深圳大学考试答题纸

(以论文、报告等形式考核专用)

二〇 二二 ～二〇 二三 学年度第 二 学期

课程编号 2801000025 课程名称 Python 实现深度学习框架 主讲教师 王浩 何志权 评分

学 号 2021280463 姓名 贺潇乐 专业年级 21 级电子信息工程 05 班

教师评语:

题目: **基于 LSTM 的股票指标预测分析**

目录

基于 LSTM 的股票指标预测分析

贺潇乐

1.摘要

本课程设计旨在利用 LSTM，构建 RNN 循环神经网络，对股票开盘价进行预测和分析，评估预测结果，即分析开盘价的预测结果与实际情况之差异，以便改善预测分析、提高预测准确度，有利于作出更符合市场趋势的股票指标预测分析。

1.Abstracts

This course is designed to use LSTM to build RNN recurrent neural network to forecast and analyze the opening price of stocks, evaluate the forecast results, that is, analyze the difference between the forecast results of the opening price and the actual situation, so as to improve the forecast analysis, improve the accuracy of the forecast, and make the forecast analysis of stock indicators more in line with the market trend.

2.关键词

RNN; LSTM; matrixslow; 开盘价

2.Keywords

RNN;LSTM;matrixslow;opening price

3.前言

时间序列是指按照时间顺序排列的一系列数据，它可以反映某一现象在时间上的变化规律。时间序列分析是利用统计学方法对时间序列数据进行研究和建模，以便对未来进行预测或控制。时间序列分析在经济、金融、社会、自然等领域有着广泛的应用。

传统的时间序列分析方法都是基于线性假设的，不能很好地处理非线性、非平稳、高维度的时间序列数据。随着深度学习技术的发展，一些基于神经网络的时间序列分析方法也逐渐被提出和应用。

神经网络是一种模仿生物神经系统的数学模型，可以从数据中自动提取特征和规律。神经网络可以分为前馈神经网络（FNN）、循环神经网络（RNN）等类型。循环神经网络（RNN）是一种反馈神经网络，它可以处理具有时序性质的数据，如语音、文本、视频等。RNN 的特点是在每个时刻都有一个隐藏状态，该状态不仅取决于当前时刻的输入，还取决于上一个时刻的隐藏状态，从而实现了对历史信息的记忆。RNN 可以看作是一个动态系统，它可以建立输入序列和输出序列之间的映射关系。

长短期记忆网络（LSTM）是一种特殊的 RNN，它可以解决 RNN 在处理长期依赖问题时遇到的梯度消失或爆炸问题。LSTM 通过引入三个门结构（输入门、遗忘门和输出门），来控制信息在单元状态中的流动，从而实现了对长期和短期记忆的有效管理。LSTM 在自然语言处理、语音识别、图像描述等领域都有着优异的表现。

本课程设计旨在利用 LSTM 对股票指标进行预测和分析。股票指标是反映股票市场运行状况的一组数据，如开盘价、收盘价、最高价、最低价、成交量、成交额等。股票指标的预测和分析对于投资者制定投资策略和控制风险具有重要意义。股票指标是一种典型的时间序列数据，它具有非线性、非平稳、高噪声等特点，因此使用 LSTM 进行建模是一种合理的选择。

本课程设计的主要内容如下：

基于 MatrixSlow 设计 LSTM。MatrixSlow 是一个基于 Python 的深度学习框架，它可以实现各种神经网络的构建和训练。通过使用 MatrixSlow，可以加深对 LSTM 和 RNN 的理论理解，并熟练掌握深度学习框架的使用方法。

基于 LSTM，对股票指标进行预测和分析。使用 Daily_ZX.csv 文件作为数据源，该文件是中兴

公司某段时间的股票信息，表中数据从第 3 列开始为股票的信息，例如开盘价、当日最高价、收盘价等。首先对数据进行预处理，如缺失值处理、异常值处理、数据归一化等；然后将数据划分为训练集和测试集，注意训练集和测试集在时间上不能有重叠；接着使用 MatrixSlow 搭建 LSTM 模型，并进行训练和测试；最后对模型的预测结果进行评估和分析，如计算 MAELoss 和 MSELoss、绘制预测曲线等。

4. 正文

压缩包介绍：

基于 LSTM 的训练过程 train.py；

基于 MLP 的训练过程 train_MLP.py；

加载训练好的 LSTM 模型并测试的过程 check.py；

LSTM 模型./model/；MLP 模型./model_MLP/；

训练生成的结果./result/；

训练产生的图片./images/

4.1 LSTM 节点的设计

LSTM 是由 Sepp Hochreiter 和 Jürgen Schmidhuber 于 1997 年联合提出的。原文中提到：“LSTM 的设计就是为了克服这些错误的反向问题。它可以学习桥接超过 1000 步的时间间隔，即使在有噪声、不可压缩的输入序列的情况下，也不会损失短时间延迟能力。这是通过一种特殊的、基于梯度的算法来实现的，它针对的是一种通过特殊单元的内部状态来执行常量(因此既不会爆炸也不会消失)的错误(假设梯度计算在某些特定的体系结构点|被截断，但这并不影响长期的错误)。”[1] 这一段的意思是，LSTM 能够处理长期依赖问题和缓解梯度消失和梯度爆炸，主要在于其特殊的内部结构，即四个门——遗忘门、输入门、记忆门和输出门——LSTM 通过巧妙的门控实现了当前输入与历史信息的选择性记忆与遗忘。一个 LSTM 节点构造如下：

