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]))}

    w_h1 = tf.summary.histogram("weights1",weights['hiddenL1'])
    w_h2 = tf.summary.histogram("weights2",weights['hiddenL2'])
    b_h1 = tf.summary.histogram("biases1",biases['hiddenL1'])
    b_h2 = tf.summary.histogram("biases2",biases['hiddenL2'])


    with tf.name_scope("a1") as scope:
    z1 = tf.matmul(weights['hiddenL1'],x) + biases['hiddenL1']
    a1 = tf.nn.relu(z1)

    with tf.name_scope("a2") as scope:
    z2 = tf.matmul(weights['hiddenL2'],a1) + biases['hiddenL2']
    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.

```
def convolutionalNeuralNetwork(x):
    weights = {'conv1':tf.Variable(tf.random_normal([5,5,1,32])),
            'conv2':tf.Variable(tf.random_normal([5,5,32,64])),
            'fullyC1':tf.Variable(tf.random_normal([7*7*64,1024])),
            'fullyC2':tf.Variable(tf.random_normal([1024,1024])),
            'out':tf.Variable(tf.random_normal([1024,nClasses]))}

    biases = {'conv1':tf.Variable(tf.random_normal([32])),
            'conv2':tf.Variable(tf.random_normal([64])),
            'fullyC1':tf.Variable(tf.random_normal([1024])),
            'fullyC2':tf.Variable(tf.random_normal([1024])),
            'out':tf.Variable(tf.random_normal([nClasses]))}

    x = tf.transpose(x)

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

```

```
    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)

    with tf.name_scope("training") as scope:
        optimizer = tf.train.AdamOptimizer().minimize(loss)

    with tf.name_scope("accuracy") as scope:
        correct = tf.equal(tf.argmax(y),tf.argmax(t))
        accuracy = tf.reduce_mean(tf.cast(correct,'float'))

    init = tf.global_variables_initializer()
    sess = tf.Session()
    mergeSummary = tf.summary.merge_all()

    sess.run(init)
    summaryWriter = tf.summary.FileWriter('../../TFout/2', sess.graph)

    for epoch in range(nEpochs):
        error = 0.0
        for i in range(int(m/batchSize)):
            batch_xs, batch_ys = mnist.train.next_batch(batchSize)
            _, er, summaryStr = sess.run([optimizer,loss,mergeSummary],feed_dict={x:batch_xs.T,t:batch_ys.T})
            summaryWriter.add_summary(summaryStr, epoch*(int(m/batchSize)) + i)
            error += er
        if (epoch+1)%printEvery == 0:
            print('Loss in ',epoch+1,' epoch is ',error)

    prediction = tf.equal(tf.argmax(y),tf.argmax(t))
    accuracy = tf.reduce_mean(tf.cast(prediction,"float"))
    print("Accuracy Train:", sess.run(accuracy,{x: trainImages, t: trainLabels}))
    print("Accuracy validation:", sess.run(accuracy,{x: validationImages, t: validationLabels}))
    print("Accuracy Test:", sess.run(accuracy,{x: testImages, t: testLabels}))
    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.

Results 2

- 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.
- The learning time increases significantly with the increase in M, the number of basis functions.