Joe Phaneuf

Computer Vision 16-720 Spring 2018 Homework 3

Mar. 7, 2018

1 Q1

1.1 Q1.1

In training deep networks, a ReLU activation can be chosen over a sigmoid as it is easily differentiable, and does not suffer slow training due to saturation.

1.2 Q1.2

Consider a one layer network with weights W^1 , bias b^1 , input X and activation function g. The output of this network is

$$out^1 = g(W^1X + b^1)$$

if g is a linear operator function, then there will be some modified weights and biases such that

$$out^1 = \hat{W}^1X + \hat{b}^1$$

For a subsequent layer, the pre-activation is

$$W^2(\hat{W}^1X + \hat{b}^1) + b^2$$

Which, due to the properties of matrix multiplication, can be reduced to some modified weights and biases

$$\hat{W^2}X + \hat{b}^2$$

$\hat{W^2}$ and $\hat{b}^2$ are of the same dimension as W^1 and b^1, meaning no additional discriminatory information can be encapsulated by the second layer.

A non-linear activation function makes it impossible to combine weights as demonstrated above, meaning each additional layer does add to the discriminatory power of the network.

2 Q2

2.1 Q2.1.1

Weight initialization is an important factor for training neural networks. Naively initializing weights to zero results in zero valued gradients, which prevents weight updates. Initializing with all equal weights also reduces the same gradients for each weight, so training does not function properly.

2.2 Q2.1.3

I chose to initialize weights from a uniform distribution ranging from -0.1 to 0.1. This was mostly from research and some experimention. Weights that are too large can create issues by way of massive gradients, weights that are too small can take a long time to train due to very small gradients.

2.3 Q2.4.1

Stochastic gradient descent is more robust to non-convex error manifolds , but may take longer to reach minima. Batch gradient descent will typically converge faster given error manifolds that are reasonably convex , but can be more expensive in terms of required RAM.

3 Q3

3.1 Q3.1.1

Figure 1 shows the training and validation accuracy and loss while training on the nmist26 dataset.

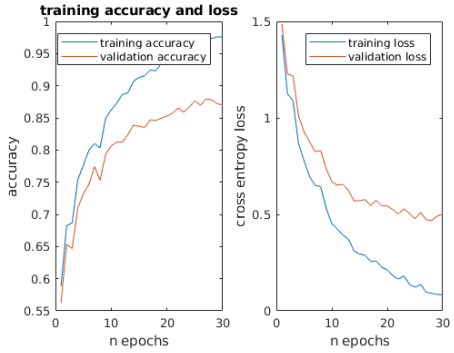

Figure 1: Training and validation accuracy and loss

3.2 Q3.1.2

Figure 1 above shows accuracy and loss for a learning rate of 0.01. Figure 2 shows accuracy and loss for a learning rate of 0.001. The higher learning rate of 0.01 performed better over 30 epochs, reaching a validation accuracy of 87 percent. versus the lower learning rate which achieved a validation accuracy of 73 percent. It is interesting to note that the accuracy and loss at the higher rate appear noisier. I suspect that too high of a learning rate could become problematic because of that.

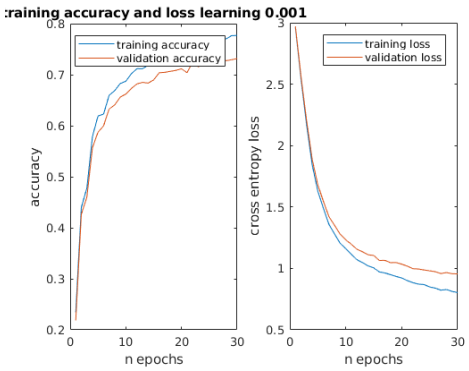

Figure 2: Training and validation accuracy and loss

3.3 Q3.1.3

As reported in section Q3.1.1 , this neural network provided the best valiation accuracy over 30 epochs using a learning rate of 0.01. Applying this network to the nist26 test data set resulted in 88.5 percent accuracy and a average cross entropy loss of 0.5. Figure 3 shows an image montage of weights in the first layer of this network. The first layer has 400 sets of 1024 length weight vectors.

Figure 5 shows the same montage with the weights initialized prior to training. The weights were initialized from a uniform distribution, and image appears to be white noise, which makes sense.

The learned weights clearly pick up on features that correspond to features of the training data set.

