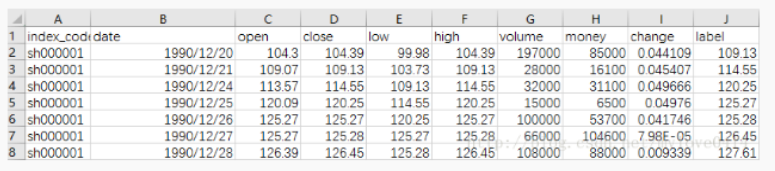
TENSORFLOW Examples of use LSTM forecast stock daily maximum price.

---------------------------------------------------------------------------------------------------

Based on the lowest price, the highest price, the opening price, the closing price, the trading volume, the transaction volume, or the increase in the stock's historical data, the highest on the next day's stock Price to predict.

The data used in the experiment looks like this:



Is the label y, which is the next day's highest price. Column C - I is the input feature.

  This example uses the previous 5800 data to do training data.

Single factor input feature and RNN, LSTM presentation Please stamp a Tensorflow example: Use LSTM to forecast stock daily maximum.

## Import the package and declare the constant

import pandas as pd

import numpy as np

import tensorflow as tf

# Define constants

rnn\_unit=10       #hidden layer units

input\_size=7

output\_size=1

lr=0.0006

f=open('datas.csv')

df=pd.read\_csv(f)     # Read the stock data

data=df.iloc[:,2:10].values   # Take the first 3-10 column

Taking into account the real training environment, here the number of training samples per batch (time\_size), time step (time\_step), the number of training set (train\_begin, train\_end) set as parameters, making the training more mobile.

#—————————— Get training set

def get\_train\_data(batch\_size=60,time\_step=20,train\_begin=0,train\_end=5800):

    batch\_index=[]

    data\_train=data[train\_begin:train\_end]

    normalized\_train\_data=(data\_train-np.mean(data\_train,axis=0))/np.std(data\_train,axis=0)  # standardization

    train\_x,train\_y=[],[]   # Training set x and y are defined first

    for i in range(len(normalized\_train\_data)-time\_step):

       if i % batch\_size==0:

           batch\_index.append(i)

       x=normalized\_train\_data[i:i+time\_step,:7]

       y=normalized\_train\_data[i:i+time\_step,7,np.newaxis]

       train\_x.append(x.tolist())

       train\_y.append(y.tolist())

    batch\_index.append((len(normalized\_train\_data)-time\_step))

    return batch\_index,train\_x,train\_y

#—————————— Get the test set

def get\_test\_data(time\_step=20,test\_begin=5800):

    data\_test=data[test\_begin:]

    mean=np.mean(data\_test,axis=0)

    std=np.std(data\_test,axis=0)

    normalized\_test\_data=(data\_test-mean)/std  # standardization

    size=(len(normalized\_test\_data)+time\_step-1)//time\_step  # There are size samples

    test\_x,test\_y=[],[]

    for i in range(size-1):

       x=normalized\_test\_data[i\*time\_step:(i+1)\*time\_step,:7]

       y=normalized\_test\_data[i\*time\_step:(i+1)\*time\_step,7]

       test\_x.append(x.tolist())

       test\_y.extend(y)

    test\_x.append((normalized\_test\_data[(i+1)\*time\_step:,:7]).tolist())

    test\_y.extend((normalized\_test\_data[(i+1)\*time\_step:,7]).tolist())

    return mean,std,test\_x,test\_y

**Constructing neural networks**

#—————————————————— Define neural network variables

def lstm(X):

    batch\_size=tf.shape(X)[0]

    time\_step=tf.shape(X)[1]

    w\_in=weights['in']

    b\_in=biases['in']

    input=tf.reshape(X,[-1,input\_size])  # Need to turn tensor into two-dimensional calculation, the calculated results as a hidden layer of the input.

    input\_rnn=tf.matmul(input,w\_in)+b\_in

    input\_rnn=tf.reshape(input\_rnn,[-1,time\_step,rnn\_unit])  # Turn tensor into 3-dimensional, as lstm cell input

    cell=tf.nn.rnn\_cell.BasicLSTMCell(rnn\_unit)

    init\_state=cell.zero\_state(batch\_size,dtype=tf.float32)

    output\_rnn,final\_states=tf.nn.dynamic\_rnn(cell, input\_rnn,initial\_state=init\_state, dtype=tf.float32)  # Output\_rnn is the result of recording each output node lstm, final\_states is the result of the last cell

    output=tf.reshape(output\_rnn,[-1,rnn\_unit]) # As input to the output layer

    w\_out=weights['out']

    b\_out=biases['out']

    pred=tf.matmul(output,w\_out)+b\_out

    return pred,final\_states

**Training model**

#—————————————————— Training model

def train\_lstm(batch\_size=80,time\_step=15,train\_begin=0,train\_end=5800):

    X=tf.placeholder(tf.float32, shape=[None,time\_step,input\_size])

    Y=tf.placeholder(tf.float32, shape=[None,time\_step,output\_size])

    batch\_index,train\_x,train\_y=get\_train\_data(batch\_size,time\_step,train\_begin,train\_end)

    pred,\_=lstm(X)

    # Loss function

    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()

    with tf.Session() as sess:

        #sess.run(tf.global\_variables\_initializer())

        saver.restore(sess, module\_file)

        # Repeated training 2000 times

        for i in range(2000):

            for step in range(len(batch\_index)-1):

                \_,loss\_=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(i,loss\_)

            if i % 200==0:

                print("Save the model",saver.save(sess,'stock2.model',global\_step=i

**Well, here that the parameters here are based on the parameters of the existing model recovery, meaning that before training the model, save the parameters of the neural network, and then take out as the initialization parameters and then training. If it is the first training, use sess.run (tf.global\_variables\_initializer ()), also do not use module\_file = tf.train.latest\_checkpoint () and saver.store (sess, module\_file).**

**Test**

#———————————————— Forecasting model

def prediction(time\_step=20):

    X=tf.placeholder(tf.float32, shape=[None,time\_step,input\_size])

    mean,std,test\_x,test\_y=get\_test\_data(time\_step)

    pred,\_=lstm(X)

    saver=tf.train.Saver(tf.global\_variables())

    with tf.Session() as sess:

        # Parameter recovery

        module\_file = tf.train.latest\_checkpoint()

        saver.restore(sess, module\_file)

        test\_predict=[]

        for step in range(len(test\_x)-1):

          prob=sess.run(pred,feed\_dict={X:[test\_x[step]]})

          predict=prob.reshape((-1))

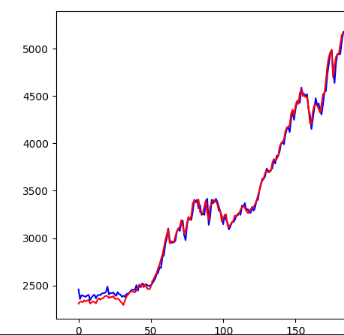
          test\_predict.extend(predict)

        test\_y=np.array(test\_y)\*std[7]+mean[7]

        test\_predict=np.array(test\_predict)\*std[7]+mean[7]

        acc=np.average(np.abs(test\_predict-test\_y[:len(test\_predict)])/test\_y[:len(test\_predict)]) # Acc is the test set deviation

**The final result is drawn like this:**



**The red polyline is the true value, and the blue polyline is the predicted value**

**The deviation is about 1.36%**

**Code and data uploaded to github, want to poke all the code**