

1 数据集获取

1.1 股票数据选取

选取股票为中兴通讯[000063]与科大讯飞[002230]，数据时间跨度为 2000 年 8 月 3 日至 2021 年 11 月 26 日，共 5000 行股票日线行情数据。

由于中兴通讯股票数据的波动与其他股相比较为不平稳，为了使预测结果的效果更加真实，防止过拟合，选取此股进行收盘价预测。

而科大讯飞股票数据的波动相较中兴通讯较为平稳，因此作为对比股进行收盘价预测实验。

1.2 数据获取方式

数据获取采用 tushare 平台提供的沪深股票日线行情接口获取数据。

TUSHARE

首页 平台介绍 数据接口 另类数据 资讯数据 数据工具 185 *** 829

沪深股票

基础数据

行情数据

日线行情

周线行情

月线行情

复权行情

复权因子

停牌复牌信息(停)

每日停牌复牌信息

每日指标

通用行情接口

个股资金流向

每日涨停涨停价格

每日涨停涨停统计

沪深港通资金流向

沪深股通十大成交股

港股通十大成交股

沪深股通持股明细

港股通每日成交统计

港股通每月成交统计

中央结算系统持股明细

备用行情

财务数据

其他数据

Search

Q

日线行情

接口: daily, 可以通过数据工具调试和查看数据。
数据说明: 交易日每天15点~16点之间。本接口是未复权行情, 停牌期间不提供数据。
调用说明: 基础积分每分钟最多调用500次, 每次5000条数据, 相当于23年历史, 用户获得超过5000积分正常调用无频次限制。
描述: 获取股票行情数据, 或通过通用行情接口获取数据, 包含了前后复权数据。

输入参数

名称	类型	必选	描述
ts_code	str	N	股票代码 (支持多个股票同时调用, 逗号分隔)
trade_date	str	N	交易日期 (YYYYMMDD)
start_date	str	N	开始日期(YYYYMMDD)
end_date	str	N	结束日期(YYYYMMDD)

注: 日期都填YYYYMMDD格式, 比如20181010

输出参数

名称	类型	描述
ts_code	str	股票代码
trade_date	str	交易日期
open	float	开盘价
high	float	最高价

其中，获取的股票日线行情格式共 11 个参数，详细描述为：

名称	类型	描述
ts_code	str	股票代码
trade_date	str	交易日期
open	float	开盘价
high	float	最高价

名称	类型	描述
low	float	最低价
close	float	收盘价
pre_close	float	昨收价
change	float	涨跌额
pct_chg	float	涨跌幅 （未复权）
vol	float	成交量 （手）
amount	float	成交额 （千元）

Python 调用接口即可获取相应股票数据。

```
In [95]: import tushare as ts
pro = ts.pro_api('1288ed74a1b0b450b1b67f57773fce56905763e070754173f8bf1974')

#查询当前所有正常上市交易的股票列表
df = pro.daily(ts_code='000063.SZ')
df

Out[95]:
```

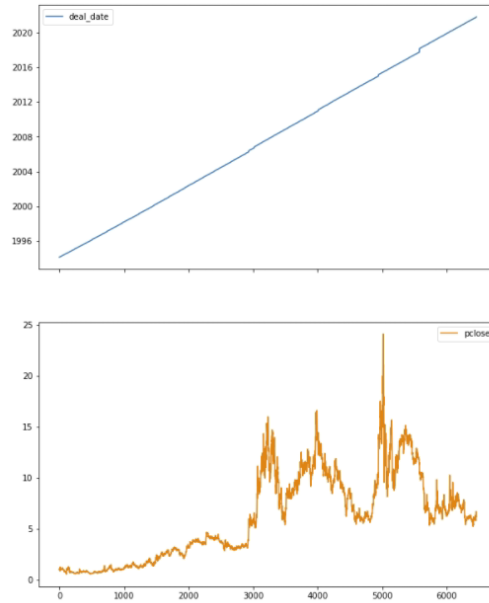
	ts_code	trade_date	open	high	low	close	pre_close	change	pct_chg	vol	amount
0	000063.SZ	20211126	32.06	32.18	31.12	31.22	32.16	-0.94	-2.9229	480646.65	1.509495e+06
1	000063.SZ	20211125	32.64	32.77	32.15	32.16	32.65	-0.49	-1.5008	308232.60	9.990552e+05
2	000063.SZ	20211124	32.68	32.80	32.20	32.65	32.42	0.23	0.7094	389904.82	1.270291e+06
3	000063.SZ	20211123	32.50	32.89	32.37	32.42	32.40	0.02	0.0617	294005.86	9.577698e+05
4	000063.SZ	20211122	31.99	32.83	31.91	32.40	31.99	0.41	1.2817	395291.03	1.283238e+06
...
4995	000063.SZ	20000829	35.60	36.05	35.50	35.84	35.78	0.06	0.1700	8817.00	3.166278e+04
4996	000063.SZ	20000828	36.00	36.10	35.58	35.78	36.15	-0.37	-1.0200	16039.00	5.747439e+04
4997	000063.SZ	20000825	35.30	36.30	35.17	36.15	35.18	0.97	2.7600	14256.00	5.090268e+04
4998	000063.SZ	20000824	35.30	35.54	34.98	35.18	35.28	-0.10	-0.2800	11754.00	4.144652e+04
4999	000063.SZ	20000823	35.60	36.00	34.00	35.28	35.87	-0.59	-1.6400	30526.00	1.064569e+05

5000 rows x 11 columns

1.3 数据特征

由于此次预测的目标是股票收盘价，因此将收盘价可视化，观察曲线特点。

```
In [36]: # 获取价格走势图形
tm.plot(figsize=(10,12), subplots=True)
Out[36]: array([<AxesSubplot:~>, <AxesSubplot:~>], dtype=object)
```



在观察股票收盘价走势过程中，发现数据具有较强的时序数据特征，数据并非孤立、割裂的，而是可以以拐点为划分依据，将数据变化描述为在变化的时间区段上，较为稳定的趋势变化。同时这种时间上的联系并非连续的、全局的，而是局部的数据。因此考虑选用 LSTM 长短时记忆网络进行股票收盘价预测。

