基于资金流向分析的门控循环神经网络模型(GRU)的股票价格预测研究

数据 201　肖楚恒

指导教师:戴宏亮

摘要： 股票价格预测长久以来都是金融领域的热门研究课题，随着神经网络模型的广泛应用，出现了许多利用神经网络对股票价格进行预测的方法。本文使用基于门控制单元(GRU)的循环神经网络(RNN)模型与股票资金流向分析，尝试利用资金流向从动力学的角度对价格趋势进行学习，并进行预测。本文使用的数据为从 A 股随机抽取的一支股票最近 1000 个交易日的收盘价、涨跌幅与资金流向。本文使用 MSE 作为模型评估指标，模型在训练后 MSE 能达到 0.014，模型预测与真实价格趋势上几乎一致，能够认为此模型对短期价格趋势预测较为准确。

关键词： 门控循环神经网络；价格预测；股票

ABSTRACT: Stock price prediction has long been a popular research topic in the financial field, and with the widespread use of neural network models, many methods for predicting stock prices using neural networks have emerged. This paper uses a Recurrent Neural Network (RNN) model based on Gated Recurrent Units (GRU) and analyzes the stock fund flows to attempt to learn and predict price trends from a dynamic perspective. The data used in this study includes the closing price, change rate, and fund flow of a randomly selected A-share stock in the most recent 1000 trading days. The Mean Squared Error (MSE) is used as the model evaluation metric, and the model's MSE can reach 0.013 after training. The model's predictions are

almost consistent with the true price trend, and it can be considered accurate for short-term price trend prediction.

KEY WORDS: RNN；Price Predicting；Stocks

前 言

股票价格预测长久以来都是金融领域的热门研究课题，随着神经网络模型的广泛应用，出现了许多利用神经网络对股票价格进行预测的方法。而对于股票价格此类时间序列数据往往使用循环神经网络进行处理，普通的循环神经网络为了解决梯度消失与梯度爆炸问题需要对梯度进行裁剪因而损失了一些信息，并且可能会过度重视不重要的参数导致模型效果不佳。幸运的是，学术界已经提出许多方法来解决上述问题。最早的是长短期记忆网络(LSTM)[1]，还有其简化变体：门控循环单元(GRU)[2]，GRU 通常可以提供与 LSTM 同等效果，并且计算速度快得多[3]。门控循环神经网络相比普通循环神经网络可以更好地捕获时间步距离很长的序列上的依赖关系。

对与时间序列数据的预测通常假设序列的分布或者说其动力学是不变的，这个假设虽然合理但常常不可被满足。在经济学上，价格波动的动力主要是供需关系，而对于股票来说，资金流向在一定程度上可以代表其供需关系，因此本模型在学习股票价格序列时理论上能获得更多分布信息，并提高准确度。

模型介绍

1. 循环神经网络

对于时间序列数据：

$$x_{t-1}, \dots, x_1$$

[1] Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, *9*(8), 1735–1780.
[2] Cho, K., Van Merriënboer, B., Bahdanau, D., & Bengio, Y. (2014). On the properties of neural machine translation: encoder-decoder approaches. *arXiv preprint arXiv:1409.1259*.
[3] Chung et al., 2014
Chung, J., Gulcehre, C., Cho, K., & Bengio, Y. (2014). Empirical evaluation of gated recurrent neural networks on sequence modeling. arXiv preprint arXiv:1412.3555.

我们常常假设x_t分布为：

$$P(x_t|x_{t-1},\dots,x_1) \tag{1}$$

并借此预测x_t。如何有效估计(1)归结为以下两种策略：

第一种策略，假设在现实情况下相当长的序列可能是不必要的，因此我们只需要满足某个长度为τ的时间跨度，即使用观测序列$x_{t-1},\dots,x_{t-\tau}$.

