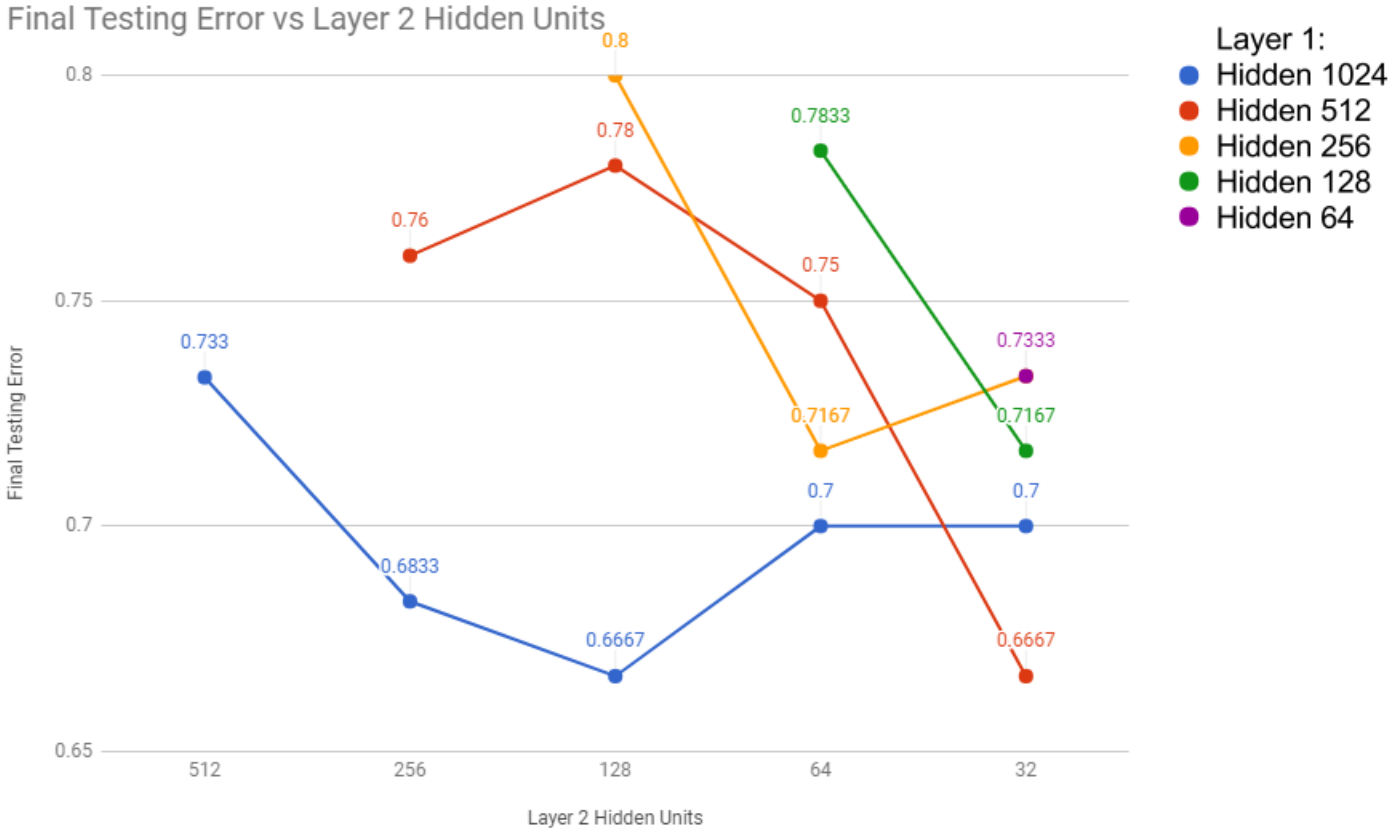
Homework 3
CMPT469 Deep Learning w/ TensorFlow

Kai Wong

11/27/17

1 Part 1: Varying the number of hidden units in each layer, which combination of neurons gave you the best testing accuracy? Why do you think that is?

hidden_units_layer1	hidden_units_layer2	final_testing_error
1024	512	0.733
1024	256	0.6833
1024	128	0.6667
1024	64	0.7
1024	32	0.7
512	256	0.76
512	128	0.78
512	64	0.75
512	32	0.6667
256	128	0.8
256	64	0.7167
256	32	0.7333
128	64	0.7833
128	32	0.7167
64	32	0.7333



A combination of 512 neurons in the first layer and 32 neurons in the second layer yielded the best final testing accuracy. A combination of 1024 neurons in the first layer and 128 neurons in the second layer also yielded an equally good final testing accuracy.

I think that this is due to the nature of an autoencoder. An autoencoder will take a set of unlabeled inputs and try to extract the most valuable information from them via encoding. The decoder portion of the network works to recreate the input as well as possible, forcing the autoencoder to select the most important features from the input to put in its compressed representation. Our model has multiple encoding layers. This is good because more layers allows us to extract different characteristics and features as the input moves through the network.

