Project 3: Classification

Avinash Kommineni, 50248877 November 20, 2017

1 Introduction

I have 2 implementations of this project. One is using Tensorflow and other just using numpy along with self implementation of backpropagation. First the Tensorflow... Through out this assignmnet, I have followed the Andrew Ng notation for the networks.

The initialisation and MNIST dataset loading.

Each image is of size 28 X 28, which is flattened into an array of size 784.

Training Images of the shape: 784 X 55000

Validation Images of the shape: 784 X 5000

Test Images of the shape: 784 X 10000

Since the labels arrays are one-hot vectors, they are of the shape 10 by number of examples.

```
1  import numpy as np
2  import tensorflow as tf
3  from tensorflow.examples.tutorials.mnist import input_data
4  from matplotlib.pyplot import imshow
5  from PIL import Image
6  import time
7
8  mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)
9
10 trainImages = mnist.train.images.T
11 testImages = mnist.test.images.T
12 trainLabels = mnist.train.labels.T
13 testLabels = mnist.test.labels.T
14 validationImages = mnist.validation.images.T
15 validationLabels = mnist.validation.labels.T
16 m = trainImages.shape[1]
17
18 batchSize = 100
```

```
printEvery = 5
nClasses = 10
learningRate = 0.005

x = tf.placeholder(tf.float32, [784,None])
t = tf.placeholder(tf.float32, [10,None])
```

2 Classifiers

Since the final layer needs to be a classifier of 10 classes, I have opted it to be softmax function whereas all the intermediate activation functions are ReLU. Initially I have tested without weight regularization, but later added it.

2.1 Logistic Regression

For more effecient usage ease of access, I have implemented functional based approach.

```
def logistiticRegression(x):
    W = tf.Variable(tf.random_normal([10,784]))
    b = tf.Variable(tf.zeros([10,1]))
    z = tf.matmul(W,x) + b
    return z
```

2.2 Neural Network

Initially I have chosen the number of hidden units in the hidden layer to be as 512.

```
def neuralNetwork(x):
    lambdaa = 0.1
    nH1 = 512
    weights = {'hiddenL1':tf.Variable(tf.random_normal([nH1,784])),
    'hiddenL2':tf.Variable(tf.random_normal([10,nH1]))}

    biases = {'hiddenL1':tf.Variable(tf.random_normal([nH1,1])),
    'hiddenL2':tf.Variable(tf.random_normal([10,1]))}
```

```
9
10      w_h1 = tf.summary.histogram("weights1",weights['hiddenL1'])
11      w_h2 = tf.summary.histogram("weights2",weights['hiddenL2'])
12      b_h1 = tf.summary.histogram("biases1",biases['hiddenL1'])
13      b_h2 = tf.summary.histogram("biases2",biases['hiddenL2'])
14
15
16      with tf.name_scope("a1") as scope:
17      z1 = tf.matmul(weights['hiddenL1'],x) + biases['hiddenL1']
18      a1 = tf.nn.relu(z1)
19
20      with tf.name_scope("a2") as scope:
21      z2 = tf.matmul(weights['hiddenL2'],a1) + biases['hiddenL2']
22      return z2
```

2.3 Convolutional Neural Network

Initially I have chosen the 32, 64 kernels in first and second convolutional layer respectively. It is then connected to two fully connected layers, each of 1024 nodes. Dropout of 0.3 is applied i.e., keep_rate of 0.7.

```
1  def convolutionalNeuralNetwork(x):
2      weights = {'conv1':tf.Variable(tf.random_normal([5,5,1,32])),
3              'conv2':tf.Variable(tf.random_normal([5,5,32,64])),
4              'fullyC1':tf.Variable(tf.random_normal([7*7*64,1024])),
5              'fullyC2':tf.Variable(tf.random_normal([1024,1024])),
6              'out':tf.Variable(tf.random_normal([1024,nClasses]))}
7
8      biases = {'conv1':tf.Variable(tf.random_normal([32])),
9              'conv2':tf.Variable(tf.random_normal([64])),
10             'fullyC1':tf.Variable(tf.random_normal([1024])),
11             'fullyC2':tf.Variable(tf.random_normal([1024])),
12             'out':tf.Variable(tf.random_normal([nClasses]))}
13
14     x = tf.transpose(x)
15
16     Img = tf.reshape(x,[-1,28,28,1])
17
18     conv1 = tf.nn.relu(convolve2D(Img,weights['conv1']) + biases['↩
```

```
        conv1'])
    conv1 = maxpool2d(conv1)

    conv2 = tf.nn.relu(convolve2D(conv1,weights['conv2']) + biases['conv2'])
    conv2 = maxpool2d(conv2)

    conv2 = tf.reshape(conv2,[-1,7*7*64])
    fcLayer1 = tf.nn.relu(tf.matmul(conv2,weights['fullyC1']) + biases['fullyC1'])
    fcLayer1 = tf.nn.dropout(fcLayer1,keepRate)
    fcLayer2 = tf.nn.relu(tf.matmul(fcLayer1,weights['fullyC2']) + biases['fullyC2'])
    output = tf.matmul(fcLayer2,weights['out']) + biases['out']

    return tf.transpose(output)
```