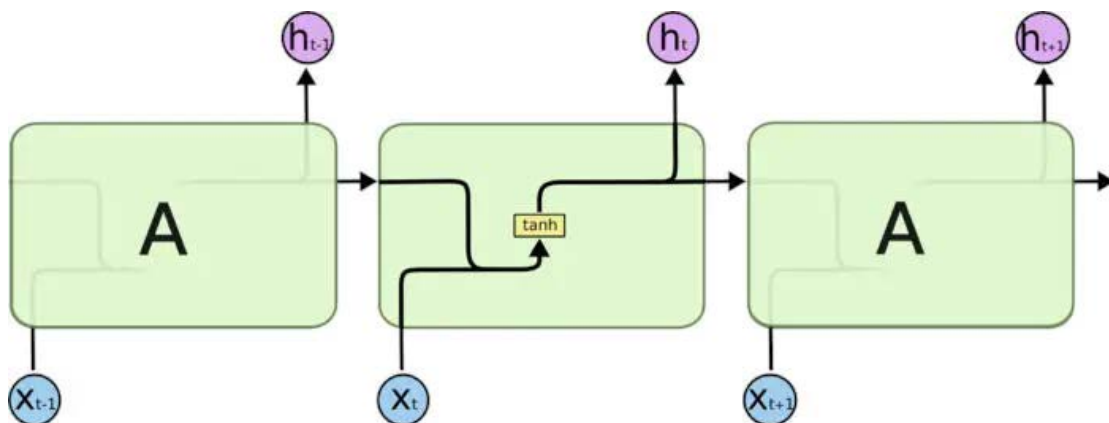
2 LSTM 长短时记忆网络

2.1 LSTM 原理

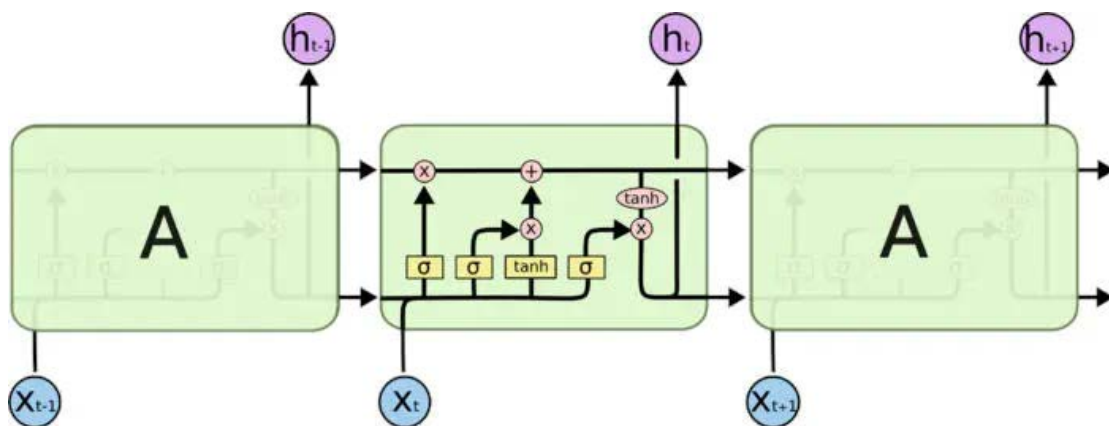
LSTM 长短时记忆网络，是一种改进之后的循环神经网络，可以解决 RNN 无法处理长距离的依赖的问题。

2.1.1 LSTM 网络

所有的 RNN 都具有一种重复神经网络模块的链式形式。在标准 RNN 中，这个重复的结构模块只有一个非常简单的结构，例如一个 tanh 层。



LSTM 同样是这样的结构，但是重复的模块拥有一个不同的结构。不同于单一神经网络层，这里是有四个，以一种非常特殊的方式进行交互。



2.2 LSTM 核心思想

LSTM 的关键在于 cell 的状态和穿过 cell 的门。

cell 状态类似于传送带。直接在整个链上运行，只有一些少量的线性交互。信息在上面流传并保持不变会很容易。

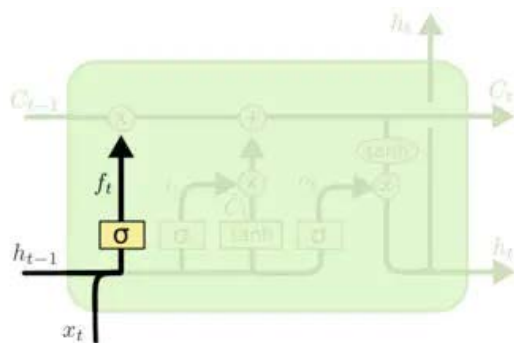
门可以实现选择性地让信息通过，主要是通过一个 sigmoid 的神经层和一个逐点相乘的操作实现的。sigmoid 层输出的每个元素都是一个在 0 和 1 之间的实数，表示让对应信息通过的权重。

LSTM 通过三个这样的门结构来实现信息的保护和控制。这三个门分别输入门、遗忘门和输出门。

2.3 LSTM 的三个门

2.3.1 遗忘门

在 LSTM 中的第一步是决定应从 cell 状态中丢弃什么信息。这个操作通过遗忘门完成，该门会读取 h_{t-1} 和 x_t ，输出一个在 0 到 1 之间的数值给每个在 cell 状态 C_{t-1} 中的数字。1 表示完全保留，而 0 表示完全舍弃。