当下获得的最直接的好处就是参数的数量总是不变的，至少在$t>\tau$时如此，这就使我们能够训练一个上面提及的深度网络。这种模型被称为自回归模型（autoregressive models），因为它们是对自己执行回归。

第二种策略，如图 1 所示， 是保留一些对过去观测的总结h_t，并且同时更新预测$\hat{x}_t$和总结h_t。这就产生了基于$\hat{x}_t = P(x_t \mid h_t)$估计$x_t$，以及公式$h_t = g(h_{t-1},x_{t-1})$更新的模型。由于$h_t$从未被观测到，这类模型也被称为隐变量自回归模型（latent autoregressive models）

[4]

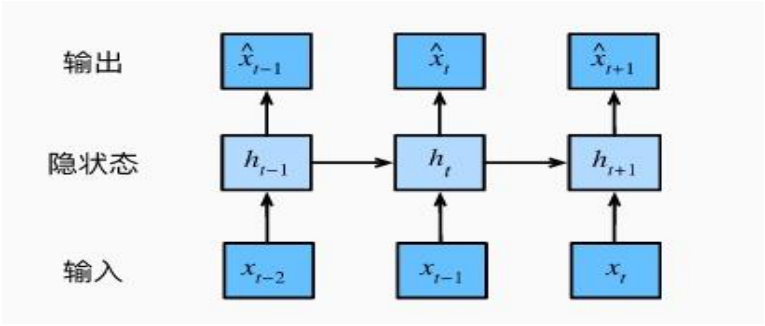


图 1

循环神经网络即是具有隐状态的神经网络。假设我们在时间步t有小批量输入$\mathbf{X_t} \in R^{n\times d}$。换言之，对于$n$个序列样本的小批量，$\mathbf{X_t}$的每一行对应于来自该序列的时间步$t$处的一个样本。接下来，用$\mathbf{H_t} \in R^{n\times h}$表示时间步$t$的隐藏变量。与多层感知机不同的是，这里保存了前一个时间步的隐藏变量$\mathbf{H_{t-1}}$，并引入了一个新的权重参数$\mathbf{W_{hh}} \in R^{h\times h}$，来描述如何在当前时间步中使用前一个时间步

[4] 阿斯顿·张（Aston Zhang） / 李沐（Mu Li） /[美] 扎卡里·C. 立顿（Zachary C. Lipton） /[德] 亚历山大·J. 斯莫拉（Alexander J. Smola）. 动手学深度学习[M]

的隐藏变量。具体地说，当前时间步隐藏变量由当前时间步的输入与前一个时间步的隐藏变量一起计算得出：

$$\mathbf{H_t} = \phi(\mathbf{X_t}\mathbf{W_{xh}} + \mathbf{H_{t-1}}\mathbf{W_{hh}} + \mathbf{b_h}).$$

从相邻时间步的隐藏变量$\mathbf{H_t}$和$\mathbf{H_{t-1}}$之间的关系可知，这些变量捕获并保留了序列直到其当前时间步的历史信息，就如当前时间步下神经网络的状态或记忆，因此这样的隐藏变量被称为隐状态（hidden state）。由于在当前时间步中，隐状态使用的定义与前一个时间步中使用的定义相同，因此计算是循环的（recurrent）。于是基于循环计算的隐状态神经网络被命名为循环神经网络（recurrent neural network）。

