金融统计课程作业

题目：比特币价格分析预测

姓　　名　刘程华

学　　号　2018011687

学　　院　清华大学计算机系

专　　业　计算机科学与技术

定稿日期　2021 年 6 月

比特币价格分析预测

摘要

本文借助时间序列模型和深度学习模型分别研究了比特币的的每日价格走势和分时价格走势。在对数据经过假设检验后，合理的使用 ARMA 模型对比特币价格进行了建模预测，并分析了该模型的优势与不足。同时本人也尝试使用深度学习模型对比特币进行建模预测，并根据结果提出了现有模型的问题和深度学习模型如何更好的预测比特币价格的思路，并给出了的预测数据集的构建想法。

关键字：ARMA,N-BEATS；ARMA；N-BEATS

1 序言

1.1 研究背景及意义

比特币是一种 P2P 形式的虚拟的加密数字货币，点对点的传输意味着一个去中心化的支付系统。自 08 年中本聪提出以来，比特币越来越频繁的出现在人们的视野中，伴随着人们对其价格的强烈反应。人们惊叹于比特币的短期翻倍似的暴涨，也惊愕于比特币几天腰斩的暴跌。2020 年全球疫情以来，比特币从不到 1 万美元一路高歌猛进，在 2021 年 4 月 13 日比特币价格再创了历史新高，突破 6.3 万美元关口，总市值超过一万亿美元。对比特币价格的分析预测有着极大的价值回报，优秀的预测能够直接带来高额的收益。

1.2 本文贡献与创新点

本文对数据经过假设检验后，不仅从传统的时间序列模型对比特币价格进行了建模预测分析，分析了时间序列模型的优势与不足，也尝试使用深度学习模型对比特币进行建模预测，并根据结果提出了现有模型的问题和深度学习模型如何更好的预测比特币价格的思路，并给出了的预测数据集的构建想法。

1.3 文章基本思路与结构

文章分为四大部分，分别为数据的获取、假设检验、时间序列模型和深度学习模型。

2 数据获取和概览

2.1 每日数据

2.1.1 数据的获取

本人用 python 从coinmarketcap.com网站爬取了从 2014 年 2 月 2 日到 2021 年 6 月 13 日的每日价格和交易量数据一共 2689 条。

2.1.2 数据概览

我们将所得到的数据（2014.02.02-2021.06.13）分别绘制价格走势图2-1和交易量走势图2-2。从价格走势图中我们可以看出，比特币的价格有两个明显的回落，18 年年初和 21 年中。由于后期比特币价格经历了非常大的增长，通过走势图我们很难发现比特币前半段时间的价格和交易量变化。为了更直观的观察比特币的价格变化，我们计算并绘制比特币价格的对数收益率图2-3。从对数收益率图中我们可以看出，整体来看，比特币每日价格的收益率是比较大的，绝对值经常会突破 0.1，有些

日子突破了 0.2，其中有天竟然达到了 0.4。为了能够验证模型的好坏，我们将数据从 2021.05.01 分为前后两段，前段用于模型的训练，后面一小段用于模型的验证。

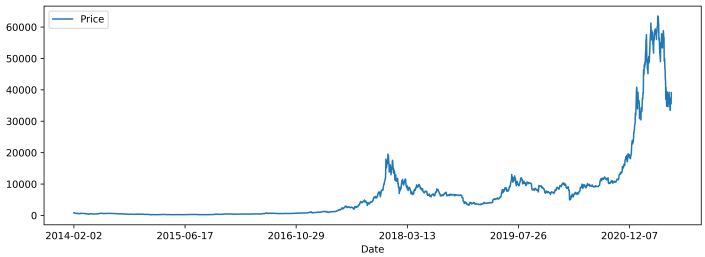

图 2-1: 比特币价格走势

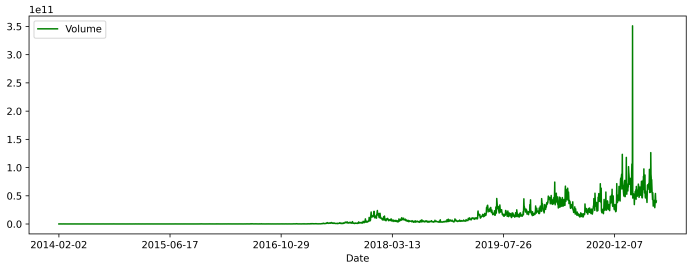

图 2-2: 比特币交易量走势

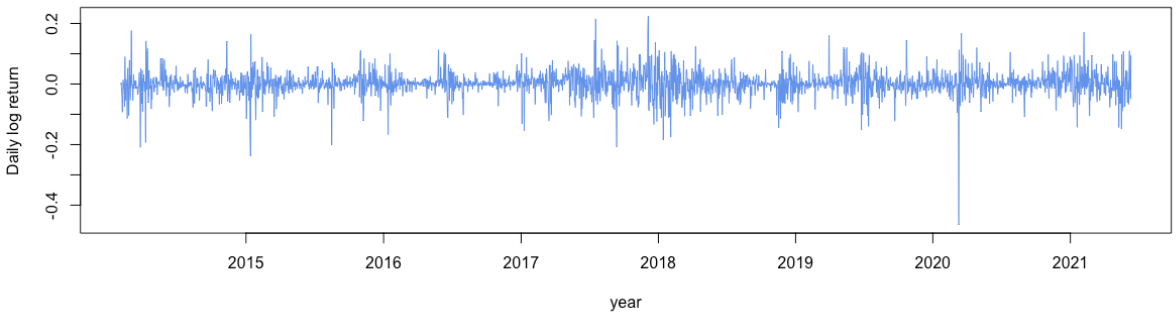

图 2-3: 比特币对数收益率

2.2 分时数据

通过时间序列模型建模得到的模型预测效果不十分令人满意，我考虑使用更细腻、维度更高的数据。每日数据的时间分隔较大，对应比特币价格的波动也变大，也缺乏交易情况的具体信息，为此我借助 python 通过调用poloniex.com的 api 获取了每隔 300 秒的交易数据。poloniex 是知名的一站式数字货币交易所，交易量较大，分时数据能够体现真实货币价格的变化情况。具体的，我们获取的分时数据共有 8 个维度，分别为日期 date，最高价格 high，最低价格 low，开始价格 open，结

束价格 close，总交易量 volume，总交易金额 quoteVolume，平均价格 weightedAverage。由于分时数据总量过于庞大，我们截取 2021 年 5 月 1 号 0 时以后的数据作为我们的研究对象。