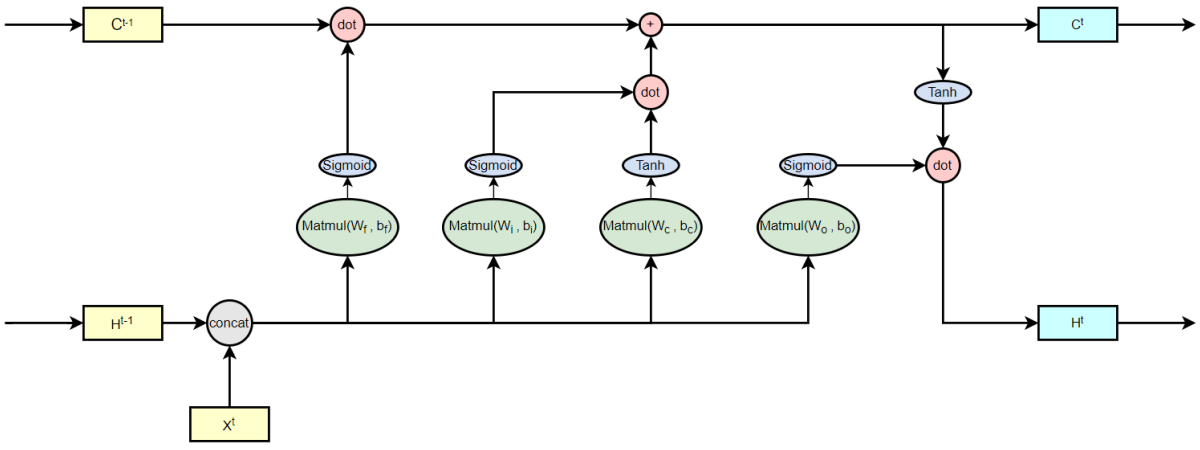

图 1

各个变量节点和操作节点解释如下：

C^{t-1}为上一个 LSTM 输出的记忆节点，作为本 LSTM 节点的第一个输入；本 LSTM 节点输出的记忆节点为C^t；H^{t-1}为上一个 LSTM 的输出，连接上一个输出节点并作为本 LSTM 的另一个输入；拼接节点$concact(H,X)=[H,X]$；遗忘门$Matmul(W_f,b_f)$ 输入门$Matmul(W_i,b_i)$ ；输出门$Matmul(W_o,b_o)$ 记忆门$Matmul(W_c,b_c)$，每个门后接一个激活函数；dot为按元素乘，+为按元

素加。

遗忘门的作用：其父节点为上一个输出和当前输入拼接而成的向量，经过 Sigmoid，被压缩到(0,1)后与上一个 LSTM 输出的记忆节点C^{t-1}按元素相乘。因此，遗忘门的作用是：根据当前输入和上一时刻的输出，选择性“遗忘”或“加深”记忆节点的内容。

输入门和记忆门的作用：同样，输入门有 Sigmoid 函数，决定了记忆门的值有多少可以生效。记忆门直接连接了H^{t-1}和X^t，起处理当前输入信息的作用

输出门的作用：前三个门通过一个加号节点，把当前输入、前时刻的记忆和前时刻的输出综合了起来，成为当前的记忆节点C^t，而输出门通过 Sigmoid 函数，与C^t按元素相乘，决定当前的记忆节点有多少信息可以成为当前的输出节点。

构建 LSTM 层的函数保存在 matrixslow/layer/layer.py 中。

4.2 含 LSTM 的 RNN 网络模型搭建

含 LSTM 的 RNN 网络模型搭建如下：(在./utils.train()函数内搭建网络结构)

