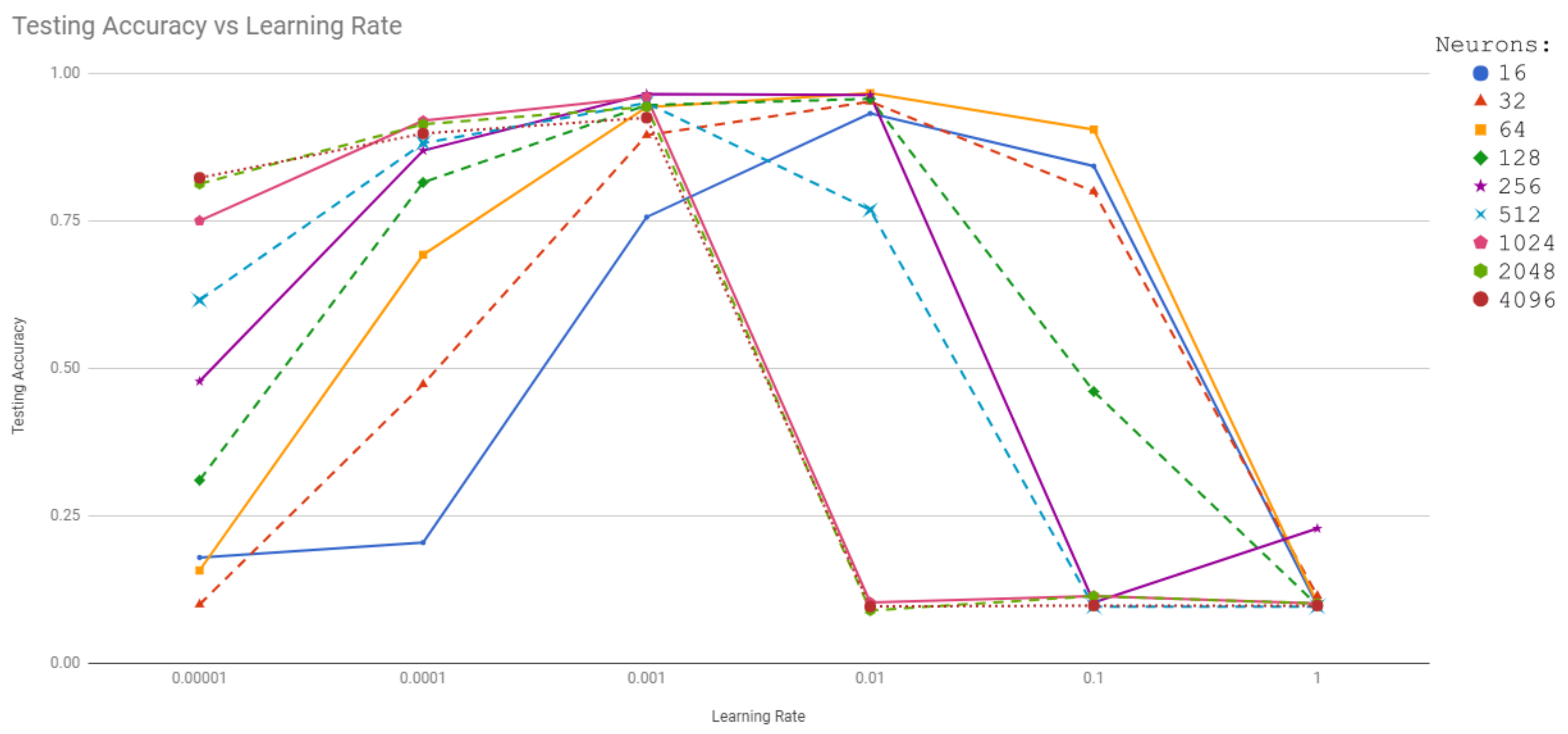
Homework 2
CMPT469 Deep Learning w/ TensorFlow

