EECE6036 - Homework 4

Wayne Stegner

November 15, 2020

1 Problem 1

1.1 Problem Summary

The goal of this problem is to create a denoising autoencoder using multi-layer feed forward neural networks. The network structure is identical to that in Homework 3 Problem 2 with the exception that the input data has noise applied to it, and it is expected to reconstruct the image without the noise. The noise model used in this problem is a salt-and-pepper noise model, where each input pixel has a probability of 0.4 to turn black and a probability of 0.05 to turn white. The pepper probability is higher to compensate for the fact that most of the images are black already.

1.2 Results

1.2.1 System Description

Table 1 shows the hyper-parameters used in training the autoencoder (both clean and denoising). I retrained my clean autoencoder because I added weight decay and wanted to properly compare the networks. These hyper-parameters were found empirically.

Table 1: Autoencoder Training Hyper-Parameters

Parameter	Clean	Noisy	Description
hidden_layer_size	128	128	Neurons in hidden layer
η	0.01	0.005	Learning rate
α	0.8	0.8	Momentum
λ	0.0001	0.00025	Weight decay
max_epochs	500	500	Maximum training epochs
patience	5	5	Patience before early stopping
es_delta	0.005	0.005	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.

My learning algorithm includes weight decay. To calculate weight updates, this adds the term $-\lambda * \eta * w$. The goal of adding weight decay is to improve generalizability, especially because the features of the autoencoder will be used to make a classifier in the next problem.

The network utilizes early stopping by monitoring a validation set, which consists of 1000 training points that are set aside before training. This leaves 3000 data points for training the weights of the networks. 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.

1.2.2 Network Results

Throughout the duration of training, the loss of the training and validation sets was tracked every 10 epochs. The loss was calculated using $J2$ loss. Figure 1 and Figure 2 show training and validation loss for the clean and denoising autoencoders respectively. The vertical lines designates the point where the validation error is minimized. The clean autoencoder achieves a minimal test loss of 1.555 at epoch 270. The denoising autoencoder achieves a minimal test loss of 9.454 at epoch 20.

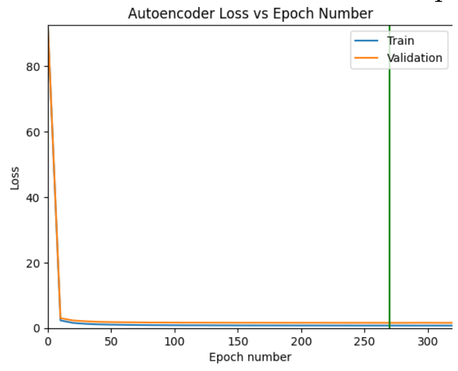


Figure 1: Training and validation loss of the clean autoencoder.

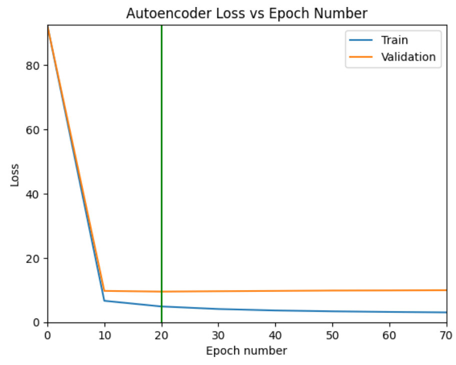


Figure 2: Training and validation loss of the denoising autoencoder.

After training, the loss in each class was measured for each autoencoder. Figure 3 and Figure 4 show the final loss of each class (and overall) for the clean and denoising autoencoders respectively.

1.2.3 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 clean autoencoder, and Figure 6 shows the features of the noisy autoencoder.

1.2.4 Sample Outputs

After training, the outputs of eight random data points from the test set were tested in both the clean and denoising autoencoders. Figure 7 shows the original and reconstructed clean images, and Figure 8 shows the original and denoised images.

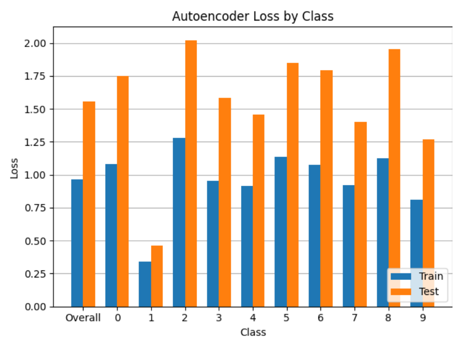


Figure 3: Loss of the clean autoencoder.

