

2020-2021 学年第二学期数学实验  
实验报告

姓名: \_\_\_\_\_ 学号: \_\_\_\_\_ 截止日期: 2021.6.25

注. 本报告分为两部分内容, 即

- 利用神经网络求解偏微分方程 (三选一)
- 利用 *LSTM* 预测金融市场 (必选)

实验内容应包括:

- 数据处理
- 网络架构
- 网络参数 (包括网络层数、每层的神经元个数等) 对训练的影响
- 尽可能给出可视化结果, 包括相关结果的图像
- 尽可能把报告内容写得丰富一些, 这可以自由发挥
- 建议大家编译 *latex* 文件时, 安装 *Ctex+TexStudio*。
- 最终的报告中, 请把无关部分注释掉。

## 1 利用神经网络求解偏微分方程

### 1.1 Schrödinger 方程

利用神经网络 PINN 求解一维非线性 Schrödinger 方程

$$ih_t + 0.5h_{xx} + |h|^2h = 0, \quad x \in [-5, 5], t \in (0, \pi/2], \quad (1a)$$

$$h(0, x) = 2 \operatorname{sech}(x), \quad x \in [-5, 5] \quad (1b)$$

$$h(t, -5) = h(t, 5), \quad t \in (0, \pi/2] \quad (1c)$$

$$h_x(t, -5) = h_x(t, 5), \quad t \in (0, \pi/2] \quad (1d)$$

其中  $i$  为虚数符号,  $h(x, t) = u(x, t) + iv(x, t)$  为复值函数, (1c)和(1d)为周期边界条件,  $\operatorname{sech}(x)$  为双曲正弦函数, 即

$$\operatorname{sech} x = \frac{2}{e^x + e^{-x}}.$$

(1)可改写为  $u$  和  $v$  的偏微分方程组

$$u_t + 0.5v_{xx} + (u^2 + v^2)v = 0, \quad x \in [-5, 5], t \in (0, \pi/2], \quad (2a)$$

$$v_t - 0.5u_{xx} - (u^2 + v^2)u = 0, \quad x \in [-5, 5], t \in (0, \pi/2], \quad (2b)$$

$$u(0, x) = 2 \operatorname{sech}(x), \quad x \in [-5, 5] \quad (2c)$$

$$v(0, x) = 0, \quad x \in [-5, 5] \quad (2d)$$

$$u(t, -5) = u(t, 5), \quad t \in (0, \pi/2] \quad (2e)$$

$$v(t, -5) = v(t, 5), \quad t \in (0, \pi/2] \quad (2f)$$

$$u_x(t, -5) = u_x(t, 5), \quad t \in (0, \pi/2] \quad (2g)$$

$$v_x(t, -5) = v_x(t, 5), \quad t \in (0, \pi/2] \quad (2h)$$

注. 薛定谔方程是一个经典的场方程, 用于研究量子力学系统, 包括非线性波在光纤和波导中的传播、玻色-爱因斯坦凝聚和等离子体波。在光学中, 非线性项来自于给定材料的与强度有关的折射率。类似地, 玻色-爱因斯坦凝聚的非线性项是相互作用的多体系统的平均场相互作用的结果。

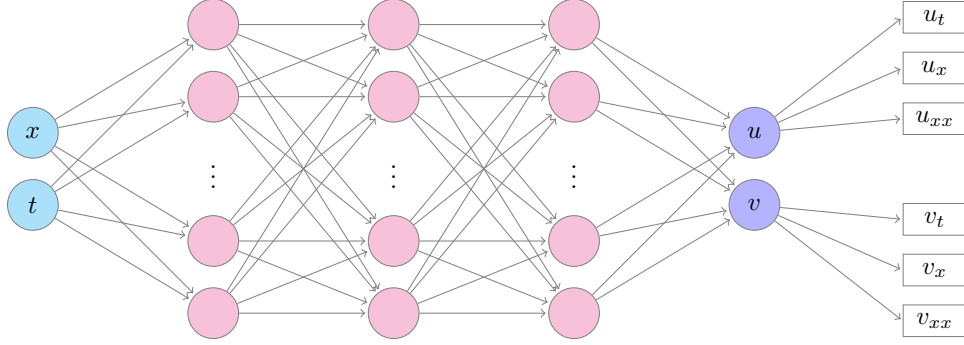


图 1: 求解薛定谔方程的神经网络架构

**损失函数** 损失函数可表示为

$$\mathcal{L}(\Theta) = \mathcal{L}_{\text{PDE}}(\Theta) + \lambda_1 \mathcal{L}_{\text{IC}}(\Theta) + \lambda_2 \mathcal{L}_{\text{BC}}(\Theta)$$

其中

$$\mathcal{L}_{\text{PDE}}(\Theta) = \frac{1}{N_{\text{PDE}}} \sum_{n=1}^{N_{\text{PDE}}} \left\{ \left[ \hat{u}_t(t_n, x_n) + 0.5\hat{v}_{xx}(t_n, x_n) + (\hat{u}(t_n, x_n)^2 + \hat{v}(t_n, x_n)^2) \hat{v}(t_n, x_n) \right]^2 \right. \\ \left. + \left[ \hat{v}_t(t_n, x_n) - 0.5\hat{u}_{xx}(t_n, x_n) - (\hat{u}(t_n, x_n)^2 + \hat{v}(t_n, x_n)^2) \hat{u}(t_n, x_n) \right]^2 \right\},$$

$$\mathcal{L}_{\text{IC}}(\Theta) = \frac{1}{N_{\text{IC}}} \sum_{n=1}^{N_{\text{IC}}} \left\{ [\hat{u}(0, x_n) - 2 \operatorname{sech}(x_n)]^2 + \hat{v}(0, x_n)^2 \right\}$$

$$\mathcal{L}_{\text{BC}}(\Theta) = \frac{1}{N_{\text{BC}}} \sum_{n=1}^{N_{\text{BC}}} \left\{ [\hat{u}(t_n, -5) - \hat{u}(t_n, 5)]^2 + [\hat{v}(t_n, -5) - \hat{v}(t_n, 5)]^2 + \right. \\ \left. [\hat{u}_x(t_n, -5) - \hat{u}_x(t_n, 5)]^2 + [\hat{v}_x(t_n, -5) - \hat{v}_x(t_n, 5)]^2 \right\}$$

为方便, 这里用  $\hat{u}$  和  $\hat{v}$  表示网络的输出, 即

$$\hat{u}(t, x) := u(t, x; \Theta), \quad \hat{v}(t, x) := v(t, x; \Theta).$$

## 1.2 网络结构搭建

### 1.2.1 导入所需要的模块

- torch 深度学习框架
- torch.optim 优化计算
- torch.nn 函数计算
- torch.optim.lr\_scheduler 调参操作
- os 处理文件和目录
- numpy 处理多维数组
- shutil 文件（夹）读写操作
- matplotlib.pyplot 绘图操作

。

### 1.2.2 构建函数

其中包括:

- 编写 problem 函数。

规定薛定谔方程定义域:  $x$  轴方向范围是  $[-5,5]$ ,  $y$  方向的范围是  $(0, \pi/2]$ 。根据方程解析式:  $ih_t + 0.5h_{xx} + |h|^2h = 0$  和  $h(0, x) = 2\text{sech}(x)$  来创建方程内函数 (其中  $\text{sech}$  为双曲正弦函数), 以及函数真值, 主要使用 numpy 中的数学函数来生成薛定谔方程表达式, 并预备将测试集训练集存储在一个一维数组中。

- 调用函数示例。

测试 problem 的正确性可用性, 给出一个 torch 中的  $5 \times 2$  数组, 调用 problem 中的 ic 函数看生成结果。

### 1.2.3 构建数据集

包括训练集的生成和测试集的生成。我们主要用上课介绍的均匀采样和 lhs 采样方法来生成我们的数据集。

- 均匀采样

先确定范围,  $x$  轴方向范围是  $[-5,5]$ ,  $y$  方向的范围是  $(0, \pi/2]$ , 将范围中的点用 meshgrid 函数进行均匀分割, 其上的点就是生成点的来源, 然后再根据薛定谔方程的边界条件  $h(t, -5) = h(t, 5)$  和  $h_x(t, -5) = h_x(t, 5)$  要写一个 bool 变量均匀采样确定生成样本是在范围中的。采样结果如下图所示:

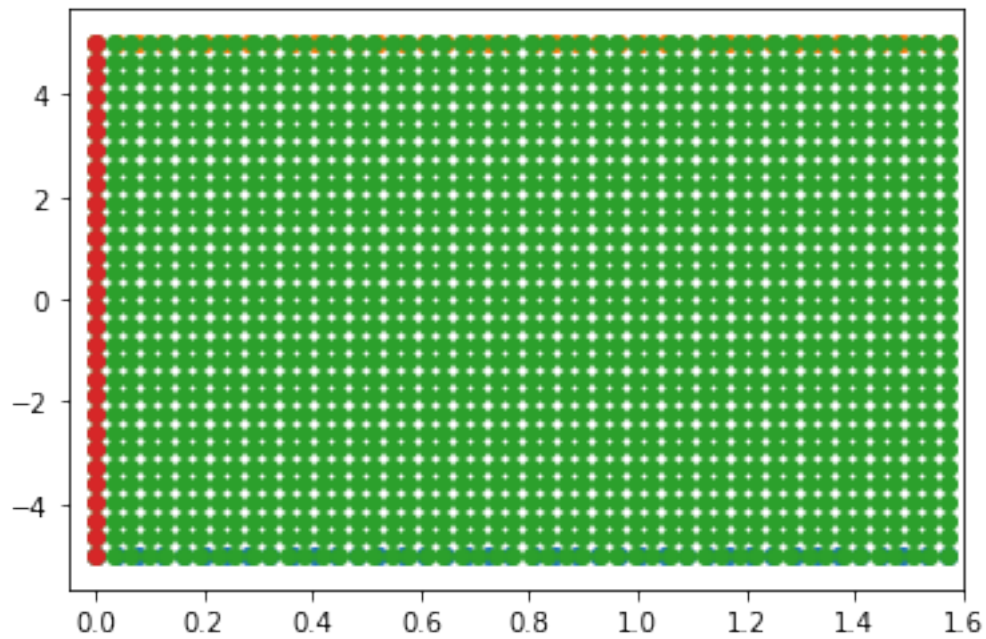


图 2: 均匀采样

- Lhs 抽样

lhs 抽样是抽样技术的最新进展，它被设计成通过较少迭代次数的抽样，准确地重建输入分布。拉丁超立方体抽样的关键是对输入概率分布进行分层。分层在累积概率尺度把累积曲线分成相等的区间。然后，从输入分布的每个区间或“分层”中随机抽取样本。我们用自带的 lhs 函数进行各个方向的抽样，并编写 lhs\_sampling 函数存贮生成的样本边界条件和范围。采样结果如图所示：

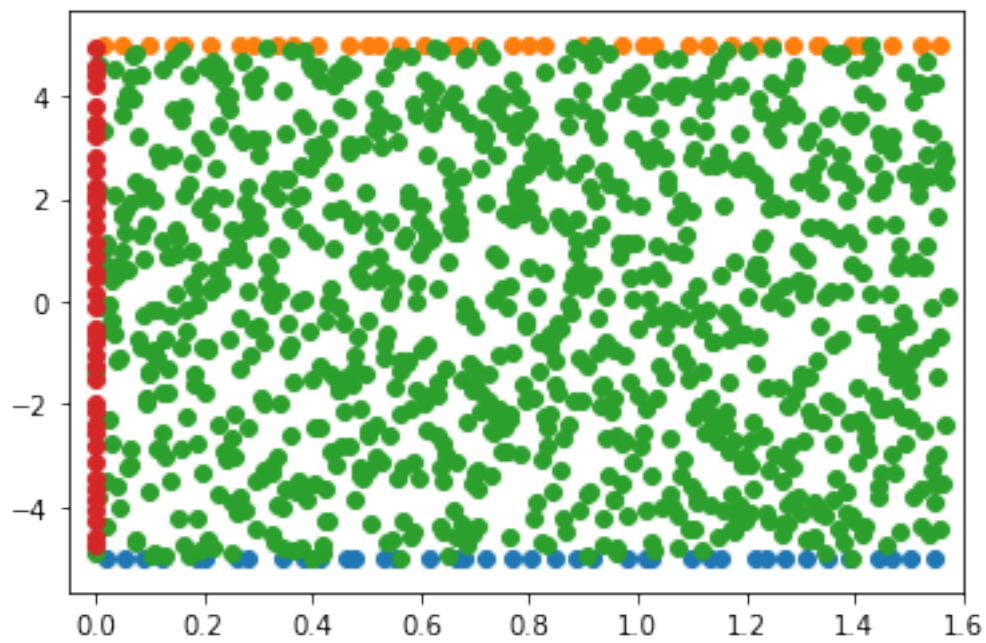


图 3: lhs 采样

采样结束后，我们分别编写了生成训练集和验证集的代码，为了代码的可复用性，我们将这两个过程分别封装为类。

对于训练集，我们编写了如下函数来完成构造过程：

- `__init__(self, *args, **kwargs)`  
确定定义域和采样方式。
- `__call__(self, verbose=None)`  
调用采样函数，并将采样点进行封装和转化数据类型。
- `_uniform_sample(self, nx, nt, n_bc, n_ic)`  
均匀采样
- `_lhs_sample(self, n, n_bc, n_ic)`  
lhs 采样

对于验证集，我们也同样将其构造过程封装为一个类：

- `__init__(self, *args, **kwargs)`  
初始化，定义定义域和采样方式。
- `__repr__(self)`
- `__call__(self, plot=False, verbose=None)`  
调用采样函数，并将采样点进行封装和转化数据类型。
- `_uniform_sample(self, n_x, t)`  
均与采样

#### 1.2.4 激活函数

可选择 `tanh`, `ReLU`, `LeakyReLU`, `sigmoid`, `softplus`，以下我们给出这些函数的解析式以及图像。

- `tanh`

$$\tanh(x) = \frac{\sinh(x)}{\cosh(x)} = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

- `ReLU`

$$f(x) = \max(0, w^T x + b)$$

- `LeakyReLU`

$$f(x) = \max(0, w^T x + b) + leak * \min(0, w^T x + b)$$

leak 为较小常数

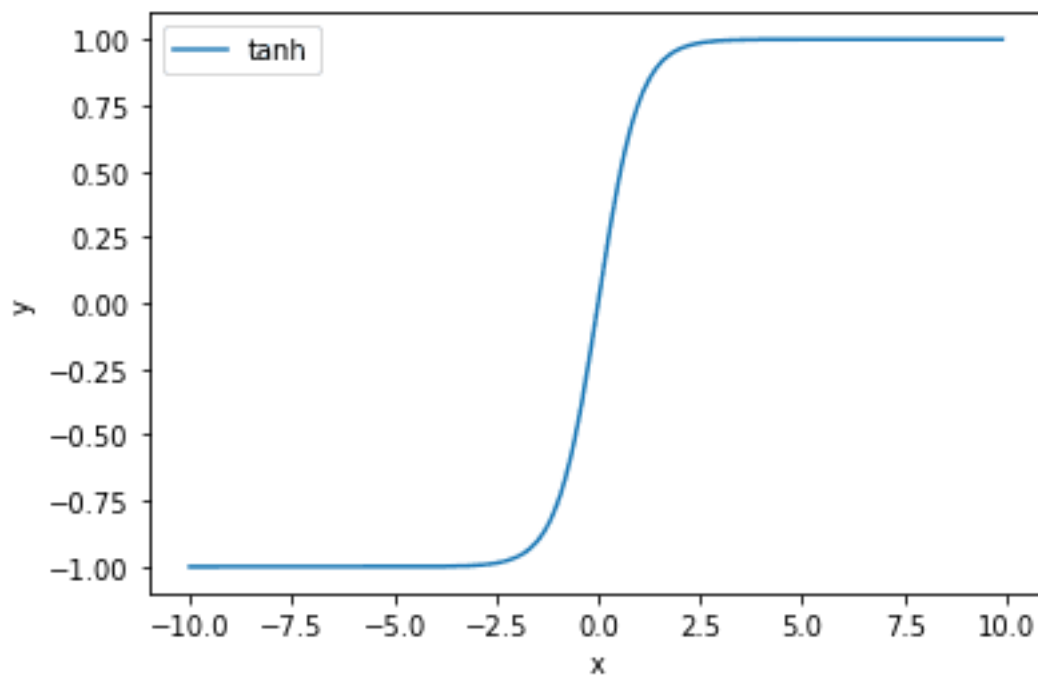


图 4: tanh 函数

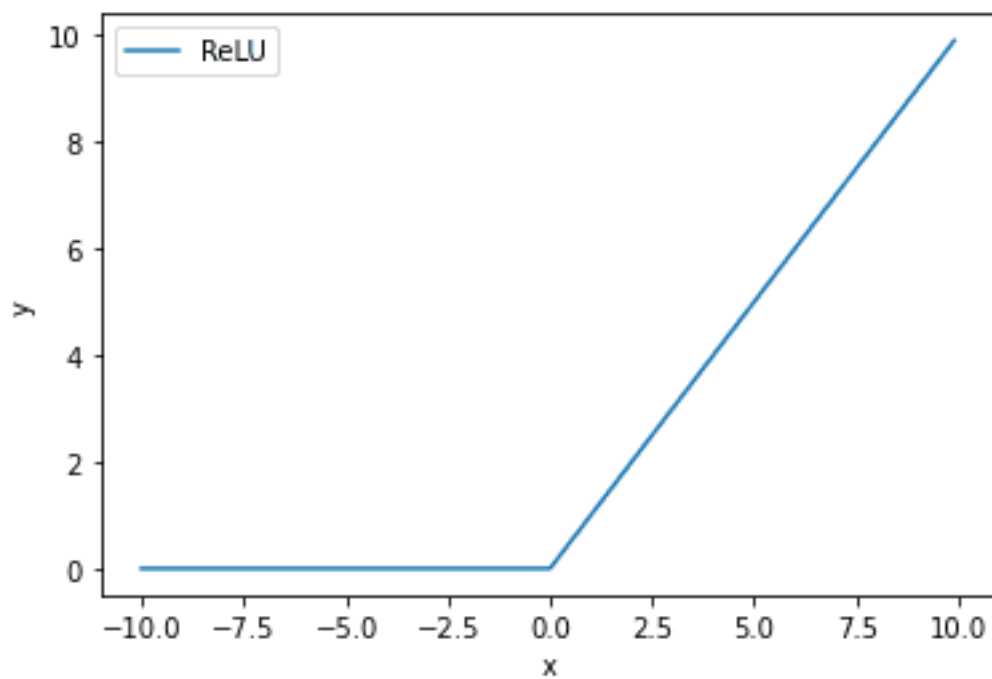


图 5: ReLU 函数

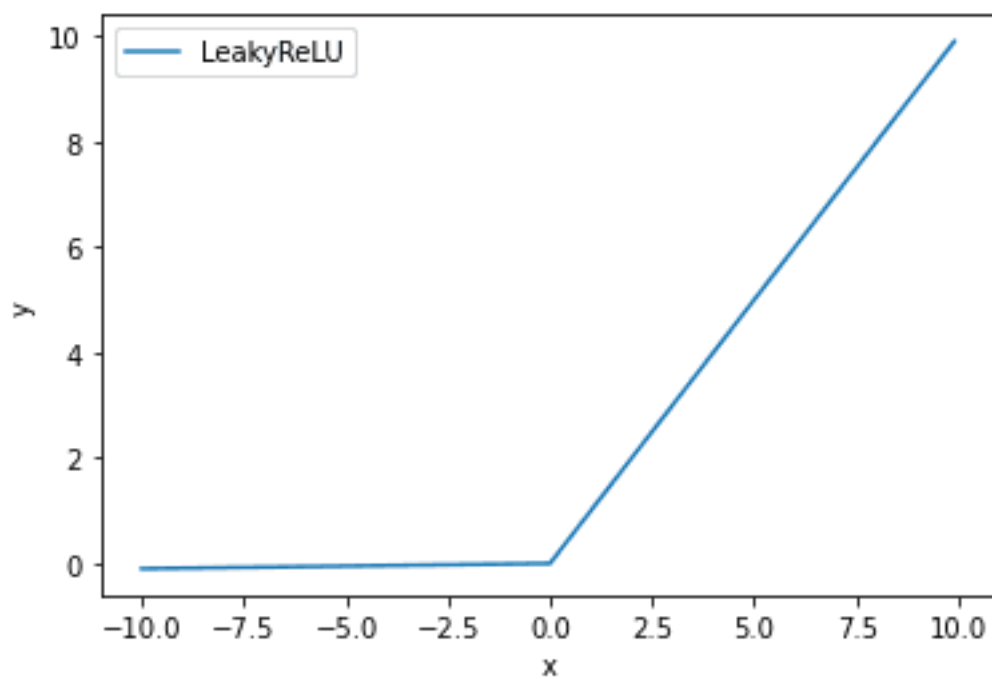


图 6: LeakyReLU 函数

- softplus

$$f(x) = \log(1 + e^x)$$

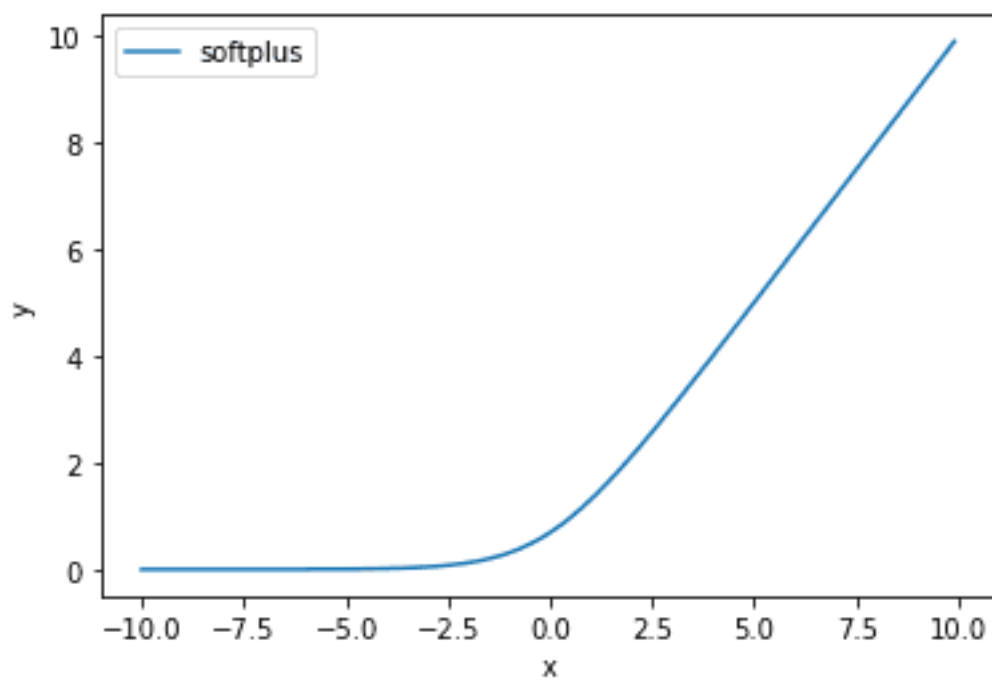


图 7: softplus 函数

- sigmoid

$$f(x) = \frac{1}{1 + e^{-x}}$$

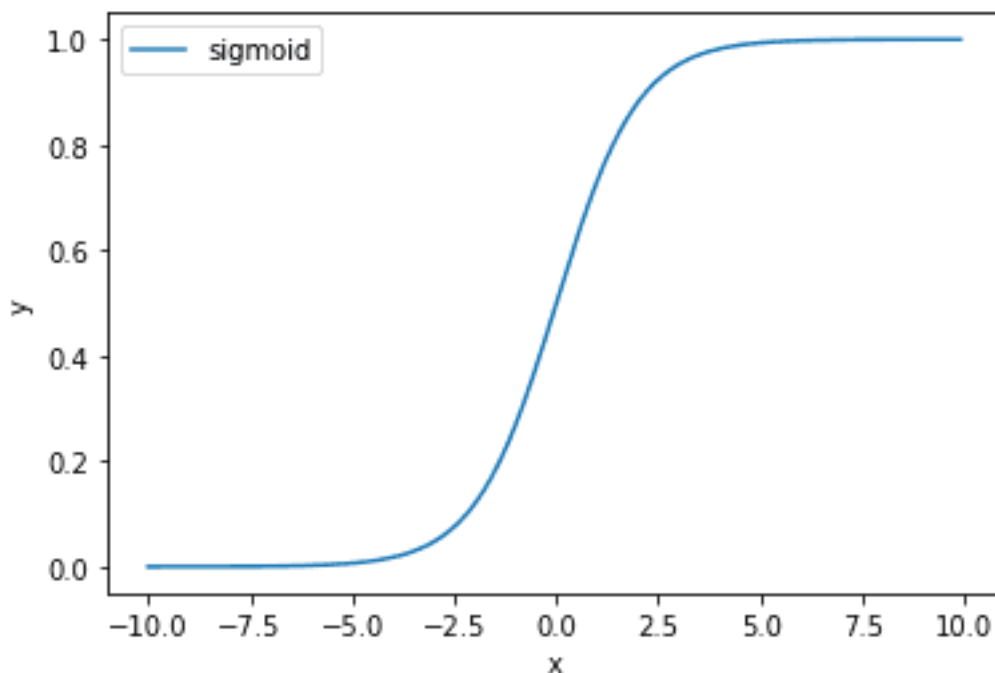


图 8: sigmoid 函数

### 1.2.5 计算网络输入，输出的梯度

计算与输入相关的输出的导数，输入的是一个 (N,D) 维数的 tensors 数组，输出是一个 (N,1) 维数的 tensors 数组。用 torch 中的 autograd 自动求导函数包里的函数 grad 来求解。

