基于LSTM的天气数据时间序列预测

2210210338 张淼

# 一、引言

随着全球气候变化研究的深入，人们逐渐认识到极端天气变化对人类和周边环境的影响越来越大，在极端天气现象所带来的影响中，极端高温或者极端低温所带来的损失是其中重要的一方面，所以，目前对天气相关数据的研究引起人们的广泛关注。在全球变暖，生态气候发生变化的情况下，研究天气的变化也变得越来越重要，因为天气关系着我们的方方面面，我们要能够充分利用天气数据并挖掘其中有价值的信息。

气象变化是一个很复杂的过程，同时天气时刻都在变化并且受到各种因素的影响，只有充分的考虑各种影响因素，才可以更加准确地对天气进行预测，达到预期效果。随着科技的进步，我们进入了大数据时代，通过数据挖掘技术，挖掘其中蕴含的信息进而指导我们的日常生活。在气象领域也产生了大量的数据，我们可以充分利用气象大数据，建立气温等数据的预测模型，进而对未来的天气现象进行预测分析。

深度学习在大数据时代也迎来了蓬勃的发展，深度学习通过对数据的拟合，使得该方法在不同领域得到应用。由于气温受多种因素的影响具有随机变化的特性，传统的气象研究主要是通过时间序列模型、马尔科夫模型、遗传算法和支持向量机等算法对气象数据进行研究。长短时记忆网络（LSTM）是一种时间递归神经网络，可以更好地拟合非线性关系，它既解决了前馈网络中只能利用固定长度的数据序列来进行预测的问题，又解决了递归网络训练困难的问题。

本文采用时间序列分析方法，选取天气相关数据之一温度的变化趋势进行了分析，并利用LSTM模型对每一时刻的温度做预测。天气是随机的、不确定的，受多种因素的影响，所以温度预测的精度不高。为了提高温度预报的精度，本文提出了一种基于LSTM模型的气温预测模型。该模型能够相对准确的对气温进行预测，利用该模型能够记住很长时间的信息的优点对气温进行预测，能准确地对气温进行预测，对农业产生一定参考价值，具有重要的应用前景。

# 二、模型构建

## (一) LSTM神经元结构

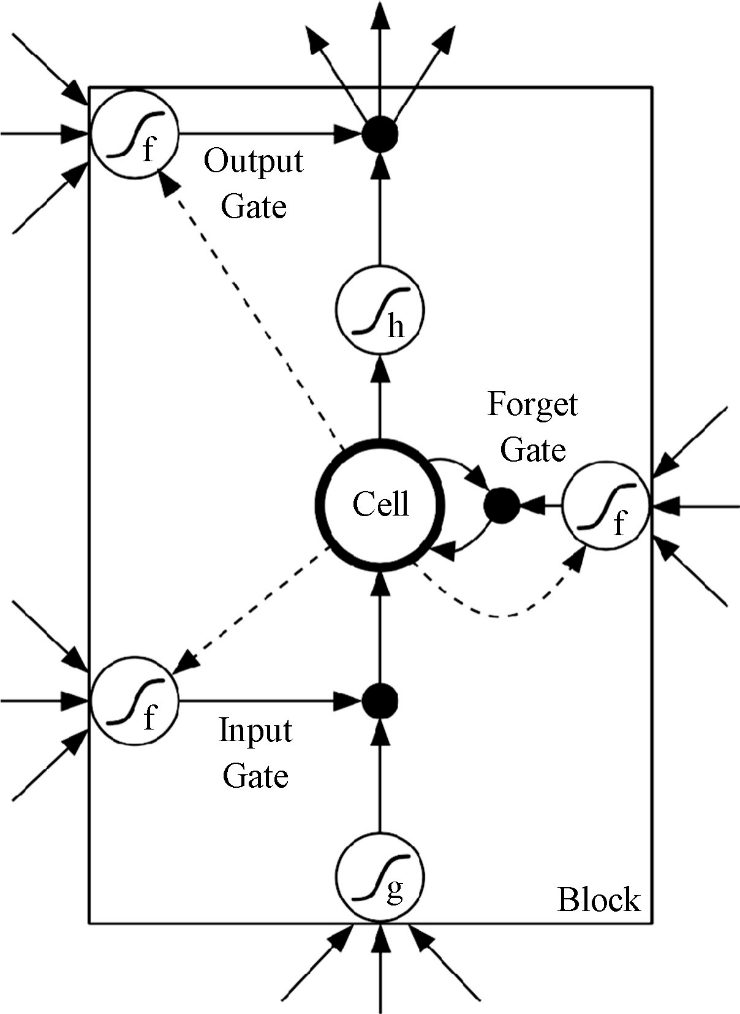
LSTM神经网络最早由Hochreiter 和 Schmidhuber (1997)提出, 由于能更好地发现长期依赖关系而被广泛用于处理序列信息, 如语音识别、机器翻译等。LSTM每个神经元的结构见图1 (Grave, 2012), 其内部包括一个记忆储存 (Cell) 和三个门控 (Gates) 设置, Cell记录神经元状态, 输入门 (Input Gate) 和输出门 (Output Gate) 用来接收、输出参数和修正参数, 遗忘门 (Forget Gate) 用来控制上一单元状态的被遗忘程度。

