features maps for the first, second, and third layers respectively. The second network uses 2 convolutional and pooling layer pairs. The convolutional layers in the first network all use a 3×3 filter size and have 8 and 16 features maps for the first and second layers respectively.

The complexity C, or number of parameters, of each CNN can be calculated as $C_M = 10 + \sum_{i=1}^{M} Size_{filter_i} \times N_{filter_i} \times N_{input\ maps_i}$, where M is the number of convolutional/pooling layer pairs, $Size_{filter_i}$ is the number of elements in each filter kernel for layer pair i, N_{filter_i} is the number of filters at the current convolution layer for layer pair i, and $N_{input\ maps_i}$ is the number of input maps for layer pair i. The additional 10 comes from the 10 weights associated with the fully connected output layer.

The $Size_{filter_i}$ is constant for all i in both the 2 and 3 layer networks and is equal to $3 \times 3 = 9$. The 2 and 3 layer pair networks use the same first two layer pairs with $N_{filter_1} = 8$ and $N_{input\ maps_1} = 1$, and $N_{filter_2} = 16$ and $N_{input\ maps_2} = 8$ while the 3 layer pair network has $N_{filter_3} = 32$ and $N_{input\ maps_3} = 16$. This formulation leads to $C_2 = 1189$ and $C_3 = 5797$. Both of these complexity values were verified by Matlab.

4 Data Processing and Training

The data training set from MNIST contains 28×28 pixel images covering digits $0 - 9$ with a class representation label corresponding to the digit, i.e. a 4 has a class label of '4'. Some random examples of the MNIST data set can be seen in Figure 1. The 60,000 training images and 10,000 testing images were combined into a total set of 70,000 unique samples. From the combined data set a training set, testing set, and validation set were formed randomly with 15%, 42.5% and 425.5% of the total number of samples, respectively. Each set of images contains an equal amount of samples from each class. Each training image was vectorized into a 784×1 vector for input into the BPNN while remaining in the original format for input to the CNNs.

The LM algorithm was used in the case of training both the BPNN and the CNN, and the cross entropy was used to evaluate performance in both cases. The training ended once the performance on the test set converged. There were instances when the performance on the testing set would eventually increase while the performance on the training set was still decreasing. This signified that the net was over-fitting the training data, and in those cases the training was terminated at the point when the performance began increasing on the test

Figure 1: Samples from the MNIST data set

set.

The best loss for the BPNN case was 0.002 as seen in Figure 2, and the BPNN was able to accurately classify 99.7% of the training set when the training was complete.

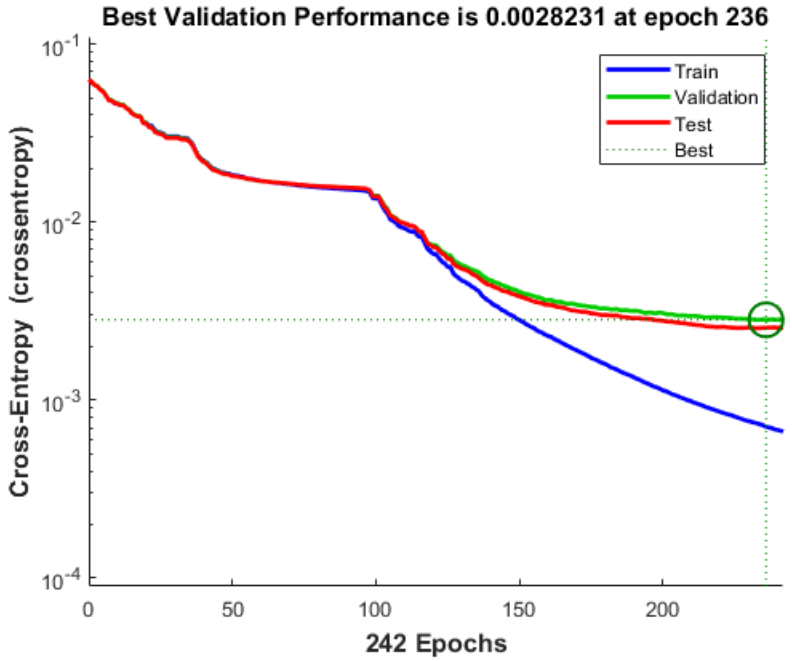
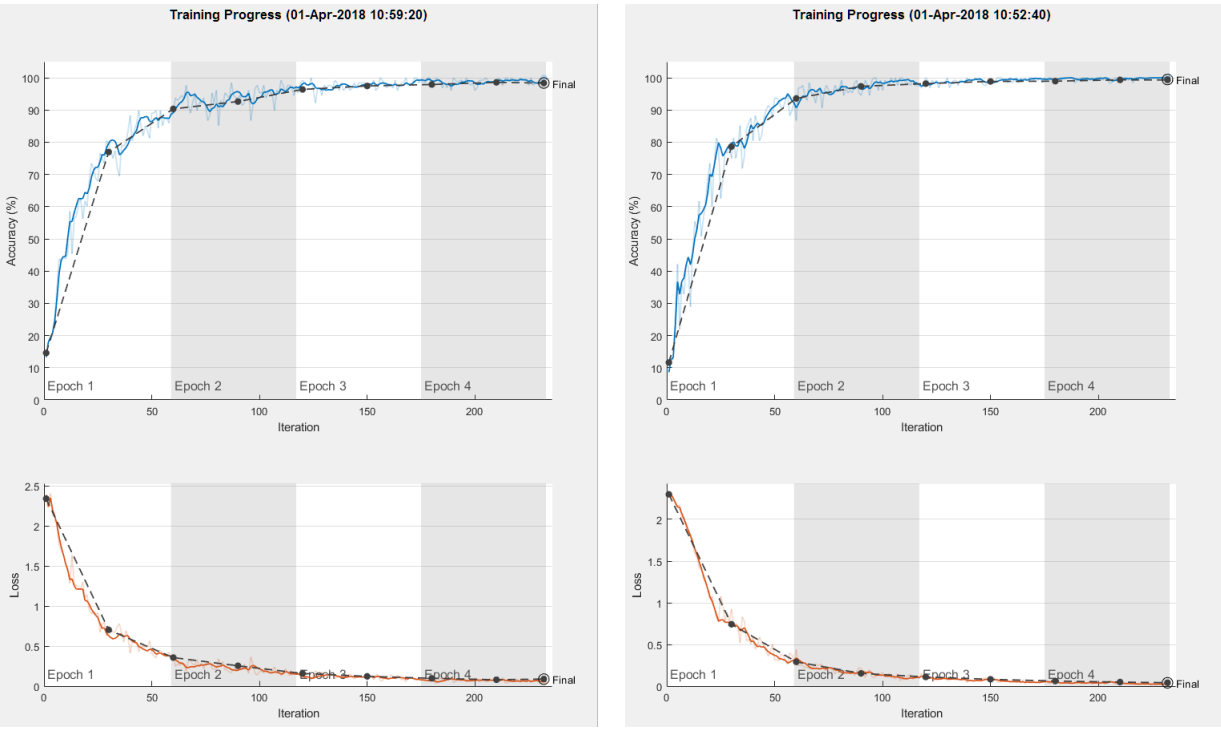


