为了预测股票，我用TensorFlow深度学习了股市数据

2019年02月26日

> 完整源码可在微信公众号：「01二进制」后台回复：「股市分析」获取
>
> 阅读此文前建议先阅读《找对象的过程中，我竟然理解了什么是机器学习！》

前言

相信大家这几天或多或少的都开始关注到股市了，虽然我还不是很懂里面的一些套路，但是从最近各个公众号的推送里面，我也看到最近的股市确实是形势大好。对很多人来说，股票就和房价一样，他的升与降牵动着众多人的心。这几天很多qq群、微信群都开始讨论起股票了，各位坊间大神也纷纷开始预测各种股票走势了。

股票价格是典型的时间序列数据（简称时序数据），会受到经济环境、政府政策、人为操作多种复杂因素的影响，所以说股票价格预测是一件非常唬人的事情。但是基于历史数据，以股票价格为例，介绍如何对时序数据进行预测，仍然值得一做。

不过今天我们不聊股市，因为我也不是很懂，今天我们来聊聊我们知道的东西。如今深度学习在金融市场的应用越来越广泛，我们能否利用已有的历史数据通过深度学习的方式来进行预测呢?

准备工作

答案自然是可以的，虽然我们无法非常准确的进行预测，但是作为一个深度学习入手的项目是完完全全可以的。

实验环境

老样子，以免出现因环境导致的问题，先将实验环境列出：

- MacOS 10.14.3
- Python 3.6.8（Anaconda）
- Jupyter Notebook
- 使用的包有：
    - TensorFlow
    - Keras
    - matplotlib
    - pandas

数据

此次实验，我们采用**STATWORX**的`S&P 500`股价数据，该数据集爬取自Google Finance API，已经进行过缺失值处理。他们的官方网站是：www.statworx.com/。

> 数据集可在微信公众号：「01二进制」后台回复：「股市数据集」获取

数据预览

在这里我们还是使用pandas，主要用于数据清洗和整理

```
import pandas as pd
data=pd.read_csv('data/data_stocks.csv')
data.info()
复制代码
```

执行`data.info()`可以查看特征的概要：

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 41266 entries, 0 to 41265
Columns: 502 entries, DATE to NYSE.ZTS
dtypes: float64(501), int64(1)
memory usage: 158.0 MB
复制代码
```

从上述结果可知：该数据集数据共502列，41266行，502列分别为：

- DATE：该行数据的时间戳
- SP500：可以理解为大盘指数
- 其他：可以理解为500支个股的股价

查看数据的前五行