## (二) LSTM神经网络的构建

深层神经网络存在两种可能影响模型训练效果的问题, 一是梯度消失导致神经网络难以收敛, 二是过拟合导致测试集失效。研究表明, Batch-Normalization (BN) 能有效解决梯度消失问题, Dropout技术通过阻止神经元共适应能够缓解过拟合问题。

## (三) 训练方法及优化器选择

优化器方面, 本文采用Adam优化器 (Adaptive Moment Estimation, 适应性矩估计)进行优化训练。Adam优化器由Kingma和Ba (2015)提出, 是目前最常用的算法。与其他自适应学习率算法相比, Adam算法收敛速度更快、学习效果更为有效。本文目标是预测气温, 故选取均方误差 (Mean Square Error, MSE)作为损失函数。本文基于Python语言环境, 并以PyTorch作为深度学习框架进行训练及预测。



**图1 LSTM神经元结构示意图(Grave, 2012)**

# 三、数据来源及样本选择

## (一) 数据来源

马克斯·普朗克研究所气象站测量的 20 个气象指标在 2020 年全年每 10 分钟记录一次。该数据集提供了包括空气温度、湿度、风向、辐射和降水在内的全面大气测量。每个变量超过 52,560 个数据点（365 天×24 小时×每小时 6 次测量），这种高频率采样为天气模式和大气条件提供了详细的见解。测量包括基本气象参数和派生量，如水汽压亏缺和潜在温度，使其适用于气象研究和实际应用。此数据来源于Kaggle，由马克斯·普朗克研究所提供。链接：https://www.kaggle.com/datasets/alistairking/weather-long-term-time-series-forecasting

## (二) 数据集描述

数据集以 CSV 格式提供，包含21列×52697行，列名及描述见表1。

|  |  |
| --- | --- |
| 列名 | 描述 |
| date | 观测日期和时间。 |
| p | 大气压力（毫巴）。 |
| T | 气温，摄氏度（°C）。 |
| Tpot | 位温（开尔文 K），表示将一个气团移动到标准气压水平时气团所具有的温度。 |
| Tdew | 露点温度，单位为摄氏度（°C），表示空气达到饱和湿度的温度。 |
| rh | 相对湿度（%），表示空气中所含水分与该温度下空气最大含水量的比值。 |
| VPmax | 最大蒸汽压（毫巴，mbar），表示在给定温度下水蒸气所施加的最大压力。 |
| VPact | 实际水汽压（毫巴），表示空气中的当前水汽压。 |
| VPdef | 蒸汽压亏缺（毫巴），衡量最大蒸汽压与实际蒸汽压之间的差异，用于评估干燥潜力。 |
| sh | 相对湿度，单位为克每千克（g/kg），表示每千克空气中的水蒸气质量。 |
| H20C | 水蒸气浓度，以每摩尔干空气的毫摩尔数（mmol/mol）表示。 |
| rho | 空气密度，单位为每立方米克（g/m³），反映单位体积内空气的质量。 |
| wv | 风速，以米每秒（m/s）为单位，测量空气的水平运动。 |
| max.wv | 最大风速（米/秒），表示该时段内记录的最高风速。 |
| wd | 风向（°），表示风从哪个方向吹来。 |
| rain | 总降水量（毫米），表示观测期间降水量。 |
| raining | 降雨持续时间（秒），记录观测期间降雨发生的时间。 |
| SWDR | 短波向下辐射，单位为每平方米瓦特（W/m²），测量入射太阳辐射。 |
| PAR | 光合有效辐射，单位为每平方米每秒微摩尔（µmol/m²/s），表示可用于光合作用的光量。 |
| max.PAR | 观测期间记录的最大光合有效辐射量，单位为µmol/m²/s。 |
| Tlog | 温度记录为摄氏度（°C），可能来自次级传感器或记录器。 |

