EECE6036 HW5

Jack Fliegel

# Problem 1

**System Specification**

Problem 1 used 250 neurons in its hidden layer. This number was chosen as less neurons resulted in a higher error rate, and it was harder for the model to distinguish the numbers at the end of the model. A learning rate of 0.2 was used and many values were tested in combination with different numbers of epochs as too little of a learning rate could not train the model at all and too high a learning rate overcorrects and trains the model too quickly. The initial weights were chosen at random. The criterion for training to stop is only when the number of epochs which is 200 which is where the model begins to plateau. A batch size of 200 data points was used to speed up training and to randomize inputs. A SoftMax activation was used to determine the output neuron with the highest output neuron being the activated one. A momentum of 0.94 was used which helped the model escape from local minimums and achieve higher overall accuracy. The loss function used for training was the J2 loss equation below:

A mathematical equation with numbers and symbols

Description automatically generated

Equation : J2 Loss Function

**Results**

**Analysis**

*Q1: Did initializing the hidden weights from the autoencoder make training go faster in this this homework compared to that in HW 4 (for both Case I and Case II)?*

Initializing the hidden weights made the training much faster for Case I and Case II compared to HW4. As we can see in figure 1 the model has about 90% accuracy within 5 epochs compared to 90% accuracy at 50 epochs for HW4’s model where the weights were set at random.

*Q2: Did training both layers (Case II) substantially improve performance over training only the output*

*layer (Case I)?*

There was a slight difference between just training the output layer versus both input and output layer.

# Problem 2

**System Description**

I kept all the system parameters the same as they were in problem 1. Learning rate = 0.2, epochs = 200, momentum = 0.94. The same mean squared loss function was shown in Equation 1.

**Results**

**A graph with a line graph

Description automatically generated**

Figure : Training/Test Loss vs Epochs

**A close-up of several images

Description automatically generated**

Figure : Grayscale Features of 20 random neurons

**Discussion**

Some of the neuron images show clear, stroke-like patterns that look like parts of handwritten digits. For instance, certain neurons show shapes similar to the curves found in "0", "6", "8", or the vertical lines found in "1", "4", "7". These neurons are able to recognize specific features of digits, such as edges, curves, and intersections. On the other hand, some neurons look noisy which could mean that neurons are not well trained or that the features it identifies are more abstract.

The loss time series is very similar to Problem 1 with the shape of a negative exponential function. It still looks like the model is avoiding local minimums for the reconstruction task.

I did change the system parameter values to see how this would affect this model just as I did in problem 1. I decided to keep the same values for consistency even though they are not the most efficient and do not get the loss as low as it could be. Adding more epochs helped lower the loss further and increasing the momentum closer to 1 (but still less than) did the same.

**AI Policy**

ChatGPT 4 September 22 Version was used during this assignment. I watched a YouTube series called “Neural Networks from Scratch” by Sentdex which walked me through the structure of setting up multiple layer networks. I then put this base code which had forward propagation and a loss function into ChatGPT so that it had a foundation of code to work from. I also asked ChatGPT to create multiple graphs including the confusion matrix. In addition I had ChatGPT add momentum into my code for when I update the biases.

The first prompt was: “How can I add logic to this code so that the weights will update between each layer and so that The classes can be divided. \*\*\*Code previously wrote\*\*\* “

Result:

class Layer:

# ... existing code ...

def backward(self, dvalues):

# Gradients on parameters

self.dweights = np.dot(self.inputs.T, dvalues)

self.dbiases = np.sum(dvalues, axis=0, keepdims=True)

# Gradient on values

self.dinputs = np.dot(dvalues, self.weights.T)

class ActivationReLU:

# ... existing code ...

def backward(self, dvalues):

self.dinputs = np.array(dvalues, copy=True)

self.dinputs[self.inputs <= 0] = 0

class ActivationSoftmax:

# ... existing code ...

def backward(self, dvalues):

self.dinputs = np.empty\_like(dvalues)

for index, (single\_output, single\_dvalues) in enumerate(zip(self.output, dvalues)):

single\_output = single\_output.reshape(-1, 1)

jacobian\_matrix = np.diagflat(single\_output) - np.dot(single\_output, single\_output.T)

self.dinputs[index] = np.dot(jacobian\_matrix, single\_dvalues)

class Loss\_CategoricalCrossentropy(Loss):

# ... existing code ...

def backward(self, dvalues, y\_true):

samples = len(dvalues)

labels = len(dvalues[0])

if len(y\_true.shape) == 1:

y\_true = np.eye(labels)[y\_true]

self.dinputs = -y\_true / dvalues

self.dinputs = self.dinputs / samples

When I got to problem 2 I had ChatGPT clean up what I did for problem 1 and put it into a single class called Autoencoder so that I could more easily train the model. There were lots of prompts that I gave ChatGPT to clean up my code over the course of several days. I found taking each problem step by step was more useful that asking ChatGPT to complete everything all at once and them fixing it.