模型训练代码.py

|  |
| --- |
| import torch.optim as optim  from tqdm import tqdm  from torch import nn  import torch  from model.LSTM import MyNet  from utils.dataset import TestDataset, TrainDataset  from torch.utils.data import DataLoader  import matplotlib.pyplot as plt  import matplotlib.dates as mdates  from datetime import datetime  import numpy as np  import time  plt.rcParams['font.sans-serif']=['SimHei']  plt.rcParams['axes.unicode\_minus'] = False  def TimePlus(fomattime):      fomattime = time.strptime(fomattime, '%Y-%m-%d')      outtime = time.mktime(fomattime) + 86400      outtime = time.strftime('%Y-%m-%d', time.localtime(outtime))      return outtime  def train():      #定义超参数      device = torch.device('cuda')      input\_size = 10      seq\_len = 30      hidden\_size = 32      output\_size = 1      #定义模型      net = MyNet(input\_size=input\_size, hidden\_size=hidden\_size, output\_size=output\_size, seq\_len=seq\_len)      net.to(device, dtype=torch.float32)      # 定义数据集      train\_dataset = TrainDataset(r'上证指数.csv', 30)      test\_dataset = TestDataset(r'上证指数.csv', 30)      max\_list, min\_list = train\_dataset.max\_list, train\_dataset.min\_list      train\_loader = DataLoader(dataset=train\_dataset, batch\_size=32, shuffle=False)      test\_loader = DataLoader(dataset=test\_dataset, batch\_size=1, shuffle=False)      # 定义损失函数和优化器      loss\_function = nn.MSELoss()      optimizer = optim.Adam(net.parameters())      total\_loss = 0.0      for epoch in tqdm(range(100)):          for index, (data, label) in enumerate(train\_loader):              optimizer.zero\_grad()              data = data.to(device, dtype=torch.float32)              label = label.to(device, dtype = torch.float32)              label = torch.unsqueeze(label, dim=1)              pred = net(data)              loss = loss\_function(pred, label)              loss.backward()              optimizer.step()              total\_loss += loss.item()          if (epoch + 1) % 10 == 0:              print('Epoch:', epoch + 1, 'loss: ', total\_loss)              torch.save(net, 'lstm\_parameters.pth')              total\_loss = 0.0          torch.save(net, 'lstm\_parameters.pth')      pred\_list, label\_list = [], []      time\_list = test\_dataset.time\_list      date\_list = [datetime.strptime(d, '%Y-%m-%d').date() for d in time\_list]      for index, (data, label) in enumerate(test\_loader):          data = data.to(device, dtype=torch.float32)          label = label.to(device, dtype=torch.float32)          pred = net(data)          pred = torch.squeeze(pred)          pred1, label = float(pred), float(label)          pred1 = pred1 \* (max\_list[1] - min\_list[1]) + min\_list[1]          label = label \* (max\_list[1] - min\_list[1]) + min\_list[1]          pred\_list.append(pred1)          label\_list.append(label)      with open('test\_result.csv','w', encoding='utf-8-sig') as f:          f.write('日期,实际值,预测值\n')          for index in range(len(date\_list)):              f.write('%s,%d,%d\n' % (time\_list[index], label\_list[index], pred\_list[index]))      plt.plot(date\_list, pred\_list, label='预测值')      plt.plot(date\_list, label\_list, label='实际值')      plt.gca().xaxis.set\_major\_formatter(mdates.DateFormatter('%Y-%m-%d'))      plt.gca().xaxis.set\_major\_locator(mdates.DayLocator())      plt.xticks(date\_list[::50])      plt.gcf().autofmt\_xdate()      plt.title('股票预测')      plt.legend()      plt.show()  if \_\_name\_\_ == '\_\_main\_\_':      train() |

上证爬虫.py

|  |
| --- |
| import requests  import re  HEADERS = {      'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/90.0.4430.72 Safari/537.36'}  def GetStockData():      url = 'http://37.push2his.eastmoney.com/api/qt/stock/kline/get?cb=jQuery331025420252272686183\_1626515664856&secid=1.000001&ut=fa5fd1943c7b386f172d6893dbfba10b&fields1=f1%2Cf2%2Cf3%2Cf4%2Cf5%2Cf6&fields2=f51%2Cf52%2Cf53%2Cf54%2Cf55%2Cf56%2Cf57%2Cf58%2Cf59%2Cf60%2Cf61&klt=101&fqt=1&end=20500101&lmt=2562&\_=1626515664863'      response = requests.get(url=url, headers= HEADERS)      html = response.text      html = eval(re.findall(r'1626515664856\((.\*?)\);', html)[0])['data']['klines']      with open('上证指数.csv', 'w',encoding='utf-8-sig') as f:          f.write('序号,时间,开盘价,收盘价,当日最高,当日最低,当天成交量,当天成交额,振幅,涨跌幅,涨跌额,换手率\n')          for index, data in enumerate(html):              print(data)              f.write(str(index +1) + ',' + data + '\n')  if \_\_name\_\_ == '\_\_main\_\_':      GetStockData() |