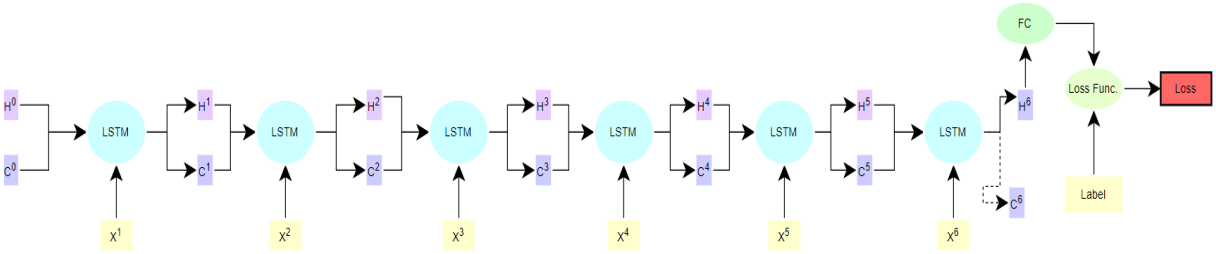

图 2

主体的 LSTM 和隐藏节点可以通过循环进行构造

```
  inputs = [ms.core.Variable(dim=(feature_num, 1), init=False, trainable=False,
name=f'input_vec_{i}') for i in
                range(days)]  # 输入节点列表

      last_step = None
      for ind, input_vec in enumerate(inputs):
          if last_step is None:
              h = ms.Variable((n, 1), init=('Gaussian', 0, 0.005), trainable=True,
name=f'LSTM_h_input_{ind}')
              c = ms.Variable((n, 1), init=('Gaussian', 0, 0.005), trainable=True,
name=f'LSTM_C_{ind}')
          h, c = ms.layer.LSTM(input_vec, h, c, id=ind, n=n,
                               l=feature_num)  # id 表示 LSTM 层所在的层数；n 表示记忆节点
H 和节点 C 的维度，l 表示输入节点 x 的维度
          last_step = h
```

由于 LSTM 较复杂的结构和电脑配置的限制，根据过往 30 天的数据进行开盘价预测在时间上是不允许的，最多只能通过过往 8 天来预测。为了节省时间，通过过往 6 天的数据进行预测。

5. 训练过程和测试结果

5.1 数据描述

“Daily_ZX.csv”中包含按时序排列的 5000 个样本，前两列的特征为编号和日期，属于无效特征，删去后剩下的特征依次为开盘价、当日最高价、当日最低价、收盘价、昨日收盘价、涨跌幅、真实涨跌幅、成交量指标、成交量，没有空值和异常值，这里认为 9 个特征都对预测下一天的开盘价有帮助，故原封不动地保留下来。

逐特征对数据作 min-max 标准化后，选取前 4000 个样本作为训练集，后 1000 个样本作为验证集。

在./utils.train()函数内预处理数据，处理过的数据保存在./data/train_data.npy 和./data/test_data.npy 内

5.2 模型及参数

5.2.1 模型结构在 4.2 中给出。模型参数如下

输入节点数为 6;Learning_rate=0.00015;Epochs=30; 损失函数为 MSELoss; 优化器选择 Adam;

Batch_size=4 一个 mini-batch 内的样本在时间上是连续的;

记忆节点C和输出节点H的维度为 10; 模型末尾的全连接层神经元数目为 40，输出维度为 1。

5.2.2 训练方式

单个 epoch 内，从 4000 个训练集中的随机位置开始，进行 500 次 mini-batch 梯度下降(所以在单个 epoch 内进行 2000 次前向传播和反向传播，500 次梯度更新)；每过一个 epoch，在验证集 1000 个样本中随机选择 500 个序列做验证，展示平均 MAELoss，并将模型保存在./model/epoch n 中；结束训练后，获取验证集损失最小的模型，将信息写入到./result/training_result.txt 中；以上的过程全部在./utils.train()中体现。

5.2.3 测试方式

接下来定义./utils.test(best_epoch)函数，用于加载特定 epoch 的模型，用验证集做验证；然后，在验证集中取随机 200 个连续序列，展示连续的开盘价预测结果和开盘价真实序列的差异;最后,将验证集 1000 个样本的 MAE 损失均值和方差保存在./result/validation_result.txt 中。

5.3 训练和测试结果

5.3.1 训练结果和损失曲线

在./main.ipynb 中调用 training(days=6,epochs=30)，训练完成后，查看./result/training_result.txt 的内容以及损失函数下降曲线。

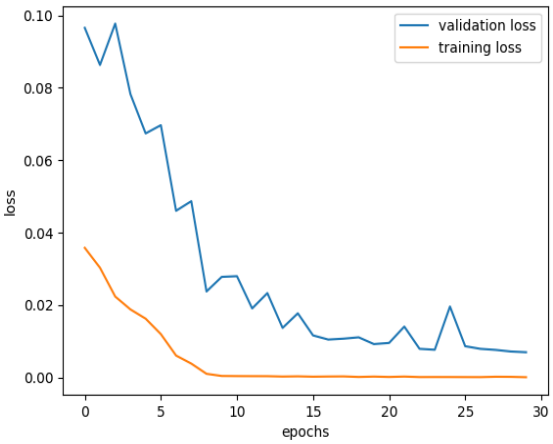

图 4

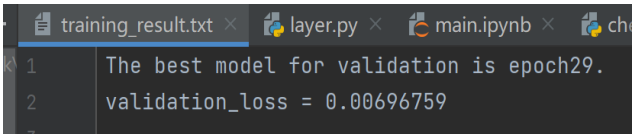

图 3

5.3.2 测试结果和连续预测对比图

结果显示第 29 个 epoch 的结果最佳，在./check.py 中调用 test(best_epoch=29,days=6) 后，画出的对比图和./result/validation_result.txt 的内容。

