Assignment 4

Armando Ordorica armando.ordorica@mail.utoronto.ca
1005592164

1. Non-Linearities

Modify the learning rate of your mini-batch SGD and report (a) a value that results in slow convergence, (b) a value that results in oscillations but still converges, and (c) a value that results to instabilities and diverges. For each value, copy-paste the training log (i.e., the list of accuracies achieved after each epoch) in the PDF you hand in on Quercus.

1.0.1 A value that results in slow convergence

$\alpha = 0.00001$

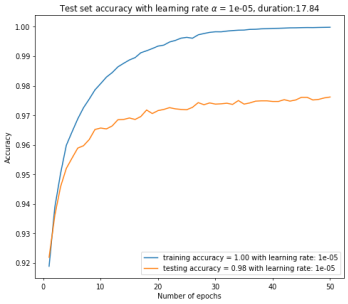
1.0.2 A value that results in oscillations but still converges

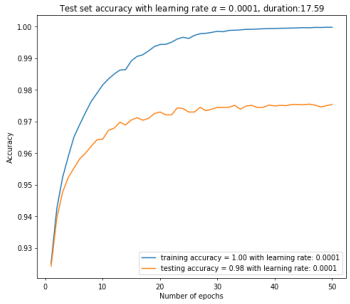
$\alpha = 0.15$

1.0.3 A value that results to instabilities and diverges. For each value, copy-paste the training log

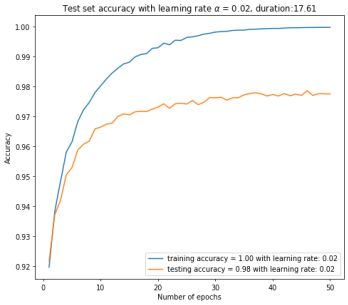
$\alpha = 10$

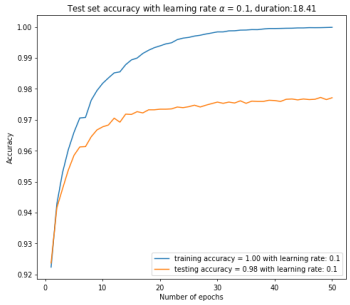
Other examples of each of these cases are shown below. In general, the lower the alpha, the slower the convergence. The higher the alpha, the more likely it is to oscillate. If the alpha is large but not too large, it is likely to result in oscillations but still converge.



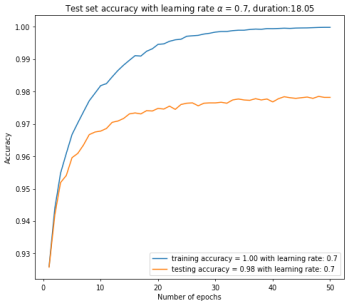


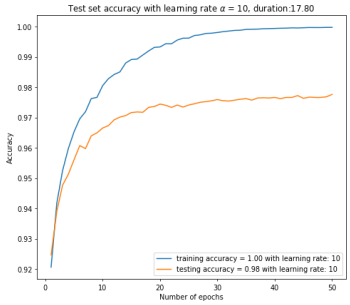
oscillations but converges





Slow convergence



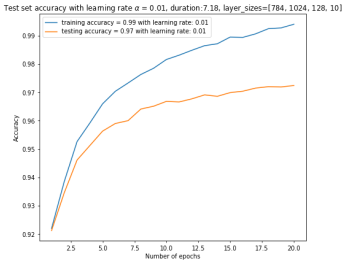


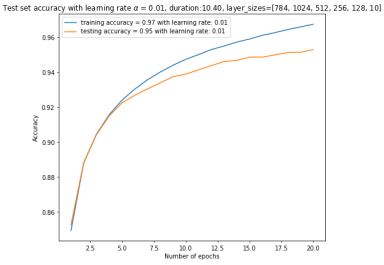
Unstable and diverges

1.1 Modify the number of neurons and/or number of layers that make up the architecture of your neural network. You can do so by modifying the list layer sizes. Report a list of integers that results in a neural network that underfits the data.

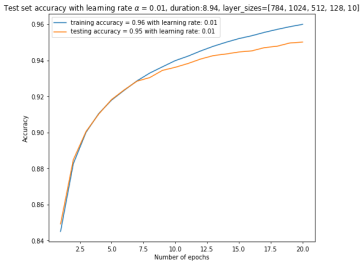
When layer_sizes = [784, 128, 128, 10], batch_size = 256, the model underfits the data, where training accuracy is the same as testing accuracy of 92%. Indeed, testing accuracy is slightly higher as shown on the plots in the attached pdf.