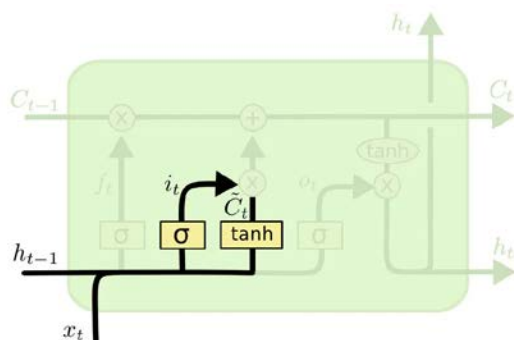


$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$$

其中， h_{t-1} 表示上一时刻隐藏层的输出。 x_t 表示当前 cell 的输入。 σ 表示 sigmoid 函数。

2.3.2 输入门

下一步是决定让多少新的信息加入到 cell 状态中。实现操作包括两个步骤：首先，一个 sigmoid 层决定哪些信息需要更新；一个 tanh 层生成一个向量，也就是备选的用来更新的内容。其次，联合上述两部分起来，对 cell 的状态进行一个更新。



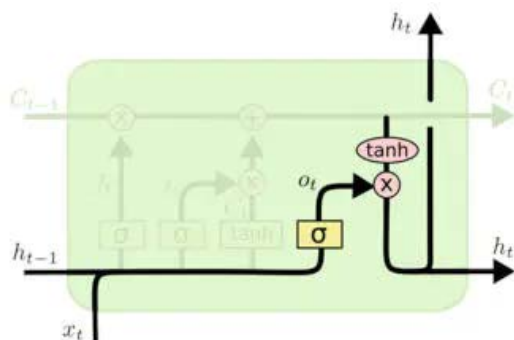
$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

https://blog.csdn.net/jerry_xu

2.3.3 输出门

最终，需要确定输出什么值。这个输出将会基于 cell 状态，但是也是一个过滤后的版本。首先，运行一个 sigmoid 层来确定 cell 状态的哪个部分将输出出去。接着，把细胞状态通过 tanh 进行处理，得到一个在-1 到 1 之间的值，并将它和 sigmoid 门的输出相乘，最终我们仅仅会输出我们确定输出的那部分。



$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$$

$$h_t = o_t * \tanh(C_t)$$

3 数据预处理

3.1 数据存储

Python 创建 MySQL 股票表，并将 tushare 接口查询到的日线行情数据存入表中，以便后续操作查询。

```
import pandas as pd

import numpy as np
import sqlalchemy
import tushare as ts

# 连接MySQL
engine =
sqlalchemy.create_engine('mysql+pymysql://root:root@localhost:3306/st
ock')

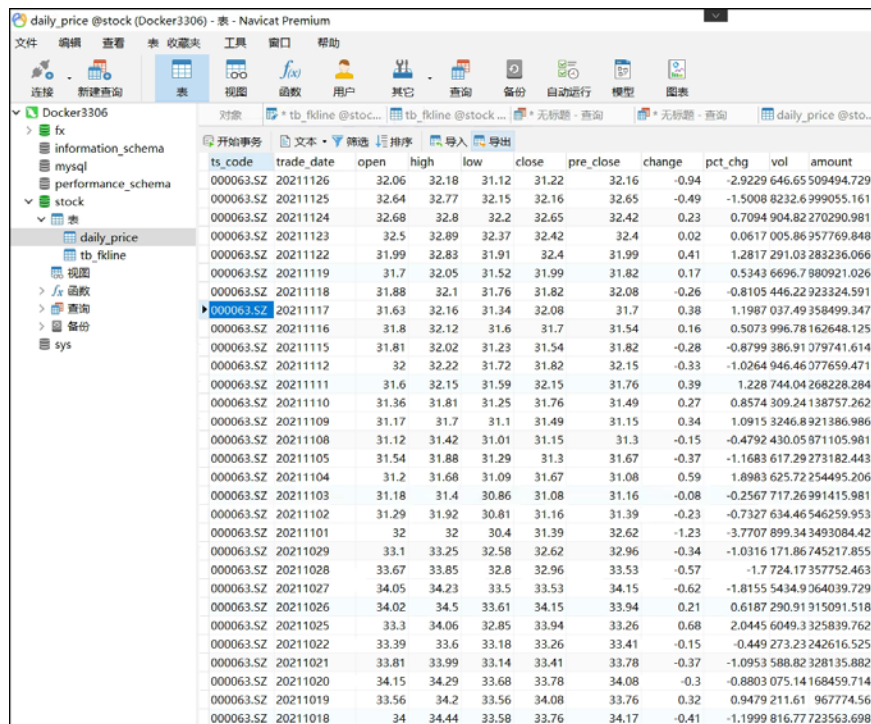
# 创建股票表
sql = '''
CREATE TABLE `daily_price` (
  `id` int NOT NULL primary key AUTO_INCREMENT,
  `ts_code` char(12) NOT NULL ,
  `trade_date` char(12) NOT NULL,
  `open_price` float(7,2) NULL,
  `high_price` float(7,2) NULL,
  `low_price` float(7,2) NULL,
  `close_price` float(7,2) NULL,
  `adj_close_price` float(7,4) NULL,
  `change` float(5,2) NULL,
  `pct_chg` float(5,2) NOT NULL,
  `vol` float(10,2) NOT NULL,
  `amount` float(10,2) NOT NULL
) ENGINE=InnoDB AUTO_INCREMENT=1 DEFAULT CHARSET=utf8;
'''

tm = pd.read_sql(sql,engine)

# 获取股票数据
pro =
ts.pro_api('1288ed74a1b0b450b1b67f57773fce56905763e070754173f8bf1974'
)
```

```
# 查询当前所有正常上市交易的股票列表
df = pro.daily(ts_code='000063.SZ') #, start_date='19990829',
end_date='20211128'
print(df)
# 将股票数据存入 MySQL
df.to_sql(name='daily_price',con=engine,if_exists='replace',index=False)
```

