tensorflow快速入门

变量定义

有初始值的变量

```
matrix1=tf.constant([[3,3]])
matrix2=tf.constant([[2],[2]])

product = tf.matmul(matrix1,matrix2)

# #method 1
# sess = tf.Session()
# result = sess.run(product)
# print(result)
# sess.close()

# method 2
with tf.Session() as sess:
    result2 = sess.run(product)
    print(result2)
```

无初始值的变量定义使用占位符placeholder占位，后续加入值

```
import tensorflow as tf

input1 = tf.placeholder(tf.float32)
input2 = tf.placeholder(tf.float32)

output = tf.multiply(input1,input2)

with tf.Session() as sess:
    print(sess.run(output,feed_dict={input1:[7.],input2:[2.]}))
```

常量定义

```
import tensorflow as tf

state = tf.Variable(0,name='counter')
#print(state.name)
one = tf.constant(1)

new_value = tf.add(state , one)
update = tf.assign(state,new_value)

init = tf.global_variables_initializer() #must have if define variable

with tf.Session() as sess:
    sess.run(init)
    for _ in range(3):
        sess.run(update)
        print(sess.run(state))
```

添加层

```
def add_layer(inputs,in_size,out_size,activation_function=None):
    #add one more layer and return the output of this layer
    Weights = tf.Variable(tf.random_normal([in_size,out_size]))
    biases = tf.Variable(tf.zeros([1,out_size])+0.1)
    Wx_plus_b = tf.add(tf.matmul(inputs,Weights),biases)
    if activation_function is None:
        outputs = Wx_plus_b
    else:
        outputs = activation_function(Wx_plus_b)
    return outputs
```

添加层和近似图

```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt

def add_layer(inputs,in_size,out_size,n_layer,activation_function=None):
    #add one more layer and return the output of this layer
    layer_name = 'layer%s' % n_layer
    with tf.name_scope('layer'):
        with tf.name_scope('weights'):
            Weights = tf.Variable(tf.random_normal([in_size, out_size]),name='W')
            tf.summary.histogram(layer_name+'/weights',Weights)
        with tf.name_scope('biases'):
            biases = tf.Variable(tf.zeros([1, out_size]) + 0.1,name='b')
            tf.summary.histogram(layer_name + '/biases', biases)
        with tf.name_scope('Wx_plus_b'):
            Wx_plus_b = tf.add(tf.matmul(inputs, Weights) , biases)
        if activation_function is None:
            outputs = Wx_plus_b
        else:
            outputs = activation_function(Wx_plus_b)
        tf.summary.histogram(layer_name + '/outputs', outputs)
        return outputs

x_data = np.linspace(-1,1,300)[:,np.newaxis]
noise = np.random.normal(0,0.05,x_data.shape)
y_data = np.square(x_data) - 0.5 + noise

#define placeholder for inputs to network
with tf.name_scope('inputs'):
    xs = tf.placeholder(tf.float32, [None, 1], name='x_input')
    ys = tf.placeholder(tf.float32, [None, 1], name='y_input')

# add hidden layer
l1 = add_layer(xs,1,10,n_layer=1,activation_function=tf.nn.relu)
# add output layer
predition = add_layer(l1,10,1,n_layer=1,activation_function=None)

# thr error between prediction and real data
with tf.name_scope('loss'):
    loss = tf.reduce_mean(tf.reduce_sum(tf.square(ys-predition),reduction_indices=[1]))
```

```
    tf.summary.scalar('loss',loss)
with tf.name_scope('train'):
    train_step = tf.train.GradientDescentOptimizer(0.1).minimize(loss)

init = tf.global_variables_initializer()

with tf.Session() as sess:
    merged = tf.summary.merge_all()
    writer = tf.summary.FileWriter("logs/",sess.graph)
    sess.run(init)

    fig = plt.figure()
    ax = fig.add_subplot(1,1,1)
    ax.scatter(x_data,y_data)
    plt.ion()
    plt.show()

    for i in range(1000):
        sess.run(train_step,feed_dict={xs:x_data,ys:y_data})
        if i%50 == 0 :
            #print(sess.run(loss,feed_dict={xs:x_data,ys:y_data}))
            # try:
            #     ax.lines.remove(lines[0])
            # except Exception:
            #     pass
            # predition_value = sess.run(predition,feed_dict={xs:x_data,ys:y_data})
            # lines = ax.plot(x_data,predition_value,'r-',lw=5)
            # plt.pause(0.1)

            result = sess.run(merged,feed_dict={xs:x_data,ys:y_data})
            writer.add_summary(result,i)
```

