Project 3: Report

CSE474/574: Introduction to Machine Learning (Fall 2016)

Instructor: Sargur N. Srihari Teaching Assistants: Jun Chu, Junfei Wang and Kyung Won Lee

-Vaibhav Sinha

- vsinha2@buffalo.edu

- Person#: 50208769

Importing MNIST Data

Imported the data using cPickle Library.

```
filename = 'mnist.pkl.gz'
f = gzip.open(filename, 'rb')
training_data, validation_data, test_data = cPickle.load(f)
```

Importing USPS Data

- Read each image using opencv
- Resize it to 28*28
- Change it to its negative image as it lead to better accuracy
- Unravelled the image to a 786 feature vector
- Normalized the intensity values

```
def GetUSPSData():
        curPath = os.path.dirname(os.path.abspath(__file__))
        curPath += '/USPSData/Numerals'
        testDataX = []
        testDataY = []

        for i in range(9):
                curFolderPath = curPath + '/' + str(i)
                imgs =  os.listdir(curFolderPath)
                for img in imgs:
                        curImg = curFolderPath + '/' + img
                        if curImg[-3:] == 'png':
                                curImg = cv2.imread(curImg,0)
                                curImg = cv2.resize(curImg,(28,28))
                                curImg = numpy.array(curImg)
                                curImg = 255 - curImg
                                curImg = curImg.ravel()
                                curImg = curImg/255.0

                                testDataX.append(curImg)
                                testDataY.append(i)

        return testDataX,testDataY
```

Training Logistic Regression

- Trained using Stochastic Gradient Descent
- Started with Random Weights of values between 0 and 1

```
def GetRandomWeights():
        w = numpy.zeros(shape=(10,784))
        for i in range(10):
                for j in range(784):
                        w[i][j] = randint(0,9)/10.0
        return w
```

- Created one hot encoding of labels

```
t = numpy.zeros(shape=(50000,10))
        for i in range(len(trainingY)):
                t[i][trainingY[i]] = 1
```

- Then we run 15 iterations on the training set,
- In each iteration we take one image at a time for all images – Stochastic Gradient Descent
- We calculate the gradient and update the weights accordingly.

Code of Gradient Descent

```
neta = 0.01
for l in range(15):
        print "Iteration = “, l

        for i in range(50000):

                x = trainingX[i]
                y = numpy.dot (w, numpy. transpose(x))

                for j in range (10):
                        y[j] += b[j]


                y = softmax(y)


                diff = y - numpy.transpose(t[i])
                grad = Multiply(diff,x)
                w = w - neta*grad
```

Code for Testing

```
def TestAccuracy(w,b,testX,testY):
        count = 0
        for i in range(len(testX)):
                y = numpy.dot(w,numpy.transpose(testX[i]))
                for j in range(10):
                        y = numpy.add(y,b[j])
                y = softmax(y)
                max = numpy.argmax(y)
                if max == testY[i]:
                        count += 1


        print "Total Correct Predictions on USPS Data = ",count
        accuracy = (count/float(len(testX))) * 100.0
        print "Accuracy on USPS Data = ",accuracy
```

Sample Result

Iteration = 0
Iteration = 1
Iteration = 2
Iteration = 3
Iteration = 4
Iteration = 5
Iteration = 6
Iteration = 7
Iteration = 8
Iteration = 9
Iteration = 10
Iteration = 11
Iteration = 12
Iteration = 13
Iteration = 14
==========
Total Correct Predictions on Training = 46055
Training Accuracy = 92.11
Total Correct Predictions on Test = 9122
Test Accuracy = 91.22
Total Correct Predictions on USPS Data = 6362
Accuracy on USPS Data = 35.1880530973

Training Neural Network

- Imported the data just like the way I imported in Logistic Regression
- Started with random weight values between 0 and 1

```
def GetRandomWeights(rows,cols):
        w = numpy.zeros(shape=(rows,cols))
        for i in range(rows):
                for j in range(cols):
                        w[i][j] = randint(0,9)/100.0
        return w

w1 = GetRandomWeights(nhl,len(trainingX[0]))
b1 = GetRandomWeights(nhl,1)

w2 = GetRandomWeights(10,nhl)
b2 = GetRandomWeights(10,1)
```

- Created one hot encoding

```
t = numpy.zeros(shape=(len(trainingX),10))
    for i in range(len(trainingY)):
            t[i][trainingY[i]] = 1
```

- We choose a learning rate of 0.1 as it led to optimum results.

Choosing Number of Hidden Layer Nodes

- Started with 20 nodes in the hidden layer and gradually increased the number of nodes.
- It was observed that as we increase the number of nodes in the hidden layer, the system was performing better on the training set.
- But it was getting computationally more and more expensive due to huge matrix operations.
- So in order to reduce this overfitting, it was decided to stop the training after not many iterations i.e. early termination
- So, the system was trained only for 5 iterations
- The number of nodes in the hidden layer were taken to be 100.

Note : It took around 259 secs (~5 mins) for 5 iterations to complete.

