EECE6036 - Homework 3

Wayne Stegner

October 31, 2020

1 Problem 1

1.1 Problem Summary

The goal of this problem is to create a multi-layer feed-forward neural network for classification of the MNIST dataset. The network was trained using gradient descent with back-propagation with momentum, and is trainable for arbitrary amounts and sizes of hidden layers. During training, the loss is calculated using J_2 loss with threshold values. For performance measurement, the loss is calculated as $1 - hit\_rate$ using winner-take-all on the output layer to define the predicted class.

1.2 Results

1.2.1 System Description

Table 1 shows the hyper-parameters used in training the classifier. These hyper-parameters were found empirically, considering both minimizing the final loss and training time. More work can to find more optimal hyper-parameters, but this set of hyper-parameters produces pretty good results.

Table 1: Classifier Training Hyper-Parameters

Parameter	Value	Description
$hidden\_layer\_size$	192	Neurons in hidden layer
η	0.05	Learning rate
α	0.8	Momentum
max_epochs	500	Maximum training epochs
L	0.25	Lower activation threshold
H	0.75	Upper activation threshold
$patience$	3	Patience before early stopping
es_delta	0.01	Delta value for early stopping

Weight initialization is done randomly on a uniform distribution between $(-a, +a)$ where $a = \sqrt{(\frac{6}{N_{source}+N_{target}})}$, and N_{source} and N_{target} are the numbers of neurons on the source and target layers respectively.

The network utilizes early stopping by monitoring a validation set, which consists of 1000 training points that are set aside before training. If the validation loss does not improve for *patience* steps (1 step = 10 epochs), training halts. The validation is considered improved if the validation is es_delta lower than the previous improved validation loss. The weights used by the final network are the weights from the epoch with the lowest validation loss. To improve the frequency of checking for early stopping, the network only uses 1000 training points per epoch.

1.2.2 Network Results

Throughout the duration of training, the loss of the training and validation sets was tracked every 10 epochs. Figure 1 shows a plot of the loss values while training the classifier. The vertical line designates the point where the validation error is minimized, which occurs at epoch 50. The weights at that epoch are the weights used in the final network, where the test loss is 0.070.

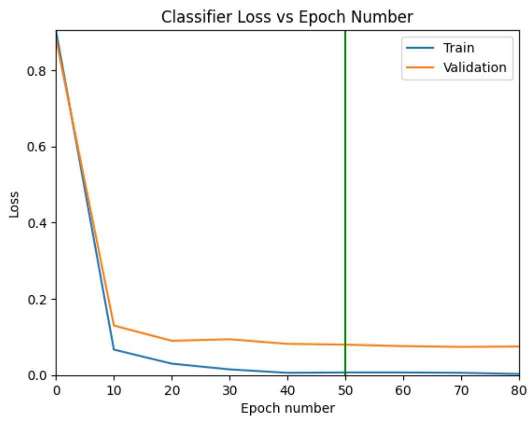
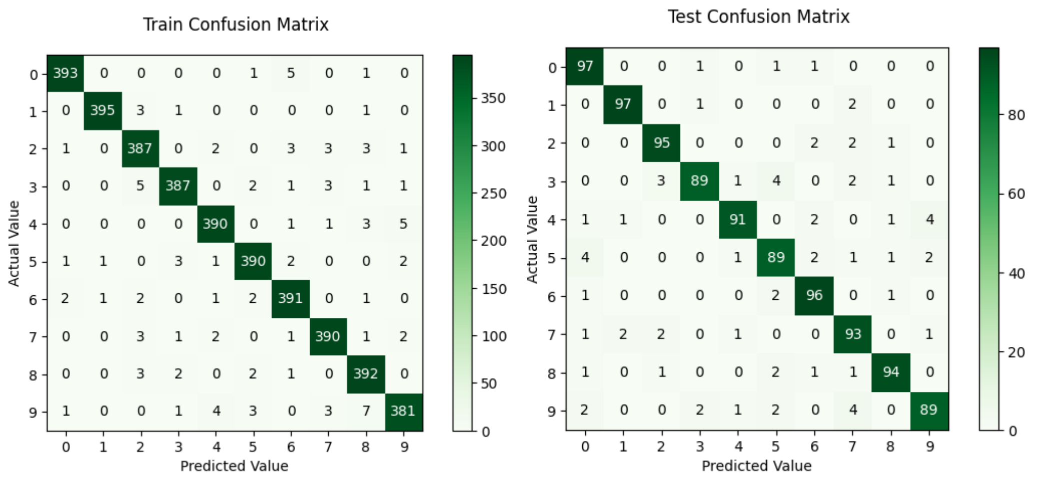


Figure 1: Training and validation loss of the classifier.