3 假设与检验

我们后面的模型建立在一定的假设之上，为了验证假设的合理性，我们需要对数据进行假设检验。篇幅有限，下面只分析每日数据，具体的见下。

3.1 收益率正态性检验

我们绘制日收益率的密度直方图以及 Q-Q 图3-1。从图中可以看出，收益率呈现明显的厚尾状。另外，在第一副图中，我们可以看出大于 0 部分要明显高于小于 0 的部分。这也和比特币涨多天数大于跌多天数的整体认知相符。在 Q-Q 图中，两端数据与直线有着较大的偏差，与正态偏离明显。

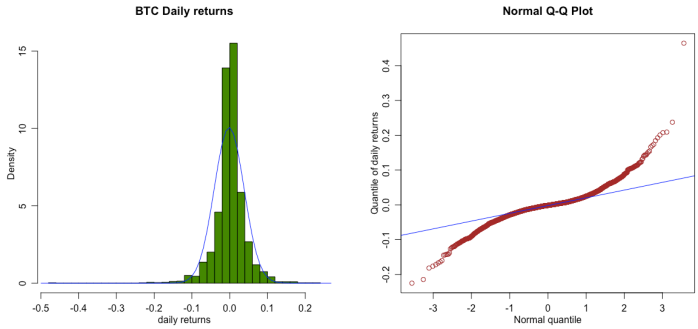

图 3-1: 收益率分布和 Q-Q 图

为了检验数据是否服从正态分布，我们采用了对大样本很有效的非参检验 Kolmogorov-Smirnov 检验3-1，和基于偏度和峰度的拟合优度的检验 Jarque-Bera 检验3-2。由于样本量远大于 50，所以我们不采用 Shapiro-Wilk 检验来检验模型的正态性。

表 3-1: Kolmogorov-Smirnov 检验

D	p-value
0.11237	< 2.2e-16

表 3-2: Jarque-Bera 检验

X-squared	p-value
14670	< 2.2e-16

我们发现两种检验给出的结果都认为数据不符合正态假设。为了解决这一问题，我尝试采用广义幂变换 Box-Cox 变换对收益率进行处理。由于收益率有负值，我们先整体向上平移 0.48 然后进行变换3-2，得到 $\lambda = 1.79798$，带入 $X = (logreturn + 0.48)^{1.79798}$ 后再次做 K-S 检验和 J-B 检验，发现对应统计量有一定的减小，但 p 值仍然小于 2.2×10^{-16}。一方面，我们的 Box-Cox 变换没有较

好的改进，另一方面，对 logreturn 的变换会导致原始价格数据极为复杂的变换，最终我并没有选择进行数据变换。

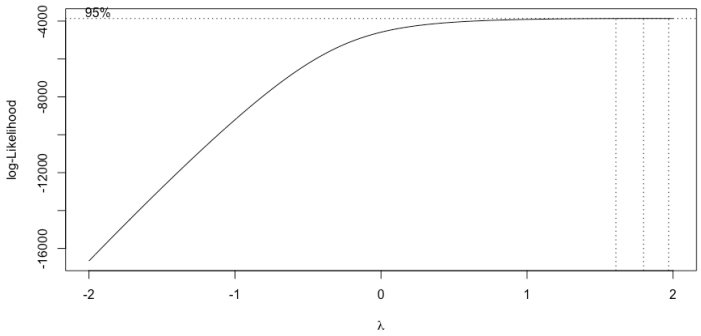

图 3-2: Box-Cox 变换

3.2 平稳性检验

时间序列的平稳性是时间序列模型有效的根基。因此，我们要优先考虑检验序列的平稳性。直观上的，我们可以通过图2-1和图2-3来判断得出价格非平稳、收益率平稳，下面的 ACF 图3-3也能较好的佐证这一点。除此之外，我们还可以通过 Dickey-Fuller 检验来验证我们的想法。经过计算，price 的 p 值为 0.7868，logreturn 的 p 值为 0.01。在显著性水平 $\alpha = 0.05$ 下，我们认为 logreturn 是平稳的，price 是非平稳的。

3.3 ACF 及 PACF

价格和收益率随时间变动时往往存在自相关的特征。为了验证并分析自相关性，对价格和收益率分别绘制自相关系数 ACF 和偏自相关系数 PACF，见图3-3。这两者在时间序列模型的构建中起着指导作用。