存储至 MySQL 的数据如下：



ts_code	trade_date	open	high	low	close	pre_close	change	pct_chg	vol	amount
000063.SZ	20211126	32.06	32.18	31.12	31.22	32.16	-0.94	-2.9229	646.65	509494.729
000063.SZ	20211125	32.64	32.77	32.15	32.16	32.65	-0.49	-1.5008	8232.6	999055.161
000063.SZ	20211124	32.68	32.8	32.2	32.65	32.42	0.23	0.7094	904.82	270290.981
000063.SZ	20211123	32.5	32.89	32.37	32.42	32.4	0.02	0.0617	005.86	957769.848
000063.SZ	20211122	31.99	32.83	31.91	32.4	31.99	0.41	1.2817	291.03	283236.066
000063.SZ	20211119	31.7	32.05	31.52	31.99	31.82	0.17	0.5343	6696.7	980921.026
000063.SZ	20211118	31.88	32.1	31.76	31.82	32.08	-0.26	-0.8105	446.22	923324.591
000063.SZ	20211117	31.63	32.16	31.34	32.08	31.7	0.38	1.1987	037.49	358499.347
000063.SZ	20211116	31.8	32.12	31.6	31.7	31.54	0.16	0.5073	996.78	162648.125
000063.SZ	20211115	31.81	32.02	31.23	31.54	31.82	-0.28	-0.8799	386.91	079741.614
000063.SZ	20211112	32	32.22	31.72	31.82	32.15	-0.33	-1.0264	946.46	077659.471
000063.SZ	20211111	31.6	32.15	31.59	32.15	31.76	0.39	1.228	744.04	268228.284
000063.SZ	20211110	31.36	31.81	31.25	31.76	31.49	0.27	0.8574	309.24	138757.262
000063.SZ	20211109	31.17	31.7	31.1	31.49	31.15	0.34	1.0915	3246.8	921386.986
000063.SZ	20211108	31.12	31.42	31.01	31.15	31.3	-0.15	-0.4792	430.05	871105.981
000063.SZ	20211105	31.54	31.88	31.29	31.3	31.67	-0.37	-1.1683	617.29	273182.443
000063.SZ	20211104	31.2	31.68	31.09	31.67	31.08	0.59	1.8983	625.72	254495.206
000063.SZ	20211103	31.18	31.4	30.86	31.08	31.16	-0.08	-0.2567	717.26	991415.981
000063.SZ	20211102	31.29	31.92	30.81	31.16	31.39	-0.23	-0.7327	634.46	546259.953
000063.SZ	20211101	32	32	30.4	31.39	32.62	-1.23	-3.7707	899.34	3493084.42
000063.SZ	20211029	33.1	33.25	32.58	32.62	32.96	-0.34	-1.0316	171.86	745217.855
000063.SZ	20211028	33.67	33.85	32.8	32.96	33.53	-0.57	-1.7724	17.35	757752.463
000063.SZ	20211027	34.05	34.23	33.5	33.53	34.15	-0.62	-1.8155	5434.9	064039.729
000063.SZ	20211026	34.02	34.5	33.61	34.15	33.94	0.21	0.6187	290.91	915091.518
000063.SZ	20211025	33.3	34.06	32.85	33.94	33.26	0.68	2.0445	6049.3	325839.762
000063.SZ	20211022	33.39	33.6	33.18	33.26	33.41	-0.15	-0.449	273.23	242616.525
000063.SZ	20211021	33.81	33.99	33.14	33.41	33.78	-0.37	-1.0953	588.82	328135.882
000063.SZ	20211020	34.15	34.29	33.68	33.78	34.08	-0.3	-0.8803	075.14	168459.714
000063.SZ	20211019	33.56	34.2	33.56	34.08	33.76	0.32	0.9479	211.61	967774.56
000063.SZ	20211018	34	34.44	33.58	33.76	34.17	-0.41	-1.1999	816.77	723563.698

3.2 数据集划分

数据集划分为 80% 的训练集和 20% 的测试集。

```
# 构造 X 和 Y

# 根据前 n 天的数据，预测未来一天的收盘价(close)， 例如：根据 1 月 1 日、1 月 2 日、1
# 月 3 日、1 月 4 日、1 月 5 日的数据（每一天的数据包含 8 个特征），预测 1 月 6 日的收盘
# 价。
sequence = sequence_length
X = []
Y = []
for i in range(df.shape[0] - sequence):
    X.append(np.array(df.iloc[i:(i + sequence), ].values,
dtype=np.float32))
    Y.append(np.array(df.iloc[(i + sequence), 0], dtype=np.float32))

# 构建 batch
total_len = len(Y)
```

```

print('total_len',total_len)

# 分训练集和测试集
trainx, trainy = X[:int(0.99 * total_len)], Y[:int(0.99 * total_len)]
testx, testy = X[int(0.99 * total_len):], Y[int(0.99 * total_len):]

# print('trainx,trainy',trainx,trainy)
# print('testx,testy',testx,testy)

train_loader = DataLoader(dataset=Mydataset(trainx, trainy,
transform=transforms.ToTensor()), batch_size=batchSize,
                           shuffle=True)
test_loader = DataLoader(dataset=Mydataset(testx, testy),
batch_size=batchSize, shuffle=True)
return close_max,close_min,train_loader,test_loader

```

3.3 加载数据集

Python 通过 MySQL 查询数据集，由于目标是预测股票收盘价，因此在加载数据集时剔除无用的特征，选取开盘价、最高价、最低价、收盘价、涨跌额、涨跌幅、成交量、成交额，共 8 个特征进行股票预测。

```

# 数据预处理，去除 id、股票代码、前一天的收盘价、交易日期等对训练无用的无效数据

# stock_data = read_csv(corpusFile)
engine =
sqlalchemy.create_engine('mysql+pymysql://root:root@localhost:3306/st
ock')
sql = '''
SELECT * FROM `daily_price`
WHERE ts_code IN ('000063.SZ')
ORDER BY trade_date
'''

stock_data = pd.read_sql(sql, engine)
stock_data.drop('ts_code', axis=1, inplace=True) # 删除第二列'股票代码'
# stock_data.drop('id', axis=1, inplace=True) # 删除第一列'id'
stock_data.drop('pre_close', axis=1, inplace=True) # 删除
列'pre_close'
stock_data.drop('trade_date', axis=1, inplace=True) # 删除
列'trade_date'

# close, open, high, low, change, pct_chg, vol, amount

```



```
close_max = stock_data['close'].max() #收盘价的最大值
close_min = stock_data['close'].min() #收盘价的最小值
```