对于时间步t，输出层的输出类似于多层感知机中的计算：

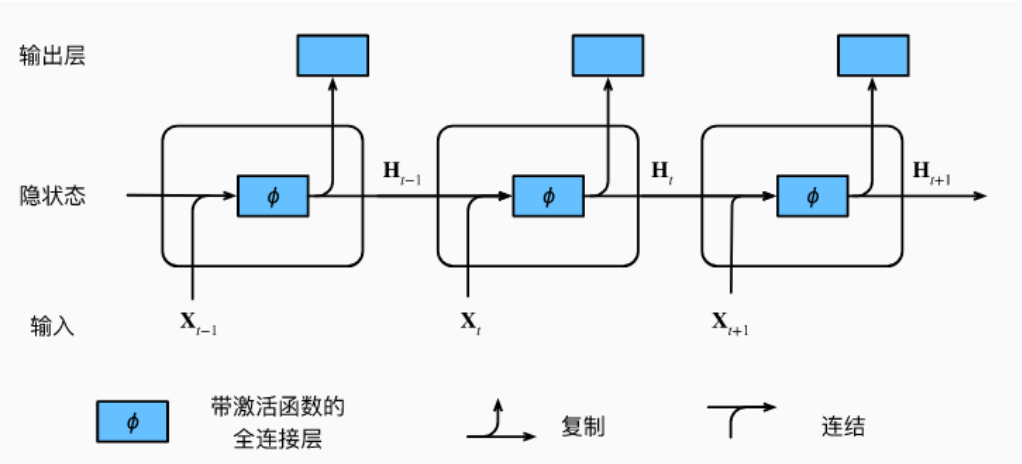
$$\mathbf{O_t} = \mathbf{H_t}\mathbf{W_{hq}} + \mathbf{b_q}.$$

循环神经网络的参数包括隐藏层的权重$\mathbf{W_{xh}} \in R^{d\times h}, \mathbf{W_{hh}} \in R^{h\times h}$和偏置$\mathbf{b_h} \in R^{1\times h}$，以及输出层的权重$\mathbf{W_{hq}} \in R^{h\times q}$和偏置$\mathbf{b_q} \in R^{1\times q}$。

即使在不同的时间步，循环神经网络也总是使用这些模型参数。因此，循环神经网络的参数开销不会随着时间步的增加而增加。

在任意时间步t，隐状态的计算可以被视为：1. 拼接当前时间步t的输入$\mathbf{X_t}$和前一时间步$t-1$的隐状态$\mathbf{H_{t-1}}$；2. 将拼接的结果送入带有激活函数ϕ的全连接层。全连接层的输出是当前时间步t的隐状态$\mathbf{H_t}$。

下图展示了循环神经网络在三个相邻时间步的计算逻辑：



2. 门控循环单元

门控循环单元与普通的循环神经网络之间的关键区别在于：前者支持隐状态的门控。这意味着模型有专门的机制来确定应该何时更新隐状态，以及应该何时重置隐状态。门控循环单元使用重置门（reset gate）和更新门（update gate）控制循环的信息。门控循环单元具有以下两个显著特征：重置门有助于捕获序列中的短期依赖关系，更新门有助于捕获序列中的长期依赖关系。

对于给定的时间步t，假设输入是一个小批量$\mathbf{X_t} \in R^{n\times d}$（样本个数$n$，输入个数$d$），上一个时间步的隐状态是$\mathbf{H_{t-1}} \in R^{n\times h}$（隐藏单元个数$h$）。那么，重置门$\mathbf{R_t} \in R^{n\times h}$和更新门$\mathbf{Z_t} \in R^{n\times h}$的计算如下所示：

$$\mathbf{R_t} = \sigma(\mathbf{X_t}\mathbf{W_{xr}} + \mathbf{H_{t-1}}\mathbf{W_{hr}} + \mathbf{b_r}),$$

$$\mathbf{Z_t} = \sigma(\mathbf{X_t}\mathbf{W_{xz}} + \mathbf{H_{t-1}}\mathbf{W_{hz}} + \mathbf{b_z})$$

其中$\mathbf{W_{xr}}, \mathbf{W_{xz}} \in R^{d\times h}$和$\mathbf{W_{hr}}, \mathbf{W_{hz}} \in R^{h\times h}$是权重参数，$\mathbf{b_r}, \mathbf{b_z} \in R^{1\times h}$是偏置参数

将重置门$\mathbf{R_t}$与 RNN 的常规隐状态更新机制集成，得到在时间步t的候选隐状态（candidate hidden state）$\widetilde{\mathbf{H}_t} \in R^{n\times h}$

$$\widetilde{\mathbf{H}_t} = \tanh(\mathbf{X_t}\mathbf{W_{xh}} + (\mathbf{R_t} \odot \mathbf{H_{t-1}})\mathbf{W_{hh}} + \mathbf{b_h}),$$