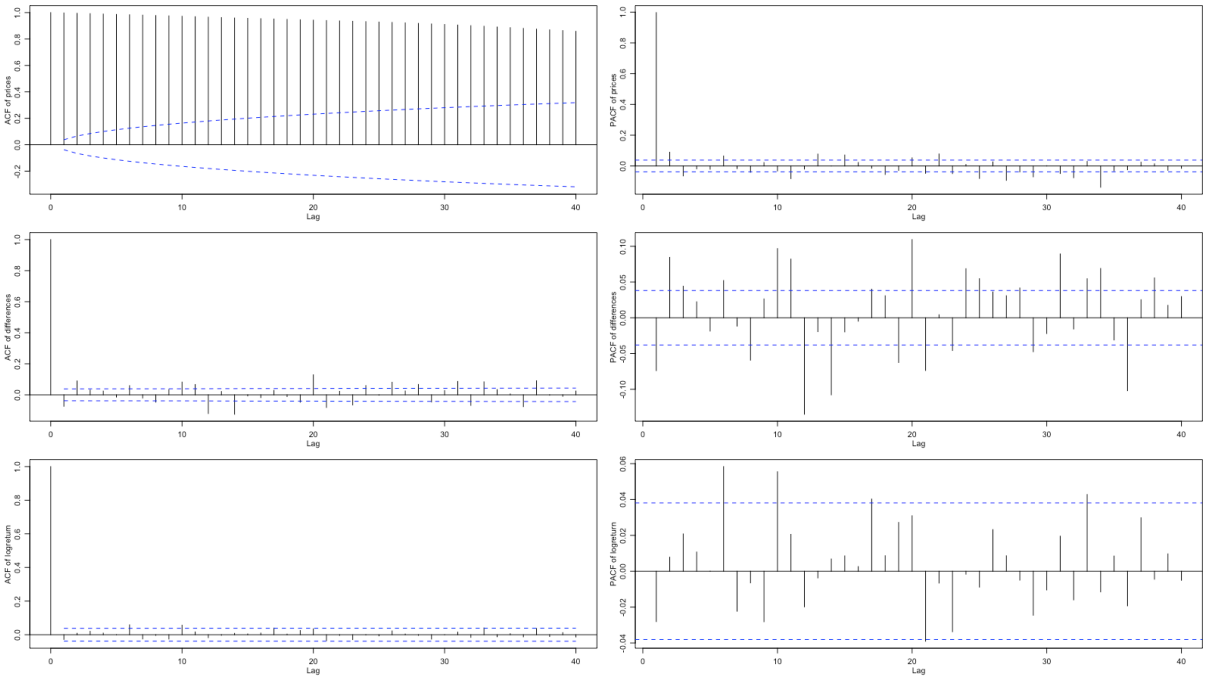

图 3-3: ACF 和 PACF

我们定义

- 截尾：在某阶后迅速趋于 0（后面大部分阶的对应值在二倍标准差以内）。
- 拖尾：按指数衰减或震荡，值到后面还有增大的情况。

原始价格 Price 的 ACF 呈拖尾，PACF 呈截尾；做差后和做差后取对数（收益率）的情况大致相同，ACF 呈不明显的截尾，PACF 呈拖尾。事实上，ACF 和 PACF 为我们选择时间序列模型提供了指南。具体的如下：

- AR 模型：ACF 拖尾，PACF 截尾，通过 PACF 定阶。
- MA 模型：ACF 截尾，PACF 拖尾，通过 ACF 定阶。
- ARMA 模型：都拖尾。通过 EACF 定阶。

4 时间序列模型

4.1 模型选择

回顾假设检验章节中平稳性检验小节，我们选择平稳的 logreturn 序列作为我们的目标对象。首先我们要分析序列是否具有季节性，考虑季节性分解。存在季节性因素的时间序列数据可以被分解为趋势因子、季节性因子和随机因子。趋势因子能捕捉到长期变化；季节性因能捕捉到一年内的周期性变化；而随机（误差）因子则能捕捉到那些不能被趋势或季节效应解释的变化。直观上来看，比

特币存在周期性减半。具体来说，比特币的总供应量为 2100 万枚。比特币的区块挖矿奖励每 21 万个区块奖励减半，按照比特币大约 10 分钟出块的时间间隔来算，减半的时间间隔大约为每四年一次。由于产量周期减半的存在，我们考虑季节性分解的可能性，但遗憾的是并没有得到结果，我认为这主要是因为周期太长导致周期数太少无法分解。但这种周期存在的可能性给我们后续的预测提供了一个好的思路。

接下来我们选择合适的时间序列模型以及合适的阶。回顾假设检验章节中 ACF 及 PACF 中的讨论，我们选择 ARMA 模型（ARMA 模型包含了 MA 模型）来尝试建模。

4.2 模型建立与检验

我们根据 AIC 赤池信息度量准则选择合适的阶。我们设置搜索范围 p 和 q 都为 0 到 10，得到最小的 AIC= -9721.63 （p= 4 ,q= 7）。比特币收益率的 ARMA（4，7）模型为

$$\begin{aligned} r_t = &-0.9148r_{t-1} - 0.4414r_{t-2} - 1.0129r_{t-3} - 0.8326r_{t-4} + \epsilon_t - 0.0994\epsilon_{t-1} - 0.4785\epsilon_{t-2} \\ &+ 0.6036\epsilon_{t-3} - 0.1777\epsilon_{t-4} - 0.8439\epsilon_{t-5} + 0.0474\epsilon_{t-6} - 0.0332\epsilon_{t-7}, \quad \epsilon_t \sim WN\left(0, \sigma^2\right) \end{aligned} \tag{1}$$

模型的 $\sigma^2 = 0.001523$，可以认为解释度是比较好的。我们对模型拟合的残差做检验，得到图4-1。用 Ljung-Box 检验得到 $Q^* = 5.0624$，p 值为 0.1673，可以认为残差是白噪声，我们的模型结果是符合假设的。

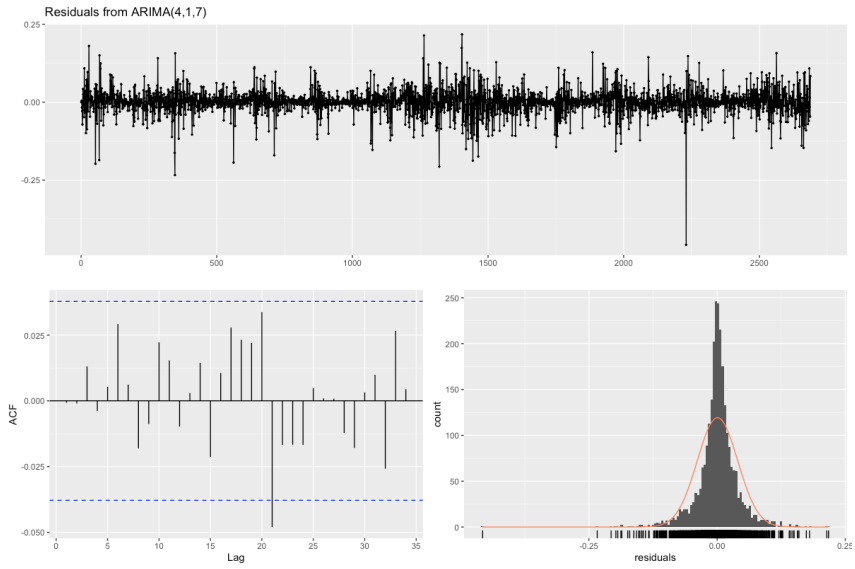

图 4-1: ACF 和 PACF

4.3 预测分析

我们用 21 年 5 月 1 号之前的数据拟合，预测此后的变化趋势，见图4-2。图中的阴影区域分别为 80% 的置信区间和%95 的置信区间，紫色曲线为我们的预测值，绿色为真实的收益率。可以看到，模型的预测效果是较差的，收益率的预测值几乎都大于等于 0，这与实际情况相差严重。但这种

现象是容易理解的，因为我们需要通过预测得到的值来预测下一时间的值，这会导致不确定性不断地放大，比较远的预测值就变得不可靠了。

值得注意的是，在预测结果的前 5 天中预测值和真实值的变化趋势是完全相同的，为了验证这一现象是否普遍成立，我选取不同的时间点来训练并检验，发现大部分情况下预测开始时后面 3 天的变化趋势是相同的，但这种趋势的相同提供给我们的信息着实有限，因为我们更多关心的是收益率的正负而非升降。

