基于深度学习模型的血糖预测探究与分析

Exploration and Analysis of Blood Sugar Prediction Based on Deep Learning Models

团队成员： 2151417 段瑞源 2152057 杨瑞华
2152955 张尧 2153393 胡峻玮

Contents

PART 01

项目概述

- 项目背景
- 数据集介绍
- 解决方案

项目概述

——项目背景

■ 血糖管理的重要性

- 血糖管理对于糖尿病患者至关重要，不仅影响日常生活质量，还关系到长期的健康结果。
- 稳定的血糖水平可以预防糖尿病的并发症，如视网膜病变、肾病和心血管疾病。

■ 现有血糖监测方法的局限

- 传统的血糖监测方法需要患者定时进行血液测试，这种方法侵入性强、不便捷。
- 虽然连续血糖监测系统（CGMS）提高了监测便捷性，但成本较高且需要定期更换传感器。

■ 利用人工智能进行血糖预测的潜力

- 人工智能，尤其是深度学习模型，为非侵入性、实时的血糖预测提供了可能性。
- 利用深度学习模型，可以通过分析病人的生活习惯、饮食、体育活动等多种因素来预测血糖水平，从而实现更个性化的血糖管理。

项目概述

——数据集介绍

■ **数据集概述**

- 数据集包括12名1型糖尿病（T1DM）患者和100名2型糖尿病（T2DM）患者的数据。

■ **文件结构和数据内容**

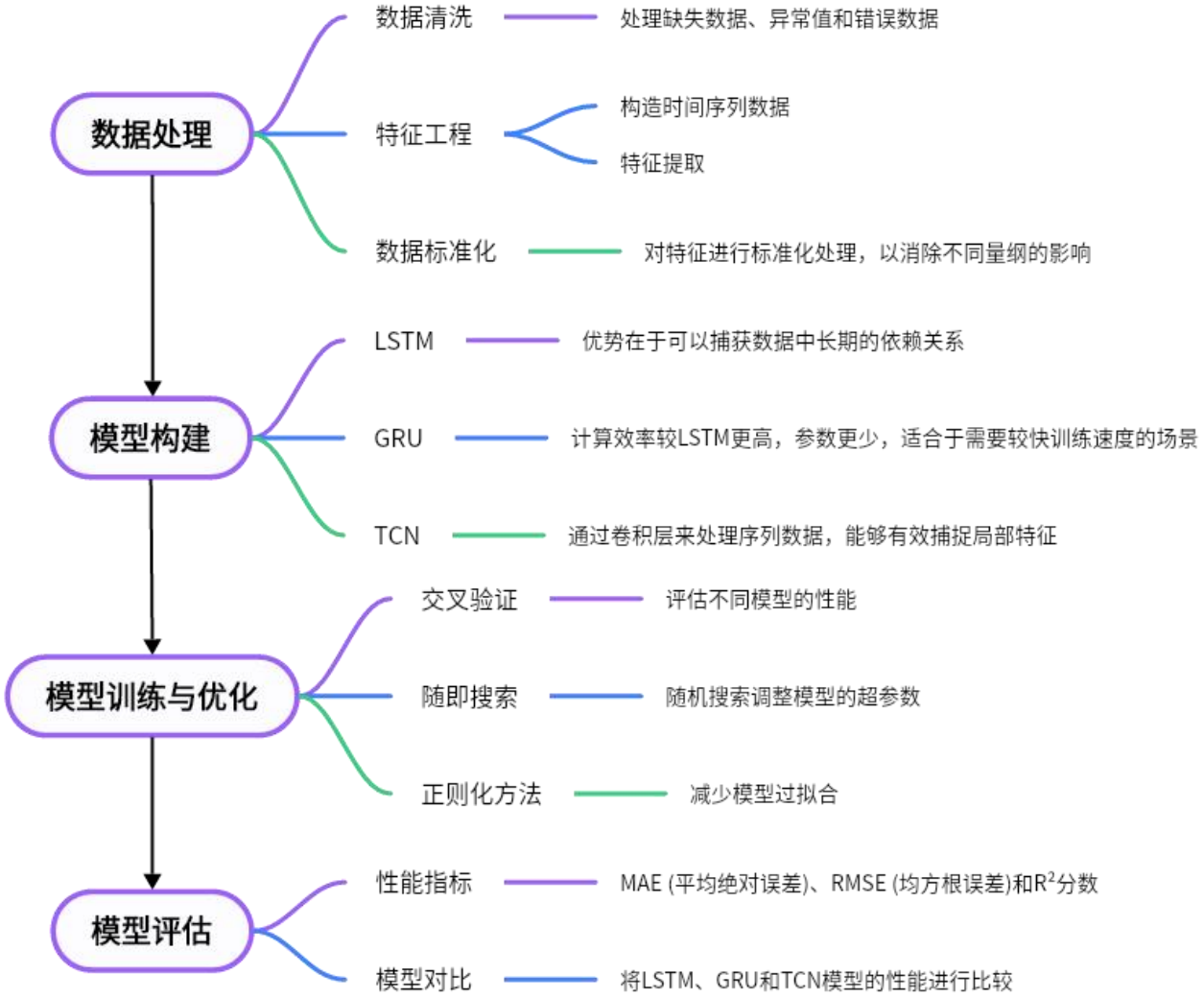
- 文件夹: 数据集被分为两个主要文件夹“Shanghai_T1DM”和“Shanghai_T2DM”，其中包含了患者连续几天（3至14天）的连续血糖监测（CGM）数据。
- 摘要表: “Shanghai_T1DM_Summary”和“Shanghai_T2DM_Summary”两个摘要表格，总结了研究中包含的患者的临床特征、实验室测量值和药物信息。

