Trinadh Gupta

Depaul University

mnist dATA cLASSIFICATION VIA NEURAL NETWORKS

# Project Abstract:

Artificial Neural networks(ANN) is an information processing technique from machine learning which is inspired from biological nervous system, which is composed of a highly interconnected complex processing units called neurons. ANN’s are predominantly used in the pattern recognition or data classification tasks.

The goal of the current project is to classify the handwritten digits using the famous MNIST data using different types of neural networks and transfer functions as mentioned below. Though ANN is seemingly vast subject with research still on, and owing to its success with today’s computational power, the focus of the current project is primarily confined to few basic neural network implementations and evaluation of one deep neural network. The list of neural network implementations are as follows.

1. One layer Feedforward Neural Network (Sigmoid)
2. Multilayer Feedforward Neural Network (Sigmoid)
3. Multilayer Feedforward Neural Network (Fast Sigmoid)
4. Multilayer Feedforward Neural Network (Tanh)
5. Deep Neural Feedforward Network

Note: Deep Neural feedforward network is an inbuilt matlab function, I haven’t implemented this but used as an evaluation function.

# Data Description

The data is sourced from the [Kaggle](https://www.kaggle.com/c/digit-recognizer) , which is an MNIST data, where we have the data files train.csv and test.csv contain gray-scale images of hand-drawn digits, from zero through nine.

Each image is 28 pixels in height and 28 pixels in width, for a total of 784 pixels in total. Each pixel has a single pixel-value associated with it, indicating the lightness or darkness of that pixel, with higher numbers meaning darker. This pixel-value is an integer between 0 and 255, inclusive.

The training data set, (train.csv), has 785 columns. The first column, called "label", is the digit that was drawn by the user. The rest of the columns contain the pixel-values of the associated image.

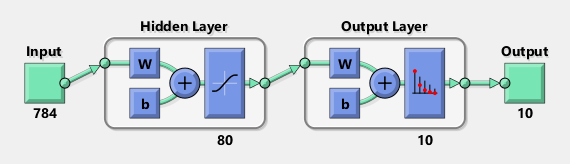
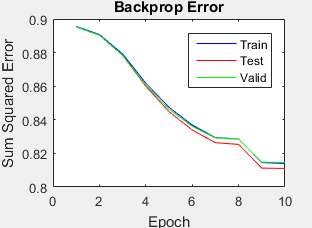
Each pixel column in the training set has a name like pixelx, where x is an integer between 0 and 783, inclusive. To locate this pixel on the image, suppose that we have decomposed x as x = i \* 28 + j, where i and j are integers between 0 and 27, inclusive. Then pixelx is located on row i and column j of a 28 x 28 matrix, (indexing by zero).

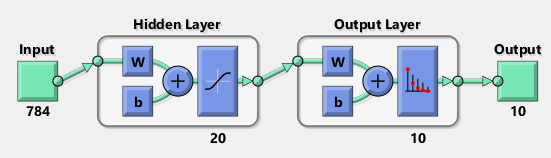
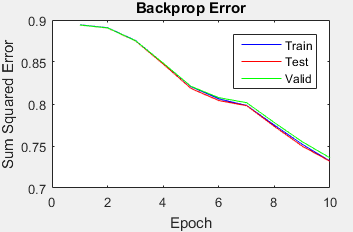
For example, pixel31 indicates the pixel that is in the fourth column from the left, and the second row from the top, as in the ascii-diagram below.

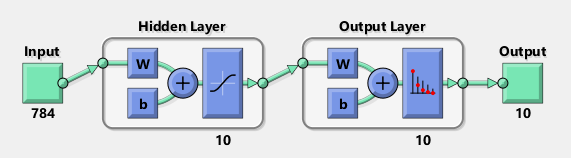
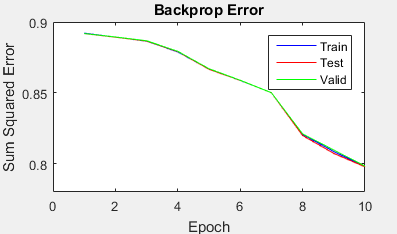
Also, since the test data provided in the link doesn’t have the outputs available for now. I had to split the given training data (42000 training instances) to be used as training, validation and test data.

