Assignment4: Neural Networks

Saeed Kalateh, MSc. Student, Robotic Engineering,

Abstract—This report includes two main sections. The first section is dedicated to task 1, which includes using MLPs to classify patterns. The second section is the implementation of auto-encoders on the MNIST dataset.

Index Terms—Multi-layer perceptron, Auto-encoders

1 Feedforward multi-layer perceptrons

Following the instruction of the assignment4 in task1, Matlab Neural Network toolbox in use to recognize patterns using shallow neural networks.

1.1 Training a Neural Network Using MATLAB

Glass Data set is imported as the input data to the neural networks. This dataset contains 214 observations, each with 9 features as shown in Figure 1. Each observation falls into one of two categories: windowed or non-windowed.
Next, the 214 observations should be partitioned into Train, Test, and Validation subsets. The default values for these partitions are 70%, 15%, and 15%, respectively. For this test, the default values are used as in Figure 2. The number of hidden layers are left to 10 (default value). After training, MATLAB illustrates the structure of our Neural Network as shown in Figure 3. The input is 9 since Glass Data set has 9 features. The output is set to 2 because of there are two possibilities for the target. As mentioned, there are 10 hidden layers in this trained MLP.

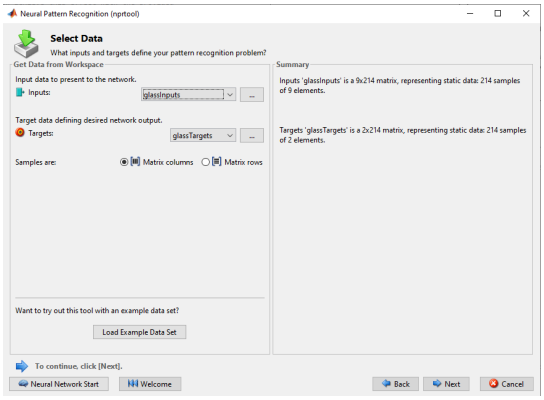

Fig. 1. Glass Data set summary

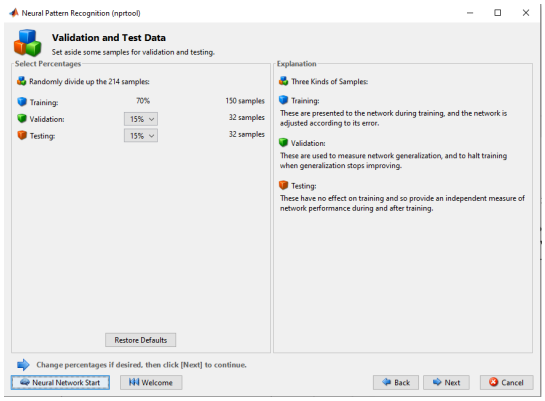

Fig. 2. Data partitioning

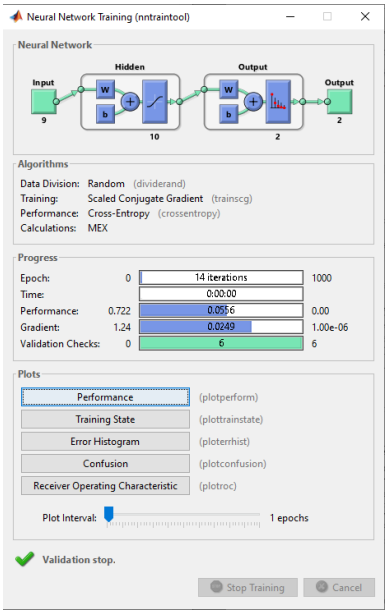

Fig. 3. Neural Network Structure

1.2 Analyzing the trained network

The MATLAB software also gives useful information about the training process. For instance, the Progress section shows the number of epochs that took place before our model converged, which in this case it is only 14. The confusion matrix can be depicted using the confusion button in the plot section of Figure 3. As shown in Figure 5, four confusion matrices have been generated, one for each of Train, Test, Validation, and Total partitions data. The Network has predicted 94.4% of all data successfully. As expected, the lowest accuracy belongs to the Test matrix with 93.8%.