■ **数据字段详解**

- 日期（Date）: 记录CGM数据的时间。
- 连续血糖监测数据（CGM）: 每15分钟记录一次的血糖数据。
- 毛细血管血糖水平（CBG）: 通过血糖仪测量的血糖水平。
- 血酮（Blood ketone）: 当疑似糖尿病酮症酸中毒（DKA）时测量的血酮值。
- 饮食摄入（Dietary intake）: 患者自报的时间和食物摄入量。
- 皮下胰岛素剂量（Insulin dose-s.c.）: 通过胰岛素笔进行的皮下注射。
- 静脉注射胰岛素剂量（Insulin dose-i.v.）: 静脉注射的胰岛素剂量。
- 非胰岛素降糖药（Non-insulin hypoglycemic agents）: 除胰岛素外的降糖药。
- CSII-餐前胰岛素（CSII-bolus insulin）: 通过胰岛素泵在餐前输送的胰岛素剂量。
- CSII-基础胰岛素（CSII-basal insulin）: 通过胰岛素泵持续输送的基础胰岛素速率。

项目概述

——解决方案



- **数据处理：**
  - 此阶段涵盖了项目开始时的数据处理工作，包括数据的收集、清洗和准备等步骤。
- **模型构建：**
  - 在数据处理后，进入模型构建阶段，这里可能包括选择合适的算法、建立模型框架、训练模型等。
- **模型调优与优化：**
  - 继模型构建之后，对模型进行调优和优化以提高其性能，可能包括调整参数、使用不同的训练方法等。
- **模型评估：**
  - 最后，通过一系列的测试和评估标准来验证模型的有效性和准确性，确保模型在实际应用中的可靠性。

PART 02

数据处理与模型构建

- 数据集处理
- 模型选择与构建
- 难点解决

数据处理与模型构建

——数据处理

一、数据处理函数

- 创建特征和目标数据集：定义createXY函数，该函数基于历史数据（npast）和未来数据（n_future）时间步长从数据集中创建特征（X）和目标（Y）数据。
- 处理文件：定义processFiles和processSummary函数，该函数遍历所有数据文件，并应用createXY函数生成所有的特征和目标数据集。

