基于 LSTM 神经网络的序列检测 DEMO

By QY

LSTM（Long Short Term Memory），是 RNN 的变种，和 RNN 的主要区别在于 LSTM 能够处理预测信息和相关信息相隔非常远的问题，在 LSTM 中可以控制需要记忆什么信息，需要遗忘什么信息。

具体理论可以看博文 Understanding LSTM Networks

http://colah.github.io/posts/2015-08-Understanding-LSTMs/

序列预测

在对理论有了初步的理解之后，使用 LSTM 实现对操作序列的分类问题。输入数据格式采用 csv 格式。

CSV（comma separated values）逗号分割值文件，文件以纯文本形式存储表格数据，主要用于存储数字和字符，由任意数目的记录组成，记录间以换行符分割；每条记录由字段组成，字段间的分隔符是英文逗号。

输入数据格式（已经预处理完）

```
1,0041,0012,0020,0059,0035,0018,0066,0011,0057,0035,0019,0036,0038,004
1,0058,0057,0021,0007,0005,0034,0020,0030,0041,0037,0024,0061,0039,007
1,0037,0011,0006,0052,0046,0013,0017,0043,0053,0042,0017,0046,0069,004
1,0022,0006,0013,0043,0038,0010,0039,0055,0014,0055,0024,0058,0017,003
```

每一行代表一个操作序列，每行的第一个数字 label 代表操作序列的正负，label=0 代表该操作序列为异常操作，label=1 代表是正常操作。

代码结构

1. 导入数据

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

rnn_unit=10       #隐藏层节点数
input_size=1438   #每一条输入序列的维度
```

```
output_size=1      #输出的大小   (0,1)之间的一个数表示概率
lr=0.0006          #学习率
th = 0.6            #阈值，用来判定多少概率以上的算是正常序列
n_train = 1750      #测试集数量


f=open('data4.csv')
df=pd.read_csv('data4.csv')    #读入数据
if df.empty:
   print("Data is empty")
data=df.iloc[0:8750:,0:1439].values
checkpoint_dir = ''
```

2. 生成训练集测试集

```
def get_train_data(batch_size=80,time_step=20,train_begin=0,train_end=7000):
    batch_index=[]
    data_train=data[train_begin:train_end]           #从输入数据中划分一部分为
训练集
    normalized_train_data=(data_train-
np.mean(data_train,axis=0))/np.std(data_train,axis=0)

    train_x,train_y=[],[]                #x 表示序列、y 表示 label

    for i in range(len(normalized_train_data)-20):
       #print(i)
       if i % batch_size==0:
           batch_index.append(i)
       x=normalized_train_data[i:i+time_step,1:1439]
       y=data[i:i+time_step,0,np.newaxis]
       train_x.append(x.tolist())
       train_y.append(y.tolist())
    batch_index.append((len(normalized_train_data)-20))
    print(train_y)
    print("finish1")
    return batch_index,train_x,train_y
```

Batch_size 每批训练的样本数，time_step 时间步

```
def get_test_data(time_step=20,test_begin=7000):
    data_test=data[test_begin:]
    #print(data_test)
    mean=np.mean(data_test,axis=0)                          #列平均值
    std=np.std(data_test,axis=0)                            #列标准差
    normalized_test_data=(data_test-mean)/std
```

```
    size=(len(normalized_test_data)+time_step-1)//time_step
    test_x,test_y=[],[]
    i=0
    print("enter test")
    for i in range(size-1):
       x=normalized_test_data[i*time_step:(i+1)*time_step,:1438]
       y=data[i*time_step:(i+1)*time_step,0]
       test_x.append(x.tolist())
       test_y.extend(y)
       i = i+1
    test_x.append((normalized_test_data[(i+1)*time_step:,1:1439]).tolist())
    test_y.extend((normalized_test_data[(i+1)*time_step:,0]).tolist())
    return mean,std,test_x,test_y
```

3. 构建神经网络

```
def lstm_1(X):
    batch_size=tf.shape(X)[0]
    time_step=tf.shape(X)[1]
    print("shape secc1")
    w_in=weights['in']
    b_in=biases['in']
    input=tf.reshape(X,[-1,input_size])
    input_rnn=tf.matmul(input,w_in)+b_in
    input_rnn=tf.reshape(input_rnn,[-1,time_step,rnn_unit])
    cell=tf.contrib.rnn.BasicLSTMCell(rnn_unit)
    init_state=cell.zero_state(batch_size,dtype=tf.float32)
    with tf.variable_scope('lstm') as lstm1:
          output_rnn,final_states=tf.nn.dynamic_rnn(cell,
input_rnn,initial_state=init_state, dtype = tf.float32)
    output=tf.reshape(output_rnn,[-1,rnn_unit])
    w_out=weights['out']
    b_out=biases['out']
    pred=tf.matmul(output,w_out)+b_out
    return pred,final_states

def lstm_2(X):
    batch_size=tf.shape(X)[0]
    time_step=tf.shape(X)[1]
    print("shape secc1")
    w_in=weights['in']
    b_in=biases['in']
    input=tf.reshape(X,[-1,input_size])
```