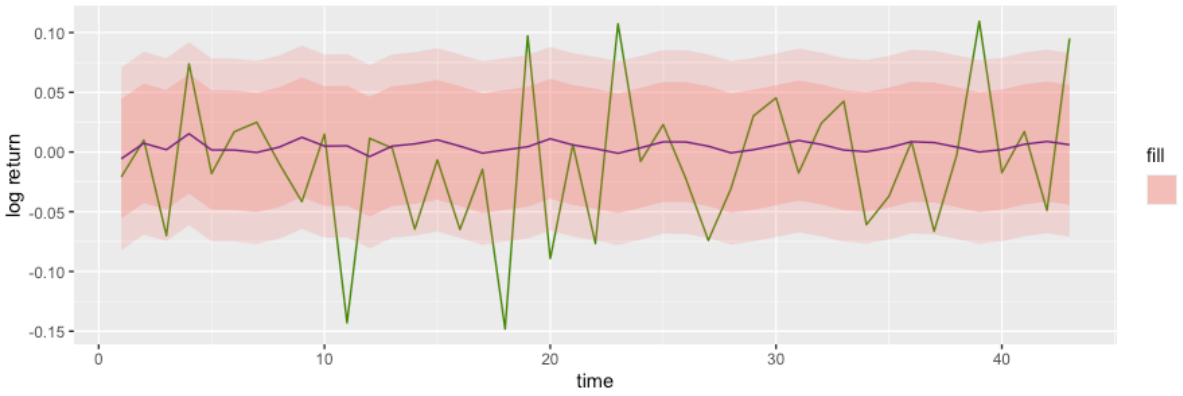

图 4-2: 43 天收益率预测

5 深度学习模型

考虑到时间序列模型所得到的结果不是非常令人满意，我考虑使用当前热门的深度学习来进行预测分析。

5.1 N-BEATS

5.1.1 模型介绍

我们尝试使用发表于 ICLR2020 中的 N-BEATS 模型。N-BEATS 文章第一次提出一种可解释的深度学习时间序列预测架构，可以实现单变量多对多预测，网络结构见图5-1。该网络结构有两个作用，一个叫复原，一个是预测，复原的作用是进行有效的时间序列特征提取，发挥深度学习自动特征学习的优势。整个网络的 Block 的设计与迭代决策树的思想类似，用网络学习残差，先让神经网络学习简单的规律，再学习复杂的规律。

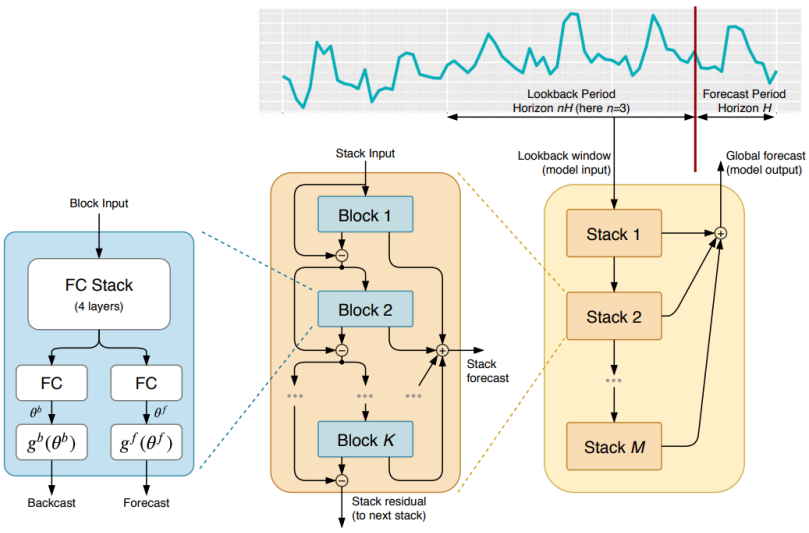

图 5-1: N-BEATS 网络结构图

其中，每个 block 分为两部分，第一部分用于生成后项展开系数和前项展开系数，原始时间序列输入经过四层全连接后，经过线性投影层生成后项展开系数 θ_l^b 和前项展开系数 θ_l^f，线性投影层就是简单的线性变换，例如 $\theta_l^f = W_l^f h_{l,4}$ 。

$$h_{\ell,1} = FC_{\ell,1}\left(x_\ell\right), \quad h_{\ell,2} = FC_{\ell,2}\left(h_{\ell,1}\right), \quad h_{\ell,3} = FC_{\ell,3}\left(h_{\ell,2}\right), \quad h_{\ell,4} = FC_{\ell,4}\left(h_{\ell,3}\right)$$

$$\theta_\ell^b = \text{LINEAR}_\ell^b\left(h_{\ell,4}\right), \quad \theta_\ell^f = \text{LINEAR}_\ell^f\left(h_{\ell,4}\right)$$

第二部分将后项展开系数 θ_l^b 和前项展开系数 θ_l^f 通过 basis layers 输出结果

$$\widehat{y}_\ell = \sum_{i=1}^{\dim\left(\theta_\ell^f\right)} \theta_{\ell,i}^f v_i^f, \quad \widehat{x}_\ell = \sum_{i=1}^{\dim\left(\theta_\ell^b\right)} \theta_{\ell,i}^b v_i^b$$

不同 bock 之间通过残差模块进行衔接。公式如下所示，下一个 block 目标是学习上一个 block 转换得到的残差。

$$x_\ell = x_{\ell-1} - \widehat{x}_{\ell-1}, \quad \widehat{y} = \sum_\ell \widehat{y}_\ell$$

接着来说一下模型的可解释性。作者提出两种结构，一种不依赖于特定经验知识，该变化可以看做在时间的波形重建，由于没有额外的限制在 V_l^f 上，导致生成的 $\hat{y}_l$ 解释性比较差。另一种则通过引入专家经验来满足可解释性要求，即通过引入时间序列的趋势性 (trend) 和季节性 (seasonality) 来实现可解释性。具体方法为人为设定与 θ 相乘的 V 矩阵使得该矩阵满足趋势性和季节性的要求。第二种结构可以通过设计相乘矩阵的数值形式来实现专家经验的引入。

$$\widehat{y}_\ell = V_\ell^f \theta_\ell^f + b_\ell^f, \quad \widehat{x}_\ell = V_\ell^b \theta_\ell^b + b_\ell^b$$

$$\widehat{y}_{s,\ell}^{tr} = T\theta_{s,\ell}^f$$

最终通过模型的集成完成了 N-BEATS 网络结构。具体的，通过两种类型的 ensemble 进行结果的输出,SMAPE, MASE 和 MAPE, SPAME 四种优化目标融合，以及不同时间窗口进行融合。