二、特征数据集构建

- 使用过去6小时的数据输入和预测未来1小时的数据来设置时间步长，处理文件并创建相应的特征和目标数据集。

三、数据集转换和特征工程

- 数据转换为DataFrame：将数据转换为DataFrame格式以便进一步处理。
- 删除无关属性：移除数据中不相关的属性，如患者编号。
- 时间特征提取：将日期转换为分钟，并使用正余弦函数将分钟转换为周期性特征。
- 数据类型转换和缺失值处理：将数据类型统一转换为float，并处理数据中的缺失值。

四、数据归一化

五、特征重要性评估

- 使用随机森林和XGBoost对处理后的特征进行重要性分析，选择最重要的特征进行模型训练。

70%数据为训练集 30%数据为验证集

数据处理与模型构建

——模型选择

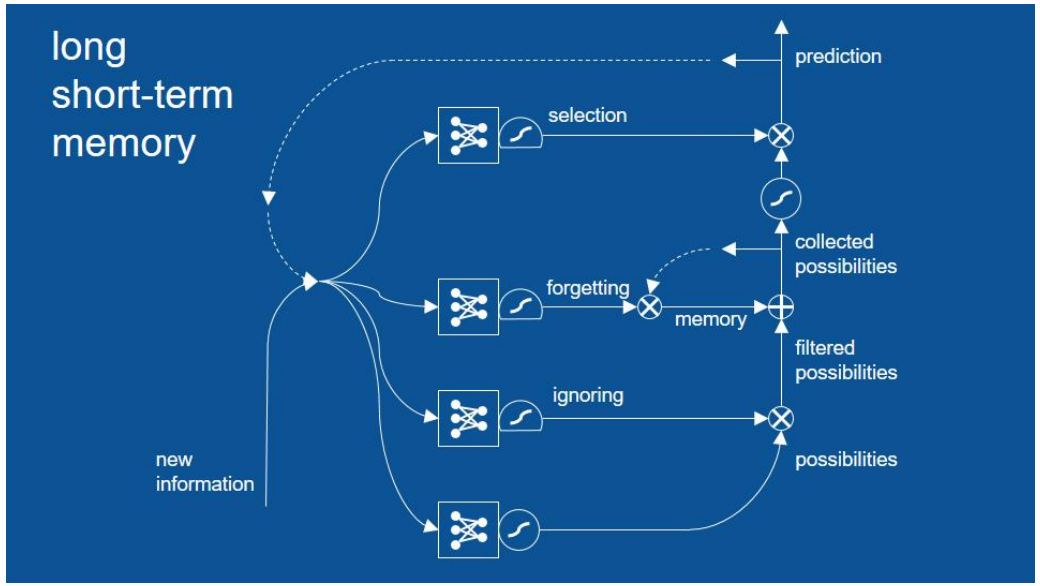
- **LSTM（长短期记忆网络）**
- 长期依赖问题的解决：LSTM 能够解决传统 RNN 面临的长期依赖问题，这是因为它的特殊结构使得信息能够在网络中持续流动而不被忘记。
- 记忆能力：LSTM的“门控单元”设计可以精细控制信息的保存和遗忘，非常适合需要记忆和利用历史信息进行预测的场景，如时间序列数据分析。
- **GRU（门控循环单元）**
- 简化的结构：GRU 结构比 LSTM 更简单，只包含两个门（更新门和重置门），这使得它在训练过程中计算效率更高。
- 灵活性和效率：GRU 在很多情况下可以达到与 LSTM 相似的性能，但参数更少、训练更快，对于需要快速实验的项目来说，GRU 是一个更经济的选择。
- **TCN（时间卷积网络）**
- 并行计算能力：与基于循环的网络结构不同，TCN 的卷积结构可以并行处理数据，这提高了模型的训练速度。
- 长距离的历史信息捕获：通过膨胀卷积，TCN 能够在不显著增加计算复杂度的前提下，扩大其感知域，捕捉长期的历史信息，这对于复杂的时间序列预测尤为重要。
- 因果关系保持：TCN 通过因果卷积确保模型在预测时仅依赖于先前的输入，适合时间序列数据的特性。