USPS Dataset

The USPS images are read from their integer folders and reshaped in 28X28 size.// The images and labels are processed and made into arrays of shape 784X19999 and 10X19999, consistent with the MNIST dataset.// Thus the accuracy of classifiers is calculated on both MNIST and USPS the same way.

```
import os
from scipy import ndimage, misc
import glob

images = []
i = 0
labels = []

for root, dirnames, filenames in os.walk("proj3_images/Numerals/"):
    if dirnames!= []:
        dirrr = dirnames
    count = 0
    for filename in filenames:
        if ".png" in filename:
```

```
            count += 1
            filepath = os.path.join(root, filename)
            image = ndimage.imread(filepath, mode="L")
            image_resized = misc.imresize(image, (28, 28))
            images.append(image_resized)
        if count != 0:
            lMid = np.zeros((count,10))
            lMid[:,int(dirrr[i])] = 1
            if labels == []:
                labels = lMid
            else:
                labels = np.vstack((labels,lMid))
            i += 1

uspsImages = np.asarray(images)
uspsLabels = np.asarray(labels)
uspsImages = uspsImages/255
uspsImages = 1 - uspsImages
uspsImages = uspsImages.reshape((-1,784))
#meanUSPSImg = np.mean(uspsImages,0)[:,np.newaxis]
uspsImages = uspsImages.T
uspsLabels = uspsLabels.T
```

Traning

The error function used is the multiclass cross-entropy error function. Since every classifier's output is result of softmax, I move that step into training and thus using Tensorflow's in-built function, softmax_cross_entropy_with_logits. The will also result in better *numerical stability*.
I have also used the inbuilt-tensorboard to better visualise and understand the networks and tensors.

```
def trainNetwork():
    with tf.name_scope("loss") as scope:
        loss = tf.reduce_mean(tf.nn.↩
           softmax_cross_entropy_with_logits(logits=tf.transpose(y)↩
           ,labels=tf.transpose(t)))
        tf.summary.scalar("loss",loss)
```

```
5
6       with tf.name_scope("training") as scope:
7           optimizer = tf.train.AdamOptimizer().minimize(loss)
8
9       with tf.name_scope("accuracy") as scope:
10          correct = tf.equal(tf.argmax(y),tf.argmax(t))
11          accuracy = tf.reduce_mean(tf.cast(correct,'float'))
12
13      init = tf.global_variables_initializer()
14      sess = tf.Session()
15      mergeSummary = tf.summary.merge_all()
16
17      sess.run(init)
18      summaryWriter = tf.summary.FileWriter('../../TFout/2', sess.↩
            graph)
19
20      for epoch in range(nEpochs):
21          error = 0.0
22          for i in range(int(m/batchSize)):
23              batch_xs, batch_ys = mnist.train.next_batch(batchSize)
24              _, er, summaryStr = sess.run([optimizer,loss,↩
                    mergeSummary],feed_dict={x:batch_xs.T,t:batch_ys.T})
25              summaryWriter.add_summary(summaryStr, epoch*(int(m/↩
                    batchSize)) + i)
26              error += er
27          if (epoch+1)%printEvery == 0:
28              print('Loss in ',epoch+1,' epoch is ',error)
29
30      prediction = tf.equal(tf.argmax(y),tf.argmax(t))
31      accuracy = tf.reduce_mean(tf.cast(prediction,"float"))
32      print("Accuracy Train:", sess.run(accuracy,{x: trainImages, t: ↩
            trainLabels}))
33      print("Accuracy validation:", sess.run(accuracy,{x: ↩
            validationImages, t: validationLabels}))
34      print("Accuracy Test:", sess.run(accuracy,{x: testImages, t: ↩
            testLabels}))
35      print("USPS Test Accuracy:", sess.run(accuracy,{x: uspsImages, ↩
            t: uspsLabels}))
```

