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

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

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

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

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

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，$sech(x)$ 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

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

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

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

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

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

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

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

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

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。

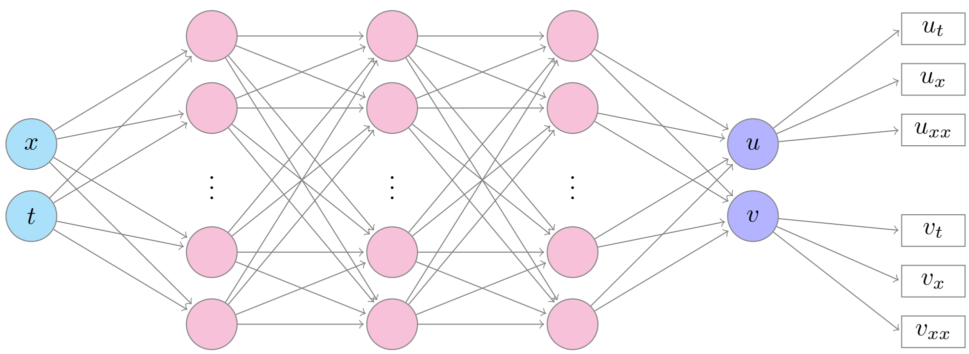


图 1: 求解薛定谔方程的神经网络架构

损失函数　损失函数可表示为

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

其中

$$\begin{aligned}\mathcal{L}_{\mathrm{PDE}}(\boldsymbol{\Theta}) = \frac{1}{N_{\mathrm{PDE}}} \sum_{n=1}^{N_{\mathrm{PDE}}} \Big\{ & \left[\hat{u}_t(t_n,x_n) + 0.5\hat{v}_{xx}(t_n,x_n) + \left(\hat{u}(t_n,x_n)^2 + \hat{v}(t_n,x_n)^2\right)\hat{v}(t_n,x_n)\right]^2 \\ & + \left[\hat{v}_t(t_n,x_n) - 0.5\hat{u}_{xx}(t_n,x_n) - \left(\hat{u}(t_n,x_n)^2 + \hat{v}(t_n,x_n)^2\right)\hat{u}(t_n,x_n)\right]^2 \Big\},\end{aligned}$$

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

$$\begin{aligned}\mathcal{L}_{\mathrm{BC}}(\boldsymbol{\Theta}) = \frac{1}{N_{\mathrm{BC}}} \sum_{n=1}^{N_{\mathrm{BC}}} \Big\{ & \left[\hat{u}(t_n,-5) - \hat{u}(t_n,5)\right]^2 + \left[\hat{v}(t_n,-5) - \hat{v}(t_n,5)\right]^2 + \\ & \left[\hat{u}_x(t_n,-5) - \hat{u}_x(t_n,5)\right]^2 + \left[\hat{v}_x(t_n,-5) - \hat{v}_x(t_n,5)\right]^2 \Big\}\end{aligned}$$

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

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

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 preblem 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 $h(0, x) = 2sech(x)$ 来创建方程内函数 (其中 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 $5 * 2$ 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：

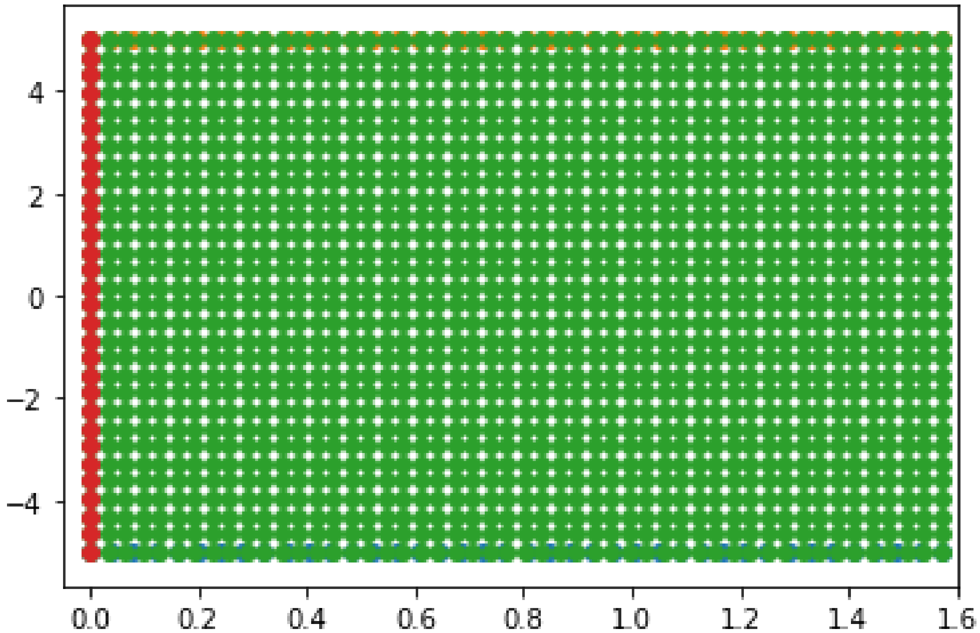


图 2: 均匀采样

- 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：

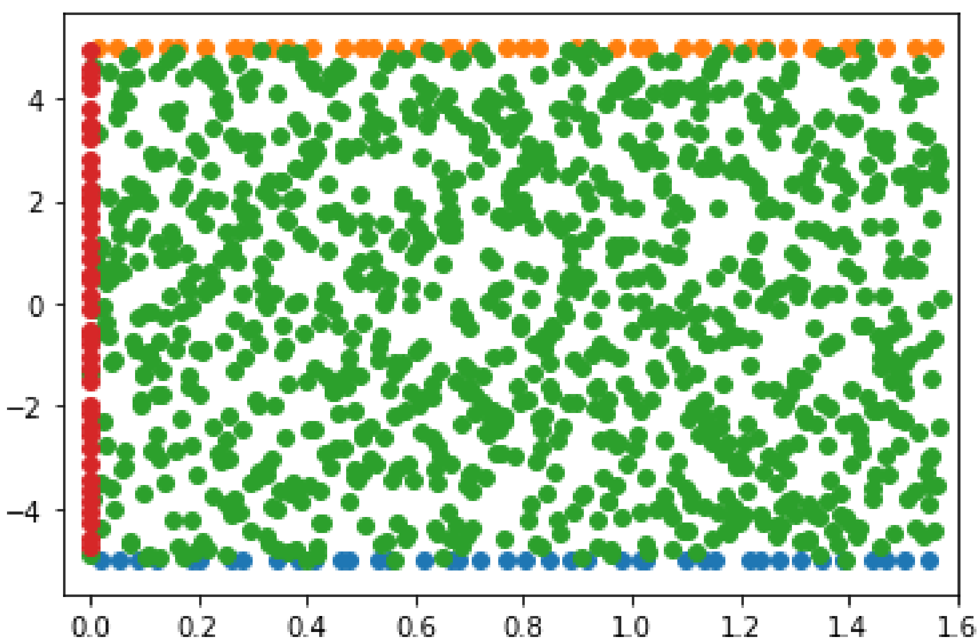


图 3: lhs 采样

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

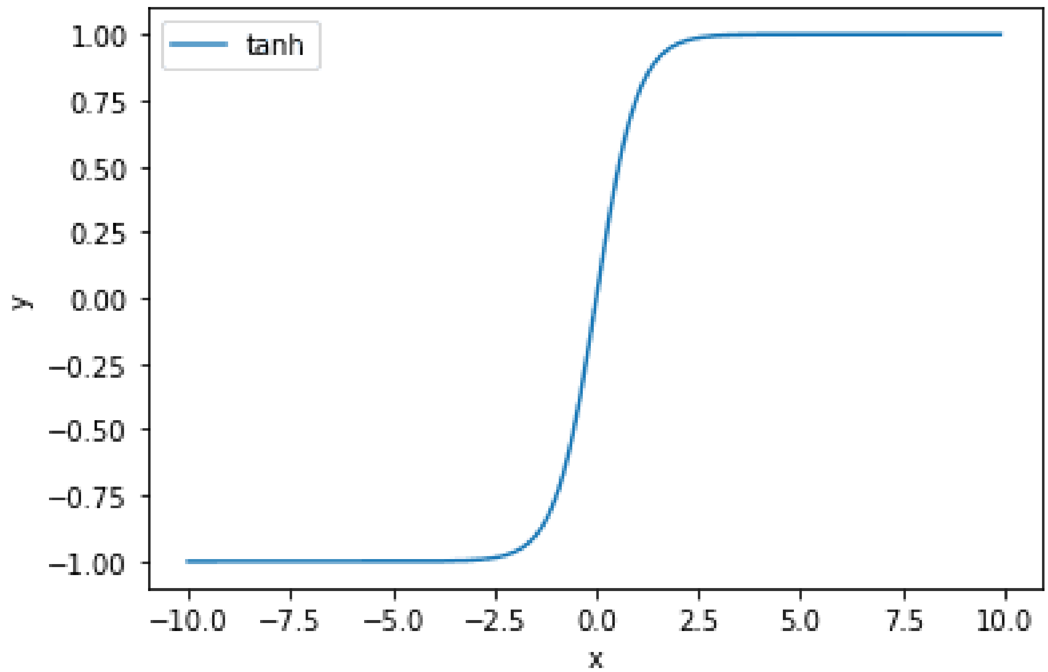


图 4: tanh 函数

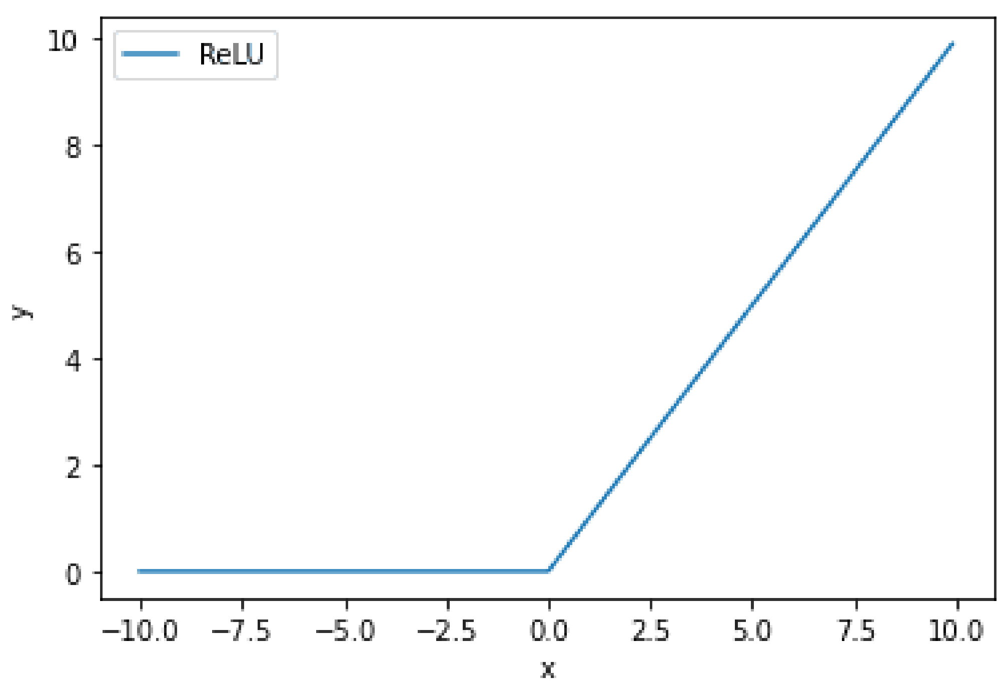


图 5: ReLU 函数

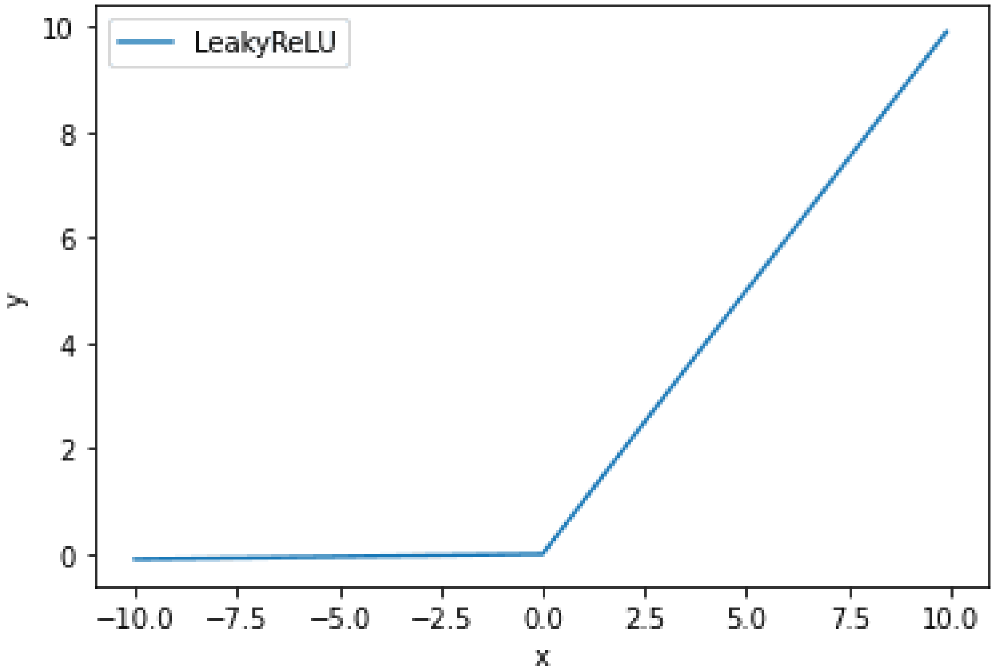


图 6: LeakyReLU 函数

- softplus

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

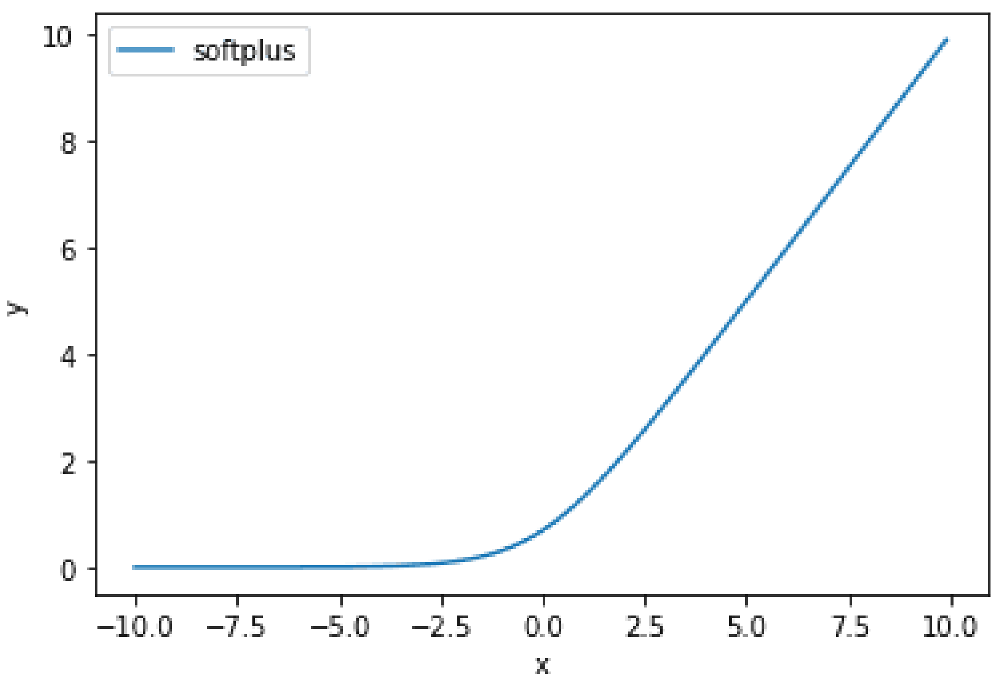


图 7: softplus 函数

- sigmoid

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

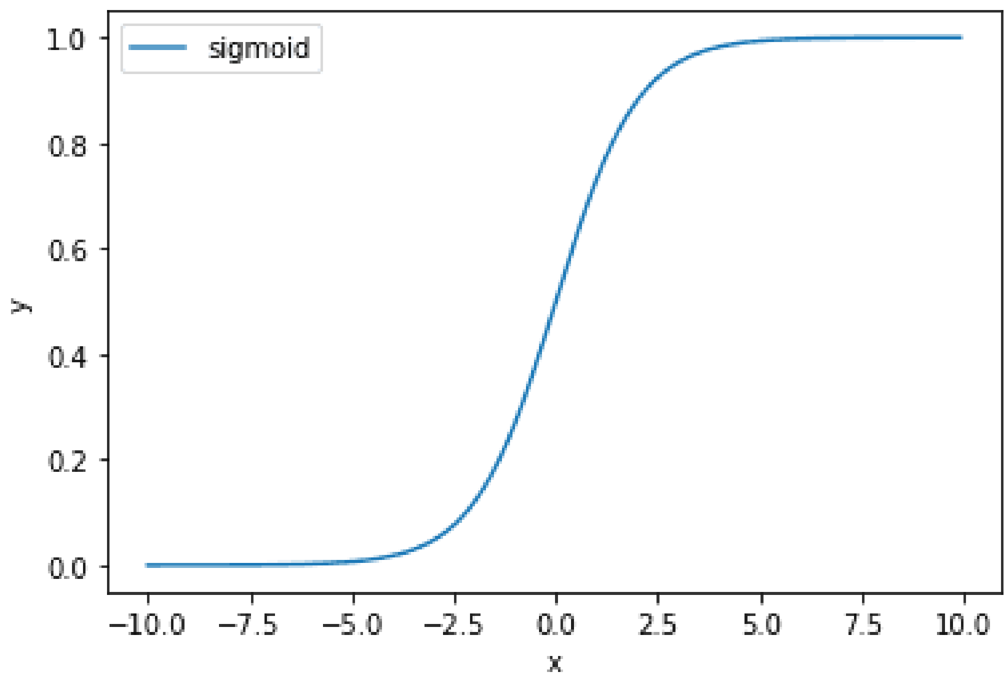


图 8: sigmoid 函数

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，所以在画图时需要对 t 或 x 进行限定。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：

$$h = u + iv$$
$$-v_t + 0.5u_x x + (u^2 + v^2)u + iu_t + i(u^2 + v^2) = 0$$

实数部分与虚数部分分别为 0：

$$-v_t + 0.5u_x x + (u^2 + v^2)u = 0$$
$$u_t + u^2 + v^2 = 0$$

在这里将初值条件设置为

$$\mid h(0, x) \mid = 2sech(x)$$

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：

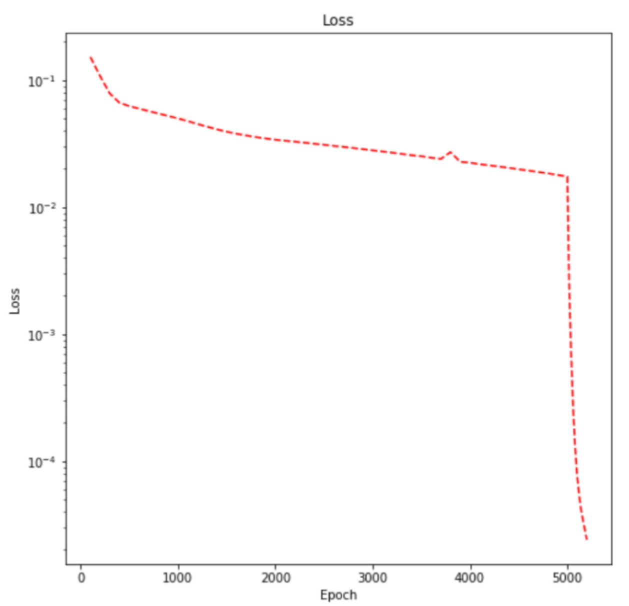


图 9: Loss 随 Epoch 增长之间的关系

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:

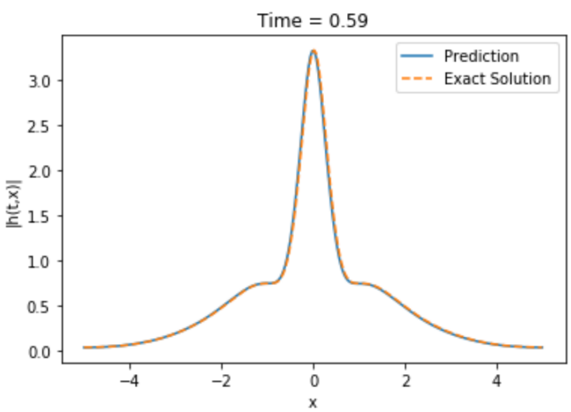


图 10: t=0.59

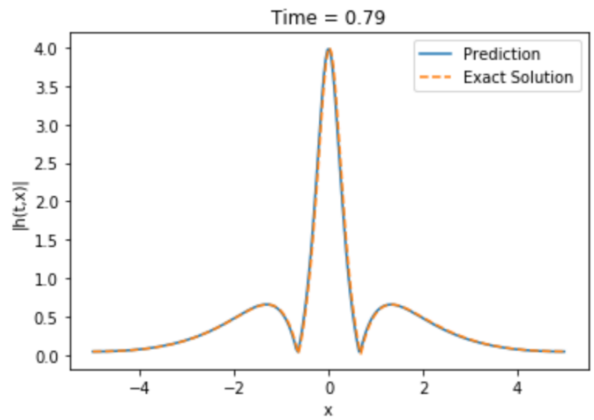


图 11: t=0.79

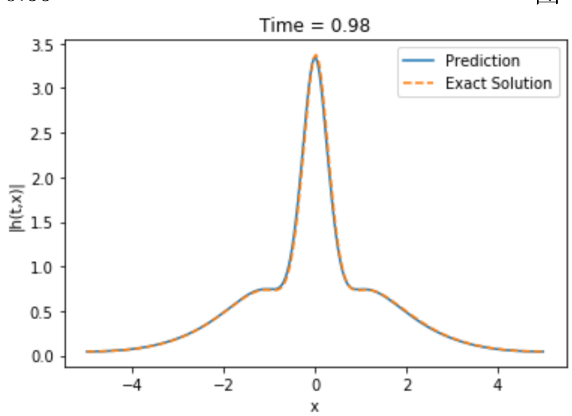


图 12: t=0.98

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:

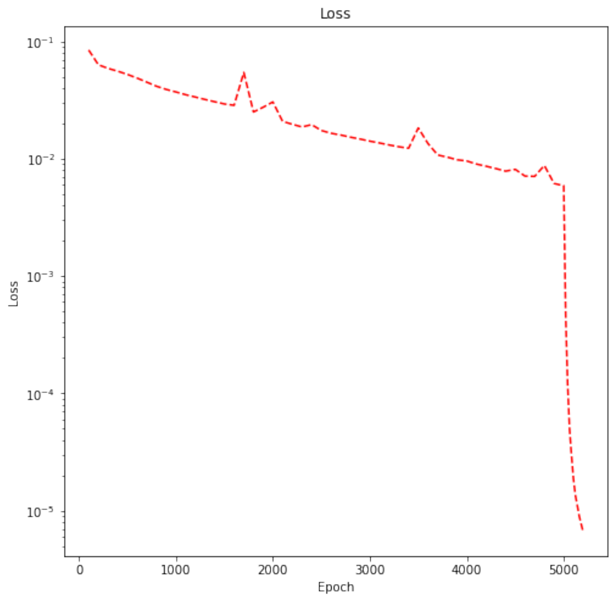


图 13: Loss 随 Epoch 增长之间的关系

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:

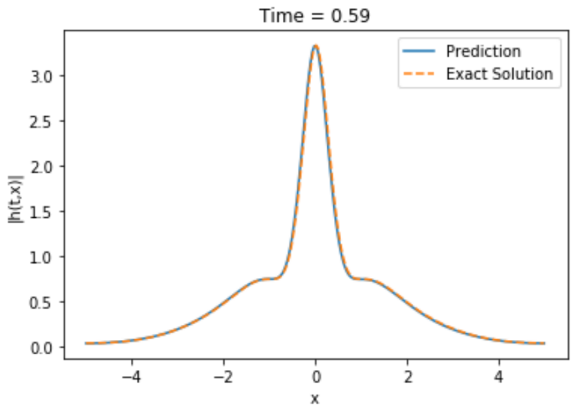


图 14: t=0.59

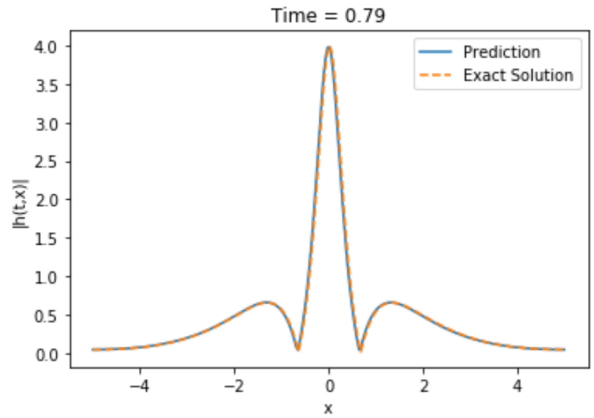


图 15: t=0.79

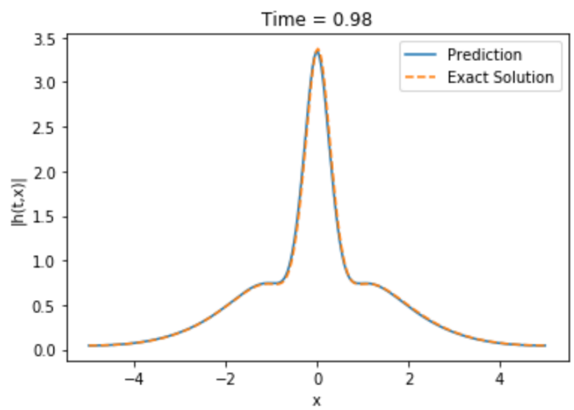


图 16: t=0.98

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。

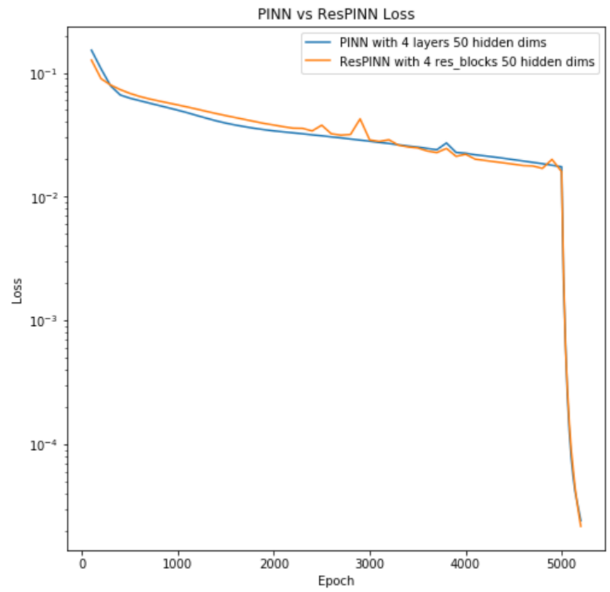


图 17: Loss 比较图

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。

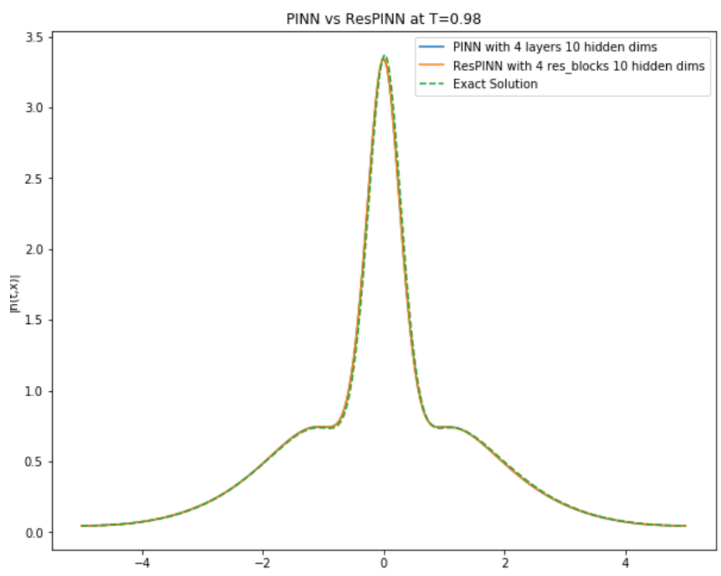


图 18: 解比较图

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。

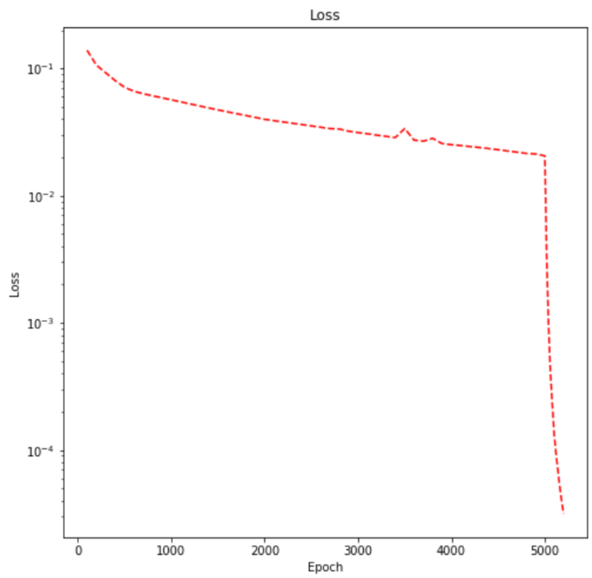


图 19: t=0.59

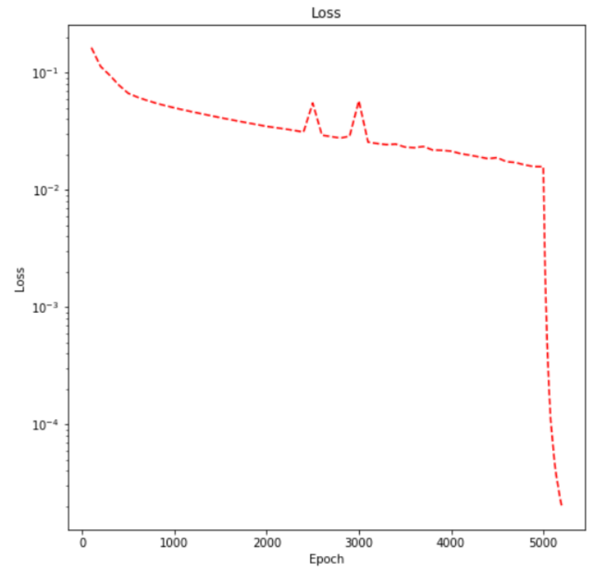


图 20: t=0.79

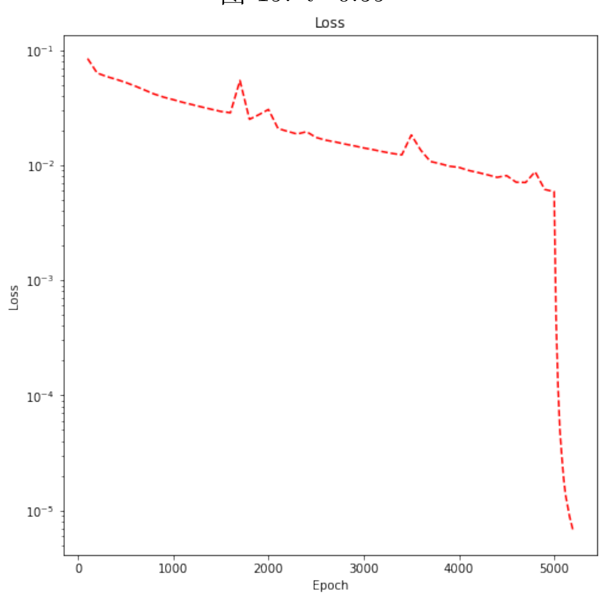


图 21: t=0.98

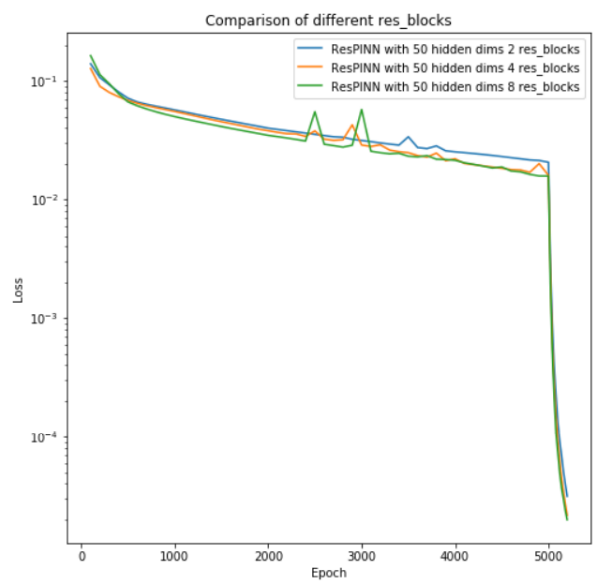


图 22: t=0.98

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 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：

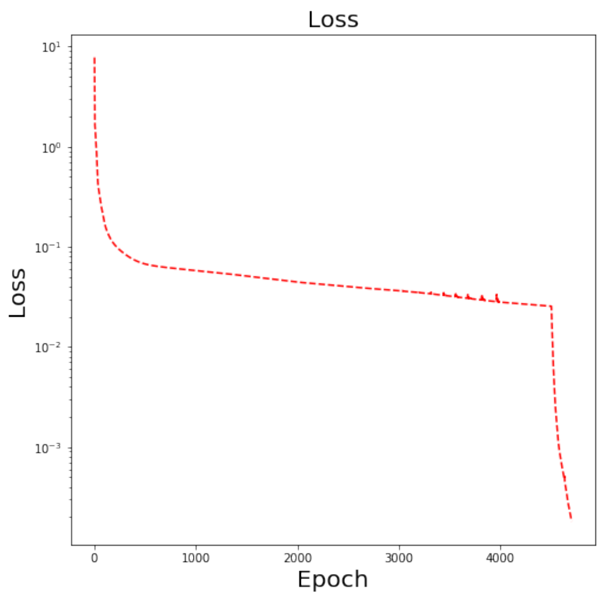


图 23: dim_hidden=20

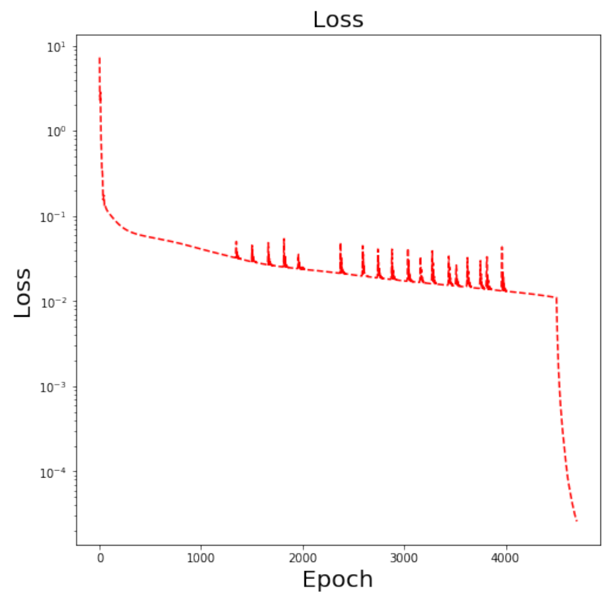


图 24: dim_hidden=35

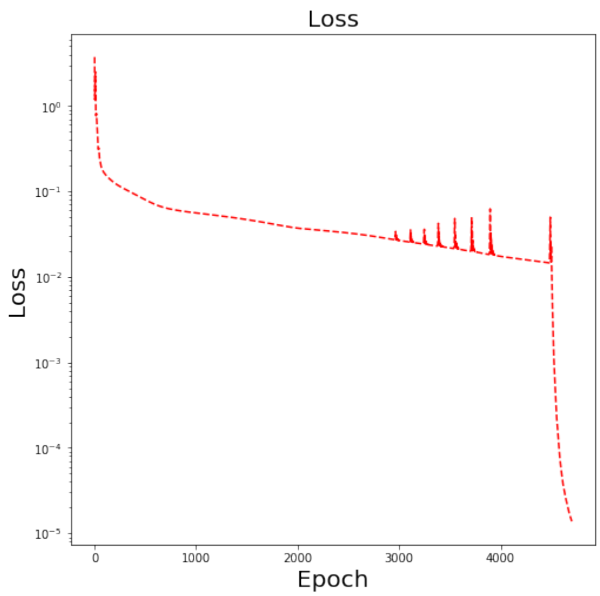


图 25: dim_hidden=50

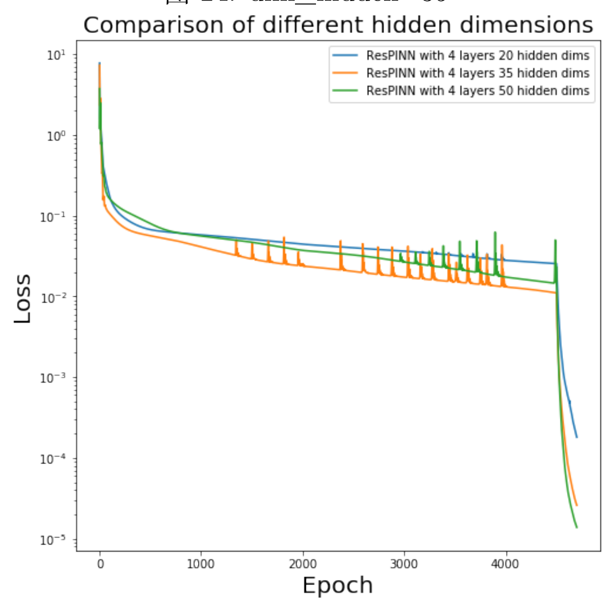


图 26: 对比图

可以发现，随着 dim_hidden 的增加，loss 值在训练过程中会由出现一些起伏，但是我们一看发现，dim_hidden 为 35 时，loss 下降最快，所以，并非 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 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：

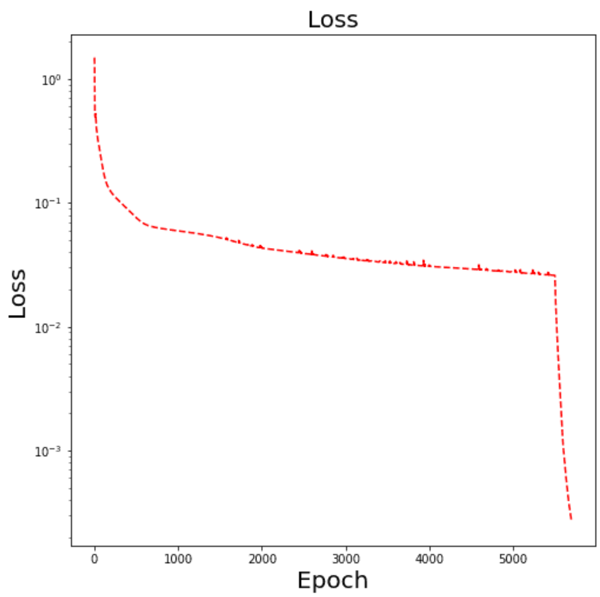


图 27: dim_hidden=20

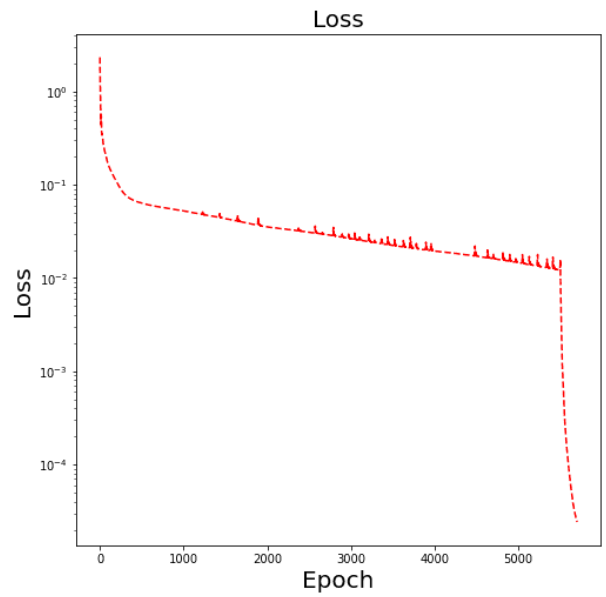


图 28: dim_hidden=35

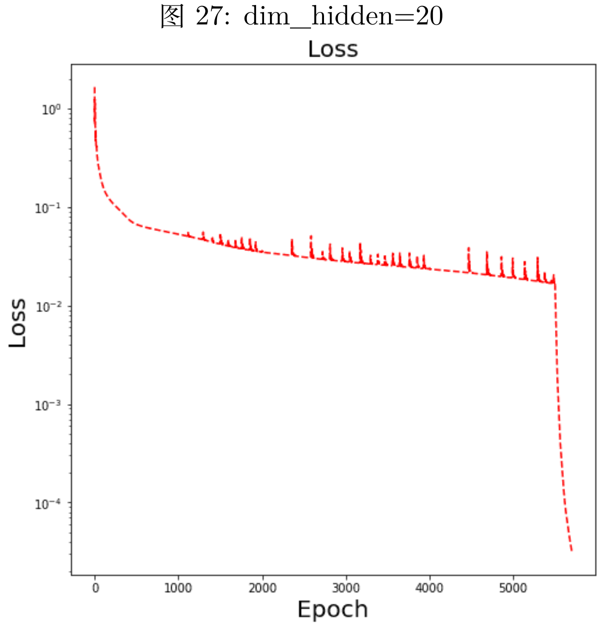


图 29: dim_hidden=50

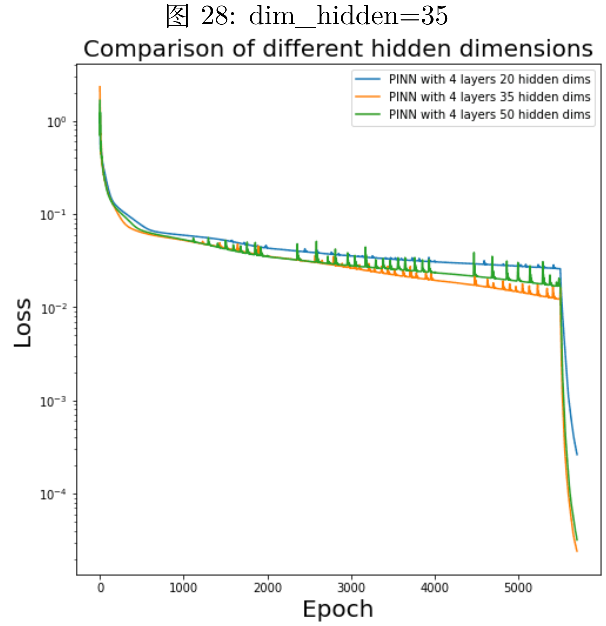


图 30: 对比图

同样地，我们依旧发现 dim_hidden 增大会使 loss 值产生一些起伏，并且这里依旧是 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