### 1.2.6 搭建神经网络

其中包括 DNN: Deep Neural Network 和 Residual DNN: Residual Deep Neural Network 和 PINN: Physics Informed Neural Networks 和 Residual PINN: Residual Physics Informed Neural Networks

- DNN

神经网络是基于感知机的扩展，而 DNN 可以理解为有很多隐藏层的神经网络，有时也叫做多层感知机 (Multi-Layer perceptron,MLP)。

从 DNN 按不同层的位置划分，DNN 内部的神经网络层可以分为三类，输入层，隐藏层和输出层，一般来说第一层是输入层，最后一层是输出层，而中间的层数都是隐藏层。而在我们的实验中，DNN 由 pytorch 的 nn.Module 继承而来，并编写了 init\_weight 函数。

- Residual DNN

ResDNN 是 DNN 的变体，其不同之处主要在于 ResDNN 设置了 ResBlock 模块。

- PINN

由 DNN 继承而来，主要定义了函数的梯度计算和确定激活函数为 tanh。



- Residual PINN

由 ResDNN 继承而来，与 PINN 类似，也是定义了函数的梯度计算即向前传播过程。这里，我们依旧选择激活函数为  $\tanh$ 。

### 1.2.7 训练过程

我们将训练过程封装为一个类，这个类里面主要定义了以下函数来完成我们的训练过程：

- `__init__(self, args)`

对一些网络参数进行定义，如优化器，损失函数，学习率以及获取训练集数据。

- `_model_path(self)`

生成路径，用于存储训练过程中产生的参数。

- `train(self)`

按预先设定好的轮数进行计算，存储从其他函数传来的 `loss` 值和模型参数。

- `train_Adam(self)`

利用 Adam 优化器进行前向传播等训练工作。

- `infos_Adam(self, epoch, loss, loss1, loss2, loss3)`

展现 Adam 优化器产生的结果对应的 `loss` 值。

- `train_LBFGS(self)`

利用 LBFGS 优化器进行前向传播等训练工作。

- `infos_LBFGS(self, epoch, loss, loss1, loss2, loss3)`

汇总并打印由 LBFGS 产生的结果对应的 `loss` 值。

- `validate(self, epoch)`

汇总所有 `loss` 值。

### 1.2.8 验证过程

同样地，我们将验证过程也封装为一个类，这里面定义了如下函数：

- `__init__(self, args)`

对类进行定义，给出训练过程中存储参数文件的路径并读取数据。

- `predict(self,t)`

利用验证集以及训练好的模型得到预测值，并通过作图将预测值与真实值进行对比。由于 Schrödinger 方程的解是多元函数，所以在画图时需要限定  $t$  或  $x$ 。这里我们选择固定  $t$ ，作  $h$  关于  $x$  的图像。

- `pred_result(self,t)`

固定  $t$ ，得出验证集上的预测值。

### 1.2.9 特殊条件处理

我们从方程的定义出发，分别从实数和虚数两个角度考虑改写方程，同时也将初值条件加以替换：

$$\begin{aligned}h &= u + iv \\ -v_t + 0.5u_x x + (u^2 + v^2)u + iu_t + i(u^2 + v^2) &= 0\end{aligned}$$

实数部分与虚数部分分别为 0：

$$\begin{aligned}-v_t + 0.5u_x x + (u^2 + v^2)u &= 0 \\ u_t + u^2 + v^2 &= 0\end{aligned}$$

在这里将初值条件设置为

$$|h(0, x)| = 2\operatorname{sech}(x)$$

## 1.3 PiNN 训练结果

网络参数选取为：

dim\_hidden=50, hidden\_layers=4

训练参数为：

epochs\_Adam=5000, epochs\_LBFGS=200

初始学习率和学习率改变速率为：

lr=0.001, step\_size=2000, gamma=0.7

训练用采样点：使用 Uniform Sampling 方法采样 20000 个点，其中 x 方向采样 200 个点，t 方向采样 100 个时刻，边界条件和初值条件使用 Uniform Sampling 方法采样 400 个点。

检验用采样点：使用 Uniform Sampling 方法采样 10000 个点，边界条件和初值条件使用 Uniform Sampling 方法采样 400 个点。下面图片是训练的损失函数随训练 Epoch 的变化过程：

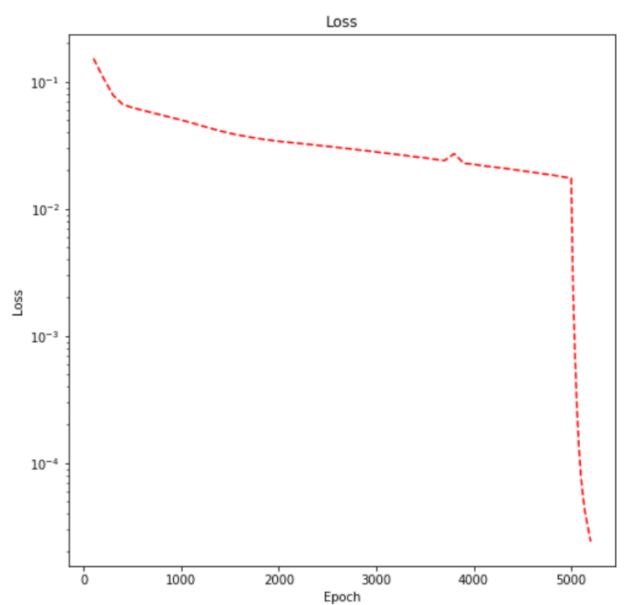


图 9: Loss 随 Epoch 增长之间的关系

测试用采样点: 使用 Uniform Sampling 方法采样 256 个点。这里我们小组参照文章中的例子选取了  $t=0.59, 0.79, 0.98$  三个时刻来展示结果测试结果显示如下:

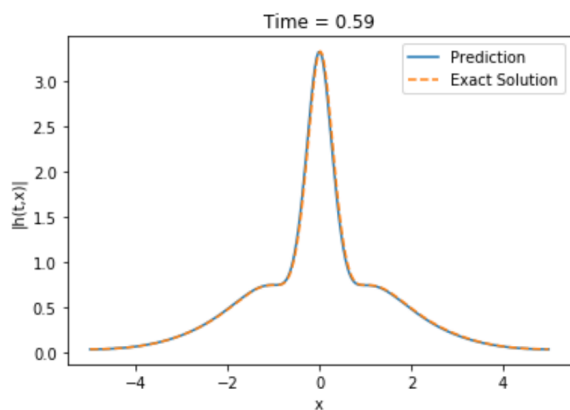


图 10:  $t=0.59$

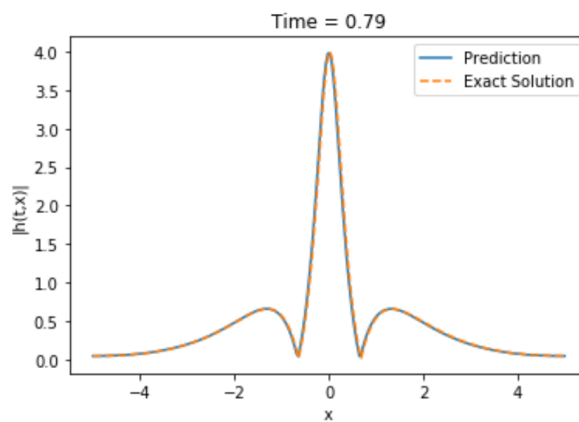


图 11:  $t=0.79$

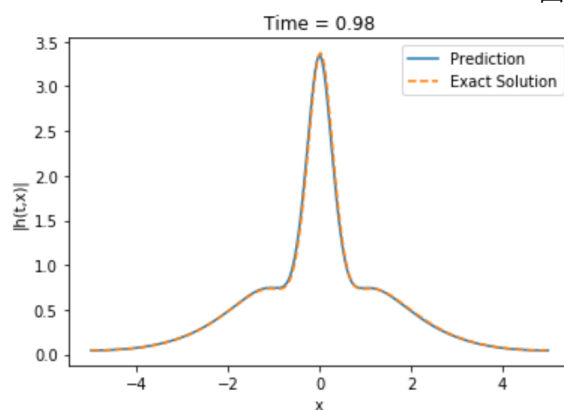


图 12:  $t=0.98$

由以上三幅图不难看出，我们的模型拟合结果很好，预测值与真实值基本吻合。

## 1.4 ResPiNN 网络训练结果

网络参数选取为:

`dim_hidden=50, hidden_layers=4`

训练参数为:

`epochs_Adam=5000, epochs_LBFGS=200`

初始学习率和学习率改变速率为:

`lr=0.001, step_size=2000, gamma=0.7`

训练用采样点: 使用 Uniform Sampling 方法采样 20000 个点, 其中  $x$  方向采样 200 个点,  $t$  方向采样 100 个时刻, 边界条件和初值条件使用 Uniform Sampling 方法采样 400 个点。

检验用采样点: 使用 Uniform Sampling 方法采样 10000 个点, 边界条件和初值条件使用 Uniform Sampling 方法采样 400 个点。下面图片是训练的损失函数随训练 Epoch 的变化过程:

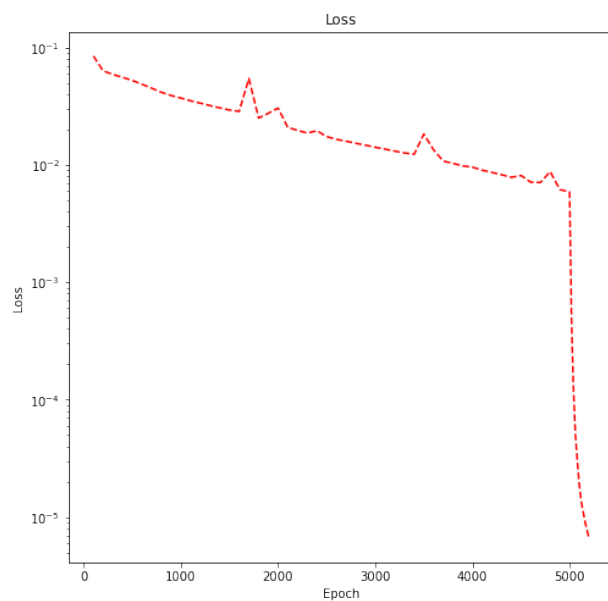


图 13: Loss 随 Epoch 增长之间的关系

测试用采样点: 使用 Uniform Sampling 方法采样 256 个点。类似地我们选择  $t=0.59, 0.79, 0.98$  三个时刻来展示结果测试结果显示如下:

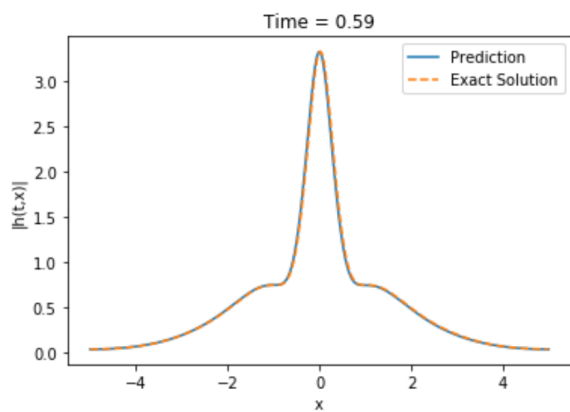


图 14:  $t=0.59$

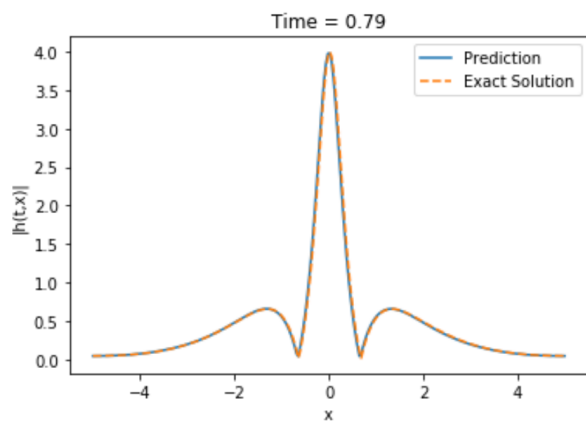


图 15:  $t=0.79$

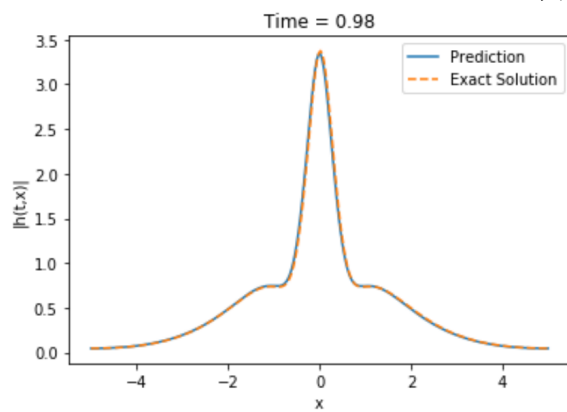


图 16:  $t=0.98$

从以上图线可以看出，预测值与真实值相差极小，这说明 ResPINN 网络表现也很优秀。

## 1.5 两种网络的比较

通过比较结果可以知道两种网络预测得到的解基本一样：不仅损失函数变化差不多，图像也是类似的。

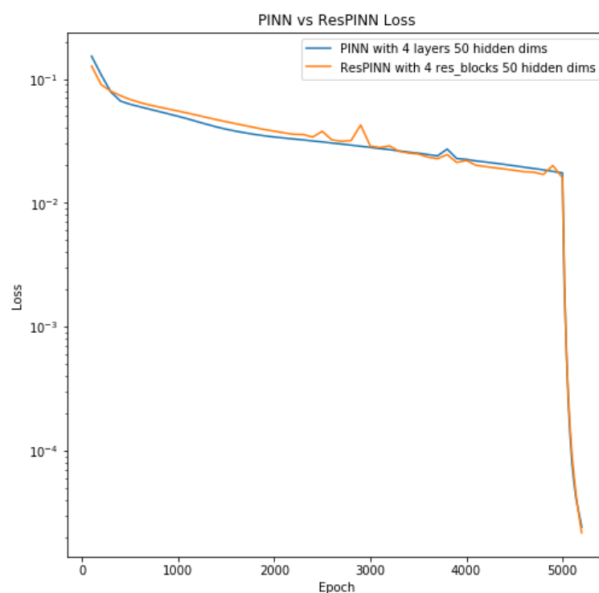


图 17: Loss 比较图

接下来选取一个时间  $t=0.98$  来分析方程结果，并与精确解作对比：上面两个图像即是我们组的最终结果。

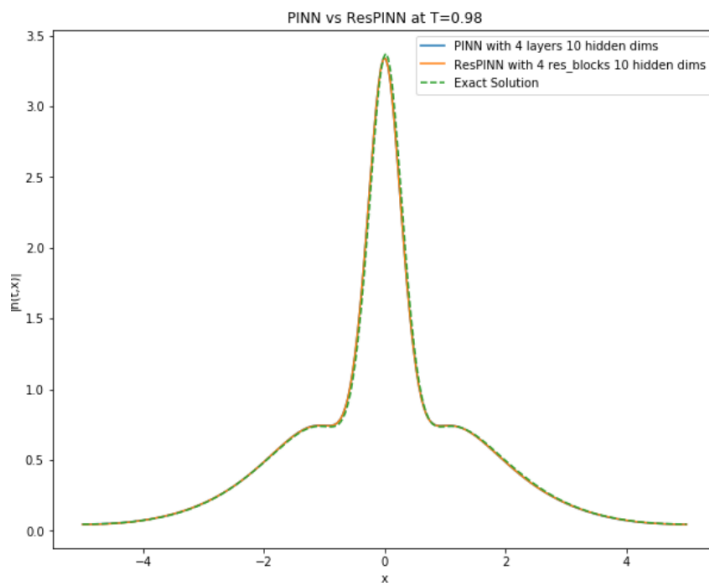


图 18: 解比较图

从这两幅图我们可以清晰地看出，在轮数较少时 PINN 网络在训练集上的效果略好于 ResPINN，但是当轮数达到一定程度（算法收敛）时，两种网络相差不大。

## 1.6 比较不同隐藏层数对训练结果的影响

此处以 ResPiNN 网络框架作为基础网络来比较；dim\_hidden 固定为 50，并对比 res\_blocks=2, 4, 8 的 loss 曲线图和结果。

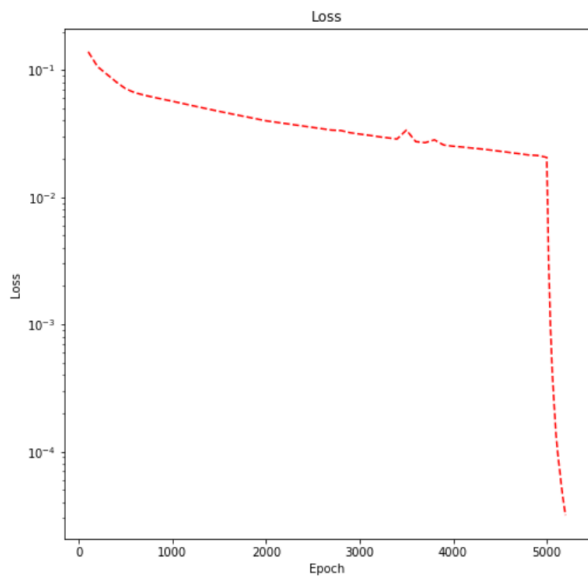


图 19:  $t=0.59$

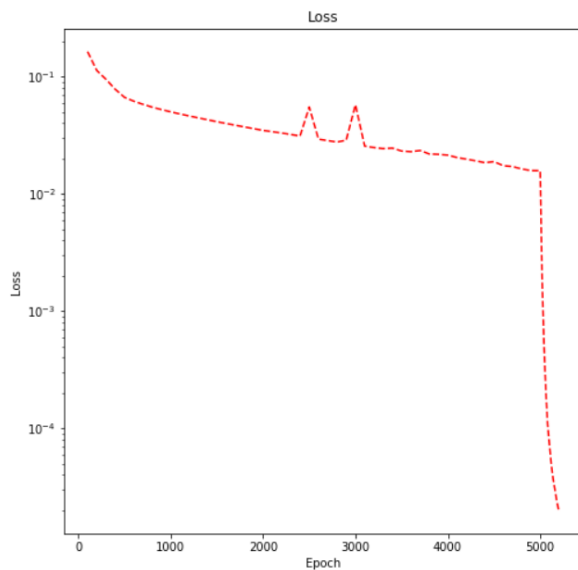


图 20:  $t=0.79$

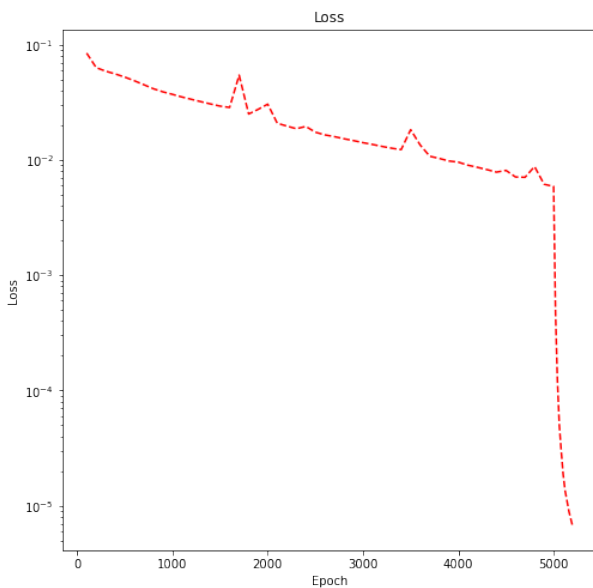


图 21:  $t=0.98$

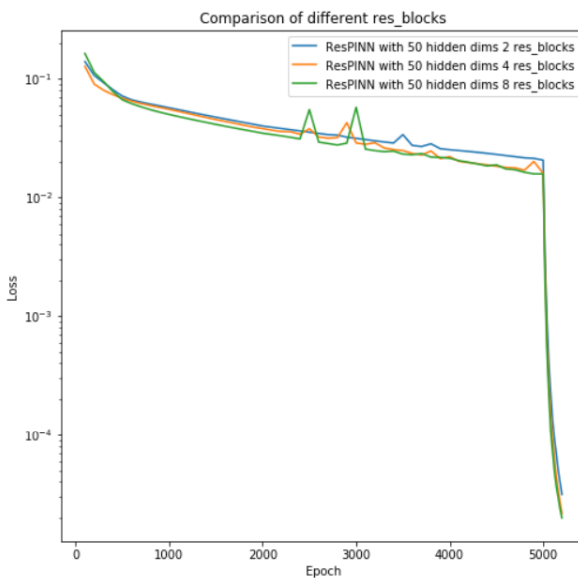


图 22:  $t=0.98$

最后将已知的隐藏层是 4 的情况，将三条曲线放一起来做对比，由图像可以看到：这三条 loss 曲线的下降都比较稳定，发现改变不同的 `res_blocks` 对网络结果的影响并不大。并且为了追求效率，`res_blocks` 选为 2 就已经足够解决我们的问题。

## 1.7 比较不同隐藏层神经元数目对结果的影响

在这里，我们同时比较了 PiNN 网络和 ResPiNN 网络，并分别做出结果分析。



### 1.7.1 PiNN

我们选取  $\text{dim\_hidden}$  等于 20, 35, 50 这三种情况, 对模型进行训练, 得到的 loss 的变化以及三种情况的对比分别如下四幅图:

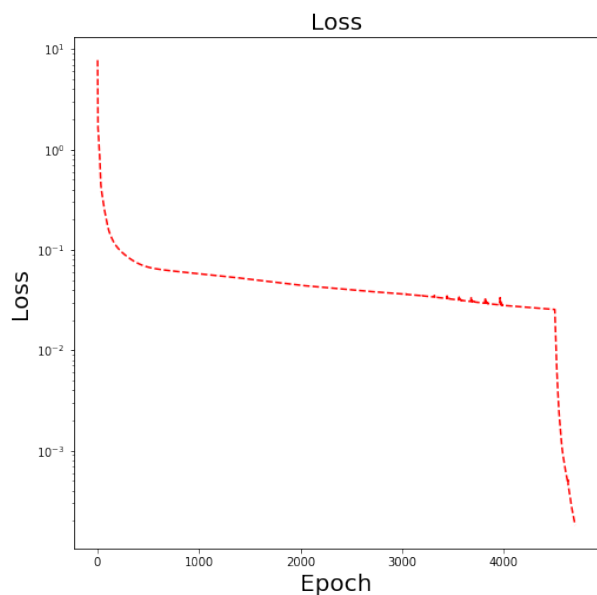


图 23:  $\text{dim\_hidden}=20$

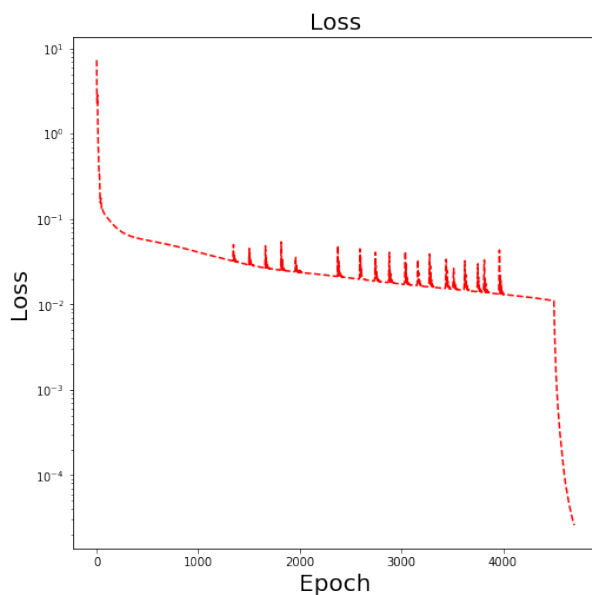


图 24:  $\text{dim\_hidden}=35$

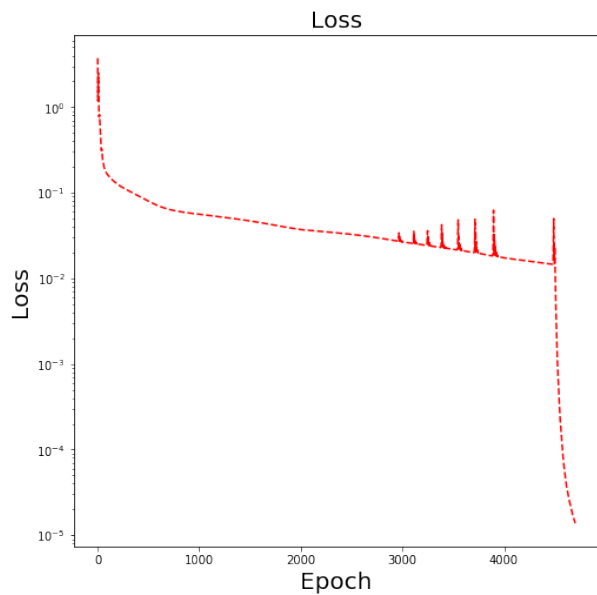


图 25:  $\text{dim\_hidden}=50$

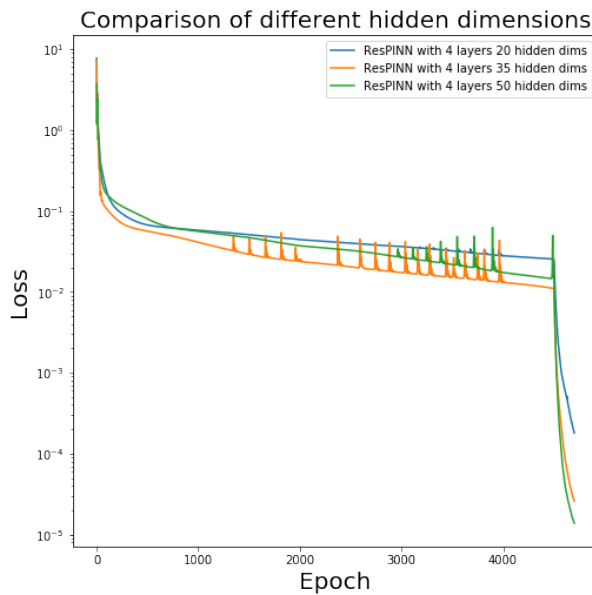


图 26: 对比图

可以发现, 随着  $\text{dim\_hidden}$  的增加, loss 值在训练过程中会出现一些起伏, 但是我们一看发现,  $\text{dim\_hidden}$  为 35 时, loss 下降最快, 所以, 并非  $\text{dim\_hidden}$  越大越好, 而应该依据实际问题具体分析。