```
data.head()
```
复制代码

```
data.head()
```

	DATE	SP500	NASDAQ.AAL	NASDAQ.AAPL	NASDAQ.ADBE	NASDAQ.ADI	NASDAQ.ADP	NASDAQ.ADSK	NASDAQ.AKAM	NASDAQ.ALXN	...	NY
0	1491226200	2363.6101	42.3300	143.6800	129.6300	82.040	102.2300	85.2200	59.760	121.52	...	
1	1491226260	2364.1001	42.3600	143.7000	130.3200	82.080	102.1400	85.6500	59.840	121.48	...	
2	1491226320	2362.6799	42.3100	143.6901	130.2250	82.030	102.2125	85.5100	59.795	121.93	...	
3	1491226380	2364.3101	42.3700	143.6400	130.0729	82.000	102.1400	85.4872	59.620	121.44	...	
4	1491226440	2364.8501	42.5378	143.6600	129.8800	82.035	102.0600	85.7001	59.620	121.60	...	

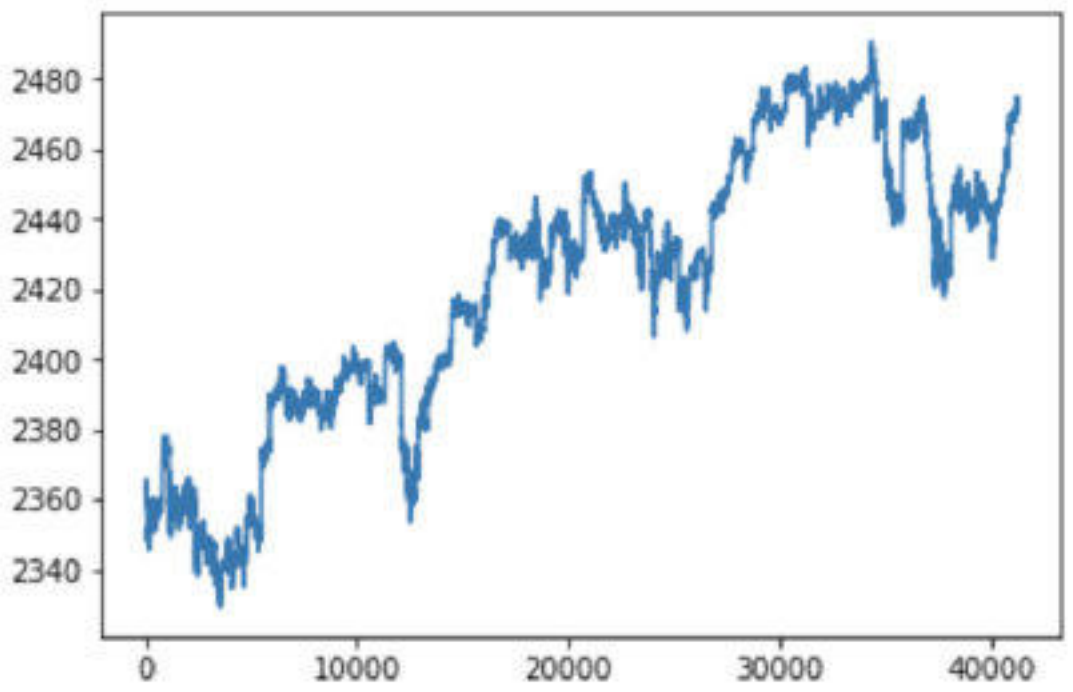
5 rows × 502 columns

绘制大盘趋势折线图

```
plt.plot(data['SP500'])
```
复制代码

结果如下：

```
[<matplotlib.lines.Line2D at 0x1a4f5974a8>]
```



> Tips：此次实验的调试环境为Jupyter Notebook，我们最好在开头导入matplotlib包的时候在加一行%matplotlib inline，这样就可以在

> Jupyter Notebook中内嵌绘图，并且省略掉plt.show()这一步。

数据整理

在这里，我们需要将数据集分为训练和测试数据。分配比为8:2，即训练数据包含总数据集的80%。当然在这之前我们需要先将DATE这个无关变量舍去。

```
data.drop('DATE', axis=1, inplace=True)
data_train = data.iloc[:int(data.shape[0] * 0.8), :]
data_test = data.iloc[int(data.shape[0] * 0.8):, :]
复制代码
```

我们来查看一下训练集和测试集的shape：

```
data_train.shape
```

```
(33012, 501)
```

```
data_test.shape
```

```
(8254, 501)
```

数据归一化

将属性缩放到一个指定的最大和最小值（通常是（-1，1））之间，这可以通过sklearn.preprocessing.MinMaxScaler类实现。

使用这种方法的目的包括：

1. 对于方差非常小的属性可以增强其稳定性。
2. 维持稀疏矩阵中为0的条目。

```
scaler = MinMaxScaler(feature_range=(-1, 1))
```

```
scaler.fit(data_train)
data_train = scaler.transform(data_train)
data_test = scaler.transform(data_test)
```
复制代码

预测

在此次预测过程中，我采用TensorFlow这个深度学习框架，它是目前领先的深度学习和神经网络计算框架。这一部分推荐**有基础的人**阅读，在这推荐我还在整理的TensorFlow系列，有兴趣的可以了解下。

这里我们需要解决的问题是：**使用当前时刻的500支个股股价，预测当前时刻的大盘指数**。即一个回归问题，**输入共500维特征，输出一维**，
即`[None, 500] => [None, 1]`

设置X与Y