3.4 数据标准化

由于预测收盘价格时需要考虑多个输入的影响，而不同的因素性质不同，具有不同的量纲和数量级（如交易量和价格之间单位和量级不同）。考虑到不同的特征之间的差异，如果直接用原始指标值进行分析，就会突出数值较高的指标在综合分析中的作用，相对削弱数值水平较低指标的作用。因此，为了保证结果的可靠性，需要对原始指标数据进行标准化处理。

方法基于原始数据的均值和标准差进行数据的标准化，使用正则表达式进行数据标准化处理，将数据集缩放到 0 到 1 之间的数值。

```
df = stock_data.apply(lambda x: (x - min(x)) / (max(x) - min(x))) #
min-max 标准化
```

4 模型构建

4.1 模型参数

参数 hidden_size 表示隐藏层的维度；参数 input_size 表示输入特征的维度；参数 num_layer 表示 LSTM 层数；参数 output_size 表示输出特征的维度，dropout 用于添加防止过拟合的 dropout 层；参数 batch_size 表示每个 batch 中训练样本的数量。

```
import torch.nn as nn

class lstm(nn.Module):

    #改变堆叠层数
    def __init__(self, input_size=8, hidden_size=32, num_layers=2 ,
output_size=1 , dropout=0, batch_first=True):
        super(lstm, self).__init__()
        # lstm的输入 #batch,seq_len, input_size
        self.hidden_size = hidden_size
        self.input_size = input_size
        self.num_layers = num_layers
        self.output_size = output_size
        self.dropout = dropout
        self.batch_first = batch_first
```

```

        self.rnn = nn.LSTM(input_size=self.input_size,
hidden_size=self.hidden_size, num_layers=self.num_layers,
batch_first=self.batch_first, dropout=self.dropout )
        self.linear = nn.Linear(self.hidden_size, self.output_size)

    def forward(self, x):
        out, (hidden, cell) = self.rnn(x) # x.shape : batch, seq_len,
hidden_size , hn.shape and cn.shape : num_layes * direction_numbers,
batch, hidden_size
        # a, b, c = hidden.shape
        # out = self.linear(hidden.reshape(a * b, c))
        out = self.linear(hidden)
        return out

```

4.2 模型训练

4.2.1 定义常用参数

将常用参数的设置单独创建为一个.py 文件，便于修改参数。可直接运行进行参数的赋值。

训练轮数选取 100 轮；LSTM 层数选取 2；由于用到的特征个数为 8，因此维度取 8；隐藏层的维度取 32；优化器 Adam 的学习率选取建议为 0.0001；sequence 的长度为用 n 天来预测收盘价，选取前十天的数据来预测下一天的收盘价。

```

import argparse

import torch

parser = argparse.ArgumentParser()

parser.add_argument('--corpusFile',
default='data/000001SH_index.csv')

# TODO 常改动参数
parser.add_argument('--gpu', default=0, type=int) # gpu 卡号
parser.add_argument('--epochs', default=100, type=int) # 训练轮数
parser.add_argument('--layers', default=2, type=int) # LSTM 层数
parser.add_argument('--input_size', default=8, type=int) #输入特征的维
度
parser.add_argument('--hidden_size', default=32, type=int) #隐藏层的维
度
parser.add_argument('--lr', default=0.0001, type=float) #learning
rate 学习率

```

```

parser.add_argument('--sequence_length', default=10, type=int) #
sequence 的长度，默认是用前五天的数据来预测下一天的收盘价
parser.add_argument('--batch_size', default=64, type=int)
parser.add_argument('--useGPU', default=True, type=bool) #是否使用 GPU
parser.add_argument('--batch_first', default=True, type=bool) #是否将
batch_size 放在第一维
parser.add_argument('--dropout', default=0.1, type=float)
parser.add_argument('--save_file', default='model/stock_000063.pkl')
# 模型保存位置

args = parser.parse_args()

device = torch.device(f"cuda:{args.gpu}" if torch.cuda.is_available()
and args.useGPU else "cpu")
args.device = device

```

4.2.2 模型参数

模型采用 LSTM，并选取 MSE 均方误差作为损失函数，均方误差是回归损失函数中最常用的误差，它是预测值 $f(x)$ 与目标值 y 之间差值平方和的均值，其优势是函数曲线光滑、连续，处处可导，便于使用梯度下降算法，是一种常用的损失函数。而且，随着误差的减小，梯度也在减小，这有利于收敛，即使使用固定的学习速率，也能较快的收敛到最小值。

优化器选取最常用的 Adam 优化器，其有着收敛速度快、调参容易等优点。学习率通常建议为 0.001，因此训练时采用的学习率也选取为 0.001。

```

def train():

    model = lstm(input_size=args.input_size,
hidden_size=args.hidden_size, num_layers=args.layers , output_size=1,
dropout=args.dropout, batch_first=args.batch_first )
    model.to(args.device)
    criterion = nn.MSELoss() # 定义损失函数

    optimizer = torch.optim.Adam(model.parameters(), lr=args.lr) #
Adam 梯度下降 学习率=0.001
    # optimizer = torch.optim.SGD(model.parameters(), lr=args.lr) #
Adam 梯度下降 学习率=0.001

    close_max, close_min, train_loader, test_loader =
getData(args.corpusFile,args.sequence_length,args.batch_size )

```

4.2.3 模型训练

将划分好的数据集按照训练轮数进行遍历训练，这是一个损失值逐渐收敛的过程，训练结束后将模型输出至结果文件夹，并绘制损失随着训练轮数增加而下降的曲线图。

```
# 训练轮数

# args.epochs = 200
print(args.epochs)

for i in range(args.epochs):
    total_loss = 0
    for idx, (data, label) in enumerate(train_loader):
        if args.useGPU:
            data1 = data.squeeze(1).cuda()
            pred = model(Variable(data1).cuda())
            # print(pred.shape)
            pred = pred[1, :, :]
            label = label.unsqueeze(1).cuda()
            # print(label.shape)
        else:
            data1 = data.squeeze(1)
            pred = model(Variable(data1))
            pred = pred[1, :, :]
            label = label.unsqueeze(1)
        loss = criterion(pred, label)
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
        total_loss += loss.item()

    # 损失
    iter.append(i)
    lossTotal.append(total_loss)

    print(total_loss)
    if i % 10 == 0:
        # torch.save(model, args.save_file)
        torch.save({'state_dict': model.state_dict()}, args.save_file)
        print('第%d epoch, 保存模型' % i)
# torch.save(model, args.save_file)
torch.save({'state_dict': model.state_dict()}, args.save_file)
#画图
plot(iter, lossTotal)
```