### 1.7.2 ResPiNN

类似得，我们依旧选取  $\text{dim\_hidden}$  等于 20, 35, 50 这三种情况，对模型进行训练，得到的 loss 的变化以及三种情况的对比分别如下四幅图：

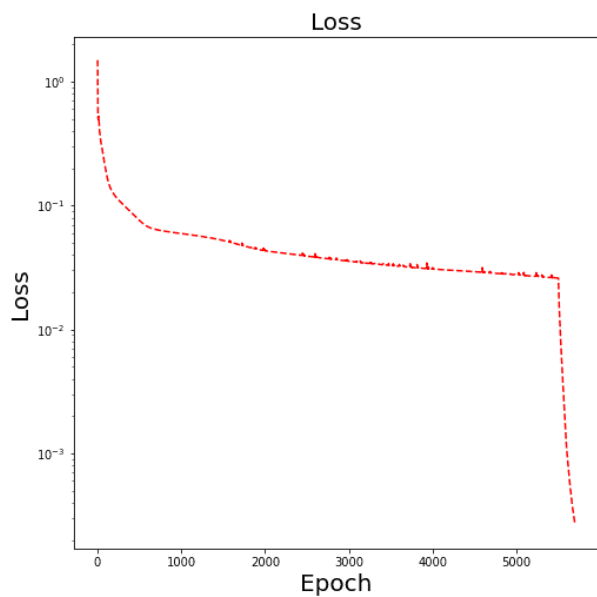


图 27:  $\text{dim\_hidden}=20$

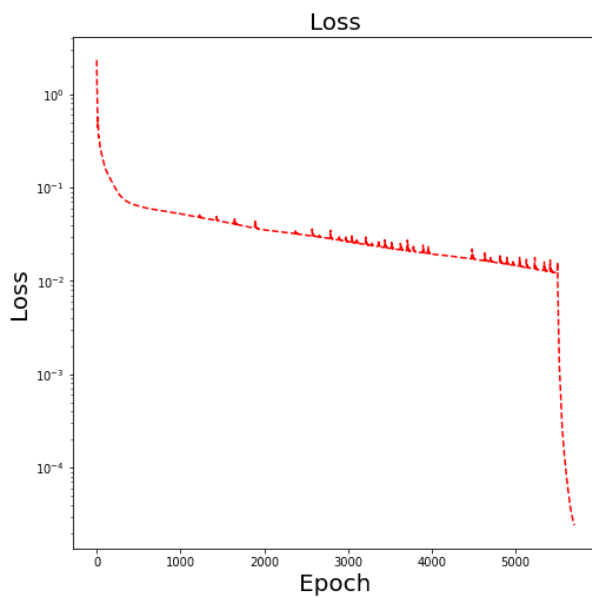


图 28:  $\text{dim\_hidden}=35$

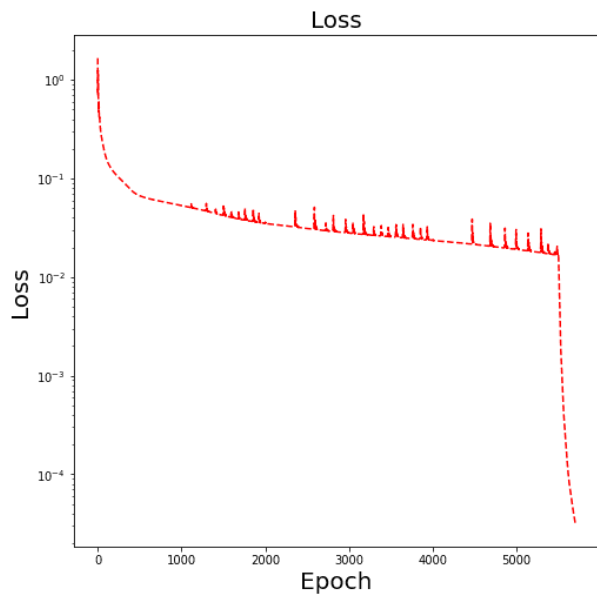


图 29:  $\text{dim\_hidden}=50$

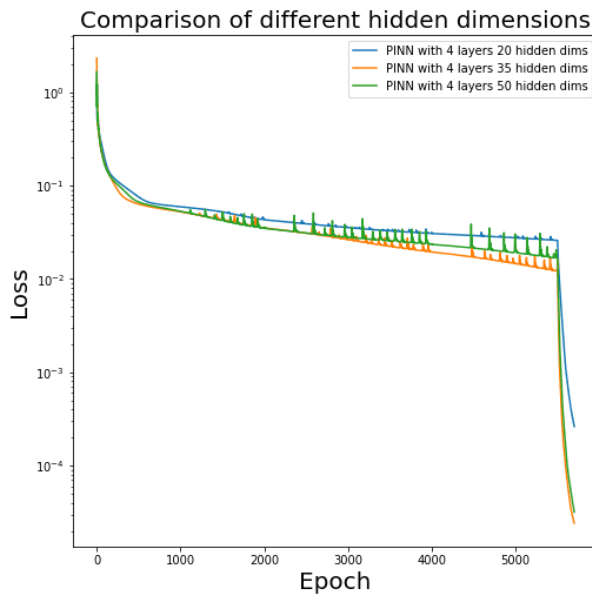


图 30: 对比图

同样地，我们依旧发现  $\text{dim\_hidden}$  增大会使 loss 值产生一些起伏，并且这里依旧是  $\text{dim\_hidden}$  为 35 时 loss 下降的最快，这样进一步验证了我们之前的推断。

## 2 利用 LSTM 预测金融市场

### 2.1 数据处理

由于需要预测股票涨跌，所以我们首先对每一天的股票涨跌情况打上标签。规则为：当股票价格的变化量大于所有股票价格变化量的中位数时，就认为股票上涨，在数据集中记为 1，而当股票价格变化量小于所有股票价格变化量中位数时认为股票下跌，在数据集中标记为 0。我们选取 240 天的股票价格数据和第 241 天的股票涨跌情况组成一个样本标签对。

由于需要使用 LSTM 网络，我们对样本标签集进行了维度上的变化。样本集最终维度为（样本数，240，1），标签集的维度为（样本数，1）。

在数据处理过程中，我们采用了以下函数：

- `centralize(data)` 对数据进行中心化
- `judge(dataset,k)` 判断第 `k` 天各支股票的涨跌情况。
- `create_dataset(dataset,look_back=240)` 生成样本标签对

具体数据处理相关代码如下：

```
1 def centralize(data):
2     min_value = np.min(data,axis=0)
3     max_value = np.max(data,axis=0)
4     data = (data - min_value) / (max_value-min_value)
5     return data
6
7 def judge(dataset,k):
8     '''
9     to see at day k, if each stock rise or fall
10    '''
11    pr_today = dataset[k]
12    pr_yesterday = dataset[k-1]
13    pr_change = pr_today - pr_yesterday
14    med = np.median(pr_change)
15    re = np.zeros_like(pr_change)
16    re[pr_change>0] = 1
17    return(re)
18
19 def create_dataset(dataset,look_back=240):
20     dataX,dataY=[],[]
21     for i in range(len(dataset)-look_back):
22         pr_change = judge(dataset,i+look_back)
23         a = dataset[i:(i+look_back)]
24         dataX.append(a)
25         dataY.append(pr_change)
```

```

26     return np.array(dataX),np.array(dataY)
27
28 look_back = 240
29 index_used = index[look_back:]
30 index_used = np.array(index_used)
31 X, Y = create_dataset(dataset,look_back)
32 print(X.shape, Y.shape)
33 a,b,c = X.shape
34
35 train_size = int(len(X) * 0.9)
36 valid_size = len(X) - train_size
37 index_size = int(len(index_used)*0.9)
38 print(train_size, valid_size)
39
40 X_train = X[:train_size]
41 Y_train = Y[:train_size]
42 index_train = index_used[:index_size]
43
44
45 X_valid = X[train_size:]
46 Y_valid = Y[train_size:]
47 index_valid = index_used[index_size:]
48
49
50 # X_train = X_train.reshape(-1,198,240)
51 # X_valid = X_valid.reshape(-1,198,240)
52 # Y_train = Y_train.reshape(-1,198,1)
53
54 X_train = X_train.reshape(train_size*c,b,1)
55 Y_train = Y_train.reshape(train_size*c,1)
56 X_valid = X_valid.reshape(valid_size*c,b,1)
57 Y_valid = Y_valid.reshape(valid_size*c,1)
58
59 # X_train = X_train.transpose(1, 0, 2)
60 # X_valid = X_valid.transpose(1, 0, 2)
61
62 X_train = torch.from_numpy(X_train)
63 Y_train = torch.from_numpy(Y_train)
64 X_valid = torch.from_numpy(X_valid)
65
66 print(X_train.shape,Y_train.shape)

```

## 2.2 算法解释及网络结构

LSTM 全程为 Long Short-Term Memory, 顾名思义, 它具有记忆长短期信息的能力的神经网络。LSTM 首先在 1997 年由 Hochreiter 和 Schmidhuber 提出, 由于深度学习在 2012 年的兴起, LSTM 又经过了若干代大牛 (Felix Gers, Fred Cummins, Santiago Fernandez, Justin Bayer, Daan Wierstra, Julian Togelius, Faustino Gomez, Matteo Gagliolo, and Alex Graves) 的发展, 由此便形成了比较系统且完整的 LSTM 框架, 并且在很多领域得到了广泛的应用, 尤其是对于时间序列数据常常能达到很好的效果。而对于股票价格数据, 则很适合采用 LSTM 网络来分析和处理。

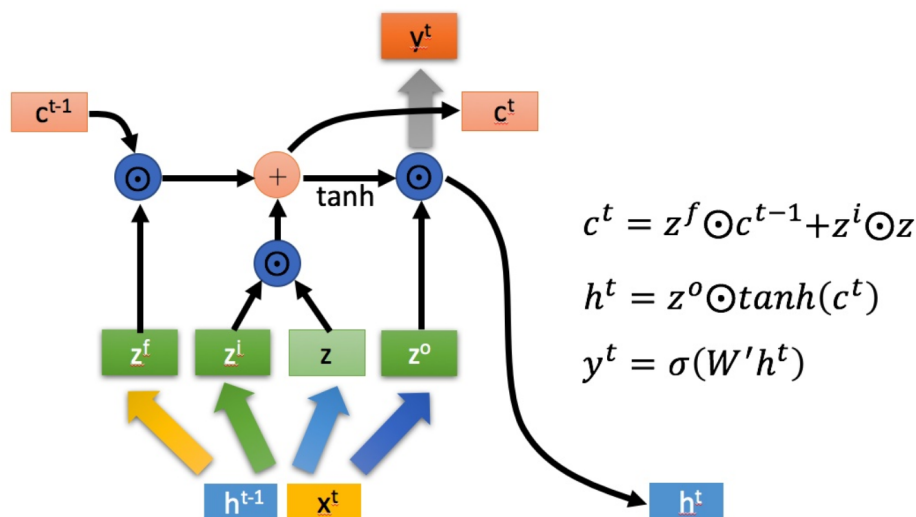


图 31: LSTM 网络图解

LSTM 内部主要有三个阶段:

- 忘记阶段。这个阶段主要是对上一个节点传进来的输入进行选择性的忘记。简单来说就是会“忘记不重要的, 记住重要的”。即利用  $z^f$  来控制上个阶段传入的  $c^{t-1}$  被遗忘的程度。
- 选择记忆阶段。这个阶段将这个阶段的输入有选择性地进行“记忆”。主要是会对输入  $x^t$  进行选择记忆。哪些重要则着重记录下来, 哪些不重要, 则少记一些。当前的输入内容由前面计算得到的  $z$  表示。而选择的门控信号则是由  $z^i$  (i 代表 information) 来进行控制
- 输出阶段。这个阶段将决定哪些将会被当成当前状态的输出。主要是通过  $z^o$  来进行控制的。并且还对上一阶段得到的  $c^o$  进行了放缩 (通过一个  $\tanh$  激活函数进行变化)。

由于本次报告采用 pytorch 框架下的 LSTM 网络, 所以在此解释所需要关注的网络参数。

- input\_size 输入数据的大小
- hidden\_size 隐藏层大小
- num\_layers 循环神经网络层数

而 LSTM 有单向 LSTM 和双向 LSTM 之分, 单向 LSTM 主要采用过去的信息, 而双向 LSTM 既采用过去的信息, 又采用现在的信息。对于本次实验, 单向 LSTM 主要是将所有股票无区别对待, 不考虑股票之间的相互影响, 直接利用样本标签对进行训练和验证。而双向 LSTM 则考虑股票之间的影响, 将同一时间段的所有股票数据看作一个样本标签对。反映到 LSTM 网络搭建中, 则是单向 LSTM 网络中我们把 input\_size 和 num\_layers 设为 1, 而隐藏层数目则在后面会进行多样性分析。在 LSTM 网络之后, 我们还连接了一个线性全连接网络以完成判别工作。为了线性网络能与 LSTM 顺利连接, 其 in\_features 与 LSTM 网络的 hidden\_size 相同, out\_features 为 1。类似地, 我们在后面也对不同的损失函数和优化器作了对比。对于双向 LSTM 网络, 我们把 input\_size 设为 240, num\_layers 设为 2。类似地, 我们依旧连接了一个全连接网络, 但是由于这里是双向 LSTM, 所以我们把 in\_features 设为 2 倍的 hidden\_size, 而 out\_features 依旧为 1。

至此, 网络的基本结构已经搭建完成。但为了更好的得到以及呈现结果, 我们小组也添加了一些其他代码及函数。

- acc(out,y\_real) 对判别结果的正确率进行计算, 输入参数为真实判别值和预测判别值。
- set\_seed(seed) 设置种子, 使得每次结果一样, 便于检测代码错误。
- 采用 DataLoader 来做 batch

具体网络搭建代码如下:

```
1 %%time
2 class LSTMRegression(nn.Module):
3     def __init__(self, input_size, hidden_size, output_size=1,
4         num_layers=1):
5         super().__init__()
6         self.lstm = nn.LSTM(input_size, hidden_size, num_layers,
7             batch_first=True)
8         self.linear = nn.Linear(hidden_size, output_size)
9
10    def forward(self, x):
11        _, (hn, cn) = self.lstm(x)
12        hn = hn.squeeze()
13        out = self.linear(hn)
14        return out
15
16 model = LSTMRegression(input_size=1, hidden_size=5, output_size=1)
17
18 criterion = torch.nn.BCEWithLogitsLoss()      #交叉熵BCEWithLogitsLoss()
19 和 MultiLabelSoftMarginLoss()
20 #criterion = torch.nn.CrossEntropyLoss()
21 optimizer = optim.Adam(model.parameters(), lr=1e-3)
22 #optimizer = optim.SGD(model.parameters(), lr=1e-1)
23
24 epochs = 100
```

```

22 batch_size = 30
23 batch = X_train.shape[0] // batch_size
24
25
26
27 torch_dataset = Data.TensorDataset(torch.tensor(X_train), torch.tensor(
    Y_train))
28 # 把 dataset 放入 DataLoader
29 loader = Data.DataLoader(
30     dataset=torch_dataset, # torch TensorDataset format
31     batch_size=batch_size, # mini batch size
32     shuffle=True, #
33     num_workers=10, # 多线程来读数据
34 )
35
36 loss_epoch = np.zeros(epochs)
37 acc_epoch = np.zeros(epochs)
38 loss_valid = np.zeros(epochs)
39 acc_valid = np.zeros(epochs)
40 for epoch in range(epochs):
41     loss_ep = np.array([])
42     acc_ep = np.array([])
43     loss_epv = np.array([])
44     acc_epv = np.array([])
45     for step,(var_x,var_y) in enumerate(loader):
46         out = model(var_x)
47         out_f = out.detach().clone().numpy()
48         var_yf = var_y.detach().clone().numpy()
49         loss = criterion(out, var_y)
50         loss_f = loss.detach().clone().numpy()
51         acc_ep = np.append(acc_ep,acc(out_f,var_yf))
52         loss_ep = np.append(loss_ep,loss_f)
53
54         optimizer.zero_grad()
55         loss.backward()
56         optimizer.step()
57
58     if (epoch + 1) % 5 == 0:
59         print(f'Epoch: {epoch:5d}, Loss: {np.mean(loss_ep):.4e}, Acc:{np
            .mean(acc_ep):.4e}')
60

```

```

61
62     loss_epoch[epoch] = np.mean(loss_ep)
63     acc_epoch[epoch] = np.mean(acc_ep)
64
65     Y_pre = model(X_valid) #计算验证集表现
66     Y_pre1 = Y_pre.clone().detach().numpy()
67     Y_valid1 = torch.from_numpy(Y_valid)
68     loss_valid[epoch] = criterion(Y_pre,Y_valid1)
69     acc_valid[epoch] = acc(Y_pre1,Y_valid)

```

为了更好地呈现结果，我们采取了一系列可视化工作，主要画出了以下图像：

- loss 随着 epoch 变化的图像
- 准确率随着 epoch 变化的图像
- 模型在验证集上的预测判别值和真实涨跌情况对比

具体代码如下：

```

1     # visulize
2     kind = 2
3     series = np.arange(kind*len(index_valid),(kind+1)*len(index_valid))
4     Y_pred_re = Y_pred
5     Y_pred_re[Y_pred_re>0] = 1
6     Y_pred_re[Y_pred_re<=0] = 0
7
8
9     filename1 = 'point' + str(index) + '.png'
10    filename2 = 'pic' + str(index) + '.png'
11
12    fig = plt.figure()
13    ax = plt.subplot()
14    type1 = ax.scatter(index_valid, Y_valid[series], alpha=0.6,color='b',
15                        label='groundtruth')
16    type2 = ax.scatter(index_valid, Y_pred_re[series], alpha=0.3,color='
17                        r',label='prediction')
18    plt.xlabel("date time")
19    plt.ylabel("0 for fall, 1 for rise")
20    ax.legend((type1, type2), (u'groundtruth', u'prediction'), loc='best
21              ')
22    plt.savefig(filename1)
23    plt.show()

```



```

22     plt.plot(loss_epoch, 'r-', label='loss')
23     plt.plot(acc_epoch, 'b-', label='accurate rate')
24     plt.legend(loc='best')
25     plt.savefig(filename2)
26     plt.show()

```

最后，采用股票价格数据训练模型。由于数据量较大，所以我们只训练了 100 轮，得到的 loss 和正确率图像如下图所示：

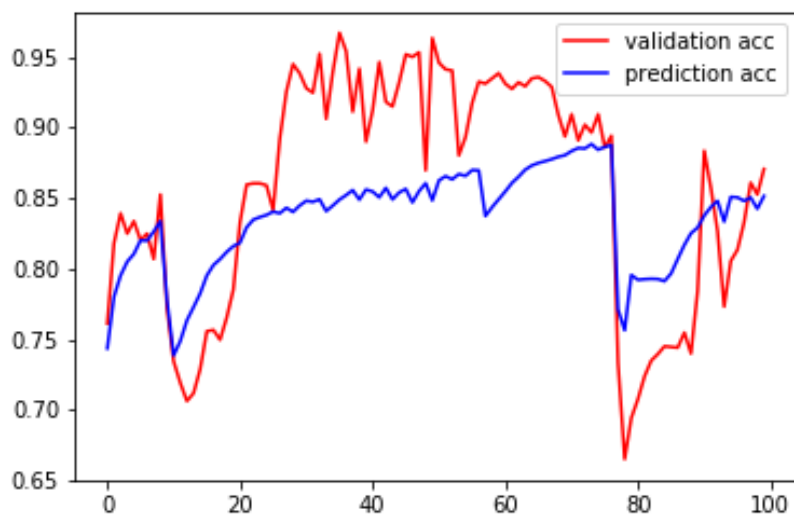


图 32: 单向 LSTM 网络训练集和验证集上的正确率变化

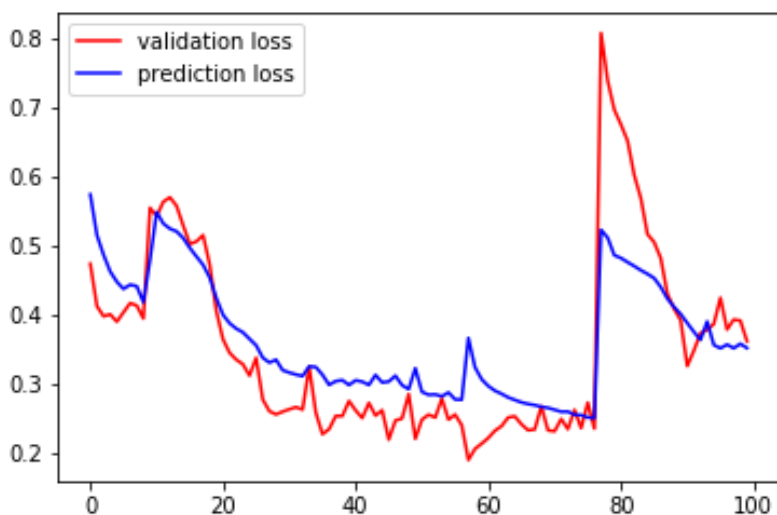


图 33: 单向 LSTM 网络训练集和验证集上的 loss 值变化

由以上两幅图可以发现，对于单向 LSTM 网络，在训练了 20 轮到 80 轮时得到的模型在验证集上表现较好且较稳定。到 80 轮之后验证集上的判断正确率产生了较大下降，猜测可能是产生了过拟合或者是网络超参数设置不当。所以，我们在后面主要对网络参数进行了分析实验，轮数一般设置在 20 到 100 之间。

对于双向 LSTM 网络，我们同样得到训练集和验证集的 loss 和准确率的变化图像，如下：

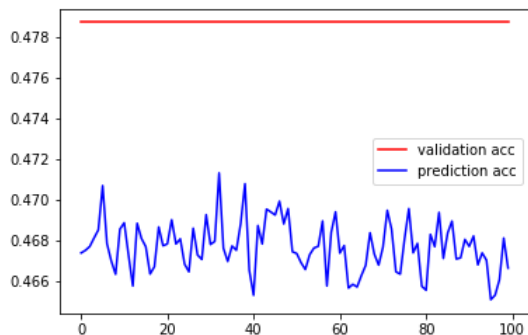


图 34: 双向 LSTM 网络训练集和验证集上的正确率变化

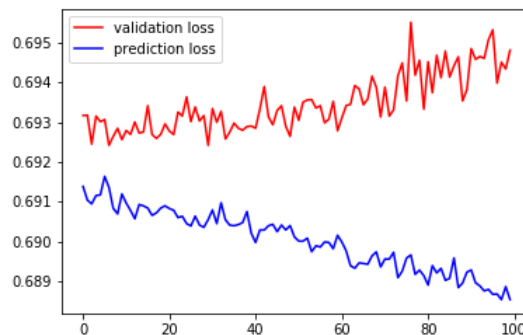


图 35: 双向 LSTM 网络训练集和验证集上的 loss 变化

由两幅图的纵坐标可以看到，无论是验证集还是训练集，loss 和准确率变化并不大，由此可见双向 LSTM 网络不适合处理股票数据或者网络参数设置有问题。

## 2.3 不同的 hidden size/神经元数目对结果的影响

<sup>1</sup> 由于 input size 和 output size 固定，所以我们主要通过改变 hidden size 来改变神经元数目，进而进行分析。