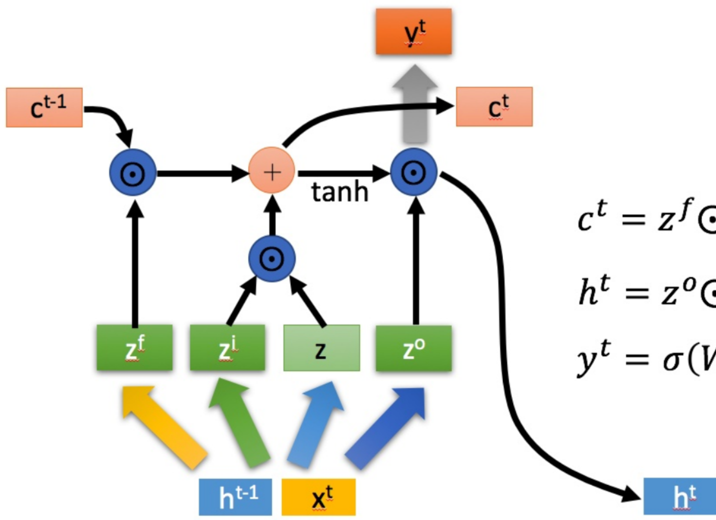
具体数据处理相关代码如下：

```
def centralize(data):
    min_value = np.min(data,axis=0)
    max_value = np.max(data,axis=0)
    data = (data - min_value) / (max_value-min_value)
    return data

def judge(dataset,k):
    '''
    to see at day k, if each stock rise or fall
    '''
    pr_today = dataset[k]
    pr_yesterday = dataset[k-1]
    pr_change = pr_today - pr_yesterday
    med = np.median(pr_change)
    re = np.zeros_like(pr_change)
    re[pr_change>0] = 1
    return(re)

def create_dataset(dataset,look_back=240):
    dataX,dataY=[],[]
    for i in range(len(dataset)-look_back):
        pr_change = judge(dataset,i+look_back)
        a = dataset[i:(i+look_back)]
        dataX.append(a)
        dataY.append(pr_change)
```

```python
    return np.array(dataX),np.array(dataY)

look_back = 240
index_used = index[look_back:]
index_used = np.array(index_used)
X, Y = create_dataset(dataset,look_back)
print(X.shape, Y.shape)
a,b,c = X.shape

train_size = int(len(X) * 0.9)
valid_size = len(X) - train_size
index_size = int(len(index_used)*0.9)
print(train_size, valid_size)

X_train = X[:train_size]
Y_train = Y[:train_size]
index_train = index_used[:index_size]


X_valid = X[train_size:]
Y_valid = Y[train_size:]
index_valid = index_used[index_size:]


# X_train = X_train.reshape(-1,198,240)
# X_valid = X_valid.reshape(-1,198,240)
# Y_train = Y_train.reshape(-1,198,1)

X_train = X_train.reshape(train_size*c,b,1)
Y_train = Y_train.reshape(train_size*c,1)
X_valid = X_valid.reshape(valid_size*c,b,1)
Y_valid = Y_valid.reshape(valid_size*c,1)

# X_train = X_train.transpose(1, 0, 2)
# X_valid = X_valid.transpose(1, 0, 2)

X_train = torch.from_numpy(X_train)
Y_train = torch.from_numpy(Y_train)
X_valid = torch.from_numpy(X_valid)

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 Gloves) 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。



$$c^t = z^f \odot c^{t-1} + z^i \odot z$$

$$h^t = z^o \odot tanh(c^t)$$

$$y^t = \sigma(W' h^t)$$

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^0 来进行控制的。并且还对上一阶段得到的 c^0 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_laryers 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：

```
%%time
class LSTMRegression(nn.Module):
    def __init__(self, input_size, hidden_size, output_size=1, num_layers=1):
        super().__init__()
        self.lstm = nn.LSTM(input_size, hidden_size, num_layers, batch_first=True)
        self.linear = nn.Linear(hidden_size, output_size)

    def forward(self, x):
        _, (hn, cn) = self.lstm(x)
        hn = hn.squeeze()
        out = self.linear(hn)
        return out

model = LSTMRegression(input_size=1, hidden_size=5, output_size=1)

criterion = torch.nn.BCEWithLogitsLoss()     #交叉熵BCEWithLogitsLoss()和MultiLabelSoftMarginLoss()
#criterion = torch.nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=1e-3)
#optimizer = optim.SGD(model.parameters(), lr=1e-1)

epochs = 100
```

```python
batch_size = 30
batch = X_train.shape[0] // batch_size



torch_dataset = Data.TensorDataset(torch.tensor(X_train), torch.tensor(
Y_train))
# 把 dataset 放入 DataLoader
loader = Data.DataLoader(
    dataset=torch_dataset,  # torch TensorDataset format
    batch_size=batch_size,  # mini batch size
    shuffle=True,  #
    num_workers=10,  # 多线程来读数据
)

loss_epoch = np.zeros(epochs)
acc_epoch = np.zeros(epochs)
loss_valid = np.zeros(epochs)
acc_valid = np.zeros(epochs)
for epoch in range(epochs):
    loss_ep = np.array([])
    acc_ep = np.array([])
    loss_epv = np.array([])
    acc_epv = np.array([])
    for step,(var_x,var_y) in enumerate(loader):
        out = model(var_x)
        out_f = out.detach().clone().numpy()
        var_yf = var_y.detach().clone().numpy()
        loss = criterion(out, var_y)
        loss_f = loss.detach().clone().numpy()
        acc_ep = np.append(acc_ep,acc(out_f,var_yf))
        loss_ep = np.append(loss_ep,loss_f)

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

    if (epoch + 1) % 5 == 0:
        print(f'Epoch: {epoch:5d}, Loss: {np.mean(loss_ep):.4e}, Acc:{np
        .mean(acc_ep):.4e}')

```

```
    loss_epoch[epoch] = np.mean(loss_ep)
    acc_epoch[epoch] = np.mean(acc_ep)

    Y_pre = model(X_valid)  #计算验证集表现
    Y_pre1 = Y_pre.clone().detach().numpy()
    Y_valid1 = torch.from_numpy(Y_valid)
    loss_valid[epoch] = criterion(Y_pre,Y_valid1)
    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：

```
# visulize
kind = 2
series = np.arange(kind*len(index_valid),(kind+1)*len(index_valid))
Y_pred_re = Y_pred
Y_pred_re[Y_pred_re>0] = 1
Y_pred_re[Y_pred_re<=0] = 0



filename1 = 'point' + str(index) + '.png'
filename2 = 'pic' + str(index) + '.png'

fig = plt.figure()
ax = plt.subplot()
type1 = ax.scatter(index_valid, Y_valid[series], alpha=0.6,color='b'
,label='groundtruth')
type2 = ax.scatter(index_valid, Y_pred_re[series], alpha=0.3,color='
r',label='prediction')
plt.xlabel("date time")
plt.ylabel("0 for fall, 1 for rise")
ax.legend((type1, type2), (u'groundtruth', u'prediction'), loc='best
')
plt.savefig(filename1)
plt.show()

```

```
    plt.plot(loss_epoch, 'r-', label='loss')
    plt.plot(acc_epoch, 'b-', label='accurate rate')
    plt.legend(loc='best')
    plt.savefig(filename2)
    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：

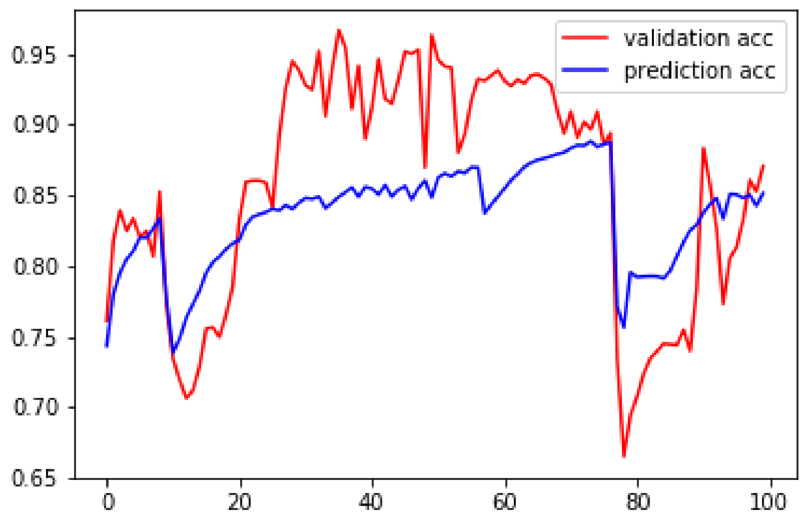


图 32: 单向 LSTM 网络训练集和验证集上的正确率变化

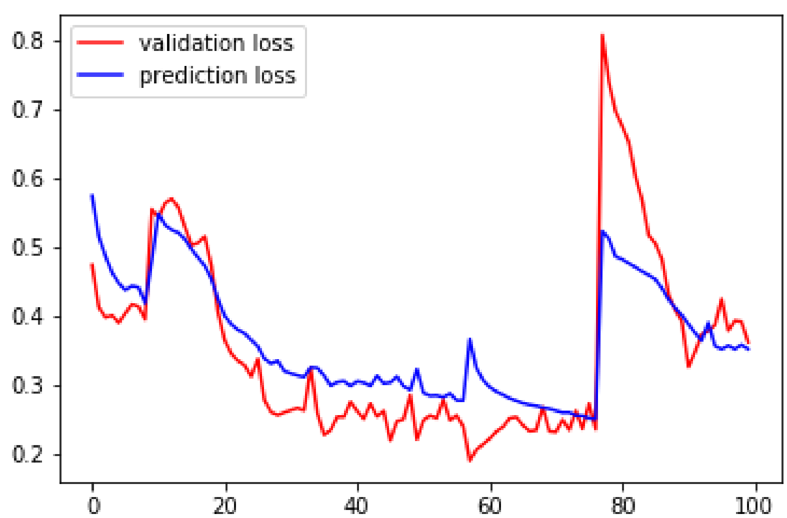


图 33: 单向 LSTM 网络训练集和验证集上的 loss 值变化

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：

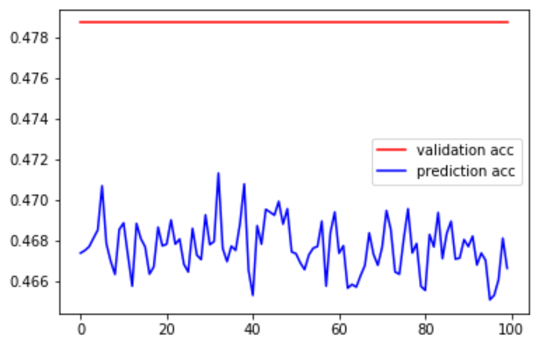


图 34: 双向 LSTM 网络训练集和验证集上的正确率变化

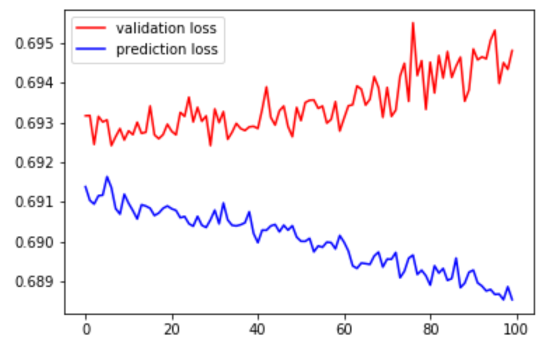


图 35: 双向 LSTM 网络训练集和验证集上的 loss 变化

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

[1] 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：

[1]本节代码附在报告最后

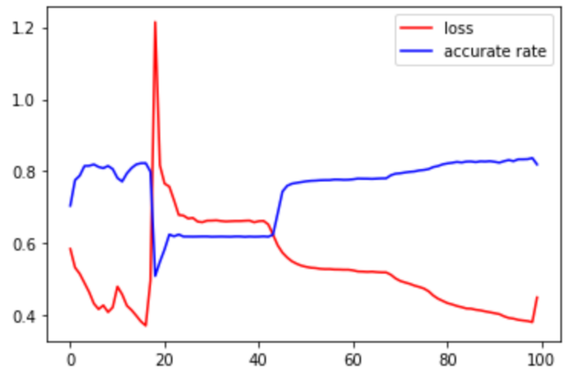


图 36: hidden size=3

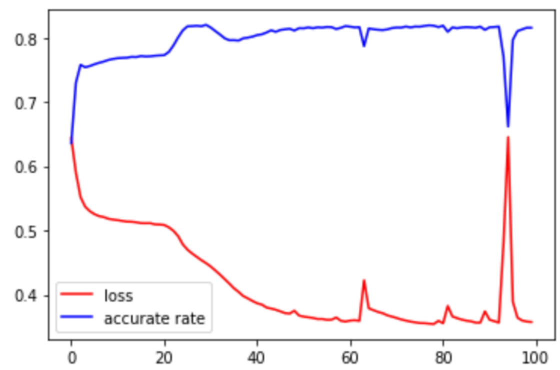


图 37: hidden size=4

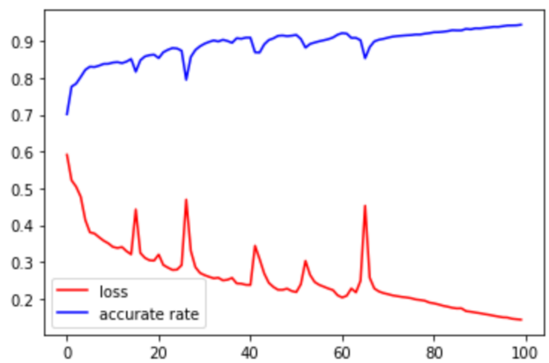


图 38: hidden size=5

训练得到的模型在验证集上的表现如下所示:

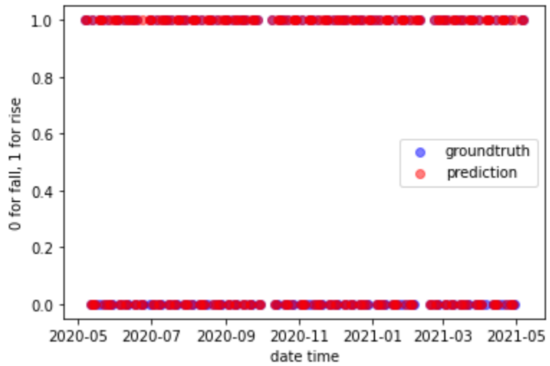


图 39: hidden size=3

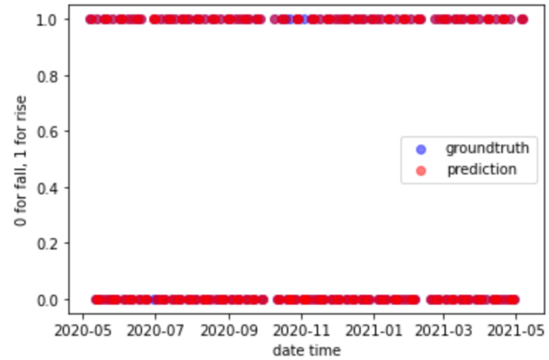


图 40: hidden size=4

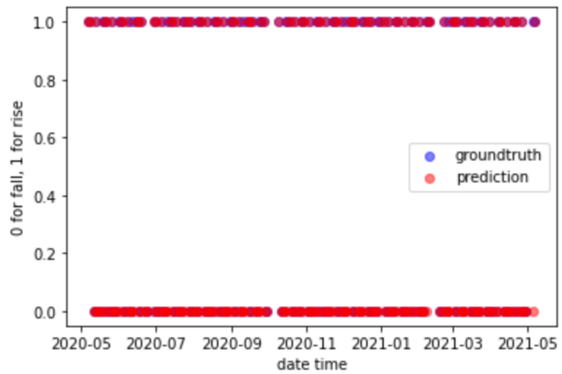


图 41: hidden size=5

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：

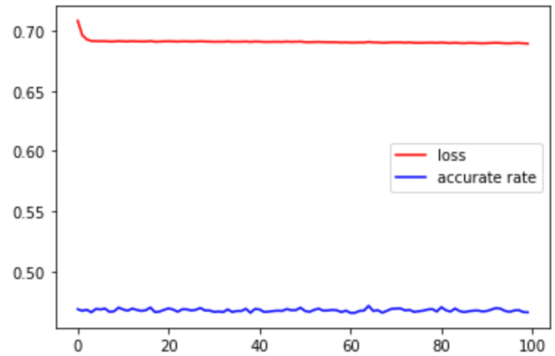


图 42: hidden size=2

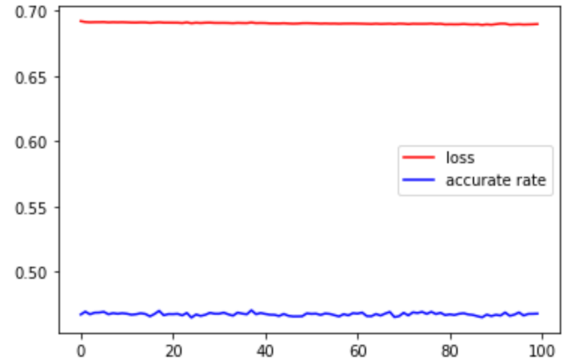


图 43: hidden size=3

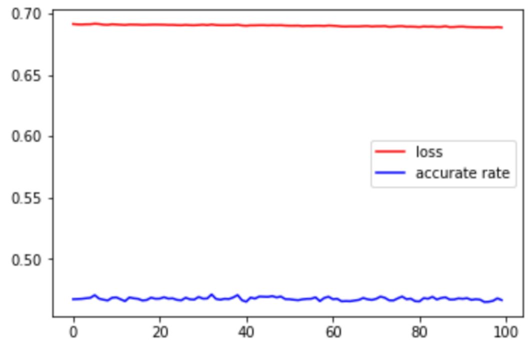


图 44: hidden size=4

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

[2] 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。

[2] 本节代码附在报告最后

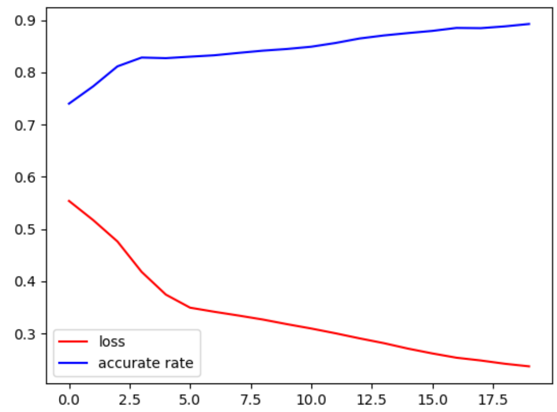


图 45: batch size=10

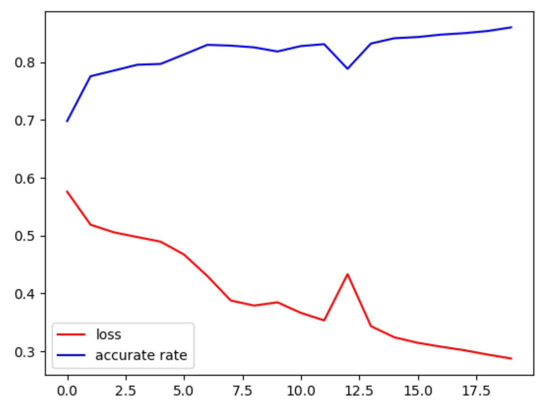


图 46: batch size=30

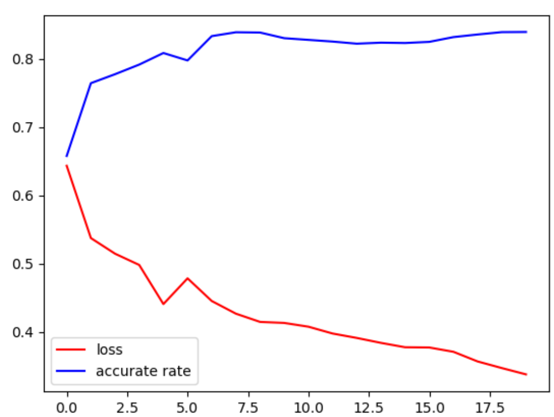


图 47: batch size=60,epoch=20

2.5 不同的学习率对结果的影响

[3] 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：

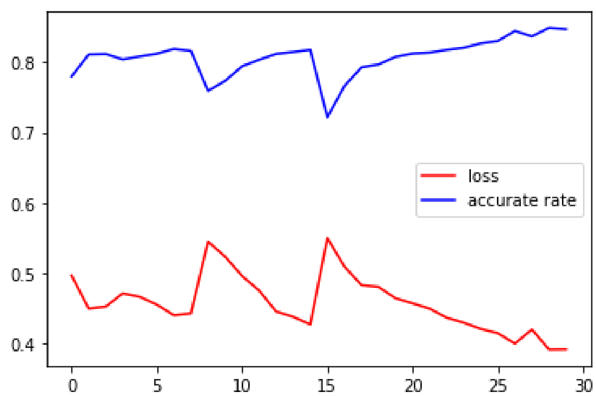


图 48: lr=0.01

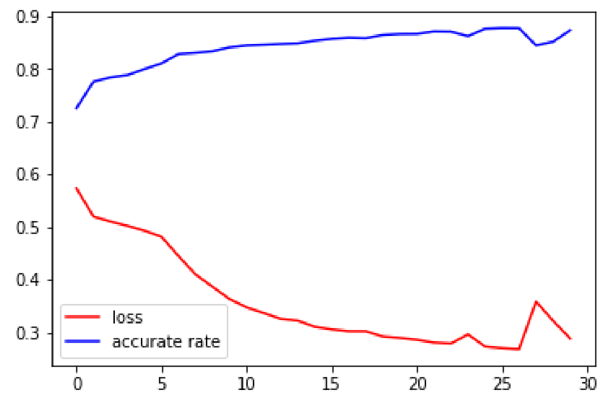


图 49: lr=0.001

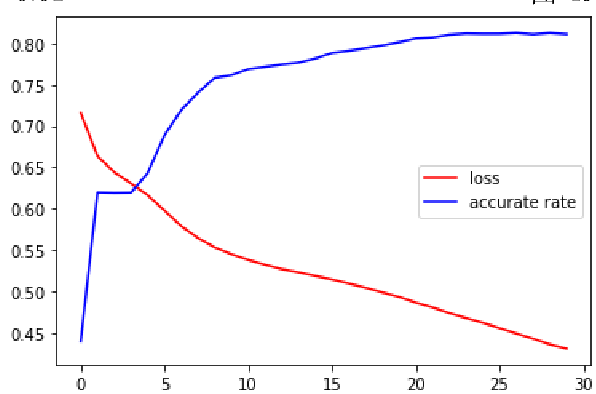


图 50: lr=0.0001

以下列出了训练过程中的 acc。

[3]本节代码附在报告最后

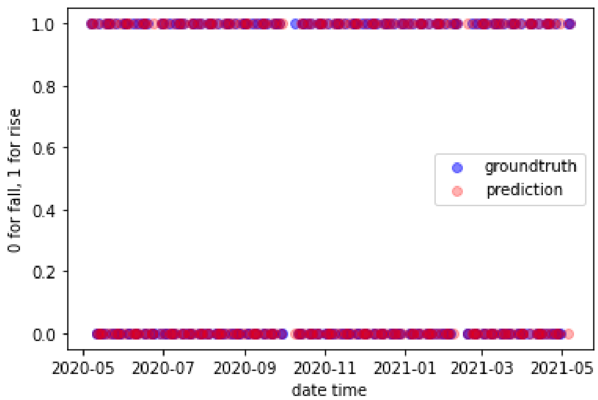


图 51: lr=0.01，验证集结果

图 52: lr=0.001，验证集结果

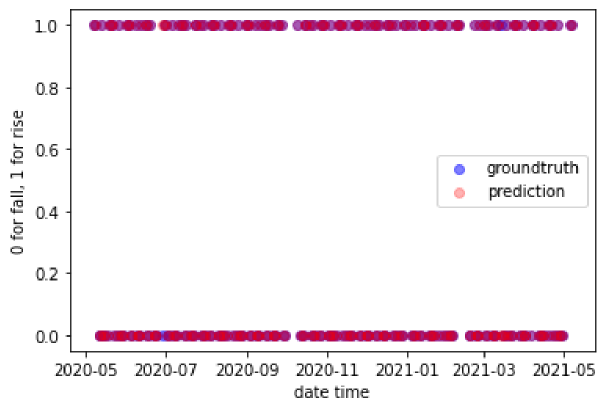


图 53: lr=0.0001，验证集结果

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,1r=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

[4] 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：

[4] 本节代码附在报告最后

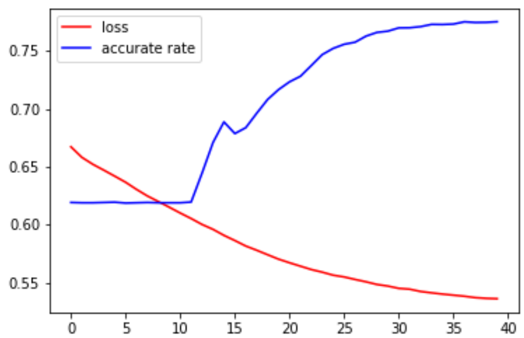


图 54: SGD

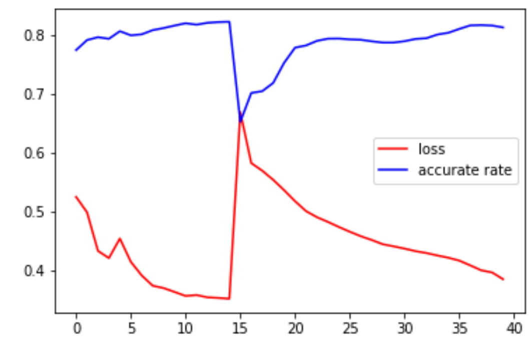


图 55: Adam

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：

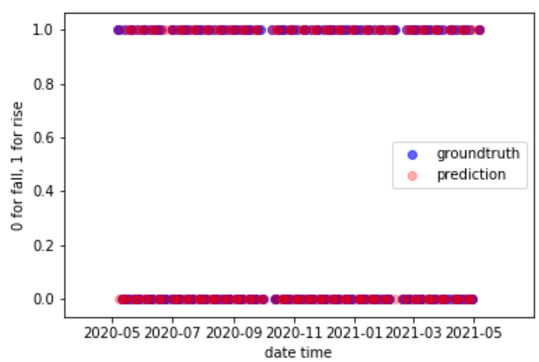


图 56: SGD

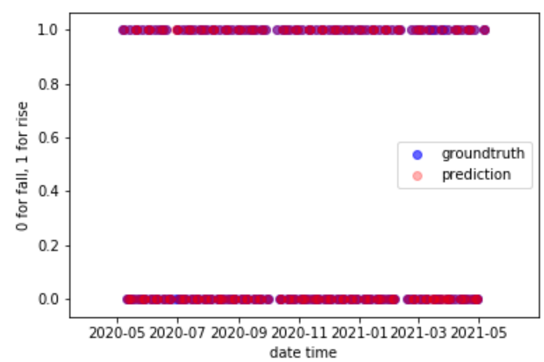


图 57: Adam

附录

A 神经网络求解偏微分方程代码

A.1 网络架构