validation_result.txt 中的内容，包括 MAELoss 的均值和方差

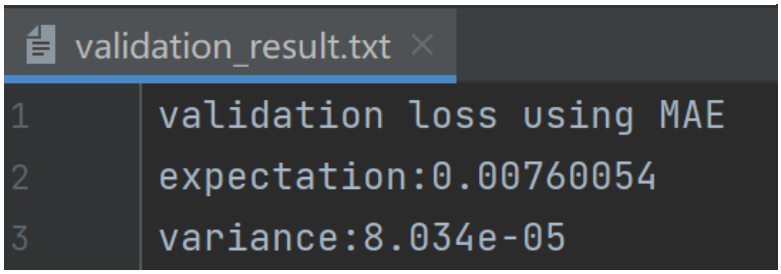

图 5

展示两组测试集中随机连续预测和真实值的对比图

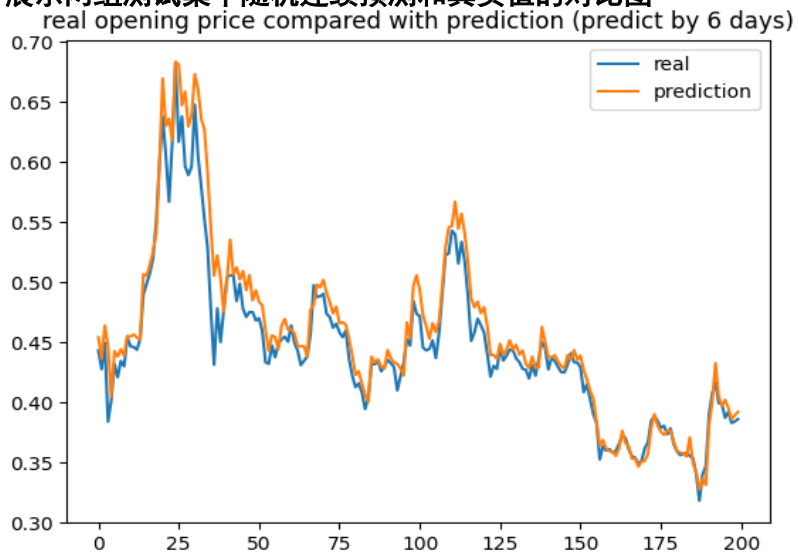

图 6

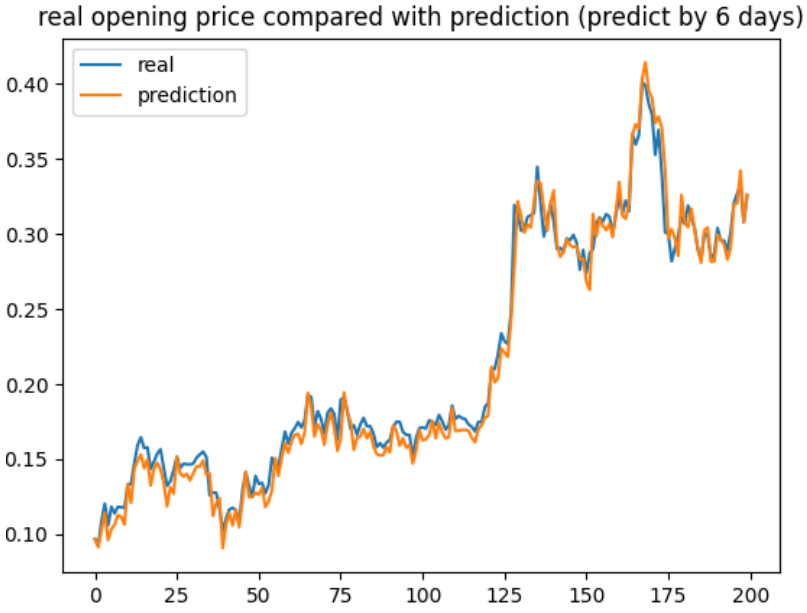

图 7

5.4 MLP 全连接神经网络

将 LSTM 改为 MLP 全连接神经网络，获取训练的结果。定义函数 train_MLP()，位置在 utils.py 中。神经网络的输入为前 7 天样本信息展平后的向量，维度为 7*9=63；输出和标签为后一天的开盘价，维度为 1。训练 30 个 epochs，发现训练集和验证集的损失均无法正常地收敛，如下：

