Milo Knowles
April 6, 2017
6.036 Project 2

6.036 Project 2 Writeup

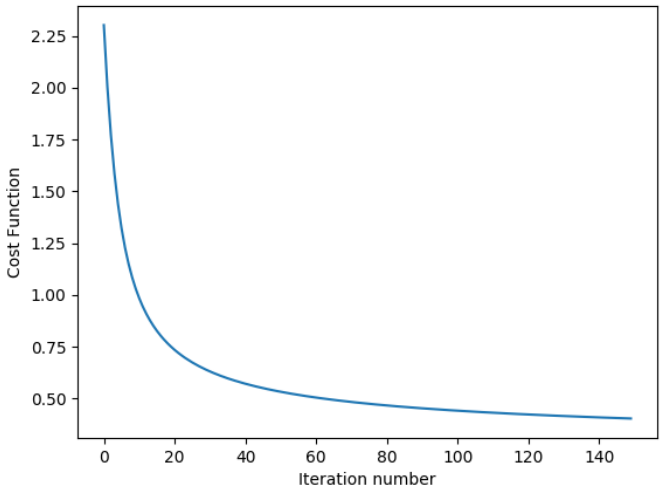
Environment:

- Keras (2.0.2)
- Tensorflow (1.0.1)
- Numpy (1.12.1)
- Python (3.5.1)

Part 1: Multinomial/Softmax Regression and Gradient Descent

4. **Final Test Error: 0.1005**

Cost Function vs. Iterations (tau = 1.0)



5. **Effect of temperature parameter:** The temperature parameter inversely affects the probability of a sample x(i) being assigned a label with a large theta. This is because a higher temperature parameter reduces the size of the softmax output for a given label. This means that labels with large theta vectors will have less dominance over labels with smaller theta vectors.

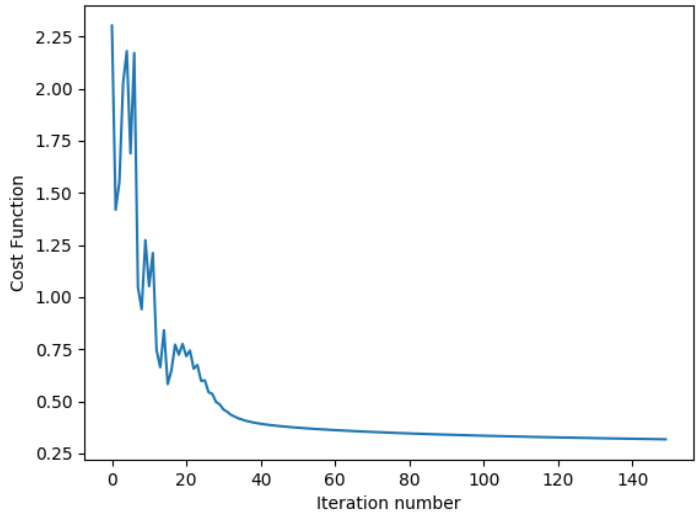
For a label with a relatively small theta, increasing the temperature parameter increase the probability of that label being chosen. This is because a larger temperature parameter will essentially "flatten out" the probability distribution among the possible labels.

6. The trend in the table below indicates that the error rate increases with increasing temperature parameter. This is likely because increasing the temperature parameter decreases the probability that a sample is assigned a label that has a large theta. The in

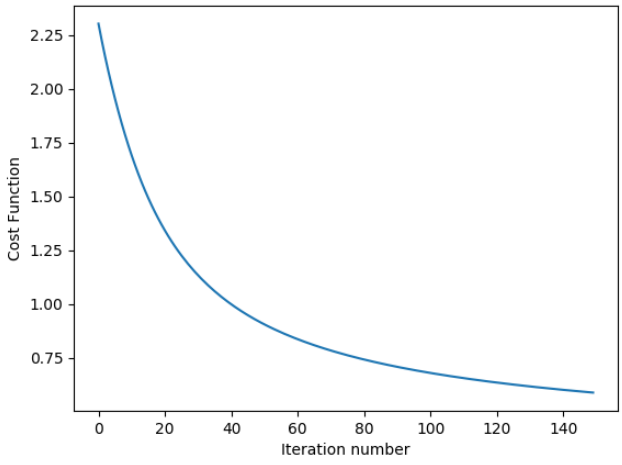
turn increases the probability that a sample is assigned the incorrect label, thus increasing the error rate.

