Oluwafemi Oladipupo
6.036 Project 2 Report

(0) Environment

- Python 3
- anaconda-fusion
- appdirs
- pyyaml
- theano
- keras==1.2.2
- opencv-python==3.2.0.6
- packaging==16.8
- protobuf==3.2.0

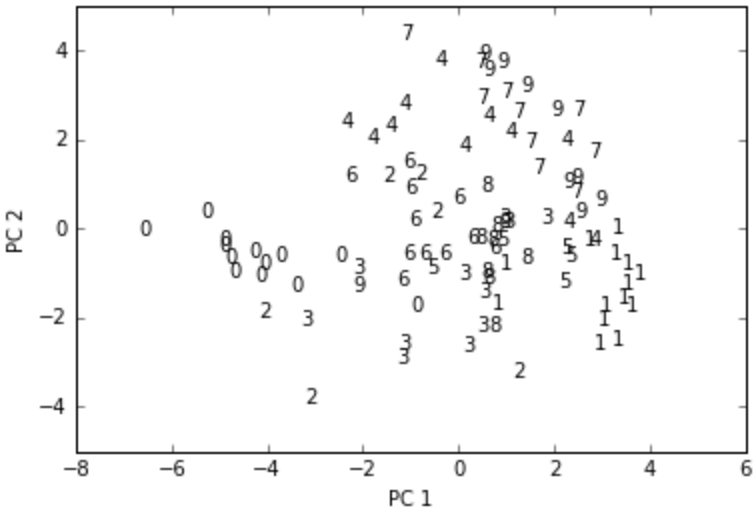
(1) Multinomial/Softmax Regression and Gradient Descent

1. *computeProbabilities* implemented
2. *computeCostFunction* implemented
3. *runGradientDescentIteration* implemented, all tests passed
4. Test Error = 0.1005
5. Increasing the temperature parameter decreases the gradient magnitude. This causes less change in theta, making x^(i) weigh more towards small theta labels. Decreasing the parameter increases the probability for large theta labels.
6. Temperature Parameters [0.5, 1.0, 2.0] => Test Errors [0.084, 0.1005, 0.1261]. The error decreases with parameter size, indicating a higher probability of small theta labels.
7. Error rate for mod 3 labels, with the normal theta is 0.0768. The error decreased because any misclassifications had a chance to correct when converted via the mod 3 operation, while any correct classifications would remain correct.
8. *runSoftmaxOnMNISTMod3* implemented
9. Error rate for mod 3 labels, with the mod 3 theta is 0.1872. The errors are different because the features used for classification do not directly relate to the property of the mod 3 value. The shape of a digit does not determine its properties.

(2) Classification using manually-crafted features

1. *projectOntoPC* implemented
2. *projectOntoPC* used to compute 18-dimensional PCA
3. *runSoftmaxOnMNISTPCA* run with PCA. Test error = 0.1483

4.



5.

6. All tests passed. Feature Map: T(x) = <x_1^3, ... , x_n^3, sqrt(3)x_1^2, ... , sqrt(3)x_n^2, sqrt(3)x_1, ... , sqrt(3)x_n, sqrt(3)$x_i^2 x_j$ for i=/=j, sqrt(6)$x_i x_j$ for i>j, sqrt(6)$x_i x_j x_k$ for i>j>k, 1>
7. Cubic test error = 0.0867

(3) Neural Network Basics

1. *rectified_linear_unit* and *rectified_linear_unit_derivative* implemented
2. *train* implemented
3. *predict* implemented, all tests passed
4. Fine tuning the learning rate, or scaling the learning rate with the number of iterations of training could improve performance, as the increased rate early would help avoid local minima, while finer adjustments later better hone in on the desired solution.
5. Having too many hidden units increases the risk of overfitting the training data.
6. Running the code for more epochs increases the training accuracy while testing predictions remain roughly within the same margins. This indicates that the testing accuracy has reached or is very near its accuracy minima, and that further epochs do not necessarily help with prediction.
7. One method is to stop training when testing accuracy levels off, as further training will only overfit testing data.

(4) Classification for MNIST using deep neural networks

1. Fully Connected Neural Network
    a. Accuracy on test set: 0.9154
    b. My final model used 1 hidden layer with 160 neurons, a learning rate of 0.02, and a momentum of 0.8. This resulted in a accuracy of 98%. I also tried variations with up to two layers, and neuron amounts ranging from 16 to 192 per layer. I varied the learning rates from 0.0005 to 0.024 and momentum from 0.5 to 0.82. All the iterations of my model can be found described in the comments of my code. The only iteration that reached 98% aws the final model described above.
2. Convolutional neural networks
    a. Training accuracy: 93.6% Validation accuracy: 97.7%
    b. Optional

(5) Overlapping, multi-digit MNIST

Images: 64, Size: 42x48
y_train [0] and y_train[1] are the two numbers in each image.

model.compile configures the learning process: loss='categorical_crossentropy' being the loss function, optimizer='sgd' being stochastic gradient descent, metrics=['accuracy'] determining how the model is evaluated.

model.fit trains the model for a certain # of epochs: X_train is the input data, [y_train[0], y_train[1]] are the input labels, nb_epoch=nb_epoch are the number of epochs, batch_size=batch_size is the number of samples per gradient update, verbose=1 determines whether or not to show progress bar logging.

1. *main* in *mlp.py* implemented
2. *main* in *conv.py* implemented
3. 5 Models with accuraies
    a. Model from 1, SGD
        i. Input
        ii. Flatten
        iii. FC(64) relu Activation
        iv. Output1 = FC(10) softmax Activation, Test Accuracy: 0.9053
        v. Output2 = FC(10) softmax Activation, Test Accuracy: 0.8988
    b. Model from 2, SGD
        i. Input
        ii. Conv2D relu, 8 filters, 3x3
        iii. MaxPooling2D relu, filters (2,2), stride (2,2)

    iv. Conv2D relu, 16 filters, 3x3
    v. MaxPooling2D relu, filters (2,2), stride (1,1)
    vi. Flatten
    vii. FC(64) relu Activation
    viii. Dropout(0.5)
    ix. Output1 = FC(10) softmax Activation, Test Accuracy: 0.9133
    x. Output2 = FC(10) softmax Activation, Test Accuracy: 0.8810
c. Model from 1, Adam
    i. Output1, Test Accuracy: 0.9280
    ii. Output2, Test Accuracy: 0.9204
d. Model from 2, Adam
    i. Output1, Test Accuracy: 0.965
    ii. Output2, Test Accuracy: 0.954
e. Model from 2, Conv2D 3x3 -> 4x4 filters
    i. Output1, Test Accuracy: 0.965
    ii. Output2, Test Accuracy: 0.950