数据处理与模型构建

——模型构建

1. LSTM（长短期记忆网络）

- 结构与原理：LSTM 是一种特殊的递归神经网络（RNN），优于传统的 RNN，因为它解决了长期依赖问题。它通过一系列互联的“细胞”构成，每个细胞含有三个关键的门控单元——输入门、遗忘门和输出门，这些门控单元决定了是否保留、更新或输出某些信息。
- 实验步骤：构建 LSTM 网络，选择合适的隐状态维度和门控单元参数。
- 遇到的挑战：
  - 计算复杂度高：由于 LSTM 结构复杂，需要较长的训练时间和更多的计算资源。
  - 参数调整困难：LSTM 模型涉及多个超参数，调参过程复杂，需要大量实验来确定最佳参数。

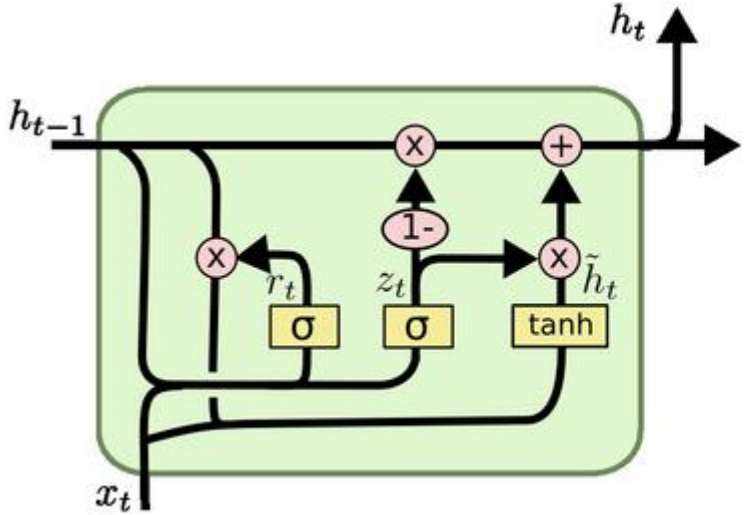


数据处理与模型构建

——模型构建

2. GRU（门控循环单元）

- 结构与原理：GRU 是 LSTM 的一个变种，具有更简洁的结构（只有两个门：重置门和更新门）。它旨在解决传统 RNN 在处理长序列数据时的梯度消失和爆炸问题。
- 实验步骤：设计 GRU 网络，设置适当的门控单元以控制信息的流动。
- 遇到的挑战：与 LSTM 类似，GRU 在训练大规模数据时也会面临计算资源的挑战。



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

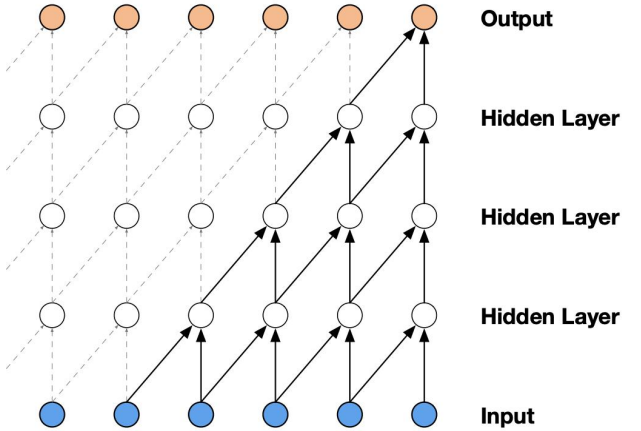
$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