Temperature Parameter	Error Rate
0.5	0.084
1.0	0.101
2.0	0.126

Cost Function vs. Iterations (tau = 0.5)



Cost Function vs. Iterations (tau = 2.0)



Test Error (mod 10): 0.1005
Test Error (mod 3): 0.0768

7. The test error (mod 3) is lower because some misclassifications are now treated as "correct" classifications. For example, if the digit 3 is misclassified as 6 or 9, it will still appear to be a correct classification, modulo 3. This means that some of the misclassifications we have when running our initial logistic regression are now accepted as correct, so the test error is lower.

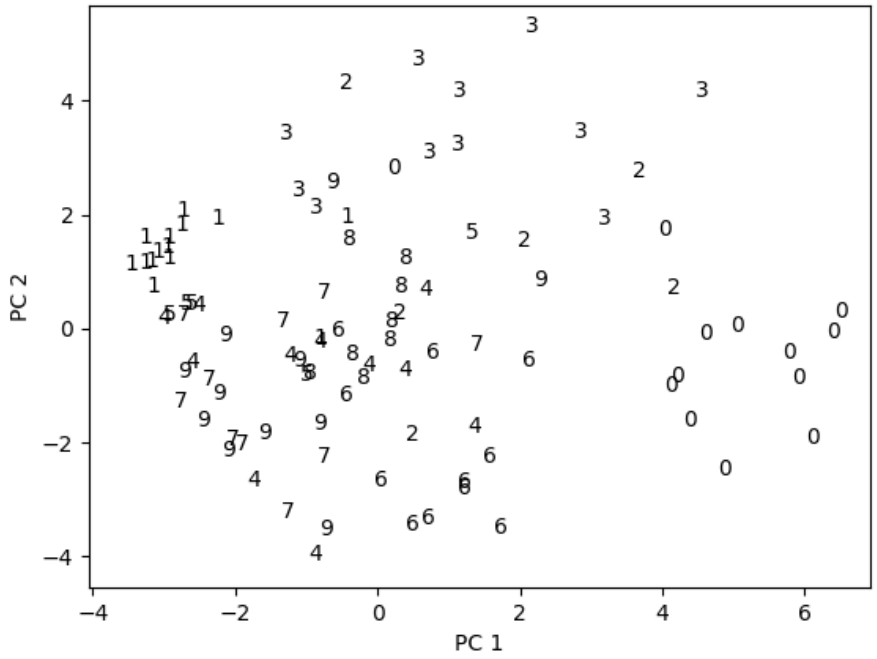
8. After training the logistic regression on the labels (mod 3):
Test Error (mod 3): 0.1872

9. When we compute the test error on modulo 3 labels using our original model, the error improves. However, when we train our model on modulo 3 labels, the test error becomes significantly worse. This is because in the former case, we are training our model to predict all 0-9 digits, and then computing the error of the model with less specificity. In the latter case, we are essentially trying to train a model to recognize only 3 digits – 0, 1, and 2. But for each label, the model has to learn 3 different handwritten digits that all correspond to the same thing in modulo three. In this way, the model is being trained with as clear of a correlation between the handwritten images it sees, and the labels it is supposed to output. For example, a 3, 6, and 9 digit are visually very different, but the model must learn a way to associate all 3 of these digits with a label of 0.

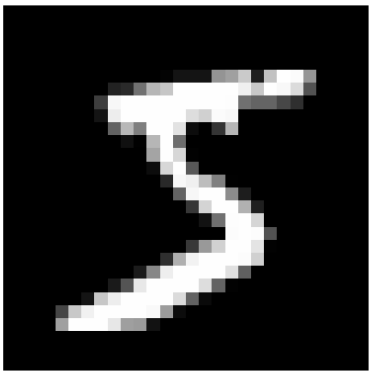
Part 2: Classification Using Manually-Crafted Features

3. Test Error (PCA): 0.171

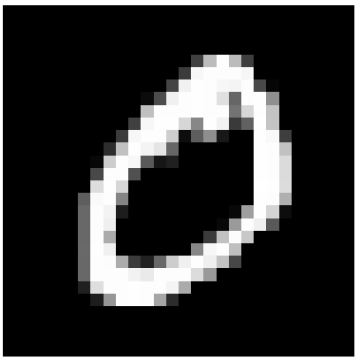
4. Plot of the first 100 MNIST images, represented by first 2 principal components



5. **First Image (Reconstructed)**



Second Image (Reconstructed)