我们分别对隐藏层为 3，4，5 的情况进行了实验，得到的损失函数值和准确率随时间变化图像如下所示：

---

<sup>1</sup>本节代码附在报告最后

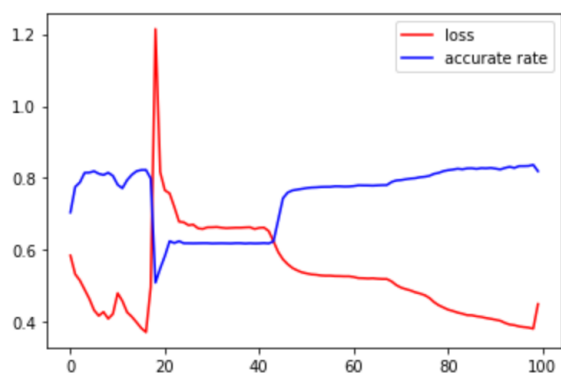


图 36: hidden size=3

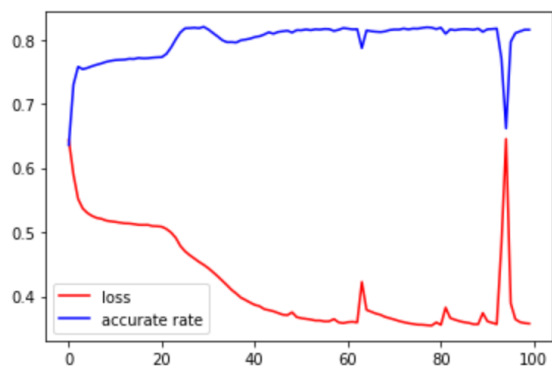


图 37: hidden size=4

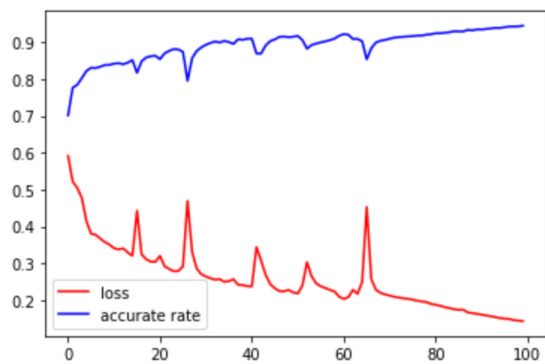


图 38: hidden size=5

训练得到的模型在验证集上的表现如下所示：

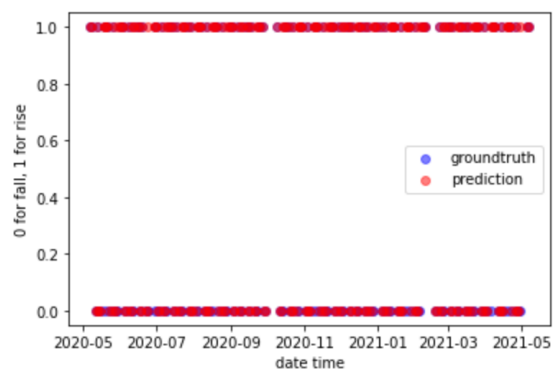


图 39: hidden size=3

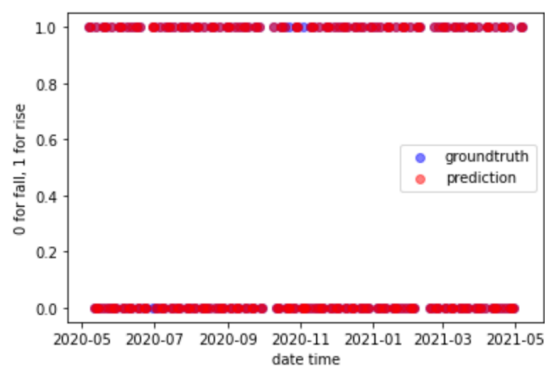


图 40: hidden size=4

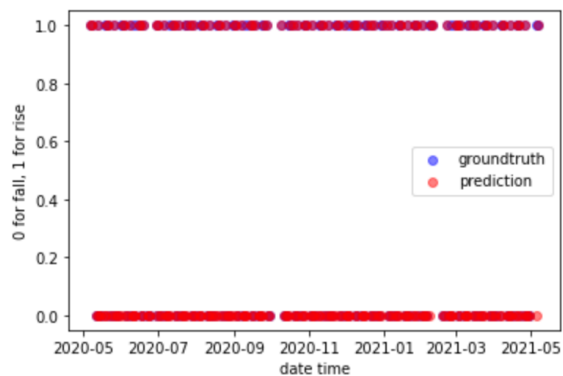


图 41: hidden size=5

根据以上结果不难看出，这三种 hidden size 的结果都比较好，虽然中间有所波动，但最终得到的正确率都比较高。并且，隐藏层为 5 时得到的模型的正确率变化最稳定，效果最好。

而对于双向 LSTM 网络，我们的训练过程中的 loss 和正确率的变化如下：

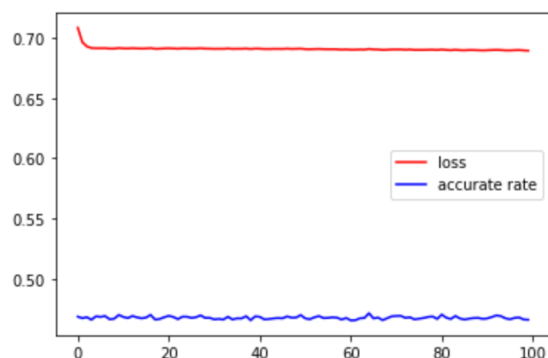


图 42: hidden size=2

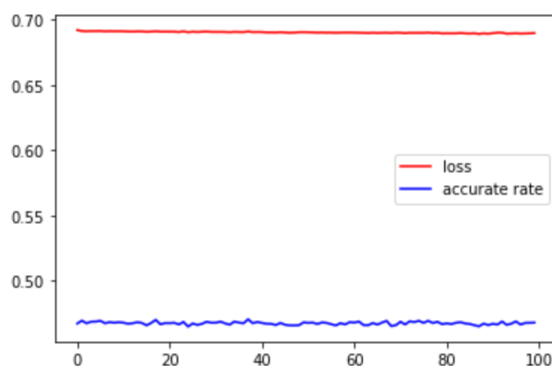


图 43: hidden size=3

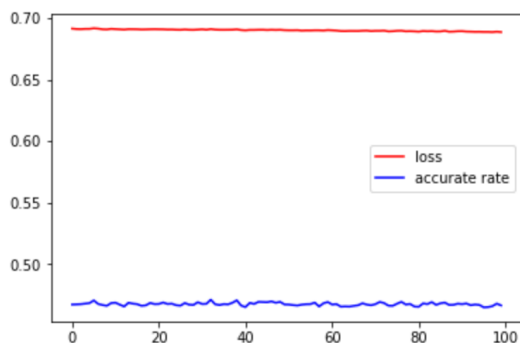


图 44: hidden size=4

我们可以看出，对于双向 LSTM 网络，在训练集上的表现并不好，loss 和正确率基本不变。这说明双向 LSTM 网络可能不太适合对股票数据进行判断，因此，在报告的后面我们也是主要集中于对单向 LSTM 网络的分析。

## 2.4 不同的 batch size 对结果的影响

<sup>2</sup> 利用控制变量的方法，保持其他变量不变，改变 batch\_size，在训练 20 轮的情况下，分析单向 lstm 网络的训练结果。每训练 2 个 epoch，打印一次相应的损失函数 loss 和准确率 acc。batch\_size 影响模型的训练过程，进而影响模型的性能。一方面影响模型的收敛时间，另一方面影响训练好的模型的泛化能力即在验证集上的效果。下面通过不同的 batch\_size 来探讨这个问题并选出较好的参数。训练过程中，随着训练轮数增加，通过观察在训练集上 acc 和 loss 的变化，可以知道模型的收敛程度。在下图中，分别画出了 batch\_size 为 10, 30, 60 的情况下，训练 20 轮的过程中，acc 和 loss 的变化。

<sup>2</sup>本节代码附在报告最后

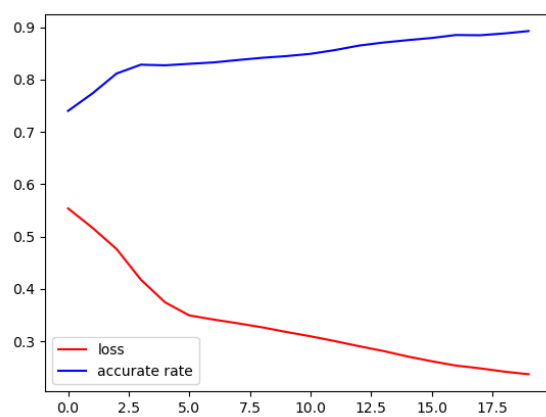


图 45: batch size=10

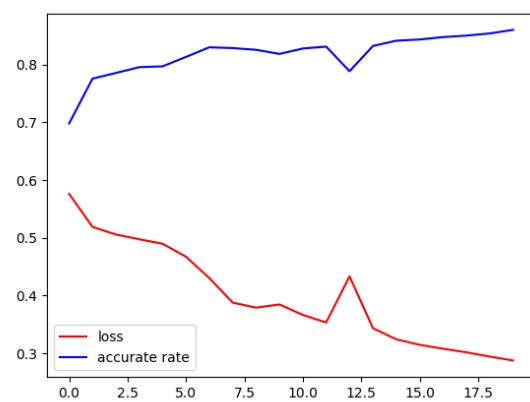


图 46: batch size=30

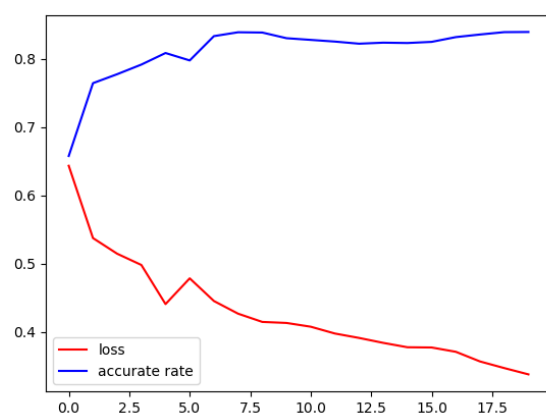


图 47: batch size=60,epoch=20

## 2.5 不同的学习率对结果的影响

<sup>3</sup> 利用控制变量的方法，保持其他变量不变，改变学习率  $lr$ (learning rate)，在训练 30 轮的情况下，分析单向 lstm 网络的训练结果。每训练 2 个 epoch，打印一次相应的损失函数 loss 和准确率 acc。学习率的影响体现为，学习率越大，输出误差对参数的影响就越大，参数更新的就越快，但同时受到异常数据的影响也就越大，很容易发散。并且在一方面影响模型的收敛时间，另一方面影响训练好的模型的泛化能力即在验证集上的效果。下面通过不同的  $lr$  来探讨这个问题并选出较好的参数。训练过程中，随着训练轮数增加，通过观察在训练集上 acc 和 loss 的变化，可以知道模型的收敛程度。在下图中，分别画出了学习率为  $1e-2, 1e-3, 1e-4$  的情况下，训练 30 轮的过程中，acc 和 loss 的变化。

以下是随训练轮数增加，预测的股票涨跌预测情况：

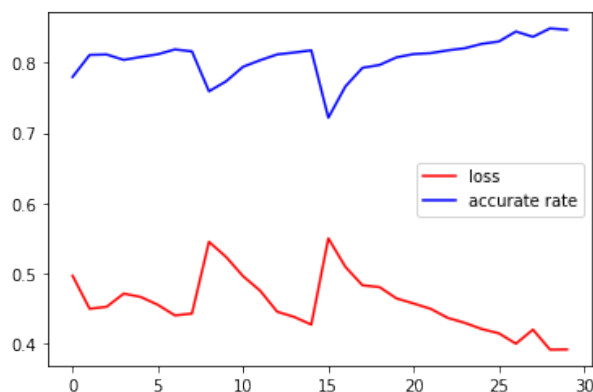


图 48:  $lr=0.01$

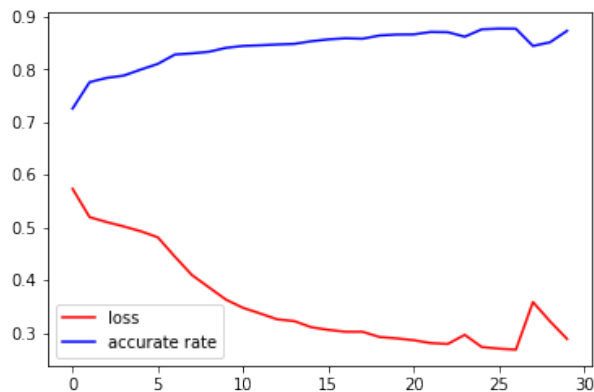


图 49:  $lr=0.001$

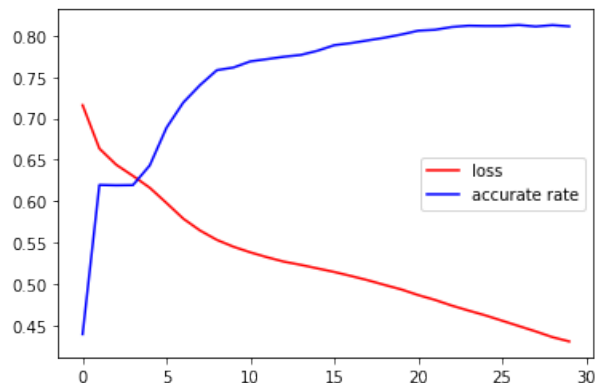


图 50:  $lr=0.0001$

以下列出了训练过程中的 acc。

---

<sup>3</sup>本节代码附在报告最后

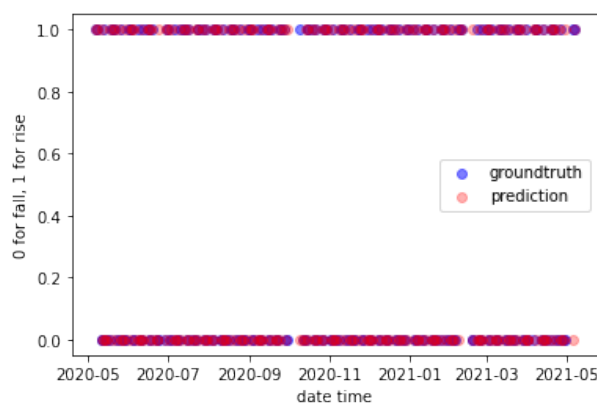


图 51:  $lr=0.01$ , 验证集结果

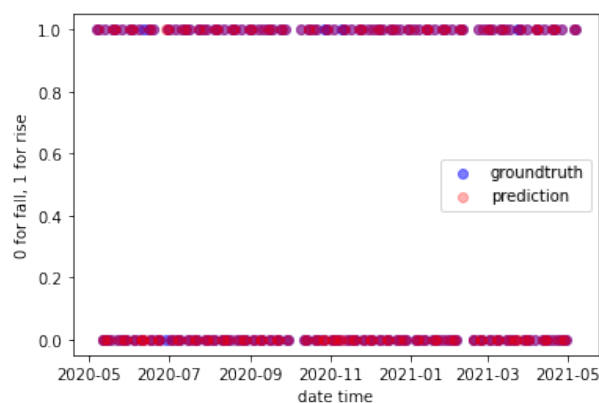


图 52:  $lr=0.001$ , 验证集结果

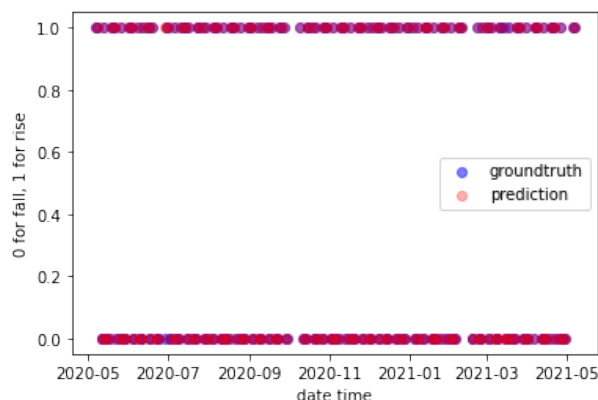


图 53:  $lr=0.0001$ , 验证集结果

运行时间上, 学习率越小, 运行 30 轮的时间越久, 但准确率也随之提高。 $lr=1e-2$  运行时间最短, $lr=1e-4$  准确率最高。

收敛速度上, 30 轮过后, 三个  $lr$  对应的  $loss$  都降到了 0.4 以下, 但具体来看,  $lr=1e-2$  时,  $loss$  呈现周期性的下降, 说明学习率过大, 不具有稳定性,  $lr=1e-3$  时, 周期性明显改善, 但最终仍有上升的趋势,  $lr=1e-4$  时,  $loss$  的下降非常光滑。三种方法的准确率最终都达到 0.8 左右, 说明该方法的在训练集上表现较好。

在验证集中验证模型, 得到不同  $lr$  对应的在验证集中的  $acc$ 。根据该数据, 可以考察模型的泛化能力。对应  $1e-2$ ,  $1e-3$ ,  $1e-4$ , 训练完在验证集上的准确率分别为 0.625, 0.901, 0.918。比较三点在验证集上的准确率,  $lr=1e-4$  的模型达到了三个中的最高值 0.918

## 2.6 不同的优化器对结果的影响

<sup>4</sup> 在这一部分, 我们考虑了 Adam 和 SGD 优化器, 并对结果进行了检验。对于 Adam 优化器, 我们发现最终在验证集下的判断准确率为 0.919, 对于 SGD 优化器, 我们得到的验证集判断准确率为 0.747。因此对于股票数据, 可能 Adam 优化器更合适。我们的训练过程中  $loss$  函数和准确率随着训练轮数的变化如下图所示:

<sup>4</sup> 本节代码附在报告最后



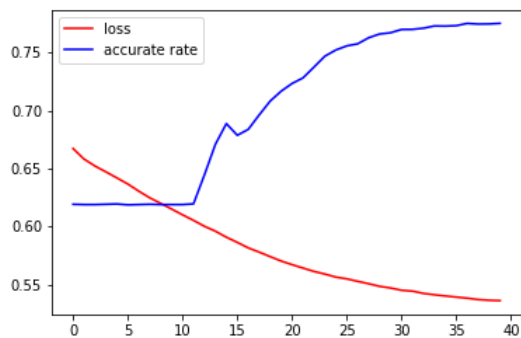


图 54: SGD

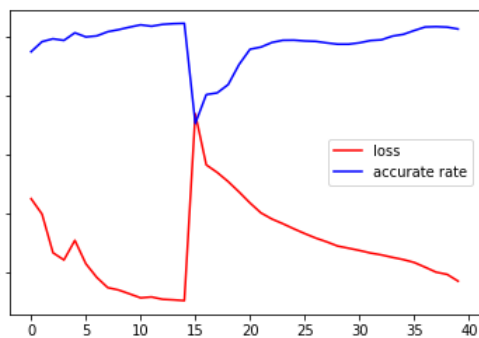


图 55: Adam

我们可以看到，Adam 优化器对应的损失函数值在中间经历了一段上升过程，分析可能的原因是陷入了局部最小，可以将学习率适当调小。最后，我们给出由训练出来的模型在验证集上的判断表现：

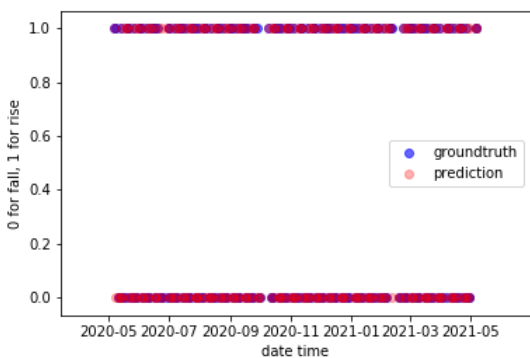


图 56: SGD

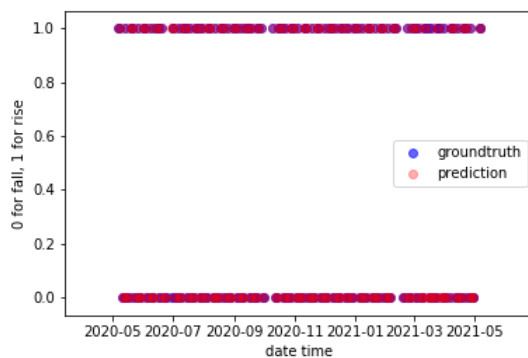


图 57: Adam

## 附录

### A 神经网络求解偏微分方程代码

#### A.1 网络架构

```
1 import torch
2 import torch.nn as nn
3 import torch.optim as optim
4 from torch.optim.lr_scheduler import StepLR
5 import os
6 import numpy as np
7 import shutil
8 import matplotlib.pyplot as plt
9 from pyDOE import lhs
10 import argparse
11 import scipy.io
12
13 def activation(name):
14     """define all the activation function
15     """
16     if name in ['tanh', 'Tanh']:
17         return nn.Tanh()
18     elif name in ['relu', 'ReLU']:
19         return nn.ReLU(inplace=True)
20     elif name in ['leaky_relu', 'LeakyReLU']:
21         return nn.LeakyReLU(inplace=True)
22     elif name in ['sigmoid', 'Sigmoid']:
23         return nn.Sigmoid()
24     elif name in ['softplus', 'Softplus']:
25         return nn.Softplus()
26     else:
27         raise ValueError(f'unknown activation function: {name}')
28
29 class Problem(object):
30     """
31     rewrite the problem;define its domain,
32     """
33     def __init__(self, domain=(0,np.pi/2,-5,5)):
34         self.domain = domain
35
```

```

36     def __repr__(self):
37         return f'{self.__doc__}'
38
39     def ic(self, x):
40         x=x.detach().cpu().numpy()
41         f_ic=-2/np.cosh(x[:,1])
42         f_ic=f_ic.reshape(f_ic.size,1)
43         return torch.from_numpy(f_ic).float()
44
45     def true_value(self, t):
46         data = scipy.io.loadmat('NLS.mat')
47         x = data['x'].flatten()[:, None]
48         Exact = data['uu']
49         Exact_u = np.real(Exact)
50         Exact_v = np.imag(Exact)
51         t=400*t/np.pi
52         t=np.int(t)
53         ut = Exact_u[:,t]
54         vt = Exact_v[:,t]
55         ht=np.sqrt(ut**2+vt**2)
56         ht=ht.reshape(ht.size,1)
57         return ht
58
59 def grad(outputs, inputs):
60     """compute the derivative of outputs associated with inputs
61         input: (N,D) tensors
62         output:(N,1) tensors
63     """
64     return torch.autograd.grad(outputs, inputs,grad_outputs=torch.
65         ones_like(outputs),create_graph=True)
66
67 class DNN(nn.Module):
68     def __init__(self, dim_in, dim_out, dim_hidden, hidden_layers,
69         act_name='tanh', init_name=None):
70         super().__init__()
71         model = nn.Sequential()
72         model.add_module('fc0', nn.Linear(dim_in, dim_hidden, bias=True)
73         )
74         model.add_module('act0', activation(act_name))
75         for i in range(1, hidden_layers):
76             model.add_module(f'fc{i}', nn.Linear(dim_hidden, dim_hidden,

```

```

        bias=True))
74         model.add_module(f'act{i}', activation(act_name))
75     model.add_module(f'fc{hidden_layers}', nn.Linear(dim_hidden,
        dim_out, bias=True))
76     self.model = model
77     if init_name is not None:
78         self.init_weight(init_name)
79
80     def init_weight(self, name):
81         if name == 'xavier_normal':
82             nn_init = nn.init.xavier_normal_
83         elif name == 'xavier_uniform':
84             nn_init = nn.init.xavier_uniform_
85         elif name == 'kaiming_normal':
86             nn_init = nn.init.kaiming_normal_
87         elif name == 'kaiming_uniform':
88             nn_init = nn.init.kaiming_uniform_
89         else:
90             raise ValueError(f'unknown initialization function: {name}')
91         for param in self.parameters():
92             if len(param.shape) > 1:
93                 nn_init(param)
94
95     def forward(self, x):
96         return self.model(x)
97
98     def forward_test(self, x):
99         print(f"{'input':<20}{str(x.shape):<40}")
100         for name, module in self.model._modules.items():
101             x = module(x)
102             print(f"{name:<20}{str(x.shape):<40}")
103         return x
104
105     def model_size(self):
106         n_params = 0
107         for param in self.parameters():
108             n_params += param.numel()
109         return n_params
110
111 class ResBlock(nn.Module):
112     def __init__(self, dim_in, dim_out, dim_hidden, act_name='tanh'):

