CSE803 HW4

Javen W. Zamojcin
zamojci1@msu.edu

2024, October 31

1 Question 1: Optimization and Fitting

For hyper-parameters, I chose a learning rate of 0.00001 and 10,000 epochs. The resulting S and t matrices were as follows:

$$S = \begin{bmatrix} 1.00008565e+00 & -5.38571036e-03 \\ 6.62493948e-05 & 6.14124829e-01 \end{bmatrix}$$

$$t = \begin{bmatrix} -0.00800811 & -0.02994507 \end{bmatrix}$$

See Figure 1 for the resulting point affine transformations.

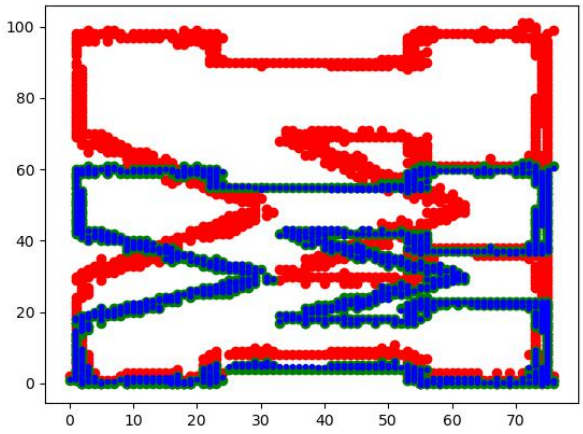


Figure 1: Q1 Points Case

2 Question 2: Softmax Classifier with One Layer Neural Network

2.1

The architecture I chose for this part involved setting up cross-validation for hyper-parameter configurations, the `SoftmaxClassifier` class for 1-layer networks, a function for converting the training history and parameters into a plot, a modified `train()` function for the cross-evaluation runs, the `softmax_loss()` function, and finally the `relu_forward/backward()` functions.

The `softmax_loss()` function implements the softmax equation to calculate the Cross Entropy Loss. The gradient is then calculated using the softmax equation applied to the input x, and averaged by the number of input samples. The `relu_forward/backward()` functions simply apply the the ReLU equation to either the input or input derivative, respectively.

For the 1-layer `SoftmaxClassifier`, logic is added to forward-feeding and backward-feeding without any hidden layers; meaning, there is only one layer of weights and biases. The ReLU functions are also not applied at this point for either the forward or backward pass. The total loss is calculated as the sum of the softmax loss and the regularized sum of squared weights. The gradients are also summed with their regularized weights.

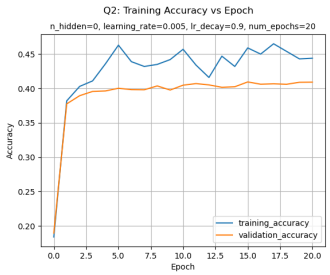
2.2

For this part, I ran three different model experiments to achieve the minimum testing accuracy of 40.0% (see Figure 2). All three had 0 hidden layers, a batch size of 128, and a regularization term of 0.0.

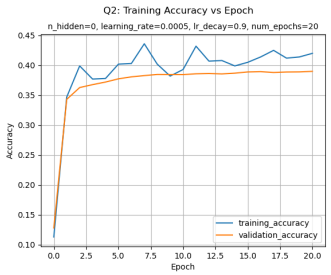
Model-1 had a learning rate of 0.005, a learning decay of 0.9, and ran for 20 epochs. This achieved a test accuracy of 0.411. This model had the best and quickest performance, considering the limited number of epochs. The larger learning rate and faster learning decay seem to work best for this task.

Model-2 had a smaller learning rate of 0.0005, a learning decay of 0.9, and ran for 20 epochs. This achieved a test accuracy of 0.3945, notably worse than the previous model. The smaller learning rate significantly hurt the performance at the beginning, affecting the total performance. Interestingly though, this model seem to have the best training-validation ratio of the three.