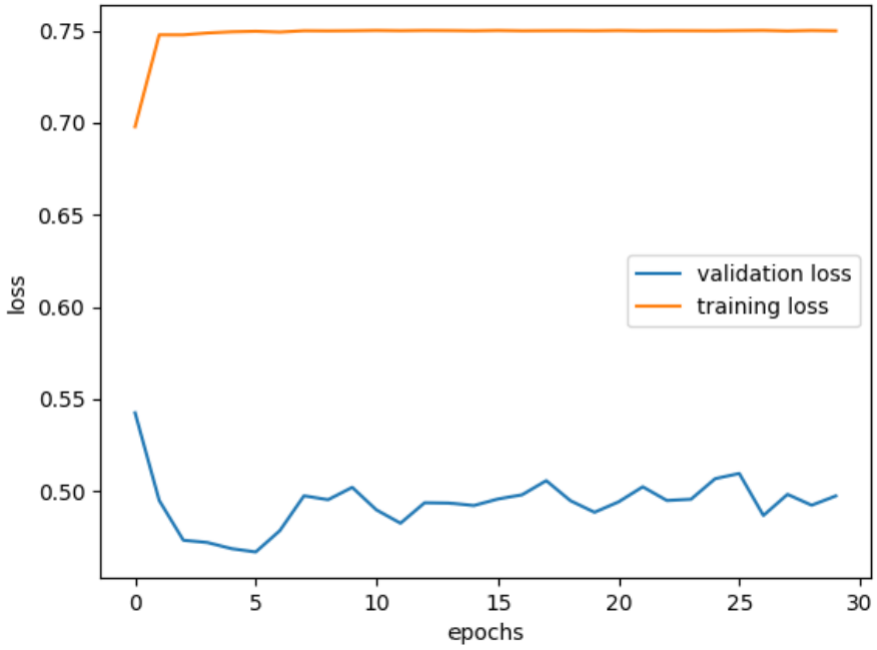

图 8

5.5 分析

用 LSTM 预测的结果与实际的开盘价相比，在数值上，尤其是在一些高频的转折点处，存在一定的误差。但是它已经能够比较准确地反映出股票的涨跌趋势和波动规律。预测的结果能够为投资者提供有价值的参考和建议，能够帮助投资者制定合理的投资策略，能够提高投资者的收益和降低风险。

MLP 没有表现出处理股价预测问题的能力，主要是因为相比于 RNN ，MLP 神经网络无法直接处理和生成变长的序列数据，并且无法捕捉序列数据的时间顺序和相关性。RNN 神经网络可以利用循环单元的内部状态来存储和传递序列数据的历史信息，从而捕捉序列数据的时间顺序和依赖关系。因此，RNN 神经网络可以更好地处理时序问题。

6. 总结

本项目的目的是利用 LSTM，根据历史数据预测股票的开盘价。项目的主要步骤如下：

数据收集和预处理。对“ZX_Daily.csv”数据进行标准化，划分了训练集和验证集。

模型构建。基于 LSTM 构建 RNN 神经网络，在时间和设备允许的基础上，选择了比较理想的模型参数。搭建 MLP 全连接神经网络，比较训练结果。

模型评估和优化。使用了均方误差 MSE、平均绝对误差 MAE，对模型的预测效果进行了评估。模型能够比较准确地预测开盘价变动的趋势，在数值上与标签存在一定的差距，但在可允许范围内。

没有能够实现根据过往 30 天的预测，需要学习如何让 matrixslow 适配 GPU。

7. 引用

[1] Sepp Hochreiter and Jürgen Schmidhuber. Long Short-Term Memory. Neural Computation 9, 1735-1780 (1997) 。