```python
import torch
import torch.nn as nn
import torch.optim as optim
from torch.optim.lr_scheduler import StepLR
import os
import numpy as np
import shutil
import matplotlib.pyplot as plt
from pyDOE import lhs
import argparse
import scipy.io

def activation(name):
    """define all the activation function
    """
    if name in ['tanh', 'Tanh']:
        return nn.Tanh()
    elif name in ['relu', 'ReLU']:
        return nn.ReLU(inplace=True)
    elif name in ['leaky_relu', 'LeakyReLU']:
        return nn.LeakyReLU(inplace=True)
    elif name in ['sigmoid', 'Sigmoid']:
        return nn.Sigmoid()
    elif name in ['softplus', 'Softplus']:
        return nn.Softplus()
    else:
        raise ValueError(f'unknown activation function: {name}')

class Problem(object):
    """
    rewrite the problem;define its domain,
    """
    def __init__(self, domain=(0,np.pi/2,-5,5)):
        self.domain = domain

```

```
    def __repr__(self):
        return f'{self.__doc__}'

    def ic(self, x):
        x_=x.detach().cpu().numpy()
        f_ic=-2/np.cosh(x_[:,1])
        f_ic=f_ic.reshape(f_ic.size,1)
        return torch.from_numpy(f_ic).float()

    def true_value(self, t):
        data = scipy.io.loadmat('NLS.mat')
        x = data['x'].flatten()[:, None]
        Exact = data['uu']
        Exact_u = np.real(Exact)
        Exact_v = np.imag(Exact)
        t=400*t/np.pi
        t=np.int(t)
        ut = Exact_u[:,t]
        vt = Exact_v[:,t]
        ht=np.sqrt(ut**2+vt**2)
        ht=ht.reshape(ht.size,1)
        return ht

def grad(outputs, inputs):
    """compute the derivative of outputs associated with inputs
       input: (N,D) tensors
       output:(N,1) tensors
    """
    return torch.autograd.grad(outputs, inputs,grad_outputs=torch.ones_like(outputs),create_graph=True)

class DNN(nn.Module):
    def __init__(self, dim_in, dim_out, dim_hidden, hidden_layers, act_name='tanh', init_name=None):
        super().__init__()
        model = nn.Sequential()
        model.add_module('fc0', nn.Linear(dim_in, dim_hidden, bias=True))
        model.add_module('act0', activation(act_name))
        for i in range(1, hidden_layers):
            model.add_module(f'fc{i}', nn.Linear(dim_hidden, dim_hidden,
```

```
                bias=True))
            model.add_module(f'act{i}', activation(act_name))
        model.add_module(f'fc{hidden_layers}', nn.Linear(dim_hidden, dim_out, bias=True))
        self.model = model
        if init_name is not None:
            self.init_weight(init_name)

    def init_weight(self, name):
        if name == 'xavier_normal':
            nn_init = nn.init.xavier_normal_
        elif name == 'xavier_uniform':
            nn_init = nn.init.xavier_uniform_
        elif name == 'kaiming_normal':
            nn_init = nn.init.kaiming_normal_
        elif name == 'kaiming_uniform':
            nn_init = nn.init.kaiming_uniform_
        else:
            raise ValueError(f'unknown initialization function: {name}')
        for param in self.parameters():
            if len(param.shape) > 1:
                nn_init(param)

    def forward(self, x):
        return self.model(x)

    def forward_test(self, x):
        print(f"{'input':<20}{str(x.shape):<40}")
        for name, module in self.model._modules.items():
            x = module(x)
            print(f"{name:<20}{str(x.shape):<40}")
        return x

    def model_size(self):
        n_params = 0
        for param in self.parameters():
            n_params += param.numel()
        return n_params

class ResBlock(nn.Module):
    def __init__(self, dim_in, dim_out, dim_hidden, act_name='tanh'):
```

```python
        super().__init__()

        assert(dim_in == dim_out)
        block = nn.Sequential()
        block.add_module('act0', activation(act_name))
        block.add_module('fc0', nn.Linear(dim_in, dim_hidden, bias=True))
        block.add_module('act1', activation(act_name))
        block.add_module('fc1', nn.Linear(dim_hidden, dim_out, bias=True))
        self.block = block

    def forward(self, x):
        identity = x
        out = self.block(x)
        return identity + out

class ResDNN(nn.Module):
    def __init__(self, dim_in, dim_out, dim_hidden, res_blocks, act_name='tanh', init_name='kaiming_normal'):
        super().__init__()

        model = nn.Sequential()
        model.add_module('fc_first', nn.Linear(dim_in, dim_hidden, bias=True))
        for i in range(res_blocks):
            res_block = ResBlock(dim_hidden, dim_hidden, dim_hidden, act_name=act_name)
            model.add_module(f'res_block{i+1}', res_block)
        model.add_module('act_last', activation(act_name))
        model.add_module('fc_last', nn.Linear(dim_hidden, dim_out, bias=True))

        self.model = model
        if init_name is not None:
            self.init_weight(init_name)

    def init_weight(self, name):
        if name == 'xavier_normal':
            nn_init = nn.init.xavier_normal_
        elif name == 'xavier_uniform':
```

```
            nn_init = nn.init.xavier_uniform_
        elif name == 'kaiming_normal':
            nn_init = nn.init.kaiming_normal_
        elif name == 'kaiming_uniform':
            nn_init =  nn.init.kaiming_uniform_
        else:
            raise ValueError(f'unknown initialization function: {name}')

        for param in self.parameters():
            if len(param.shape) > 1:
                nn_init(param)

    def forward(self, x):
        return self.model(x)

    def forward_test(self, x):
        print(f"{'input':<20}{str(x.shape):<40}")
        for name, module in self.model._modules.items():
              x = module(x)
              print(f"{name:<20}{str(x.shape):<40}")
        return x

    def model_size(self):
        n_params = 0
        for param in self.parameters():
            n_params += param.numel()
        return n_params

class PINN(DNN):
    def __init__(self, dim_in, dim_out, dim_hidden, hidden_layers, act_name='tanh', init_name='xavier_normal'):
        super().__init__(dim_in, dim_out, dim_hidden, hidden_layers, act_name=act_name, init_name=init_name)
    def forward(self, x):
        x.requires_grad_(True)
        h= super().forward(x)
        a = torch.split(h, 1, dim=1)
        u = a[0]
        v = a[1]
        grad_u = grad(u, x)[0]
        u_x = grad_u[:, [1]]
```

```
        u_t = grad_u[:, [0]]
        u_xx = grad(u_x, x)[0][:, [1]]
        grad_v = grad(v, x)[0]
        v_x = grad_v[:, [1]]
        v_t = grad_v[:, [0]]
        v_xx = grad(v_x, x)[0][:, [1]]
        f_u = u_t + 0.5*v_xx + (u**2 + v**2)*v
        f_v = v_t - 0.5*u_xx - (u**2 + v**2)*u
        return u, v, u_x, v_x, f_u, f_v

class ResPINN(ResDNN):
    def __init__(self, dim_in, dim_out, dim_hidden, res_blocks,act_name=
    'tanh', init_name='xavier_normal'):
        super().__init__(dim_in, dim_out, dim_hidden, res_blocks,
        act_name=act_name, init_name=init_name)
    def forward(self,x):
        x.requires_grad_(True)
        h = super().forward(x)
        a = torch.split(h, 1, dim=1)
        u = a[0]
        v = a[1]
        grad_u = grad(u, x)[0]
        u_x = grad_u[:, [1]]
        u_t = grad_u[:, [0]]
        u_xx = grad(u_x, x)[0][:, [1]]
        grad_v = grad(v, x)[0]
        v_x = grad_v[:, [1]]
        v_t = grad_v[:, [0]]
        v_xx = grad(v_x, x)[0][:, [1]]
        f_u = u_t + 0.5*v_xx + (u**2+v**2)*v
        f_v = v_t - 0.5*u_xx - (u**2+v**2)*u
        return u, v, u_x, v_x, f_u, f_v

class Options(object):
    def __init__(self):
        parser = argparse.ArgumentParser()
        parser.add_argument('--no_cuda', action='store_true', default=
        False, help='disable CUDA or not')
        parser.add_argument('--dim_hidden', type=int, default=50, help='
        neurons in hidden layers')
        parser.add_argument('--hidden_layers', type=int, default=4, help
```

```
='number of hidden layers')
        parser.add_argument('--res_blocks', type=int, default=4, help='number of residual blocks')
        parser.add_argument('--lam', type=float, default=1, help='weight in loss function')
        parser.add_argument('--lr', type=float, default=1e-3, help='initial learning rate')
        parser.add_argument('--epochs_Adam', type=int, default=4500, help='epochs for Adam optimizer')
        parser.add_argument('--epochs_LBFGS', type=int, default=200, help='epochs for LBFGS optimizer')
        parser.add_argument('--step_size', type=int, default=2000, help='step size in lr_scheduler for Adam optimizer')
        parser.add_argument('--gamma', type=float, default=0.7, help='gamma in lr_scheduler forAdam optimizer')
        parser.add_argument('--resume', type=bool, default=False, help='resume or not')
        parser.add_argument('--sample_method', type=str, default='lhs', help='sample method')
        parser.add_argument('--n_x', type=int, default=100, help='sample points in x-direction for uniform sample')
        parser.add_argument('--n_t', type=int, default=100, help='sample points in t-direction for uniform sample')
        parser.add_argument('--n', type=int, default=10000, help='sample points in domain for lhs sample')
        parser.add_argument('--n_bc', type=int, default=400, help='sample points on the boundary for lhs sample')
        parser.add_argument('--n_ic', type=int, default=400, help='sample points on the initial time for lhs sample')
        self.parser = parser
    def parse(self):
        arg = self.parser.parse_args(args=[])
        arg.cuda = not arg.no_cuda and torch.cuda.is_available()
        arg.device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')  #choose the environment according to whether you have cuda or not.

        return arg
def save_model(state, is_best=None, save_dir=None):
    """save the best and the last model
    """
```

```
    last_model = os.path.join(save_dir, 'last_model.pth.tar')
    torch.save(state, last_model)
    if is_best:
        best_model = os.path.join(save_dir, 'best_model.pth.tar')
        shutil.copyfile(last_model, best_model)
```

A.2 训练集