Figure 2: BPNN performance

The best loss for the CNN case was 0.08 and 0.006 for the two and three layer CNN respectively, and they were able to accurately classify 100% of the training set. The training of

the CNNs was restricted to 4 epochs. These results can be seen as the solid blue lines in the upper subgraphs of Figure 3a and 3b.



(a) 2 layer CNN

(b) 3 layer CNN

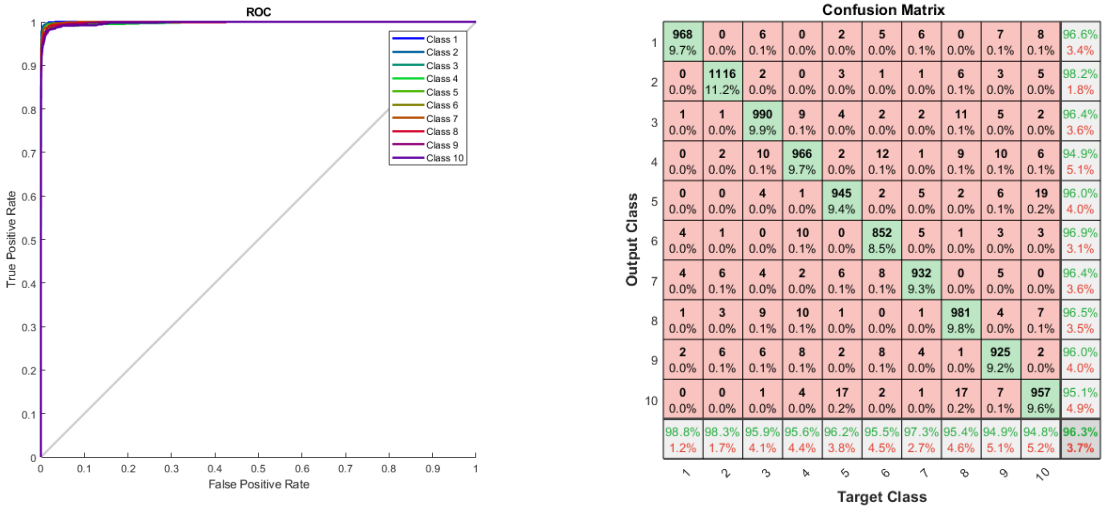
Figure 3: Performance and loss functions for CNNs

5 Results and Discussion

The results of the training are quite remarkable for both the BPNN and both versions of the CNN. The confusion matrix for the BPNN can be seen in Figure 4b, with the overall accuracy of the classifier being 96.3%.

It should be noted that due to limitations in scripts used for generating the confusion matrix that all of the class labels have been incremented by 1, i.e. class 0 is represented by 1 in the figure. During the training phase there were times when the accuracy was fairly poor, around 60%, due to the uniform random sampling of the entire dataset not being truly random resulting in the training set missing representatives from one or more classes. This was readily apparent when looking at the ROC curves for the outputs and seeing one or more classes that had a concave curve. The ROC curves for the best BPNN can be seen in Figure 4a.