An autoencoder model performs best when the first hidden layer is large. This allows the model to compute a powerful representation of the data and learn the input effectively. These good representations of the data also allow the model to help smaller subsequent hidden layers better compute its denser representations. Our best versions of the model consist of this; the number of neurons in the first hidden layer is relatively large: 1024 and 512. When the first hidden layer in an autoencoder does not have enough neurons, the model will generally perform worse, as it will not have enough neurons to create a good representation of the input data. Our model with 1024 hidden units in its first layer performs the best overall, while lower numbers of hidden units generally perform worse.

The subsequent hidden layers should be smaller. This will force the model to better learn and extract hidden relationships from the initial representation of the data from the first layer. Our best versions of the model consist of this; the number of neurons in the second hidden layer is relatively small compared to the first hidden layer: 128 (compared to 1024) and 32 (compared to 512). Versions of our model where the number of neurons in the second layer was similar to the number of neurons in the first layer yielded poor testing results; for example, 512 in the first layer and 256 or 128 in the second, 256 in the first layer and 128 in the second, 128 in the first layer and 64 in the second, and so on.

2 Part 2: Varying the learning rate in each layer, which combination of learning rates gave you the best testing accuracy? Why do you think that is?

learning_rate_layer1	learning_rate_layer2	final_testing_error
0.00001	0.00001	0.8
0.0001	0.00001	0.7667
0.0001	0.0001	0.6667
0.0001	0.001	0.7167
0.0001	0.01	0.7833
0.0001	0.1	failed
0.001	0.00001	0.6833
0.001	0.0001	0.6167
0.001	0.001	0.65
0.001	0.01	0.65
0.01	0.0001	failed
0.01	0.01	failed
0.1	0.0001	failed
0.1	0.1	failed

With a combination of 512 neurons in the first layer and 32 neurons in the second layer, a learning rate of 0.001 in the first layer and 0.0001 in the second layer gives the best test accuracy.

As discussed previously, this is due to the nature of an autoencoder. Our network has multiple encoding layers, producing better results for reasons stated previously. The learning rate affects the step size

taken in the loss function in order to minimize the testing error. Choosing the correct learning rate for each layer is important, as these encoding and decoding layers need to be as effective as possible. The output of one layer is used as input for the second layer, further increasing the importance of proper learning rates in . In general, too small of a learning rate leads to the optimizer either getting stuck in a local minimum, or it leads to the model learning the detail or noise of the data too well, causing the model to fail in generalization. It also may lead to the minimum never being reached, with the step size being too small. Too large of a learning rate causes the optimizer to constantly overshoot the global minimum, causing it to never be able to find the minimum in the function, and even leading to model instability. The ideal learning rate will find the minimum in the function, but not such that it finds the exact minimum, but rather, it stays in the area of the minimum; this allows it to generalize well.

In an autoencoder with multiple layers, it is important for the outer layers to have a higher learning rate and the inner layers to have a smaller learning rate. This allows the model to first learn a higher level representation of the data, and then learn more of its detail such that it can extract hidden features.

Therefore, it is best for the learning rate in the first layer to not be too small or too large, as the first layer is learning a high level representation of the input data. As a result, the derivative of the error function (the error surface) is smoother and easier to traverse, thus we can minimize it easier with a larger step size. Furthermore, the model should not learn the detail so much in this layer (by having too small of a learning rate), as it is up to the next layer to learn more detail and extract hidden features. If the model learns too much detail in the first layer, the model will fail to generalize well; it will just learn to copy the input. Our best version of the model has a medium-sized learning rate of .001 for its first layer, allowing it to effectively traverse the loss function. Other versions of the models with larger step sizes (.01 and higher) fail with error, while models with smaller step sizes (.0001 and lower) see increased final testing error. This data further supports that the learning rate for the first layer should be not too large or small.

For the second layer, the learning rate should be small. The output from the first layer is treated as the input for the second layer, and since this data is encoded and dense (more data represented in fewer neurons), there is more room for error. This makes the derivative of the error function (the error surface) more difficult to traverse, such that a smaller step size is needed to find the global minimum. Furthermore, this layer needs to better learn the data in order to extract hidden features. However, for reasons stated before, the learning rate cannot be too small, otherwise there will be an increase in error. Our best version of the model has a relatively small learning rate of .0001 for its second layer, allowing it to find the minimum in the loss function effectively. When we increase the learning rate, the testing error worsens to the point of where the model fails. When we decrease the learning rate, the testing error improves, but decreasing the learning rate by too much (0.00001) increases test error. This data further supports that the learning rate for the second layer should be small.

3 Part 3: Repeat the above but using RBMs (bonus)

I modified the RBM code to use our dataset, but I seemed to be getting extremely low accuracy ratings or zero accuracy ratings for testing. Therefore, I don't think I can draw any substantial conclusions from the data I obtained, since I think that the dataset is flawed (at least when compared to the original MNIST dataset the code was made for, or the CIFAR dataset). However, I wanted to demonstrate my attempt. Please see my code and outputs on GitHub!