```

```

113         super().__init__()
114
115         assert(dim_in == dim_out)
116         block = nn.Sequential()
117         block.add_module('act0', activation(act_name))
118         block.add_module('fc0', nn.Linear(dim_in, dim_hidden, bias=True)
119         )
120         block.add_module('act1', activation(act_name))
121         block.add_module('fc1', nn.Linear(dim_hidden, dim_out, bias=True
122         ))
123         self.block = block
124
125     def forward(self, x):
126         identity = x
127         out = self.block(x)
128         return identity + out
129
130 class ResDNN(nn.Module):
131     def __init__(self, dim_in, dim_out, dim_hidden, res_blocks, act_name
132     ='tanh', init_name='kaiming_normal'):
133         super().__init__()
134
135         model = nn.Sequential()
136         model.add_module('fc_first', nn.Linear(dim_in, dim_hidden, bias=
137         True))
138         for i in range(res_blocks):
139             res_block = ResBlock(dim_hidden, dim_hidden, dim_hidden,
140             act_name=act_name)
141             model.add_module(f'res_block{i+1}', res_block)
142         model.add_module('act_last', activation(act_name))
143         model.add_module('fc_last', nn.Linear(dim_hidden, dim_out, bias=
144         True))
145
146         self.model = model
147         if init_name is not None:
148             self.init_weight(init_name)
149
150     def init_weight(self, name):
151         if name == 'xavier_normal':
152             nn_init = nn.init.xavier_normal_
153         elif name == 'xavier_uniform':

```

```

148         nn_init = nn.init.xavier_uniform_
149     elif name == 'kaiming_normal':
150         nn_init = nn.init.kaiming_normal_
151     elif name == 'kaiming_uniform':
152         nn_init = nn.init.kaiming_uniform_
153     else:
154         raise ValueError(f'unknown initialization function: {name}')
155
156     for param in self.parameters():
157         if len(param.shape) > 1:
158             nn_init(param)
159
160     def forward(self, x):
161         return self.model(x)
162
163     def forward_test(self, x):
164         print(f"{'input':<20}{str(x.shape):<40}")
165         for name, module in self.model._modules.items():
166             x = module(x)
167             print(f"{name:<20}{str(x.shape):<40}")
168         return x
169
170     def model_size(self):
171         n_params = 0
172         for param in self.parameters():
173             n_params += param.numel()
174         return n_params
175
176 class PINN(DNN):
177     def __init__(self, dim_in, dim_out, dim_hidden, hidden_layers,
178                 act_name='tanh', init_name='xavier_normal'):
179         super().__init__(dim_in, dim_out, dim_hidden, hidden_layers,
180                         act_name=act_name, init_name=init_name)
181     def forward(self, x):
182         x.requires_grad_(True)
183         h= super().forward(x)
184         a = torch.split(h, 1, dim=1)
185         u = a[0]
186         v = a[1]
187         grad_u = grad(u, x)[0]
188         u_x = grad_u[:, [1]]

```

```

187     u_t = grad_u[:, [0]]
188     u_xx = grad(u_x, x)[0][:, [1]]
189     grad_v = grad(v, x)[0]
190     v_x = grad_v[:, [1]]
191     v_t = grad_v[:, [0]]
192     v_xx = grad(v_x, x)[0][:, [1]]
193     f_u = u_t + 0.5*v_xx + (u**2 + v**2)*v
194     f_v = v_t - 0.5*u_xx - (u**2 + v**2)*u
195     return u, v, u_x, v_x, f_u, f_v
196
197 class ResPINN(ResDNN):
198     def __init__(self, dim_in, dim_out, dim_hidden, res_blocks, act_name=
        'tanh', init_name='xavier_normal'):
199         super().__init__(dim_in, dim_out, dim_hidden, res_blocks,
        act_name=act_name, init_name=init_name)
200     def forward(self, x):
201         x.requires_grad_(True)
202         h = super().forward(x)
203         a = torch.split(h, 1, dim=1)
204         u = a[0]
205         v = a[1]
206         grad_u = grad(u, x)[0]
207         u_x = grad_u[:, [1]]
208         u_t = grad_u[:, [0]]
209         u_xx = grad(u_x, x)[0][:, [1]]
210         grad_v = grad(v, x)[0]
211         v_x = grad_v[:, [1]]
212         v_t = grad_v[:, [0]]
213         v_xx = grad(v_x, x)[0][:, [1]]
214         f_u = u_t + 0.5*v_xx + (u**2+v**2)*v
215         f_v = v_t - 0.5*u_xx - (u**2+v**2)*u
216         return u, v, u_x, v_x, f_u, f_v
217
218 class Options(object):
219     def __init__(self):
220         parser = argparse.ArgumentParser()
221         parser.add_argument('--no_cuda', action='store_true', default=
        False, help='disable CUDA or not')
222         parser.add_argument('--dim_hidden', type=int, default=50, help='
        neurons in hidden layers')
223         parser.add_argument('--hidden_layers', type=int, default=4, help=

```

```

    = 'number of hidden layers')
224 parser.add_argument('--res_blocks', type=int, default=4, help='
    number of residual blocks')
225 parser.add_argument('--lam', type=float, default=1, help='weight
    in loss function')
226 parser.add_argument('--lr', type=float, default=1e-3, help='
    initial learning rate')
227 parser.add_argument('--epochs_Adam', type=int, default=4500,
    help='epochs for Adam optimizer')
228 parser.add_argument('--epochs_LBFGS', type=int, default=200,
    help='epochs for LBFGS optimizer')
229 parser.add_argument('--step_size', type=int, default=2000, help='
    step size in lr_scheduler for Adam optimizer')
230 parser.add_argument('--gamma', type=float, default=0.7, help='
    gamma in lr_scheduler forAdam optimizer')
231 parser.add_argument('--resume', type=bool, default=False, help='
    resume or not')
232 parser.add_argument('--sample_method', type=str, default='lhs',
    help='sample method')
233 parser.add_argument('--n_x', type=int, default=100, help='sample
    points in x-direction for uniform sample')
234 parser.add_argument('--n_t', type=int, default=100, help='sample
    points in t-direction for uniform sample')
235 parser.add_argument('--n', type=int, default=10000, help='sample
    points in domain for lhs sample')
236 parser.add_argument('--n_bc', type=int, default=400, help='
    sample points on the boundary for lhs sample')
237 parser.add_argument('--n_ic', type=int, default=400, help='
    sample points on the initial time for lhs sample')
238 self.parser = parser
239 def parse(self):
240     arg = self.parser.parse_args(args=[])
241     arg.cuda = not arg.no_cuda and torch.cuda.is_available()
242     arg.device = torch.device('cuda' if torch.cuda.is_available()
    else 'cpu') #choose the environment according to whether you
    have cuda or not.
243
244     return arg
245 def save_model(state, is_best=None, save_dir=None):
246     """save the best and the last model
247     """

```



```

248     last_model = os.path.join(save_dir, 'last_model.pth.tar')
249     torch.save(state, last_model)
250     if is_best:
251         best_model = os.path.join(save_dir, 'best_model.pth.tar')
252         shutil.copyfile(last_model, best_model)

```

## A.2 训练集

```

1 class Trainset(object):
2
3     def __init__(self, *args, **kwargs):
4         self.domain = (0,np.pi/2,-5,5)
5         self.args = args
6         self.method = kwargs['method']
7
8     def __call__(self,verbose=None):
9         if self.method == 'uniform':
10             nx, nt, n_bc, n_ic = self.args[0], self.args[1], self.args
11                 [2], self.args[3]
12             x, x_bc_left, x_bc_right, x_ic = self._uniform_sample(nx, nt
13                 , n_bc, n_ic)
14         elif self.method == 'lhs':
15             n, n_bc, n_ic = self.args[0], self.args[1], self.args[2]
16             x, x_bc_left, x_bc_right, x_ic = self._lhs_sample(n, n_bc,
17                 n_ic)
18
19         if verbose == 'tensor':
20             x = torch.from_numpy(x).float()
21             x_bc_left = torch.from_numpy(x_bc_left).float()
22             x_bc_right = torch.from_numpy(x_bc_right).float()
23             x_ic = torch.from_numpy(x_ic).float()
24             return x, x_bc_left, x_bc_right, x_ic
25         return x, x_bc_left, x_bc_right, x_ic
26
27     def _uniform_sample(self, nx,nt, n_bc, n_ic):
28         t_min, t_max, x_min, x_max = self.domain
29         x = np.linspace(x_min,x_max,nx)
30         t = np.linspace(t_min, t_max, nt)
31         t, x = np.meshgrid(t, x)
32         tx = np.hstack((t.reshape(t.size, -1), x.reshape(x.size, -1)))
33
34         t = np.linspace(t_min, t_max, n_bc)

```

```

31     t, xl = np.meshgrid(t, x_min)
32     x_bc_left = np.hstack((t.reshape(t.size, -1), xl.reshape(xl.size
    , -1)))
33     t, xr = np.meshgrid(t, x_max)
34     x_bc_right = np.hstack((t.reshape(t.size, -1), xr.reshape(xr.
    size, -1)))
35
36     x = np.linspace(x_min,x_max,n_ic)
37     t_ic, x = np.meshgrid(t_min, x)
38     x_ic = np.hstack((t_ic.reshape(t_ic.size, -1), x.reshape(x.size,
    -1)))
39     return tx, x_bc_left, x_bc_right, x_ic
40
41     def _lhs_sample(self, n, n_bc, n_ic):
42         t_min, t_max, x_min, x_max = self.domain
43         lb = np.array([t_min, x_min])
44         ub = np.array([t_max, x_max])
45         x = lb + (ub - lb) * lhs(2, n)
46         lb = np.array([t_min, x_min])
47         ub = np.array([t_max, x_min])
48         temp = lb + (ub - lb) * lhs(2, n_bc)
49         x_bc_left = temp
50         lb = np.array([t_min, x_max])
51         ub = np.array([t_max, x_max])
52         temp = lb + (ub - lb) * lhs(2, n_bc)
53         x_bc_right = temp
54         lb = np.array([t_min, x_min])
55         ub = np.array([t_min, x_max])
56         temp = lb + (ub - lb) * lhs(2, n_ic)
57         x_ic = temp
58         return x, x_bc_left, x_bc_right, x_ic

```

### A.3 验证集

```

1 class Testset(object):
2     """The dataset is based on a square domain
3     """
4
5     def __init__(self, *args, **kwargs):
6         self.domain = (0,np.pi/2,-5,5)
7         self.args = args

```

```

8         self.method = kwargs['method']
9
10    def __repr__(self):
11        return f'{self.__doc__}'
12
13    def __call__(self, plot=False, verbose=None):
14        if self.method == 'uniform':
15            n_x, t = self.args[0], self.args[1]
16            X, x = self._uniform_sample(n_x, t)
17            if verbose == 'tensor':
18                X = torch.from_numpy(X).float()
19            return X, x
20
21    def _uniform_sample(self, n_x, t):
22        t_min, t_max, x_min, x_max = self.domain
23        x = np.linspace(x_min, x_max, n_x)
24        t, x = np.meshgrid(t, x)
25        X = np.hstack((t.reshape(t.size, -1), x.reshape(x.size, -1)))
26        return X, x

```

## A.4 训练过程

```

1 class Trainer(object):
2     def __init__(self, args): # args includes all paraments needed in the
        net
3         self.device = args.device #cpu or gpu
4         self.problem = args.problem
5
6         self.lam = args.lam # weight of the initial model
7         self.criterion = nn.MSELoss()
8
9         self.model = args.model
10        self.model_name = self.model.__class__.__name__
11        self.model_path = self._model_path()
12
13        self.epochs_Adam = args.epochs_Adam
14        self.epochs_LBFGS = args.epochs_LBFGS
15        self.optimizer_Adam = optim.Adam(self.model.parameters(), lr=
args.lr) # learning rate
16        self.optimizer_LBFGS = optim.LBFGS(self.model.parameters(),
17                                            max_iter=20,

```

```

18                                     tolerance_grad=1.e-8,
19                                     tolerance_change=1.e-12)    #
                                     everytime a parent is
                                     updated, an equation is
                                     solved and so there has to be
                                     a maximum of the iteration
20 self.lr_scheduler = StepLR(self.optimizer_Adam,
21                             step_size=args.step_size,
22                             gamma=args.gamma)
23
24 self.model.to(self.device)
25 self.model.zero_grad()
26
27 self.x, self.x_bc_left, self.x_bc_right, self.x_ic = args.
trainset(verbose='tensor')
28 self.x_val, self.x_bc_left_val, self.x_bc_right_val, self.
x_ic_val = args.validset(verbose='tensor')
29 self.true_ic = args.problem.ic(self.x_ic)
30 self.true_ic_val = args.problem.ic(self.x_ic_val)
31 if self.device == torch.device(type='cuda'):
32     self.x, self.x_bc_left, self.x_bc_right, self.x_ic = self.x.
to(self.device), self.x_bc_left.to(self.device), self.
x_bc_right.to(self.device), self.x_ic.to(self.device)
33 self.x_val, self.x_bc_left_val, self.x_bc_right_val, self.
x_ic_val = self.x_val.to(self.device), self.x_bc_left_val.to
(self.device), self.x_bc_right_val.to(self.device), self.
x_ic_val.to(self.device)
34 self.true_ic, self.true_ic_val = self.true_ic.to(self.device
), self.true_ic_val.to(self.device)
35
36 def _model_path(self):
37     if not os.path.exists('checkpoints'):
38         os.mkdir('checkpoints')
39         #path = os.path.join('checkpoints', self.model_name, f'{args
.dim_hidden}_{args.hidden_layers}')
40     path = f'checkpoints/{self.model_name}_{args.dim_hidden}_{args.
hidden_layers}'
41     if not os.path.exists(path):
42         os.mkdir(path)
43     return path
44

```

```

45     def train(self):
46         best_loss = 1.e10
47         for epoch in range(self.epochs_Adam):
48             loss, loss1, loss2, loss3 = self.train_Adam()
49             if (epoch + 1) % 2 == 0:
50                 # self.infos_Adam(epoch + 1, loss, loss1, loss2, loss3)
51                 nums.append(epoch + 1)
52                 losses.append(loss)
53                 valid_loss = self.validate(epoch)
54                 is_best = valid_loss < best_loss
55                 best_loss = valid_loss if is_best else best_loss
56                 state = {
57                     'epoch': epoch,
58                     'state_dict': self.model.state_dict(),
59                     'best_loss': best_loss
60                 }
61                 save_model(state, is_best, save_dir=self.model_path)
62             if (epoch + 1) % 5 == 0:
63                 self.infos_Adam(epoch + 1, loss, loss1, loss2, loss3)
64         for epoch in range(self.epochs_Adam, self.epochs_Adam + self.
epochs_LBFGS):
65             loss, loss1, loss2, loss3 = self.train_LBFGS()
66             if (epoch + 1) % 2 == 0:
67                 # self.infos_LBFGS(epoch + 1, loss, loss1, loss2, loss3)
68                 nums.append(epoch + 1)
69                 losses.append(loss)
70                 valid_loss = self.validate(epoch)
71                 is_best = valid_loss < best_loss
72                 best_loss = valid_loss if is_best else best_loss
73                 state = {
74                     'epoch': epoch,
75                     'state_dict': self.model.state_dict(),
76                     'best_loss': best_loss
77                 }
78                 save_model(state, is_best, save_dir=self.model_path)
79             if (epoch + 1) % 5 == 0:
80                 self.infos_LBFGS(epoch + 1, loss, loss1, loss2, loss3)
81     def train_Adam(self):
82         self.optimizer_Adam.zero_grad()
83         _, _, _, _, f_u, f_v = self.model(self.x)
84         u_bc_left, v_bc_left, ux_bc_left, vx_bc_left, _, _ = self.model(

```

```

self.x_bc_left)
85     u_bc_right, v_bc_right, ux_bc_right, vx_bc_right, _, _ = self.
        model(self.x_bc_right)
86     u_ic, v_ic, _, _, _, _ = self.model(self.x_ic)
87     h_ic=torch.sqrt(u_ic**2+v_ic**2)
88     loss1 = self.criterion(f_u, torch.zeros_like(f_u)) + self.
        criterion(f_v, torch.zeros_like(f_v))
89     loss2 = self.criterion(u_bc_left, u_bc_right) + self.criterion(
        v_bc_left, v_bc_right) + \
90         self.criterion(ux_bc_left, ux_bc_right) + self.criterion
            (vx_bc_left, vx_bc_right)
91     loss3 = self.criterion(u_ic, self.true_ic) + self.criterion(v_ic
        , torch.zeros_like(self.true_ic))
92     loss = loss1 + loss2 + loss3
93     loss.backward()
94     self.optimizer_Adam.step()
95     self.lr_scheduler.step()
96     return loss.item(), loss1.item(), loss2.item(), loss3.item()
97 def infos_Adam(self, epoch, loss, loss1, loss2, loss3):
98     infos = 'Adam ' + \
99         f'Epoch #{epoch:5d}/{self.epochs_Adam + self.
            epochs_LBFGS} ' + \
100         f'Loss: {loss:.4e} = {loss1:.4e} + {loss2:.4e} + {loss3
            :.4e} ' + \
101         f'lr: {self.lr_scheduler.get_lr()[0]:.2e} '
102     print(infos)
103 def train_LBFGS(self):
104     # only used to compute loss_int and loss_bc1 for monitoring
105     _, _, _, _, f_u, f_v = self.model(self.x)
106     u_bc_left, v_bc_left, ux_bc_left, vx_bc_left, _, _ = self.model(
        self.x_bc_left)
107     u_bc_right, v_bc_right, ux_bc_right, vx_bc_right, _, _ = self.
        model(self.x_bc_right)
108     u_ic, v_ic, _, _, _, _ = self.model(self.x_ic)
109     h_ic=torch.sqrt(u_ic**2+v_ic**2)
110     loss1 = self.criterion(f_u, torch.zeros_like(f_u)) + self.
        criterion(f_v, torch.zeros_like(f_v))
111     loss2 = self.criterion(u_bc_left, u_bc_right) + self.criterion(
        v_bc_left, v_bc_right) + \
112         self.criterion(ux_bc_left, ux_bc_right) + self.criterion
            (vx_bc_left, vx_bc_right)

```

```

113     loss3 = self.criterion(u_ic, self.true_ic) + self.criterion(v_ic
    , torch.zeros_like(self.true_ic))
114 # loss3 = self.criterion(h_ic, self.true_ic)
115     loss = loss1 + loss2 + loss3
116     def closure():
117         if torch.is_grad_enabled():
118             self.optimizer_LBFGS.zero_grad()
119             _, _, _, _, f_u, f_v = self.model(self.x)
120             u_bc_left, v_bc_left, ux_bc_left, vx_bc_left, _, _ = self.
    model(self.x_bc_left)
121             u_bc_right, v_bc_right, ux_bc_right, vx_bc_right, _, _ =
    self.model(self.x_bc_right)
122             u_ic, v_ic, _, _, _, _ = self.model(self.x_ic)
123             h_ic = torch.sqrt(u_ic ** 2 + v_ic ** 2)
124             loss1 = self.criterion(f_u, torch.zeros_like(f_u)) + self.
    criterion(f_v, torch.zeros_like(f_v))
125             loss2 = self.criterion(u_bc_left, u_bc_right) + self.
    criterion(v_bc_left, v_bc_right) + \
126                 self.criterion(ux_bc_left, ux_bc_right) + self.
    criterion(vx_bc_left, vx_bc_right)
127             loss3 = self.criterion(u_ic, self.true_ic) + self.criterion(
    v_ic, torch.zeros_like(self.true_ic))
128             # loss3 = self.criterion(h_ic, self.true_ic)
129             loss = loss1 + loss2 + loss3
130             if loss.requires_grad:
131                 loss.backward()
132             return loss
133     self.optimizer_LBFGS.step(closure)
134     loss = closure()
135     return loss.item(), loss1.item(), loss2.item(), loss3
136
137 def infos_LBFGS(self, epoch, loss, loss1, loss2, loss3):
138     infos = 'LBFGS ' + \
139         f'Epoch #{epoch:5d}/{self.epochs_Adam + self.
    epochs_LBFGS} ' + \
140         f'Loss: {loss:.2e} = {loss1:.2e} + {loss2:.2e} + {loss3
    :.2e}'
141     print(infos)
142 def validate(self, epoch):
143     self.model.eval()
144     _, _, _, _, f_u_val, f_v_val = self.model(self.x_val)

```

```

145     u_bc_left_val, v_bc_left_val, ux_bc_left_val, vx_bc_left_val, _,
        _ = self.model(self.x_bc_left_val)
146     u_bc_right_val, v_bc_right_val, ux_bc_right_val, vx_bc_right_val
        , _, _ = self.model(self.x_bc_right_val)
147     u_ic_val, v_ic_val, _, _, _, _ = self.model(self.x_ic_val)
148     h_ic_val = torch.sqrt(u_ic_val ** 2 + v_ic_val ** 2)
149     loss1 = self.criterion(f_u_val, torch.zeros_like(f_u_val)) +
        self.criterion(f_v_val, torch.zeros_like(f_v_val))
150     loss2 = self.criterion(u_bc_left_val, u_bc_right_val) + self.
        criterion(v_bc_left_val, v_bc_right_val) + \
151             self.criterion(ux_bc_left_val, ux_bc_right_val) + self.
        criterion(vx_bc_left_val, vx_bc_right_val)
152     loss3 = self.criterion(u_ic_val, self.true_ic) + self.criterion(
        v_ic_val, torch.zeros_like(self.true_ic))
153     # loss3 = self.criterion(h_ic_val, self.true_ic_val)
154     loss = loss1 + loss2 + loss3
155     infos = 'Valid ' + \
156             f'Epoch #{epoch + 1:5d}/{self.epochs_Adam + self.
        epochs_LBFGS} ' + \
157             f'Loss: {loss:.4e} '
158     # print(infos)
159     self.model.train()
160     return loss.item()

```

## A.5 验证过程

```

1 class Tester(object):
2     def __init__(self, args):
3         self.device = args.device
4         self.problem = args.problem
5         self.criterion = nn.MSELoss()
6         self.model = args.model
7         model_name = self.model.__class__.__name__
8         # model_path = os.path.join('checkpoints',
9         # model_name,
10        # 'best_model.pth.tar')
11        model_path = f'checkpoints/{model_name}_{args.dim_hidden}_{args.
        hidden_layers}/best_model.pth.tar'
12        best_model = torch.load(model_path)
13        self.model.load_state_dict(best_model['state_dict'])
14        self.model.to(self.device)

```



```

15         self.X, self.x= args.testset(verbose='tensor')
16         if self.device == torch.device(type='cuda'):
17             self.X = self.X.to(self.device)
18     def predict(self,t):
19         self.model.eval()
20         u,v,_,_,_,_ = self.model(self.X)
21         u = u.detach().cpu().numpy()
22         u = u.reshape(self.x.shape)
23         v = v.detach().cpu().numpy()
24         v = v.reshape(self.x.shape)
25         h=np.sqrt(u**2+v**2)
26         truesln=self.problem.true_value(t)
27         fig, axes = plt.subplots()
28         axes.plot(self.x,h,label='Prediction')
29         axes.plot(self.x,truesln,'--',label='Exact Solution')
30         axes.legend()
31         axes.set_title(f'Time = {t}')
32         axes.set_xlabel('x')
33         axes.set_ylabel('|h(t,x)|')
34         fig.savefig(f't_{t:4f}.png')
35         plt.show()
36     def pred_result(self,t):
37         self.model.eval()
38         u,v,_,_,_,_ = self.model(self.X)
39         u = u.detach().cpu().numpy()
40         u = u.reshape(self.x.shape)
41         v = v.detach().cpu().numpy()
42         v = v.reshape(self.x.shape)
43         h=np.sqrt(u**2+v**2)
44         return self.x, h

```

## A.6 PINN 与 ResPINN 对比

```

1 %time
2 args = Options().parse()
3 args.problem = Problem()
4 args.model = PINN(2, 2, dim_hidden=args.dim_hidden, hidden_layers=args.
    hidden_layers)
5 args.trainset = Trainset(200,100,args.n_bc,args.n_ic,method='uniform')
6 args.validset = Trainset(args.n_x, args.n_t, args.n_bc, args.n_ic,
    method='uniform')

```

```

7  nums = []
8  losses=[]
9  trainer = Trainer(args)
10 trainer.train()
11 loss_path=f'checkpoints/{args.model.__class__.__name__}_{args.dim_hidden
    }_{args.hidden_layers}/loss.txt'
12 np.savetxt(loss_path, np.vstack((nums, losses)).T)
13 fig, axes = plt.subplots(figsize=(8, 8))
14 axes.semilogy(nums, losses, 'r--')
15 axes.set_title('Loss')
16 axes.set_xlabel('Epoch')
17 axes.set_ylabel('Loss')
18 plt.savefig('loss.png')
19
20 %time
21 args = Options().parse()
22 args.problem = Problem()
23 # args.model = PINN(dim_in=2,
24 # dim_out=1,
25 # dim_hidden=args.dim_hidden,
26 # hidden_layers=args.hidden_layers,
27 # act_name='sin',
28 # dropout=args.dropout)
29 args.model = ResPINN(2, 2, dim_hidden=args.dim_hidden, res_blocks=args.
    hidden_layers)
30 # args.trainset = Trainset(args.n, args.n_bc, args.n_ic, method='lhs')
31 args.trainset = Trainset(200, 100, args.n_bc, args.n_ic, method='uniform
    ')
32 args.validset = Trainset(args.n_x, args.n_t, args.n_bc, args.n_ic,
    method='uniform')
33 nums = []
34 losses = []
35 trainer = Trainer(args)
36 trainer.train()
37 loss_path=f'checkpoints/{args.model.__class__.__name__}_{args.dim_hidden
    }_{args.hidden_layers}/loss.txt'
38 np.savetxt(loss_path, np.vstack((nums, losses)).T)
39 fig, axes = plt.subplots(figsize=(8, 8))
40 axes.semilogy(nums, losses, 'r')
41 axes.set_title('Loss')
42 axes.set_xlabel('Epoch')