Figure 2 shows the confusion matrices for the train and test sets using the weights from the best epoch.



(a) (b)

Figure 2: Confusion matrices of the train and test sets.

1.3 Discussion and Analysis of Results

Overall, the results look pretty good. The best weights happen pretty early at epoch 50 and get a fairly low loss of 0.070. While testing hyper-parameters, it was observed that when *es_delta* is set to 0 and *patience* is increased to around 5, the test loss would sometimes get as low as 0.05, but that generally occurred after several hundred epochs and is also dependent on weight initialization numbers and dataset shuffling during training. With the early stopping hyper-parameters used by this model, it sacrifices some performance, but it trains significantly faster, making these hyper-parameters ideal for implementing and troubleshooting the network from scratch.

The diagonal of the confusion matrices in Figure 2 is very dark, with hardly any coloring in the incorrect boxes. Many of the incorrect classifications are in two classes which look similar. For example, Figure 2a shows four 9s classified as 4s and five 4s classified as 9s. This mix-up makes a lot of sense, because the shapes of 4s and 9s tend to be similar, especially with some of the sloppy handwriting in MNIST.

1.4 Conclusion

This multi-layer feed-forward neural network classifier is able to effectively classify the MNIST dataset given in this problem. While the results are certainly not state-of-the-art, the trained network classifies over 90% of the test set digits correctly. Further testing in optimizing the hyper-parameters of the network can help to improve the accuracy even further.

2 Problem 2

2.1 Problem Summary

The goal of this problem is to create a multi-layer feed-forward neural network similar to Problem 1, except in this case it will be trained to be an autoencoder. This network has the same features as the network in Problem 1, except the performance measurement is calculated using J_2 loss instead of $1 - hit\_rate$. Additionally, winner-take-all is not used in the output layer, because it does not make sense in the context of an autoencoder.

2.2 Results

2.2.1 System Description

Table 2 shows the hyper-parameters used in training the autoencoder. As with the classifier, these hyper-parameters were found empirically with the same consideration for minimizing final loss and training time.

Table 2: Autoencoder Training Hyper-Parameters

Parameter	Value	Description
$hidden\_layer\_size$	192	Neurons in hidden layer
η	0.005	Learning rate
α	0.8	Momentum
max_epochs	500	Maximum training epochs
L	0	Lower activation threshold
H	1	Upper activation threshold
$patience$	3	Patience before early stopping
es_delta	0.1	Delta value for early stopping

The weight initialization process is the same as in Problem 1, as are the processes for allocating the validation set and applying early stopping.

2.2.2 Network Results

Throughout the duration of training, the loss of the training and validation sets was tracked every 10 epochs. Figure 3 shows a plot of the loss values while training the autoencoder. The vertical line designates the point where the validation error is minimized, which occurs at epoch 160. The weights at that epoch are the weights used in the final network, where the test loss is 2.087.

After training, the loss in each class was measured. Figure 4 shows the loss of each class for the train and test sets using the weights from the best epoch.

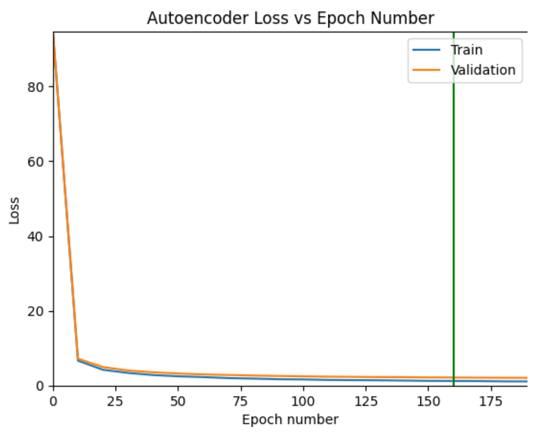


Figure 3: Training and validation loss of the autoencoder.

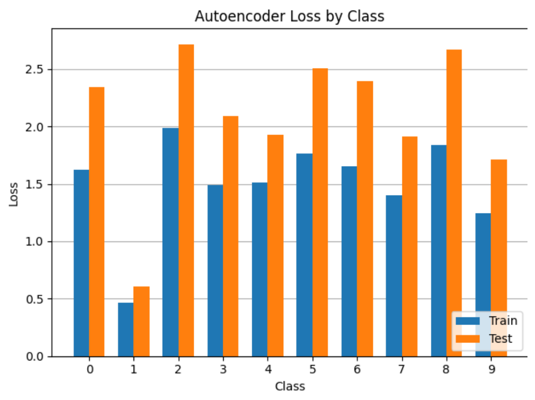


Figure 4: Loss of each class of the train and test sets.