```
class Trainset(object):

    def __init__(self, *args, **kwargs):
        self.domain = (0,np.pi/2,-5,5)
        self.args = args
        self.method = kwargs['method']

    def __call__(self,verbose=None):
        if self.method == 'uniform':
            nx, nt, n_bc, n_ic = self.args[0], self.args[1], self.args
            [2], self.args[3]
            x, x_bc_left, x_bc_right, x_ic = self._uniform_sample(nx, nt
            , n_bc, n_ic)
        elif self.method == 'lhs':
            n, n_bc, n_ic = self.args[0], self.args[1], self.args[2]
            x, x_bc_left, x_bc_right, x_ic = self._lhs_sample(n, n_bc,
            n_ic)

        if verbose == 'tensor':
            x = torch.from_numpy(x).float()
            x_bc_left = torch.from_numpy(x_bc_left).float()
            x_bc_right = torch.from_numpy(x_bc_right).float()
            x_ic = torch.from_numpy(x_ic).float()
            return x, x_bc_left, x_bc_right, x_ic
        return x, x_bc_left, x_bc_right, x_ic
    def _uniform_sample(self, nx,nt, n_bc, n_ic):
        t_min, t_max, x_min, x_max = self.domain
        x = np.linspace(x_min,x_max,nx)
        t = np.linspace(t_min, t_max, nt)
        t, x = np.meshgrid(t, x)
        tx = np.hstack((t.reshape(t.size, -1), x.reshape(x.size, -1)))

        t = np.linspace(t_min, t_max, n_bc)
```

```
        t, xl = np.meshgrid(t, x_min)
        x_bc_left = np.hstack((t.reshape(t.size, -1), xl.reshape(xl.size
        , -1)))
        t, xr = np.meshgrid(t, x_max)
        x_bc_right = np.hstack((t.reshape(t.size, -1), xr.reshape(xr.
        size, -1)))

        x = np.linspace(x_min,x_max,n_ic)
        t_ic, x = np.meshgrid(t_min, x)
        x_ic = np.hstack((t_ic.reshape(t_ic.size, -1), x.reshape(x.size,
         -1)))
        return tx, x_bc_left, x_bc_right, x_ic

    def _lhs_sample(self, n, n_bc, n_ic):
        t_min, t_max, x_min, x_max = self.domain
        lb = np.array([t_min, x_min])
        ub = np.array([t_max, x_max])
        x = lb + (ub - lb) * lhs(2, n)
        lb = np.array([t_min, x_min])
        ub = np.array([t_max, x_min])
        temp = lb + (ub - lb) * lhs(2, n_bc)
        x_bc_left = temp
        lb = np.array([t_min, x_max])
        ub = np.array([t_max, x_max])
        temp = lb + (ub - lb) * lhs(2, n_bc)
        x_bc_right = temp
        lb = np.array([t_min, x_min])
        ub = np.array([t_min, x_max])
        temp = lb + (ub - lb) * lhs(2, n_ic)
        x_ic = temp
        return x, x_bc_left, x_bc_right, x_ic
```

A.3 验证集

```
class Testset(object):
    """The dataset is based on a square domain
    """

    def __init__(self, *args, **kwargs):
        self.domain = (0,np.pi/2,-5,5)
        self.args = args
```

```python
        self.method = kwargs['method']

    def __repr__(self):
        return f'{self.__doc__}'

    def __call__(self, plot=False, verbose=None):
        if self.method == 'uniform':
            n_x, t = self.args[0], self.args[1]
            X, x = self._uniform_sample(n_x, t)
        if verbose == 'tensor':
            X = torch.from_numpy(X).float()
        return X, x

    def _uniform_sample(self, n_x, t):
        t_min, t_max, x_min, x_max = self.domain
        x = np.linspace(x_min, x_max, n_x)
        t, x = np.meshgrid(t, x)
        X = np.hstack((t.reshape(t.size, -1), x.reshape(x.size, -1)))
        return X, x
```

A.4 训练过程

```python
class Trainer(object):
    def __init__(self, args): # args includs all paraments needed in the
     net
        self.device  = args.device  #cpu or gpu
        self.problem = args.problem

        self.lam = args.lam       # weight of the initial model
        self.criterion = nn.MSELoss()

        self.model = args.model
        self.model_name = self.model.__class__.__name__
        self.model_path = self._model_path()

        self.epochs_Adam = args.epochs_Adam
        self.epochs_LBFGS = args.epochs_LBFGS
        self.optimizer_Adam = optim.Adam(self.model.parameters(), lr=
        args.lr)  # learning rate
        self.optimizer_LBFGS = optim.LBFGS(self.model.parameters(),
                                            max_iter=20,
```

```
                                                tolerance_grad=1.e-8,
                                                tolerance_change=1.e-12)   # everytime a parement is updataed, an equation is solved and so there has to be a maxinum of the iteration
        self.lr_scheduler = StepLR(self.optimizer_Adam,
                                   step_size=args.step_size,
                                   gamma=args.gamma)

        self.model.to(self.device)
        self.model.zero_grad()

        self.x, self.x_bc_left, self.x_bc_right, self.x_ic = args.trainset(verbose='tensor')
        self.x_val, self.x_bc_left_val, self.x_bc_right_val, self.x_ic_val = args.validset(verbose='tensor')
        self.true_ic = args.problem.ic(self.x_ic)
        self.true_ic_val = args.problem.ic(self.x_ic_val)
        if self.device == torch.device(type='cuda'):
            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)
            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)
            self.true_ic, self.true_ic_val = self.true_ic.to(self.device), self.true_ic_val.to(self.device)

    def _model_path(self):
        if not os.path.exists('checkpoints'):
            os.mkdir('checkpoints')
            #path = os.path.join('checkpoints', self.model_name, f'{args.dim_hidden}_{args.hidden_layers}')
        path = f'checkpoints/{self.model_name}_{args.dim_hidden}_{args.hidden_layers}'
        if not os.path.exists(path):
            os.mkdir(path)
        return path

```

```
    def train(self):
        best_loss = 1.e10
        for epoch in range(self.epochs_Adam):
            loss, loss1, loss2, loss3 = self.train_Adam()
            if (epoch + 1) % 2 == 0:
                # self.infos_Adam(epoch + 1, loss, loss1, loss2, loss3)
                nums.append(epoch + 1)
                losses.append(loss)
                valid_loss = self.validate(epoch)
                is_best = valid_loss < best_loss
                best_loss = valid_loss if is_best else best_loss
                state = {
                    'epoch': epoch,
                    'state_dict': self.model.state_dict(),
                    'best_loss': best_loss
                }
                save_model(state, is_best, save_dir=self.model_path)
            if (epoch + 1) % 5 == 0:
                self.infos_Adam(epoch + 1, loss, loss1, loss2, loss3)
        for epoch in range(self.epochs_Adam, self.epochs_Adam + self.
        epochs_LBFGS):
            loss, loss1, loss2, loss3 = self.train_LBFGS()
            if (epoch + 1) % 2 == 0:
                # self.infos_LBFGS(epoch + 1, loss, loss1, loss2, loss3)
                nums.append(epoch + 1)
                losses.append(loss)
                valid_loss = self.validate(epoch)
                is_best = valid_loss < best_loss
                best_loss = valid_loss if is_best else best_loss
                state = {
                    'epoch': epoch,
                    'state_dict': self.model.state_dict(),
                    'best_loss': best_loss
                }
                save_model(state, is_best, save_dir=self.model_path)
            if (epoch + 1) % 5 == 0:
                self.infos_LBFGS(epoch + 1, loss, loss1, loss2, loss3)
    def train_Adam(self):
        self.optimizer_Adam.zero_grad()
        _, _, _, _, f_u, f_v = self.model(self.x)
        u_bc_left, v_bc_left, ux_bc_left, vx_bc_left, _, _ = self.model(
```

```
        self.x_bc_left)
        u_bc_right, v_bc_right, ux_bc_right, vx_bc_right, _, _ = self.model(self.x_bc_right)
        u_ic, v_ic, _, _, _, _ = self.model(self.x_ic)
        h_ic=torch.sqrt(u_ic**2+v_ic**2)
        loss1 = self.criterion(f_u, torch.zeros_like(f_u)) + self.criterion(f_v, torch.zeros_like(f_v))
        loss2 = self.criterion(u_bc_left, u_bc_right) + self.criterion(v_bc_left, v_bc_right) + \
                self.criterion(ux_bc_left, ux_bc_right) + self.criterion(vx_bc_left, vx_bc_right)
        loss3 = self.criterion(u_ic, self.true_ic) + self.criterion(v_ic, torch.zeros_like(self.true_ic))
        loss = loss1 + loss2 + loss3
        loss.backward()
        self.optimizer_Adam.step()
        self.lr_scheduler.step()
        return loss.item(), loss1.item(), loss2.item(), loss3.item()
    def infos_Adam(self, epoch, loss, loss1, loss2, loss3):
        infos = 'Adam ' + \
                f'Epoch #{epoch:5d}/{self.epochs_Adam + self.epochs_LBFGS} ' + \
                f'Loss: {loss:.4e} = {loss1:.4e} + {loss2:.4e} + {loss3:.4e} ' + \
                f'lr: {self.lr_scheduler.get_lr()[0]:.2e} '
        print(infos)
    def train_LBFGS(self):
    # only used to compute loss_int and loss_bc1 for monitoring
        _, _, _, _, f_u, f_v = self.model(self.x)
        u_bc_left, v_bc_left, ux_bc_left, vx_bc_left, _, _ = self.model(self.x_bc_left)
        u_bc_right, v_bc_right, ux_bc_right, vx_bc_right, _, _ = self.model(self.x_bc_right)
        u_ic, v_ic, _, _, _, _ = self.model(self.x_ic)
        h_ic=torch.sqrt(u_ic**2+v_ic**2)
        loss1 = self.criterion(f_u, torch.zeros_like(f_u)) + self.criterion(f_v, torch.zeros_like(f_v))
        loss2 = self.criterion(u_bc_left, u_bc_right) + self.criterion(v_bc_left, v_bc_right) + \
                self.criterion(ux_bc_left, ux_bc_right) + self.criterion(vx_bc_left, vx_bc_right)
```

```
        loss3 = self.criterion(u_ic, self.true_ic) + self.criterion(v_ic, torch.zeros_like(self.true_ic))
       # loss3 = self.criterion(h_ic, self.true_ic)
        loss = loss1 + loss2 + loss3
        def closure():
            if torch.is_grad_enabled():
                self.optimizer_LBFGS.zero_grad()
            _, _, _, _, f_u, f_v = self.model(self.x)
            u_bc_left, v_bc_left, ux_bc_left, vx_bc_left, _, _ = self.model(self.x_bc_left)
            u_bc_right, v_bc_right, ux_bc_right, vx_bc_right, _, _ = self.model(self.x_bc_right)
            u_ic, v_ic, _, _, _, _ = self.model(self.x_ic)
            h_ic = torch.sqrt(u_ic ** 2 + v_ic ** 2)
            loss1 = self.criterion(f_u, torch.zeros_like(f_u)) + self.criterion(f_v, torch.zeros_like(f_v))
            loss2 = self.criterion(u_bc_left, u_bc_right) + self.criterion(v_bc_left, v_bc_right) + \
                    self.criterion(ux_bc_left, ux_bc_right) + self.criterion(vx_bc_left, vx_bc_right)
            loss3 = self.criterion(u_ic, self.true_ic) + self.criterion(v_ic, torch.zeros_like(self.true_ic))
            # loss3 = self.criterion(h_ic, self.true_ic)
            loss = loss1 + loss2 + loss3
            if loss.requires_grad:
                loss.backward()
            return loss
        self.optimizer_LBFGS.step(closure)
        loss = closure()
        return loss.item(), loss1.item(), loss2.item(), loss3

    def infos_LBFGS(self, epoch, loss, loss1, loss2, loss3):
        infos = 'LBFGS ' + \
                f'Epoch #{epoch:5d}/{self.epochs_Adam + self.epochs_LBFGS} ' + \
                f'Loss: {loss:.2e} = {loss1:.2e} + {loss2:.2e} + {loss3:.2e}'
        print(infos)
    def validate(self, epoch):
        self.model.eval()
        _, _, _, _, f_u_val, f_v_val = self.model(self.x_val)
```

```
        u_bc_left_val, v_bc_left_val, ux_bc_left_val, vx_bc_left_val, _, _ = self.model(self.x_bc_left_val)
        u_bc_right_val, v_bc_right_val, ux_bc_right_val, vx_bc_right_val, _, _ = self.model(self.x_bc_right_val)
        u_ic_val, v_ic_val, _, _, _, _ = self.model(self.x_ic_val)
        h_ic_val = torch.sqrt(u_ic_val ** 2 + v_ic_val ** 2)
        loss1 = self.criterion(f_u_val, torch.zeros_like(f_u_val)) + self.criterion(f_v_val, torch.zeros_like(f_v_val))
        loss2 = self.criterion(u_bc_left_val, u_bc_right_val) + self.criterion(v_bc_left_val, v_bc_right_val) + \
                self.criterion(ux_bc_left_val, ux_bc_right_val) + self.criterion(vx_bc_left_val,vx_bc_right_val)
        loss3 = self.criterion(u_ic_val, self.true_ic) + self.criterion(v_ic_val, torch.zeros_like(self.true_ic))
        # loss3 = self.criterion(h_ic_val, self.true_ic_val)
        loss = loss1 + loss2 + loss3
        infos = 'Valid ' + \
                f'Epoch #{epoch + 1:5d}/{self.epochs_Adam + self.epochs_LBFGS} ' + \
                f'Loss: {loss:.4e} '
      # print(infos)
        self.model.train()
        return loss.item()
```

A.5 验证过程

```
class Tester(object):
    def __init__(self, args):
        self.device = args.device
        self.problem = args.problem
        self.criterion = nn.MSELoss()
        self.model = args.model
        model_name = self.model.__class__.__name__
        # model_path = os.path.join('checkpoints',
        # model_name,
        # 'best_model.pth.tar')
        model_path = f'checkpoints/{model_name}_{args.dim_hidden}_{args.hidden_layers}/best_model.pth.tar'
        best_model = torch.load(model_path)
        self.model.load_state_dict(best_model['state_dict'])
        self.model.to(self.device)
```

```
        self.X, self.x= args.testset(verbose='tensor')
        if self.device == torch.device(type='cuda'):
            self.X = self.X.to(self.device)
    def predict(self,t):
        self.model.eval()
        u,v,_,_,_,_ = self.model(self.X)
        u = u.detach().cpu().numpy()
        u = u.reshape(self.x.shape)
        v = v.detach().cpu().numpy()
        v = v.reshape(self.x.shape)
        h=np.sqrt(u**2+v**2)
        truesln=self.problem.true_value(t)
        fig, axes = plt.subplots()
        axes.plot(self.x,h,label='Prediction')
        axes.plot(self.x,truesln,'--',label='Exact Solution')
        axes.legend()
        axes.set_title(f'Time = {t}')
        axes.set_xlabel('x')
        axes.set_ylabel('|h(t,x)|')
        fig.savefig(f't_{t:4f}.png')
        plt.show()
    def pred_result(self,t):
        self.model.eval()
        u,v,_,_,_,_ = self.model(self.X)
        u = u.detach().cpu().numpy()
        u = u.reshape(self.x.shape)
        v = v.detach().cpu().numpy()
        v = v.reshape(self.x.shape)
        h=np.sqrt(u**2+v**2)
        return self.x, h
```

A.6 PINN 与 ResPINN 对比

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

```
nums = []
losses=[]
trainer = Trainer(args)
trainer.train()
loss_path=f'checkpoints/{args.model.__class__.__name__}_{args.dim_hidden
}_{args.hidden_layers}/loss.txt'
np.savetxt(loss_path, np.vstack((nums, losses)).T)
fig, axes = plt.subplots(figsize=(8, 8))
axes.semilogy(nums, losses, 'r--')
axes.set_title('Loss')
axes.set_xlabel('Epoch')
axes.set_ylabel('Loss')
plt.savefig('loss.png')

%time
args = Options().parse()
args.problem = Problem()
# args.model = PINN(dim_in=2,
# dim_out=1,
# dim_hidden=args.dim_hidden,
# hidden_layers=args.hidden_layers,
# act_name='sin',
# dropout=args.dropout)
args.model = ResPINN(2, 2, dim_hidden=args.dim_hidden, res_blocks=args.
hidden_layers)
# args.trainset = Trainset(args.n, args.n_bc, args.n_ic, method='lhs')
args.trainset = Trainset(200, 100, args.n_bc, args.n_ic, method='uniform
')
args.validset = Trainset(args.n_x, args.n_t, args.n_bc, args.n_ic,
method='uniform')
nums = []
losses = []
trainer = Trainer(args)
trainer.train()
loss_path=f'checkpoints/{args.model.__class__.__name__}_{args.dim_hidden
}_{args.hidden_layers}/loss.txt'
np.savetxt(loss_path, np.vstack((nums, losses)).T)
fig, axes = plt.subplots(figsize=(8, 8))
axes.semilogy(nums, losses, 'r')
axes.set_title('Loss')
axes.set_xlabel('Epoch')
```

```
axes.set_ylabel('Loss')
plt.savefig('loss.png')
plt.show()

data1=np.loadtxt('checkpoints/PINN_50_4/loss.txt')
data2=np.loadtxt('checkpoints/ResPINN_50_4/loss.txt')
x=data1[:,0]
loss1=data1[:,1]
loss2=data2[:,1]
fig, axes = plt.subplots(figsize=(8, 8))
axes.semilogy(x, loss1, label='PINN with 4 layers 50 hidden dims')
axes.semilogy(x, loss2, label='ResPINN with 4 res_blocks 50 hidden dims'
)
axes.legend()
axes.set_title('PINN vs ResPINN Loss')
axes.set_xlabel('Epoch')
axes.set_ylabel('Loss')
```

A.7 不同的隐藏层数目对结果的影响

```
%time
args = Options().parse()
args.problem = Problem()
# args.model = PINN(dim_in=2,
# dim_out=1,
# dim_hidden=args.dim_hidden,
# hidden_layers=args.hidden_layers,
# act_name='sin',
# dropout=args.dropout)
args.dim_hidden=10
args.model = PINN(2, 2, dim_hidden=args.dim_hidden, hidden_layers=args.
hidden_layers)
# args.trainset = Trainset(args.n, args.n_bc, args.n_ic, method='lhs')
args.trainset = Trainset(200, 100, args.n_bc, args.n_ic, method='uniform
')
args.validset = Trainset(args.n_x, args.n_t, args.n_bc, args.n_ic,
method='uniform')
nums = []
losses = []
trainer = Trainer(args)
trainer.train()
```

```
loss_path = f'checkpoints/{args.model.__class__.__name__}_{args.dim_hidden}_{args.hidden_layers}/loss.txt'
np.savetxt(loss_path, np.vstack((nums, losses)).T)
fig, axes = plt.subplots(figsize=(8, 8))
axes.semilogy(nums, losses, 'r--')
axes.set_title('Loss')
axes.set_xlabel('Epoch')
axes.set_ylabel('Loss')
plt.savefig('loss.png')

args = Options().parse()
args.problem = Problem()
# args.model = PINN(dim_in=2,
# dim_out=1,
# dim_hidden=args.dim_hidden,
# hidden_layers=args.hidden_layers,
# act_name='sin',
# dropout=args.dropout)
args.dim_hidden=35
args.model = PINN(2, 2, dim_hidden=args.dim_hidden, hidden_layers=args.hidden_layers)
# args.trainset = Trainset(args.n, args.n_bc, args.n_ic, method='lhs')
args.trainset = Trainset(200, 100, args.n_bc, args.n_ic, method='uniform')
args.validset = Trainset(args.n_x, args.n_t, args.n_bc, args.n_ic, method='uniform')
nums = []
losses = []
trainer = Trainer(args)
trainer.train()
loss_path = f'checkpoints/{args.model.__class__.__name__}_{args.dim_hidden}_{args.hidden_layers}/loss.txt'
np.savetxt(loss_path, np.vstack((nums, losses)).T)
fig, axes = plt.subplots(figsize=(8, 8))
axes.semilogy(nums, losses, 'r--')
axes.set_title('Loss')
axes.set_xlabel('Epoch')
axes.set_ylabel('Loss')
plt.savefig('loss.png')

args = Options().parse()
```

```
args.problem = Problem()
# args.model = PINN(dim_in=2,
# dim_out=1,
# dim_hidden=args.dim_hidden,
# hidden_layers=args.hidden_layers,
# act_name='sin',
# dropout=args.dropout)
args.dim_hidden=50
args.model = PINN(2, 2, dim_hidden=args.dim_hidden, hidden_layers=args.
    hidden_layers)
# args.trainset = Trainset(args.n, args.n_bc, args.n_ic, method='lhs')
args.trainset = Trainset(200, 100, args.n_bc, args.n_ic, method='uniform
    ')
args.validset = Trainset(args.n_x, args.n_t, args.n_bc, args.n_ic,
    method='uniform')
nums = []
losses = []
trainer = Trainer(args)
trainer.train()
loss_path = f'checkpoints/{args.model.__class__.__name__}_{args.
    dim_hidden}_{args.hidden_layers}/loss.txt'
np.savetxt(loss_path, np.vstack((nums, losses)).T)
fig, axes = plt.subplots(figsize=(8, 8))
axes.semilogy(nums, losses, 'r--')
axes.set_title('Loss')
axes.set_xlabel('Epoch')
axes.set_ylabel('Loss')
plt.savefig('loss.png')

data1=np.loadtxt('checkpoints/PINN_20_4/loss.txt')
data2=np.loadtxt('checkpoints/PINN_35_4/loss.txt')
data3=np.loadtxt('checkpoints/PINN_50_4/loss.txt')
x=data1[:,0]
loss1=data1[:,1]
loss2=data2[:,1]
loss3=data3[:,1]
fig, axes = plt.subplots(figsize=(8, 8))
axes.semilogy(x, loss1, label='PINN with 4 layers 10 hidden dims')
axes.semilogy(x, loss2, label='PINN with 4 layers 50 hidden dims')
axes.semilogy(x, loss3, label='PINN with 4 layers 100 hidden dims')
axes.legend()
```