The confusion matrix for the CNNs can be seen in Figure 5b and 5d for the two and three layer networks. The two layer was able to classify correctly 98.6% of the time while the three layer was able to get another 1% performance boost on top of that. This is likely due to an additional amount of location invariance from the third pair of convolutional and pooling



(a) ROC curve

(b) Confusion matrix

Figure 4: ROC curve and Confusion matrix for BPNN

layers that extracted lower level features. The ROC curves for the best CNNs can be seen in Figure 5b and 5d.

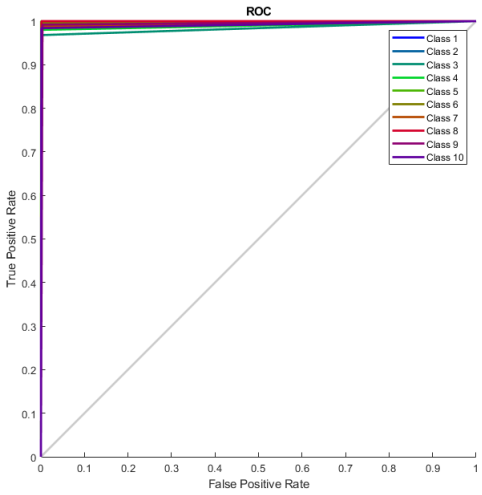
It is clear that the CNNs are able to out perform the BPNN when trained and validated on the same data. I was particularly impressed with the ability of all three networks to correctly classify the digits with over 95% accuracy. The complexity of the CNN is much less than that of the BPNN in both cases, and the training time is significantly less as well. For these reasons, the CNN is a much better candidate for doing image classification on the MNIST image dataset than the BPNN.

6 Conclusion

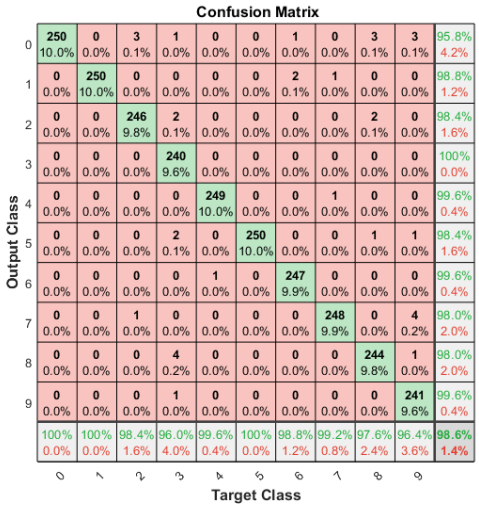
In this assignment we looked at the performance between BPNN and CNN networks for use in pattern recognition, and in particular with the MNIST digit database. The LM algorithm was used to train the nets while cross entropy was used as a performance metric. It was found that the CNN network type was able to train in a smaller number of iterations and perform more accurately while doing so.

7 References

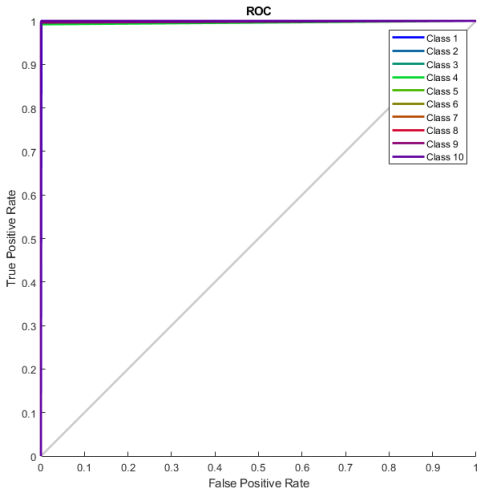
Neural Networks and Learning Machines
ECE656 Class Notes



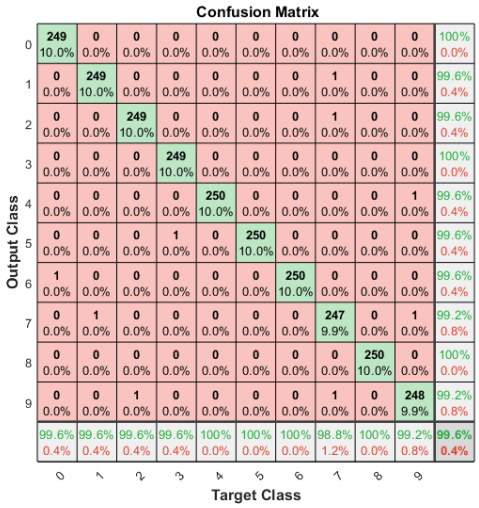
(a) ROC curve for 2 layer



(b) Confusion matrix for 2 layer



(c) ROC curve for 3 layer



(d) Confusion matrix for 3 layer

Figure 5: ROC curve and Confusion matrix for CNNs