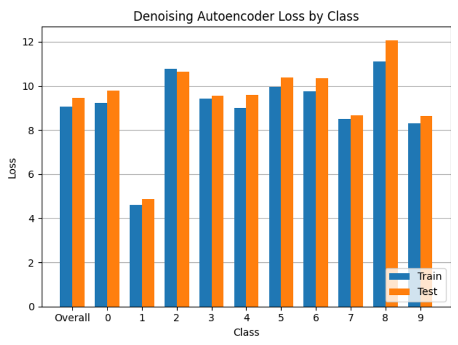


Figure 4: Loss of the denoising autoencoder.

1.3 Discussion and Analysis of Results

Overall, the results look pretty good. The denoising autoencoder does not reconstruct the images as well as the clean autoencoder, but this is expected because denoising is a more difficult task than simple reconstruction. To compensate for the increased difficulty, the denoising autoencoder was trained with a lower η value and a higher λ value. The higher λ is evident because the loss values of the train and test sets are closer together for the denoising autoencoder than for the clean autoencoder.

In Figure 3 and Figure 4, the loss for both the clean and denoising autoencoders are the lowest for reconstructing 1s. This makes sense because a 1 is simply a straight line, so it should be quite simple to recognize and reconstruct. None of the loss values stand out as particularly high, at least not as much as the 1s stand out for being low.

The clean features in Figure 5 are just a bunch of dots, while the denoising features in Figure 6 contain more lines and curves. The denoising features bear more resemblance to actual numbers than the clean features. Because of the weight decay, the borders of the clean features are uniform color because of the absence of input in those regions, while the borders of the denoising features appear as TV static because they were stimulated by the noise.

1.4 Conclusion

The denoising autoencoder is able to effectively remove salt-and-pepper noise and reconstruct digits from the MNIST dataset. While the performance is not as good as the clean autoencoder it is able to clean out the salt-and-pepper noise very well, which is a much more difficult task than simply reconstructing the digits. Further testing in optimizing the hyper-parameters of the network can help to improve the reconstruction accuracy even further.

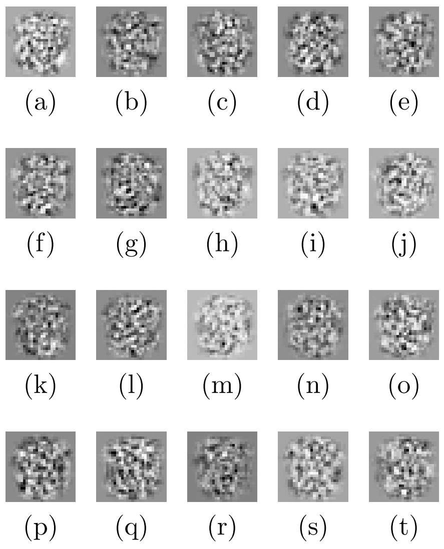


Figure 5: Clean features.

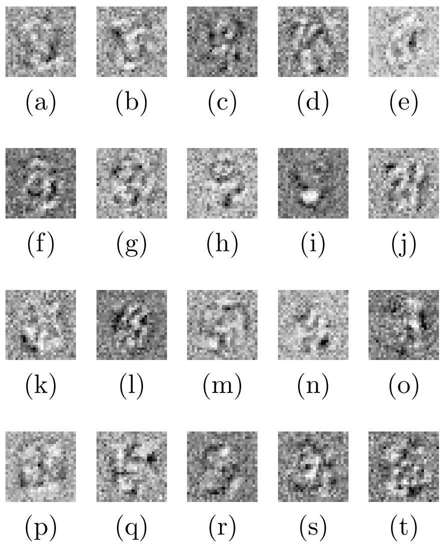


Figure 6: Denoising features.

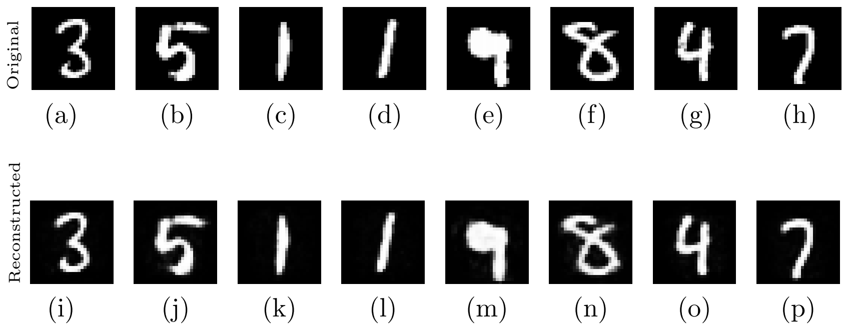


Figure 7: Original (top) and reconstructed (bottom) data points.