2.2.3 Features

After training, the features learned by 20 arbitrary neurons in the hidden layer of both the classifier and autoencoder were observed by normalizing the weights from 0 to 1 and then displaying them like an image. The images are mapped so 1 is white and 0 is black. Figure 5 shows the features of the classifier, and Figure 6 shows the features of the autoencoder.

2.2.4 Sample Outputs

After training, the outputs of eight random data points from the test set were fed reconstructed the autoencoder, meaning the input data point was presented to the network and the output ”prediction” was obtained. Figure 7 shows the original and reconstructed images.

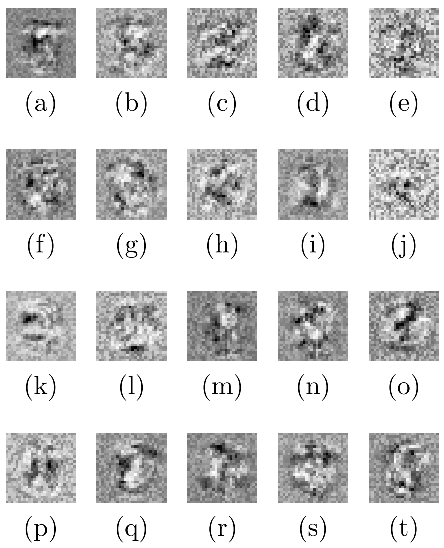


Figure 5: Classifier features.

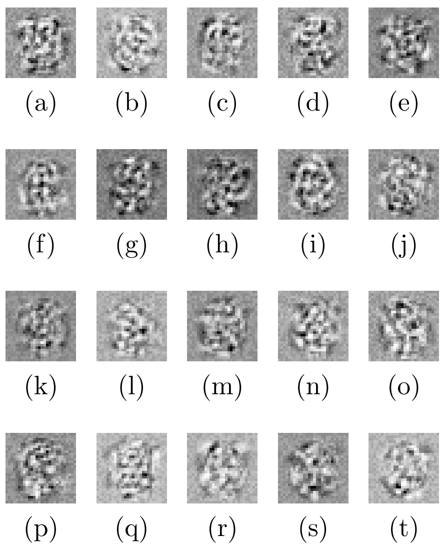


Figure 6: Autoencoder features.

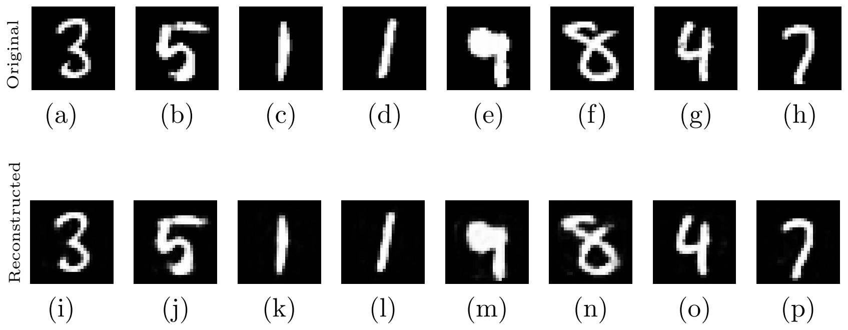

Original
(a) (b) (c) (d) (e) (f) (g) (h)
Reconstructed
(i) (j) (k) (l) (m) (n) (o) (p)

Figure 7: Original (top) and reconstructed (bottom) data points.

2.3 Discussion and Analysis of Results

Overall, the results look pretty good. The autoencoder takes longer to train, with its best weights occurring at epoch 160 with a loss of 2.087. In Figure 4, the loss for the 1s is particularly salient because it is drastically lower than the other classes loss values. This makes sense because a 1 is simply a straight line, so it should be quite simple to recognize and then reconstruct. While none of the loss values are as noticeably high as 1's loss is low, classes 2, 8, and 5 all have loss values above 2.5. I do not have any intuition as to why those numbers have the highest loss, but those digits are kind of similar in the sense that a 2 is roughly a backward 5, and stacking a 2 on top of a 5 roughly makes an 8. At first, I was surprised that 6 has a higher loss than

9, because 6 is just an upside-down 9, but it makes sense that 5 and 6 have similar features and ended up having similar loss values.

The features in Figure 5 and Figure 6 are somewhat difficult to interpret. Between the two feature sets, there is a trend where the center of the image appears to have some distinct patterns, while the edge of the image is either somewhat uniform gray or random TV static. In Figure 5, the shapes generally seem to be lines or curves, while in Figure 6 the features look like a bunch of dark dots with a few brighter spots.

2.4 Conclusion

This multi-layer feed-forward neural network autoencoder is able to effectively reconstruct digits from the MNIST dataset. Further testing in optimizing the hyper-parameters of the network can help to improve the reconstruction accuracy even further.