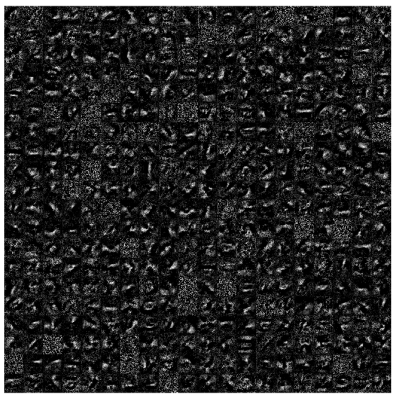

Figure 3: Visual montage of first layer weights after training

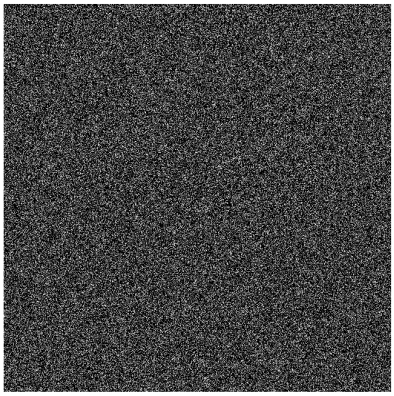

Figure 4: Visual montage of initial first layer weights

3.4 Q3.1.4

Figure 5 shows a graphical confusion matrix for the nist26 test data. Correct predictions are shown in blue, incorrect predictions are shown in red. Each point is plotted with a low alpha, so darker points show higher frequency of occurence. This indicates that the most problematic classifications were predicting I for a true label of Z , and predicting an O for a true label of D. Honestly, I have trouble with that myself, so I can't be mad at my neural net on this one.

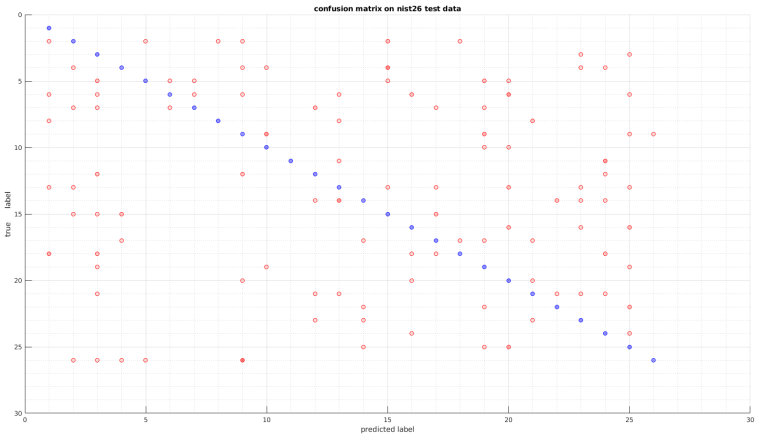

Figure 5: Confusion matrix for test data

3.5 Q3.2.1

3.6 Q3.2.2

Figure 6 shows the nist36 net's first layer weights after being initialized from nist26model60iters. Figure 7 shows the net's first layer weights after retraining with hand drawn numbers in addition to letters. Each weight image is different, in some cases only slightly different. This

is, however, difficult to parse with a feeble human brain. I was unable to pick out distinct features that are clearly associated with numbers. Applying the retrained net to the nist36 test data resulted in an accuracy of 78 percent and a loss of 0.71