```
axes.set_title('Comparison of different hidden dimensions')
axes.set_xlabel('Epoch')
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

```
#hidden=20

%time
args = Options().parse()
args.problem = Problem()
# args.model = PINN(dim_in=2,
# dim_out=1,
# dim_hidden=args.dim_hidden,
# hidden_layers=args.hidden_layers,
# act_name='sin',
# dropout=args.dropout)
args.dim_hidden=20
args.model = PINN(2, 2, dim_hidden=args.dim_hidden, hidden_layers=args.
    hidden_layers)
# args.trainset = Trainset(args.n, args.n_bc, args.n_ic, method='lhs')
args.trainset = Trainset(200, 100, args.n_bc, args.n_ic, method='uniform
    ')
args.validset = Trainset(args.n_x, args.n_t, args.n_bc, args.n_ic,
    method='uniform')
nums = []
losses = []
trainer = Trainer(args)
trainer.train()
loss_path = f'checkpoints/{args.model.__class__.__name__}_{args.
    dim_hidden}_{args.hidden_layers}/loss.txt'
np.savetxt(loss_path, np.vstack((nums, losses)).T)
fig, axes = plt.subplots(figsize=(8, 8))
axes.semilogy(nums, losses, 'r--')
axes.set_title('Loss')
axes.set_xlabel('Epoch')
axes.set_ylabel('Loss')
plt.savefig('loss.png')

#hidden=35

```

```
args = Options().parse()
args.problem = Problem()
# args.model = PINN(dim_in=2,
# dim_out=1,
# dim_hidden=args.dim_hidden,
# hidden_layers=args.hidden_layers,
# act_name='sin',
# dropout=args.dropout)
args.dim_hidden=35
args.model = PINN(2, 2, dim_hidden=args.dim_hidden, hidden_layers=args.
    hidden_layers)
# args.trainset = Trainset(args.n, args.n_bc, args.n_ic, method='lhs')
args.trainset = Trainset(200, 100, args.n_bc, args.n_ic, method='uniform
    ')
args.validset = Trainset(args.n_x, args.n_t, args.n_bc, args.n_ic,
    method='uniform')
nums = []
losses = []
trainer = Trainer(args)
trainer.train()
loss_path = f'checkpoints/{args.model.__class__.__name__}_{args.
    dim_hidden}_{args.hidden_layers}/loss.txt'
np.savetxt(loss_path, np.vstack((nums, losses)).T)
fig, axes = plt.subplots(figsize=(8, 8))
axes.semilogy(nums, losses, 'r--')
axes.set_title('Loss')
axes.set_xlabel('Epoch')
axes.set_ylabel('Loss')
plt.savefig('loss.png')

#hidden=50

args = Options().parse()
args.problem = Problem()
# args.model = PINN(dim_in=2,
# dim_out=1,
# dim_hidden=args.dim_hidden,
# hidden_layers=args.hidden_layers,
# act_name='sin',
# dropout=args.dropout)
args.dim_hidden=50
```

```
args.model = PINN(2, 2, dim_hidden=args.dim_hidden, hidden_layers=args.hidden_layers)
# args.trainset = Trainset(args.n, args.n_bc, args.n_ic, method='lhs')
args.trainset = Trainset(200, 100, args.n_bc, args.n_ic, method='uniform')
args.validset = Trainset(args.n_x, args.n_t, args.n_bc, args.n_ic, method='uniform')
nums = []
losses = []
trainer = Trainer(args)
trainer.train()
loss_path = f'checkpoints/{args.model.__class__.__name__}_{args.dim_hidden}_{args.hidden_layers}/loss.txt'
np.savetxt(loss_path, np.vstack((nums, losses)).T)
fig, axes = plt.subplots(figsize=(8, 8))
axes.semilogy(nums, losses, 'r--')
axes.set_title('Loss')
axes.set_xlabel('Epoch')
axes.set_ylabel('Loss')
plt.savefig('loss.png')


data1=np.loadtxt('checkpoints/PINN_20_4/loss.txt')
data2=np.loadtxt('checkpoints/PINN_35_4/loss.txt')
data3=np.loadtxt('checkpoints/PINN_50_4/loss.txt')
x=data1[:,0]
loss1=data1[:,1]
loss2=data2[:,1]
loss3=data3[:,1]
fig, axes = plt.subplots(figsize=(8, 8))
axes.semilogy(x, loss1, label='PINN with 4 layers 10 hidden dims')
axes.semilogy(x, loss2, label='PINN with 4 layers 50 hidden dims')
axes.semilogy(x, loss3, label='PINN with 4 layers 100 hidden dims')
axes.legend()
axes.set_title('Comparison of different hidden dimensions')
axes.set_xlabel('Epoch')
axes.set_ylabel('Loss')
```

A.9 隐藏层的不同神经元数目对结果的影响-ResPINN

```
%time
```

```
args = Options().parse()
args.problem = Problem()
# args.model = PINN(dim_in=2,
# dim_out=1,
# dim_hidden=args.dim_hidden,
# hidden_layers=args.hidden_layers,
# act_name='sin',
# dropout=args.dropout)
args.dim_hidden=20
args.model = ResPINN(2, 2, dim_hidden=args.dim_hidden, res_blocks=args.
hidden_layers)
# args.trainset = Trainset(args.n, args.n_bc, args.n_ic, method='lhs')
args.trainset = Trainset(200, 100, args.n_bc, args.n_ic, method='uniform
')
args.validset = Trainset(args.n_x, args.n_t, args.n_bc, args.n_ic,
method='uniform')
nums = []
losses = []
trainer = Trainer(args)
trainer.train()
loss_path = f'checkpoints/{args.model.__class__.__name__}_{args.
dim_hidden}_{args.hidden_layers}/loss.txt'
np.savetxt(loss_path, np.vstack((nums, losses)).T)
fig, axes = plt.subplots(figsize=(8, 8))
axes.semilogy(nums, losses, 'r--')
axes.set_title('Loss',fontsize=20)
axes.set_xlabel('Epoch',fontsize=20)
axes.set_ylabel('Loss',fontsize=20)
plt.savefig('loss.png')

hidden=35

args = Options().parse()
args.problem = Problem()
# args.model = PINN(dim_in=2,
# dim_out=1,
# dim_hidden=args.dim_hidden,
# hidden_layers=args.hidden_layers,
# act_name='sin',
# dropout=args.dropout)
args.dim_hidden=35
```

```
args.model = ResPINN(2, 2, dim_hidden=args.dim_hidden, res_blocks=args.hidden_layers)
# args.trainset = Trainset(args.n, args.n_bc, args.n_ic, method='lhs')
args.trainset = Trainset(200, 100, args.n_bc, args.n_ic, method='uniform')
args.validset = Trainset(args.n_x, args.n_t, args.n_bc, args.n_ic, method='uniform')
nums = []
losses = []
trainer = Trainer(args)
trainer.train()
loss_path = f'checkpoints/{args.model.__class__.__name__}_{args.dim_hidden}_{args.hidden_layers}/loss.txt'
np.savetxt(loss_path, np.vstack((nums, losses)).T)
fig, axes = plt.subplots(figsize=(8, 8))
axes.semilogy(nums, losses, 'r--')
axes.set_title('Loss',fontsize=20)
axes.set_xlabel('Epoch',fontsize=20)
axes.set_ylabel('Loss',fontsize=20)
plt.savefig('loss.png')

hidden=50

args = Options().parse()
args.problem = Problem()
# args.model = PINN(dim_in=2,
# dim_out=1,
# dim_hidden=args.dim_hidden,
# hidden_layers=args.hidden_layers,
# act_name='sin',
# dropout=args.dropout)
args.dim_hidden=50
args.model = ResPINN(2, 2, dim_hidden=args.dim_hidden, res_blocks=args.hidden_layers)
# args.trainset = Trainset(args.n, args.n_bc, args.n_ic, method='lhs')
args.trainset = Trainset(200, 100, args.n_bc, args.n_ic, method='uniform')
args.validset = Trainset(args.n_x, args.n_t, args.n_bc, args.n_ic, method='uniform')
nums = []
losses = []
```

```
trainer = Trainer(args)
trainer.train()
loss_path = f'checkpoints/{args.model.__class__.__name__}_{args.
dim_hidden}_{args.hidden_layers}/loss.txt'
np.savetxt(loss_path, np.vstack((nums, losses)).T)
fig, axes = plt.subplots(figsize=(8, 8))
axes.semilogy(nums, losses, 'r--')
axes.set_title('Loss',fontsize=20)
axes.set_xlabel('Epoch',fontsize=20)
axes.set_ylabel('Loss',fontsize=20)
plt.savefig('loss.png')

data1=np.loadtxt('checkpoints/ResPINN_20_4/loss.txt')
data2=np.loadtxt('checkpoints/ResPINN_35_4/loss.txt')
data3=np.loadtxt('checkpoints/ResPINN_50_4/loss.txt')
x=data1[:,0]
loss1=data1[:,1]
loss2=data2[:,1]
loss3=data3[:,1]
fig, axes = plt.subplots(figsize=(8, 8))
axes.semilogy(x, loss1, label='ResPINN with 4 layers 20 hidden dims')
axes.semilogy(x, loss2, label='ResPINN with 4 layers 35 hidden dims')
axes.semilogy(x, loss3, label='ResPINN with 4 layers 50 hidden dims')
axes.legend()
axes.set_title('Comparison of different hidden dimensions',fontsize=20)
axes.set_xlabel('Epoch',fontsize=20)
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

```
#!/usr/bin/env python
import numpy as np
import pandas as pd
import torch
import matplotlib.pyplot as plt
from torch.autograd import Variable
import torch.utils.data as Data
import random

# split a univariate sequence into samples
def split_sequence(sequence, n_steps):
    X, y = [], []
    for i in range(len(sequence)):
        # find the end of this pattern
        end_idx = i + n_steps
        # check if we are beyond the sequence
        if end_idx > len(sequence) - 1:
            break
        # gather input and output parts if the pattern
        seq_x, seq_y = sequence[i:end_idx], sequence[end_idx]
        X.append(seq_x)
        y.append(seq_y)
    return np.array(X), np.array(y)

#!/usr/bin/env python
import torch
import torch.nn as nn
from torch.nn import functional as F
from torch import optim

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt


data0 = pd.read_excel("D:/mathematical experiment/code/code/2011-2020
```

```
price.xlsx")
col_name = list(data0.columns)
data = data0[col_name[1:]]
index = data0[col_name[0]]

# plt.plot(data)
# plt.show()


dataset = data.dropna().values.astype('float32')
dataset = dataset.reshape(-1,1)

print(dataset.shape)

def centralize(data):
    min_value = np.min(data,axis=0)
    max_value = np.max(data,axis=0)
    data = (data - min_value) / (max_value-min_value)
    return data

def judge(dataset,k):
    '''
    to see at day k, if each stock rise or fall
    '''
    pr_today = dataset[k]
    pr_yesterday = dataset[k-1]
    pr_change = pr_today - pr_yesterday
    med = np.median(pr_change)
    re = np.zeros_like(pr_change)
    re[pr_change>0] = 1
    return(re)

def acc(out,y_real):
    out1 = np.zeros_like(out)
    out1[out>0] = 1
    out1[out<=0] = 0
    return 1-np.sum(np.sum(np.abs(y_real-out1)))/(np.prod(y_real.shape))


def create_dataset(dataset,look_back=240):
    dataX,dataY=[],[]
```

```
    for i in range(len(dataset)-look_back):
        pr_change = judge(dataset,i+look_back)
        a = dataset[i:(i+look_back)]
        dataX.append(a)
        dataY.append(pr_change)
    return np.array(dataX),np.array(dataY)

def set_seed(seed):
    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
    torch.cuda.manual_seed(seed)  # gpu 为当前GPU设置随机种子
    torch.backends.cudnn.deterministic = True  # cudnn
    np.random.seed(seed)  # numpy
    random.seed(seed)

look_back = 240
index_used = index[look_back:]
index_used = np.array(index_used)
X, Y = create_dataset(dataset,look_back)
print(X.shape, Y.shape)
a,b,c = X.shape

train_size = int(len(X) * 0.9)
valid_size = len(X) - train_size
index_size = int(len(index_used)*0.9)
print(train_size, valid_size)

X_train = X[:train_size]
Y_train = Y[:train_size]
index_train = index_used[:index_size]


X_valid = X[train_size:]
Y_valid = Y[train_size:]
index_valid = index_used[index_size:]


# X_train = X_train.reshape(-1,198,240)
# X_valid = X_valid.reshape(-1,198,240)
# Y_train = Y_train.reshape(-1,198,1)

```

```
X_train = X_train.reshape(train_size*c,b,1)
Y_train = Y_train.reshape(train_size*c,1)
X_valid = X_valid.reshape(valid_size*c,b,1)
Y_valid = Y_valid.reshape(valid_size*c,1)

# X_train = X_train.transpose(1, 0, 2)
# X_valid = X_valid.transpose(1, 0, 2)

X_train = torch.from_numpy(X_train)
Y_train = torch.from_numpy(Y_train)
X_valid = torch.from_numpy(X_valid)

print(X_train.shape,Y_train.shape)




index_train = index[:index_size]
index_valid = index[int(len(index)*0.9):]

index_valid.shape

X.shape

%%time
class LSTMRegression(nn.Module):
    def __init__(self, input_size, hidden_size, output_size=1,
    num_layers=1):
        super().__init__()
        self.lstm = nn.LSTM(input_size, hidden_size, num_layers,
        batch_first=True)
        self.linear = nn.Linear(hidden_size, output_size)

    def forward(self, x):
        _, (hn, cn) = self.lstm(x)
        hn = hn.squeeze()
        out = self.linear(hn)
        return out

model = LSTMRegression(input_size=1, hidden_size=5, output_size=1)

```

```python
criterion = torch.nn.BCEWithLogitsLoss()     #交叉熵BCEWithLogitsLoss()和MultiLabelSoftMarginLoss()
#criterion = torch.nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=1e-3)
#optimizer = optim.SGD(model.parameters(), lr=1e-1)

epochs = 100
batch_size = 30
batch = X_train.shape[0] // batch_size



torch_dataset = Data.TensorDataset(torch.tensor(X_train), torch.tensor(Y_train))
# 把 dataset 放入 DataLoader
loader = Data.DataLoader(
    dataset=torch_dataset,  # torch TensorDataset format
    batch_size=batch_size,  # mini batch size
    shuffle=True,  #
    num_workers=10,  # 多线程来读数据
)

loss_epoch = np.zeros(epochs)
acc_epoch = np.zeros(epochs)
loss_valid = np.zeros(epochs)
acc_valid = np.zeros(epochs)
for epoch in range(epochs):
    loss_ep = np.array([])
    acc_ep = np.array([])
    loss_epv = np.array([])
    acc_epv = np.array([])
    for step,(var_x,var_y) in enumerate(loader):
        out = model(var_x)
        out_f = out.detach().clone().numpy()
        var_yf = var_y.detach().clone().numpy()
        loss = criterion(out, var_y)
        loss_f = loss.detach().clone().numpy()
        acc_ep = np.append(acc_ep,acc(out_f,var_yf))
        loss_ep = np.append(loss_ep,loss_f)

        optimizer.zero_grad()
```

```
            loss.backward()
            optimizer.step()

        if (epoch + 1) % 5 == 0:
            print(f'Epoch: {epoch:5d}, Loss: {np.mean(loss_ep):.4e}, Acc:{np
            .mean(acc_ep):.4e}')


        loss_epoch[epoch] = np.mean(loss_ep)
        acc_epoch[epoch] = np.mean(acc_ep)

        Y_pre = model(X_valid)  #计算验证集表现
        Y_pre1 = Y_pre.clone().detach().numpy()
        Y_valid1 = torch.from_numpy(Y_valid)
        loss_valid[epoch] = criterion(Y_pre,Y_valid1)
        acc_valid[epoch] = acc(Y_pre1,Y_valid)



# test
#X = X.reshape(-1,198,240)
#X = torch.from_numpy(X_valid)
Y_pred = model(X_valid)
Y_pred = Y_pred.clone().detach().numpy()
pred_acc = acc(Y_pred,Y_valid)
#Y_pred = Y_pred.view(-1).data.numpy()

# visulize
kind = 2
series = np.arange(kind*len(index_valid),(kind+1)*len(index_valid))
Y_pred_re = Y_pred
Y_pred_re[Y_pred_re>0] = 1
Y_pred_re[Y_pred_re<=0] = 0


fig = plt.figure()
ax = plt.subplot()
type1 = ax.scatter(index_valid, Y_valid[series], alpha=0.5,color='b',
label='groundtruth')
type2 = ax.scatter(index_valid, Y_pred_re[series], alpha=0.3,color='r',
label='prediction')
```

```
plt.xlabel("date time")
plt.ylabel("0 for fall, 1 for rise")
ax.legend((type1, type2), (u'groundtruth', u'prediction'), loc='best')
plt.show()

plt.plot(acc_valid, 'r-', label='validation acc')
plt.plot(acc_epoch, 'b-', label='prediction acc')
plt.legend(loc='best')
plt.savefig("acc.png")
plt.show()

plt.plot(loss_valid, 'r-', label='validation loss')
plt.plot(loss_epoch, 'b-', label='prediction loss')
plt.legend(loc='best')
plt.savefig("loss.png")
plt.show()
```

双向 LSTM 网络

```
#!/usr/bin/env python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from torch.autograd import Variable
import torch.utils.data as Data
import datetime
import random

# split a univariate sequence into samples
def split_sequence(sequence, n_steps):
    X, y = [], []
    for i in range(len(sequence)):
        # find the end of this pattern
        end_idx = i + n_steps
        # check if we are beyond the sequence
        if end_idx > len(sequence) - 1:
            break
        # gather input and output parts if the pattern
        seq_x, seq_y = sequence[i:end_idx], sequence[end_idx]
        X.append(seq_x)
        y.append(seq_y)
    return np.array(X), np.array(y)
```

```
#!/usr/bin/env python
import torch
import torch.nn as nn
from torch.nn import functional as F
from torch import optim

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

data0 = pd.read_excel("D:/mathematical experiment/code/code/2011-2020
price.xlsx")
col_name = list(data0.columns)
data = data0[col_name[1:]]
index = data0[col_name[0]]

# plt.plot(data)
# plt.show()


dataset = data.dropna().values.astype('float32')

# max_value = np.max(dataset,axis=0)
# min_value = np.min(dataset,axis=0)
# dataset = (dataset - min_value) / (max_value-min_value)
print(dataset.shape)

data0 = pd.read_excel("D:/mathematical experiment/code/code/2011-2020
price.xlsx")
col_name = list(data0.columns)
data = data0[col_name[1:]]
index = data0[col_name[0]]

# plt.plot(data)
# plt.show()


dataset = data.dropna().values.astype('float32')

# max_value = np.max(dataset,axis=0)
```

```
# min_value = np.min(dataset,axis=0)
# dataset = (dataset - min_value) / (max_value-min_value)
print(dataset.shape)

def centralize(data):
    min_value = np.min(data,axis=0)
    max_value = np.max(data,axis=0)
    data = (data - min_value) / (max_value-min_value)
    return data

def judge(dataset,k):
    '''
    to see at day k, if each stock rise or fall
    '''
    pr_today = dataset[k]
    pr_yesterday = dataset[k-1]
    pr_change = pr_today - pr_yesterday
    med = np.median(pr_change)
    re = np.zeros_like(pr_change)
    re[pr_change>0] = 1
    return(re)

def acc(out,y_real):
    #y_real = y_real.detach().numpy()
    #out = out.detach().numpy()
    out1 = np.zeros_like(out)
    out1[out>0] = 1
    out1[out<=0] = 1
    return 1-np.sum(np.sum(np.sum(np.abs(y_real-out1))))/(np.prod(y_real
    .shape))


def create_dataset(dataset,look_back=240):
    dataX,dataY=[],[]
    for i in range(len(dataset)-look_back):
        pr_change = judge(dataset,i+look_back)
        a = dataset[i:(i+look_back)]
        dataX.append(a)
        dataY.append(pr_change)
    return centralize(np.array(dataX)),np.array(dataY)

```