```

```

43 axes.set_ylabel('Loss')
44 plt.savefig('loss.png')
45 plt.show()
46
47 data1=np.loadtxt('checkpoints/PINN_50_4/loss.txt')
48 data2=np.loadtxt('checkpoints/ResPINN_50_4/loss.txt')
49 x=data1[:,0]
50 loss1=data1[:,1]
51 loss2=data2[:,1]
52 fig, axes = plt.subplots(figsize=(8, 8))
53 axes.semilogy(x, loss1, label='PINN with 4 layers 50 hidden dims')
54 axes.semilogy(x, loss2, label='ResPINN with 4 res_blocks 50 hidden dims'
55 )
56 axes.legend()
57 axes.set_title('PINN vs ResPINN Loss')
58 axes.set_xlabel('Epoch')
59 axes.set_ylabel('Loss')

```

## A.7 不同的隐藏层数目对结果的影响

```

1 %time
2 args = Options().parse()
3 args.problem = Problem()
4 # args.model = PINN(dim_in=2,
5 # dim_out=1,
6 # dim_hidden=args.dim_hidden,
7 # hidden_layers=args.hidden_layers,
8 # act_name='sin',
9 # dropout=args.dropout)
10 args.dim_hidden=10
11 args.model = PINN(2, 2, dim_hidden=args.dim_hidden, hidden_layers=args.
12     hidden_layers)
13 # args.trainset = Trainset(args.n, args.n_bc, args.n_ic, method='lhs')
14 args.trainset = Trainset(200, 100, args.n_bc, args.n_ic, method='uniform
15 ')
16 args.validset = Trainset(args.n_x, args.n_t, args.n_bc, args.n_ic,
17     method='uniform')
18 nums = []
19 losses = []
20 trainer = Trainer(args)
21 trainer.train()

```

```

19 loss_path = f'checkpoints/{args.model.__class__.__name__}_{args.
    dim_hidden}_{args.hidden_layers}/loss.txt'
20 np.savetxt(loss_path, np.vstack((nums, losses)).T)
21 fig, axes = plt.subplots(figsize=(8, 8))
22 axes.semilogy(nums, losses, 'r--')
23 axes.set_title('Loss')
24 axes.set_xlabel('Epoch')
25 axes.set_ylabel('Loss')
26 plt.savefig('loss.png')
27
28 args = Options().parse()
29 args.problem = Problem()
30 # args.model = PINN(dim_in=2,
31 # dim_out=1,
32 # dim_hidden=args.dim_hidden,
33 # hidden_layers=args.hidden_layers,
34 # act_name='sin',
35 # dropout=args.dropout)
36 args.dim_hidden=35
37 args.model = PINN(2, 2, dim_hidden=args.dim_hidden, hidden_layers=args.
    hidden_layers)
38 # args.trainset = Trainset(args.n, args.n_bc, args.n_ic, method='lhs')
39 args.trainset = Trainset(200, 100, args.n_bc, args.n_ic, method='uniform
    ')
40 args.validset = Trainset(args.n_x, args.n_t, args.n_bc, args.n_ic,
    method='uniform')
41 nums = []
42 losses = []
43 trainer = Trainer(args)
44 trainer.train()
45 loss_path = f'checkpoints/{args.model.__class__.__name__}_{args.
    dim_hidden}_{args.hidden_layers}/loss.txt'
46 np.savetxt(loss_path, np.vstack((nums, losses)).T)
47 fig, axes = plt.subplots(figsize=(8, 8))
48 axes.semilogy(nums, losses, 'r--')
49 axes.set_title('Loss')
50 axes.set_xlabel('Epoch')
51 axes.set_ylabel('Loss')
52 plt.savefig('loss.png')
53
54 args = Options().parse()

```

```

55 args.problem = Problem()
56 # args.model = PINN(dim_in=2,
57 # dim_out=1,
58 # dim_hidden=args.dim_hidden,
59 # hidden_layers=args.hidden_layers,
60 # act_name='sin',
61 # dropout=args.dropout)
62 args.dim_hidden=50
63 args.model = PINN(2, 2, dim_hidden=args.dim_hidden, hidden_layers=args.
    hidden_layers)
64 # args.trainset = Trainset(args.n, args.n_bc, args.n_ic, method='lhs')
65 args.trainset = Trainset(200, 100, args.n_bc, args.n_ic, method='uniform
    ')
66 args.validset = Trainset(args.n_x, args.n_t, args.n_bc, args.n_ic,
    method='uniform')
67 nums = []
68 losses = []
69 trainer = Trainer(args)
70 trainer.train()
71 loss_path = f'checkpoints/{args.model.__class__.__name__}_{args.
    dim_hidden}_{args.hidden_layers}/loss.txt'
72 np.savetxt(loss_path, np.vstack((nums, losses)).T)
73 fig, axes = plt.subplots(figsize=(8, 8))
74 axes.semilogy(nums, losses, 'r--')
75 axes.set_title('Loss')
76 axes.set_xlabel('Epoch')
77 axes.set_ylabel('Loss')
78 plt.savefig('loss.png')
79
80 data1=np.loadtxt('checkpoints/PINN_20_4/loss.txt')
81 data2=np.loadtxt('checkpoints/PINN_35_4/loss.txt')
82 data3=np.loadtxt('checkpoints/PINN_50_4/loss.txt')
83 x=data1[:,0]
84 loss1=data1[:,1]
85 loss2=data2[:,1]
86 loss3=data3[:,1]
87 fig, axes = plt.subplots(figsize=(8, 8))
88 axes.semilogy(x, loss1, label='PINN with 4 layers 10 hidden dims')
89 axes.semilogy(x, loss2, label='PINN with 4 layers 50 hidden dims')
90 axes.semilogy(x, loss3, label='PINN with 4 layers 100 hidden dims')
91 axes.legend()

```

```

92 axes.set_title('Comparison of different hidden dimensions')
93 axes.set_xlabel('Epoch')
94 axes.set_ylabel('Loss')

```

## A.8 隐藏层的不同神经元数目对结果的影响-PINN.py

```

1 #hidden=20
2
3 %time
4 args = Options().parse()
5 args.problem = Problem()
6 # args.model = PINN(dim_in=2,
7 # dim_out=1,
8 # dim_hidden=args.dim_hidden,
9 # hidden_layers=args.hidden_layers,
10 # act_name='sin',
11 # dropout=args.dropout)
12 args.dim_hidden=20
13 args.model = PINN(2, 2, dim_hidden=args.dim_hidden, hidden_layers=args.
    hidden_layers)
14 # args.trainset = Trainset(args.n, args.n_bc, args.n_ic, method='lhs')
15 args.trainset = Trainset(200, 100, args.n_bc, args.n_ic, method='uniform
    ')
16 args.validset = Trainset(args.n_x, args.n_t, args.n_bc, args.n_ic,
    method='uniform')
17 nums = []
18 losses = []
19 trainer = Trainer(args)
20 trainer.train()
21 loss_path = f'checkpoints/{args.model.__class__.__name__}_{args.
    dim_hidden}_{args.hidden_layers}/loss.txt'
22 np.savetxt(loss_path, np.vstack((nums, losses)).T)
23 fig, axes = plt.subplots(figsize=(8, 8))
24 axes.semilogy(nums, losses, 'r--')
25 axes.set_title('Loss')
26 axes.set_xlabel('Epoch')
27 axes.set_ylabel('Loss')
28 plt.savefig('loss.png')
29
30 #hidden=35
31

```

```

32 args = Options().parse()
33 args.problem = Problem()
34 # args.model = PINN(dim_in=2,
35 # dim_out=1,
36 # dim_hidden=args.dim_hidden,
37 # hidden_layers=args.hidden_layers,
38 # act_name='sin',
39 # dropout=args.dropout)
40 args.dim_hidden=35
41 args.model = PINN(2, 2, dim_hidden=args.dim_hidden, hidden_layers=args.
    hidden_layers)
42 # args.trainset = Trainset(args.n, args.n_bc, args.n_ic, method='lhs')
43 args.trainset = Trainset(200, 100, args.n_bc, args.n_ic, method='uniform
    ')
44 args.validset = Trainset(args.n_x, args.n_t, args.n_bc, args.n_ic,
    method='uniform')
45 nums = []
46 losses = []
47 trainer = Trainer(args)
48 trainer.train()
49 loss_path = f'checkpoints/{args.model.__class__.__name__}_{args.
    dim_hidden}_{args.hidden_layers}/loss.txt'
50 np.savetxt(loss_path, np.vstack((nums, losses)).T)
51 fig, axes = plt.subplots(figsize=(8, 8))
52 axes.semilogy(nums, losses, 'r--')
53 axes.set_title('Loss')
54 axes.set_xlabel('Epoch')
55 axes.set_ylabel('Loss')
56 plt.savefig('loss.png')
57
58 #hidden=50
59
60 args = Options().parse()
61 args.problem = Problem()
62 # args.model = PINN(dim_in=2,
63 # dim_out=1,
64 # dim_hidden=args.dim_hidden,
65 # hidden_layers=args.hidden_layers,
66 # act_name='sin',
67 # dropout=args.dropout)
68 args.dim_hidden=50

```

```

69 args.model = PINN(2, 2, dim_hidden=args.dim_hidden, hidden_layers=args.
    hidden_layers)
70 # args.trainset = Trainset(args.n, args.n_bc, args.n_ic, method='lhs')
71 args.trainset = Trainset(200, 100, args.n_bc, args.n_ic, method='uniform
    ')
72 args.validset = Trainset(args.n_x, args.n_t, args.n_bc, args.n_ic,
    method='uniform')
73 nums = []
74 losses = []
75 trainer = Trainer(args)
76 trainer.train()
77 loss_path = f'checkpoints/{args.model.__class__.__name__}_{args.
    dim_hidden}_{args.hidden_layers}/loss.txt'
78 np.savetxt(loss_path, np.vstack((nums, losses)).T)
79 fig, axes = plt.subplots(figsize=(8, 8))
80 axes.semilogy(nums, losses, 'r--')
81 axes.set_title('Loss')
82 axes.set_xlabel('Epoch')
83 axes.set_ylabel('Loss')
84 plt.savefig('loss.png')
85
86
87 data1=np.loadtxt('checkpoints/PINN_20_4/loss.txt')
88 data2=np.loadtxt('checkpoints/PINN_35_4/loss.txt')
89 data3=np.loadtxt('checkpoints/PINN_50_4/loss.txt')
90 x=data1[:,0]
91 loss1=data1[:,1]
92 loss2=data2[:,1]
93 loss3=data3[:,1]
94 fig, axes = plt.subplots(figsize=(8, 8))
95 axes.semilogy(x, loss1, label='PINN with 4 layers 10 hidden dims')
96 axes.semilogy(x, loss2, label='PINN with 4 layers 50 hidden dims')
97 axes.semilogy(x, loss3, label='PINN with 4 layers 100 hidden dims')
98 axes.legend()
99 axes.set_title('Comparison of different hidden dimensions')
100 axes.set_xlabel('Epoch')
101 axes.set_ylabel('Loss')

```

## A.9 隐藏层的不同神经元数目对结果的影响-ResPINN

```
1 %time
```



```

2 args = Options().parse()
3 args.problem = Problem()
4 # args.model = PINN(dim_in=2,
5 # dim_out=1,
6 # dim_hidden=args.dim_hidden,
7 # hidden_layers=args.hidden_layers,
8 # act_name='sin',
9 # dropout=args.dropout)
10 args.dim_hidden=20
11 args.model = ResPINN(2, 2, dim_hidden=args.dim_hidden, res_blocks=args.
    hidden_layers)
12 # args.trainset = Trainset(args.n, args.n_bc, args.n_ic, method='lhs')
13 args.trainset = Trainset(200, 100, args.n_bc, args.n_ic, method='uniform
    ')
14 args.validset = Trainset(args.n_x, args.n_t, args.n_bc, args.n_ic,
    method='uniform')
15 nums = []
16 losses = []
17 trainer = Trainer(args)
18 trainer.train()
19 loss_path = f'checkpoints/{args.model.__class__.__name__}_{args.
    dim_hidden}_{args.hidden_layers}/loss.txt'
20 np.savetxt(loss_path, np.vstack((nums, losses)).T)
21 fig, axes = plt.subplots(figsize=(8, 8))
22 axes.semilogy(nums, losses, 'r--')
23 axes.set_title('Loss',fontsize=20)
24 axes.set_xlabel('Epoch',fontsize=20)
25 axes.set_ylabel('Loss',fontsize=20)
26 plt.savefig('loss.png')
27
28 hidden=35
29
30 args = Options().parse()
31 args.problem = Problem()
32 # args.model = PINN(dim_in=2,
33 # dim_out=1,
34 # dim_hidden=args.dim_hidden,
35 # hidden_layers=args.hidden_layers,
36 # act_name='sin',
37 # dropout=args.dropout)
38 args.dim_hidden=35

```

```

39 args.model = ResPINN(2, 2, dim_hidden=args.dim_hidden, res_blocks=args.
    hidden_layers)
40 # args.trainset = Trainset(args.n, args.n_bc, args.n_ic, method='lhs')
41 args.trainset = Trainset(200, 100, args.n_bc, args.n_ic, method='uniform
    ')
42 args.validset = Trainset(args.n_x, args.n_t, args.n_bc, args.n_ic,
    method='uniform')
43 nums = []
44 losses = []
45 trainer = Trainer(args)
46 trainer.train()
47 loss_path = f'checkpoints/{args.model.__class__.__name__}_{args.
    dim_hidden}_{args.hidden_layers}/loss.txt'
48 np.savetxt(loss_path, np.vstack((nums, losses)).T)
49 fig, axes = plt.subplots(figsize=(8, 8))
50 axes.semilogy(nums, losses, 'r--')
51 axes.set_title('Loss',fontsize=20)
52 axes.set_xlabel('Epoch',fontsize=20)
53 axes.set_ylabel('Loss',fontsize=20)
54 plt.savefig('loss.png')
55
56 hidden=50
57
58 args = Options().parse()
59 args.problem = Problem()
60 # args.model = PINN(dim_in=2,
61 # dim_out=1,
62 # dim_hidden=args.dim_hidden,
63 # hidden_layers=args.hidden_layers,
64 # act_name='sin',
65 # dropout=args.dropout)
66 args.dim_hidden=50
67 args.model = ResPINN(2, 2, dim_hidden=args.dim_hidden, res_blocks=args.
    hidden_layers)
68 # args.trainset = Trainset(args.n, args.n_bc, args.n_ic, method='lhs')
69 args.trainset = Trainset(200, 100, args.n_bc, args.n_ic, method='uniform
    ')
70 args.validset = Trainset(args.n_x, args.n_t, args.n_bc, args.n_ic,
    method='uniform')
71 nums = []
72 losses = []

```

```

73 trainer = Trainer(args)
74 trainer.train()
75 loss_path = f'checkpoints/{args.model.__class__.__name__}_{args.
    dim_hidden}_{args.hidden_layers}/loss.txt'
76 np.savetxt(loss_path, np.vstack((nums, losses)).T)
77 fig, axes = plt.subplots(figsize=(8, 8))
78 axes.semilogy(nums, losses, 'r--')
79 axes.set_title('Loss',fontsize=20)
80 axes.set_xlabel('Epoch',fontsize=20)
81 axes.set_ylabel('Loss',fontsize=20)
82 plt.savefig('loss.png')
83
84 data1=np.loadtxt('checkpoints/ResPINN_20_4/loss.txt')
85 data2=np.loadtxt('checkpoints/ResPINN_35_4/loss.txt')
86 data3=np.loadtxt('checkpoints/ResPINN_50_4/loss.txt')
87 x=data1[:,0]
88 loss1=data1[:,1]
89 loss2=data2[:,1]
90 loss3=data3[:,1]
91 fig, axes = plt.subplots(figsize=(8, 8))
92 axes.semilogy(x, loss1, label='ResPINN with 4 layers 20 hidden dims')
93 axes.semilogy(x, loss2, label='ResPINN with 4 layers 35 hidden dims')
94 axes.semilogy(x, loss3, label='ResPINN with 4 layers 50 hidden dims')
95 axes.legend()
96 axes.set_title('Comparison of different hidden dimensions',fontsize=20)
97 axes.set_xlabel('Epoch',fontsize=20)
98 axes.set_ylabel('Loss',fontsize=20)

```

## B LSTM 预测股票涨跌代码

### B.1 主体代码

单向 LSTM 网络

```
1 #!/usr/bin/env python
2 import numpy as np
3 import pandas as pd
4 import torch
5 import matplotlib.pyplot as plt
6 from torch.autograd import Variable
7 import torch.utils.data as Data
8 import random
9
10 # split a univariate sequence into samples
11 def split_sequence(sequence, n_steps):
12     X, y = [], []
13     for i in range(len(sequence)):
14         # find the end of this pattern
15         end_idx = i + n_steps
16         # check if we are beyond the sequence
17         if end_idx > len(sequence) - 1:
18             break
19         # gather input and output parts if the pattern
20         seq_x, seq_y = sequence[i:end_idx], sequence[end_idx]
21         X.append(seq_x)
22         y.append(seq_y)
23     return np.array(X), np.array(y)
24
25 #!/usr/bin/env python
26 import torch
27 import torch.nn as nn
28 from torch.nn import functional as F
29 from torch import optim
30
31 import numpy as np
32 import pandas as pd
33 import matplotlib.pyplot as plt
34
35
36 data0 = pd.read_excel("D:/mathematical experiment/code/code/2011-2020
```

```

    price.xlsx")
37 col_name = list(data0.columns)
38 data = data0[col_name[1:]]
39 index = data0[col_name[0]]
40
41 # plt.plot(data)
42 # plt.show()
43
44
45 dataset = data.dropna().values.astype('float32')
46 dataset = dataset.reshape(-1,1)
47
48 print(dataset.shape)
49
50 def centralize(data):
51     min_value = np.min(data,axis=0)
52     max_value = np.max(data,axis=0)
53     data = (data - min_value) / (max_value-min_value)
54     return data
55
56 def judge(dataset,k):
57     '''
58     to see at day k, if each stock rise or fall
59     '''
60     pr_today = dataset[k]
61     pr_yesterday = dataset[k-1]
62     pr_change = pr_today - pr_yesterday
63     med = np.median(pr_change)
64     re = np.zeros_like(pr_change)
65     re[pr_change>0] = 1
66     return(re)
67
68 def acc(out,y_real):
69     out1 = np.zeros_like(out)
70     out1[out>0] = 1
71     out1[out<=0] = 0
72     return 1-np.sum(np.sum(np.abs(y_real-out1)))/(np.prod(y_real.shape))
73
74
75 def create_dataset(dataset,look_back=240):
76     dataX,dataY=[],[]

```

```

77     for i in range(len(dataset)-look_back):
78         pr_change = judge(dataset,i+look_back)
79         a = dataset[i:(i+look_back)]
80         dataX.append(a)
81         dataY.append(pr_change)
82     return np.array(dataX),np.array(dataY)
83
84 def set_seed(seed):
85     torch.manual_seed(seed) # cpu 为CPU设置种子用于生成随机数，以使得结
      果是确定的
86     torch.cuda.manual_seed(seed) # gpu 为当前GPU设置随机种子
87     torch.backends.cudnn.deterministic = True # cudnn
88     np.random.seed(seed) # numpy
89     random.seed(seed)
90
91 look_back = 240
92 index_used = index[look_back:]
93 index_used = np.array(index_used)
94 X, Y = create_dataset(dataset,look_back)
95 print(X.shape, Y.shape)
96 a,b,c = X.shape
97
98 train_size = int(len(X) * 0.9)
99 valid_size = len(X) - train_size
100 index_size = int(len(index_used)*0.9)
101 print(train_size, valid_size)
102
103 X_train = X[:train_size]
104 Y_train = Y[:train_size]
105 index_train = index_used[:index_size]
106
107
108 X_valid = X[train_size:]
109 Y_valid = Y[train_size:]
110 index_valid = index_used[index_size:]
111
112
113 # X_train = X_train.reshape(-1,198,240)
114 # X_valid = X_valid.reshape(-1,198,240)
115 # Y_train = Y_train.reshape(-1,198,1)
116

```

```

117 X_train = X_train.reshape(train_size*c,b,1)
118 Y_train = Y_train.reshape(train_size*c,1)
119 X_valid = X_valid.reshape(valid_size*c,b,1)
120 Y_valid = Y_valid.reshape(valid_size*c,1)
121
122 # X_train = X_train.transpose(1, 0, 2)
123 # X_valid = X_valid.transpose(1, 0, 2)
124
125 X_train = torch.from_numpy(X_train)
126 Y_train = torch.from_numpy(Y_train)
127 X_valid = torch.from_numpy(X_valid)
128
129 print(X_train.shape,Y_train.shape)
130
131
132
133
134 index_train = index[:index_size]
135 index_valid = index[int(len(index)*0.9):]
136
137 index_valid.shape
138
139 X.shape
140
141 %%time
142 class LSTMRegression(nn.Module):
143     def __init__(self, input_size, hidden_size, output_size=1,
144                 num_layers=1):
145         super().__init__()
146         self.lstm = nn.LSTM(input_size, hidden_size, num_layers,
147                             batch_first=True)
148         self.linear = nn.Linear(hidden_size, output_size)
149
150     def forward(self, x):
151         _, (hn, cn) = self.lstm(x)
152         hn = hn.squeeze()
153         out = self.linear(hn)
154         return out
155
156 model = LSTMRegression(input_size=1, hidden_size=5, output_size=1)

```

```

156 criterion = torch.nn.BCEWithLogitsLoss()      #交叉熵BCEWithLogitsLoss()
    和MultiLabelSoftMarginLoss()
157 #criterion = torch.nn.CrossEntropyLoss()
158 optimizer = optim.Adam(model.parameters(), lr=1e-3)
159 #optimizer = optim.SGD(model.parameters(), lr=1e-1)
160
161 epochs = 100
162 batch_size = 30
163 batch = X_train.shape[0] // batch_size
164
165
166
167 torch_dataset = Data.TensorDataset(torch.tensor(X_train), torch.tensor(
    Y_train))
168 # 把 dataset 放入 DataLoader
169 loader = Data.DataLoader(
170     dataset=torch_dataset, # torch TensorDataset format
171     batch_size=batch_size, # mini batch size
172     shuffle=True, #
173     num_workers=10, # 多线程来读数据
174 )
175
176 loss_epoch = np.zeros(epochs)
177 acc_epoch = np.zeros(epochs)
178 loss_valid = np.zeros(epochs)
179 acc_valid = np.zeros(epochs)
180 for epoch in range(epochs):
181     loss_ep = np.array([])
182     acc_ep = np.array([])
183     loss_epv = np.array([])
184     acc_epv = np.array([])
185     for step,(var_x,var_y) in enumerate(loader):
186         out = model(var_x)
187         out_f = out.detach().clone().numpy()
188         var_yf = var_y.detach().clone().numpy()
189         loss = criterion(out, var_y)
190         loss_f = loss.detach().clone().numpy()
191         acc_ep = np.append(acc_ep,acc(out_f,var_yf))
192         loss_ep = np.append(loss_ep,loss_f)
193
194     optimizer.zero_grad()

```



```

195         loss.backward()
196         optimizer.step()
197
198     if (epoch + 1) % 5 == 0:
199         print(f'Epoch: {epoch:5d}, Loss: {np.mean(loss_ep):.4e}, Acc:{np
200             .mean(acc_ep):.4e}')
201
202     loss_epoch[epoch] = np.mean(loss_ep)
203     acc_epoch[epoch] = np.mean(acc_ep)
204
205     Y_pre = model(X_valid) #计算验证集表现
206     Y_pre1 = Y_pre.clone().detach().numpy()
207     Y_valid1 = torch.from_numpy(Y_valid)
208     loss_valid[epoch] = criterion(Y_pre,Y_valid1)
209     acc_valid[epoch] = acc(Y_pre1,Y_valid)
210
211
212
213 # test
214 #X = X.reshape(-1,198,240)
215 #X = torch.from_numpy(X_valid)
216 Y_pred = model(X_valid)
217 Y_pred = Y_pred.clone().detach().numpy()
218 pred_acc = acc(Y_pred,Y_valid)
219 #Y_pred = Y_pred.view(-1).data.numpy()
220
221 # visulize
222 kind = 2
223 series = np.arange(kind*len(index_valid),(kind+1)*len(index_valid))
224 Y_pred_re = Y_pred
225 Y_pred_re[Y_pred_re>0] = 1
226 Y_pred_re[Y_pred_re<=0] = 0
227
228
229 fig = plt.figure()
230 ax = plt.subplot()
231 type1 = ax.scatter(index_valid, Y_valid[series], alpha=0.5,color='b',
232     label='groundtruth')
233 type2 = ax.scatter(index_valid, Y_pred_re[series], alpha=0.3,color='r',
234     label='prediction')

```

```

233 plt.xlabel("date time")
234 plt.ylabel("0 for fall, 1 for rise")
235 ax.legend((type1, type2), (u'groundtruth', u'prediction'), loc='best')
236 plt.show()
237
238 plt.plot(acc_valid, 'r-', label='validation acc')
239 plt.plot(acc_epoch, 'b-', label='prediction acc')
240 plt.legend(loc='best')
241 plt.savefig("acc.png")
242 plt.show()
243
244 plt.plot(loss_valid, 'r-', label='validation loss')
245 plt.plot(loss_epoch, 'b-', label='prediction loss')
246 plt.legend(loc='best')
247 plt.savefig("loss.png")
248 plt.show()