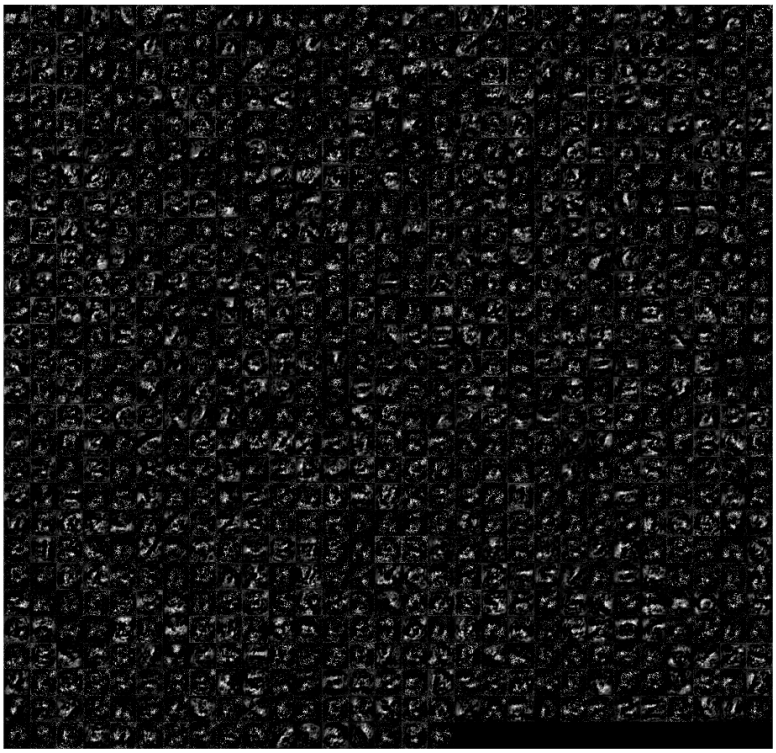

Figure 6: Visual montage of nist26 pre-trained weights

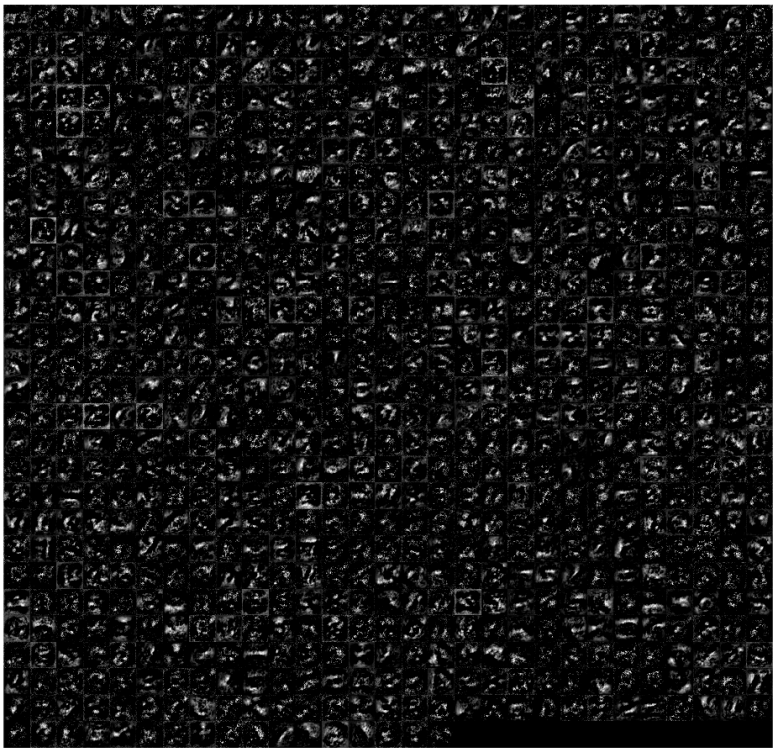

Figure 7: Visual montage of nist26 weights after retraining

3.7 Q3.2.3

Adding more classes to the neural net made classification more difficult and resulted in a decrease in test accuracy as compared to the nist26 data set performance. A confusion matrix for the nist36 performance is shown in Figure 9. The confusion matrix shows that this net often predicts an S for a true label of 5 , and also frequently predicts a 0 for a true lable of O. The latter is confusing for humans, and is really humanity's fault for picking symbols that look alike.

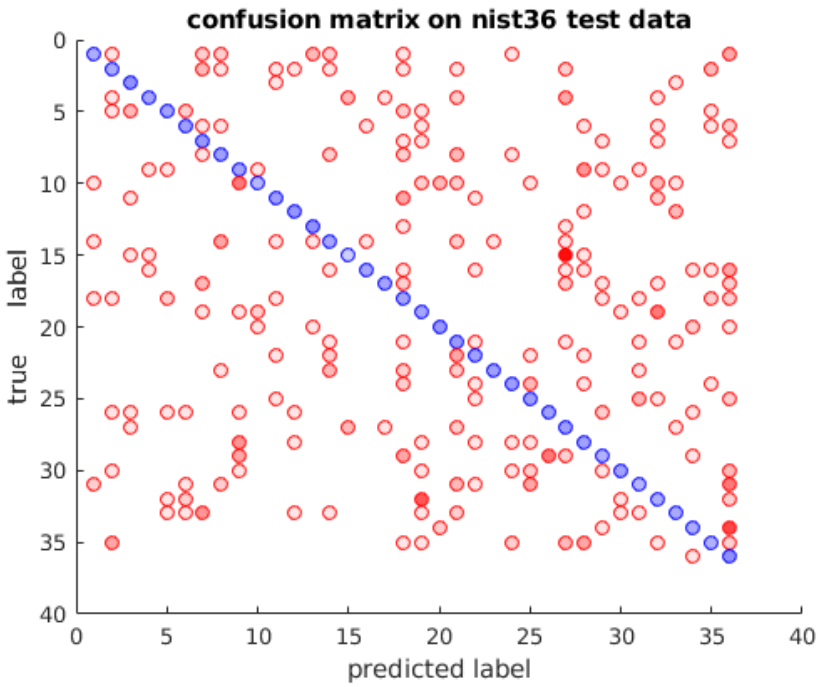

Figure 8: Confusion matrix for nist36 test data

4 Q4

4.1 Q4.1

The character extraction technique we will employ has a makes some assumptions for the sake of simplicity, but these assumptions will result in occasional errors. This method makes an assumption that the text will be dark and background light, which may not be the case. This method also assumes that characters will be laid out left to right on horizontal lines, which will not always be the case. Another critical assumption is that only valid characters are present, so punctation or other non-character markings may cause errors. https://stackoverflow.com/questions/43888023/how-to-extract-text-from-image-in-java https://www.welov elements-text-mix-vector/

5 Q5

5.1 Q5.1

The define autoencoder script does not reflect the architecture laid out in the homework writeup. Many of the input and output definitions between conolutional layers are incorrect. I modified each layer definition to match the proposed homework writeup architecture.