其中$\mathbf{W_{xh}} \in R^{d\times h}$和$\mathbf{W_{hh}} \in R^{h\times h}$是权重参数，$\mathbf{b_h} \in R^{1\times h}$是偏置项，符号⊙是按元素乘积运算符。这里使用 tanh 将值限制在(-1,1)。

计算候选隐状态后，结合更新门$\mathbf{Z_t}$的效果。这一步确定新的隐状态$\mathbf{H_t} \in R^{n\times h}$在多大程度上来自旧的状态$\mathbf{H_{t-1}}$和新的候选状态$\widetilde{\mathbf{H}_t}$。更新门$\mathbf{Z_t}$仅需要在$\mathbf{H_{t-1}}$和$\widetilde{\mathbf{H}_t}$之间进行按元素的凸组合就可以实现这个目标。

这就得出了门控循环单元的最终更新公式：

$$\mathbf{H_t} = \mathbf{Z_t} \odot \mathbf{H_{t-1}} + (1 - \mathbf{Z_t}) \odot \widetilde{\mathbf{H}_t}.$$

实验过程

1. 数据来源

本次实验数据来源于从新浪财经，使用 request 包爬取的随机一个股票最近 1000 个交易日的资金流向数据，包括：″日期″, ″收盘价″, ″涨跌幅″, ″换

手率", "净流入额","净流入占比", "超大单净流入", "大单净流入", "小单净流入", "散单净流入"。

2. 数据预处理

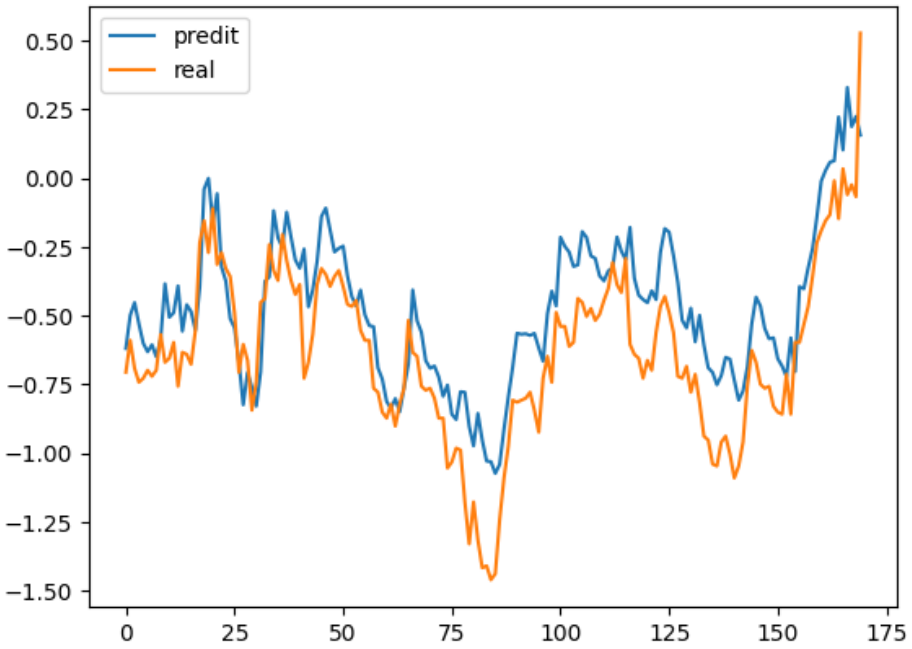
使用零均值归一化的方法进行预处理，并使用滑动窗口方法转换为时间步长为 30 的时间序列数据。以 0.8 的比例划分训练集与测试集。