```python
def set_seed(seed):
    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
    torch.cuda.manual_seed(seed)  # gpu 为当前GPU设置随机种子
    torch.backends.cudnn.deterministic = True  # cudnn
    np.random.seed(seed)  # numpy
    random.seed(seed)

look_back = 240
index = index[look_back:]
index = np.array(index)
X, Y = create_dataset(dataset,look_back)
print(X.shape, Y.shape)

train_size = int(len(X) * 0.7)
valid_size = len(X) - train_size
print(train_size, valid_size)

X_train = X[:train_size]
Y_train = Y[:train_size]
index_train = index[:train_size]

X_valid = X[train_size:]
Y_valid = Y[train_size:]
index_valid = index[train_size:]

X_train = X_train.reshape(-1,198,240)
X_valid = X_valid.reshape(-1,198,240)
Y_train = Y_train.reshape(-1,198,1)

# X_train = X_train.transpose(1, 0, 2)
# X_valid = X_valid.transpose(1, 0, 2)

X_train = torch.from_numpy(X_train)
Y_train = torch.from_numpy(Y_train)
X_valid = torch.from_numpy(X_valid)

print(X_train.shape,Y_train.shape)


%%time
```

```python
class LSTMRegression(nn.Module):
    def __init__(self, input_size, hidden_size, output_size=1, num_layers=2,bidirectional=True):
        super().__init__()
        self.lstm = nn.LSTM(input_size, hidden_size, num_layers, batch_first=True,bidirectional=True)
        self.linear = nn.Linear(2*hidden_size, output_size)

    def forward(self, x):
        x, _ = self.lstm(x) # (seq, batch, hidden)
        s, b, h = x.shape
#          print(s,b)
        x = x.contiguous().view(s*b, h) # 转换成线性层的输入格式
        x = self.linear(x)
        x = x.view(s, b, -1)
        return x



torch.manual_seed(7) #cpu
torch.cuda.manual_seed(7) #gpu

np.random.seed(7) #numpy
random.seed(7) # random and transforms
torch.backends.cudnn.deterministic=True #cudnn

model = LSTMRegression(input_size=240, hidden_size=4, output_size=1)

criterion = torch.nn.BCEWithLogitsLoss()     #交叉熵BCEWithLogitsLoss()和MultiLabelSoftMarginLoss()
#criterion = torch.nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=1e-3)
#optimizer = optim.SGD(model.parameters(), lr=1e-1)

epochs = 100
batch_size = 40
batch = X_train.shape[0] // batch_size

torch_dataset = Data.TensorDataset(torch.tensor(X_train), torch.tensor(Y_train))
# 把 dataset 放入 DataLoader
```

```
loader = Data.DataLoader(
    dataset=torch_dataset,  # torch TensorDataset format
    batch_size=batch_size,  # mini batch size
    shuffle=True,  #
    num_workers=2,  # 多线程来读数据
)

loss_epoch = np.zeros(epochs)
acc_epoch = np.zeros(epochs)
loss_valid = np.zeros(epochs)
acc_valid = np.zeros(epochs)
for epoch in range(epochs):
    acc_epo = 0
    loss_ep = np.array([])
    acc_ep = np.array([])
    loss_epv = np.array([])
    acc_epv = np.array([])
    for step,(var_x,var_y) in enumerate(loader):
        out = model(var_x)
        out_f = out.detach().clone().numpy()
        var_yf = var_y.detach().clone().numpy()
        loss = criterion(out, var_y)
        loss_f = loss.detach().clone().numpy()
        acc_ep = np.append(acc_ep,acc(out_f,var_yf))
        loss_ep = np.append(loss_ep,loss_f)

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

    if (epoch + 1) % 10 == 0:
        #print(f'Epoch: {epoch:5d}, Loss: {loss.item():.4e}, Acc:{
        acc_epo/(X_train.shape[0]*X_train.shape[1]):.4e}')
        print(f'Epoch: {epoch:5d}, Loss: {np.mean(loss_ep):.4e}, ACC: {
        np.mean(acc_ep):.5e}')

    loss_epoch[epoch] = np.mean(loss_ep)
    acc_epoch[epoch] = np.mean(acc_ep)

    Y_pre = model(X_valid)
    Y_pre1 = Y_pre.clone().detach().numpy()
```

```
    Y_valid1 = torch.from_numpy(Y_valid)
    a,b=Y_valid1.shape
    Y_valid2 = Y_valid1.reshape(a,b,1)
    Y_valid3 = Y_valid.reshape(a,b,1)
    loss_valid[epoch] = criterion(Y_pre,Y_valid2)
    acc_valid[epoch] = acc(Y_pre1,Y_valid3)

# test
X_valid = X_valid.reshape(-1,198,240)
#X_valid = torch.from_numpy(X_valid)
Y_pred = model(X_valid)

Y_pred = torch.squeeze(Y_pred,2)
Y_pred = Y_pred.clone().detach().numpy()
pred_acc = acc(Y_pred,Y_valid)

Y_pred_re = Y_pred[:,kind]
Y_pred_re[Y_pred_re>0] = 1
Y_pred_re[Y_pred_re<=0] = 0
k = len(Y_pred_re)
series = np.arange(1,k,k//100)

fig = plt.figure()
ax = plt.subplot()
type1 = ax.scatter(index_valid[series], Y_valid[series,kind], alpha=0.5,
color='b',label='groundtruth')
type2 = ax.scatter(index_valid[series], Y_pred_re[series], alpha=0.5,
color='r',label='prediction')
plt.xlabel("date time")
plt.ylabel("0 for fall, 1 for rise")
ax.legend((type1, type2), (u'groundtruth', u'prediction'), loc='best')
plt.show()

plt.plot(loss_epoch, 'r-', label='loss')
plt.plot(acc_epoch, 'b-', label='accurate rate')
plt.legend(loc='best')
plt.show()

plt.plot(acc_valid, 'r-', label='validation acc')
plt.plot(acc_epoch, 'b-', label='prediction acc')
plt.legend(loc='best')
```

```python
plt.savefig("dacc.png")
plt.show()

plt.plot(loss_valid, 'r-', label='validation loss')
plt.plot(loss_epoch, 'b-', label='prediction loss')
plt.legend(loc='best')
plt.savefig("dloss.png")
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

```python
model = LSTMRegression(input_size=1, hidden_size=5, output_size=1)

criterion = torch.nn.BCEWithLogitsLoss()     #交叉熵BCEWithLogitsLoss()和MultiLabelSoftMarginLoss()
#criterion = torch.nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=1e-3)
#optimizer = optim.SGD(model.parameters(), lr=1e-1)

epochs = 100
batch_size = 30
batch = X_train.shape[0] // batch_size



torch_dataset = Data.TensorDataset(torch.tensor(X_train), torch.tensor(Y_train))
# 把 dataset 放入 DataLoader
loader = Data.DataLoader(
    dataset=torch_dataset,  # torch TensorDataset format
    batch_size=batch_size,  # mini batch size
    shuffle=True,  #
    num_workers=10,  # 多线程来读数据
)

loss_epoch = np.zeros(epochs)
acc_epoch = np.zeros(epochs)
for epoch in range(epochs):
    loss_ep = np.array([])
    acc_ep = np.array([])
```

```
    for step,(var_x,var_y) in enumerate(loader):
        out = model(var_x)
        out_f = out.detach().clone().numpy()
        var_yf = var_y.detach().clone().numpy()
        loss = criterion(out, var_y)
        loss_f = loss.detach().clone().numpy()
        acc_ep = np.append(acc_ep,acc(out_f,var_yf))
        loss_ep = np.append(loss_ep,loss_f)

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

    if (epoch + 1) % 5 == 0:
        print(f'Epoch: {epoch:5d}, Loss: {np.mean(loss_ep):.4e}, Acc:{np
        .mean(acc_ep):.4e}')


    loss_epoch[epoch] = np.mean(loss_ep)
    acc_epoch[epoch] = np.mean(acc_ep)

# test
#X = X.reshape(-1,198,240)
#X = torch.from_numpy(X_valid)
Y_pred = model(X_valid)
Y_pred = Y_pred.clone().detach().numpy()
pred_acc = acc(Y_pred,Y_valid)
#Y_pred = Y_pred.view(-1).data.numpy()

# visulize
kind = 2
series = np.arange(kind*len(index_valid),(kind+1)*len(index_valid))
Y_pred_re = Y_pred
Y_pred_re[Y_pred_re>0] = 1
Y_pred_re[Y_pred_re<=0] = 0


fig = plt.figure()
ax = plt.subplot()
type1 = ax.scatter(index_valid, Y_valid[series], alpha=0.5,color='b',
label='groundtruth')
```

```
type2 = ax.scatter(index_valid, Y_pred_re[series], alpha=0.5,color='r',
    label='prediction')
plt.xlabel("date time")
plt.ylabel("0 for fall, 1 for rise")
ax.legend((type1, type2), (u'groundtruth', u'prediction'), loc='best')
plt.show()

plt.plot(loss_epoch, 'r-', label='loss')
plt.plot(acc_epoch, 'b-', label='accurate rate')
plt.legend(loc='best')
plt.show()

model = LSTMRegression(input_size=1, hidden_size=2, output_size=1)

criterion = torch.nn.BCEWithLogitsLoss()
optimizer = optim.Adam(model.parameters(), lr=1e-3)

epochs = 100
batch_size = 30
batch = X_train.shape[0] // batch_size



torch_dataset = Data.TensorDataset(torch.tensor(X_train), torch.tensor(
    Y_train))
# 把 dataset 放入 DataLoader
loader = Data.DataLoader(
    dataset=torch_dataset,  # torch TensorDataset format
    batch_size=batch_size,  # mini batch size
    shuffle=True,  #
    num_workers=10,  # 多线程来读数据
)

loss_epoch = np.zeros(epochs)
acc_epoch = np.zeros(epochs)
for epoch in range(epochs):
    loss_ep = np.array([])
    acc_ep = np.array([])
    for step,(var_x,var_y) in enumerate(loader):
        out = model(var_x)
        out_f = out.detach().clone().numpy()
```

```
        var_yf = var_y.detach().clone().numpy()
        loss = criterion(out, var_y)
        loss_f = loss.detach().clone().numpy()
        acc_ep = np.append(acc_ep,acc(out_f,var_yf))
        loss_ep = np.append(loss_ep,loss_f)

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

    if (epoch + 1) % 5 == 0:
        print(f'Epoch: {epoch:5d}, Loss: {np.mean(loss_ep):.4e}, Acc:{np
        .mean(acc_ep):.4e}')


    loss_epoch[epoch] = np.mean(loss_ep)
    acc_epoch[epoch] = np.mean(acc_ep)

Y_pred = model(X_valid)
Y_pred = Y_pred.clone().detach().numpy()
pred_acc = acc(Y_pred,Y_valid)

kind = 2
series = np.arange(kind*len(index_valid),(kind+1)*len(index_valid))
Y_pred_re = Y_pred
Y_pred_re[Y_pred_re>0] = 1
Y_pred_re[Y_pred_re<=0] = 0


fig = plt.figure()
ax = plt.subplot()
type1 = ax.scatter(index_valid, Y_valid[series], alpha=0.5,color='b',
label='groundtruth')
type2 = ax.scatter(index_valid, Y_pred_re[series], alpha=0.5,color='r',
label='prediction')
plt.xlabel("date time")
plt.ylabel("0 for fall, 1 for rise")
ax.legend((type1, type2), (u'groundtruth', u'prediction'), loc='best')
plt.show()

plt.plot(loss_epoch, 'r-', label='loss')
```

```python
plt.plot(acc_epoch, 'b-', label='accurate rate')
plt.legend(loc='best')
plt.show()

model = LSTMRegression(input_size=1, hidden_size=3, output_size=1)

criterion = torch.nn.BCEWithLogitsLoss()
optimizer = optim.Adam(model.parameters(), lr=1e-3)

epochs = 100
batch_size = 30
batch = X_train.shape[0] // batch_size



torch_dataset = Data.TensorDataset(torch.tensor(X_train), torch.tensor(
    Y_train))
# 把 dataset 放入 DataLoader
loader = Data.DataLoader(
    dataset=torch_dataset,  # torch TensorDataset format
    batch_size=batch_size,  # mini batch size
    shuffle=True,  #
    num_workers=10,  # 多线程来读数据
)

loss_epoch = np.zeros(epochs)
acc_epoch = np.zeros(epochs)
for epoch in range(epochs):
    loss_ep = np.array([])
    acc_ep = np.array([])
    for step,(var_x,var_y) in enumerate(loader):
        out = model(var_x)
        out_f = out.detach().clone().numpy()
        var_yf = var_y.detach().clone().numpy()
        loss = criterion(out, var_y)
        loss_f = loss.detach().clone().numpy()
        acc_ep = np.append(acc_ep,acc(out_f,var_yf))
        loss_ep = np.append(loss_ep,loss_f)

        optimizer.zero_grad()
        loss.backward()
```

```
            optimizer.step()

        if (epoch + 1) % 5 == 0:
            print(f'Epoch: {epoch:5d}, Loss: {np.mean(loss_ep):.4e}, Acc:{np
            .mean(acc_ep):.4e}')


        loss_epoch[epoch] = np.mean(loss_ep)
        acc_epoch[epoch] = np.mean(acc_ep)

Y_pred = model(X_valid)
Y_pred = Y_pred.clone().detach().numpy()
pred_acc = acc(Y_pred,Y_valid)

kind = 2
series = np.arange(kind*len(index_valid),(kind+1)*len(index_valid))
Y_pred_re = Y_pred
Y_pred_re[Y_pred_re>0] = 1
Y_pred_re[Y_pred_re<=0] = 0


fig = plt.figure()
ax = plt.subplot()
type1 = ax.scatter(index_valid, Y_valid[series], alpha=0.5,color='b',
label='groundtruth')
type2 = ax.scatter(index_valid, Y_pred_re[series], alpha=0.5,color='r',
label='prediction')
plt.xlabel("date time")
plt.ylabel("0 for fall, 1 for rise")
ax.legend((type1, type2), (u'groundtruth', u'prediction'), loc='best')
plt.show()

plt.plot(loss_epoch, 'r-', label='loss')
plt.plot(acc_epoch, 'b-', label='accurate rate')
plt.legend(loc='best')
plt.show()
```

双向 LSTM 网络

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

```
    num_layers=2,bidirectional=True):
        super().__init__()
        self.lstm = nn.LSTM(input_size, hidden_size, num_layers, batch_first=True,bidirectional=True)
        self.linear = nn.Linear(2*hidden_size, output_size)

    def forward(self, x):
        x, _ = self.lstm(x) # (seq, batch, hidden)
        s, b, h = x.shape
#          print(s,b)
        x = x.contiguous().view(s*b, h) # 转换成线性层的输入格式
        x = self.linear(x)
        x = x.view(s, b, -1)
        return x



torch.manual_seed(7) #cpu
torch.cuda.manual_seed(7) #gpu

np.random.seed(7) #numpy
random.seed(7) # random and transforms
torch.backends.cudnn.deterministic=True #cudnn

model = LSTMRegression(input_size=240, hidden_size=4, output_size=1)

criterion = torch.nn.BCEWithLogitsLoss()     #交叉熵BCEWithLogitsLoss()和MultiLabelSoftMarginLoss()
#criterion = torch.nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=1e-3)
#optimizer = optim.SGD(model.parameters(), lr=1e-1)

epochs = 100
batch_size = 40
batch = X_train.shape[0] // batch_size

torch_dataset = Data.TensorDataset(torch.tensor(X_train), torch.tensor(Y_train))
# 把 dataset 放入 DataLoader
loader = Data.DataLoader(
    dataset=torch_dataset,  # torch TensorDataset format
```

```
    batch_size=batch_size,  # mini batch size
    shuffle=True,  #
    num_workers=2,  # 多线程来读数据
)

loss_epoch = np.zeros(epochs)
acc_epoch = np.zeros(epochs)
for epoch in range(epochs):
    acc_epo = 0
    loss_ep = np.array([])
    acc_ep = np.array([])
    for step,(var_x,var_y) in enumerate(loader):
        out = model(var_x)
        out_f = out.detach().clone().numpy()
        var_yf = var_y.detach().clone().numpy()
        loss = criterion(out, var_y)
        loss_f = loss.detach().clone().numpy()
        acc_ep = np.append(acc_ep,acc(out_f,var_yf))
        loss_ep = np.append(loss_ep,loss_f)

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

    if (epoch + 1) % 10 == 0:
        #print(f'Epoch: {epoch:5d}, Loss: {loss.item():.4e}, Acc:{
        acc_epo/(X_train.shape[0]*X_train.shape[1]):.4e}')
        print(f'Epoch: {epoch:5d}, Loss: {np.mean(loss_ep):.4e}, ACC: {
        np.mean(acc_ep):.5e}')

    loss_epoch[epoch] = np.mean(loss_ep)
    acc_epoch[epoch] = np.mean(acc_ep)

# test
X_valid = X_valid.reshape(-1,198,240)
#X_valid = torch.from_numpy(X_valid)
Y_pred = model(X_valid)

kind = 3
Y_pred = torch.squeeze(Y_pred,2)
Y_pred = Y_pred.clone().detach().numpy()
```

```
pred_acc = acc(Y_pred,Y_valid)

Y_pred_re = Y_pred[:,kind]
Y_pred_re[Y_pred_re>0] = 1
Y_pred_re[Y_pred_re<=0] = 0
k = len(Y_pred_re)
series = np.arange(1,k,k//100)

fig = plt.figure()
ax = plt.subplot()
type1 = ax.scatter(index_valid[series], Y_valid[series,kind], alpha=0.5,
  color='b',label='groundtruth')
type2 = ax.scatter(index_valid[series], Y_pred_re[series], alpha=0.3,
  color='r',label='prediction')
plt.xlabel("date time")
plt.ylabel("0 for fall, 1 for rise")
ax.legend((type1, type2), (u'groundtruth', u'prediction'), loc='best')
plt.show()

plt.plot(loss_epoch, 'r-', label='loss')
plt.plot(acc_epoch, 'b-', label='accurate rate')
plt.legend(loc='best')
plt.show()
```

B.3 不同的 batch size 对结果的影响的代码

```
#!/usr/bin/env python
import numpy as np
import pandas as pd
import torch
import matplotlib.pyplot as plt
from torch.autograd import Variable
import torch.utils.data as Data
import random
# split a univariate sequence into samples
def split_sequence(sequence, n_steps):
    X, y = [], []
    for i in range(len(sequence)):
        # find the end of this pattern
        end_idx = i + n_steps
        # check if we are beyond the sequence
```

```
        if end_idx > len(sequence) - 1:
            break
        # gather input and output parts if the pattern
        seq_x, seq_y = sequence[i:end_idx], sequence[end_idx]
        X.append(seq_x)
        y.append(seq_y)
    return np.array(X), np.array(y)
#!/usr/bin/env python
import torch
import torch.nn as nn
from torch.nn import functional as F
from torch import optim

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
data0 = pd.read_excel("2011-2020price.xlsx")
col_name = list(data0.columns)
data = data0[col_name[1:]]
index = data0[col_name[0]]

# plt.plot(data)
# plt.show()


dataset = data.dropna().values.astype('float32')
dataset = dataset.reshape(-1,1)

print(dataset.shape)


def centralize(data):
    min_value = np.min(data, axis=0)
    max_value = np.max(data, axis=0)
    data = (data - min_value) / (max_value - min_value)
    return data


def judge(dataset, k):
    '''
    to see at day k, if each stock rise or fall
```

```
    '''
    pr_today = dataset[k]
    pr_yesterday = dataset[k - 1]
    pr_change = pr_today - pr_yesterday
    med = np.median(pr_change)
    re = np.zeros_like(pr_change)
    re[pr_change > 0] = 1
    return (re)


def acc(out, y_real):
    out1 = np.zeros_like(out)
    out1[out > 0] = 1
    out1[out <= 0] = 0
    return 1 - sum(sum(abs(y_real - out1))) / (np.prod(y_real.shape))


def create_dataset(dataset, look_back=240):
    dataX, dataY = [], []
    for i in range(len(dataset) - look_back):
        pr_change = judge(dataset, i + look_back)
        a = dataset[i:(i + look_back)]
        dataX.append(a)
        dataY.append(pr_change)
    return np.array(dataX), np.array(dataY)


def set_seed(seed):
    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
    torch.cuda.manual_seed(seed)  # gpu 为当前GPU设置随机种子
    torch.backends.cudnn.deterministic = True  # cudnn
    np.random.seed(seed)  # numpy
    random.seed(seed)


look_back = 240
index_used = index[look_back:]
index_used = np.array(index_used)
X, Y = create_dataset(dataset, look_back)
print(X.shape, Y.shape)
```

```
a, b, c = X.shape

train_size = int(len(X) * 0.9)
valid_size = len(X) - train_size
index_size = int(len(index_used) * 0.9)
print(train_size, valid_size)

X_train = X[:train_size]
Y_train = Y[:train_size]
index_train = index_used[:index_size]

X_valid = X[train_size:]
Y_valid = Y[train_size:]
index_valid = index_used[index_size:]

# X_train = X_train.reshape(-1,198,240)
# X_valid = X_valid.reshape(-1,198,240)
# Y_train = Y_train.reshape(-1,198,1)

X_train = X_train.reshape(train_size * c, b, 1)
Y_train = Y_train.reshape(train_size * c, 1)
X_valid = X_valid.reshape(valid_size * c, b, 1)
Y_valid = Y_valid.reshape(valid_size * c, 1)

# X_train = X_train.transpose(1, 0, 2)
# X_valid = X_valid.transpose(1, 0, 2)

X_train = torch.from_numpy(X_train)
Y_train = torch.from_numpy(Y_train)
X_valid = torch.from_numpy(X_valid)

print(X_train.shape, Y_train.shape)


index_train = index[:index_size]
index_valid = index[int(len(index)*0.9):]

index_valid.shape
X.shape


