## Assignment IV

### Using the cross-entropy to classify MNIST digits

Please use the program network2.py that incorporates the cross-entropy cost function. In this exercise, apply the cross-entropy cost function to classify the MNIST digits. Use a network with 30 hidden neurons, and a mini-batch size of 10. Set the learning rate to $\eta = 0.5$* and train for 30 epochs.

Documentation about network2.py's interface can be found by using commands such as help(network2.Network.SGD) in a Python shell. Use the following Python shell:

```
>>> import mnist_loader
>>> training_data, validation_data, test_data = \. . .  mnist_loader.load_data_wrapper()
>>> import network2
>>> net = network2.Network([784, 30, 10], cost=network2.CrossEntropyCost)
>>> net.large_weight_initializer()
>>> net.SGD(training_data, 30, 10, 0.5, evaluation_data=test_data,...
monitor_evaluation_accuracy=True)
```

- Note that the net.large_weight_initializer() command is used to initialize the weights and biases. This command needs to be run because later the default weight initialization in our networks will be changed.
- Also look at the case 100 hidden neurons are used. Based on the error rate using the quadratic cost function, what is the improvement if any?

Caution – any improvements in the use of the cross-entropy cost function over the quadratic cost function do not conclusively prove that the cross-entropy is a better choice. The reason is that a thorough job of optimizing the hyper-parameters (such as learning rate, mini-batch size, etc.) needs to be performed. Therefore, in general, the interpretation of improvements is always problematic. Still, keep in mind that such tests fall short of definitive proof, and remain alert to signs that the arguments are breaking down.

### Overfitting

Use the 30 hidden neuron network, with its 23,860 parameters. Train the network using just the first 1,000 training images and not all 50,000 MNIST training images. Train the network using the cross-entropy cost function, with a learning rate of $\eta = 0.5$ and a mini-

batch size of 10. However, train for 400 epochs since we are not using as many training examples. This construction should illustrate an example of a network that does a bad job generalizing it results.

Use network2 to study how the cost function changes:

```
>>> import mnist_loader>>> training_data, validation_data, test_data = \
... mnist_loader.load_data_wrapper()
>>> import network2
>>> 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)
```

- Use the program overfitting.py to plot the results: Cost on the training data as the network learns vs. Epoch, the classification accuracy (%) on the test data vs. Epoch. What can be concluded about the overfitting?
- Suppose we interchanged the cost and the classification accuracy between the two data sets. So again using the program overfitting.py, plot the results for: Cost on the test data as the network learns vs. Epoch, the classification accuracy (%) on the training data vs. Epoch.

## validation_data

Recall that in the MNIST data, three data sets can be loaded. To setup the validation training set replace 'test_data' with validation_data as illustrated below:

```
>>> import mnist_loader>>> training_data, validation_data, test_data = \
... mnist_loader.load_data_wrapper()
>>> import network2
>>> net = network2.Network([784, 30, 10], cost=network2.CrossEntropyCost)
>>> net.large_weight_initializer()
>>> net.SGD(training_data[:1000], 400, 10, 0.5, evaluation_data=validation_data, ...
monitor_evaluation_accuracy=True, monitor_training_cost=True)
```

Up to now, only the training_data and test_data have been used while ignoring a third dataset known as the validation_data. In this experiment, the validation_data contains 10, 000 images of digits, images, which are different from the 50, 000 images in the MNIST training set, and the 10, 000 images in the MNIST test set. The strategy is to use the test_data as described above. However, the classification accuracy is computed on the validation_data at the end of each epoch. Once the classification accuracy on the validation_data has saturated, stop training. This strategy is called *early stopping*. In

practice we won't know when the accuracy has saturated. Instead, continue training until we're confident that the accuracy has saturated.

Summary:

Training Set – this data set is used to adjust the weights on the neural network.

Validation Set – this data set is used to minimize overfitting. The weights of the network are not adjusted with this data set. Its only used to verify that any increase in accuracy over the training data set actually yields an increase in accuracy over a data set that has not been shown to the network before, or at least the network hasn't trained on it (i.e. validation data set). If the accuracy over the training data set increases, but the accuracy over then validation data set stays the same or decreases, then you're overfitting your neural network and you should stop training.

Testing Set – this data set is used only for testing the final solution in order to confirm the actual predictive power of the network.