3. 模型建立与训练

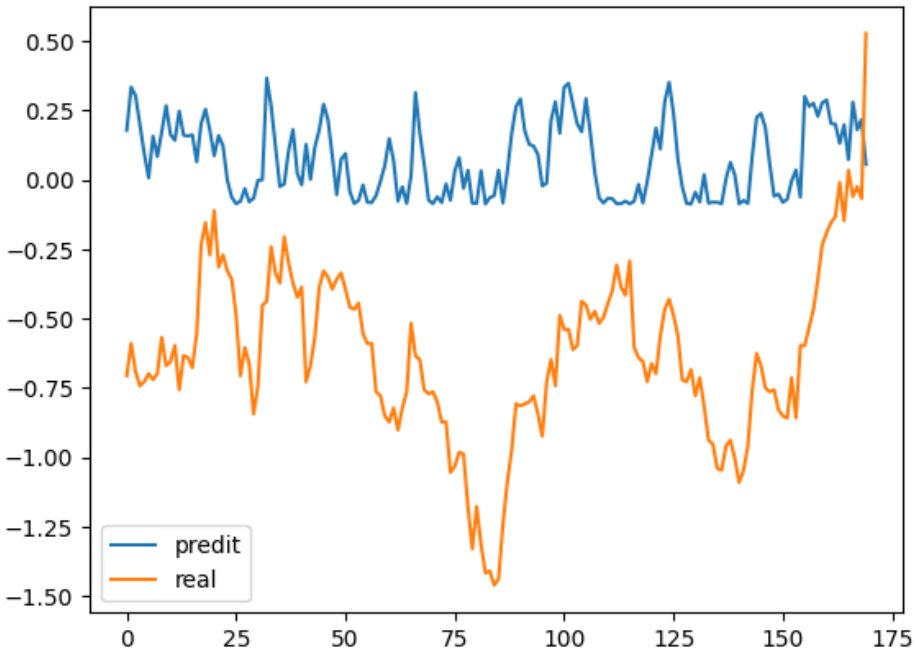
使用 TensorFlow 的深度学习框架构建五层神经网络模型，其中前四层为 GRU 层，隐藏单元均设置为 128，dropout 均设置为 0.2；最后一层为一个单元的全连接层；循环训练次数(epoch)为 10 次，循环训练的小批量样本数(batch)为 16；使用 Adam 算法优化器；MSE 作为损失函数；学习率 0.001。

4. 模型预测

使用训练好的模型对测试集进行预测，得到 MSE 为 0.014，将真实数据与预测数据进行比较：



同时，使用几个经典机器学习算法：普通 BP 神经网络，支持向量机(SVM)，logit 回归，决策树对相同数据进行训练与预测，MSE 都是本文模型的上百倍，且其价格趋势也与真实情况不符合，下图是普通 BP 神经网络的预测与真实对比：



这几个模型都与之类似，可以说是没有准确的预测能力。

结论

本本文使用 MSE 作为模型评估指标，模型在训练后 MSE 能达到 0.014，模型预测与真实价格趋势上几乎一致，能够认为此模型对短期价格趋势预测较为准确。本文利用价格波动的动力主要是供需关系，而对于股票来说，资金流向在一定程度上可以代表其供需关系，因此本模型在学习股票价格序列时能获得更多分布信息，提高了准确度。

[i] 阿斯顿·张（Aston Zhang） / 李沐（Mu Li） / [美] 扎卡里·C. 立顿（Zachary C. Lipton） / [德] 亚历山大·J. 斯莫拉（Alexander J. Smola）. 动手学深度学习[M]