```

## 双向 LSTM 网络

```

1  #!/usr/bin/env python
2  import numpy as np
3  import pandas as pd
4  import matplotlib.pyplot as plt
5  from torch.autograd import Variable
6  import torch.utils.data as Data
7  import datetime
8  import random
9
10 # split a univariate sequence into samples
11 def split_sequence(sequence, n_steps):
12     X, y = [], []
13     for i in range(len(sequence)):
14         # find the end of this pattern
15         end_idx = i + n_steps
16         # check if we are beyond the sequence
17         if end_idx > len(sequence) - 1:
18             break
19         # gather input and output parts if the pattern
20         seq_x, seq_y = sequence[i:end_idx], sequence[end_idx]
21         X.append(seq_x)
22         y.append(seq_y)
23     return np.array(X), np.array(y)

```

```

24
25 #!/usr/bin/env python
26 import torch
27 import torch.nn as nn
28 from torch.nn import functional as F
29 from torch import optim
30
31 import numpy as np
32 import pandas as pd
33 import matplotlib.pyplot as plt
34
35 data0 = pd.read_excel("D:/mathematical experiment/code/code/2011-2020
    price.xlsx")
36 col_name = list(data0.columns)
37 data = data0[col_name[1:]]
38 index = data0[col_name[0]]
39
40 # plt.plot(data)
41 # plt.show()
42
43
44 dataset = data.dropna().values.astype('float32')
45
46 # max_value = np.max(dataset,axis=0)
47 # min_value = np.min(dataset,axis=0)
48 # dataset = (dataset - min_value) / (max_value-min_value)
49 print(dataset.shape)
50
51 data0 = pd.read_excel("D:/mathematical experiment/code/code/2011-2020
    price.xlsx")
52 col_name = list(data0.columns)
53 data = data0[col_name[1:]]
54 index = data0[col_name[0]]
55
56 # plt.plot(data)
57 # plt.show()
58
59
60 dataset = data.dropna().values.astype('float32')
61
62 # max_value = np.max(dataset,axis=0)

```

```

63 # min_value = np.min(dataset,axis=0)
64 # dataset = (dataset - min_value) / (max_value-min_value)
65 print(dataset.shape)
66
67 def centralize(data):
68     min_value = np.min(data,axis=0)
69     max_value = np.max(data,axis=0)
70     data = (data - min_value) / (max_value-min_value)
71     return data
72
73 def judge(dataset,k):
74     '''
75     to see at day k, if each stock rise or fall
76     '''
77     pr_today = dataset[k]
78     pr_yesterday = dataset[k-1]
79     pr_change = pr_today - pr_yesterday
80     med = np.median(pr_change)
81     re = np.zeros_like(pr_change)
82     re[pr_change>0] = 1
83     return(re)
84
85 def acc(out,y_real):
86     #y_real = y_real.detach().numpy()
87     #out = out.detach().numpy()
88     out1 = np.zeros_like(out)
89     out1[out>0] = 1
90     out1[out<=0] = 1
91     return 1-np.sum(np.sum(np.sum(np.abs(y_real-out1))))/(np.prod(y_real
    .shape))
92
93
94 def create_dataset(dataset,look_back=240):
95     dataX,dataY=[],[]
96     for i in range(len(dataset)-look_back):
97         pr_change = judge(dataset,i+look_back)
98         a = dataset[i:(i+look_back)]
99         dataX.append(a)
100         dataY.append(pr_change)
101     return centralize(np.array(dataX)),np.array(dataY)
102

```

```

103 def set_seed(seed):
104     torch.manual_seed(seed) # cpu 为CPU设置种子用于生成随机数, 以使得结
        果是确定的
105     torch.cuda.manual_seed(seed) # gpu 为当前GPU设置随机种子
106     torch.backends.cudnn.deterministic = True # cudnn
107     np.random.seed(seed) # numpy
108     random.seed(seed)
109
110 look_back = 240
111 index = index[look_back:]
112 index = np.array(index)
113 X, Y = create_dataset(dataset, look_back)
114 print(X.shape, Y.shape)
115
116 train_size = int(len(X) * 0.7)
117 valid_size = len(X) - train_size
118 print(train_size, valid_size)
119
120 X_train = X[:train_size]
121 Y_train = Y[:train_size]
122 index_train = index[:train_size]
123
124 X_valid = X[train_size:]
125 Y_valid = Y[train_size:]
126 index_valid = index[train_size:]
127
128 X_train = X_train.reshape(-1, 198, 240)
129 X_valid = X_valid.reshape(-1, 198, 240)
130 Y_train = Y_train.reshape(-1, 198, 1)
131
132 # X_train = X_train.transpose(1, 0, 2)
133 # X_valid = X_valid.transpose(1, 0, 2)
134
135 X_train = torch.from_numpy(X_train)
136 Y_train = torch.from_numpy(Y_train)
137 X_valid = torch.from_numpy(X_valid)
138
139 print(X_train.shape, Y_train.shape)
140
141
142 %%time

```

```

143 class LSTMRegression(nn.Module):
144     def __init__(self, input_size, hidden_size, output_size=1,
145                 num_layers=2, bidirectional=True):
146         super().__init__()
147         self.lstm = nn.LSTM(input_size, hidden_size, num_layers,
148                             batch_first=True, bidirectional=True)
149         self.linear = nn.Linear(2*hidden_size, output_size)
150
151     def forward(self, x):
152         x, _ = self.lstm(x) # (seq, batch, hidden)
153         s, b, h = x.shape
154         # print(s,b)
155         x = x.contiguous().view(s*b, h) # 转换成线性层的输入格式
156         x = self.linear(x)
157         x = x.view(s, b, -1)
158         return x
159
160 torch.manual_seed(7) #cpu
161 torch.cuda.manual_seed(7) #gpu
162
163 np.random.seed(7) #numpy
164 random.seed(7) # random and transforms
165 torch.backends.cudnn.deterministic=True #cudnn
166
167 model = LSTMRegression(input_size=240, hidden_size=4, output_size=1)
168
169 criterion = torch.nn.BCEWithLogitsLoss() #交叉熵BCEWithLogitsLoss()
170 #criterion = torch.nn.CrossEntropyLoss()
171 #optimizer = optim.Adam(model.parameters(), lr=1e-3)
172 #optimizer = optim.SGD(model.parameters(), lr=1e-1)
173
174 epochs = 100
175 batch_size = 40
176 batch = X_train.shape[0] // batch_size
177
178 torch_dataset = Data.TensorDataset(torch.tensor(X_train), torch.tensor(
179     Y_train))
180 # 把 dataset 放入 DataLoader

```

```

180 loader = Data.DataLoader(
181     dataset=torch_dataset, # torch TensorDataset format
182     batch_size=batch_size, # mini batch size
183     shuffle=True, #
184     num_workers=2, # 多线程来读数据
185 )
186
187 loss_epoch = np.zeros(epochs)
188 acc_epoch = np.zeros(epochs)
189 loss_valid = np.zeros(epochs)
190 acc_valid = np.zeros(epochs)
191 for epoch in range(epochs):
192     acc_epo = 0
193     loss_ep = np.array([])
194     acc_ep = np.array([])
195     loss_epv = np.array([])
196     acc_epv = np.array([])
197     for step, (var_x, var_y) in enumerate(loader):
198         out = model(var_x)
199         out_f = out.detach().clone().numpy()
200         var_yf = var_y.detach().clone().numpy()
201         loss = criterion(out, var_y)
202         loss_f = loss.detach().clone().numpy()
203         acc_ep = np.append(acc_ep, acc(out_f, var_yf))
204         loss_ep = np.append(loss_ep, loss_f)
205
206         optimizer.zero_grad()
207         loss.backward()
208         optimizer.step()
209
210     if (epoch + 1) % 10 == 0:
211         #print(f'Epoch: {epoch:5d}, Loss: {loss.item():.4e}, Acc:{
212             acc_epo/(X_train.shape[0]*X_train.shape[1]):.4e}')
213         print(f'Epoch: {epoch:5d}, Loss: {np.mean(loss_ep):.4e}, ACC: {
214             np.mean(acc_ep):.5e}')
215
216     loss_epoch[epoch] = np.mean(loss_ep)
217     acc_epoch[epoch] = np.mean(acc_ep)
218
219     Y_pre = model(X_valid)
220     Y_pre1 = Y_pre.clone().detach().numpy()

```

```

219     Y_valid1 = torch.from_numpy(Y_valid)
220     a,b=Y_valid1.shape
221     Y_valid2 = Y_valid1.reshape(a,b,1)
222     Y_valid3 = Y_valid.reshape(a,b,1)
223     loss_valid[epoch] = criterion(Y_pre,Y_valid2)
224     acc_valid[epoch] = acc(Y_pre1,Y_valid3)
225
226 # test
227 X_valid = X_valid.reshape(-1,198,240)
228 #X_valid = torch.from_numpy(X_valid)
229 Y_pred = model(X_valid)
230
231 Y_pred = torch.squeeze(Y_pred,2)
232 Y_pred = Y_pred.clone().detach().numpy()
233 pred_acc = acc(Y_pred,Y_valid)
234
235 Y_pred_re = Y_pred[:,kind]
236 Y_pred_re[Y_pred_re>0] = 1
237 Y_pred_re[Y_pred_re<=0] = 0
238 k = len(Y_pred_re)
239 series = np.arange(1,k,k//100)
240
241 fig = plt.figure()
242 ax = plt.subplot()
243 type1 = ax.scatter(index_valid[series], Y_valid[series,kind], alpha=0.5,
    color='b',label='groundtruth')
244 type2 = ax.scatter(index_valid[series], Y_pred_re[series], alpha=0.5,
    color='r',label='prediction')
245 plt.xlabel("date time")
246 plt.ylabel("0 for fall, 1 for rise")
247 ax.legend((type1, type2), (u'groundtruth', u'prediction'), loc='best')
248 plt.show()
249
250 plt.plot(loss_epoch, 'r-', label='loss')
251 plt.plot(acc_epoch, 'b-', label='accurate rate')
252 plt.legend(loc='best')
253 plt.show()
254
255 plt.plot(acc_valid, 'r-', label='validation acc')
256 plt.plot(acc_epoch, 'b-', label='prediction acc')
257 plt.legend(loc='best')

```



```

258 plt.savefig("dacc.png")
259 plt.show()
260
261 plt.plot(loss_valid, 'r-', label='validation loss')
262 plt.plot(loss_epoch, 'b-', label='prediction loss')
263 plt.legend(loc='best')
264 plt.savefig("dloss.png")
265 plt.show()

```

## B.2 不同的 hidden size/神经元数目对结果的影响的代码

单向 LSTM 网络

```

1 model = LSTMRegression(input_size=1, hidden_size=5, output_size=1)
2
3 criterion = torch.nn.BCEWithLogitsLoss()      #交叉熵BCEWithLogitsLoss()
        和MultiLabelSoftMarginLoss()
4 #criterion = torch.nn.CrossEntropyLoss()
5 optimizer = optim.Adam(model.parameters(), lr=1e-3)
6 #optimizer = optim.SGD(model.parameters(), lr=1e-1)
7
8 epochs = 100
9 batch_size = 30
10 batch = X_train.shape[0] // batch_size
11
12
13
14 torch_dataset = Data.TensorDataset(torch.tensor(X_train), torch.tensor(
    Y_train))
15 # 把 dataset 放入 DataLoader
16 loader = Data.DataLoader(
17     dataset=torch_dataset, # torch TensorDataset format
18     batch_size=batch_size, # mini batch size
19     shuffle=True, #
20     num_workers=10, # 多线程来读数据
21 )
22
23 loss_epoch = np.zeros(epochs)
24 acc_epoch = np.zeros(epochs)
25 for epoch in range(epochs):
26     loss_ep = np.array([])
27     acc_ep = np.array([])

```

```

28     for step,(var_x,var_y) in enumerate(loader):
29         out = model(var_x)
30         out_f = out.detach().clone().numpy()
31         var_yf = var_y.detach().clone().numpy()
32         loss = criterion(out, var_y)
33         loss_f = loss.detach().clone().numpy()
34         acc_ep = np.append(acc_ep,acc(out_f,var_yf))
35         loss_ep = np.append(loss_ep,loss_f)
36
37         optimizer.zero_grad()
38         loss.backward()
39         optimizer.step()
40
41     if (epoch + 1) % 5 == 0:
42         print(f'Epoch: {epoch:5d}, Loss: {np.mean(loss_ep):.4e}, Acc:{np
43             .mean(acc_ep):.4e}')
44
45     loss_epoch[epoch] = np.mean(loss_ep)
46     acc_epoch[epoch] = np.mean(acc_ep)
47
48 # test
49 #X = X.reshape(-1,198,240)
50 #X = torch.from_numpy(X_valid)
51 Y_pred = model(X_valid)
52 Y_pred = Y_pred.clone().detach().numpy()
53 pred_acc = acc(Y_pred,Y_valid)
54 #Y_pred = Y_pred.view(-1).data.numpy()
55
56 # visulize
57 kind = 2
58 series = np.arange(kind*len(index_valid),(kind+1)*len(index_valid))
59 Y_pred_re = Y_pred
60 Y_pred_re[Y_pred_re>0] = 1
61 Y_pred_re[Y_pred_re<=0] = 0
62
63
64 fig = plt.figure()
65 ax = plt.subplot()
66 type1 = ax.scatter(index_valid, Y_valid[series], alpha=0.5,color='b',
67     label='groundtruth')

```

```

67 type2 = ax.scatter(index_valid, Y_pred_re[series], alpha=0.5,color='r',
    label='prediction')
68 plt.xlabel("date time")
69 plt.ylabel("0 for fall, 1 for rise")
70 ax.legend((type1, type2), (u'groundtruth', u'prediction'), loc='best')
71 plt.show()
72
73 plt.plot(loss_epoch, 'r-', label='loss')
74 plt.plot(acc_epoch, 'b-', label='accurate rate')
75 plt.legend(loc='best')
76 plt.show()
77
78 model = LSTMRegression(input_size=1, hidden_size=2, output_size=1)
79
80 criterion = torch.nn.BCEWithLogitsLoss()
81 optimizer = optim.Adam(model.parameters(), lr=1e-3)
82
83 epochs = 100
84 batch_size = 30
85 batch = X_train.shape[0] // batch_size
86
87
88
89 torch_dataset = Data.TensorDataset(torch.tensor(X_train), torch.tensor(
    Y_train))
90 # 把 dataset 放入 DataLoader
91 loader = Data.DataLoader(
92     dataset=torch_dataset, # torch TensorDataset format
93     batch_size=batch_size, # mini batch size
94     shuffle=True, #
95     num_workers=10, # 多线程来读数据
96 )
97
98 loss_epoch = np.zeros(epochs)
99 acc_epoch = np.zeros(epochs)
100 for epoch in range(epochs):
101     loss_ep = np.array([])
102     acc_ep = np.array([])
103     for step,(var_x,var_y) in enumerate(loader):
104         out = model(var_x)
105         out_f = out.detach().clone().numpy()

```

```

106     var_yf = var_y.detach().clone().numpy()
107     loss = criterion(out, var_y)
108     loss_f = loss.detach().clone().numpy()
109     acc_ep = np.append(acc_ep, acc(out_f, var_yf))
110     loss_ep = np.append(loss_ep, loss_f)
111
112     optimizer.zero_grad()
113     loss.backward()
114     optimizer.step()
115
116     if (epoch + 1) % 5 == 0:
117         print(f'Epoch: {epoch:5d}, Loss: {np.mean(loss_ep):.4e}, Acc:{np
118             .mean(acc_ep):.4e}')
119
120     loss_epoch[epoch] = np.mean(loss_ep)
121     acc_epoch[epoch] = np.mean(acc_ep)
122
123 Y_pred = model(X_valid)
124 Y_pred = Y_pred.clone().detach().numpy()
125 pred_acc = acc(Y_pred, Y_valid)
126
127 kind = 2
128 series = np.arange(kind*len(index_valid), (kind+1)*len(index_valid))
129 Y_pred_re = Y_pred
130 Y_pred_re[Y_pred_re>0] = 1
131 Y_pred_re[Y_pred_re<=0] = 0
132
133
134 fig = plt.figure()
135 ax = plt.subplot()
136 type1 = ax.scatter(index_valid, Y_valid[series], alpha=0.5, color='b',
137     label='groundtruth')
137 type2 = ax.scatter(index_valid, Y_pred_re[series], alpha=0.5, color='r',
138     label='prediction')
138 plt.xlabel("date time")
139 plt.ylabel("0 for fall, 1 for rise")
140 ax.legend((type1, type2), (u'groundtruth', u'prediction'), loc='best')
141 plt.show()
142
143 plt.plot(loss_epoch, 'r-', label='loss')

```

```

144 plt.plot(acc_epoch, 'b-', label='accurate rate')
145 plt.legend(loc='best')
146 plt.show()
147
148 model = LSTMRegression(input_size=1, hidden_size=3, output_size=1)
149
150 criterion = torch.nn.BCEWithLogitsLoss()
151 optimizer = optim.Adam(model.parameters(), lr=1e-3)
152
153 epochs = 100
154 batch_size = 30
155 batch = X_train.shape[0] // batch_size
156
157
158
159 torch_dataset = Data.TensorDataset(torch.tensor(X_train), torch.tensor(
    Y_train))
160 # 把 dataset 放入 DataLoader
161 loader = Data.DataLoader(
162     dataset=torch_dataset, # torch TensorDataset format
163     batch_size=batch_size, # mini batch size
164     shuffle=True, #
165     num_workers=10, # 多线程来读数据
166 )
167
168 loss_epoch = np.zeros(epochs)
169 acc_epoch = np.zeros(epochs)
170 for epoch in range(epochs):
171     loss_ep = np.array([])
172     acc_ep = np.array([])
173     for step, (var_x, var_y) in enumerate(loader):
174         out = model(var_x)
175         out_f = out.detach().clone().numpy()
176         var_yf = var_y.detach().clone().numpy()
177         loss = criterion(out, var_y)
178         loss_f = loss.detach().clone().numpy()
179         acc_ep = np.append(acc_ep, acc(out_f, var_yf))
180         loss_ep = np.append(loss_ep, loss_f)
181
182     optimizer.zero_grad()
183     loss.backward()

```

```

184         optimizer.step()
185
186         if (epoch + 1) % 5 == 0:
187             print(f'Epoch: {epoch:5d}, Loss: {np.mean(loss_ep):.4e}, Acc:{np
188                 .mean(acc_ep):.4e}')
189
190         loss_epoch[epoch] = np.mean(loss_ep)
191         acc_epoch[epoch] = np.mean(acc_ep)
192
193     Y_pred = model(X_valid)
194     Y_pred = Y_pred.clone().detach().numpy()
195     pred_acc = acc(Y_pred, Y_valid)
196
197     kind = 2
198     series = np.arange(kind*len(index_valid), (kind+1)*len(index_valid))
199     Y_pred_re = Y_pred
200     Y_pred_re[Y_pred_re>0] = 1
201     Y_pred_re[Y_pred_re<=0] = 0
202
203
204     fig = plt.figure()
205     ax = plt.subplot()
206     type1 = ax.scatter(index_valid, Y_valid[series], alpha=0.5, color='b',
207         label='groundtruth')
208     type2 = ax.scatter(index_valid, Y_pred_re[series], alpha=0.5, color='r',
209         label='prediction')
210     plt.xlabel("date time")
211     plt.ylabel("0 for fall, 1 for rise")
212     ax.legend((type1, type2), (u'groundtruth', u'prediction'), loc='best')
213     plt.show()
214
215     plt.plot(loss_epoch, 'r-', label='loss')
216     plt.plot(acc_epoch, 'b-', label='accurate rate')
217     plt.legend(loc='best')
218     plt.show()

```

## 双向 LSTM 网络

```

1 %%time
2 class LSTMRegression(nn.Module):
3     def __init__(self, input_size, hidden_size, output_size=1,

```

```

        num_layers=2,bidirectional=True):
4         super().__init__()
5         self.lstm = nn.LSTM(input_size, hidden_size, num_layers,
        batch_first=True,bidirectional=True)
6         self.linear = nn.Linear(2*hidden_size, output_size)
7
8     def forward(self, x):
9         x, _ = self.lstm(x) # (seq, batch, hidden)
10        s, b, h = x.shape
11        #         print(s,b)
12        x = x.contiguous().view(s*b, h) # 转换成线性层的输入格式
13        x = self.linear(x)
14        x = x.view(s, b, -1)
15        return x
16
17
18
19 torch.manual_seed(7) #cpu
20 torch.cuda.manual_seed(7) #gpu
21
22 np.random.seed(7) #numpy
23 random.seed(7) # random and transforms
24 torch.backends.cudnn.deterministic=True #cudnn
25
26 model = LSTMRegression(input_size=240, hidden_size=4, output_size=1)
27
28 criterion = torch.nn.BCEWithLogitsLoss()          #交叉熵BCEWithLogitsLoss()
        和MultiLabelSoftMarginLoss()
29 #criterion = torch.nn.CrossEntropyLoss()
30 optimizer = optim.Adam(model.parameters(), lr=1e-3)
31 #optimizer = optim.SGD(model.parameters(), lr=1e-1)
32
33 epochs = 100
34 batch_size = 40
35 batch = X_train.shape[0] // batch_size
36
37 torch_dataset = Data.TensorDataset(torch.tensor(X_train), torch.tensor(
        Y_train))
38 # 把 dataset 放入 DataLoader
39 loader = Data.DataLoader(
40     dataset=torch_dataset, # torch TensorDataset format

```

```

41     batch_size=batch_size, # mini batch size
42     shuffle=True, #
43     num_workers=2, # 多线程来读数据
44 )
45
46 loss_epoch = np.zeros(epochs)
47 acc_epoch = np.zeros(epochs)
48 for epoch in range(epochs):
49     acc_epo = 0
50     loss_ep = np.array([])
51     acc_ep = np.array([])
52     for step,(var_x,var_y) in enumerate(loader):
53         out = model(var_x)
54         out_f = out.detach().clone().numpy()
55         var_yf = var_y.detach().clone().numpy()
56         loss = criterion(out, var_y)
57         loss_f = loss.detach().clone().numpy()
58         acc_ep = np.append(acc_ep,acc(out_f,var_yf))
59         loss_ep = np.append(loss_ep,loss_f)
60
61         optimizer.zero_grad()
62         loss.backward()
63         optimizer.step()
64
65     if (epoch + 1) % 10 == 0:
66         #print(f'Epoch: {epoch:5d}, Loss: {loss.item():.4e}, Acc:{
67             acc_epo/(X_train.shape[0]*X_train.shape[1]):.4e}')
68         print(f'Epoch: {epoch:5d}, Loss: {np.mean(loss_ep):.4e}, ACC: {
69             np.mean(acc_ep):.5e}')
70
71     loss_epoch[epoch] = np.mean(loss_ep)
72     acc_epoch[epoch] = np.mean(acc_ep)
73
74 # test
75 X_valid = X_valid.reshape(-1,198,240)
76 #X_valid = torch.from_numpy(X_valid)
77 Y_pred = model(X_valid)
78
79 kind = 3
80 Y_pred = torch.squeeze(Y_pred,2)
81 Y_pred = Y_pred.clone().detach().numpy()

```



```

80 pred_acc = acc(Y_pred,Y_valid)
81
82 Y_pred_re = Y_pred[:,kind]
83 Y_pred_re[Y_pred_re>0] = 1
84 Y_pred_re[Y_pred_re<=0] = 0
85 k = len(Y_pred_re)
86 series = np.arange(1,k,k//100)
87
88 fig = plt.figure()
89 ax = plt.subplot()
90 type1 = ax.scatter(index_valid[series], Y_valid[series,kind], alpha=0.5,
    color='b',label='groundtruth')
91 type2 = ax.scatter(index_valid[series], Y_pred_re[series], alpha=0.3,
    color='r',label='prediction')
92 plt.xlabel("date time")
93 plt.ylabel("0 for fall, 1 for rise")
94 ax.legend((type1, type2), (u'groundtruth', u'prediction'), loc='best')
95 plt.show()
96
97 plt.plot(loss_epoch, 'r-', label='loss')
98 plt.plot(acc_epoch, 'b-', label='accurate rate')
99 plt.legend(loc='best')
100 plt.show()

```

### B.3 不同的 batch size 对结果的影响的代码

```

1 #!/usr/bin/env python
2 import numpy as np
3 import pandas as pd
4 import torch
5 import matplotlib.pyplot as plt
6 from torch.autograd import Variable
7 import torch.utils.data as Data
8 import random
9 # split a univariate sequence into samples
10 def split_sequence(sequence, n_steps):
11     X, y = [], []
12     for i in range(len(sequence)):
13         # find the end of this pattern
14         end_idx = i + n_steps
15         # check if we are beyond the sequence