5.1.2 实验结果

我们使用间隔为 300 秒的分时数据来对模型进行训练，选取预测长度为 100，对价格进行了归一化处理（以价格除以最高价格代替）。实验结果见图5-2。

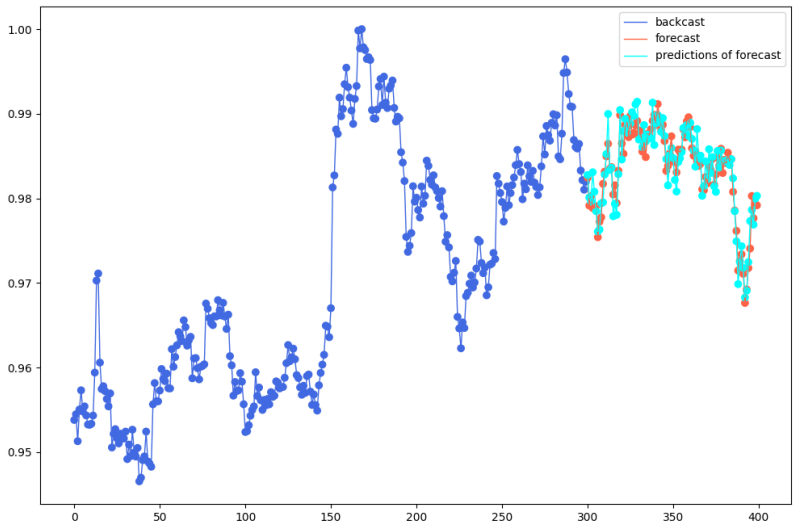

图 5-2: steps=10000

可以看到模型对价格有着很好的预测，令人难以相信。但另一方面，我们不禁担忧这是否是过拟合导致的结果？这需要做更多的实验和测试，限于时间原因，我并没有继续深入探究。

5.2 LSTM

另一种适合此问题非常流行的网络是长短期记忆网络 LSTM。LSTM 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。针对 RNN 存在的问题，LSTM 主要有两点改进：

1. 设置专门的变量 C_t 来存储单元状态（Cell State），从而使网络具有长期记忆。
2. 引入”门运算”，将梯度中的累乘变为累加，解决梯度消失问题。

和上一个模型结果类似，我们拿价格数据来做训练和预测出现了非常严重的过拟合情况，曲线几乎重合，但细看发现预测曲线延迟一个时间点，这意味着训练出来的照抄了前一点的值，这样训练出来的模型没有用处。

5.3 思考与改进

通过上面的实验，可以发现神经网络具有很强的学习能力，这也意味着非常容易过拟合。如果我们想让模型学习到有用的信息，则需要大规模多维度数据集，因此需要准备大量、高质量并且带有干净标签的数据。由于时间有限，我无法准备其他维度的数据，下面我将分析对模型预测比较有用的数据。

比特币发行方不是公司，因此没有公司资产负债表，也无法依靠成本、收入或利润来衡量其发展状况。与投资传统货币也不同，因为它不是由中央银行发行，也不是由政府支持。因此，通常影响货币价值的货币政策、通货膨胀率和经济增长指标并不适用。但比特币的价格是非管制价格，他的价格也是由供需关系决定的。从供应端来看，矿工挖矿的产出和拥有比特币的人出售两大因素。从需求端来看，主要有以下三种情况：

1. 购买比特币进行储值、投资，避免法币通货膨胀。
2. 购买比特币进行使用，例如支付。跨国汇款、结算。
3. 购买比特币进行投机、炒作。

在此基础上通过分析得到以下几个我认为比较有用的外部数据：

1. 市场情绪。一个可行的做法就是抓取网络中玩家关于比特币的言论，分析其中的情感。具体的，我们可以爬去 Twitter、Facebook 的 bitcoin 话题下的实时内容。但单一的市场情绪是不够的，因为大部分玩家的认知和真实情况存在偏差，因此我们不能只局限于此。
2. 比特币产出。比特币矿工的挖矿产量并不是恒定不变的，受到比特币价格、挖矿工具的影响。我们可以关注比特币实时全球总产量和比特币矿机的更新情况。
3. 货币政策。2021 年随着美联储大水漫灌，比特币价格也水涨船高。比特币的价格和全球的的货币政策有着密切地联系，我们可以关注世界各大主要经济体的货币政策信息（尤其是美国）用于比特币的预测。
4. 法律监管。过去经验表明，比特币的暴跌往往由于监管政策的出台。我们需要格外各国政府的监管政策，这会导致比特币的剧烈波动。

6 参考文献

[1] Oreshkin B N, Carpov D, Chapados N, et al. N-BEATS: Neural basis expansion analysis for interpretable time series forecasting[J]. arXiv preprint arXiv:1905.10437, 2019.

7 附录

7.1 核心代码

7.1.1 数据获取和概览

```
import time
import requests
import json
import csv

time_stamp = int(time.time())
print(f"Now timestamp: {time_stamp}")
request_link = f"https://web-api.coinmarketcap.com/v1/cryptocurrency/ohlcv/historical?convert=USD&slug=bitcoin&time_end={time_stamp}&time_start=1391221787"
print("Request link: " + request_link)
r = requests.get(url = request_link)
#print(r.content)
# 返回的数据是 JSON 格式，使用 json 模块解析
content = json.loads(r.content)
#print(type(content))
quoteList = content['data']['quotes']
#print(quoteList)
with open('BTC.csv','w' ,encoding='utf8',newline='') as f:
    csv_write = csv.writer(f)
    csv_head = ["Date","Price","Volume"]
    csv_write.writerow(csv_head)

    for quote in quoteList:
        quote_date = quote["time_open"][:10]
        quote_price = "{:.2f}".format(quote["quote"]["USD"]["close"])
        quote_volume = "{:.2f}".format(quote["quote"]["USD"]["volume"])
        csv_write.writerow([quote_date, quote_price, quote_volume])

print("Done")
```

```
import pandas as pd

series = pd.DataFrame()
df = pd.read_csv("BTC.csv")
series['Date'] = df['Date'].tolist()
series['Price'] = df['Price'].tolist()
series['Volume'] = df['Volume'].tolist()

print(series)


import matplotlib.pyplot as plt


ax = plt.gca()
series.plot(kind='line', x='Date', y='Price', ax=ax)
plt.savefig('price.png')
plt.show()

plt.cla()
ax = plt.gca()
series.plot(kind='line', x='Date', y='Volume',color="green", ax=ax)
plt.savefig('volume.png')
plt.show()
```

```
import json
import numpy as np
import os
import pandas as pd
import urllib.request as urllib2

import ssl
ssl._create_default_https_context = ssl._create_unverified_context

coins = ['BTC']
```

```
df_list=[]
for coin in coins:
    url = 'https://poloniex.com/public?command=returnChartData&currencyPair
        =USDT_'+coin+'&start=1619798400&end=9999999999&period=300&
        resolution=auto'
    openUrl = urllib2.urlopen(url)
    r = openUrl.read()
    openUrl.close()
    d = json.loads(r.decode())
    print (pd.DataFrame(d).shape)



df = pd.DataFrame(d)
original_columns=[u'close', u'date', u'high', u'low', u'open']
new_columns = ['Close','Timestamp','High','Low','Open']
df = df.loc[:,original_columns]
df.columns = new_columns
df.head()


df.to_csv('/Users/liuchenghua/Downloads/金融统计模板/code/bitcoin2021.5
    toNOW.csv',index=None)



import pandas as pd
import numpy as np
import h5py
input_step_size = 50
output_size = 30
sliding_window = False
file_name= 'bitcoi_prediction.h5'