6. $\boldsymbol{\phi(x) = \{x_1^3,\ \sqrt{3}x_1^2x_2,\ \sqrt{3}x_1^2,\ \sqrt{3}x_1^2x_2^2,\ \sqrt{6}x_1x_2,\ \sqrt{3}x_1,\ x_2^3,\ \sqrt{3}x_2^2,\ \sqrt{3}x_2,\ 1}$

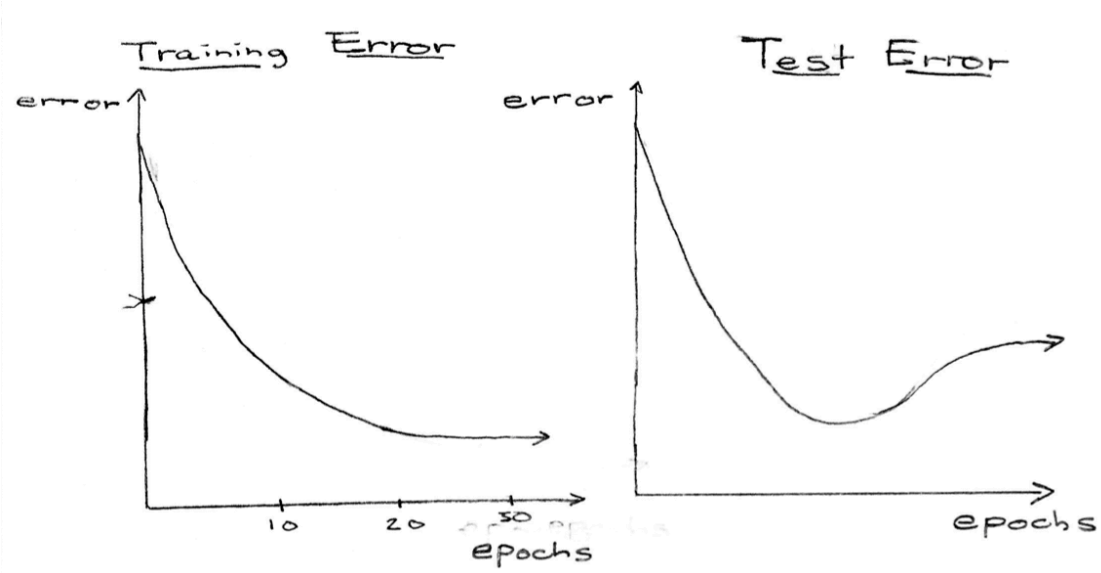
7. Test Error With PCA10 and Cubic Feature Mapping: 0.0867

Part 3: Neural Network Basics

4. The learning rate can be varied during the training process to improve the training efficiency and accuracy. In particular, the learning rate should start out a higher rate, and be decreased at each training epoch. Once such method is to divide the learning rate by the current epoch number – at very late epochs, the learning rate will be very small, and make very small updates to the model.

5. Increasing the number of hidden units in the network runs the risk of over-fitting to the training data. With many hidden units, the model can fit very precisely to the training data it sees, and not generalize well to test data.

6. As the model is trained on more epochs, training error will montonically decrease, and eventually reach an asymptotic minimum. Test error will decrease with more epochs of training, but begin to increase again eventually. This is because at a high number of epochs, the model will begin to overfit to training data, and performance on the test examples will suffer. Consider the qualitative plot below:



7. Once method that could be used to optimize the number of training epochs is to either use a the test set or a separate validation set to check the error of the model at every training epoch. It is important that this validation set is separate from the examples we train on. We continue to run more training epochs until the validation error stops decreasing and begins to increase instead. At this point, we know that we are beginning to overfit to the training data, so the previous epoch was at the optimal number of epochs.

Part 4: Classification for MNIST using deep neural networks

1a. **Accuracy of Example Model:** 0.9152
1b. **Accuracy of Modified Model:** 0.9803

First, using default parameters, I added a second dense hidden layer. This increased the test accuracy to 0.926, an improvement. Next, keeping two dense hidden layers, I tried increasing the number of hidden units from 128 to 200. This also showed improvement, with a test accuracy of 0.931. However, when I increased the number of hidden units from 200 to 256, the test accuracy worsened slightly, likely due to over-fitting. I saw improvement after adding a third and fourth layer, which brought accuracy up to 0.94.