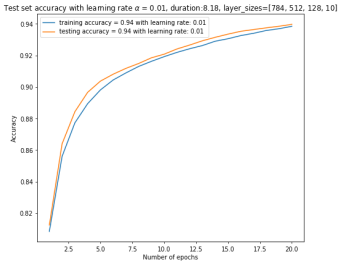
Test vs training accuracy as we change the list of layer sizes:

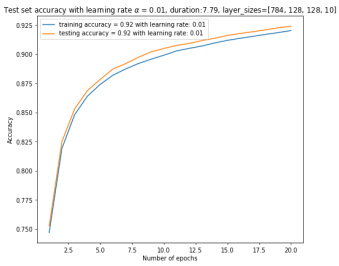




Overfits the data (Divergence between test and train)







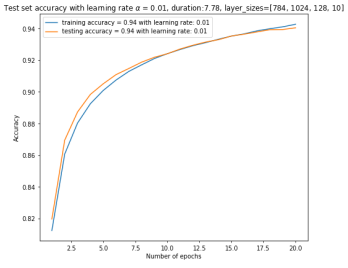
Underfits the data (low accuracy and testing is higher than training)

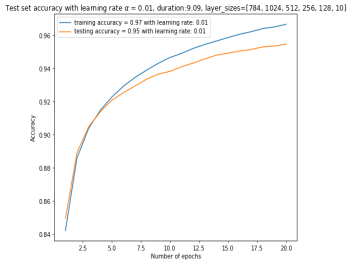
1.2 Find a setting in which your neural network overfits. To this end, modify the set of hyperparameters discussed so far, i.e., learning rate and architecture, as well as the number of epochs if that's necessary. You may also find it useful to set create outliers to True and reload the MNIST data by executing mnist() again. This will mislabel half of the training data, which makes it easier to find a setting in which the model overfits. Report the set of hyperparameters that result in a neural network that overfits the data

Overfitting the data with $\alpha = 0.01$, batch_size = 256, layer_sizes = [784, 1024, 128, 10]. We can see this because the training accuracy is significantly higher than the testing accuracy in a divergent manner.

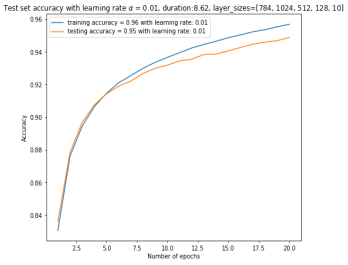
In general, overfitting can be observed when there is a divergence between training and testing set, where high accuracy is attained for the training set and not as high for the testing set. In contrast, underfitting can be observed when testing set is indeed higher than training set.

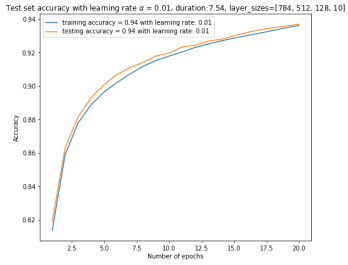
Test vs training accuracy under constant learning rate but varying layer sizes after introducing outliers





This configuration overfits the data
(training and testing accuracies diverge)





2. Modify model architecture to train a convolutional network.

2.1 Update the cell which defines a stax.serial model to replace some of the fullyconnected layers (they are called Dense layers in stax) by the conv+maxpool layer pairs we studied in class. A convolutional layer is defined using stax.Conv and a maxpool layer using stax.MaxPool. You will also need to insert a ReLU non-linearity with stax.Relu

```
In [0]: init_random_params, predict = stax.serial(
            stax.Conv(32, (3, 3), padding='SAME'), stax.Relu,
            stax.Conv(120, (5, 5), (1,1), padding='VALID'), stax.Relu,
            stax.MaxPool((2, 2)), stax.Flatten,
            stax.Dense(128), stax.Relu,
            stax.Dense(10), stax.LogSoftmax,
        )
```

Figure 1: Architecture that yields the accuracy required.