Fig. 4. Confusion matrix for 80% training data

Fig. 5. Confusion matrix for Glass data set

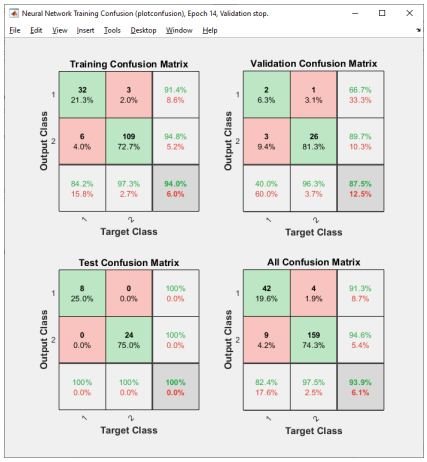

Fig. 6. Confusion matrix with 5 neurons

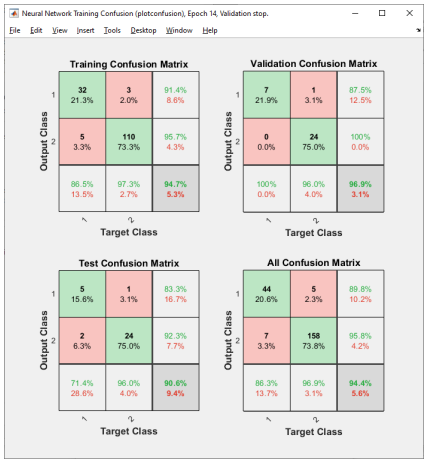

Fig. 7. Confusion matrix with 15 neurons

1.3 Confusion Matrix with a different design for Network

Again we use Glass Data set in MATLAB and investigate the effect of different Neural Network designs with respect to different parameters. It is assumed that the settings in our previous section is our default setting and the modifications are made to this default configuration for each case.

First, we assign more percentage of our data to train (80%) and leave the other configurations the same (Figure 4). The prediction on the training and validation data set have been improved substantially. However, The accuracy on Test set has slightly decreased. These number show the effect of overfitting.

Second, the effect of number of hidden layers on the Network has been investigated. The default number of hidden layer is 10. The network has been evaluated using 5, 15, and 20 hidden layers.

The confusion matrices show that the overall accuracy is improving as we increase the number of neurons. However, this increase is mainly due to the training accuracy. Since Validation and Test accuracy don't follow any patterns, increasing the number of neurons might lead to overfitting.

Therefore, increasing the number of hidden layers might improve the accuracy or might not, it really depends on the complexity of the problem. Increasing the number of hidden layers more than the sufficient number of layers will cause accuracy in the test set to decrease. It will cause your network to overfit to the training set, that is, it will learn the training data, but it won't be able to generalize to new unseen data.

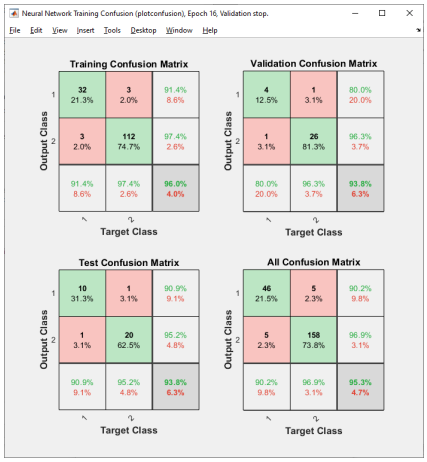

Fig. 8. Confusion matrix with 20 neurons

Fig. 9. Confusion matrix for Iris data set

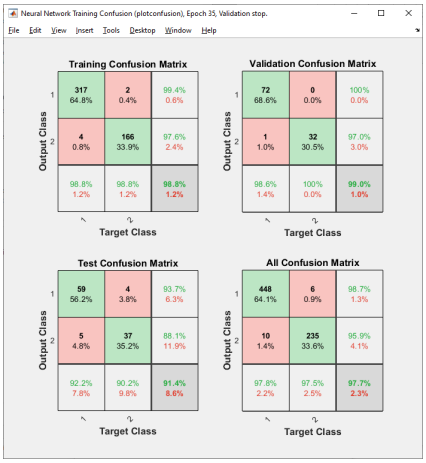

Fig. 10. Confusion matrix for Breast Cancer data set

1.4 Using different problems to train the network

In this section, the process is repeated using other datasets available in MATLAB, including Iris flowers and Breast Cancer.