数据处理与模型构建

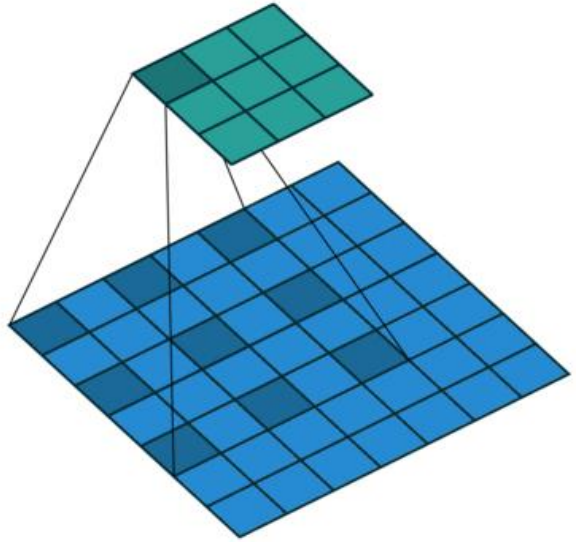
——模型构建

3. TCN（时间卷积网络）

- 结构与原理：TCN 是一种基于卷积神经网络的架构，用于处理序列数据。它的主要优点包括并行化计算、灵活的接收域和长期依赖处理能力。TCN 通过因果卷积和膨胀卷积确保在预测当前时刻的值时只使用当前时刻及之前的数据。
- 实验步骤：构建 TCN 模型，设置适当的卷积层和膨胀率以捕捉长距离的历史信息。
- 遇到的挑战：设计一个高效的网络结构以适应时间序列数据的特点，并在保持模型因果性的同时捕获长期依赖。



因果卷积



膨胀卷积

PART 03

难点分析与解决

- 数据集处理
- 模型选择与构建
- 难点解决

难点分析与解决

难点一：模型参数选取

问题描述：LSTM 和 GRU 模型涉及多个超参数（如层数、隐藏单元数、学习率等），这些参数的选择对模型性能有显著影响，但往往难以找到最优配置。

解决方法：

自动化超参数调优：使用网格搜索、随机搜索或更高级的算法（如贝叶斯优化）来自动寻找最优的超参数设置。

```
# 开始随机搜索
for params in param_list:
    num_epochs = params['num_epochs']
    initial_learning_rate = params['initial_learning_rate']
    batch_size = params['batch_size']
    step_size = params['step_size']
    gamma = params['gamma']
    hidden_size = params['hidden_size']
    num_layers = params['num_layers']

    print(f'Training with params: {params}')

    # 准备数据
    train_dataset = TensorDataset(X_train_t, y_train_t)
    train_loader = DataLoader(dataset=train_dataset, batch_size=batch_size, shuffle=True, pin_memory=False, num_workers=0)

    # 定义模型、优化器和学习率调度器
    lstm_model = LSTM(input_size, hidden_size, num_layers, output_size).to(device)
    optimizer = torch.optim.Adam(lstm_model.parameters(), lr=initial_learning_rate)
    scheduler = torch.optim.lr_scheduler.StepLR(optimizer, step_size=step_size, gamma=gamma)

    # 训练循环
    for epoch in range(num_epochs):
        lstm_model.train()
        for x_batch, y_batch in train_loader:
            x_batch = x_batch.to(device, non_blocking=True)
            y_batch = y_batch.to(device, non_blocking=True)
            outputs = lstm_model(x_batch)
            optimizer.zero_grad()
            loss = criterion(outputs, y_batch)
            loss.backward()
            optimizer.step()

        # 每个epoch结束后更新学习率
        scheduler.step()

    # 记录最优模型
    final_loss = loss.item()
```

难点分析与解决

难点二：长期依赖问题

问题描述：尽管 LSTM 设计用来处理长期依赖问题，但在实际应用中仍可能因时间跨度增大而难以捕捉到远期的依赖关系。

解决方法：

使用 TCN：TCN 通过其特殊的膨胀卷积设计，能够有效捕获长期依赖，为处理长序列数据提供了另一种有效的解决方案。

序列分割：将长序列数据分割成较短的片段进行训练，有助于缓解因序列长度过长引起的问题。

解决方案一：

```
num_epochs = 200
learning_rate = 0.00005
batch_size = 128 # 32
criterion = nn.MSELoss()
```