Kai Wong

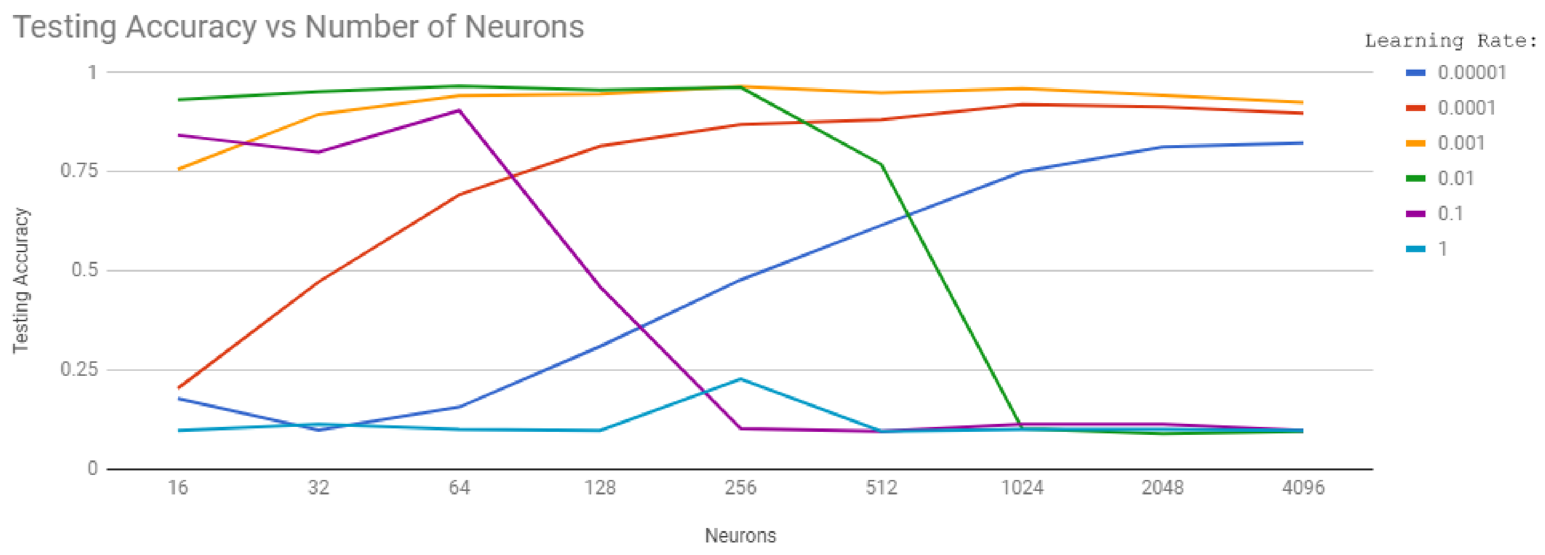
10/23/17

1 Here is what you need to do: you will vary the number of hidden units exponentially as follows, 16, 32, 64, 128, 256, 512, 1024, 2048, and 4096; you will also vary the learning rate exponentially as follows: 0.00001, 0.0001, 0.001, 0.01, 0.1, and 1.0.

Neurons/Learning Rate	0.00001	0.0001	0.001	0.01	0.1	1
16	0.1789	0.2042	0.7562	0.9319	0.8426	0.0974
32	0.0992	0.4726	0.895	0.952	0.7996	0.114
64	0.157	0.6923	0.9425	0.9663	0.9047	0.101
128	0.31	0.815	0.9461	0.9567	0.4604	0.098
256	0.4777	0.8691	0.9649	0.9631	0.1028	0.2281
512	0.6151	0.8814	0.9496	0.7684	0.0958	0.0958
1024	0.75	0.9195	0.96	0.1028	0.1135	0.101
2048	0.8126	0.9136	0.943	0.0892	0.1135	0.1009
4096	0.8228	0.8979	0.9247	0.0958	0.0974	0.0974



Testing Accuracy vs Learning Rate
Neurons:
● 16
▲ 32
■ 64
◆ 128
★ 256
✕ 512
⬟ 1024
⬢ 2048
● 4096
Testing Accuracy
1.00
0.75
0.50
0.25
0.00
0.00001
0.0001
0.001
0.01
0.1
1
Learning Rate



Testing Accuracy vs Number of Neurons
Learning Rate:
0.00001
0.0001
0.001
0.01
0.1
1
Testing Accuracy
1
0.75
0.5
0.25
0
16
32
64
128
256
512
1024
2048
4096
Neurons

1.1 What is this telling you about the training of an LSTM with respect to the number of neurons or learning rate or both? What can you learn from this?