Code for Gradient Descent of Single Layer Neural Network

```
neta = 0.1
for l in range(5):
        print "Iteration = ",l
        for i in range(len(trainingX)):

            x = trainingX[i]
            z = numpy.dot(w1,numpy.transpose(x))

            for j in range(nhl):
                z[j] += b1[j]
            z = 1.0/(1.0+numpy.exp(-z))

            a = numpy.dot(w2,numpy.transpose(z))

            for j in range(10):
                a[j] += b2[j]

            a = softmax(a)
            dk = a - numpy.transpose(t[i])
            grad2 = Multiply(dk,z)

            temp1 = numpy.dot(numpy.transpose(w2),dk)
            dz = numpy.multiply(z,1-z) * temp1
            grad1 = Multiply(dz,x)

            w1 = w1 - neta*grad1
            w2 = w2 - neta*grad2
```

Sample Result of Neural Network

```
Iteration =  0
Iteration =  1
Iteration =  2
Iteration =  3
Iteration =  4
Results on Training Set
Total Correct Predictions =  49170
Accuracy =  98.34
Results on Test Set
Total Correct Predictions =  9706
Accuracy =  97.06
Results on USPS Data
Total Correct Predictions =  9691
Accuracy =  53.6006637168
[Finished in 259.1s]
```

Training Convolutional Neural Network

- Tensorflow was used to train convolutional neural network
- Tutorials on tensorflow website was used as reference to train the network
- Imported the data with one hot encoding parameter to true

```
mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)
```

- Started with weights and biases which were normally distributed

```
def weight_variable(shape):
    initial = tf.truncated_normal(shape,stddev=0.1)
    return tf.Variable(initial)

def bias_variable(shape):
    initial = tf.constant(0.1, shape = shape)
    return tf.Variable(initial)
```

- Defined Convolution and Max Pooling Functions

```
def conv2d(x, W):
    return tf.nn.conv2d(x, W, strides=[1, 1, 1, 1], padding='SAME')

def max_pool_2x2(x):
    return tf.nn.max_pool(x, ksize=[1,2,2,1], strides=[1,2,2,1], padding='SAME')
```

Operation to be performed on input Data

```
x = tf.placeholder(tf.float32, [None, 784])
y_ = tf.placeholder(tf.float32, [None, 10])
sess = tf.InteractiveSession()

W_conv1 = weight_variable([5, 5, 1, 32])
b_conv1 = bias_variable([32])

x_image = tf.reshape(x, [-1,28,28,1])

h_conv1 = tf.nn.relu(conv2d(x_image, W_conv1) + b_conv1)
h_pool1 = max_pool_2x2(h_conv1)

W_conv2 = weight_variable([5, 5, 32, 64])
b_conv2 = bias_variable([64])
```

```
h_conv2 = tf.nn.relu(conv2d(h_pool1, W_conv2) + b_conv2)
h_pool2 = max_pool_2x2(h_conv2)

W_fc1 = weight_variable([7*7*64, 1024])
b_fc1 = bias_variable([1024])

h_pool2_flat = tf.reshape(h_pool2, [-1, 7*7*64])
h_fc1 = tf.nn.relu(tf.matmul(h_pool2_flat, W_fc1) + b_fc1)

keep_prob = tf.placeholder(tf.float32)
h_fc1_drop = tf.nn.dropout(h_fc1, keep_prob)

w_fc2 = weight_variable([1024, 10])
b_fc2 = bias_variable([10])

y_conv = tf.matmul(h_fc1_drop, w_fc2) + b_fc2
```

Training the Convolutional Network

```
cross_entropy = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(y_conv, y_))
train_step = tf.train.AdamOptimizer(1e-4).minimize(cross_entropy)
correct_prediction = tf.equal(tf.argmax(y_conv,1), tf.argmax(y_,1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
sess.run(tf.initialize_all_variables())
for i in range(20000):
  batch = mnist.train.next_batch(50)
  if i%100 == 0:
    train_accuracy = accuracy.eval(feed_dict={
        x:batch[0], y_: batch[1], keep_prob: 1.0})
    print("step %d, training accuracy %g"%(i, train_accuracy))
  train_step.run(feed_dict={x: batch[0], y_: batch[1], keep_prob: 0.5})
```

Conclusion

Logistic Regression Accuracy:

On Training Set: 92%
On Testing Set: 91%
USPS Data: 35%

Logistic Regression Accuracy:

On Training Set: 98%
On Testing Set: 96-7%
USPS Data: 53%

Convolution Neural Network Accuracy:

On Training Set: 100%
On Testing Set: 99.2%

Note: For each run on the script, weights are initialized to random values, so accuracy can change a little bit in every run, but it is almost the same every time.

No Free Lunch Theorem Conclusion

We can see that we are getting low accuracy on USPS Data, but good accuracy on MNIST Data. Since, we trained on MNIST Data, we were getting good results on that dataset, but since we did not train on USPS kind of data, we won't be getting good results on that.
Basically, we have to earn the lunch(by training in USPS Data) as we cannot have it for free.