运行tensorboard

```
tensorboard --logdir=log
```

mnist数据集—— 分类学习

```
import tensorflow as tf
from tensorflow.examples.tutorials.mnist import input_data

#number 1 to 10 data
mnist = input_data.read_data_sets('Mnist_data/',one_hot=True)

def add_layer(inputs,in_size,out_size,activation_function=None):
    #add one more layer and return the output of this layer
    Weights = tf.Variable(tf.random_normal([in_size,out_size]))
    biases = tf.Variable(tf.zeros([1,out_size])+0.1)
    Wx_plus_b = tf.add(tf.matmul(inputs,Weights),biases)
    if activation_function is None:
        outputs = Wx_plus_b
    else:
        outputs = activation_function(Wx_plus_b)
    return outputs
```

```
def compute_accuracy(v_xs,v_ys):
    global prediction
    y_pre = sess.run(prediction,feed_dict={xs:v_xs})
    correct_prediction = tf.equal(tf.argmax(y_pre,1),tf.argmax(v_ys,1))
    accuracy = tf.reduce_mean(tf.cast(correct_prediction,tf.float32))
    result = sess.run(accuracy,feed_dict={xs:v_xs,ys:v_ys})
    return result


# define placeholder for inputs to network
xs = tf.placeholder(tf.float32,[None,784]) #28X28
ys = tf.placeholder(tf.float32,[None,10])

# add output layer
prediction = add_layer(xs,784,10,activation_function=tf.nn.softmax)

# the error between prediction and real data
cross_entropy = tf.reduce_mean(-tf.reduce_sum(ys*tf.log(prediction),reduction_indices=[1])) #loss
train_step = tf.train.GradientDescentOptimizer(0.5).minimize(cross_entropy)

with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    for i in range(1000):
        batch_xs,batch_ys = mnist.train.next_batch(100)
        sess.run(train_step,feed_dict={xs:batch_xs,ys:batch_ys})
        if i%50 == 0:
            print(compute_accuracy(mnist.test.images,mnist.test.labels))
```

使用dropout处理过拟合问题

```
import tensorflow as tf
from sklearn.datasets import load_digits
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelBinarizer

#load data
digits = load_digits()
X = digits.data
y = digits.target
y = LabelBinarizer().fit_transform(y)
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=.3)

def add_layer(inputs,in_size,out_size,layer_name,activation_function=None):
    #add one more layer and return the output of this layer
    Weights = tf.Variable(tf.random_normal([in_size,out_size]))
    biases = tf.Variable(tf.zeros([1,out_size])+0.1)
    Wx_plus_b = tf.add(tf.matmul(inputs,Weights),biases)
    Wx_plus_b = tf.nn.dropout(Wx_plus_b,keep_prob)
    if activation_function is None:
        outputs = Wx_plus_b
    else:
        outputs = activation_function(Wx_plus_b)
    tf.summary.histogram(layer_name+'outputs',outputs)
    return outputs

#define placeholder for inputs to network
keep_prob = tf.placeholder(tf.float32)
xs = tf.placeholder(tf.float32,[None,64]) #8X8
ys = tf.placeholder(tf.float32,[None,10])
```

```
#add output layer
l1 = add_layer(xs,64,50,'l1',activation_function=tf.nn.tanh)
prediction = add_layer(l1,50,10,'l2',activation_function=tf.nn.softmax)

#the loss between prediction and real data
cross_entropy = tf.reduce_mean(-tf.reduce_sum(ys*tf.log(prediction),reduction_indices=[1])) #loss
tf.summary.scalar('loss',cross_entropy)
train_step = tf.train.GradientDescentOptimizer(0.6).minimize(cross_entropy)

with tf.Session() as sess:
    merged = tf.summary.merge_all()
    #summery writer goes in here
    train_writer = tf.summary.FileWriter("logs/train",sess.graph)
    test_writer = tf.summary.FileWriter("logs/test",sess.graph)

    sess.run(tf.global_variables_initializer())
    for i in range(500):
        sess.run(train_step,feed_dict={xs:X_train,ys:y_train,keep_prob:0.5})
        if i%50 == 0:
            #record loss
            train_result = sess.run(merged,feed_dict={xs:X_train,ys:y_train,keep_prob:1})
            test_result = sess.run(merged,feed_dict={xs:X_test,ys:y_test,keep_prob:1})
            train_writer.add_summary(train_result,i)
            test_writer.add_summary(test_result,i)
```