5 模型预测

5.1 加载模型并预测

首先将训练后的模型通过结果文件夹读出，再将划分好的数据集进行模型预测，选取前十天的数据来预测下一天的收盘价，并将预测结果存入 list 中，便于模型评估及结果输出。

```
def eval(result_predict, result_real):  
  
    # model = torch.load(args.save_file)  
    model = lstm(input_size=args.input_size,  
hidden_size=args.hidden_size, num_layers=args.layers , output_size=1)  
    model.to(args.device)  
    checkpoint = torch.load(args.save_file)  
    model.load_state_dict(checkpoint['state_dict'])  
    preds = []  
    labels = []  
    close_max, close_min, train_loader, test_loader =  
getData(args.corpusFile, args.sequence_length, args.batch_size)  
    for idx, (x, label) in enumerate(test_loader):  
        if args.useGPU:  
            x = x.squeeze(1).cuda() # batch_size, seq_len, input_size  
        else:  
            x = x.squeeze(1)  
        pred = model(x)  
        list = pred.data.squeeze(1).tolist()  
        preds.extend(list[-1])  
        labels.extend(label.tolist())
```

5.2 模型结果输出

将预测值与真实值的数值输出，并绘制折线图。以预测值与真实值的偏差程度计算模型的评估结果，偏差程度越小表示模型的预测值越准确。

```
for i in range(len(preds)):  
  
    print('预测值是%.2f,真实值是%.2f' % (  
        preds[i][0] * (close_max - close_min) + close_min, labels[i] *  
(close_max - close_min) + close_min))  
  
    result_predict.append(preds[i][0] * (close_max - close_min) +  
close_min)  
    result_real.append(labels[i] * (close_max - close_min) +  
close_min)
```

```

# 画图
def plot():
    # 声明预测列表
    result_predict = []
    result_real = []
    eval(result_predict, result_real)

    result_predict = np.array(result_predict)
    result_real = np.array(result_real)

    # 计算模型评估结果
    acc = np.average(np.abs(result_predict -
result_real[:len(result_predict)])) /
result_real[:len(result_predict)]) # 偏差程度
    print('真实数据和预测数据的偏差值为: ', acc)

    # 画图
    plt.plot(result_real, label='real')
    plt.plot(result_predict, label='pred')
    plt.legend()
    plt.rcParams['font.sans-serif'] = ['SimHei']
    plt.rcParams['axes.unicode_minus'] = False
    plt.xlabel('Time/天数')
    plt.ylabel('收盘价')
    plt.show()