Results 1

The results are obtained by using the following...
The below mentioned is for the regularised classifiers.

```
    start_time = time.time()
    y, regLoss = logistiticRegression(x)
    nEpochs = 100
    trainNetwork()
    print("--- %s seconds ---" % (time.time() - start_time))

    start_time = time.time()
    y, regLoss = neuralNetwork(x)
    nEpochs = 25
    trainNetwork()
    print("--- %s seconds ---" % (time.time() - start_time))

    nEpochs = 14
    keepRate = 0.7
    start_time = time.time()
    y, regLoss = convolutionalNeuralNetwork(x)
    trainNetwork()
    print("--- %s seconds ---" % (time.time() - start_time))
```

The weight regularization is obtained by adding the regularization loss of l2_norm(weights) to the cross entropy loss. This is acheived by returning **tf.nn.l2_loss(W)**.

```
#Logistic Regression
regLoss = tf.nn.l2_loss(W)
#Neural Network
regLoss = tf.nn.l2_loss(weights['hiddenL1']) + tf.nn.l2_loss(↩
    weights['hiddenL1'])
#CNN
regLoss = tf.nn.l2_loss(weights['conv1']) + tf.nn.l2_loss(weights['↩
    conv2']) + tf.nn.l2_loss(weights['fullyC1']) + tf.nn.l2_loss(↩
    weights['fullyC1']) + tf.nn.l2_loss(weights['out'])

#Modified Loss function
loss = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(↩
    logits=tf.transpose(y),labels=tf.transpose(t)) + lambdaa*regLoss↩
```

)

Code Output

Listing 1: Code output.

```
Loss in  5  epoch is  434.564588502
Loss in  10  epoch is  283.053186908
Loss in  15  epoch is  230.45016335
Loss in  20  epoch is  203.656288765
Loss in  25  epoch is  186.209673814
Loss in  30  epoch is  174.424843781
Loss in  35  epoch is  165.422096811
Loss in  40  epoch is  158.593220886
Loss in  45  epoch is  153.709981691
Loss in  50  epoch is  149.085173644
Accuracy Train: 0.927236
Accuracy validation: 0.9184
Accuracy Test: 0.9183
USPS Test Accuracy: 0.382269
--- 152.29985189437866 seconds ---

Loss in  5  epoch is  80293.9880066
Loss in  10  epoch is  906.527707696
Loss in  15  epoch is  199.356616914
Loss in  20  epoch is  185.630731091
Loss in  25  epoch is  173.179935709
Accuracy Train: 0.961691
Accuracy validation: 0.9562
Accuracy Test: 0.958
USPS Test Accuracy: 0.473374
--- 436.300900220871 seconds ---

Loss in  5  epoch is  18000747.418
Loss in  10  epoch is  15646912.9043
Loss in  15  epoch is  13774894.5742
Accuracy Train: 0.993891
Accuracy validation: 0.98
```

```
33 Accuracy Test: 0.9792
34 USPS Test Accuracy: 0.611831
35 --- 1639.0891389846802 seconds ---
```

Results 2

The following are the brief obserations from the network

Classifier		Accuracy	Comments
Logistic Regression	Test	0.9183	
	Validation	0.9184	
	Training	0.927236	
	USPS	0.382269	
Neural Network	Test	0.958	λ = 0 n = 512
	Validation	0.9562	
	Training	0.961691	
	USPS	0.473374	
Neural Network	Test	0.9626	λ = 0.01 n = 512
	Validation	0.9598	
	Training	0.966055	
	USPS	0.502725	
Neural Network	Test	0.9648	λ = 0.01 n = 256
	Validation	0.9638	
	Training	0.969291	
	USPS	0.500475	
CNN	Test	0.9792	
	Validation	0.98	
	Training	0.993891	
	USPS	0.611831	

- The effect of regularization is pretty evident in the accuracy results.
- The accuracy initailly decreased when ran the same number of epochs but an increase in the number of epochs has increased the accuracy steadily.
- It appears the sweet spot for neural networks in around 256 hidden units.

- The weight regularization has definitely improved the UPSPS accuracy under identitcal conditions.
- Regularization gives us the freedom to have higher number of Epochs because of the confidence of not over fitting. This was clearly observed in neural network and CNN whereas the whole accuracy of Logistic went down where regularisation is applied.
- From the above accuracy results, the 'No Free Lunch' theorem is proved.