解决方案二：

```
# 使用TCN进行4步长的预测
from torch import nn
from pytorch_tcn import TCN

class TCNModel(nn.Module):
    def __init__(self,_input_size,_output_size,num_steps):
        super(TCNModel,self).__init__()
        self.num_steps = num_steps
        self.tcn=TCN(
            num_inputs=_input_size,
            kernel_size=4,
            dropout=0.1,
            num_channels=[16,32,64],
            use_skip_connections=True,
            causal=True,
            dilation_reset = 16,
        )
        self.fc=nn.Linear(6,_output_size)
    def forward(self,x):
        out=self.tcn(x)
        out=self.fc(out[:,-1,:])
        return out
```

难点分析与解决

难点三：梯度消失和梯度爆炸

问题描述：尤其是在深层网络或长序列数据中，传统的 RNN 及其变体如 LSTM 和 GRU 可能会遇到梯度消失或爆炸的问题。

解决方法：

梯度裁剪：在训练过程中对梯度进行裁剪，限制梯度的最大值，防止梯度爆炸。

使用门控机制：GRU 和 LSTM 的门控机制本身就是为了解决梯度消失问题，确保模型中的关键信息可以长时间保留。

```
for epoch in range(num_epochs):
    lstm_model.train()
    for x_batch, y_batch in train_loader:
        x_batch = x_batch.to(device,non_blocking=True)
        y_batch = y_batch.to(device,non_blocking=True)
        outputs = lstm_model(x_batch)
        optimizer.zero_grad()
        loss = criterion(outputs, y_batch)
        loss.backward()
        # 梯度裁剪
        torch.nn.utils.clip_grad_norm_(lstm_model.parameters(), max_norm=1.0)
        optimizer.step()

    if (epoch + 1) % 2 == 0:
        print(f'Epoch [{epoch + 1}/{num_epochs}], Loss: {loss.item():.4f}')
```

