**CSC 4760 – Homework 8**

**Objectives:**

1. Design a recurrent Neural Network for handwritten digits classification
2. Show the code, show the training and testing information, and show the test results.

**Results:**

The plan was to review the sample model, comment its code, and see how it could be improved. To determine if there was actually any improvement, I decided to run the model first.

A white background with blue and red text

Description automatically generated

Figure 1 – Test Results For The Model

A screenshot of a computer program

Description automatically generated

Figure 2 – Attempting to Add Another LSTM layer initialized with zeroes.

My next thought was: If one LSTM layer is good, two must be better! So I created a new LSTM layer with a larger hidden size, and initialized its internal memory with zeroes. However, this did not lead to better results, in fact it lead to much worse results:

A screenshot of a computer

Description automatically generated

Figure 3 – Test results from adding an additional LSTM layer, please note that the accuracy has dopped to 9.58% - which assuming random distribution of digits to be identified means that my model is performing WORSE than just guessing.

I didn’t want to simply increase the number of layers on the NN.LSTM() function, that felt lazy. So I concocted a very silly idea: what if I simply created double the layer as before, and concatenate the input \* n instead of using zeroes? GENIUS

A screenshot of a computer program

Description automatically generated

Figure 4 – Attempt to improve model by using the first LSTM layer as training for a second LSTM layer. This did not work – this is because feeding training data between layers relies on layers being the same size – simply doubling the testing information using concatenation does not work. Deep down inside I knew this wouldn’t work – but I had a sliver of hope that the order of weights for the neurons wasn’t as relevant than the initialization of values. Would the order of these values be maintained if I instead shrunk the second layer? I’m guessing the issue would remain the same.

A screenshot of a computer

Description automatically generated

Figure 5 – It should be noted that despite this being a terrible idea – accuracy did improve by comparison of simply reusing some information – showing that some values seems to produce more results than zeroes. I refused to give up – I already built it, I might as well play with the numbers. Instead of doubling the number of layers, why not multiply it by 5? I’m sure that will change the output.

A screenshot of a computer

Description automatically generated

Figure 6 – There is an improvement in accuracy! It’s miniscule, but it is an improvement – this teaches me that sometimes increasing the size of the hidden layer is beneficial – but overall maintaining learning information is more important.

A screenshot of a computer

Description automatically generated

Figure 7 – Results when I put everything back together to normal the way the sample was. Boo. WAIT A SECOND. WHY DIDN’T I JUST DO THE OBVIOUS (If you haven’t figured out already, I write my reports concurrently to my labs)

A screenshot of a computer program

Description automatically generated

Figure 8 – In this design reattempt I decided to maintain the internal values of the hidden layers across training passes. However, if you do not detach the tensors – the computer will attempt to calculate the differences between training passes, and throw an error because it’s the same variable that hasn’t changed.

A screenshot of a computer program

Description automatically generated

Figure 9 – Test results from this new construction – wow look at that, a 0.3% improvement from putting everything back to normal. However, I wonder if this is in part due to the shuffle function for the batch input, because it you note that there was a 0.4% difference in test accuracy between the first time I tested the sample code, and when I ran it with the alterations.

A white background with black text

Description automatically generated

Figure 10 – I ran it again, as I suspected, the accuracy fluctuated due to the shuffle nature of the batch input. I am very disappointed in myself – I thought that retaining information regarding the inputs between states would improve accuracy – but it appears to have had minimal impact.

A screenshot of a computer

Description automatically generated

Figure 11 – Code lines 1-35: Notes to self regarding objectives and strategy, library importing.

A screenshot of a computer program

Description automatically generated

Figure 12 – Code lines 36-64: Device configuration and parameter definition.

A screenshot of a computer program

Description automatically generated

Figure 13 – Code lines 65-97: Data acquisition and dataloader creation. This is where that shuffle function caused me falsely to believe that I was getting gains from my data – when in reality I was not.

A screenshot of a computer program

Description automatically generated

Figure 14 – Code lines 99-163: Model definition, featuring my attempt to retain information between passes without relying upon gradient descent. I don’t know why I thought it would work – this just means that the LSTM has no way of testing itself because by detaching its parameters the weights and accuracies were never corrected using the loss function (I am not sure if this is true, but I’m pretty sure this is what was happening).

A screenshot of a computer program

Description automatically generated

Figure 15 – Code lines 164-224: Training and testing section