```

```
df = pd.read_csv('/Users/liuchenghua/Downloads/金融统计模板/code/
    bitcoin2021.5toNOW.csv').dropna().tail(1000000)
df['Datetime'] = pd.to_datetime(df['Timestamp'],unit='s')
df.head()

prices= df.loc[:,'Close'].values
times = df.loc[:,'Datetime'].values
prices.shape


outputs = []
inputs = []
output_times = []
input_times = []
if sliding_window:
    for i in range(len(prices)-input_step_size-output_size):
        inputs.append(prices[i:i + input_step_size])
        input_times.append(times[i:i + input_step_size])
        outputs.append(prices[i + input_step_size: i + input_step_size+
            output_size])
        output_times.append(times[i + input_step_size: i + input_step_size+
             output_size])
else:
    for i in range(0,len(prices)-input_step_size-output_size, input_step_
        size):
        inputs.append(prices[i:i + input_step_size])
        input_times.append(times[i:i + input_step_size])
        outputs.append(prices[i + input_step_size: i + input_step_size+
            output_size])
        output_times.append(times[i + input_step_size: i + input_step_size+
             output_size])
inputs= np.array(inputs)
outputs= np.array(outputs)
output_times = np.array(output_times)
```

```
input_times = np.array(input_times)

with h5py.File(file_name, 'w') as f:
    f.create_dataset("inputs", data = inputs)
    f.create_dataset('outputs', data = outputs)
#     f.create_dataset("input_times", data = input_times)
#     f.create_dataset('output_times', data = output_times)

np.save(file_name[:-3] + 'input_times',input_times)
np.save(file_name[:-3] + 'output_times',output_times)
##########

import h5py
with h5py.File(''.join(['/Users/liuchenghua/Downloads/金融统计模板/code/
    bitcoi_prediction.h5']), 'r') as hf:
    datas = hf['inputs']
    labels = hf['outputs']
```

7.1.2 假设检验和时间序列模型

```
btc<- read.csv(file ="BTC.csv")


price=btc["Price"]
d1=price[1:2688,1]
d2=price[2:2689,1]
plot(log(d2)-log(d1), type='l', ylab="Daily log return", xaxt="n",xlab='
    year',col="cornflowerblue")
axis(1, at=c(334, 700, 1066, 1431, 1796, 2161, 2527),
     labels=c("2015", "2016", "2017", "2018", "2019", "2020", "2021"))



d1=price[1:2688,1]
d2=price[2:2689,1]
```

```r

ps.options(horizontal=F, width=2, height=4, pointsize=12)
par(mar=c(5,4,4,2),mfrow=c(1,2), mgp=c(1.8,0.8,0))

hist(log(d2/d1), probability=T, nclass=35,
     col="chartreuse4",main="BTC Daily returns",xlab='daily returns')

lines(x, dnorm(x, mean(log(d1/d2)), sd(log(d1/d2))), col="blue")


qqnorm(log(d1/d2),xlab='Normal quantile', ylab='Quantile of daily returns'
    ,
       col="brown")
qqline(log(d1/d2), col="blue")


library(tseries)
logreturn=log(d2/d1)
jarque.bera.test(logreturn)
ks.test(logreturn,"pnorm",mean=mean(logreturn),sd=sd(logreturn))
shapiro.test(logreturn)


library(MASS)
b=boxcox((logreturn+0.48)~1) # 定义函数类型和数据
I=which(b$y==max(b$y))
b$x[I]

newreturn=(logreturn+0.48)^ 1.79798
jarque.bera.test(newreturn)
ks.test(newreturn,"pnorm",mean=mean(newreturn),sd=sd(newreturn))
ps.options(horizontal=F, width=2, height=4, pointsize=12)
par(mar=c(5,4,4,2),mfrow=c(1,2), mgp=c(1.8,0.8,0))

```

```
hist(log(d2/d1), probability=T, nclass=35,
     col="chartreuse4",main="BTC Daily returns",xlab='daily returns')

lines(x, dnorm(x, mean(newreturn), sd(newreturn)), col="blue")


qqnorm(newreturn,xlab='Normal quantile', ylab='Quantile of daily returns',
       col="brown")
qqline(newreturn, col="blue")



ps.options(horizontal=F, width=4, height=6, pointsize=10)
N=100
d=diff(price[1:2689,1])
par(mar=c(3,3,1,1),mfrow=c(3,2), mgp=c(1.8,0.8,0))
acf(price[N:2689,1], lag=N, ci.type='ma', main='', ylab='ACF of prices')
acf(price[N:2689,1], type='partial', lag=N, main='', ylab='PACF of prices'
    )
acf(d[N:2688], lag=N, ci.type='ma', main='', ylab='ACF of differences')
acf(d[N:2688], type='partial', lag=N, main='', ylab='PACF of differences')
acf(logreturn[N:2688], lag=N, ci.type='ma', main='', ylab='ACF of
    logreturn')
acf(logreturn[N:2688], type='partial', lag=N, main='', ylab='PACF of
    logreturn')

adf.test(price[1:2689,1])
adf.test(logreturn)

fit1 <- stl(price[1:2689,1])
fit2 <- stl(logreturn)

armaSearch<-function(data){
  armacoef<-data.frame()
```

```
  for (p in 0:10){
    for (q in 0:10) {
      data.arma = arima(data, order = c(p, 1, q))
      armacoef<-rbind(armacoef,c(p,q,data.arma$aic))
    }
  }
  colnames(armacoef)<-c("p","q","AIC")
  pos<-which(armacoef$AIC==min(armacoef$AIC))
  cat("the min AIC=",armacoef$AIC[pos], ",p=",armacoef$p[pos],",q=",
      armacoef$q[pos])
  return(armacoef)
}

armaSearch(logreturn)

library(forecast)
auto.arima(logreturn,d=1,max.p = 10,
          max.q = 10, ic ="aic", stepwise = FALSE, trace = FALSE,
              allowdrift = TRUE, lambda = NULL)

traindata=logreturn[1:2646]
fit = arima(traindata, order = c(4, 1, 7))
summary(fit)


prediction <- forecast(fit)
checkresiduals(prediction)

truth=logreturn[2646:2688]
truth
p=forecast(fit, 43)
autoplot(forecast(fit, 43))