```

6 模型评估与结果展示

6.1 结果展示

通过 Python 绘图程序, 分别将结果绘制成了训练损失收敛图与收盘价预测图, 并将结果输出至控制台。

6.1.1 训练损失收敛图与损失输出

选取中兴通讯股票与科大讯飞股票进行了对比实验, 以此综合评估模型的准确度和稳定性。

实验采取十分常用的均方误差损失, 也可以理解为是最小二乘法, 一般在回归问题中比较常见, 基本原理为: 最优拟合直线是使各点到回归直线的距离和最

小的直线，即平方和最小。同时在实际应用中，均方误差也经常被用为衡量模型的标准。

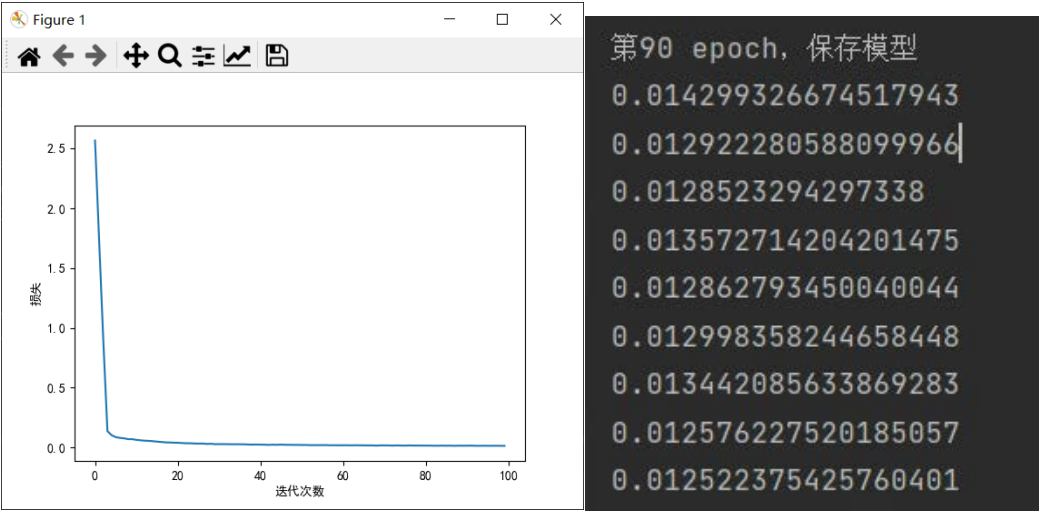


图 6.1 中兴通讯损失图

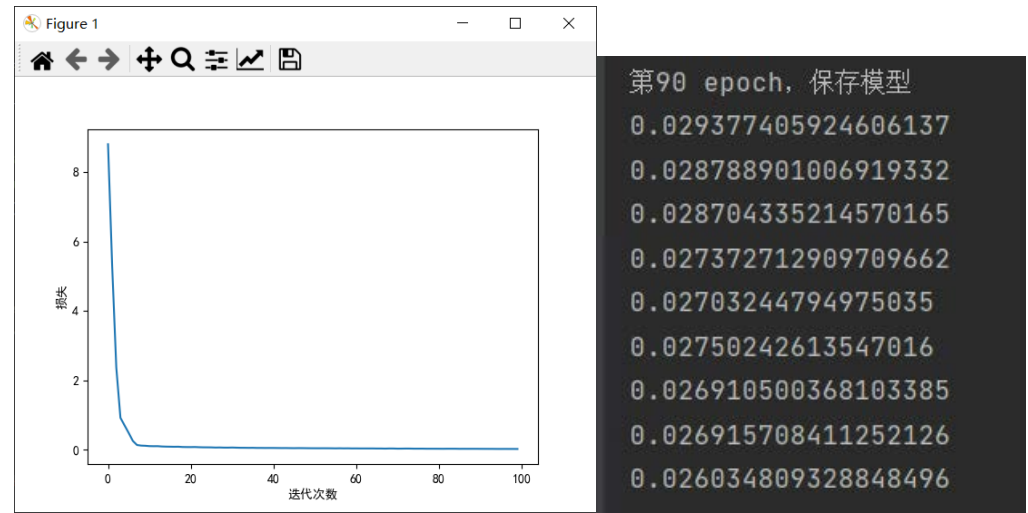
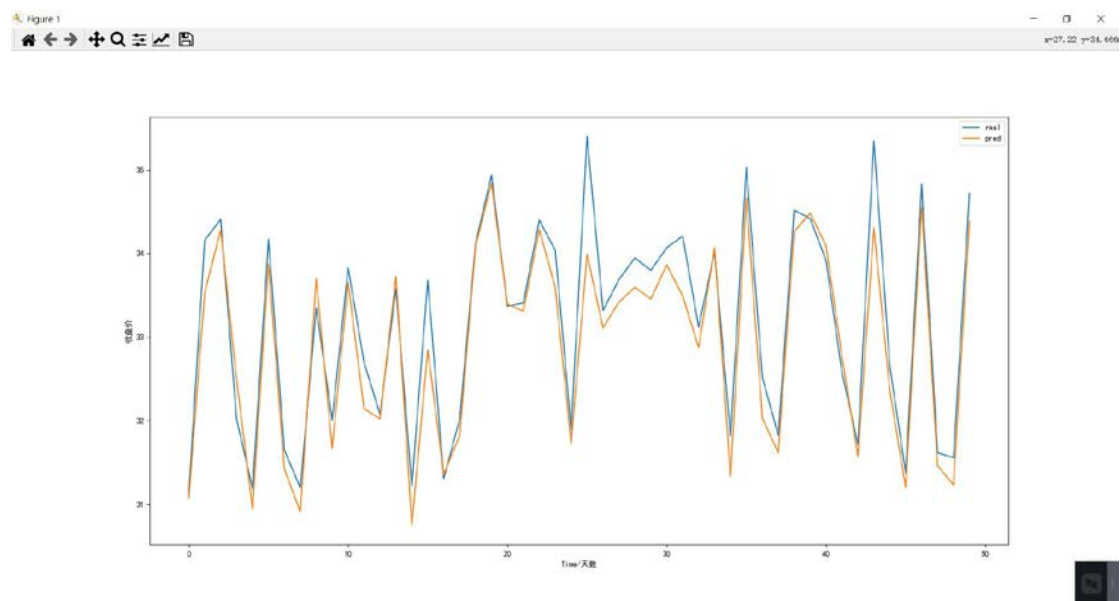


图 6.2 科大讯飞损失图

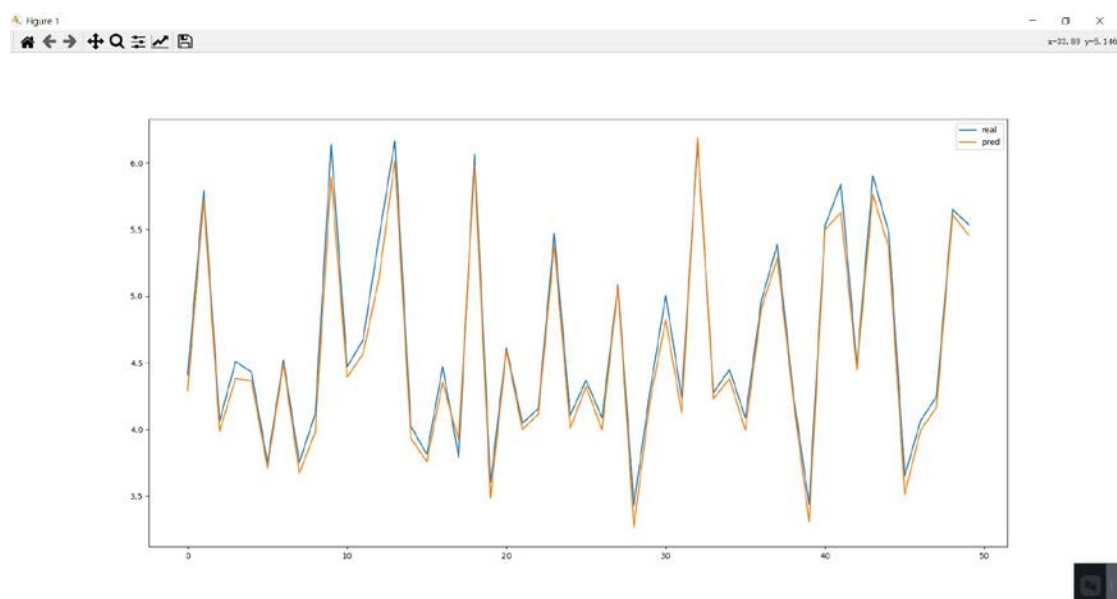
6.1.2 收盘价预测图与预测值输出

折线图中的蓝色折线即为收盘价真实值，而橙色折线即为收盘价预测值。



预测值是32.86, 真实值是32.58
预测值是30.87, 真实值是31.22
预测值是33.65, 真实值是34.16
预测值是33.22, 真实值是33.32
预测值是31.73, 真实值是31.83
预测值是32.47, 真实值是32.66
预测值是34.82, 真实值是35.03
预测值是34.41, 真实值是34.52
预测值是31.78, 真实值是32.01
预测值是32.98, 真实值是33.12
预测值是33.62, 真实值是34.21
预测值是33.59, 真实值是33.80
预测值是32.26, 真实值是32.70
预测值是31.33, 真实值是31.38
预测值是33.51, 真实值是33.37
预测值是34.99, 真实值是34.94
预测值是33.73, 真实值是33.95
预测值是34.43, 真实值是35.35

