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Schrodinger方程

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

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

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

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

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为双曲正弦函数，即

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

(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。

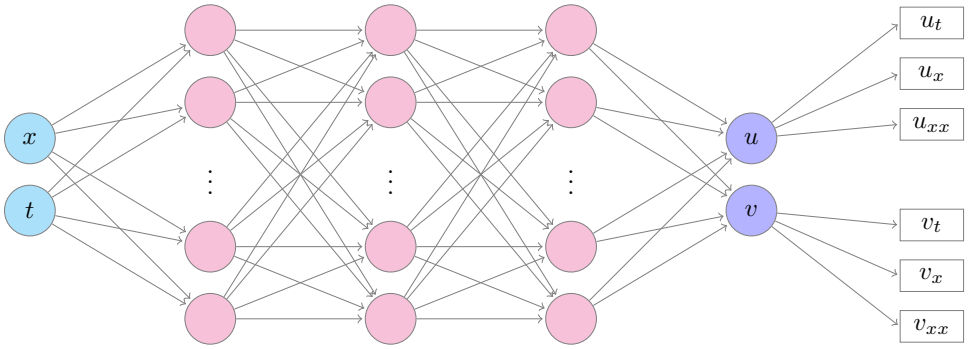

图 1: 求解薛定谔方程的神经网络架构

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

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

1.2 网络结构搭建

1.2.1 导入所需要的模块

(代码在最后) 深度学习框架torch，优化计算.torch.optim，函数计算torch.nn，调参操作torch.optim.lr_scheduler，处理文件和目录os，处理多维数组numpy，文件文件夹读写操作shutil，绘图操作matplotlib.pyplot。

1.2.2 构建函数

其中包括:1.规定定义域，创建方程内函数，以及函数真值。2.调用函数示例

1.2.3 构建数据集

包括训练集的生成和测试集的生成

1.2.4 激活函数

可选择tanh, ReLU, LeakyReLU, sigmoid, softplus

1.2.5 计算网络输入，输出的梯度

compute the derivative of outputs associated with inputs.

1.2.6 搭建神经网络Network Archetecture

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

1.2.7 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

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

1.2.8 训练过程及检验过程

接下来显示各种结果

1.3 不同网络训练结果之PiNN

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

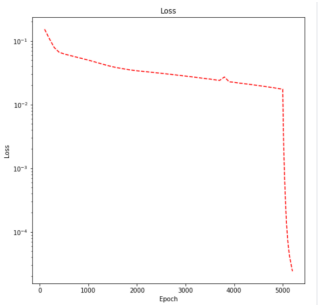

图 2: Loss随Epoch增长之间的关系

测试用采样点: 使用Uniform Sampling方法采样256个点生成的数据由给定的参数 t 决定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:

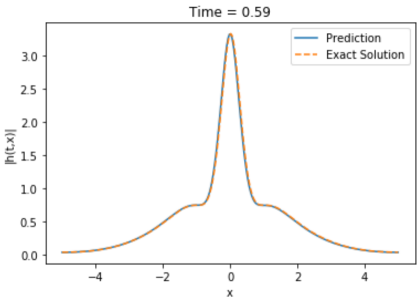

图 3: t=0.59

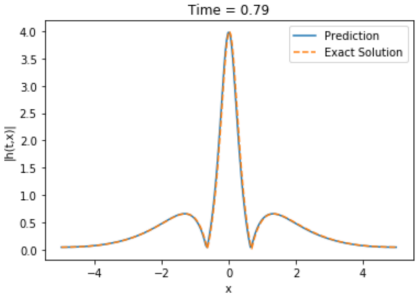

图 4: t=0.79

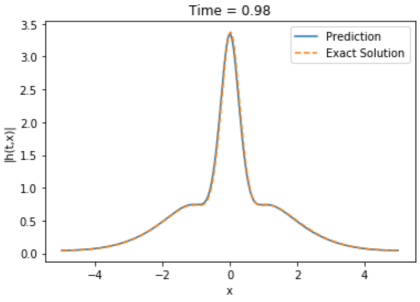

图 5: t=0.98

1.4 不同网络训练结果之ResPiNN

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

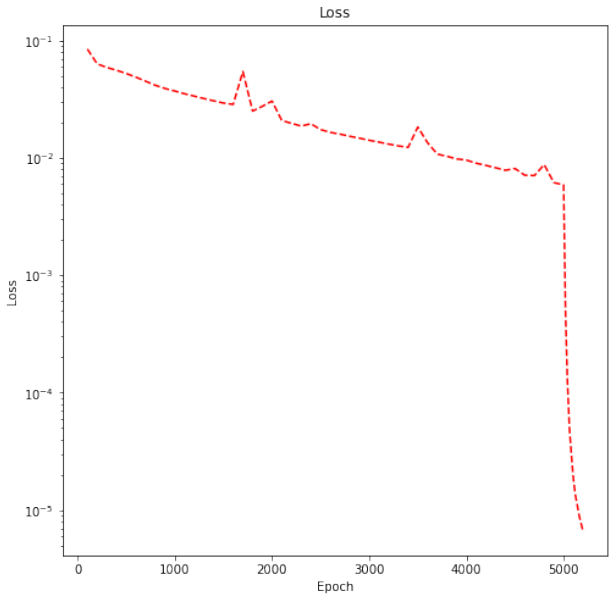

图 6: Loss随Epoch增长之间的关系

测试用采样点: 使用Uniform Sampling方法采样256个点生成的数据由给定的参数 t 决定类似地我们选择t=0.59, 0.79, 0.98三个时刻来展示结果测试结果显示如下:

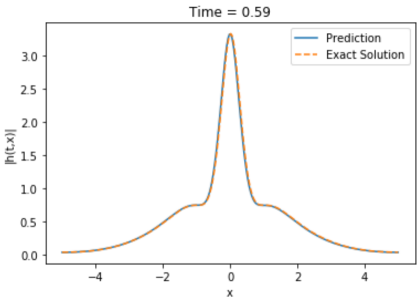

图 7: t=0.59

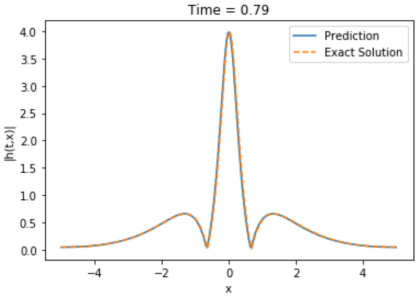

图 8: t=0.79

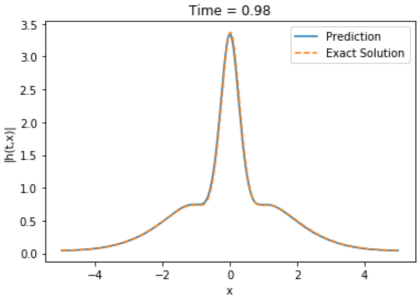

图 9: t=0.98

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。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。

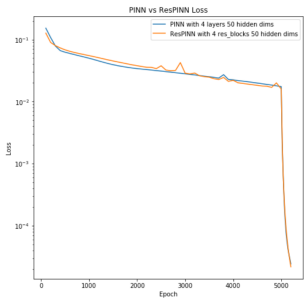

图 10: Loss比较图

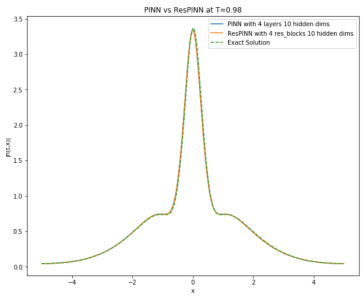

图 11: 解比较图

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。

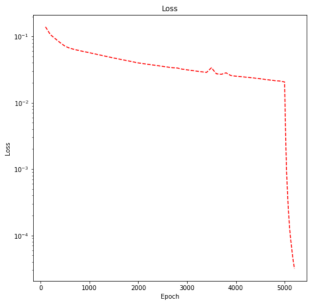

图 12: hidden_layer=2; t=0.79

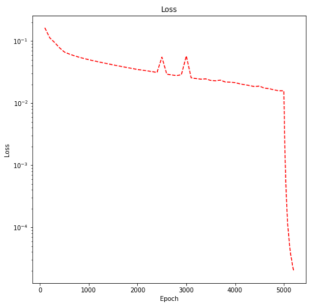

图 13: hidden_layer=8; t=0.79

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

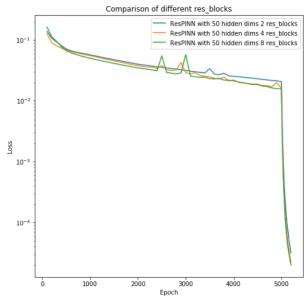

图 14: comparision of hidden_layers; t=0.79

2 利用LSTM预测金融市场

代码插入示例

```
#!/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

data = pd.read_csv('data.csv', usecols=[1])
# plt.plot(data)
# plt.show()

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

max_value = np.max(dataset)
min_value = np.min(dataset)
dataset = (dataset - min_value) / (max_value-min_value)
print(dataset.shape)


def create_dataset(dataset, look_back=10):
    dataX, dataY = [], []
    for i in range(len(dataset)-look_back):
        a = dataset[i:(i+look_back)]
        dataX.append(a)
        dataY.append(dataset[i+look_back])
    return np.array(dataX), np.array(dataY)


X, Y = create_dataset(dataset)
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]

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

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)


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)
        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=4, output_size=1)

criterion = nn.MSELoss()
optimizer = optim.Adam(model.parameters(), lr=1e-2)

epochs = 5000
for epoch in range(epochs):
    out = model(X_train)
    loss = criterion(out, Y_train)

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

```
    if (epoch + 1) % 100 == 0:
        print(f'Epoch: {epoch:5d}, Loss: {loss.item():.4e}')


# test
X = X.transpose(1, 0, 2)
X = torch.from_numpy(X)
Y_pred = model(X)
Y_pred = Y_pred.view(-1).data.numpy()

plt.plot(Y_pred, 'r', label='prediction')
plt.plot(Y, 'b-', label='groundtruth')
plt.legend(loc='best')
plt.show()
```