MNIST/MNIST_Softmax.py
753e6c9 on 5 Jun 2018
@rajitnikhare rajitnikhare Add files via upload

254 lines (125 sloc) 5.41 KB

```
# coding: utf-8

# In[150]:


#Importing tensorflow
import tensorflow as tf


# In[151]:


#Importing MNIST Dataset
from tensorflow.examples.tutorials.mnist import input_data
mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)

# one hot vectors is 0 in most dimension except for a single which denotes a value. Give a
picture example


# In[152]:


#Creating placeholders
```

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

#placeholder is not a variable. It is a value created for the tensorflow to run computation.
#MNIST data is in 28*28 pixel shape which needs to be flattened into 784 dimension vector(28*28 = 784). Thus, a 2-D tensor
#is created with a shape of [None, 784](None means dimension can be of varied length).

explain more about placeholders with pictures and diagram

In[153]:

#Creating weights and biases
w = tf.Variable(tf.zeros([784,10]))
b = tf.Variable(tf.zeros([10]))

#Variables here is a value that lives in TensorFlow`s computation graph. This can be changed and used by the computation.
#initially w, b are zeros of (784,10) and (10) in dimension because it needs to learn the values so it does
#not matter if the initial values are zeros.
#our model is y = softmax((x*w) + b). Thus, w is of the dimension (784*10). Matrix multiplication of x and w is going
#to result in a vector of shape (10). This can be added to b.

#need a diagram here

In[154]:

# our model
y = tf.nn.softmax(tf.matmul(x,w) + b)

#why this model. Explain more about softmax.

```
# In[155]:


#training our model
y_ = tf.placeholder(tf.float32, [None,10])

#cross entropy function
cross_entropy = tf.reduce_mean(-tf.reduce_sum(y_ * tf.log(y), reduction_indices=[1]))


#cross entropy function describes how inefficient our predictions are. It is the loss function.

#what`s cross entropy? Why use it here? equation of cross entropy function?



# In[ ]:


train_step = tf.train.GradientDescentOptimizer(0.5).minimize(cross_entropy)

#gradient optimizer tries to move to that direction in the computation graph where the cost or
loss is reduced.
#learning rate/ step length  is 0.5. It will descend through cross_entropy.

#what`s Gradient descent? Explain with diagram? fast.ai approach is good here


#init = tf.initialize_all_variables()
init = tf.global_variables_initializer()
# Variables are not initialized on their own. The above command is needed to initialize them.


# In[157]:


#interactive session:
sess = tf.Session()
sess.run(init) # reset values to incorrect defaults.

#Tensorflow uses a C++ backend. The connection to the backend is a session. Normally, a
computation graph is
```

```
# created and then the session is launched.


# In[158]:


for i in range(100):
    batch_xs, batch_ys = mnist.train.next_batch(100)
    sess.run(train_step, feed_dict={x : batch_xs, y_: batch_ys})


# In[159]:


#Evaluating the model:
prediction = tf.equal(tf.argmax(y,1), tf.argmax(y_,1))


# In[160]:


accuracy = tf.reduce_mean(tf.cast(prediction, tf.float32))


# In[161]:


print(sess.run(accuracy, feed_dict={x: mnist.test.images, y_: mnist.test.labels}))


# In[162]:


#  Multilayer Convolutional Layer:

#functions to define weights and biases to be used in the computation ahead.
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)
```

```
# In[175]:


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[177]:


get_ipython().run_line_magic('pinfo', 'tf.nn.conv2d')


# In[164]:


#First Convolutional Layer
W_conv1 = weight_variable([5, 5, 1, 32])
b_conv1 = bias_variable([32])


# In[165]:


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


# In[166]:


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


# In[167]:
```

```
#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)


# In[168]:


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)


# In[169]:


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


# In[170]:


#Readout Layer


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

y_conv = tf.matmul(h_fc1_drop, W_fc2) + b_fc2


# In[174]:


with tf.Session() as sess:
```

```
cross_entropy = tf.reduce_mean(
    tf.nn.softmax_cross_entropy_with_logits(labels=y_, logits=y_conv))
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.global_variables_initializer())

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})

print("test accuracy %g"%accuracy.eval(feed_dict={
    x: mnist.test.images, y_: mnist.test.labels, keep_prob: 1.0}))
```