1.4.1 Iris data set

This data set has 150 samples of 4 elements. The confusion matrix is as shown in Figure 9. The matrices are 3x3 since the target has 3 levels. Although the number of observations is less than the Glass Data set, the network stills shows 96.7% overall accuracy. The 100% accuracy on the test set draws attentions which shows that the network has been well generalized for test set.

1.4.2 Breast Cancer data set

This dataset can be used to design a neural network that classifies cancers as either benign or malignant depending on the characteristics of sample biopsies. Cancer input includes a 9x699 matrix defining nine attributes of 699 biopsies. Cancer target is a 2x699 matrix where each column indicates a correct category with a one in either Benign or Malignant elements. Figure 10 shows the confusion matrix for the trained network using Breast Cancer data set.

2 Autoencoder

In order to split the data into subsets of different classes, loadMNIST() function is used to import Mnist data set:

$$[x1,x1\_label] = loadMNIST(0,[1]);$$

The above line in MATLAB imports handwritten number 1 images in x1 and the relevant labels in x1_labels. This processed has been repeated for every 10 classes. To create a training set with only 2 classes, two of these x sets have been concatenated. 1 and 8 have been chosen for our case:

$$train\_data=vertcat(x1,x8);$$

The train data is not in MATLAB acceptable form and it needs to be transposed before training. Next, it is time to train auto encoder using the trainAutoencoder command.

$$trainAutoencoder(train\_data, 10, 'MaxEpochs', 300);$$

The parameters are left to their default values except *MaxEpochs*. Since the data set is large, it has been limited to 300 epochs. In addition, 10 hidden units have been chosen for our single hidden layer (Figure 11).

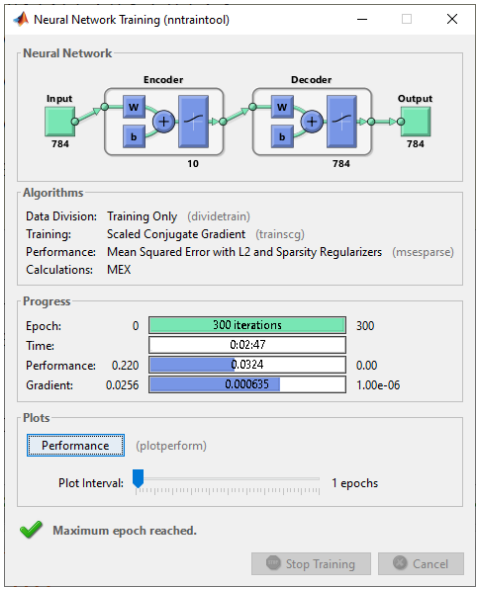

Fig. 11. Training the auto encoder

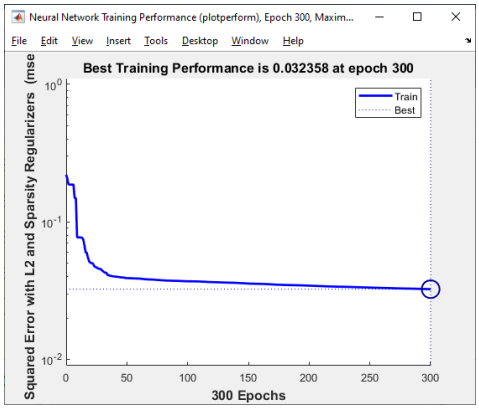

Fig. 12. Performance of the auto encoder

After 300 epochs, the network is still at its best performance. This means that further accuracy is feasible if more training epochs are set. Training was done on numbers 1 and 8 splits of data set. Figure 13 shows the Original images from data set and the Predicted images. The network