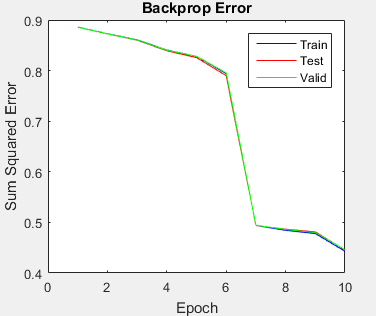
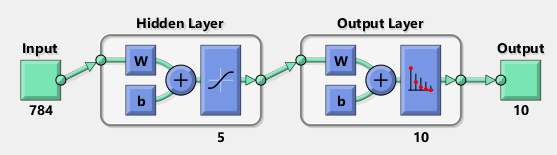
# One layer Feedforward Neural Network (Sigmoid):

This is the basic level of neural network implementation with sigmoid as its activation function and it takes 784 input vector to generate an output that classifies the output into one of the 10 digits that it has already learnt previously. In this case I have used different combinations of hidden neurons, just to find out that the neural network is working best when we use 5 hidden neurons.

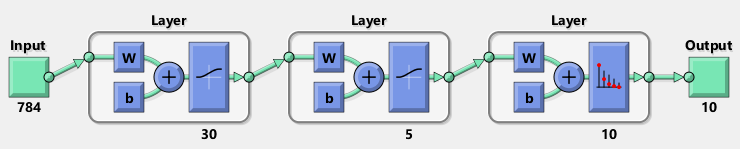
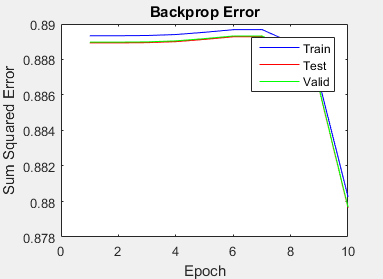


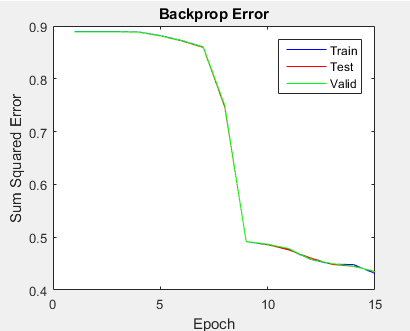
# Multilayer Feedforward Neural Network (Sigmoid):

This is the multilayer neural network implementation with sigmoid as its activation function at each layer and it takes 784 input vector to generate an output that classifies the output into one of the 10 digits that it has already learnt previously. In this case I have used different combinations of hidden neurons and at different epochs not more than 15, because of the fact that after 10 epochs the gradient decent seems to stuck in local minima for a pretty long time and its taking almost 20 mins for each epoch after though the learning rate and momentum are around 0.1 and 0.1. Haven’t had a chance to investigate at a much slower learning rate, due to the time constraints. This can be of future study scope.



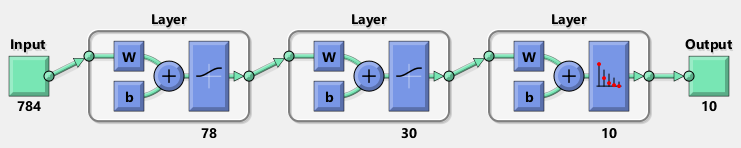




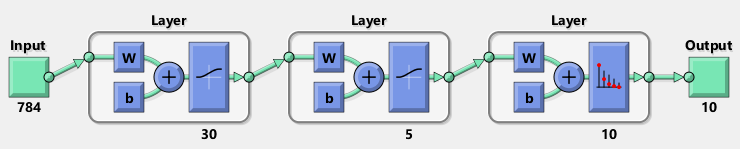


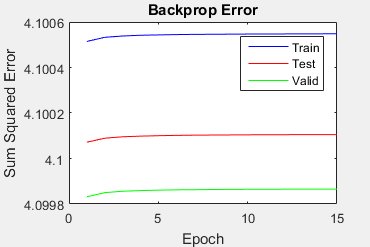
# Multilayer Feedforward Neural Network (Fast Sigmoid):

Though the main purpose of this function has been served, which is time reduction, the purpose is not served, if you see the SSE, it is way out of normal compared to sigmoid function. In this case I have used different combinations of hidden neurons just only to find that there is no convergence of error to 0. Haven’t had a chance to investigate the bias factor in this scenario, due to the time constraints. This can be of future study scope.



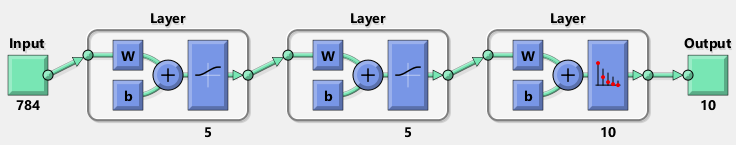
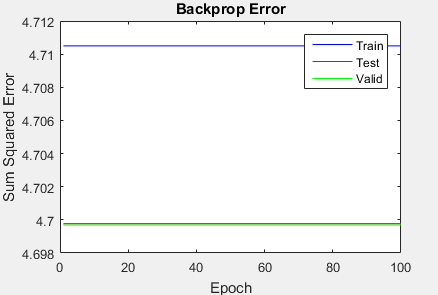