```
X_train = data_train[:, 1:]
y_train = data_train[:, 0]
X_test = data_test[:, 1:]
y_test = data_test[:, 0]
```
复制代码

这里的x和y是已经分类好的数据集，只是用X和Y两个变量表示，可以理解为函数里面的X和Y。

设置超参数

```
input_dim = X_train.shape[1]
output_dim = 1
hidden_1 = 1024
hidden_2 = 512
hidden_3 = 256
hidden_4 = 128
batch_size = 256
epochs = 10
```
复制代码

这里我们设置了几个超参数，`input_dim`表示输入数据的维度，即

500。output_dim表示输出数据的维度，即1。在该模型中设置了4层隐藏层，第一层包含1024个神经元，略大于输入大小的两倍。 后续的隐藏层总是前一层的一半大小，即分别为512,256和128个神经元。 每个后续层的神经元数量的减少压缩了网络在先前层中识别的信息。 当然，其他网络架构和神经元配置也是可能的，只是由于本文只是一个入门的项目所以并未深究。

> 在机器学习中，**超参数**是在开始学习过程之前设置值的参数，而不是通过训练得到的参数数据。通常情况下，需要对超参数进行优化，给学习机选择一组最优超参数，以提高学习的性能和效果。

设置占位符（placeholder）

```
X = tf.placeholder(shape=[None, input_dim], dtype=tf.float32)
Y = tf.placeholder(shape=[None], dtype=tf.float32)
复制代码
```

为了适应我们的模型，我们需要两个占位符：X（神经网络的输入）和Y（神经网络的输出）。

设置神经网络

根据之前设置好的超参数进行神经网络的配置，其中**w**为权重，**b**为偏置值

```
# 第一层
W1 = tf.get_variable('W1', [input_dim, hidden_1], initializer=tf.contrib.la
b1 = tf.get_variable('b1', [hidden_1], initializer=tf.zeros_initializer())
# 第二层
W2 = tf.get_variable('W2', [hidden_1, hidden_2], initializer=tf.contrib.lay
b2 = tf.get_variable('b2', [hidden_2], initializer=tf.zeros_initializer())
# 第三层
W3 = tf.get_variable('W3', [hidden_2, hidden_3], initializer=tf.contrib.lay
b3 = tf.get_variable('b3', [hidden_3], initializer=tf.zeros_initializer())
# 第四层
W4 = tf.get_variable('W4', [hidden_3, hidden_4], initializer=tf.contrib.lay
b4 = tf.get_variable('b4', [hidden_4], initializer=tf.zeros_initializer())
# 输出层
W5 = tf.get_variable('W5', [hidden_4, output_dim], initializer=tf.contrib.l
```