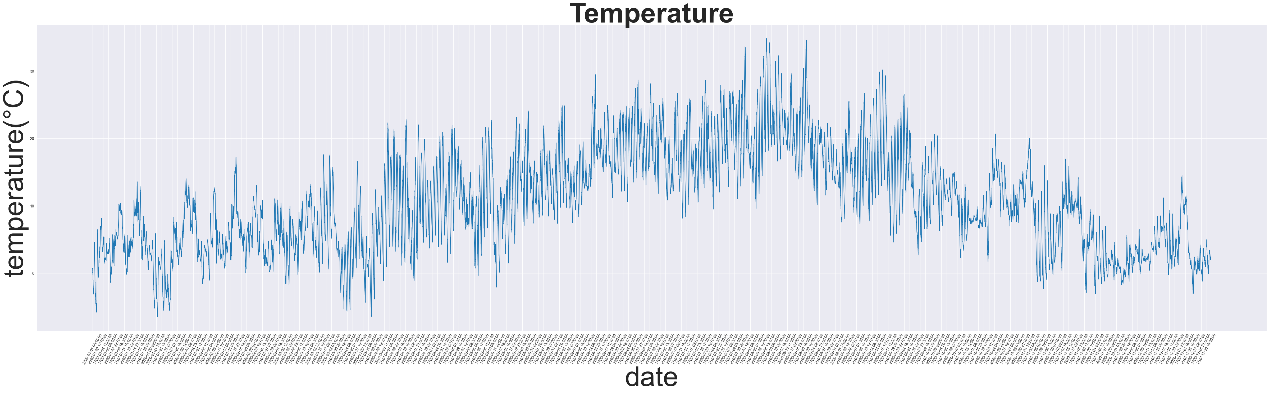
**表1 数据集的描述**

## (二) 总样本区间及训练集、测试集划分

本文选取date列作为时间序列，样本特征的选取可以选择任意列，这里本文使用T（温度）列作为样本特征，整体数据集的前80%作为训练集，后20%作为测试集。

# 四、结果分析

利用matplotlib绘制原始气温时序图如下图2所示。

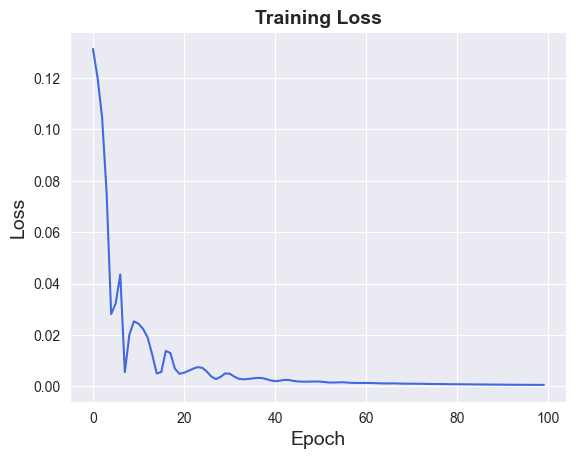


**图2 原始气温时序图**

经过模型100轮训练后，损失值与训练时间如下表2,图3所示。

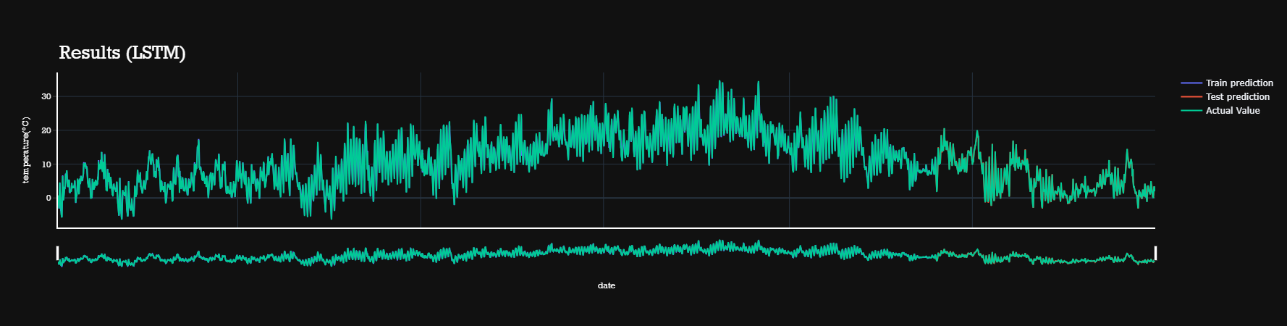
|  |  |  |
| --- | --- | --- |
| 训练轮次(Epoch) | 损失值(MSE) | 训练时间 |
| 0 | 0.13136909902095795 | 200.8978180885315 |
| 1 | 0.1203685775399208 |
| 2 | 0.10493992269039154 |
| 3 | 0.07536657899618149 |
| … | … |
| 99 | 0.000554429367184639 |

**表2 100轮次的损失值及训练时间表**

****

**图3 100轮次的测试损失图**

将模型预测的结果利用plotly可视化并与真实值进行比较，如下图4所示。

****

**图4 LSTM预测结果图**

放大图像如图5所示，即可看到模型预测具有很好的效果。



**图5 LSTM预测结果图的局部放大**

# 五、结论

天气数据是受多种因素影响的时间序列，构建精确的天气数据预测模型比较困难。本文建立了长短时记忆网络模型(LSTM)的气温预测模型，实现了对温度时间序列的预测分析。该模型可以更好地对温度进行预测分析，研究结果具有较好的实用性。由于天气具有随机性和不确定性的特点，在进一步的研究中可以选择更多的气象影响因素进行预测，进而更好地对天气数据进行分析，进一步提高不同天气数据的预测精度。