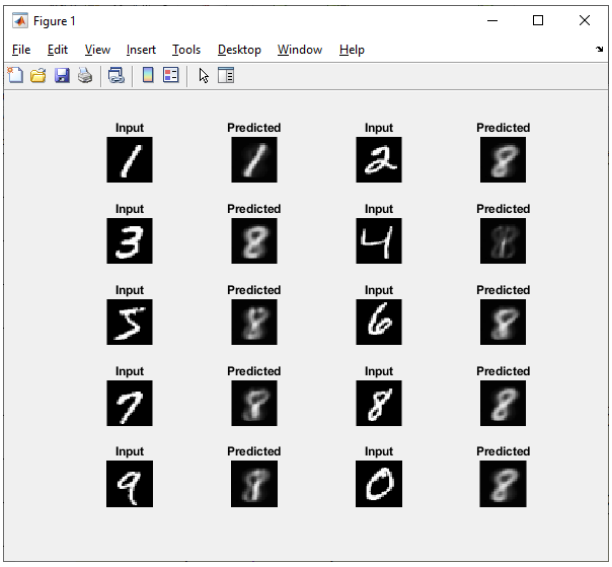

Fig. 13. Trained with 1 and 8 splits of data set and 10 hidden neurons

tries to regenerated numbers with extracted features. It can successfully regenerate numbers 1 and 8. Now, the hidden layer units are increased to 100 to investigate the results.

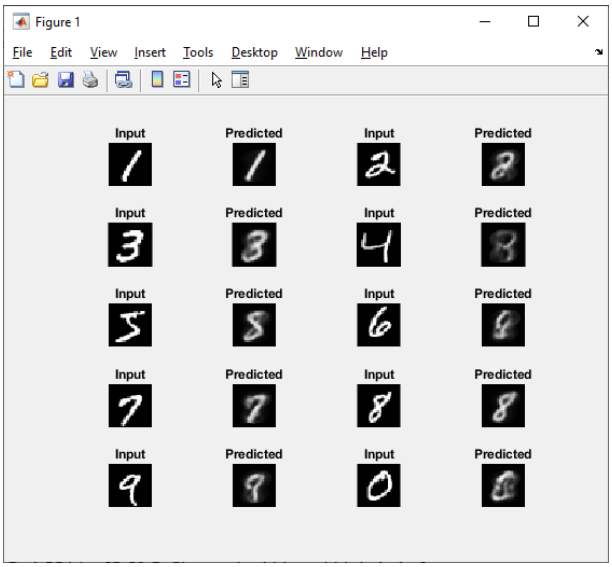

Fig. 14. Trained with 1 and 8 splits of data set and 100 hidden neurons

Figure 14 shows obvious improvements. By increasing the number of neurons in the hidden layer, more features have been extracted and the network performs better at regenerating numbers.

Now, let's try to train the network using different numbers. Pictures of numbers 4 and 6 have been used as the training data. The number of neurons is again 100, since 10 failed to predict well. Figure 15 illustrates the results of this test.

As a result, the results are dependent on number of units in the hidden layer and the train set. Different numbers have different features and these features help the network to train and reconstruct the the numbers. The more similar features between train and test, the better the output. Increasing number of neurons might help the training and feature extraction. However, it is not always good practice to set a large number to neurons of hidden layer since it might lead to overfitted model.

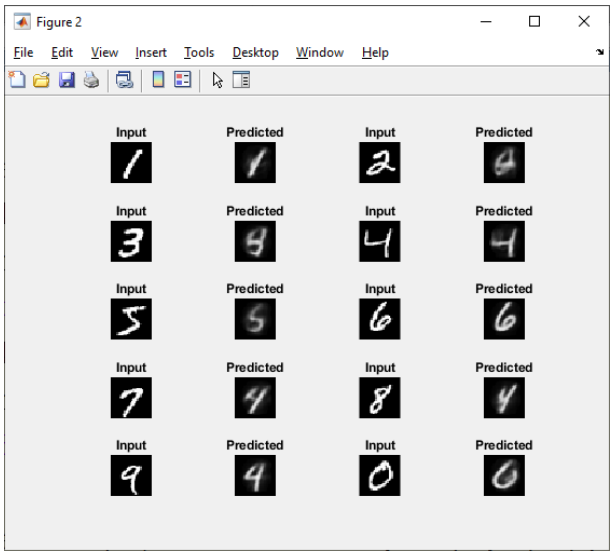

Fig. 15. Trained with 4 and 6 splits of data set and 100 hidden neurons