1.1.1 Learning Rate

As learning rate increases across all different neurons, testing accuracy goes up and reaches a peak after a certain learning rate, then decreases drastically after passing the peak. Why does this happen?

```
optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate).minimize(cost)
```

We can see that the optimizer is based off of the learning rate, which is how fast the derivative adjusts itself. Basically, the learning rate is the size of the steps the gradient descent optimizer takes in order to find the minimum error in a loss function. It does this by taking steps toward the lower point of the loss function, and reversing when it starts taking steps away from the lower point. The testing accuracy goes up as learning rate increases because the optimizer can find the lowest point in the loss function faster and use that to reduce the error in testing. However, if the steps the optimizer takes are too large, it can continuously miss the lowest point in the loss function. Ideally, we want to start with a big learning rate initially, so that the optimizer takes big steps until it finds a good spot to be in. Most importantly, we want to decay this learning rate over time, so that we can start taking smaller steps to find the lowest point. If large steps are continuously taken, the lowest point will be extremely hard to find, as the optimizer will keep on overshooting the low point. In this code, we don't decay the learning rate over time as the training progresses. Therefore, with too large of a learning rate, the optimizer can never find the lowest point in the loss function and reduce the error in testing. Therefore, the testing accuracy is very low and minibatch loss very high for high learning rates.

1.1.2 Neurons

As we increase the number of neurons for a learning rate of 1, the testing accuracy starts out horrible, and does not improve in the slightest. This is because the steps taken by the gradient descent optimizer are too large, so the loss function is never minimized, thus the error in testing is large.

As we increase the number of neurons for a learning rate of .1 and .01, the testing accuracy starts out and stays great for a few increments, but after hitting a certain number of neurons, the testing accuracy drops severely. .01's certain number of neurons is higher than .1's certain number of neurons. With a few neurons and a relatively high learning rate, we can see that a network isn't impacted too hard. This is probably because the optimizer can effectively find the lowest point in the loss function to reduce the error in testing. However, once the amount of neurons increases past a certain point, the testing accuracy drops severely, because at this point, we run into the phenomenon known as overfitting, where the network has such a large information processing capacity that there is not enough information to train all the neurons, or that the model learns the detail and noise to such an extent that it negatively impacts the performance of the model (as the model starts to pick up noise as concepts and learns from it). This causes the loss function to worsen, and thus makes it harder/impossible for the optimizer to find the lowest point in the loss function, especially in this instance where the optimizer's step size is already quite large. We can see this by comparing the number of neurons where the accuracy drops. The testing accuracy for a learning rate of .01 drops at 256 neurons, while the accuracy for a learning rate of .1 drops at 64 neurons; this shows how an optimizer can handle more neurons in a network with a lower learning rate, as opposed to one with a higher learning rate, as it can minimize the loss easier with a smaller learning rate.

As we increase the number of neurons for a learning rate of .001, the testing accuracy starts out fairly accurate, and improves a little bit. This shows that this learning rate is good for the network, for as we increase the network's ability to process information (neurons), the optimizer can still minimize the loss function effectively in order to reduce testing error.

As we increase the number of neurons for a learning rate of .0001 and .00001, the testing accuracy starts from very inaccurate and gets more accurate until it's fairly accurate. A learning rate of .00001's test accuracy with 4096 neurons is slightly less accurate than .0001, and .001's testing accuracy is the best. This happens most likely because if we start with a low learning rate and low number of neurons, the optimizer will never reach the minimal point in the loss function because the steps it takes are too small. Additionally, the low number of neurons means there is less information (embedding) about the data in the network, resulting in an underfitting of the network, where there is not enough neurons in the hidden layers to detect signals and make an accurate test prediction. Even with 4096 neurons, a learning rate of .00001 and .0001 are never quite as accurate as .001, due to the optimizer never being able to quite reach the lowest point in the loss function and minimize the error the most. A learning rate of .001 is the “Goldilocks” in this network.

We can also see that in 0.00001, 0.0001, and 0.001 learning rates, the more neurons there are, the higher the testing accuracy is. This is due to the fact of how more neurons means a recurrent neural network will have more embedding and be able to pass more information between each hidden layer. The more information a network has, the more accurate it will be and will make better predictions.

We can also see that the fewer neurons there are, the less it’s impacted as the learning rate approaches 1. This most likely has to do with the number of neurons itself; with fewer neurons, there is less error to propagate within the recurrent neural network, thus the testing accuracy remains relatively high until it reaches 1. For example, 16, 32, and 64 neurons’ testing accuracy aren’t impacted too heavily by the large learning rate (until hitting a learning rate of one)