Homework1

January 23, 2018

In [ ]:
```
## Importing Libraries
%matplotlib inline
import matplotlib.pyplot as plt
import tensorflow as tf
import numpy as np
from tensorflow.examples.tutorials.mnist import input_data
```

In [ ]:
```
#Read Data
data = input_data.read_data_sets(".",one_hot = True)
```

In [ ]:
```
#fully connected neural network
#Placeholder for input image
img = tf.placeholder(tf.float32, [None, 784])

#Placeholder for one-hot labels
y = tf.placeholder(tf.float32, [None, 10])

#Placeholder for labels(For Val/test)
y_labels = tf.placeholder(tf.int64, [None])

#Weights
w1 = tf.get_variable("w1",[784,1024],initializer = tf.random_normal_initializer(stddev=

#Bias
b1 = tf.get_variable("b1",[1024],initializer = tf.constant_initializer(0.0))

#Linear Layer
h1 = tf.matmul(img,w1) + b1
#Apply Batchnorm
h1 = tf.contrib.layers.batch_norm(h1)
#Apply RELU
h1 = tf.nn.relu(h1)

#Weights
w2 = tf.get_variable("w2",[1024,1024],initializer=tf.random_normal_initializer(stddev =
#bias
b2 = tf.get_variable("b2",[1024],initializer=tf.constant_initializer(0.0))
```

```
        #Linear Layer
        h2 = tf.matmul(h1,w2) + b2
        #Apply Batchnorm
        h2 = tf.contrib.layers.batch_norm(h2)
        #Apply RELU
        h2 = tf.nn.relu(h2)

        #Weights
        w3 = tf.get_variable("w3",[1024,10],initializer=tf.random_normal_initializer(stddev = (
        #bias
        b3 = tf.get_variable("b3",[10],initializer=tf.constant_initializer(0.0))

        #Linear Layer
        logits = tf.matmul(h2, w3) + b3

        #apply Batchnorm
        logits = tf.contrib.layers.batch_norm(logits)
        #Apply Softmax
        output = tf.nn.softmax(logits)
        output_class = tf.argmax(output,axis=1)
```

In [ ]:
```
        #Define loss function
        loss = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(logits = logits, labels =

        #Define Optimizer
        optim = tf.train.GradientDescentOptimizer(learning_rate = 0.1).minimize(loss)
```

In [ ]:
```
        #Accuracy
        correct_labels = tf.equal(output_class, y_labels)
        accuracy = tf.reduce_mean(tf.cast(correct_labels, tf.float32))
```

In [ ]:
```
        #running results for fully connected neural network
        sess = tf.InteractiveSession()
        tf.global_variables_initializer().run()
        #Make batches to train
        num_iter = 1000
        batch_size = 64
        loss_val = []
        iteration = []
        for i in range(num_iter):
            batch_img, batch_y = data.train.next_batch(batch_size)
            _, l = sess.run([optim, loss],feed_dict = {img: batch_img , y: batch_y})
            loss_val.append(l)
            iteration.append(i + 1)
        labels = np.array([label.argmax() for label in data.test.labels])
        accuracy = sess.run([accuracy],feed_dict = {img: data.test.images, y: data.test.labels
        print ("Accuracy" ,accuracy)
```

In [ ]:
```
        ## adding the convolutional neural network
        def weight_variable(shape):
```

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

In [ ]: def bias_variable(shape):
        initial = tf.constant(0.1, shape=shape)
        return tf.Variable(initial)

In [ ]: 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')

In [ ]: #computational graph for convolutional neural network
    img = tf.placeholder(tf.float32, [None, 784])
    x_image = tf.reshape(img, [-1, 28, 28, 1])

    #Placeholder for one-hot labels
    y = tf.placeholder(tf.float32, [None, 10])

    #Placeholder for labels(For Val/test)
    y_labels = tf.placeholder(tf.int64, [None])

    #fisrt convolutional layer
    W_conv1 = weight_variable([5, 5, 1, 32])
    b_conv1 = bias_variable([32])

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

    #second convolutional layer
    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)

    #fully connected layer
    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.matmul(h_pool2_flat, W_fc1) + b_fc1
    #apply batch normalization
    h_fc1 = tf.contrib.layers.batch_norm(h_fc1)
    #apply relu
    h_fc1 = tf.nn.relu(h_fc1)
```

```
        W_fc2 = weight_variable([1024, 10])
        b_fc2 = bias_variable([10])

        logits = tf.matmul(h_fc1, W_fc2) + b_fc2

        #apply Batchnorm
        logits = tf.contrib.layers.batch_norm(logits)
        #Apply Softmax
        output = tf.nn.softmax(logits)
        output_class = tf.argmax(output,axis=1)
```

In [ ]:
```
        #Define loss function
        loss = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(logits = logits, labels =

        #Define Optimizer
        optim = tf.train.AdamOptimizer(learning_rate=1e-4).minimize(loss)
```

In [ ]:
```
        #Accuracy
        correct_labels = tf.equal(output_class, y_labels)
        accuracy = tf.reduce_mean(tf.cast(correct_labels, tf.float32))
```

In [ ]:
```
        #running results for convolutional neural network
        sess = tf.InteractiveSession()
        tf.global_variables_initializer().run()
        #Make batches to train
        num_iter = 1000
        batch_size = 64
        loss_val = []
        iteration = []
        for i in range(num_iter):
            batch_img, batch_y = data.train.next_batch(batch_size)
            _, l = sess.run([optim, loss],feed_dict = {img: batch_img , y: batch_y})
            loss_val.append(l)
            iteration.append(i + 1)
        labels = np.array([label.argmax() for label in data.test.labels])
        accuracy = sess.run([accuracy],feed_dict = {img: data.test.images, y: data.test.labels
        print ("Accuracy" ,accuracy)
```

In [ ]:
```
        #plot loss against iteration
        plt.plot(iteration, loss_val)
        plt.ylabel('Loss')
        plt.xlabel('Iterations')
        plt.show()
```