Next, I kept the network architecture I arrived at above and began tuning SGD parameters. Increasing the learning rate resulted in substantial improvement in performance – 0.0045 seemed to be the best value that I tried. Decreasing the momentum parameter from 0.5 to 0.4 improved test accuracy incrementally. However, increasing the

momentum parameter improved accuracy by a larger amount. I continued to increase the momentum parameter, and found that 0.95 was the best value for this parameter, yielding an accuracy of 0.982.

After I was happy with the SGD parameters, I returned to network archecture, noticing that I could achieve the same performance with just 2 hidden layers of 200 units each. This sped up training time, so I went with the model with fewer hidden layers.

The final architecture and parameters of my model are described in the table below:

Final Test Error (Optimized): 0.9803

Model Architecture and Parameters

Layers	2
Hidden Units	200
Learning Rate	0.0045
Momentum	0.95

2a. **Accuracy of CNN (1 epoch):** 0.9814

Part 5: Overlapping, multi-digit MNIST

Input Dimensions: There are 40,000 images in the training data, and 4,000 images in the test data. Each image is 42 pixels by 28 pixels.

Model.fit Arguments:

- X_train: the training examples
- Y_train[0] and Y_train[1] are passed in the form of a list, because there are the two outputs for our model
- Nb_epoch: the number of training epochs
- Batch_size: the number of training examples that are used during SGD at one time
- Verbose: whether to print out verbose information during the training process

Model.compile Arguments:

- As before, we are using categorical cross entropy as our loss function
- As before, the optimizer is stochastic gradient descent
- Loss_weights tells the model how much "importance" each of our losses has during optimization

Label Indices: Y_train[0] and Y_train[1] correspond to the two labels for every data point: the first and second digits present in the image.

Original Models (implemented in mlp.py and conv.py)

Model #1: Multi-layer Perceptron (mlp.py)

Performance (Test Set): Accuracy #1: 0.9135, Accuracy #2: 0.8998
Training Time: 98 seconds

Model Architecture:

- Input layer
- Flatten layer
- Dense layer with 64 hidden units and ReLU activation
- Dense Output Layer #1 with 10 hidden units and Softmax activation
- Dense Output Layer #2 with 10 hidden units and Softmax activation

Parameter	Value
Loss	Categorical Cross Entropy
Optimizer	SGD (default)
Epochs	30
Batch Size	64

Model #2: Convolutional Neural Net (conv.py)

Performance (Test Set): Accuracy #1: 0.886, Accuracy #2: 0.863
Training Time: 307 seconds

Model Architecture:

- Input layer
- 2D Convolutional layer with 8 filters, (3, 3) kernel, and ReLU activation
- 2D Max Pooling layer with size (2, 2) filter, (2, 2) strides, and ReLU activation
- 2D Convolutional layer with 16 filters, (3, 3) kernel, and ReLU activation
- 2D Max Pooling layer with size (2, 2) filter, (1, 1) strides, and ReLU activation
- Flatten layer
- Dense layer with 64 hidden units and ReLU activation
- Dropout layer with value 0.5
- Dense Output Layer #1 with 10 hidden units and Softmax activation
- Dense Output Layer #2 with 10 hidden units and Softmax activation

Parameter	Value
Loss	Categorical Cross Entropy
Optimizer	SGD (default)
Epochs	3
Batch Size	64

Modified Models (implemented for question 3)

Model #3: CNN with Modified Architecture
Unless specified in the table below, all of the parameters of this model are the same as the suggested ones given in Model #2.

Performance (Test Set): Accuracy #1: 0.8697, Accuracy #2: 0.8497
Training Time: 304 seconds

Parameter	Value
Dense Hidden Layers	2
Units in Dense Hidden Layers	128

Model #4: CNN with Modified Optimization Parameters
Unless specified in the table below, all of the parameters of this model are the same as the suggested ones given in Model #2.

Performance (Test Set): Accuracy #1: 0.9499, Accuracy #2: 0.9345
Training Time: 376 seconds

Parameter	Value
Optimizer	Adagrad
Learning Rate	0.01
Epsilon	1e-08
Decay	0.0

Model #5: MLP with Modified Architecture and Optimization Parameters
Unless specified in the table below, all of the parameters of this model are the same as the suggested ones given in Model #1.

Performance (Test Set): Accuracy #1: 0.9250, Accuracy #2: 0.9235
Training Time: 184 seconds

Parameter	Value
Dense Hidden Layers	2
Hidden Units	64
Optimizer	SGD
Learning Rate	0.01
Momentum	0.9
Decay	0.0

Nesterov	True