数据集加载.py

|  |
| --- |
| from torch.utils.data import DataLoader, Dataset, random\_split  import numpy as np  class TrainDataset(Dataset):      def \_\_init\_\_(self, path, sequence):          self.sequence = sequence          with open(path, 'r', encoding='utf-8-sig') as f:              content\_list = f.readlines()          # 归一化          max\_list, min\_list = [], []          data\_list = []          time\_list = []          for index, one\_line in enumerate(content\_list):              if index == 0:                  continue              data\_list.append(one\_line.replace('\n', '').split(',')[2:])              time\_list.append(one\_line.replace('\n', '').split(',')[1])          data\_array = np.array(data\_list, dtype=np.float32)          for i in range(data\_array.shape[1]):              max\_list.append(np.max(data\_array[:,i]))              min\_list.append(np.min(data\_array[:,i]))              data\_array[:,i] = (data\_array[:,i] - np.min(data\_array[:,i])) / (np.max(data\_array[:,i]) - np.min(data\_array[:,i]))          self.max\_list, self.min\_list = max\_list, min\_list          content\_list = list(data\_array)          content\_list = content\_list[0: int(0.7 \* len(content\_list))]          time\_list = time\_list[0:int(0.7 \* len(time\_list))]          self.time\_list = time\_list          self.content\_list = content\_list      def \_\_getitem\_\_(self, item):          data = self.content\_list[item: item + self.sequence]          label = self.content\_list[item + self.sequence][1]          data = np.array(data, dtype=np.float32)          label = np.array(label, dtype=np.float32)          return data, label      def \_\_len\_\_(self):          return len(self.content\_list) - self.sequence  class TestDataset(Dataset):      def \_\_init\_\_(self, path, sequence):          self.sequence = sequence          with open(path, 'r', encoding='utf-8-sig') as f:              content\_list = f.readlines()          # 归一化          max\_list, min\_list = [], []          data\_list = []          time\_list = []          for index, one\_line in enumerate(content\_list):              if index == 0:                  continue              data\_list.append(one\_line.replace('\n', '').split(',')[2:])              time\_list.append(one\_line.replace('\n', '').split(',')[1])          data\_array = np.array(data\_list, dtype=np.float32)          for i in range(data\_array.shape[1]):              max\_list.append(np.max(data\_array[:,i]))              min\_list.append(np.min(data\_array[:,i]))              data\_array[:,i] = (data\_array[:,i] - np.min(data\_array[:,i])) / (np.max(data\_array[:,i]) - np.min(data\_array[:,i]))          content\_list = list(data\_array)          content\_list = content\_list[int(0.7 \* len(content\_list)): len(content\_list) - 1]          time\_list = time\_list[int(0.7 \* len(time\_list)): len(time\_list) - 1]          time\_list = time\_list[self.sequence:]          self.time\_list = time\_list          self.content\_list = content\_list          self.max\_list, self.min\_list = max\_list, min\_list      def \_\_getitem\_\_(self, item):          data = self.content\_list[item: item + self.sequence]          label = self.content\_list[item + self.sequence][1]          data = np.array(data, dtype=np.float32)          label = np.array(label, dtype=np.float32)          return data, label      def \_\_len\_\_(self):          return len(self.content\_list) - self.sequence  if \_\_name\_\_ == '\_\_main\_\_':      # train\_dataset = TrainDataset(r'C:\Users\12517\Desktop\股票分析\上证指数.csv', 30)      test\_dataset = TestDataset(r'C:\Users\12517\Desktop\股票分析\上证指数.csv', 30)      # train\_length = int(0.8 \* len(dataset))      # test\_length = len(dataset) - train\_length      # train\_dataset, test\_dataset = random\_split(dataset=dataset, lengths=[train\_length, test\_length])      train\_loader = DataLoader(dataset=test\_dataset, batch\_size=32, shuffle=False)      for data, label in train\_loader:          break |

模型搭建.py

|  |
| --- |
| import torch.nn as nn  class MyNet(nn.Module):      def \_\_init\_\_(self, input\_size=7, hidden\_size=32, output\_size=1, seq\_len=30):          super(MyNet, self).\_\_init\_\_()          self.input\_size = input\_size          self.hidden\_size = hidden\_size          self.output\_size = output\_size          self.lstm = nn.LSTM(input\_size=input\_size, hidden\_size=hidden\_size, batch\_first=True)          self.fc = nn.Linear(self.hidden\_size \* seq\_len, self.output\_size)      def forward(self, input):          out, \_ = self.lstm(input)          b, s, h = out.size()          out = self.fc(out.reshape(b, s \* h))          return out  if \_\_name\_\_ =='\_\_main\_\_':      net = MyNet()      print(list(net.parameters())) |