```

```r
x <- 1:43
y <- truth
mean<-p["mean"]
ci_l <- p["lower"]
ci_r <- p["upper"]
ci_l=as.data.frame(ci_l)
ci_r=as.data.frame(ci_r)
dat_plot <- data.frame(x, y, mean,ci_l,ci_r)

library(ggplot2)
plot=ggplot(dat_plot, aes(x = x)) +           # x轴在此处添加，目的为了置信区间与拟合线共享同一个x
  geom_ribbon(aes(ymin = ci_l[1:43,1], ymax = ci_r[1:43,1],fill = ""), alpha = 0.3) + # 添加置信区间
  geom_line(aes(y = y),color="chartreuse4") +
  geom_line(aes(y = mean),color="purple4")+
  geom_ribbon(aes(ymin = ci_l[1:43,2], ymax = ci_r[1:43,2],fill = ""), alpha = 0.2) # 标题居中
plot+ xlab("time") + ylab("log return")
```

7.1.3 N-BEATS

```

import collections

import numpy as np

import tensorflow.compat.v1 as tf
tf.disable_v2_behavior()

EAGER_EXECUTION = False # used for debugging.
if EAGER_EXECUTION:
    tf.enable_eager_execution()

```

```
13 # from tensorflow.keras.layers import MaxPooling2D,Conv2D,Input,Add,
       MaxPool2D,Flatten,AveragePooling2D,Dense,BatchNormalization,
       ZeroPadding2D,Activation,Concatenate,UpSampling2D
14 # from tensorflow.keras.models import Model
15
16 def mae(y_true, y_pred):
17     return tf.reduce_sum(tf.abs(y_true - y_pred))
18
19
20 def mse(y_true, y_pred):
21     return tf.reduce_sum(tf.square(y_true - y_pred))
22
23
24 def smape(y_true, y_pred):
25     return tf.reduce_mean(2 * tf.abs(y_true - y_pred) / (tf.abs(y_true) +
           tf.abs(y_pred)))
26
27
28 def mase(y_true, y_pred, m=12): # m = period.
29     return tf.reduce_mean(
30         tf.abs(y_true - y_pred) / tf.reduce_mean(y_true[m + 1:] - y_true[0:
               tf.shape(y_true)[0] - (m + 1)]))
31
32
33 def owa(y_true, y_pred, m=12):
34     return 0.5 * smape(y_true, y_pred) + 0.5 * mase(y_true, y_pred, m)
35
36
37 # https://machinelearningmastery.com/time-series-seasonality-with-python/
38
39
40 def linear_space(length, fwd_looking=True):
41     if fwd_looking:
```

```
42          t = tf.linspace(0.0, tf.cast(length, tf.float32), tf.cast(length,
                tf.int32))
43      else:
44          t = tf.linspace(-tf.cast(length, tf.float32), 0.0, tf.cast(length,
                tf.int32))
45      t = t / tf.cast(length, tf.float32) # normalise.
46      return t
47
48
49  def trend_model(thetas, length, is_forecast=True):
50      p = thetas.get_shape().as_list()[-1]
51      t = linear_space(length, fwd_looking=is_forecast)
52      T = tf.stack([t ** i for i in range(p)], axis=0)
53      return tf.matmul(thetas, T)
54
55
56  def seasonality_model(thetas, h, is_forecast=True):
57      p = thetas.get_shape().as_list()[-1]
58      t = linear_space(h, fwd_looking=is_forecast)
59      p1, p2 = (p // 2, p // 2) if p % 2 == 0 else (p // 2, p // 2 + 1)
60      s1 = tf.stack([tf.cos(2 * np.pi * i * t) for i in range(p1)], axis=0) #
            H/2-1
61      s2 = tf.stack([tf.sin(2 * np.pi * i * t) for i in range(p2)], axis=0)
62      S = tf.concat([s1, s2], axis=0)
63      return tf.matmul(thetas, S)
64
65
66  def block(x, theta_transforms, units=256, nb_thetas=64, block_type='
        generic', backcast_length=10, forecast_length=5):
67      print(block_type)
68      for _ in range(4):
69          x = tf.layers.Dense(units, activation='relu')(x)
70
71      # 3.1 Basic block. Phi_theta^f : R^{dim(x)} -> theta_f.
```

```
72      # 3.1 Basic block. Phi_theta^b : R^{dim(x)} -> theta_b.
73      if block_type == 'generic':
74          # no constraint for generic arch.
75          theta_b = tf.layers.Dense(nb_thetas, activation='linear')(x)
76          theta_f = tf.layers.Dense(nb_thetas, activation='linear')(x)
77          backcast = tf.layers.Dense(backcast_length, activation='linear')(
                theta_b) # generic. 3.3.
78          forecast = tf.layers.Dense(forecast_length, activation='linear')(
                theta_f) # generic. 3.3.
79      elif block_type == 'trend':
80          theta_f = theta_b = theta_transforms['trend'](x)
81          backcast = trend_model(theta_b, backcast_length, is_forecast=False)
                 # 3.3 g_f = g_b
82          forecast = trend_model(theta_f, forecast_length, is_forecast=True)
83      elif block_type == 'seasonality':
84          # length(theta) is pre-defined here.
85          theta_f = theta_b = theta_transforms['seasonality'](x)
86          backcast = seasonality_model(theta_b, backcast_length, is_forecast=
                False) # 3.3 g_f = g_b
87          forecast = seasonality_model(theta_f, forecast_length, is_forecast=
                True)
88      else:
89          raise Exception('Unknown block_type.')
90
91      return backcast, forecast
92
93
94  def net(x, units=256, nb_stacks=2, nb_thetas=10, nb_blocks=3,
95          block_types=['seasonality'] * 2, backcast_length=10, forecast_
                length=5):
96      assert len(block_types) == nb_stacks
97      metadata = {
98          'backcasts': [],
99          'residuals': [],
```

```
        'forecasts': []
    }

    theta_transforms = {
        'trend': tf.layers.Dense(nb_thetas, activation='linear'),
        # should be forecast_length but isn't it an error of the paper?
        'seasonality': tf.layers.Dense(backcast_length, activation='linear'
            )
    }
    forecasts = []
    for j in range(nb_stacks):
        block_connections = []
        for i in range(nb_blocks):
            b, f = block(x, theta_transforms, units, nb_thetas, block_types[
                j], backcast_length, forecast_length)
            x = x - b
            block_connections.append(f)
            metadata['forecasts'].append(f)
            metadata['backcasts'].append(b)