```
    input_rnn=tf.matmul(input,w_in)+b_in
    input_rnn=tf.reshape(input_rnn,[-1,time_step,rnn_unit])
    cell=tf.contrib.rnn.BasicLSTMCell(rnn_unit)
    init_state=cell.zero_state(batch_size,dtype=tf.float32)
    with tf.variable_scope('lstm', reuse=True) as lstm2:
          output_rnn,final_states=tf.nn.dynamic_rnn(cell,
input_rnn,initial_state=init_state, dtype = tf.float32)
    output=tf.reshape(output_rnn,[-1,rnn_unit])
    w_out=weights['out']
    b_out=biases['out']
    pred=tf.matmul(output,w_out)+b_out
    return pred,final_states
```

为了解决 RNN 中遇到的如下问题

ValueError: Variable rnn/rnn/basic_lstm_cell/weights already exists, disallowed.

定义了两个 LSTM 类，其实没什么区别

4. 训练模型

```
def train_lstm(batch_size=80,time_step=20,train_begin=0,train_end=7000):
    X=tf.placeholder(tf.float32, shape=[None,time_step,input_size])
    Y=tf.placeholder(tf.float32, shape=[None,time_step,output_size])
    print("enter train")

batch_index,train_x,train_y=get_train_data(batch_size,time_step,train_begin,
train_end)
    print("get train data")
    pred,_=lstm_1(X)
    print("finish model")
    loss1 = 0

    loss=tf.reduce_mean(tf.square(tf.reshape(pred,[-1])-tf.reshape(Y, [-1])))
    train_op=tf.train.AdamOptimizer(lr).minimize(loss)
    saver=tf.train.Saver(tf.global_variables(),max_to_keep=15)
    #module_file = tf.train.latest_checkpoint('')        #模型路经
    print("enter train")
    with tf.Session() as sess:
        sess.run(tf.global_variables_initializer())        #初次训练
        #saver.restore(sess, module_file)             #使用之前训练好的模型

        for i in range(2000):         #训练次数
```

```
            for step in range(len(batch_index)-1):
                _,loss1
=sess.run([train_op,loss],feed_dict={X:train_x[batch_index[step]:batch_index
[step+1]],Y:train_y[batch_index[step]:batch_index[step+1]]})
            print("loss")
            print(i,loss1)
            if i % 20==0:
                print("save                                                    model:
",saver.save(sess,'D:\LSTM1\stock2.model',global_step=i))
                print("finish!")
```

如果需要基于已有的模型来进行训练，那么就使用

module_file=tf.train.latest_checkpoint()和

sever.store(sess,module.file)。

第一次训练使用 sess.run(tf.global_variables_initializer())

5. 测试模型

```
def prediction(time_step=20):
    X=tf.placeholder(tf.float32, shape=[None,time_step,input_size])
    #Y=tf.placeholder(tf.float32, shape=[None,time_step,output_size])
    mean,std,test_x,test_y=get_test_data(time_step)
    #print(test_x)
    pred,_=lstm_2(X)
    saver=tf.train.Saver(tf.global_variables())
    result = []
    sum_acc = 0
    with tf.Session() as sess:

        module_file = tf.train.latest_checkpoint('')
        saver.restore(sess, module_file)
        test_predict=[]
        print("acc")
        i=0
        for step in range(len(test_x)-1):
          prob=sess.run(pred,feed_dict={X:[test_x[step]]})
          predict=prob.reshape((-1))
          test_predict.extend(predict)
        print("acc finish")
        #test_y=np.array(test_y)*std[0]+mean[0]
        #test_predict=np.array(test_predict)*std[0]+mean[0]
```

```
        #acc=np.average(np.abs(test_predict-
test_y[:len(test_predict)])/test_y[:len(test_predict)])
        i=0
        max_pre = max(test_predict)
        while(i<len(test_predict)):
            test_predict[i] = (test_predict[i] + max_pre)/(2*max_pre)
            i=i+1
        i=0
        print(len(test_predict))
        while(i<len(test_predict)):
            #print(i)
            if(test_predict[i] > th):
                result.append(1)
            else:
                result.append(0)
            i=i+1
        i=0
        print(result)
        while(i<len(test_predict)):
            print(sum_acc)
            if(result[i] == test_y[i]):
                    sum_acc = sum_acc+1
            i=i+1
        acc = sum_acc/n_train
```

测试方法主要是通过向已经训练好的模型中输入测试序列，得到输出的概率再和阈值作比较然后计算所有测试集中正确序列数量。

其他

USL&SL

无监督和有监督的训练的转换只需要改变是否在训练的时候讲 train_y 加入网络的输入即可。

数据集的读入

数据集的读入调用的是 pandas 库，其好处就是可以运用 dataframe 的结构来保存数据，数据的访问获取可以通过切片的方式来实现：

Data = df.iloc[a,b:c,d].values

其中 a,b 维度表示行，a 表示起始行，b 表示结束行。c,d 维度表示列，c 表示起始列，d 表示结束列。参数的缺省值表示从头开始或者一直到结尾。