# 六、代码报告

|  |
| --- |
| import numpy as np  import pandas as pd  import matplotlib.pyplot as plt  import seaborn as sns  filepath = r'C:\Users\hp\Desktop\cleaned\_weather.csv'  data = pd.read\_csv(filepath)  data = data.sort\_values('date')  print(data.head())  print(data.shape)  sns.set\_style("darkgrid")  plt.figure(figsize=(60, 15))  plt.plot(data[['T']])  plt.xticks(range(0, data.shape[0], 250), data['date'].loc[::250], rotation=60)  plt.title("Temperature", fontsize=75, fontweight='bold')  plt.xlabel('date', fontsize=75)  plt.ylabel('temperature(°C)', fontsize=75)  plt.show()  # 选取温度作为特征  Temperature = data[['T']]# 从数据集中选取温度（T）列作为特征  print(Temperature.info())# 打印选取特征后的数据框信息  from sklearn.preprocessing import MinMaxScaler # 从sklearn库中导入MinMaxScaler进行数据缩放  scaler = MinMaxScaler(feature\_range=(-1, 1)) # 创建MinMaxScaler对象，设置缩放范围为-1到1  Temperature['T'] = scaler.fit\_transform(Temperature['T'].values.reshape(-1, 1))  # 对温度数据进行缩放，并将结果更新到Temperature数据框中  # 先将温度数据转换为二维数组，再进行缩放  print(Temperature['T'].shape) # 打印缩放后的数据形状  # 此时刻的温度预测下一时刻的温度  # lookback表示观察的跨度  from sklearn.model\_selection import train\_test\_split  def split\_data(stock, lookback):  data\_raw = stock.to\_numpy() # 将数据框转换为NumPy数组  data = [] # 初始化存储切分数据的列表  for index in range(len(data\_raw) - lookback):  data.append(data\_raw[index: index + lookback]) # 遍历数据，生成长度为lookback的滑动窗口序列并添加到data列表中  data = np.array(data) # 将列表转换为NumPy数组  test\_set\_size = int(np.round(0.2 \* data.shape[0])) # 计算测试集大小，占总数据的20%  train\_set\_size = data.shape[0] - (test\_set\_size) # 计算训练集大小，占总数据的80%  x\_train = data[:train\_set\_size, :-1, :] # 提取训练集的输入特征（不包括最后一个数据点）  y\_train = data[:train\_set\_size, -1, :] # 提取训练集的目标值（包括最后一个数据点）  x\_test = data[train\_set\_size:, :-1] # 提取测试集的输入特征（不包括最后一个数据点）  y\_test = data[train\_set\_size:, -1, :] # 提取测试集的目标值（包括最后一个数据点）  return [x\_train, y\_train, x\_test, y\_test] # 返回训练集和测试集的特征和目标值  lookback = 20 # 定义lookback的跨度为20个时刻  x\_train, y\_train, x\_test, y\_test = split\_data(Temperature, lookback) # 调用split\_data函数进行数据切分  print('x\_train.shape = ', x\_train.shape) # 打印训练集特征的形状  print('y\_train.shape = ', y\_train.shape) # 打印训练集目标的形状  print('x\_test.shape = ', x\_test.shape) # 打印测试集特征的形状  print('y\_test.shape = ', y\_test.shape) # 打印测试集目标的形状  import torch # 导入PyTorch库  import torch.nn as nn # 导入PyTorch的神经网络模块  # 将NumPy数组转换为PyTorch张量，并指定数据类型为Tensor  x\_train = torch.from\_numpy(x\_train).type(torch.Tensor)  x\_test = torch.from\_numpy(x\_test).type(torch.Tensor)  y\_train\_lstm = torch.from\_numpy(y\_train).type(torch.Tensor)  y\_test\_lstm = torch.from\_numpy(y\_test).type(torch.Tensor)  y\_train\_gru = torch.from\_numpy(y\_train).type(torch.Tensor)  y\_test\_gru = torch.from\_numpy(y\_test).type(torch.Tensor)  # 输入的维度为1，只有Temperature温度  input\_dim = 1  # 隐藏层特征的维度  hidden\_dim = 32  # 循环的层数  num\_layers = 2  # 预测后下一时刻的温度  output\_dim = 1  # 训练的轮数  num\_epochs = 100  # 定义LSTM模型  class LSTM(nn.Module):  def \_\_init\_\_(self, input\_dim, hidden\_dim, num\_layers, output\_dim):  super(LSTM, self).\_\_init\_\_()  self.hidden\_dim = hidden\_dim # 隐藏层特征的维度  self.num\_layers = num\_layers # LSTM的层数  self.lstm = nn.LSTM(input\_dim, hidden\_dim, num\_layers, batch\_first=True)  # 定义LSTM层，输入维度、隐藏层维度、层数以及batch\_first参数  self.fc = nn.Linear(hidden\_dim, output\_dim)  # 定义全连接层，将隐藏层输出映射到输出维度  def forward(self, x):  h0 = torch.zeros(self.num\_layers, x.size(0), self.hidden\_dim).requires\_grad\_()  # 初始化隐藏状态h0，形状为（层数，批次大小，隐藏层维度）  c0 = torch.zeros(self.num\_layers, x.size(0), self.hidden\_dim).requires\_grad\_()  # 初始化细胞状态c0，形状同上  out, (hn, cn) = self.lstm(x, (h0.detach(), c0.detach()))  # 前向传播，通过LSTM层，输出为out，最后的隐藏状态和细胞状态为hn和cn  out = self.fc(out[:, -1, :]) # 取LSTM的最后一个时间步的输出并通过全连接层  return out # 返回输出  # 创建LSTM模型实例  model = LSTM(input\_dim=input\_dim, hidden\_dim=hidden\_dim, output\_dim=output\_dim, num\_layers=num\_layers)  criterion = torch.nn.MSELoss() # 定义损失函数为均方误差损失  optimiser = torch.optim.Adam(model.parameters(), lr=0.01)  # 定义优化器为Adam，并设置学习率为0.01  import time # 导入时间模块，用于计算训练时间  hist = np.zeros(num\_epochs) # 初始化一个数组，用于存储每个epoch的损失值  start\_time = time.time() # 记录训练开始时间  lstm = [] # 初始化一个列表，用于存储LSTM模型的输出（暂时未使用）  for t in range(num\_epochs): # 开始训练循环，共进行num\_epochs次训练  y\_train\_pred = model(x\_train) # 前向传播，计算训练集的预测值  loss = criterion(y\_train\_pred, y\_train\_lstm)  # 计算损失函数值，比较预测值与真实值的均方误差  print("Epoch ", t, "MSE: ", loss.item())  # 打印当前epoch的损失值  hist[t] = loss.item() # 将损失值存储到hist数组中  optimiser.zero\_grad() # 清零梯度，以防止梯度累积  loss.backward() # 反向传播，计算梯度  optimiser.step() # 更新模型参数  training\_time = time.time() - start\_time  # 计算训练所用的总时间  print("Training time: {}".format(training\_time))  # 打印训练时间  # 将预测值和真实值进行逆缩放，转换回原始的温度  predict = pd.DataFrame(scaler.inverse\_transform(y\_train\_pred.detach().numpy()))  original = pd.DataFrame(scaler.inverse\_transform(y\_train\_lstm.detach().numpy()))  import seaborn as sns # 导入Seaborn库，用于绘图  sns.set\_style("darkgrid") # 