8. 代码

```
def LSTM(x, *input, **kwargs):
    '''
    X：当前输入节点
    C_0：上一个记忆节点
    H_0：
    Wf Bf：遗忘门权重
    Wi Bi：输入门权重
    Wo Bo：输出门权重
    Wc Bc：
    C_1：输出的记忆细胞
    H_1：
    '''
    id = kwargs.get('id', 'default')
    n = kwargs.get('n', 10)
    l = kwargs.get('l', 16)
    X = x
    H_0, C_0 = input
    concat = Concat(H_0, X, axis=0, name=f'LSTM_concat_{id}')

    Wf = Variable((n, n + l), init=('Gaussian', 0, 0.005), trainable=True, name=f'LSTM_Wf_{id}')
    Bf = Variable((n, 1), init=('Gaussian', 0, 0.005), trainable=True, name=f'LSTM_Bf_{id}')

    Wi = Variable((n, n + l), init=('Gaussian', 0, 0.005), trainable=True, name=f'LSTM_Wi_{id}')
    Bi = Variable((n, 1), init=('Gaussian', 0, 0.005), trainable=True, name=f'LSTM_Bi_{id}')

    Wc = Variable((n, n + l), init=('Gaussian', 0, 0.005), trainable=True, name=f'LSTM_Wc_{id}')
    Bc = Variable((n, 1), init=('Gaussian', 0, 0.005), trainable=True, name=f'LSTM_Bc_{id}')

    Wo = Variable((n, n + l), init=('Gaussian', 0, 0.005), trainable=True, name=f'LSTM_Wo_{id}')
    Bo = Variable((n, 1), init=('Gaussian', 0, 0.005), trainable=True, name=f'LSTM_Bo_{id}')

    gate_forget = Logistic(Add(MatMul(Wf, concat), Bf), name=f'LSTM_gateforget_{id}')
    gate_input = Logistic(Add(MatMul(Wi, concat), Bi), name=f'LSTM_gateinput_{id}')
    C_hat = Logistic(Add(MatMul(Wc, concat), Bc), name=f'LSTM_C-hat_{id}')
    gate_output = Logistic(Add(MatMul(Wo, concat), Bo), name=f'LSTM_gateoutput_{id}')

    multiply_C0_forget = Multiply(C_0, gate_forget, name=f'LSTM_multiply_C0_forget_{id}')
    multiply_input_Chat = Multiply(gate_input, C_hat, name=f'LSTM_multiply_gateinput_Chat_{id}')

    add = Add(multiply_C0_forget, multiply_input_Chat, name=f'LSTM_add_{id}')
    C_1 = add
    tanh = Tanh(add, name=f'LSTM_tanh_{id}')
    H_1 = Multiply(tanh, gate_output, name=f'LSTM_h_{id}')
    return H_1, C_1
```

```
 # utils.py
import matrixslow as ms
import numpy as np
```

```
import pandas as pd
from matplotlib import pyplot as plt


def train(lr=0.00015, days=7, epochs=30, batch_size=4):
    data_csv = pd.read_csv('Daily_ZX.csv')
    data = data_csv.to_numpy()[:, 2:].astype(np.float64)
    data = (data - np.min(data, axis=0)) / (np.max(data, axis=0) - np.min(data, axis=0))  # 特征标准化

    train_data = data[:4000, :]
    test_data = data[4000:, :]
    np.save('./data/train_data.npy', train_data)
    np.save('./data/test_data.npy', test_data)

    # 构造计算图，输入节点数为 days，输出一个节点
    feature_num = 9
    n = 10
    inputs = [ms.core.Variable(dim=(feature_num, 1), init=False, trainable=False, name=f'input_vec_{i}')
for i in
              range(days)]  # 输入节点列表

    last_step = None
    for ind, input_vec in enumerate(inputs):
        if last_step is None:
            h = ms.Variable((n, 1), init=('Gaussian', 0, 0.005), trainable=True,
name=f'LSTM_h_input_{ind}')
            c = ms.Variable((n, 1), init=('Gaussian', 0, 0.005), trainable=True, name=f'LSTM_C_{ind}')
        h, c = ms.layer.LSTM(input_vec, h, c, id=ind, n=n,
                             l=feature_num)  # id 表示 LSTM 层所在的层数；n 表示记忆节点
H 和节点 C 的维度，l 表示输入节点 x 的维度
        last_step = h

    fc1 = ms.layer.fc(last_step, n, 40, "ReLU")  # 记忆节点到输出节点的全连接层
    output = ms.layer.fc(fc1, 40, 1, "ReLU")
    label = ms.core.Variable((1, 1), trainable=False)
    loss = ms.ops.loss.MeanSquaredErrorLoss(output, label)  # 训练使用的 loss 函数为 MSE
    test_loss = ms.ops.MAELoss(output, label)  # 测试使用的 loss 函数为 MAE
    learning_rate = lr
    optimizer = ms.optimizer.Adam(ms.default_graph, loss, learning_rate)

    test_loss_list_eachepoch = []
    train_loss_list_eachepoch = []

    for epoch in range(epochs):
        loss_one_iter = []
        for itr_count in range(500):  # 每个 epoch 进行 1000 次 minibatch 梯度下降
            pos = np.random.randint(len(train_data) - 1 - days - batch_size)
            # 单次梯度下降过程
            loss_list = []
            for batch_k in range(batch_size):
                # 提取连续的时间序列，赋给输入节点
                start = pos + batch_k
                seq = data[start: start + days]
                for j, x in enumerate(inputs):
                    x.set_value(np.mat(np.expand_dims(seq[j].T, axis=1)))
                # 给标签节点赋值
                target_ch = data[start + days, 0]
```

```
                label.set_value(np.mat(target_ch))
                optimizer.one_step()
                loss_list.append(loss.value[0,0])