5.2 Q5.2.1

Using the default settings, I noticed that the autoencoder was unable to converge. With a tweak to the learning rate, the loss and RMSE decreased for 3 epochs, but not close enough to desired RMSE.

5.3 Q5.2.2

To drive the error lower, I adjusted the weights in the conv2D weight initialization to a standard deviation of 0.5, adjusted the learning rate to 5e-4 , applied L2 regularization with value 1e-2, and dropped the mini batch size to 5. Figure 9 shows the loss and RMS error over 3 epochs of training.

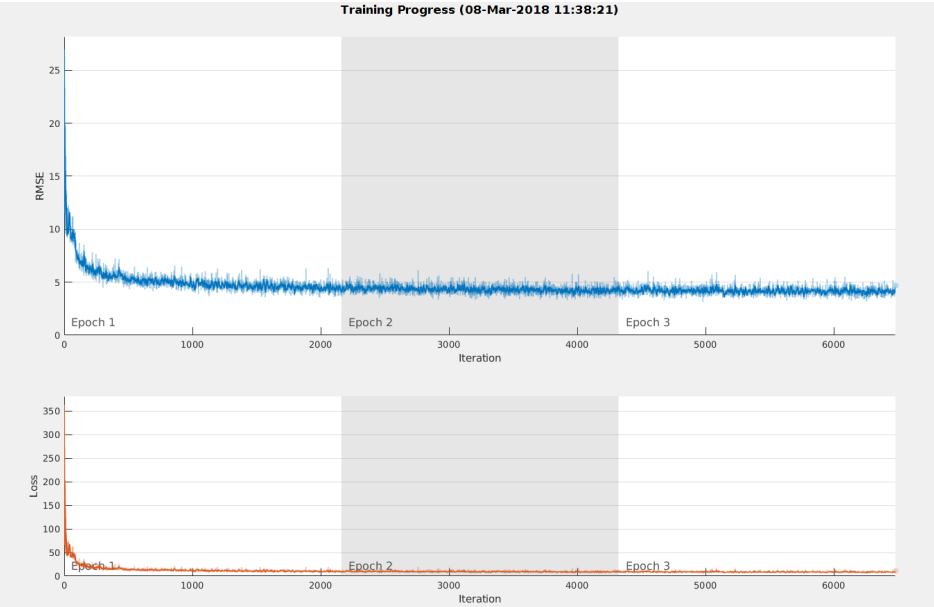

Figure 9: Loss and RMSE for autoencoder training

5.4 Q5.3.1

The autoencoder outputs (shown in Figure 10) show noisy versions of the original input images, they are however visibly similar to the originals.

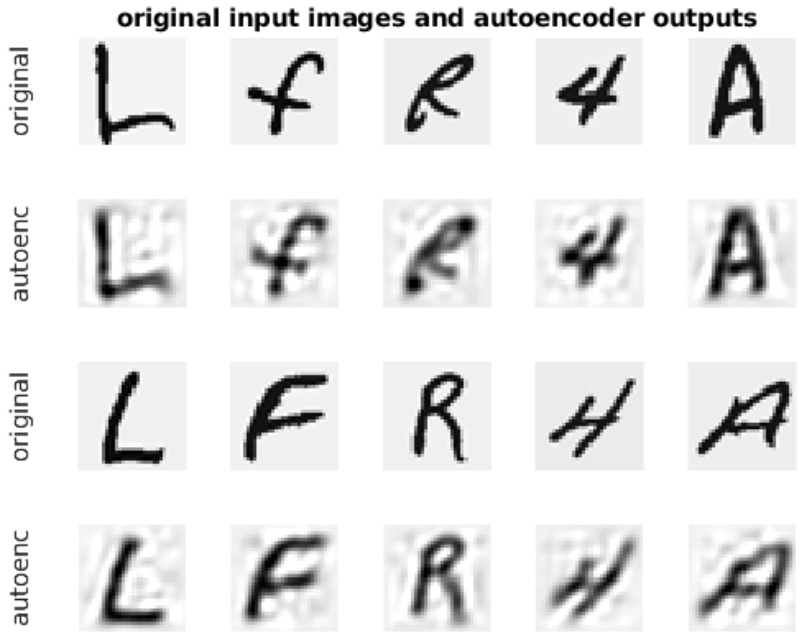

Figure 10: Loss and RMSE for autoencoder training

5.5 Q5.3.2

I looped across all images in the test set and applied the autoencoder to each, while accumulating the peak signal-to-noise ratio for each image/encoded image pair. The average value of the PSNR was 18.0117 .