```
b5 = tf.get_variable('b5', [output_dim], initializer=tf.zeros_initializer()
```
复制代码

了解输入层，隐藏层和输出层之间所需的变量尺寸非常重要。作为多层感知器(MLP，这里使用的网络类型)的一个经验法则，前一层的第二维是当前层中权重矩阵的第一维。这可能听起来很复杂，但实质上只是每个图层都将其输出作为输入传递到下一图层。偏差维度等于当前图层的权重矩阵的第二维度，其对应于该层中的神经元的数量。

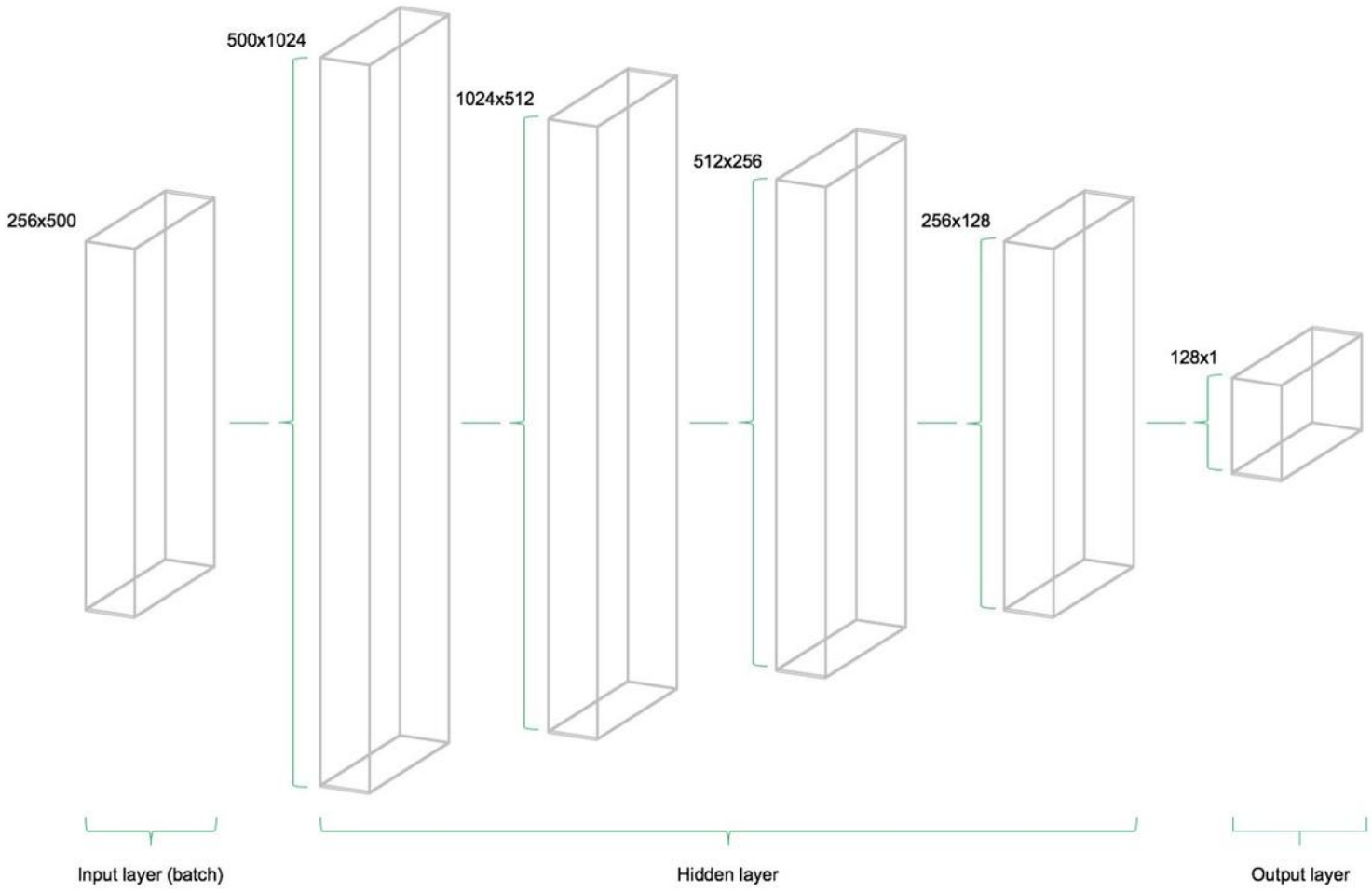
设置网络体系结构

在定义所需的权重和偏置变量后，需要指定网络拓扑结构和网络结构。因此，占位符(数据)和变量(权重和偏置)需要组合成一个连续矩阵乘法系统。