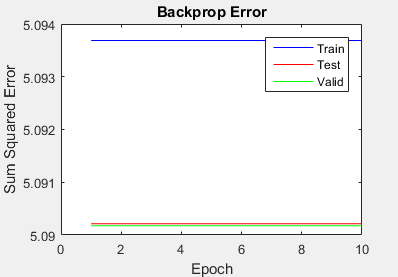


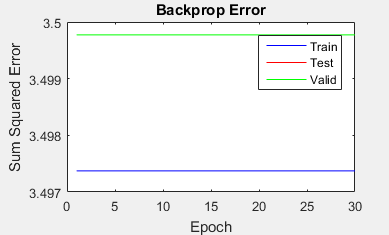


# Multilayer Feedforward Neural Network (Tanh):

This just another variation of sigmoid function where time is the primary factor but if we see the SSE at different epochs for the same set of hidden neurons, though the error is constant over the different runs of epochs, it seems to behave inconsistently and in addition the error is way out of normal compared to sigmoid function. Due to time limitation I haven’t investigated much further, but this can be a good scope of further study.

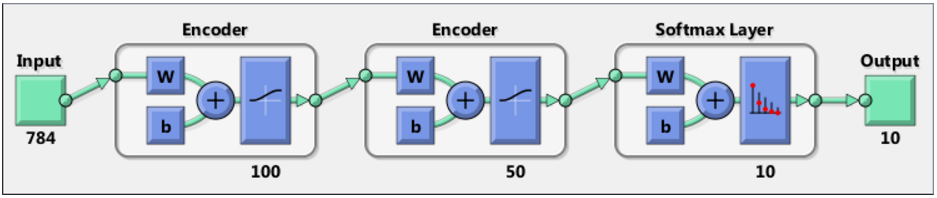


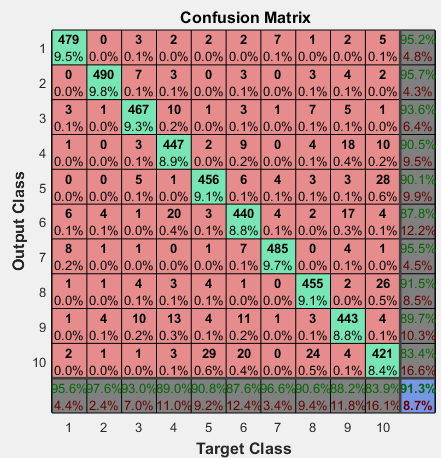
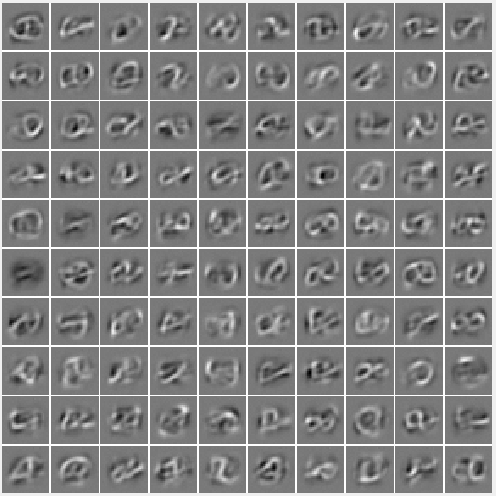




# Deep Neural Network Feedforward Neural Network

This is not an implementation but an evaluation of deep neural network from matlab examples, adapted to my dataset. This neural network uses auto-encoder technique, the auto-encoder learns a compressed representation of the input. We have two encoders here the first encoder converts the 784 inputs instance vector to 100, and the second one converts that 100 input vector to 50 in size vector. Both the encoders are unsupervised learning of features. Then at the final stage the output of second encoder is sent to soft max layer to emit the digit classification between 0-9. The confusion matrix shows that the accuracy of classification is 91.3% which is pretty outstanding.





# Conclusion & further study

* Based on observation of sum squared errors, its apparent that the neural network works best with 5 hidden neurons in “One layer feed forward network (sigmoid)” and the classification is best with 30 layer1 neurons and 5 layer 2 neurons in “Multilayer feedforward neural network (sigmoid)”.
* Both Fast sigmoid & “Tanh” are no way in comparison
* Deep Neural network seems best of all with a 91.3% accuracy.
* As said in the above sections we may need to do some further study in the area of fast sigmoid and tanh function and as well need to identify how to avoid the gradient descent stuck in local minima for long time after 10 epochs.

# Appendices

Deep neural network is simple straight forward to run, as code is all set. But for one and multi-layer neural network, instructions are mentioned in the comments in “CodeData.m” file right from building data to executing each type of neural networks.