```

```

16         if end_idx > len(sequence) - 1:
17             break
18         # gather input and output parts if the pattern
19         seq_x, seq_y = sequence[i:end_idx], sequence[end_idx]
20         X.append(seq_x)
21         y.append(seq_y)
22     return np.array(X), np.array(y)
23 #!/usr/bin/env python
24 import torch
25 import torch.nn as nn
26 from torch.nn import functional as F
27 from torch import optim
28
29 import numpy as np
30 import pandas as pd
31 import matplotlib.pyplot as plt
32 data0 = pd.read_excel("2011-2020price.xlsx")
33 col_name = list(data0.columns)
34 data = data0[col_name[1:]]
35 index = data0[col_name[0]]
36
37 # plt.plot(data)
38 # plt.show()
39
40
41 dataset = data.dropna().values.astype('float32')
42 dataset = dataset.reshape(-1,1)
43
44 print(dataset.shape)
45
46
47 def centralize(data):
48     min_value = np.min(data, axis=0)
49     max_value = np.max(data, axis=0)
50     data = (data - min_value) / (max_value - min_value)
51     return data
52
53
54 def judge(dataset, k):
55     '''
56     to see at day k, if each stock rise or fall

```

```

57     '''
58     pr_today = dataset[k]
59     pr_yesterday = dataset[k - 1]
60     pr_change = pr_today - pr_yesterday
61     med = np.median(pr_change)
62     re = np.zeros_like(pr_change)
63     re[pr_change > 0] = 1
64     return (re)
65
66
67 def acc(out, y_real):
68     out1 = np.zeros_like(out)
69     out1[out > 0] = 1
70     out1[out <= 0] = 0
71     return 1 - sum(sum(abs(y_real - out1))) / (np.prod(y_real.shape))
72
73
74 def create_dataset(dataset, look_back=240):
75     dataX, dataY = [], []
76     for i in range(len(dataset) - look_back):
77         pr_change = judge(dataset, i + look_back)
78         a = dataset[i:(i + look_back)]
79         dataX.append(a)
80         dataY.append(pr_change)
81     return np.array(dataX), np.array(dataY)
82
83
84 def set_seed(seed):
85     torch.manual_seed(seed) # cpu 为CPU设置种子用于生成随机数, 以使得结
      果是确定的
86     torch.cuda.manual_seed(seed) # gpu 为当前GPU设置随机种子
87     torch.backends.cudnn.deterministic = True # cudnn
88     np.random.seed(seed) # numpy
89     random.seed(seed)
90
91
92 look_back = 240
93 index_used = index[look_back:]
94 index_used = np.array(index_used)
95 X, Y = create_dataset(dataset, look_back)
96 print(X.shape, Y.shape)

```

```

97 a, b, c = X.shape
98
99 train_size = int(len(X) * 0.9)
100 valid_size = len(X) - train_size
101 index_size = int(len(index_used) * 0.9)
102 print(train_size, valid_size)
103
104 X_train = X[:train_size]
105 Y_train = Y[:train_size]
106 index_train = index_used[:index_size]
107
108 X_valid = X[train_size:]
109 Y_valid = Y[train_size:]
110 index_valid = index_used[index_size:]
111
112 # X_train = X_train.reshape(-1,198,240)
113 # X_valid = X_valid.reshape(-1,198,240)
114 # Y_train = Y_train.reshape(-1,198,1)
115
116 X_train = X_train.reshape(train_size * c, b, 1)
117 Y_train = Y_train.reshape(train_size * c, 1)
118 X_valid = X_valid.reshape(valid_size * c, b, 1)
119 Y_valid = Y_valid.reshape(valid_size * c, 1)
120
121 # X_train = X_train.transpose(1, 0, 2)
122 # X_valid = X_valid.transpose(1, 0, 2)
123
124 X_train = torch.from_numpy(X_train)
125 Y_train = torch.from_numpy(Y_train)
126 X_valid = torch.from_numpy(X_valid)
127
128 print(X_train.shape, Y_train.shape)
129
130
131 index_train = index[:index_size]
132 index_valid = index[int(len(index)*0.9):]
133
134 index_valid.shape
135 X.shape
136
137

```

```

138
139
140 class LSTMRegression(nn.Module):
141     def __init__(self, input_size, hidden_size, output_size=1,
142                 num_layers=1):
143         super().__init__()
144         self.lstm = nn.LSTM(input_size, hidden_size, num_layers,
145                             batch_first=True)
146         self.linear = nn.Linear(hidden_size, output_size)
147
148     def forward(self, x):
149         _, (hn, cn) = self.lstm(x)
150         hn = hn.squeeze()
151         out = self.linear(hn)
152         return out
153
154
155 model = LSTMRegression(input_size=1, hidden_size=5, output_size=1)
156
157 criterion = torch.nn.BCEWithLogitsLoss() # 交叉熵BCEWithLogitsLoss()和
158         MultiLabelSoftMarginLoss()
159 # criterion = torch.nn.CrossEntropyLoss()
160 optimizer = optim.Adam(model.parameters(), lr=1e-3)
161 # optimizer = optim.SGD(model.parameters(), lr=1e-1)
162
163 epochs = 20
164 batch_size = 60
165 batch = X_train.shape[0] // batch_size
166
167 torch_dataset = Data.TensorDataset(torch.tensor(X_train), torch.tensor(
168     Y_train))
169 # 把 dataset 放入 DataLoader
170 loader = Data.DataLoader(
171     dataset=torch_dataset, # torch TensorDataset format
172     batch_size=batch_size, # mini batch size
173     shuffle=True, #
174     num_workers=10, # 多线程来读数据
175 )
176
177 loss_epoch = np.zeros(epochs)
178 acc_epoch = np.zeros(epochs)

```

```

175 for epoch in range(epochs):
176     loss_ep = np.array([])
177     acc_ep = np.array([])
178     for step, (var_x, var_y) in enumerate(loader):
179         out = model(var_x)
180         out_f = out.detach().clone().numpy()
181         var_yf = var_y.detach().clone().numpy()
182         loss = criterion(out, var_y)
183         loss_f = loss.detach().clone().numpy()
184         acc_ep = np.append(acc_ep, acc(out_f, var_yf))
185         loss_ep = np.append(loss_ep, loss_f)
186
187         optimizer.zero_grad()
188         loss.backward()
189         optimizer.step()
190
191     if (epoch + 1) % 2 == 0:
192         print(f'Epoch: {epoch:5d}, Loss: {np.mean(loss_ep):.4e}, Acc:{np
            .mean(acc_ep):.4e}')
193
194     loss_epoch[epoch] = np.mean(loss_ep)
195     acc_epoch[epoch] = np.mean(acc_ep)
196
197 # test
198 #X = X.reshape(-1,198,240)
199 #X = torch.from_numpy(X_valid)
200 Y_pred = model(X_valid)
201 Y_pred = Y_pred.clone().detach().numpy()
202 pred_acc = acc(Y_pred,Y_valid)
203 print(f'pred_acc{pred_acc}')
204 #Y_pred = Y_pred.view(-1).data.numpy()
205 # visulize
206 kind = 2
207 series = np.arange(kind*len(index_valid),(kind+1)*len(index_valid))
208 Y_pred_re = Y_pred
209 Y_pred_re[Y_pred_re>0] = 1
210 Y_pred_re[Y_pred_re<=0] = 0
211
212
213 fig = plt.figure()
214 ax = plt.subplot()

```

```

215 type1 = ax.scatter(index_valid, Y_valid[series], alpha=0.5,color='b',
    label='groundtruth')
216 type2 = ax.scatter(index_valid, Y_pred_re[series], alpha=0.5,color='r',
    label='prediction')
217 plt.xlabel("date time")
218 plt.ylabel("0 for fall, 1 for rise")
219 ax.legend((type1, type2), (u'groundtruth', u'prediction'), loc='best')
220 plt.show()
221
222 plt.plot(loss_epoch, 'r-', label='loss')
223 plt.plot(acc_epoch, 'b-', label='accurate rate')
224 plt.legend(loc='best')
225 plt.show()

```

## B.4 不同的学习率对结果的影响的代码

```

1  #!/usr/bin/env python
2  import numpy as np
3  import pandas as pd
4  import torch
5  import matplotlib.pyplot as plt
6  from torch.autograd import Variable
7  import torch.utils.data as Data
8  import random
9
10 # split a univariate sequence into samples
11 def split_sequence(sequence, n_steps):
12     X, y = [], []
13     for i in range(len(sequence)):
14         # find the end of this pattern
15         end_idx = i + n_steps
16         # check if we are beyond the sequence
17         if end_idx > len(sequence) - 1:
18             break
19         # gather input and output parts if the pattern
20         seq_x, seq_y = sequence[i:end_idx], sequence[end_idx]
21         X.append(seq_x)
22         y.append(seq_y)
23     return np.array(X), np.array(y)
24
25 #!/usr/bin/env python

```

```

26 import torch
27 import torch.nn as nn
28 from torch.nn import functional as F
29 from torch import optim
30
31 import numpy as np
32 import pandas as pd
33 import matplotlib.pyplot as plt
34
35 data0 = pd.read_excel("D:/mathematical experiment/code/code/2011-2020
    price.xlsx")
36 col_name = list(data0.columns)
37 data = data0[col_name[1:]]
38 index = data0[col_name[0]]
39
40 # plt.plot(data)
41 # plt.show()
42
43
44 dataset = data.dropna().values.astype('float32')
45 dataset = dataset.reshape(-1,1)
46
47 print(dataset.shape)
48
49 def centralize(data):
50     min_value = np.min(data,axis=0)
51     max_value = np.max(data,axis=0)
52     data = (data - min_value) / (max_value-min_value)
53     return data
54
55 def judge(dataset,k):
56     '''
57     to see at day k, if each stock rise or fall
58     '''
59     pr_today = dataset[k]
60     pr_yesterday = dataset[k-1]
61     pr_change = pr_today - pr_yesterday
62     med = np.median(pr_change)
63     re = np.zeros_like(pr_change)
64     re[pr_change>0] = 1
65     return(re)

```



```

66
67 def acc(out,y_real):
68     out1 = np.zeros_like(out)
69     out1[out>0] = 1
70     out1[out<=0] = 0
71     return 1-sum(sum(abs(y_real-out1)))/(np.prod(y_real.shape))
72
73
74 def create_dataset(dataset,look_back=240):
75     dataX,dataY=[],[]
76     for i in range(len(dataset)-look_back):
77         pr_change = judge(dataset,i+look_back)
78         a = dataset[i:(i+look_back)]
79         dataX.append(a)
80         dataY.append(pr_change)
81     return np.array(dataX),np.array(dataY)
82
83 def set_seed(seed):
84     torch.manual_seed(seed) # cpu 为CPU设置种子用于生成随机数, 以使得结
      果是确定的
85     torch.cuda.manual_seed(seed) # gpu 为当前GPU设置随机种子
86     torch.backends.cudnn.deterministic = True # cudnn
87     np.random.seed(seed) # numpy
88     random.seed(seed)
89
90 look_back = 240
91 index_used = index[look_back:]
92 index_used = np.array(index_used)
93 X, Y = create_dataset(dataset,look_back)
94 print(X.shape, Y.shape)
95 a,b,c = X.shape
96
97 train_size = int(len(X) * 0.9)
98 valid_size = len(X) - train_size
99 index_size = int(len(index_used)*0.9)
100 print(train_size, valid_size)
101
102 X_train = X[:train_size]
103 Y_train = Y[:train_size]
104 index_train = index_used[:index_size]
105

```

```

106
107 X_valid = X[train_size:]
108 Y_valid = Y[train_size:]
109 index_valid = index_used[index_size:]
110
111
112 # X_train = X_train.reshape(-1,198,240)
113 # X_valid = X_valid.reshape(-1,198,240)
114 # Y_train = Y_train.reshape(-1,198,1)
115
116 X_train = X_train.reshape(train_size*c,b,1)
117 Y_train = Y_train.reshape(train_size*c,1)
118 X_valid = X_valid.reshape(valid_size*c,b,1)
119 Y_valid = Y_valid.reshape(valid_size*c,1)
120
121 # X_train = X_train.transpose(1, 0, 2)
122 # X_valid = X_valid.transpose(1, 0, 2)
123
124 X_train = torch.from_numpy(X_train)
125 Y_train = torch.from_numpy(Y_train)
126 X_valid = torch.from_numpy(X_valid)
127
128 print(X_train.shape,Y_train.shape)
129
130
131 %%time
132 class LSTMRegression(nn.Module):
133     def __init__(self, input_size, hidden_size, output_size=1,
134                 num_layers=1):
135         super().__init__()
136         self.lstm = nn.LSTM(input_size, hidden_size, num_layers,
137                             batch_first=True)
138         self.linear = nn.Linear(hidden_size, output_size)
139
140     def forward(self, x):
141         _, (hn, cn) = self.lstm(x)
142         hn = hn.squeeze()
143         out = self.linear(hn)
144         return out
145
146 lr = [0.01,0.001,0.0001]

```

```

145
146 for i in range(3):
147
148     model = LSTMRegression(input_size=1, hidden_size=5, output_size=1)
149
150     criterion = torch.nn.BCEWithLogitsLoss()      #交叉熵
151     BCEWithLogitsLoss()和MultiLabelSoftMarginLoss()
152     #criterion = torch.nn.CrossEntropyLoss()
153     optimizer = optim.Adam(model.parameters(), lr[i])
154     #optimizer = optim.SGD(model.parameters(), lr=1e-1)
155
156     epochs = 30
157     batch_size = 30
158     batch = X_train.shape[0] // batch_size
159
160
161     torch_dataset = Data.TensorDataset(torch.tensor(X_train), torch.
162     tensor(Y_train))
163     # 把 dataset 放入 DataLoader
164     loader = Data.DataLoader(
165         dataset=torch_dataset, # torch TensorDataset format
166         batch_size=batch_size, # mini batch size
167         shuffle=True, #
168         num_workers=0,
169     )
170
171     loss_epoch = np.zeros(epochs)
172     acc_epoch = np.zeros(epochs)
173     for epoch in range(epochs):
174         loss_ep = np.array([])
175         acc_ep = np.array([])
176         for step,(var_x,var_y) in enumerate(loader):
177             out = model(var_x)
178             out_f = out.detach().clone().numpy()
179             var_yf = var_y.detach().clone().numpy()
180             loss = criterion(out, var_y)
181             loss_f = loss.detach().clone().numpy()
182             acc_ep = np.append(acc_ep,acc(out_f,var_yf))
183             loss_ep = np.append(loss_ep,loss_f)

```

```

184         optimizer.zero_grad()
185         loss.backward()
186         optimizer.step()
187
188     if (epoch + 1) % 5 == 0:
189         print(f'Epoch: {epoch:5d}, Loss: {np.mean(loss_ep):.4e}, Acc
          :{np.mean(acc_ep):.4e}')
190
191
192     loss_epoch[epoch] = np.mean(loss_ep)
193     acc_epoch[epoch] = np.mean(acc_ep)
194
195
196
197     # test
198     #X = X.reshape(-1,198,240)
199     #X = torch.from_numpy(X_valid)
200     Y_pred = model(X_valid)
201     Y_pred = Y_pred.clone().detach().numpy()
202     pred_acc = acc(Y_pred,Y_valid)
203     print(pred_acc)
204     #Y_pred = Y_pred.view(-1).data.numpy()
205
206     # visulize
207     kind = 2
208     series = np.arange(kind*len(index_valid),(kind+1)*len(index_valid))
209     Y_pred_re = Y_pred
210     Y_pred_re[Y_pred_re>0] = 1
211     Y_pred_re[Y_pred_re<=0] = 0
212
213     filename1 = 'fall and rise' + str(i) + '.png'
214     filename2 = 'loss and accuracy' + str(i) + '.png'
215
216     fig = plt.figure()
217     ax = plt.subplot()
218     type1 = ax.scatter(index_valid, Y_valid[series], alpha=0.5,color='b'
          ,label='groundtruth')
219     type2 = ax.scatter(index_valid, Y_pred_re[series], alpha=0.3,color='
          r',label='prediction')
220     plt.xlabel("date time")
221     plt.ylabel("0 for fall, 1 for rise")

```

```

222     ax.legend((type1, type2), (u'groundtruth', u'prediction'), loc='best
      ')
223     plt.show()
224     plt.savefig(filename1)
225
226     plt.plot(loss_epoch, 'r-', label='loss')
227     plt.plot(acc_epoch, 'b-', label='accurate rate')
228     plt.legend(loc='best')
229     plt.show()
230     plt.savefig(filename2)

```

## B.5 不同的优化器对结果的影响的代码

```

1  #!/usr/bin/env python
2  import numpy as np
3  import pandas as pd
4  import torch
5  import matplotlib.pyplot as plt
6  from torch.autograd import Variable
7  import torch.utils.data as Data
8  import random
9
10 # split a univariate sequence into samples
11 def split_sequence(sequence, n_steps):
12     X, y = [], []
13     for i in range(len(sequence)):
14         # find the end of this pattern
15         end_idx = i + n_steps
16         # check if we are beyond the sequence
17         if end_idx > len(sequence) - 1:
18             break
19         # gather input and output parts if the pattern
20         seq_x, seq_y = sequence[i:end_idx], sequence[end_idx]
21         X.append(seq_x)
22         y.append(seq_y)
23     return np.array(X), np.array(y)
24
25 #!/usr/bin/env python
26 import torch
27 import torch.nn as nn
28 from torch.nn import functional as F

```

```

29 from torch import optim
30
31 import numpy as np
32 import pandas as pd
33 import matplotlib.pyplot as plt
34
35
36 data0 = pd.read_excel("D:/mathematical experiment/code/code/2011-2020
    price.xlsx")
37 col_name = list(data0.columns)
38 data = data0[col_name[1:10]]
39 index = data0[col_name[0]]
40
41 # plt.plot(data)
42 # plt.show()
43
44
45 dataset = data.dropna().values.astype('float32')
46 dataset = dataset.reshape(-1,1)
47
48 print(dataset.shape)
49
50 def centralize(data):
51     min_value = np.min(data,axis=0)
52     max_value = np.max(data,axis=0)
53     data = (data - min_value) / (max_value-min_value)
54     return data
55
56 def judge(dataset,k):
57     '''
58     to see at day k, if each stock rise or fall
59     '''
60     pr_today = dataset[k]
61     pr_yesterday = dataset[k-1]
62     pr_change = pr_today - pr_yesterday
63     med = np.median(pr_change)
64     re = np.zeros_like(pr_change)
65     re[pr_change>0] = 1
66     return(re)
67
68 def acc(out,y_real):

```

```

69     out1 = np.zeros_like(out)
70     out1[out>0] = 1
71     out1[out<=0] = 0
72     return 1-sum(sum(abs(y_real-out1)))/(np.prod(y_real.shape))
73
74
75 def create_dataset(dataset,look_back=240):
76     dataX,dataY=[],[]
77     for i in range(len(dataset)-look_back):
78         pr_change = judge(dataset,i+look_back)
79         a = dataset[i:(i+look_back)]
80         dataX.append(a)
81         dataY.append(pr_change)
82     return np.array(dataX),np.array(dataY)
83
84 def set_seed(seed):
85     torch.manual_seed(seed) # cpu 为CPU设置种子用于生成随机数，以使得结
      果是确定的
86     torch.cuda.manual_seed(seed) # gpu 为当前GPU设置随机种子
87     torch.backends.cudnn.deterministic = True # cudnn
88     np.random.seed(seed) # numpy
89     random.seed(seed)
90
91 look_back = 240
92 index_used = index[look_back:]
93 index_used = np.array(index_used)
94 X, Y = create_dataset(dataset,look_back)
95 print(X.shape, Y.shape)
96 a,b,c = X.shape
97
98 train_size = int(len(X) * 0.9)
99 valid_size = len(X) - train_size
100 index_size = int(len(index_used)*0.9)
101 print(train_size, valid_size)
102
103 X_train = X[:train_size]
104 Y_train = Y[:train_size]
105 index_train = index_used[:index_size]
106
107
108 X_valid = X[train_size:]

```

```

109 Y_valid = Y[train_size:]
110 index_valid = index_used[index_size:]
111
112
113 # X_train = X_train.reshape(-1,198,240)
114 # X_valid = X_valid.reshape(-1,198,240)
115 # Y_train = Y_train.reshape(-1,198,1)
116
117 X_train = X_train.reshape(train_size*c,b,1)
118 Y_train = Y_train.reshape(train_size*c,1)
119 X_valid = X_valid.reshape(valid_size*c,b,1)
120 Y_valid = Y_valid.reshape(valid_size*c,1)
121
122 # X_train = X_train.transpose(1, 0, 2)
123 # X_valid = X_valid.transpose(1, 0, 2)
124
125 X_train = torch.from_numpy(X_train)
126 Y_train = torch.from_numpy(Y_train)
127 X_valid = torch.from_numpy(X_valid)
128
129 print(X_train.shape,Y_train.shape)
130
131
132
133
134
135 %%time
136 class LSTMRegression(nn.Module):
137     def __init__(self, input_size, hidden_size, output_size=1,
138                 num_layers=1):
139         super().__init__()
140         self.lstm = nn.LSTM(input_size, hidden_size, num_layers,
141                             batch_first=True)
142         self.linear = nn.Linear(hidden_size, output_size)
143
144     def forward(self, x):
145         _, (hn, cn) = self.lstm(x)
146         hn = hn.squeeze()
147         out = self.linear(hn)
148         return out

```



```

148 model = LSTMRegression(input_size=1, hidden_size=5, output_size=1)
149
150 for index in range(2):
151     print(index)
152
153     criterion = torch.nn.BCEWithLogitsLoss()      #交叉熵
154     BCEWithLogitsLoss()和MultiLabelSoftMarginLoss()
155     if index==1:
156         optimizer = optim.Adam(model.parameters(), lr=1e-3)
157     else:
158         optimizer = optim.SGD(model.parameters(), lr=1e-3)
159
160 #     else:
161 #         print(index+2)
162 #         criterion = torch.nn.CrossEntropyLoss()
163 #         if index==2:
164 #             optimizer = optim.Adam(model.parameters(), lr=1e-3)
165 #         else:
166 #             optimizer = optim.SGD(model.parameters(), lr=1e-3)
167
168 #optimizer = optim.Adam(model.parameters(), lr=1e-3)
169 #optimizer = optim.SGD(model.parameters(), lr=1e-1)
170
171 epochs = 40
172 batch_size = 30
173 batch = X_train.shape[0] // batch_size
174
175
176
177 torch_dataset = Data.TensorDataset(torch.tensor(X_train), torch.
178 tensor(Y_train))
179 # 把 dataset 放入 DataLoader
180 loader = Data.DataLoader(
181     dataset=torch_dataset, # torch TensorDataset format
182     batch_size=batch_size, # mini batch size
183     shuffle=True, #
184     num_workers=10, # 多线程来读数据
185 )
186
187 loss_epoch = np.zeros(epochs)

```

```

187     acc_epoch = np.zeros(epochs)
188     for epoch in range(epochs):
189         loss_ep = np.array([])
190         acc_ep = np.array([])
191         for step,(var_x,var_y) in enumerate(loader):
192             out = model(var_x)
193             out_f = out.detach().clone().numpy()
194             var_yf = var_y.detach().clone().numpy()
195             loss = criterion(out, var_y)
196             loss_f = loss.detach().clone().numpy()
197             acc_ep = np.append(acc_ep,acc(out_f,var_yf))
198             loss_ep = np.append(loss_ep,loss_f)
199
200             optimizer.zero_grad()
201             loss.backward()
202             optimizer.step()
203
204         if (epoch + 1) % 10 == 0:
205             print(f'Epoch: {epoch:5d}, Loss: {np.mean(loss_ep):.4e}, Acc
                :{np.mean(acc_ep):.4e}')
206
207
208         loss_epoch[epoch] = np.mean(loss_ep)
209         acc_epoch[epoch] = np.mean(acc_ep)
210
211
212
213     # test
214     #X = X.reshape(-1,198,240)
215     #X = torch.from_numpy(X_valid)
216     Y_pred = model(X_valid)
217     Y_pred = Y_pred.clone().detach().numpy()
218     pred_acc = acc(Y_pred,Y_valid)
219     print(f'acc_rate:{pred_acc}')
220     #Y_pred = Y_pred.view(-1).data.numpy()
221
222     # visulize
223     kind = 2
224     series = np.arange(kind*len(index_valid),(kind+1)*len(index_valid))
225     Y_pred_re = Y_pred
226     Y_pred_re[Y_pred_re>0] = 1

```

```

227     Y_pred_re[Y_pred_re<=0] = 0
228
229
230     filename1 = 'point' + str(index) + '.png'
231     filename2 = 'pic' + str(index) + '.png'
232
233     fig = plt.figure()
234     ax = plt.subplot()
235     type1 = ax.scatter(index_valid, Y_valid[series], alpha=0.6,color='b'
236                        ,label='groundtruth')
237     type2 = ax.scatter(index_valid, Y_pred_re[series], alpha=0.3,color='
238                        r',label='prediction')
239     plt.xlabel("date time")
240     plt.ylabel("0 for fall, 1 for rise")
241     ax.legend((type1, type2), (u'groundtruth', u'prediction'), loc='best
242                ')
243     plt.savefig(filename1)
244     plt.show()
245
246     plt.plot(loss_epoch, 'r-', label='loss')
247     plt.plot(acc_epoch, 'b-', label='accurate rate')
248     plt.legend(loc='best')
249     plt.savefig(filename2)
250     plt.show()

```