```
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))
```
复制代码

此外，网络的隐藏层需要被激活函数激活。激活函数是网络体系结构的重要组成部分，因为它们将非线性引入到系统中。这里采用最常见的ReLU激活函数。

下图说明了网络架构。 该模型由三个主要构建块组成。 输入层，隐藏层和输出层。 该体系结构称为前馈网络。 前馈表示该批数据仅从左向右流动。 其他网络架构，例如递归神经网络，也允许数据在网络中“向后”流动。



设置损失函数（loss function）和优化器（Optimizer）

```
loss = tf.reduce_mean(tf.squared_difference(out, Y))
optimizer = tf.train.AdamOptimizer().minimize(loss)
```
复制代码

这一部分没啥好说的，记住这么写就可以了，想了解的话可以去我的TensorFlow中了解下。

执行训练过程

在定义好神经网络的占位符，结构，损失函数函数和优化器之后，我们就可以开始对模型进行训练了。训练数据集分为n / batch_size批次，这些批次按顺序送入网络。 此时占位符X和Y开始起作用。 它们存储输入和目标数据，并将它们作为输入和目标呈现给网络。

数据X分批次流经网络，直到到达输出层。 在那里，TensorFlow将模型预测与当前批次中实际观察到的目标Y进行比较。 之后，TensorFlow进行优化步骤并更新与所选学习方案相对应的网络参数。 更新了权重和偏差后，对下一批进行采样，并重复该过程。 该过程将继续，直到所有批次都已呈现给网络。对所有数据进行一次全面扫描被称为一个

epoch（轮）。

一旦达到了最大数量的epochs或用户定义的另一个停止标准，网络的训练就会停止。

```
with tf.Session() as sess:
    # 初始化所有变量
    sess.run(tf.global_variables_initializer())

    for e in range(epochs):
        # 将数据打乱
        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})

            if i % 50 == 0:
                print('MSE Train:', sess.run(loss, feed_dict={X: X_train, Y
                print('MSE Test:', sess.run(loss, feed_dict={X: X_test, Y:
                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('Epoch ' + str(e) + ', Batch ' + str(i))
                plt.legend()
                plt.show()
```

复制代码

运行结果为：

```
MSE Train: 9.56518e-05
MSE Test: 0.0025863606
MSE Train: 6.0618047e-05
MSE Test: 0.0025002975
MSE Train: 0.00014856807
MSE Test: 0.0017371146
MSE Train: 0.00016200903
MSE Test: 0.0025396077
MSE Train: 0.00010259051
MSE Test: 0.0030134947
```

```
MSE Train: 7.979905e-05
MSE Test: 0.0023832247
MSE Train: 5.92488e-05
MSE Test: 0.0032762515
MSE Train: 8.747634e-05
MSE Test: 0.004848172
MSE Train: 8.5051965e-05
MSE Test: 0.0032768336
```
复制代码

最后测试集的loss在0.003左右，可以说是比较精确了。

可视化训练结果：

有很多方法可以进一步改善这一结果：增加隐藏层和改进神经元的设计，选择不同的初始化和激活方案，提前停止等等。 此外，不同类型的深度学习模型，例如循环神经网络，可以在此任务上实现更好的性能。 但是，这不是这篇介绍性文章的范围。有兴趣的小伙伴可以自行查找资料。

结论

正如开头所说，股票的价格会受到经济环境、政府政策、人为操作多种复杂因素的影响，真正想要预测股市走向单靠这篇文章里面所叙述的远远不够，本文旨在结合时下热点进行一次有关TensorFlow的技术推荐。

TensorFlow的发布是深度学习研究中的一个里程碑事件。作为一个学生，笔者也在积极的学习中，有兴趣学习的小伙伴可以在公众号后台（就是文末的那个公众号）回复「**TensorFlow视频**」获取一份质量较高的TensorFlow视频，也可以添加我的微信一起交流进步。

> 完整源码可在微信公众号：「01二进制」后台回复：「股市分析」获取

参考资料

- 使用TensorFlow进行股票价格预测的简单深度学习模型
- 深度有趣 | 10 股票价格预测

“万水千山都是情，给个关注行不行 👇”