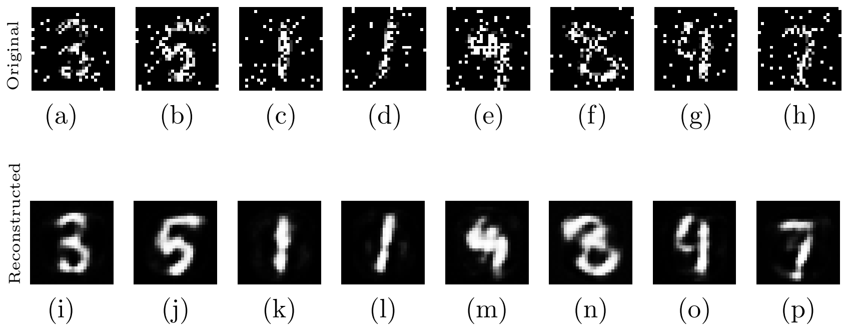


Figure 8: Original (top) and denoised (bottom) data points.

2 Problem 2

2.1 Problem Summary

The goal of this problem is to create a classifier using the features learned from the clean and denoising autoencoders trained in Problem 1. This is done by taking the weights of the hidden layers of the autoencoders and then randomly initializing weights for a classification layer for the output. During training, only the output weights are updated, so the network will not change the features learned by the hidden layer.

For the purposes of this report, the "clean classifier" refers to the classifier using the weights of the autoencoder with no noise applied, and the "noisy classifier" refers to the classifier using weights from the denoising autoencoder.

2.2 Results

2.2.1 System Description

Table 2 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 2: Classifier Training Hyper-Parameters

Parameter	Clean	Noisy	Description
$hidden\_layer\_size$	128	128	Neurons in hidden layer
η	0.01	0.01	Learning rate
α	0.8	0.8	Momentum
λ	0.0	0.0	Weight decay
max_epochs	500	500	Maximum training epochs
L	0.25	0.25	Lower activation threshold
H	0.75	0.75	Upper activation threshold
$patience$	5	5	Patience before early stopping
es_delta	0.0	0.0	Delta value for early stopping

This network uses the same weight initialization, early stopping, and weight decay as described in Problem 1. The difference here is that the values on the output layer are thresholded during training. If the neuron fires above H and its target is 1, the error is considered 0 for training. If the neuron fires below L and its target is 0, the error is also considered 0 for training.

2.2.2 Network Results

Throughout the duration of training, the loss of the training and validation sets was tracked every 10 epochs. The loss was calculated using $1 - balanced\_accuracy$ with a winner-take-all approach on the output layer. Figure 9 and Figure 10 show plots of the loss values while training the clean and noisy classifiers respectively. The vertical lines designate the point where the validation error is minimized. The clean classifier achieves a minimal test loss of 0.121 at epoch 80. The noisy classifier achieves a minimal test loss of 0.103 at epoch 40.

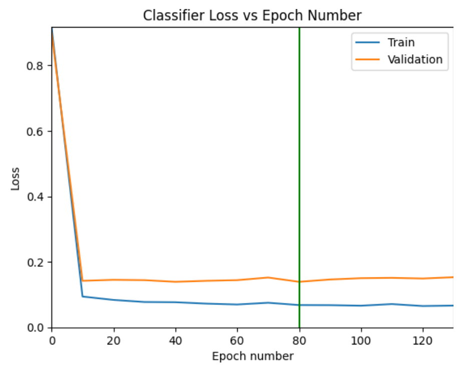


Figure 9: Training and validation loss of the clean classifier.