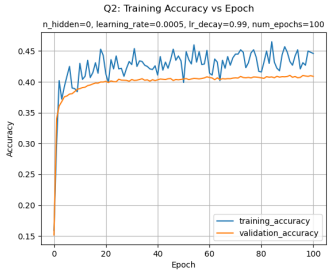
Model-3 also had a learning rate of 0.0005, a slower learning decay of 0.99, and ran much longer for 100 epochs. This achieved a test accuracy of 0.4112, slightly better than the first model and the best of all three. It seems the additional epochs only contributed diminishing performance gains over time, not worth the extra computation time.



(a) Model-1



(b) Model-2



(c) Model-3

Figure 2: Q2 Model Training

3 Question 3: Softmax Classifier with Hidden Layers

3.1

The only architecture difference in this part was that the `SoftmaxClassifier` class was modified to allow hidden layer networks. If the number of hidden nodes are specified, logic is ran to add a second layer of weights and biases. The input layer connects to the first weight layer, which connects to the hidden second layer, and the second layer connects to the output layer. Additionally, for forward-feeding, the hidden layer utilizes the `relu_forward()` activation function and a second dot product between the hidden and second weight layers.

For the backward pass gradients, the second weight layer is calculated from the hidden layer and derivative of the scores. The first weight layer first calculates the derivative of the hidden layer using the `relu_backward()` function, then takes the dot product between this and the image inputs.

3.2

For this part, I ran six different model experiments to achieve the minimum testing accuracy of 50.0% (see Figure 3). All had a varying number of hidden layer nodes, a batch size of 128, and varying regularization terms but mostly 0.0.

Model-1 had 300 hidden layer nodes, a learning rate of 0.05, a learning decay of 0.95, ran for 20 epochs, and had a regularization term of 0.0. It achieved a final testing accuracy of 0.5274. While the validation accuracy only stayed around the 0.52 range, the training accuracy fluctuated greatly to above 0.70. A very large training-validation ratio.

Model-2 also had 300 hidden layer nodes, a learning rate of 0.05, a learning decay of 0.95, ran for 20 epochs, but had a regularization term of 0.1. It achieved a final testing accuracy of 0.4431, significantly worse than with no regularization term. It had a very smooth validation accuracy curve though, and a smaller ratio with training accuracy.

Model-3 had 500 hidden layer nodes, a learning rate of 0.05, a learning decay of 0.95, ran for 20 epochs, and had a regularization term of 0.0. It achieved a final testing accuracy of 0.5296, the best of all model evaluations. The higher number of hidden nodes allowed for better testing accuracy, but we also see a large divergence between training and validation/testing accuracy; possibly beginning to overfit the training data.