附录：

```
import numpy as np
from d2l import tensorflow as d2l
import tensorflow as tf
import matplotlib.pyplot as plt
import pandas as pd
import json
import requests
response = requests.get(
    "http://vip.stock.finance.sina.com.cn/quotes_service/api/json_v2.php
/MoneyFlow.ssl_qsfx_lscjfb?page=1&num=1000&sort=opendate&asc=0&daima=sz
300311",
)

# 通过json.loads()方法将json字符串转换为python对象

a = json.loads(response.text)
# 将a转换为DataFrame

data = pd.DataFrame(a)
data.drop(['r0', 'r1', 'r2', 'r3'], axis=1, inplace=True)

# 修改列名
data.columns = ["日期", "收盘价", "涨跌幅", "换手率", "净流入额",
                "净流入占比", "超大单净流入", "大单净流入", "小单净流入", "
散单净流入"]
#
# 修改数据类型
data['日期'] = pd.to_datetime(data['日期'])
data['收盘价'] = data['收盘价'].astype(float)
data['涨跌幅'] = data['涨跌幅'].astype(float)
data['换手率'] = data['换手率'].astype(float)
data['净流入额'] = data['净流入额'].astype(float)
data['净流入占比'] = data['净流入占比'].astype(float)
data['超大单净流入'] = data['超大单净流入'].astype(float)
data['大单净流入'] = data['大单净流入'].astype(float)
data['小单净流入'] = data['小单净流入'].astype(float)
data['散单净流入'] = data['散单净流入'].astype(float)
data = data.sort_values(by='日期', ascending=True)
# 将日期设置为索引
data.set_index('日期', inplace=True)
```

```
data.head()
# 画出走势图

plt.figure(figsize=(20, 10))
plt.plot(data.index, data['收盘价'])

# 使用 GRU 对涨幅单步预测

# 数据预处理
# 归一化
data = (data - data.mean()) / data.std()
n_samples = 1000
n_features = 9
seq_len = 30
# 使用滑动窗口将数据转换为时间序列数据


def sliding_window(DataSet, X_width, y_width, gap=1, multi_vector=None,
X_data=True):
    '''
    DataSet has to be as a DataFrame
    '''
    if X_data:
        if multi_vector:
            a, b = DataSet.shape
        else:
            a = DataSet.shape[0]
            b = 1
        c = (a-X_width-y_width-a % gap)/gap
        X = np.reshape(DataSet.iloc[0:X_width, :].values, (1, X_width,
b))
        for i in range(len(DataSet) - X_width - y_width):
            i += 1
            if i > c:
                break
            j = i * gap
            tmp = DataSet.iloc[j:j + X_width, :].values
            tmp = np.reshape(tmp, (1, X_width, b))
            X = np.concatenate([X, tmp], 0)
        return X
    else:
        if multi_vector:
            print('y_data-error: expect 1D ,given %dD' %
DataSet.shape[1])
            return
```

```
        else:
            a = DataSet.shape[0]
        c = (a-X_width-y_width-a % gap)/gap
        y = np.reshape(
            DataSet.iloc[X_width:X_width + y_width].values, (1,
y_width))
        for i in range(len(DataSet) - X_width - y_width):
            i += 1
            if i > c:
                break
            j = i * gap + X_width
            tmp = DataSet.iloc[j:j + y_width].values
            tmp = np.reshape(tmp, (1, y_width))
            y = np.concatenate([y, tmp])
        return y


X = sliding_window(data, seq_len, 1, 1, True, True)

y = sliding_window(data["收盘价"], seq_len, 1, 1, multi_vector=False,
X_data=None)

train_rate = 0.8
train_num = int(n_samples * train_rate)


X_train = np.array(X[:train_num])
X_test = np.array(X[train_num:])

y_train = y[:train_num]
y_test = y[train_num:]
X_train.shape, y_train.shape


# 定义模型结构
model = tf.keras.Sequential([
    tf.keras.layers.GRU(256, input_shape=(seq_len, n_features),
                        return_sequences=True, dropout=0.2),
    tf.keras.layers.GRU(256, return_sequences=True, dropout=0.2),
    tf.keras.layers.GRU(128, return_sequences=True, dropout=0.2),
    tf.keras.layers.GRU(256, dropout=0.4),
    tf.keras.layers.Dense(1),

])
```

```
# 编译模型
model.compile(optimizer='adam', loss='mse',
              metrics=['mse'])

# 训练模型
model.fit(X_train, y_train, epochs=10, batch_size=16,
          validation_data=(X_test, y_test))

# 模型预测
preds = model.predict(X_test)

# 模型评估
loss = model.evaluate(X_test, y_test)
print('Test loss:', loss)


preds.shape

# 画图并显示图例

plt.plot(preds, label='predit',)
plt.plot(y_test, label='real',)
plt.legend()
```