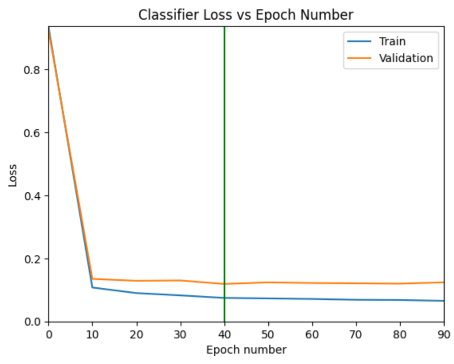


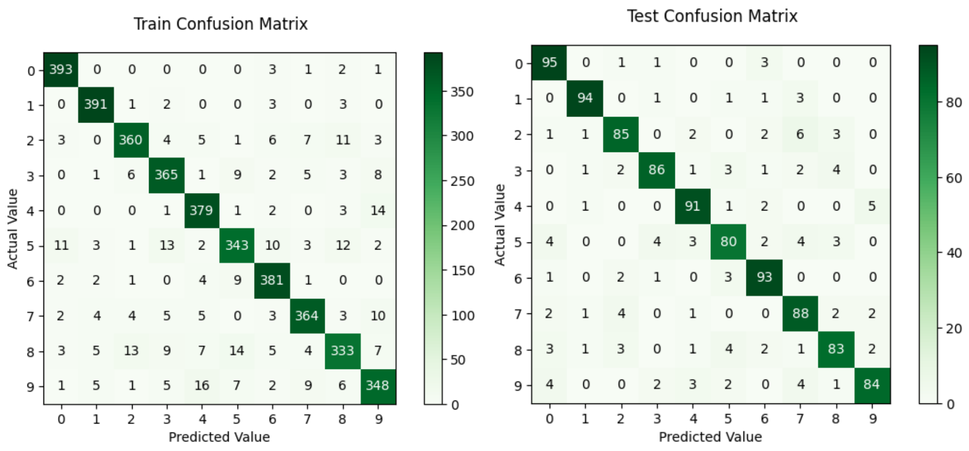
Figure 10: Training and validation loss of the noisy classifier.

Figure 11 and Figure 12 show the confusion matrices for the train and test sets of the clean and noisy classifier respectively.

2.3 Discussion and Analysis of Results

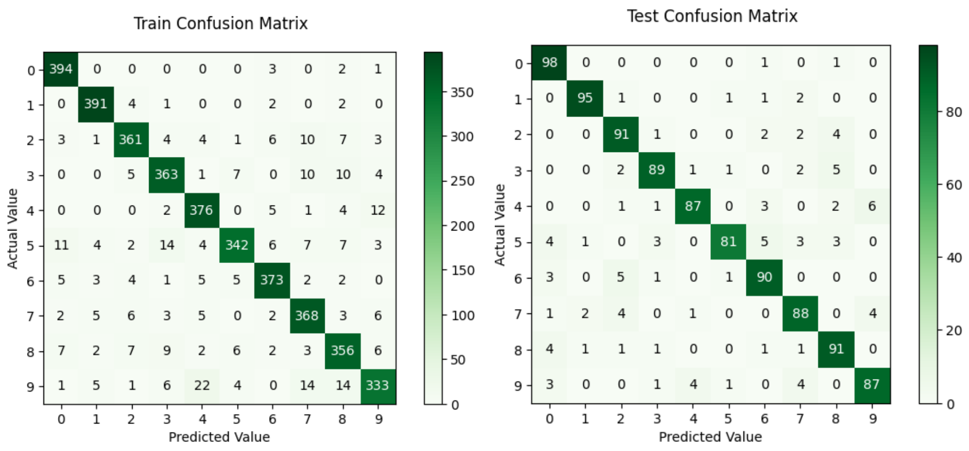
Overall, the results of both classifiers look pretty good. The diagonals of the confusion matrices in Figure 11 and Figure 12 are very dark, with hardly any coloring in the incorrect boxes. Many of the incorrect classifications are in two classes which look similar. For example, both networks had many mix-ups between 4s and 9s. In general, it looks like both classifiers were confused about the same classes. The noisy classifier performed better overall than the clean classifier. From an interpretability performance, this makes sense to me because the features learned by the denoising autoencoder in Figure 6 make much more sense than the features learned by the clean autoencoder in Figure 5.

The classification performance was not as good as the classifier from Homework 3, where a loss of 0.070 was achieved. This is likely because the classifier in Homework 3 is able to fine tune its features to the task of classification,



(a) (b)

Figure 11: Confusion matrices of the clean classifier.



(a) (b)

Figure 12: Confusion matrices of the noisy classifier.

while the features learned for this homework were specifically for image reconstruction. It may be possible to increase the accuracy by using the weights from the autoencoder to initialize the classifier, but allowing them to train.

2.4 Conclusion

Using the features learned by the autoencoders, the classifiers are able to effectively classify the MNIST dataset. While they do not perform as well as the classifier where all of the layers are trained, they train much more quickly and effectively demonstrate how the autoencoders' features are useful for distinguishing between different digits.