```

```


class LSTMRegression(nn.Module):
    def __init__(self, input_size, hidden_size, output_size=1,
    num_layers=1):
        super().__init__()
        self.lstm = nn.LSTM(input_size, hidden_size, num_layers,
        batch_first=True)
        self.linear = nn.Linear(hidden_size, output_size)

    def forward(self, x):
        _, (hn, cn) = self.lstm(x)
        hn = hn.squeeze()
        out = self.linear(hn)
        return out


model = LSTMRegression(input_size=1, hidden_size=5, output_size=1)

criterion = torch.nn.BCEWithLogitsLoss()  # 交叉熵BCEWithLogitsLoss()和
MultiLabelSoftMarginLoss()
# criterion = torch.nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=1e-3)
# optimizer = optim.SGD(model.parameters(), lr=1e-1)

epochs =20
batch_size = 60
batch = X_train.shape[0] // batch_size

torch_dataset = Data.TensorDataset(torch.tensor(X_train), torch.tensor(
Y_train))
# 把 dataset 放入 DataLoader
loader = Data.DataLoader(
    dataset=torch_dataset,  # torch TensorDataset format
    batch_size=batch_size,  # mini batch size
    shuffle=True,  #
    num_workers=10,  # 多线程来读数据
)

loss_epoch = np.zeros(epochs)
acc_epoch = np.zeros(epochs)
```

```python
for epoch in range(epochs):
    loss_ep = np.array([])
    acc_ep = np.array([])
    for step, (var_x, var_y) in enumerate(loader):
        out = model(var_x)
        out_f = out.detach().clone().numpy()
        var_yf = var_y.detach().clone().numpy()
        loss = criterion(out, var_y)
        loss_f = loss.detach().clone().numpy()
        acc_ep = np.append(acc_ep, acc(out_f, var_yf))
        loss_ep = np.append(loss_ep, loss_f)

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

    if (epoch + 1) % 2 == 0:
        print(f'Epoch: {epoch:5d}, Loss: {np.mean(loss_ep):.4e}, Acc:{np
        .mean(acc_ep):.4e}')

    loss_epoch[epoch] = np.mean(loss_ep)
    acc_epoch[epoch] = np.mean(acc_ep)

# test
#X = X.reshape(-1,198,240)
#X = torch.from_numpy(X_valid)
Y_pred = model(X_valid)
Y_pred = Y_pred.clone().detach().numpy()
pred_acc = acc(Y_pred,Y_valid)
print(f'pred_acc{pred_acc}')
#Y_pred = Y_pred.view(-1).data.numpy()
# visulize
kind = 2
series = np.arange(kind*len(index_valid),(kind+1)*len(index_valid))
Y_pred_re = Y_pred
Y_pred_re[Y_pred_re>0] = 1
Y_pred_re[Y_pred_re<=0] = 0


fig = plt.figure()
ax = plt.subplot()
```

```
type1 = ax.scatter(index_valid, Y_valid[series], alpha=0.5,color='b',
label='groundtruth')
type2 = ax.scatter(index_valid, Y_pred_re[series], alpha=0.5,color='r',
label='prediction')
plt.xlabel("date time")
plt.ylabel("0 for fall, 1 for rise")
ax.legend((type1, type2), (u'groundtruth', u'prediction'), loc='best')
plt.show()

plt.plot(loss_epoch, 'r-', label='loss')
plt.plot(acc_epoch, 'b-', label='accurate rate')
plt.legend(loc='best')
plt.show()
```

B.4 不同的学习率对结果的影响的代码

```
#!/usr/bin/env python
import numpy as np
import pandas as pd
import torch
import matplotlib.pyplot as plt
from torch.autograd import Variable
import torch.utils.data as Data
import random

# split a univariate sequence into samples
def split_sequence(sequence, n_steps):
    X, y = [], []
    for i in range(len(sequence)):
        # find the end of this pattern
        end_idx = i + n_steps
        # check if we are beyond the sequence
        if end_idx > len(sequence) - 1:
            break
        # gather input and output parts if the pattern
        seq_x, seq_y = sequence[i:end_idx], sequence[end_idx]
        X.append(seq_x)
        y.append(seq_y)
    return np.array(X), np.array(y)

#!/usr/bin/env python
```

```
import torch
import torch.nn as nn
from torch.nn import functional as F
from torch import optim

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

data0 = pd.read_excel("D:/mathematical experiment/code/code/2011-2020
price.xlsx")
col_name = list(data0.columns)
data = data0[col_name[1:]]
index = data0[col_name[0]]

# plt.plot(data)
# plt.show()


dataset = data.dropna().values.astype('float32')
dataset = dataset.reshape(-1,1)

print(dataset.shape)

def centralize(data):
    min_value = np.min(data,axis=0)
    max_value = np.max(data,axis=0)
    data = (data - min_value) / (max_value-min_value)
    return data

def judge(dataset,k):
    '''
    to see at day k, if each stock rise or fall
    '''
    pr_today = dataset[k]
    pr_yesterday = dataset[k-1]
    pr_change = pr_today - pr_yesterday
    med = np.median(pr_change)
    re = np.zeros_like(pr_change)
    re[pr_change>0] = 1
    return(re)
```

```python

def acc(out,y_real):
    out1 = np.zeros_like(out)
    out1[out>0] = 1
    out1[out<=0] = 0
    return 1-sum(sum(abs(y_real-out1)))/(np.prod(y_real.shape))


def create_dataset(dataset,look_back=240):
    dataX,dataY=[],[]
    for i in range(len(dataset)-look_back):
        pr_change = judge(dataset,i+look_back)
        a = dataset[i:(i+look_back)]
        dataX.append(a)
        dataY.append(pr_change)
    return np.array(dataX),np.array(dataY)

def set_seed(seed):
    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
    torch.cuda.manual_seed(seed)  # gpu 为当前GPU设置随机种子
    torch.backends.cudnn.deterministic = True  # cudnn
    np.random.seed(seed)  # numpy
    random.seed(seed)

look_back = 240
index_used = index[look_back:]
index_used = np.array(index_used)
X, Y = create_dataset(dataset,look_back)
print(X.shape, Y.shape)
a,b,c = X.shape

train_size = int(len(X) * 0.9)
valid_size = len(X) - train_size
index_size = int(len(index_used)*0.9)
print(train_size, valid_size)

X_train = X[:train_size]
Y_train = Y[:train_size]
index_train = index_used[:index_size]

```

```
X_valid = X[train_size:]
Y_valid = Y[train_size:]
index_valid = index_used[index_size:]


# X_train = X_train.reshape(-1,198,240)
# X_valid = X_valid.reshape(-1,198,240)
# Y_train = Y_train.reshape(-1,198,1)

X_train = X_train.reshape(train_size*c,b,1)
Y_train = Y_train.reshape(train_size*c,1)
X_valid = X_valid.reshape(valid_size*c,b,1)
Y_valid = Y_valid.reshape(valid_size*c,1)

# X_train = X_train.transpose(1, 0, 2)
# X_valid = X_valid.transpose(1, 0, 2)

X_train = torch.from_numpy(X_train)
Y_train = torch.from_numpy(Y_train)
X_valid = torch.from_numpy(X_valid)

print(X_train.shape,Y_train.shape)


%%time
class LSTMRegression(nn.Module):
    def __init__(self, input_size, hidden_size, output_size=1,
    num_layers=1):
        super().__init__()
        self.lstm = nn.LSTM(input_size, hidden_size, num_layers,
        batch_first=True)
        self.linear = nn.Linear(hidden_size, output_size)

    def forward(self, x):
        _, (hn, cn) = self.lstm(x)
        hn = hn.squeeze()
        out = self.linear(hn)
        return out

lr = [0.01,0.001,0.0001]
```

```python

for i in range(3):

    model = LSTMRegression(input_size=1, hidden_size=5, output_size=1)

    criterion = torch.nn.BCEWithLogitsLoss()     #交叉熵
    BCEWithLogitsLoss()和MultiLabelSoftMarginLoss()
    #criterion = torch.nn.CrossEntropyLoss()
    optimizer = optim.Adam(model.parameters(), lr[i])
    #optimizer = optim.SGD(model.parameters(), lr=1e-1)

    epochs = 30
    batch_size = 30
    batch = X_train.shape[0] // batch_size



    torch_dataset = Data.TensorDataset(torch.tensor(X_train), torch.
    tensor(Y_train))
    # 把 dataset 放入 DataLoader
    loader = Data.DataLoader(
        dataset=torch_dataset,  # torch TensorDataset format
        batch_size=batch_size,  # mini batch size
        shuffle=True,  #
        num_workers=0,
    )

    loss_epoch = np.zeros(epochs)
    acc_epoch = np.zeros(epochs)
    for epoch in range(epochs):
        loss_ep = np.array([])
        acc_ep = np.array([])
        for step,(var_x,var_y) in enumerate(loader):
            out = model(var_x)
            out_f = out.detach().clone().numpy()
            var_yf = var_y.detach().clone().numpy()
            loss = criterion(out, var_y)
            loss_f = loss.detach().clone().numpy()
            acc_ep = np.append(acc_ep,acc(out_f,var_yf))
            loss_ep = np.append(loss_ep,loss_f)

```

```
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()

        if (epoch + 1) % 5 == 0:
            print(f'Epoch: {epoch:5d}, Loss: {np.mean(loss_ep):.4e}, Acc:{np.mean(acc_ep):.4e}')


        loss_epoch[epoch] = np.mean(loss_ep)
        acc_epoch[epoch] = np.mean(acc_ep)



    # test
    #X = X.reshape(-1,198,240)
    #X = torch.from_numpy(X_valid)
    Y_pred = model(X_valid)
    Y_pred = Y_pred.clone().detach().numpy()
    pred_acc = acc(Y_pred,Y_valid)
    print(pred_acc)
    #Y_pred = Y_pred.view(-1).data.numpy()

    # visulize
    kind = 2
    series = np.arange(kind*len(index_valid),(kind+1)*len(index_valid))
    Y_pred_re = Y_pred
    Y_pred_re[Y_pred_re>0] = 1
    Y_pred_re[Y_pred_re<=0] = 0

    filename1 = 'fall and rise' + str(i) + '.png'
    filename2 = 'loss and accuracy' + str(i) + '.png'

    fig = plt.figure()
    ax = plt.subplot()
    type1 = ax.scatter(index_valid, Y_valid[series], alpha=0.5,color='b',label='groundtruth')
    type2 = ax.scatter(index_valid, Y_pred_re[series], alpha=0.3,color='r',label='prediction')
    plt.xlabel("date time")
    plt.ylabel("0 for fall, 1 for rise")
```

```
    ax.legend((type1, type2), (u'groundtruth', u'prediction'), loc='best
')
    plt.show()
    plt.savefig(filename1)

    plt.plot(loss_epoch, 'r-', label='loss')
    plt.plot(acc_epoch, 'b-', label='accurate rate')
    plt.legend(loc='best')
    plt.show()
    plt.savefig(filename2)
```

B.5 不同的优化器对结果的影响的代码

```
#!/usr/bin/env python
import numpy as np
import pandas as pd
import torch
import matplotlib.pyplot as plt
from torch.autograd import Variable
import torch.utils.data as Data
import random

# split a univariate sequence into samples
def split_sequence(sequence, n_steps):
    X, y = [], []
    for i in range(len(sequence)):
        # find the end of this pattern
        end_idx = i + n_steps
        # check if we are beyond the sequence
        if end_idx > len(sequence) - 1:
            break
        # gather input and output parts if the pattern
        seq_x, seq_y = sequence[i:end_idx], sequence[end_idx]
        X.append(seq_x)
        y.append(seq_y)
    return np.array(X), np.array(y)

#!/usr/bin/env python
import torch
import torch.nn as nn
from torch.nn import functional as F
```

```
from torch import optim

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt


data0 = pd.read_excel("D:/mathematical experiment/code/code/2011-2020
price.xlsx")
col_name = list(data0.columns)
data = data0[col_name[1:10]]
index = data0[col_name[0]]

# plt.plot(data)
# plt.show()


dataset = data.dropna().values.astype('float32')
dataset = dataset.reshape(-1,1)

print(dataset.shape)

def centralize(data):
    min_value = np.min(data,axis=0)
    max_value = np.max(data,axis=0)
    data = (data - min_value) / (max_value-min_value)
    return data

def judge(dataset,k):
    '''
    to see at day k, if each stock rise or fall
    '''
    pr_today = dataset[k]
    pr_yesterday = dataset[k-1]
    pr_change = pr_today - pr_yesterday
    med = np.median(pr_change)
    re = np.zeros_like(pr_change)
    re[pr_change>0] = 1
    return(re)

def acc(out,y_real):
```

```
    out1 = np.zeros_like(out)
    out1[out>0] = 1
    out1[out<=0] = 0
    return 1-sum(sum(abs(y_real-out1)))/(np.prod(y_real.shape))


def create_dataset(dataset,look_back=240):
    dataX,dataY=[],[]
    for i in range(len(dataset)-look_back):
        pr_change = judge(dataset,i+look_back)
        a = dataset[i:(i+look_back)]
        dataX.append(a)
        dataY.append(pr_change)
    return np.array(dataX),np.array(dataY)

def set_seed(seed):
    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
    torch.cuda.manual_seed(seed)  # gpu 为当前GPU设置随机种子
    torch.backends.cudnn.deterministic = True  # cudnn
    np.random.seed(seed)  # numpy
    random.seed(seed)

look_back = 240
index_used = index[look_back:]
index_used = np.array(index_used)
X, Y = create_dataset(dataset,look_back)
print(X.shape, Y.shape)
a,b,c = X.shape

train_size = int(len(X) * 0.9)
valid_size = len(X) - train_size
index_size = int(len(index_used)*0.9)
print(train_size, valid_size)

X_train = X[:train_size]
Y_train = Y[:train_size]
index_train = index_used[:index_size]


X_valid = X[train_size:]
```

```
Y_valid = Y[train_size:]
index_valid = index_used[index_size:]


# X_train = X_train.reshape(-1,198,240)
# X_valid = X_valid.reshape(-1,198,240)
# Y_train = Y_train.reshape(-1,198,1)

X_train = X_train.reshape(train_size*c,b,1)
Y_train = Y_train.reshape(train_size*c,1)
X_valid = X_valid.reshape(valid_size*c,b,1)
Y_valid = Y_valid.reshape(valid_size*c,1)

# X_train = X_train.transpose(1, 0, 2)
# X_valid = X_valid.transpose(1, 0, 2)

X_train = torch.from_numpy(X_train)
Y_train = torch.from_numpy(Y_train)
X_valid = torch.from_numpy(X_valid)

print(X_train.shape,Y_train.shape)





%%time
class LSTMRegression(nn.Module):
    def __init__(self, input_size, hidden_size, output_size=1,
    num_layers=1):
        super().__init__()
        self.lstm = nn.LSTM(input_size, hidden_size, num_layers,
        batch_first=True)
        self.linear = nn.Linear(hidden_size, output_size)

    def forward(self, x):
        _, (hn, cn) = self.lstm(x)
        hn = hn.squeeze()
        out = self.linear(hn)
        return out

```

```
model = LSTMRegression(input_size=1, hidden_size=5, output_size=1)

for index in range(2):
    print(index)

    criterion = torch.nn.BCEWithLogitsLoss()     #交叉熵
    BCEWithLogitsLoss()和MultiLabelSoftMarginLoss()
    if index==1:
        optimizer = optim.Adam(model.parameters(), lr=1e-3)
    else:
        optimizer = optim.SGD(model.parameters(), lr=1e-3)

#     else:
#         print(index+2)
#         criterion = torch.nn.CrossEntropyLoss()
#         if index==2:
#             optimizer = optim.Adam(model.parameters(), lr=1e-3)
#         else:
#             optimizer = optim.SGD(model.parameters(), lr=1e-3)


    #optimizer = optim.Adam(model.parameters(), lr=1e-3)
    #optimizer = optim.SGD(model.parameters(), lr=1e-1)

    epochs = 40
    batch_size = 30
    batch = X_train.shape[0] // batch_size



    torch_dataset = Data.TensorDataset(torch.tensor(X_train), torch.
    tensor(Y_train))
    # 把 dataset 放入 DataLoader
    loader = Data.DataLoader(
        dataset=torch_dataset,  # torch TensorDataset format
        batch_size=batch_size,  # mini batch size
        shuffle=True,  #
        num_workers=10,  # 多线程来读数据
    )

    loss_epoch = np.zeros(epochs)
```

```
    acc_epoch = np.zeros(epochs)
    for epoch in range(epochs):
        loss_ep = np.array([])
        acc_ep = np.array([])
        for step,(var_x,var_y) in enumerate(loader):
            out = model(var_x)
            out_f = out.detach().clone().numpy()
            var_yf = var_y.detach().clone().numpy()
            loss = criterion(out, var_y)
            loss_f = loss.detach().clone().numpy()
            acc_ep = np.append(acc_ep,acc(out_f,var_yf))
            loss_ep = np.append(loss_ep,loss_f)

            optimizer.zero_grad()
            loss.backward()
            optimizer.step()

        if (epoch + 1) % 10 == 0:
            print(f'Epoch: {epoch:5d}, Loss: {np.mean(loss_ep):.4e}, Acc
            :{np.mean(acc_ep):.4e}')


        loss_epoch[epoch] = np.mean(loss_ep)
        acc_epoch[epoch] = np.mean(acc_ep)



    # test
    #X = X.reshape(-1,198,240)
    #X = torch.from_numpy(X_valid)
    Y_pred = model(X_valid)
    Y_pred = Y_pred.clone().detach().numpy()
    pred_acc = acc(Y_pred,Y_valid)
    print(f'acc_rate:{pred_acc}')
    #Y_pred = Y_pred.view(-1).data.numpy()

    # visulize
    kind = 2
    series = np.arange(kind*len(index_valid),(kind+1)*len(index_valid))
    Y_pred_re = Y_pred
    Y_pred_re[Y_pred_re>0] = 1
```

```
    Y_pred_re[Y_pred_re<=0] = 0


    filename1 = 'point' + str(index) + '.png'
    filename2 = 'pic' + str(index) + '.png'

    fig = plt.figure()
    ax = plt.subplot()
    type1 = ax.scatter(index_valid, Y_valid[series], alpha=0.6,color='b'
    ,label='groundtruth')
    type2 = ax.scatter(index_valid, Y_pred_re[series], alpha=0.3,color='
    r',label='prediction')
    plt.xlabel("date time")
    plt.ylabel("0 for fall, 1 for rise")
    ax.legend((type1, type2), (u'groundtruth', u'prediction'), loc='best
    ')
    plt.savefig(filename1)
    plt.show()

    plt.plot(loss_epoch, 'r-', label='loss')
    plt.plot(acc_epoch, 'b-', label='accurate rate')
    plt.legend(loc='best')
    plt.savefig(filename2)
    plt.show()
```