设置绘图风格为darkgrid  fig = plt.figure() # 创建一个新的图形对象  ax = sns.lineplot(x=original.index, y=original[0], label="Data", color='royalblue')  # 绘制原始收盘价曲线  ax = sns.lineplot(x=predict.index, y=predict[0], label="Training Prediction (LSTM)", color='tomato')  # 绘制LSTM模型的训练预测曲线  ax.set\_title('Temperature', size=75, fontweight='bold')  # 设置第一个子图的标题  ax.set\_xlabel("Days", size=75) # 设置x轴标签  ax.set\_ylabel("temperature(°C)", size=75) # 设置y轴标签  ax.set\_xticklabels('', size=75) # 设置x轴刻度标签大小  fig.set\_figheight(15) # 设置图形高度  fig.set\_figwidth(60) # 设置图形宽度  plt.show()  ax = sns.lineplot(data=hist, color='royalblue')  # 绘制训练损失值曲线  ax.set\_xlabel("Epoch", size=14) # 设置x轴标签  ax.set\_ylabel("Loss", size=14) # 设置y轴标签  ax.set\_title("Training Loss", size=14, fontweight='bold')  # 设置第二个子图的标题  fig.set\_figheight(6) # 设置图形高度  fig.set\_figwidth(16) # 设置图形宽度  plt.show() # 显示图形  import math, time # 导入数学和时间模块  from sklearn.metrics import mean\_squared\_error, r2\_score  # 导入均方误差和R2评分函数  # make predictions  y\_test\_pred = model(x\_test) # 使用模型对测试集进行预测  # invert predictions  y\_train\_pred = scaler.inverse\_transform(y\_train\_pred.detach().numpy())  # 对训练集的预测结果进行逆缩放  y\_train = scaler.inverse\_transform(y\_train\_lstm.detach().numpy())  # 对训练集的真实值进行逆缩放  y\_test\_pred = scaler.inverse\_transform(y\_test\_pred.detach().numpy())  # 对测试集的预测结果进行逆缩放  y\_test = scaler.inverse\_transform(y\_test\_lstm.detach().numpy())  # 对测试集的真实值进行逆缩放  # calculate root mean squared error  trainScore = math.sqrt(mean\_squared\_error(y\_train[:, 0], y\_train\_pred[:, 0]))  # 计算训练集的均方根误差  print('Train Score: %.2f RMSE' % (trainScore))  testScore = math.sqrt(mean\_squared\_error(y\_test[:, 0], y\_test\_pred[:, 0]))  # 计算测试集的均方根误差  print('Test Score: %.2f RMSE' % (testScore))  trainr2Score = r2\_score(y\_train[:, 0], y\_train\_pred[:, 0])  # 计算训练集的R2评分  print('Train Score: %.2f R2' % (trainr2Score))  testr2Score = r2\_score(y\_test[:, 0], y\_test\_pred[:, 0])  # 计算测试集的R2评分  print('Test Score: %.2f R2' % (testr2Score))  lstm.append(trainScore) # 将训练集RMSE添加到lstm列表  lstm.append(testScore) # 将测试集RMSE添加到lstm列表  lstm.append(training\_time) # 将训练时间添加到lstm列表  # shift train predictions for plotting  trainPredictPlot = np.empty\_like(Temperature) # 创建与Temperature形状相同的空数组  trainPredictPlot[:, :] = np.nan # 将所有值初始化为NaN  trainPredictPlot[lookback:len(y\_train\_pred) + lookback, :] = y\_train\_pred  # 将训练集预测值放入对应的位置  # shift test predictions for plotting  testPredictPlot = np.empty\_like(Temperature) # 创建与Temperature形状相同的空数组  testPredictPlot[:, :] = np.nan # 将所有值初始化为NaN  testPredictPlot[len(y\_train\_pred) + lookback - 1:len(Temperature) - 1, :] = y\_test\_pred  # 将测试集预测值放入对应的位置  original = scaler.inverse\_transform(Temperature['T'].values.reshape(-1, 1))  # 对原始的温度数据进行逆缩放  # Combine predictions for plotting  predictions = np.append(trainPredictPlot, testPredictPlot, axis=1)  # 将训练集和测试集的预测值合并  predictions = np.append(predictions, original, axis=1)  # 将原始的温度数据添加到预测值中  result = pd.DataFrame(predictions) # 将结果转换为DataFrame  import plotly.express as px  import plotly.graph\_objects as go  fig = go.Figure()  fig.add\_trace(go.Scatter(go.Scatter(x=data['date'], y=result[0],  mode='lines',  name='Train prediction')))  fig.add\_trace(go.Scatter(x=data['date'], y=result[1],  mode='lines',  name='Test prediction'))  fig.add\_trace(go.Scatter(go.Scatter(x=data['date'], y=result[2],  mode='lines',  name='Actual Value')))  fig.update\_layout(  xaxis=dict(  title\_text='date',  titlefont=dict(  family='Rockwell',  size=12,  color='white',),  showline=True,  showgrid=True,  showticklabels=False,  linecolor='white',  linewidth=2  ),  yaxis=dict(  title\_text='temperature(°C)',  titlefont=dict(  family='Rockwell',  size=12,  color='white',  ),  showline=True,  showgrid=True,  showticklabels=True,  linecolor='white',  linewidth=2,  ticks='outside',  tickfont=dict(  family='Rockwell',  size=12,  color='white',  ),  ),  showlegend=True,  template = 'plotly\_dark'  )  annotations = []  annotations.append(dict(xref='paper', yref='paper', x=0.0, y=1.05,  xanchor='left', yanchor='bottom',  text='Results (LSTM)',  font=dict(family='Rockwell',  size=26,  color='white'),  showarrow=False))  fig.update\_layout(annotations=annotations)  fig.update\_xaxes(rangeslider\_visible=True)#添加时间拉条  fig.show() |

源代码及文件已上传至Github: https://github.com/zhangmiao0616/Weather-Data-Time-Series-Prediction-Based-on-LSTM

# 参考文献

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

[1]杨青,王晨蔚.基于深度学习LSTM神经网络的全球股票指数预测研究[J].统计研究,2019,36(03):65-77.DOI:10.19343/j.cnki.11-1302/c.2019.03.006.

[2]张美英,何杰.时间序列预测模型研究综述[J].数学的实践与认识,2011,41(18):189-195.

[3]武双新.基于LSTM的气温数据建模研究[J].数据通信,2021,(02):47-51.

[4]王源昊.基于ARIMA模型和LSTM神经网络的全球气温预测分析[J].科学技术创新,2021,(35):166-170.

[5]杨丽,吴雨茜,王俊丽,等.循环神经网络研究综述[J].计算机应用,2018,38(S2):1-6+26.

[6]王鑫,吴际,刘超,等.基于LSTM循环神经网络的故障时间序列预测[J].北京航空航天大学学报,2018,44(04):772-784.DOI:10.13700/j.bh.1001-5965.2017.0285.