## 标普500股票价格大盘回归实验

|  |  |
| --- | --- |
| 【实验名称】 | 标普500股票价格大盘回归实验 |
| 【知识点准备】 | |
| Python基础，tensorflow，科学计算包基础（pandas，numpy） | |
| 【实验目的】 | |
| 1. 熟悉深度学习模型构建过程及训练流程  2. 掌握深度学习的基本套路 | |
| 【实验内容】 | |
| 加载csv文件中的数据  查看数据整体情况  对数据进行处理  构建网络  分批训练 | |
| 【实验要求】 | |
| 在预测数据上的损失小于0.2 | |
| 【实验步骤】 | |
| * **Tensorflow实现**  1. 打开jupyter新建python3文件，将标普500数据上传至同级目录，引入必要模块   import pandas as pd  import numpy as np import tensorflow as tf import matplotlib.pyplot as plt %matplotlib inline from sklearn.preprocessing import MinMaxScaler import time  2 .加载训练数据  #panda加载csv文件  data = pd.read\_csv('data\_stocks.csv') #查看总览信息，最大值最小值均值 data.describe()  3 .查看数据概况  3.1查看表的行数、列数、大小、字段的类型  data.info()    3.2查看数据的前5行    3.3查看时间范围  #查看时间、将时间戳转化为时间  print(time.strftime('%Y-%m-%d',time.localtime(data['DATE'].min()))) print(time.strftime('%Y-%m-%d',time.localtime(data['DATE'].max())))    3.4查看大盘指数的变化趋势  #打印出来大盘指数的变化趋势  plt.plot(data['SP500'])    4处理数据  4.1删除时间戳列  #删除时间戳列  data.drop('DATE',axis=1,inplace=True) data.head()  4.2制定训练集和测试集  #将所有数据分成训练集和测试集  data\_train = data.iloc[:int(data.shape[0]\*0.8),:] data\_test = data.iloc[int(data.shape[0]\*0.8):,:] print(data\_train.shape,data\_test.shape)    4.3归一化处理  #进行归一化处理  scaler = MinMaxScaler(feature\_range=(-1,1)) scaler.fit(data\_train) data\_train = scaler.transform(data\_train) data\_test = scaler.transform(data\_test)  4.4定义输入和输出（端到端）  #将训练数据分出输入和输出  X\_train = data\_train[:,1:] y\_train = data\_train[:,0] #将测试数据分为输入和输出 X\_test = data\_test[:,1:] y\_test = data\_test[:,0]  5构建网络  #定义各层层数  input\_dim = X\_train.shape[1] hidden1 = 1024 hidden2 = 512 hidden3 = 256 hidden4 = 128 output\_dim = 1  #定义批的大小 batch\_size = 256 #定义跑多少轮 epochs = 5  #重置计算图（为了安全） tf.reset\_default\_graph()  #定义输入层 X = tf.placeholder(shape=[None,input\_dim],dtype=tf.float32) Y = tf.placeholder(shape=[None],dtype=tf.float32)  #定义隐层变量 W1 = tf.get\_variable('W1',[input\_dim,hidden1],initializer=tf.contrib.layers.xavier\_initializer(seed = 1)) b1 = tf.get\_variable('b1',[hidden1],initializer=tf.zeros\_initializer())  W2 = tf.get\_variable('W2',[hidden1,hidden2],initializer=tf.contrib.layers.xavier\_initializer(seed = 1)) b2 = tf.get\_variable('b2',[hidden2],initializer=tf.zeros\_initializer())  W3 = tf.get\_variable('W3',[hidden2,hidden3],initializer=tf.contrib.layers.xavier\_initializer(seed = 1)) b3 = tf.get\_variable('b3',[hidden3],initializer=tf.zeros\_initializer())  W4 = tf.get\_variable('W4',[hidden3,hidden4],initializer=tf.contrib.layers.xavier\_initializer(seed = 1)) b4 = tf.get\_variable('b4',[hidden4],initializer=tf.zeros\_initializer())  W5 = tf.get\_variable('W5',[hidden4,output\_dim],initializer=tf.contrib.layers.xavier\_initializer(seed = 1)) b5 = tf.get\_variable('b5',[output\_dim],initializer=tf.zeros\_initializer())  #定义logits回归 h1 = tf.nn.relu(tf.add(tf.matmul(X,W1),b1)) h2 = tf.nn.relu(tf.add(tf.matmul(h1,W2),b2)) h3 = tf.nn.relu(tf.add(tf.matmul(h2,W3),b3)) h4 = tf.nn.relu(tf.add(tf.matmul(h3,W4),b4)) out = tf.transpose(tf.add(tf.matmul(h4,W5),b5))  #定义损失 cost = tf.reduce\_mean(tf.squared\_difference(out,Y)) #定义优化方式 optimizer = tf.train.AdamOptimizer().minimize(cost)  6开始图的运算  #开始图的运行阶段  with tf.Session() as sess:  sess.run(tf.global\_variables\_initializer())    for e in range(epochs):  shuffle\_indices = np.random.permutation(np.arange(y\_train.shape[0]))  X\_train = X\_train[shuffle\_indices]  y\_train = y\_train[shuffle\_indices]    for i in range(y\_train.shape[0]//batch\_size):  start = i \* batch\_size  batch\_x = X\_train[start : start + batch\_size]  batch\_y = y\_train[start: start + batch\_size]    sess.run(optimizer,feed\_dict={X:batch\_x,Y: batch\_y})     #到此处图的定义及图的运行已完毕  # 看训练的过程  if i % 50 == 0:  saver = tf.train.Saver()  saver.save(sess, "./model.ckpt", meta\_graph\_suffix='meta', write\_meta\_graph=True)  tf.train.write\_graph(sess.graph\_def, "./Model", "model.pb", False)  print("在训练集上的损失：",sess.run(cost,feed\_dict={X:X\_train,Y:y\_train}))  print("在测试集上的损失：",sess.run(cost,feed\_dict={X:X\_test,Y:y\_test}))  #预测值  y\_pred = sess.run(out,feed\_dict={X:X\_test})  y\_pred = np.squeeze(y\_pred)  plt.plot(y\_test,label = 'test')  plt.plot(y\_pred,label = 'pred')  plt.title("这是第"+str(e) + "次循环的第"+str(i)+"批")  plt.legend()  plt.show()  开始时：    训练N轮后： | |
| * **keras实现**   from keras.layers import Input, Dense  from keras.models import Model  X\_train = data\_train[:, 1:] y\_train = data\_train[:, 0] X\_test = data\_test[:, 1:] y\_test = data\_test[:, 0]  input\_dim = X\_train.shape[1] hidden\_1 = 1024 hidden\_2 = 512 hidden\_3 = 256 hidden\_4 = 128 output\_dim = 1 batch\_size = 256 epochs = 10  X = Input(shape=[input\_dim,]) h = Dense(hidden\_1, activation='relu')(X) h = Dense(hidden\_2, activation='relu')(h) h = Dense(hidden\_3, activation='relu')(h) h = Dense(hidden\_4, activation='relu')(h) Y = Dense(output\_dim, activation='sigmoid')(h)  model = Model(X, Y) model.compile(loss='mean\_squared\_error', optimizer='adam') model.fit(X\_train, y\_train, epochs=epochs, batch\_size=batch\_size, shuffle=False) y\_pred = model.predict(X\_test) print('MSE Train:', model.evaluate(X\_train, y\_train, batch\_size=batch\_size)) print('MSE Test:', model.evaluate(X\_test, y\_test, batch\_size=batch\_size)) plt.plot(y\_test, label='test') plt.plot(y\_pred, label='pred') plt.legend() plt.show() | |