            loss_one_iter.append(np.mean(loss_list))
            print(f'epoch:{epoch}\t\titer_count:{itr_count}\t\tMSELoss:{round(loss.value[0, 0], 9)}',
end='\n')
            optimizer.update()

        train_loss_list_eachepoch.append(np.mean(loss_one_iter))

        test_loss_list = []
        # 每过一个 epoch 测试一次,在 1000 个验证样本中取 500 个
        print('testing...')
        for itr_count in range(500):
            print(f'{itr_count}/500', end='')
            pos = np.random.randint(len(test_data) - 1 - days)
            seq = test_data[pos: pos + days]
            for j, x in enumerate(inputs):
                x.set_value(np.mat(np.expand_dims(seq[j].T, axis=1)))
            label.set_value(np.mat(test_data[pos + days, 0]))
            test_loss.forward()
            test_loss_list.append(test_loss.value)
            print('\r', end='')
        print(
            f'epoch:{epoch}\ttesting loss using MAE\texpectation:{np.round(np.mean(test_loss_list),
8)}\tvariance:{np.round(np.var(test_loss_list), 8)}')
        test_loss_list_eachepoch.append(np.round(np.mean(test_loss_list), 8))
        saver = ms.trainer.Saver(root_dir=f'./model/epoch{epoch}')  # 每个 epoch 保存模型
        saver.save(graph=ms.default_graph)

    min_loss, best_epoch = np.min(test_loss_list_eachepoch), np.argmin(test_loss_list_eachepoch)
    with open('result/training_result.txt', 'w') as f:
        f.write(f'The best model for validation is epoch{best_epoch}.\nvalidation_loss = {min_loss}')

    # 画出测试 loss 收敛曲线
    plt.figure()
    plt.plot(test_loss_list_eachepoch)
    plt.plot(train_loss_list_eachepoch)
    plt.xlabel('epochs')
    plt.ylabel('loss')
    plt.legend(['validation loss', 'training loss'])
    plt.show()
    print('Done.')


def test(best_epoch, days=7):  # 取所有的验证样本进行验证
    test_data = np.load('./data/test_data.npy')
    feature_num = 9
    n = 10
    inputs = [ms.core.Variable(dim=(feature_num, 1), init=False, trainable=False, name=f'input_vec_{i}')
for i in
              range(days)]  # 输入节点列表

    last_step = None
    for ind, input_vec in enumerate(inputs):
        if last_step is None:
            h = ms.Variable((n, 1), init=('Gaussian', 0, 0.005), trainable=True,
```

```
name=f'LSTM_h_input_{ind}')
                c = ms.Variable((n, 1), init=('Gaussian', 0, 0.005), trainable=True, name=f'LSTM_C_{ind}')
        h, c = ms.layer.LSTM(input_vec, h, c, id=ind, n=n,
                                        l=feature_num)  # id 表示 LSTM 层所在的层数；n 表示记忆节点 H 和节点 C 的维度，l 表示输入节点 x 的维度
        last_step = h

    fc1 = ms.layer.fc(last_step, n, 40, "ReLU")  # 记忆节点到输出节点的全连接层
    output = ms.layer.fc(fc1, 40, 1, "ReLU")
    label = ms.core.Variable((1, 1), trainable=False)
    test_loss = ms.ops.MAELoss(output, label)  # 测试使用的 loss 函数为 MAE，因为在这里 MAE 能直观体现预测和标签的差异

    saver = ms.trainer.Saver(f'./model/epoch{best_epoch}/')
    saver.load(model_file_name='model.json', weights_file_name='weights.npz')
    test_loss_list = []

    '''
    先在测试集中选取长度为 200 的连续序列查看标准化后的输出结果
    '''
    length = 200
    pos = np.random.randint(len(test_data) - 1 - 2 * days - length)
    real = test_data[pos:pos + length, 0]
    pred = []
    print('Preparing prediction curve...')
    for i in range(length):
        print(f'{i}/{length}', end='')
        seq = test_data[pos - days + i: pos + i]
        for j, x in enumerate(inputs):
            x.set_value(np.mat(np.expand_dims(seq[j].T, axis=1)))
        label.set_value(np.mat(test_data[pos + days, 0]))
        test_loss.forward()
        pred.append(output.value[0, 0])
        print('\r', end='')
    pred = np.array(pred)
    plt.figure()
    plt.plot(real)
    plt.plot(pred)
    plt.title(f'real opening price compared with prediction (predict by {days} days)')
    plt.legend(['real', 'prediction'])
    plt.savefig(f'images/real opening price compared with prediction (predict by {days} days).png')
    plt.show()

    print('Testing...')
    for itr_count in range(1000 - days):
        print(f'{itr_count}/{1000 - days + 1}', end=' ')
        pos = itr_count
        seq = test_data[pos: pos + days]
        for j, x in enumerate(inputs):
            x.set_value(np.mat(np.expand_dims(seq[j].T, axis=1)))

        label.set_value(np.mat(test_data[pos + days, 0]))
        test_loss.forward()
        test_loss_list.append(test_loss.value)
        print('\r', end='')