PART 04

结果分析与比较

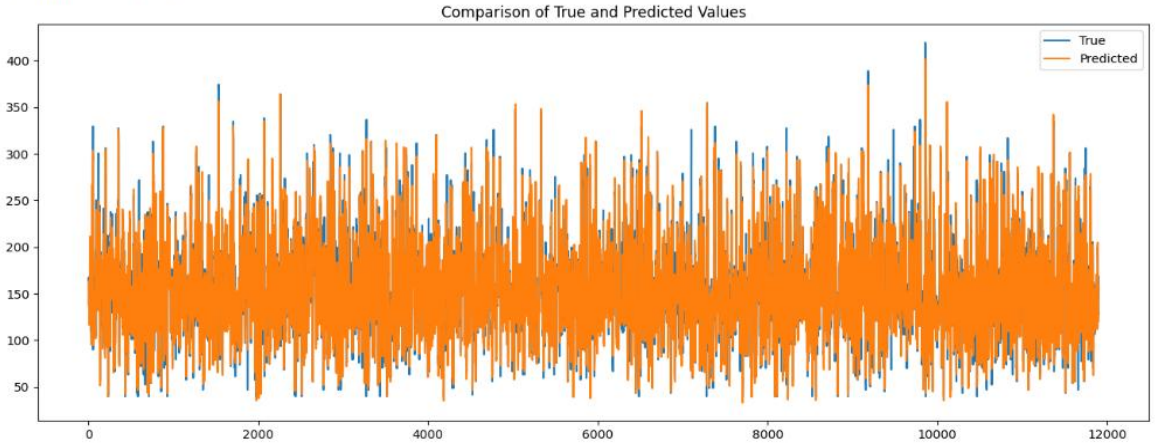
- 三种模型结果
- 三种模型评价指标
- 横向对比

结果分析与比较

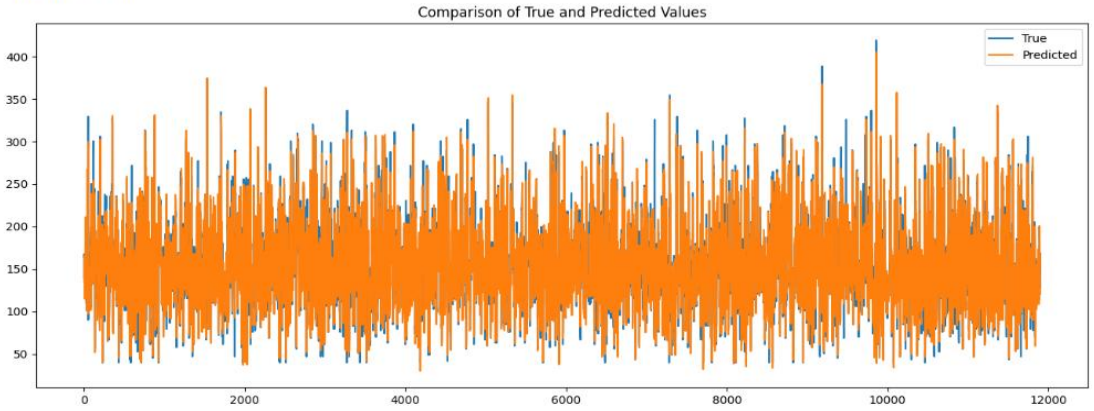
——模型结果与评价指标

三个模型预测效果：

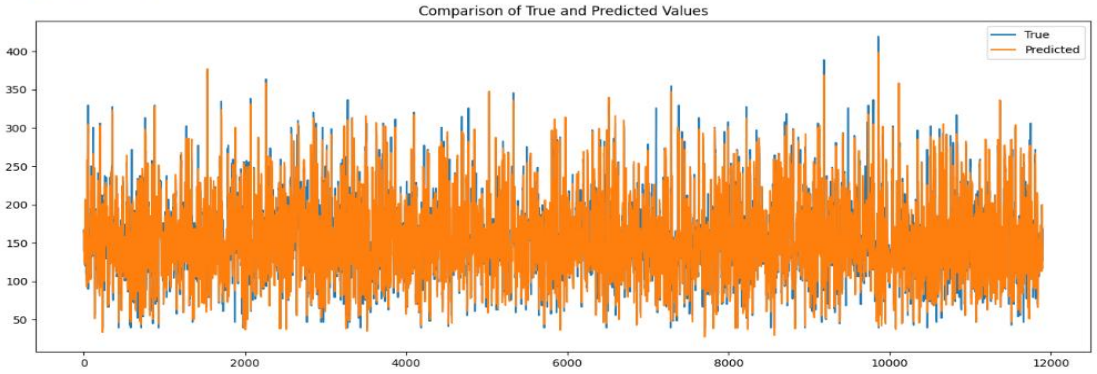
Test loss: 120.2895
LSTM MAE: 7.4900



Test loss: 119.3730
GRU MAE: 7.4805



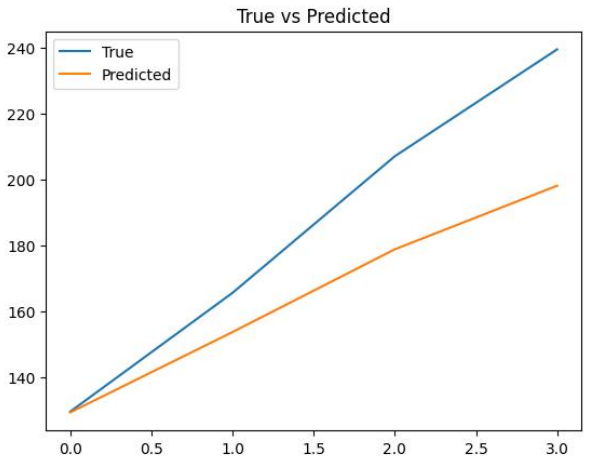
Test loss: 137.7269
TCN MAE: 7.9930



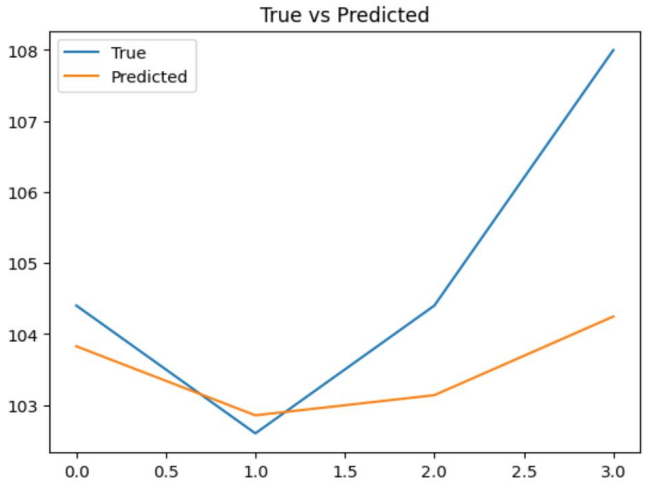
结果分析与比较

——模型结果与评价指标

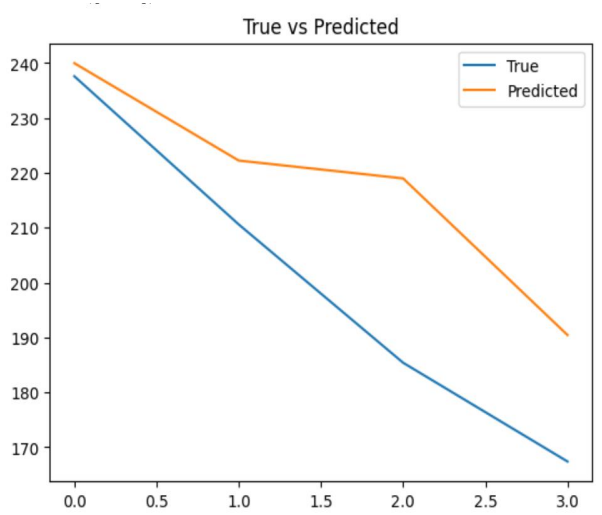
LSTM



GRU



TCN



结果分析与比较

——模型对比与总结

1. 模型表现概览

GRU 模型在这次比较中表现最好，其测试损失为119.3730，MAE为7.4805。

LSTM 模型的表现紧随其后，测试损失为120.2895，MAE为7.4900。

TCN 模型在这组数据上的表现稍逊，测试损失为137.7269，MAE为7.9930。

2. 详细分析

GRU vs. LSTM: GRU和LSTM的表现非常接近，但GRU的参数少于LSTM，这使得GRU在训练时更快且需求计算资源较少。这在有限的数据集上可能使GRU略有优势，因为它可以更快地收敛且不太过拟合。

TCN的独特之处: TCN使用卷积层来处理序列数据，这使得它在某些情况下比循环神经网络更有效，特别是在可以并行计算的情况下。然而，TCN在这个比较中表现较差可能是因为其在处理非常长的序列依赖关系时不如GRU和LSTM有效，或者模型架构需要进一步调整以适应此类数据。

3. 应用分析

任务适应性: 如果任务涉及需要捕获长期依赖关系的复杂时间序列预测，LSTM可能更为合适。但如果关注模型的训练效率和运行速度，GRU可能是更好的选择。

数据和任务类型: 对于具有显著季节性或周期性的数据，调整TCN的结构可能会获得更好的结果。同时，考虑到TCN的并行处理能力，对于大规模数据集，TCN也许能展现出更好的性能。

Thanks!

Team 07