图 6.3 中兴通讯收盘价预测图



预测值是56.25,真实值是55.75
 预测值是57.41,真实值是57.67
 预测值是59.01,真实值是58.89
 预测值是53.35,真实值是53.44
 预测值是55.12,真实值是54.73
 预测值是57.72,真实值是58.55
 预测值是56.15,真实值是56.33
 预测值是57.30,真实值是58.18
 预测值是54.36,真实值是54.47
 预测值是58.33,真实值是58.31
 预测值是57.72,真实值是57.57
 预测值是54.26,真实值是53.43
 预测值是53.97,真实值是53.40
 预测值是58.71,真实值是59.16
 预测值是56.96,真实值是56.96

图 6.4 科大讯飞收盘价预测图

6.2 模型评估

6.1.1 损失变化评估

观察损失变化图可得知，在 LSTM 模型下可以实现股票收盘价预测值较为准确和稳定。对 LSTM 模型进行参数调整，发现两支股票的迭代次数在 20 次左右，损失值变化明显放缓，在迭代次数超过 100 次后，损失值无明显趋势变化，说明构建网络为轻量型网络。

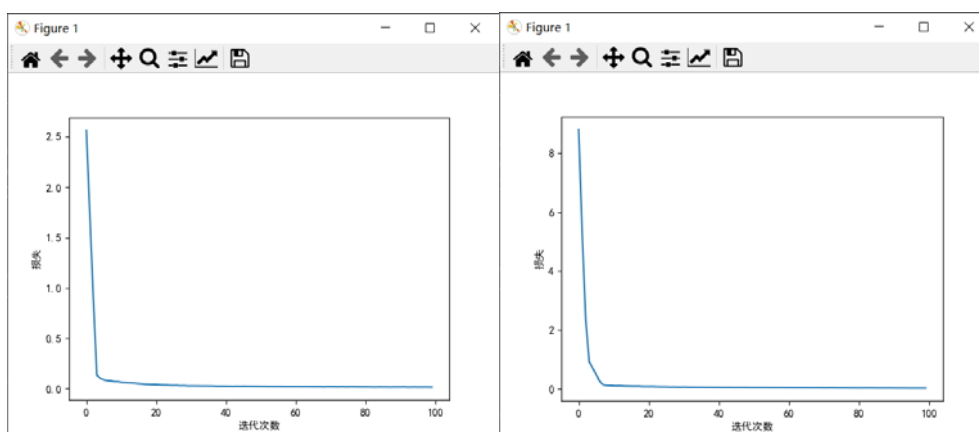


图 6.5 中兴通讯与科大讯飞损失图对比

6.1.2 模型准确度评估

经过真实数据与预测数据的偏差度计算可得，偏差值较小，因此预测数据较为准确。由于中兴通讯股价波动较大，而科大讯飞股价较为平稳，因此模型对科大讯飞的预测准确度较高一些。

真实数据和预测数据的偏差值为： 0.007010914725360335

图 6.6 科大讯飞股票预测偏差度

真实数据和预测数据的偏差值为： 0.007608630132572314

图 6.7 中兴通讯股票预测偏差度

对比中兴通讯和科大讯飞的收盘价及其预测值可得，在 LSTM 模型下对于数据密集型或趋势较为明显的区域可以实现股票收盘价预测值较为准确和稳定。对于波动较大的波峰和波谷区域以及转折点预测效果有限，在实验过程中，发现通过调整时间步长，偏差值大小可以改善其表现情况，而增加神经元个数和迭代次数所带来的改变有限。

7 总结及编程心路历程

7.1 总结

LSTM 是 RNN 的一个优秀的变种模型，继承了大部分 RNN 模型的特性由于对于过往数据都会存到“记忆神经”，也就是遗忘门、输入门、输出门中。也就不是只简单看一个平均，所以预测可能会激进偏颇一点，但是对于原始数据波动比较大时，效果更好。

LSTM 非常适合用于处理与时间序列高度相关的问题，股票数据与 LSTM 模型很契合，相比较其他如线性回归、ARIMA 等模型，能够较好地预测股票走势。

本次股票收盘价预测仅仅基于股票的日线行情，但股票价格的形成机制相当复杂。各种因素的综合运用和个体因素的特殊行为，包括政治、经济、市场因素以及技术和投资者行为，都会导致股价变化。因此，股票价格是不断变化的，在涉及机器学习的方法中，若想得到更准确的预测，还需要结合该公司新闻信息等舆论因素进行综合分析。

7.2 心路历程

本次作业通过观察中兴通讯和科大讯飞股票的收盘价的特点，选取了 LSTM 模型，进行两支股票的对比实验。通过 LSTM 模型实现了中兴通讯的股票收盘价的预测，对预测结果绘制了折线图，并通过与真实值的偏差值评估了模型的准确度。

虽然对于模型预测股票的准确度尚且满意，但在实践 LSTM 中也发现了模型存在的不足，例如：LSTM 的重要参数通常由经验决定、主观性强，难以确定最优值，这样会导致模型的拟合能力降低等问题。接下来的优化工作可以考虑结合粒子群算法，优化 LSTM 网络中的关键参数，减少人为因素影响。

8 GitHub 地址

本次程序的 github 地址为：

<https://github.com/StrontiaForWork/machineLearningCourse.git>

`git@github.com:StrontiaForWork/machineLearningCourse.git`