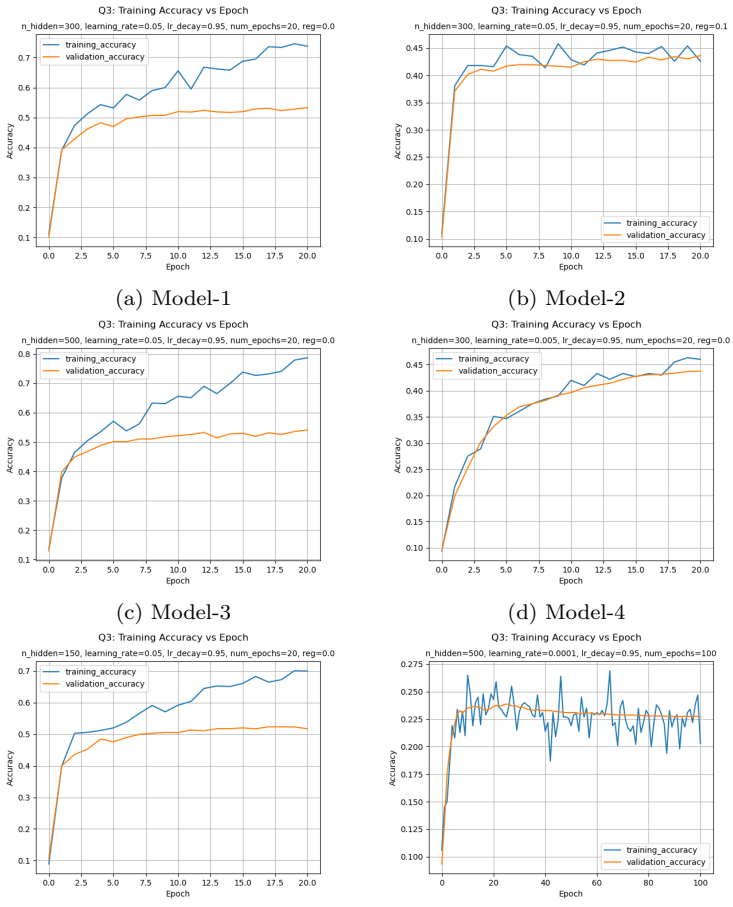
Model-4 had 300 hidden layer nodes, a smaller learning rate of 0.005, a learning decay of 0.95, ran for 20 epochs, and had a regularization term of 0.0. It achieved a final testing accuracy of 0.4468, worse than most other models because of the slower learning rate. We do see however a very smooth curve for both training and validation accuracy.

Model-5 had 150 hidden layer nodes, a learning rate of 0.05, a learning decay of 0.95, ran for 20 epochs, and had a regularization term of 0.0. It achieved a respectable final testing accuracy of 0.5149, and also evaluated much quicker.

Model-6 had 500 hidden layer nodes, a very slow learning rate of 0.0001, a learning decay of 0.95, ran for 100 epochs, and had a regularization term of 0.0. It achieved a miserable final testing accuracy of 0.228, despite the large number of hidden nodes and epochs. The very small learning rate destroyed the performance. Interestingly, we see at many times the training accuracy dipping below validation accuracy.

4 Question 4: Fooling Images

The trained model I loaded for this part was Model-3, the model with a 500 node hidden layer and a testing accuracy of 55%. The initialization image I chose that my model predicted correctly had a class label 7, the horse. The fooling class I chose was 176. It took 86 iterations with a learning rate of $1e-2$ to generate the fooling image. See Figure 4.



(a) Model-1 (b) Model-2

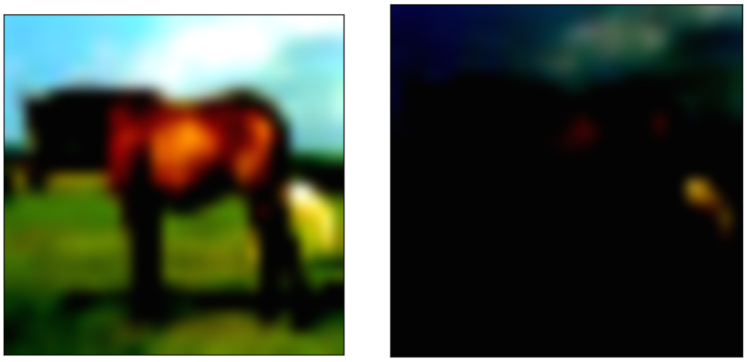
(c) Model-3 (d) Model-4

(e) Model-5 (f) Model-6

Figure 3: Q3 Model Training

Interestingly, the difference between the fooling image and the original, is that the fooling became much darker but not universally black. This was achieved using gradient ascent, or maximizing the class probability values for the target class.

I believe the robustness of my model made it difficult to generate a fooling image. Initially, I had chosen a target class of 3, but failed to generate a fooling image after $10,000$ iterations, using both gradient directions. For these failed experiments, I printed and monitored the class probabilities to confirm they were approaching their correct respective directions, just very slowly.



(a) Original Image (b) Fooling Image

Figure 4: Q4 Image Fooling