Backpropogation

- Measures have taken to maintain the numerical stability of the code such as...
- Normalisisng the image by subtracting the avg of the whole dataset from each iage.
- Softmax is not directly applied, instead the exponentials are just used for denominator as the $\log$ of the numerator results in same value as before $\exp$.
- The test accuracy is 0.91 for Logistic and 0.93 for Neural Networks.

```
#Logistitc
W = np.random.randn(nClasses,nFeatures)
b = np.zeros(nClasses)

def findOut(x):
    z1 = W.dot(x) + b[:,np.newaxis]
    z1 -= np.max(z1,0)
    zExp = np.exp(z1)
    a = zExp/np.sum(zExp,0)
    return a

for _ in range(nEpochs):
    loss = 0.0
    #Forward....
    z = W.dot(trainImages) + b[:,np.newaxis]
    z -= np.max(z,0)
    zExp = np.exp(z)
```

```
18      loss = -np.sum(np.multiply(trainLabels,(z-np.log(np.sum(zExp,0)↩
            ))),0)
19      a = zExp/np.sum(zExp,0)
20      #     print(np.equal(np.argmax(findOut(testImages),0),np.argmax↩
            (testLabels,0)).shape)
21      corrert = np.sum(np.equal(np.argmax(findOut(testImages),0),np.↩
            argmax(testLabels,0)))/testLabels.shape[1]
22      corrert2 = np.sum(np.equal(np.argmax(a,0),np.argmax(trainLabels↩
            ,0)))/m
23      print('Loss:',np.sum(loss),corrert,corrert2)
24
25      #Backward
26      dz = (zExp/np.sum(zExp,0))-trainLabels
27      dW = dz.dot(trainImages.T)
28      db = np.sum(dz,1)
29
30      W = W - learningRate*dW - lambdaa*W
31      b = b - learningRate*db
32
33  #Neural Networks
34  W1 = np.random.randn(nH1,nFeatures)
35  b1 = np.zeros(nH1)
36
37  W2 = np.random.randn(nClasses,nH1)
38  b2 = np.zeros(nClasses)
39
40  def findOut(x):
41      z1 = W1.dot(x) + b1[:,np.newaxis]
42      a1 = np.maximum(z1,0,z1)
43
44      z2 = W2.dot(a1) + b2[:,np.newaxis]
45      z2 -= np.max(z2,0)
46      z2Exp = np.exp(z2)
47      a = z2Exp/np.sum(z2Exp,0)
48      #     print(a.shape)
49      return a
50
51  for _ in range(nEpochs):
52      loss = 0.0
53      #Forward....
54      z1 = W1.dot(trainImages) + b1[:,np.newaxis]
```

```
55      #     a1 = np.maximum(z1,0,z1)
56      a1 = np.maximum(z1,0)
57
58      z2 = W2.dot(a1) + b2[:,np.newaxis]
59      z2 -= np.max(z2,0)
60      z2Exp = np.exp(z2)
61      #     z2Exp[z2Exp<=0] = 1e-10
62      loss = -np.sum(np.multiply(trainLabels,z2-np.log(np.sum(z2Exp↩
            ,0))),0)
63
64      #     print(np.equal(np.argmax(findOut(testImages),0),np.argmax↩
            (testLabels,0)).shape)
65      correct1 = np.sum(np.equal(np.argmax(findOut(trainImages),0),np↩
            .argmax(trainLabels,0)))/m
66      correct2 = np.sum(np.equal(np.argmax(findOut(testImages),0),np.↩
            argmax(testLabels,0)))/testLabels.shape[1]
67      correct3 = np.sum(np.equal(np.argmax(findOut(validationImages)↩
            ,0),np.argmax(validationLabels,0)))/validationLabels.shape↩
            [1]
68      print('Loss:',np.sum(loss), correct1, correct2, correct3)
69
70      #Backward
71      dz2 = (z2Exp/np.sum(z2Exp,0))-trainLabels
72      #     print(np.sum(dz2))
73      dW2 = dz2.dot(a1.T)
74      db2 = np.sum(dz2,1)
75
76      da1 = W2.T.dot(dz2)
77      dz1 = np.zeros_like(da1)
78      dz1[da1>0] = 1
79
80      dW1 = dz1.dot(trainImages.T)
81      db1 = np.sum(dz1,1)
82
83      W2 = W2 - learningRate*dW2 - lambdaa*W2
84      b2 = b2 - learningRate*db2
85      W1 = W1 - learningRate*dW1 - lambdaa*W1
86      b1 = b1 - learningRate*db1
```