    test_result = f'validation loss using MAE\nexpectation:{np.round(np.mean(test_loss_list), 8)}\nvariance:{np.round(np.var(test_loss_list), 8)}'
```

```
    with open('./result/validation_result.txt', 'w') as f:
        f.write(test_result)
    print('Done.')

def train_MLP(lr=0.0001, days=7, epochs=30, batch_size=4):
    data_csv = pd.read_csv('Daily_ZX.csv')
    data = data_csv.to_numpy()[:, 2:].astype(np.float64)
    data = (data - np.min(data, axis=0)) / (np.max(data, axis=0) - np.min(data, axis=0))  # 特征标准化

    train_data = data[:4000, :]
    test_data = data[4000:, :]
    np.save('./data/train_data.npy', train_data)
    np.save('./data/test_data.npy', test_data)

    # 构造 MLP，输入维度为 9*days，输出维度为 1
    feature_num = 9
    n = 128
    input = ms.Variable((feature_num * days, 1), init=False, trainable=False, name='input')
    W1 = ms.Variable((n, feature_num * days), init=('Gaussian', 0, 0.005), trainable=True, name='W1')
    b1 = ms.Variable((n, 1), init=('Gaussian', 0, 0.005), trainable=True, name='b1')

    W2 = ms.Variable((n, n), init=('Gaussian', 0, 0.005), trainable=True, name='W2')
    b2 = ms.Variable((n, 1), init=('Gaussian', 0, 0.005), trainable=True, name='b2')

    W3 = ms.Variable((1, n), init=('Gaussian', 0, 0.005), trainable=True, name='W3')
    b3 = ms.Variable((1, 1), init=('Gaussian', 0, 0.005), trainable=True, name='b3')

    fc1 = ms.ops.ReLU(ms.ops.Add(ms.ops.MatMul(W1, input), b1), name='ReLU1')
    fc2 = ms.ops.ReLU(ms.ops.Add(ms.ops.MatMul(W2, fc1), b2), name='ReLU2')
    output = ms.ops.Tanh(ms.ops.Add(ms.ops.MatMul(W3, fc2), b3), name='output')
    label = ms.core.node.Variable((1, 1), trainable=False)
    loss = ms.ops.Add(ms.ops.loss.MAELoss(output, label), ms.ops.loss.MeanSquaredErrorLoss(output,
label))
    test_loss = ms.ops.MAELoss(output, label)  # 测试使用的 loss 函数为 MAE
    learning_rate = lr
    optimizer = ms.optimizer.Adam(ms.default_graph, loss, learning_rate)

    test_loss_list_eachepoch = []
    train_loss_list_eachepoch = []

    for epoch in range(epochs):
        loss_one_iter = []
        for itr_count in range(1000):  # 每个 epoch 进行 1000 次 minibatch 梯度下降
            pos = np.random.randint(len(train_data) - 1 - days - batch_size)
            # 单次梯度下降过程
            loss_list = []
            for batch_k in range(batch_size):
                # 提取连续的时间序列，赋给输入节点
                start = pos + batch_k
                seq = data[start: start + days]
                input.set_value(np.mat(seq.ravel()).T)
                # 给标签节点赋值
                target_ch = data[start + days, 0]
                label.set_value(np.mat(target_ch))
                optimizer.one_step()
                loss_list.append(loss.value[0, 0])
            loss_one_iter.append(np.mean(loss_list))
```

```
            print('\r', end='')
            print(f'epoch:{epoch}\t\titer_count:{itr_count}\t\tLoss:{round(loss.value[0, 0], 9)}', end='')
            optimizer.update()

        train_loss_list_eachepoch.append(np.mean(loss_one_iter))

        test_loss_list = []
        # 每过一个 epoch 测试一次,在 1000 个验证样本中取 500 个
        print('\ntesting...')
        for itr_count in range(500):
            print(f'{itr_count}/500', end='')
            pos = np.random.randint(len(test_data) - 1 - days)
            seq = test_data[pos: pos + days]
            input.set_value(np.mat(seq.ravel()).T)
            label.set_value(np.mat(test_data[pos + days, 0]))
            test_loss.forward()
            test_loss_list.append(test_loss.value)
            print('\r', end='')
        print(
            f'epoch:{epoch}\ttesting loss\texpectation:{np.round(np.mean(test_loss_list),
8)}\tvariance:{np.round(np.var(test_loss_list), 8)}')
        test_loss_list_eachepoch.append(np.round(np.mean(test_loss_list), 8))
        saver = ms.trainer.Saver(root_dir=f'./model_MLP/epoch{epoch}')  # 每个 epoch 保存模型
        saver.save(graph=ms.default_graph)

    min_loss, best_epoch = np.min(test_loss_list_eachepoch), np.argmin(test_loss_list_eachepoch)
    with open('result/training_result_MLP.txt', 'w') as f:
        f.write(f'The best model for validation is epoch{best_epoch}.\nvalidation_loss = {min_loss}')

    # 画出测试 loss 收敛曲线
    plt.figure()
    plt.plot(test_loss_list_eachepoch)
    plt.plot(train_loss_list_eachepoch)
    plt.xlabel('epochs')
    plt.ylabel('loss')
    plt.legend(['validation loss', 'training loss'])
    plt.show()
    print('Done.')
```