ENEE 6583 – Neural Nets
Project 2 – Backpropagation

Week 1:

We will be experimenting with code on the following github:
Original Python 2.7 code:
https://github.com/mnielsen/neural-networks-and-deep-learning/tree/master/src
Modified to Python 3.5.2 :
https://github.com/MichalDanielDobrzanski/DeepLearningPython35
MNIST Data:
https://github.com/mnielsen/neural-networks-and-deep-learning/tree/master/data

1. Start by getting the MNIST data
2. Unzip the data and take a look at the images it contains.
3. Use the msnist_loader.py to load the data. Determine size of each image, range of data, number of classes, number of images in the training/validation/testing sets.
```
training_data, validation_data, test_data = \
... mnist_loader.load_data_wrapper()
```

4. Use the network.py code to create a neural network with 784 input neurons, 10 output neurons, and 30 hidden neurons:
```
net = network.Network([784, 30, 10])
```

5. Train the network using stochastic gradient descent for 30 epochs, with a mini-batch size of 10, and a learning rate of 3.0:
```
net.SGD(training_data, 30, 10, 3.0, test_data=test_data)
```

6. Repeat training step 4,5 but for 100 hidden neurons.
7. Repeat training step 4,5 but for a learning rate of 0.001 and another time for 100.0.
8. Use the network2.py and repeat 4,5 using cross-entropy cost function, learning rate of 0.5, using 10000 training data samples:
```
net = network2.Network([784, 30, 10], cost=network2.CrossEntropyCost)
net.large_weight_initializer()
net.SGD(training_data[:1000], 400, 10, 0.5, evaluation_data=test_data,
... monitor_evaluation_accuracy=True, monitor_training_cost=True)
```

9. For the 8, make plots: cost vs epoch, accuracy vs epoch (see overfitting.py for plotting help)
10. Repeat 8,9 using a regularization parameter λ=0.1 (note python has a reserved word lambda):
```
net.SGD(training_data[:1000], 400, 10, 0.5,
... evaluation_data=test_data, lmbda = 0.1,
... monitor_evaluation_cost=True, monitor_evaluation_accuracy=True,
... monitor_training_cost=True, monitor_training_accuracy=True)
```

11. Repeat 8,9 for a λ=5.0
12. Train the network with entire training data, the improved weight initializer, L2 regularization, and a λ=5:
```
net = network2.Network([784, 100, 10],
... cost=network2.CrossEntropyCost)
net.SGD(training_data, 30, 10, 0.5, lmbda=5.0,
... evaluation_data=validation_data,
... monitor_evaluation_accuracy=True)
```

13. Repeat 12 for a deep network with two hidden layers, with 30 neuron in each.
14. Reapet 12 for a deep network with three hidden layers, with 30 neuron in each.
15. Reapet 12 for a deep network with four hidden layers, with 30 neuron in each.
16. Modify the network2.py code by adding a tanh activation function, and its gradient.
17. Repeat 12, 15 for the tanh function.
18. Modify the network2.py code by adding Lecun's tanh function, and its gradient.
19. Repeat 12, 15 for new tanh function.
20. ~~Modify the network2.py code by adding RECLU activation function, and its gradient.~~
21. ~~Repeat 12, 15 for the RECLU function.~~

Week2

1. Change the initialization scheme so that it does Xavier initialization
2. Change the learning so that it follows Adagrad
3. Use 1,2 and repeat step 15 from week1.
4. Compare convergence time and accuracy to step 15 of week1
5. Download the notMNIST data set
   Start by browsing the following github py code:
   https://github.com/tensorflow/tensorflow/blob/master/tensorflow/examples/udacity/1_notmnist.ipynb
6. Repeat 3 on the notMNIST data.
7. Modify the code to include a RELU and softmax function, and their gradients.
8. Create a network with 784 inputs, 4 hidden layers x 30 ReLU neurons, and 10 softmax outputs.
9. Change the initialization scheme into a ReLU initialization scheme.
10. Process the notMNIST data.

~~Reproduce steps 12-15, 17, 19, 21 from week 1 but using the notMNIST dataset.~~