            metadata['residuals'].append(x)
        y = tf.add_n(block_connections)
        forecasts.append(y)
    y = tf.add_n(forecasts)
    return x, y, metadata


def get_data(length, test_starts_at, signal_type='generic', random=False):
    if random:
        offset = np.random.standard_normal() * 0.1
    else:
        offset = 1
    if signal_type in ['trend', 'generic']:
        x = np.arange(0, 1, 1 / length) + offset
    elif signal_type == 'seasonality':
```

```
132         random_period_coefficient = np.random.randint(low=6, high=10)
133         random_period_coefficient_2 = np.random.randint(low=2, high=6)
134         x = np.cos(random_period_coefficient * np.pi * np.arange(0, 1, 1 /
                length))
135         x += 3 * np.sign(offset) * np.arange(0, 1, 1 / length) + offset
136         x += np.cos(random_period_coefficient_2 * np.pi * np.arange(0, 1, 1
                / length))
137         # import matplotlib.pyplot as plt
138         # plt.plot(x)
139         # plt.show()
140     else:
141         raise Exception('Unknown signal type.')
142     x /= np.max(np.abs(x))
143     x = np.expand_dims(x, axis=0)
144     y = x[:, test_starts_at:]
145     x = x[:, :test_starts_at]
146     return x, y
147
148 def get_data_bitcoin(length, test_starts_at):
149     import pandas as pd
150     series = pd.DataFrame()
151     # df = pd.read_csv("BTC.csv")
152     # series['Date'] = df['Date'].tolist()
153     # series['Price'] = df['Price'].tolist()
154     # series['Volume'] = df['Volume'].tolist()
155     # x0= np.array(series['Price'])
156     # x=x0[2688 -length:2688]
157     df = pd.read_csv("bitcoin2021.5toNOW.csv")
158     series['Close'] = df['Close'].tolist()
159     x0= np.array(series['Close'])
160     x=x0[13173 -length:13173]
161     x /= np.max(np.abs(x))
162     x = np.expand_dims(x, axis=0)
163     y = x[:, test_starts_at:]
```

```
164     x = x[:, :test_starts_at]
165
166     return x, y
167
168 def train():
169     forecast_length = 100
170     backcast_length = 3 * forecast_length # 4H in [2H, 7H].
171
172     signal_type = 'seasonality'
173     block_types = ['trend', 'seasonality']
174
175     sess = tf.compat.v1.Session()
176
177     if EAGER_EXECUTION:
178         x, y = get_data(length=backcast_length + forecast_length,
179                         test_starts_at=backcast_length,
180                         signal_type=signal_type)
181         x_inputs = tf.constant(dtype=tf.float32, value=x)
182         y_true = tf.constant(dtype=tf.float32, value=y)
183     else:
184         x_inputs = tf.placeholder(dtype=tf.float32, shape=(None, backcast_
              length))
185         y_true = tf.placeholder(dtype=tf.float32, shape=(None, forecast_
              length))
186     res, output, metadata = net(x_inputs,
187                                 units=256,
188                                 nb_stacks=len(block_types),
189                                 nb_thetas=2,
190                                 nb_blocks=3,
191                                 block_types=block_types,
192                                 backcast_length=backcast_length,
193                                 forecast_length=forecast_length)
194
195     if EAGER_EXECUTION:
```

```
196         exit(1) # stop here. eager used for debugging.
197
198     loss = mae(y_true, output)
199     train_op = tf.compat.v1.train.AdamOptimizer(learning_rate=1e-4).
            minimize(loss)
200
201     sess.run(tf.global_variables_initializer())
202     running_loss = collections.deque(maxlen=100)
203     for step in range(10001):
204         x, y = get_data_bitcoin(length=backcast_length + forecast_length,
205                                 test_starts_at=backcast_length)
206         feed_dict = {x_inputs: x, y_true: y}
207         current_loss, _ = sess.run([loss, train_op], feed_dict)
208         running_loss.append(current_loss)
209         if step % 100 == 0:
210             if step % 2000 == 0:
211                 predictions, residual = sess.run([output, res], feed_dict)
212                 import matplotlib.pyplot as plt
213                 plt.grid(True)
214                 plt.figure(figsize=(12, 8))
215                 x_y = np.concatenate([x, y], axis=-1).flatten()
216                 plt.plot(list(range(backcast_length)), x.flatten(), color='
                        royalblue',lw=1,ms=3)
217                 plt.plot(list(range(len(x_y) - forecast_length, len(x_y))),
                        y.flatten(), color='tomato',lw=1,ms=3)
218                 plt.plot(list(range(len(x_y) - forecast_length, len(x_y))),
                        predictions.flatten(), color='cyan',lw=1,ms=3)
219                 plt.scatter(range(len(x_y)), x_y.flatten(), color=['
                        royalblue'] * backcast_length + ['tomato'] * forecast_
                        length)
220                 plt.scatter(list(range(len(x_y) - forecast_length, len(x_y))
                        ), predictions.flatten(),
221                             color=['cyan'] * forecast_length)
```

```
222                 plt.legend(['backcast', 'forecast', 'predictions of forecast
                        '])
223
224                 plt.savefig('foreast'+str(step)+'.png')
225                 plt.show()
226
227                 plt.plot(residual.flatten())
228                 plt.title('Residual'+str(step)+'.png')
229                 plt.show()
230
231             print(step, running_loss[-1], np.mean(running_loss))
232
233
234 if __name__ == '__main__':
235     train()
```