2.2 Report an architecture (you can copy-paste the stax.serial model definition in the PDF you hand in on Quercus) and set of hyperparameters (learning rate, batch size, number of epochs) that allow you to train a convnet with at least 99% test accuracy. Also report the exact accuracy you achieve (mean and standard deviation over 5 runs).

The values for the other hyperparameters were the following:

- learning rate = 0.18
- batch size = 128
- number of epochs = 10

```
In [0]: init_random_params, predict = stax.serial(
            stax.Conv(32, (3, 3), padding='SAME'), stax.Relu,
            stax.Conv(120, (5, 5), (1,1), padding='VALID'), stax.Relu,
            stax.MaxPool((2, 2)), stax.Flatten,
            stax.Dense(128), stax.Relu,
            stax.Dense(10), stax.LogSoftmax,
        )
```

Figure 2: Architecture that yields the accuracy required.

```
Test set loss, accuracy (%): (0.05, 98.24)
Test set loss, accuracy (%): (0.04, 98.77)
Test set loss, accuracy (%): (0.03, 99.10)
Test set loss, accuracy (%): (0.03, 98.96)
Test set loss, accuracy (%): (0.03, 99.02)
Test set loss, accuracy (%): (0.03, 98.93)
Test set loss, accuracy (%): (0.03, 99.13)
Test set loss, accuracy (%): (0.03, 99.15)
Test set loss, accuracy (%): (0.03, 98.98)
Test set loss, accuracy (%): (0.03, 99.08)
```

Figure 3: This is the log output showing each of the accuracies for each of the epochs.

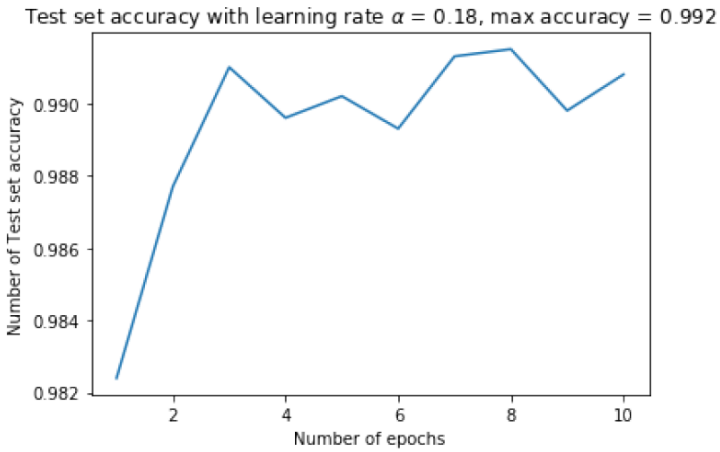


Figure 4: Accuracy by number of epochs for the architecture described above.

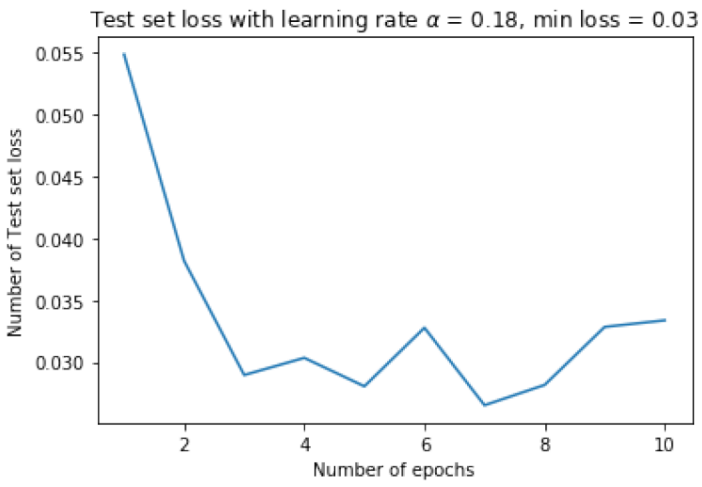


Figure 5: Loss by number of epochs for the architecture described above.

```
Standard deviation of accuracies: 0.00255467533133924

In [223]: print("Mean of accuracies: {}".format(np.mean(np.array(test_accs))))
Mean of accuracies: 0.9893600344657898

In [224]: print("Max of accuracies: {}".format(np.max(np.array(test_accs))))
Max of accuracies: 0.9915000200271606
```

Figure 6: This shows the minimum and maximum accuracy rates over 10 epochs as well as the standard deviation.