机器学习实验报告

张顾潇 20244227046

2025.1.13

一、 问题介绍

共享单车系统革新了传统的自行车租赁服务，实现了从用户注册、租赁自行车到归还的全 流程高度自动化。借助这一系统，用户可以方便地在指定站点租借自行车，并在任意站点归还。截至目前，全球已有超过 500 个共享单车项目，共计投放了 50 多万辆自行车。由于共享单车在缓解交通拥堵、改善环境以及促进公众健康等方面发挥了重要作用，越来越受到社会各界的关注和欢迎。

1.1 背景和动机

对共享单车租赁数量进行预测可以帮助运营商合理规划和优化车辆调度，提升运营效率和服务质量。 而共享单车租赁需求与环境和季节高度相关。例如，天气条件、降水、星期几、季节、一天中的时间段等都会影响租赁行为。如果能合理预测，运营商能在短期内准确预判租赁需求，提前调配、投放共享单车，避免“供给不足”或“车辆堆积”的情况；同时也能帮助运维部门提高管理效率、缩减成本，并提升用户满意度。

1.2 数据和来源

数据来源是美国华盛顿的两年历史骑行数据，和对应时段的天气信息。原始日志记录每天每小时的租赁总量，以及当时的天气、节假日、工作日等条件，可能反映出租赁数量的变化的。数据包含时间类特征：季节（season）、节假日（holiday）、是否工作日（workingday）、星期几（weekday）、小时（hr）等。天气类特征：天气情况（weathersit）、温度（temp）、体感温度（atemp）、湿度（hum）、风速（windspeed）等。

1.3 目标和要求

构建能够多步预测的时间序列模型：基于 LSTM 或 Transformer 深度学习模型，对过去 96 小时的特征进行学习和建模，预测未来 96 和 240 小时的共享单车租借总量 cnt；在测试集上，通过 MSE、MAE 指标评估预测误差，并多次实验计算标准差反应模型稳定性，最后

画曲线图与现实数据进行对比。

1.4 解决思路

a) **数据处理与滑动窗口构建：**按照时间顺序构建输入序列 X 和输出序列 y 的滑动窗口，将原始数据按天、小时进行清洗、特征提取与归一化，形成适合深度学习模型输入的数据格式；

b) **设计和使用模型：**构建 LSTM、Transformer 序列模型，提出的新模型；

c) **训练模型：**模型训练，选择合适的参数；

d) **评估模型：**在测试集上验证模型性能，计算 MSE、MAE，以及标准差，同时可视化预测曲线与真实曲线的对比。

二、 模型

2.1 LSTM

长短期记忆网络（LSTM, Long Short-Term Memory）是一种常用的循环神经网络（RNN）变体，专门为了解决 RNN 在长序列上训练时可能出现的梯度消失或梯度爆炸问题。它通过在内部维护“输入门、遗忘门、输出门”等机制，对历史状态进行一定程度的“记忆”或“遗忘”，从而在长时间跨度上仍能较好地学习时序依赖。

在本次实验中，LSTM 用于多步时间序列预测：给定过去 I=96 小时的特征数据（温度、湿度、天气、节假日、工作日、总租借量等），一次性预测未来 O=96 和 O=240 小时的共享单车租借量。

2.1.1 LSTM 原理简述

LSTM 保持一条贯穿始终的细胞状态 Cell State，可以在不被大幅修改的情况下长距离传递信息。

输入门、遗忘门、输出门：

- **输入门**决定当下时刻新信息如何写入细胞状态；
- **遗忘门**决定历史信息如何在细胞状态中被保留或遗忘；
- **输出门**决定当前时刻要输出多少信息到隐藏状态。

通过以上门控机制，LSTM 能够在面对长时间序列时依然保持稳定的梯度传递。

2.1.2 LSTM 结构

本实验使用的 LSTM 结构比较简洁：

1. **输入层**：batch, seq_len, input_size，其中 input_size=9 即特征列数；
2. **LSTM 层**：隐藏单元数（hidden_size）=64，能够处理长度为 seq_len=96 的序列；
3. **全连接层**：将 LSTM 最后时刻的隐状态映射到输出维度 output_size=96 一次性预测未来 96 小时或者 output_size=240 预测未来 240 小时。

2.1.3 伪代码

```
initialize model LSTMPredictor with:
    LSTM(input_size=9, hidden_size=64, num_layers=1)
    Linear(in_features=64, out_features=96)

define loss_function = MSELoss()
define optimizer = Adam(model.parameters(), lr=1e-3)

for epoch in [1..epochs]:
    model.train()
    train_mse_sum, train_mae_sum = 0, 0
    total_train_samples = 0

    for (X_batch, y_batch) in train_loader:
        # X_batch: shape (batch_size, 96, 9)
        # y_batch: shape (batch_size, 96)

        # 1) forward
        outputs = model(X_batch)  # shape (batch_size, 96)

        # 2) compute loss
        loss = MSELoss(outputs, y_batch)

        # 3) backward
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        # 4) accumulate MSE & MAE
        train_mse_sum += ...
        train_mae_sum += ...
        total_train_samples += batch_size

    # compute average train MSE & MAE
    train_mse = train_mse_sum / total_train_samples
    train_mae = train_mae_sum / total_train_samples

    # validation step (similar)
    ...
    val_mse, val_mae = ...
    print epoch log

# model(X_batch) 的前向过程
function forward(x):
    # x shape: (batch, seq_len=96, input_size=9)
    out, (h_n, c_n) = LSTM(x)
    # out shape: (batch, seq_len, hidden_size=64)
    # 取最后时刻的隐藏状态 out[:, -1, :]
    out_last = out[:, -1, :]

    # 全连接 -> (batch, 96)
    predictions = fc(out_last)

    return predictions
```

2.1.3. 训练与评估

1. **数据预处理：**

- 将数据按时间顺序构建“滑动窗口”，即 “过去 96 小时” -> “未来 96 小时”。
- 归一化特征 temp, hum, cnt 等范围到[0,1]。

2. **批量训练：**

- 使用 DataLoader 将(X, y)分批输入 LSTM。
- 用 MSE、MAE、Loss 评估损失并反向传播。

3. **验证集性能：**

- 每个 epoch 结束后，在验证集上做一次前向预测，计算 MSE、MAE 并观察是否过拟合，从而确定合适的 epoch。

4. **最终测试：**

- 使用最后训练好的模型，在 test_data 测试数据上执行同样的滑动窗口预测，评估其 MSE、MAE 并可视化预测曲线。

5. **多次训练和测试**

- 通过多次训练和测试，评估模型稳定性。

2.2 Transformer

Transformer 模型最早由 Google 的《Attention is All You Need》论文提出，用于机器翻译等序列到序列任务。它摆脱了 RNN 的循环结构，通过多头自注意力机制并行地捕捉序列内部的长依赖关系。如今在自然语言处理和时间序列预测等领域都获得了大量应用。

2.2.1 Transformer 时间序列预测

优势：可以并行处理序列中的各个位置，且对于长序列时不会像 RNN 那样在时间维度上累积误差。自注意力机制可以更灵活地“关注”远距离信息。

难点：时间序列通常需要显式位置编码来保留时序信息。在长序列上，Transformer 可能有计算和内存开销较大的问题。

2.2.2 本实验的 Transformer 结构

1. **输入投影**：将每个时刻的多维特征 9 维映射到 d_model 空间。
2. **位置编码**：使用正余弦位置编码让模型区分时间步顺序。
3. **Encoder**：多层 Self-Attention + FeedForward，对过去 96 小时进行编码。
4. **Decoder**：以全零构造 96 个步长的“目标占位”向量，逐步解码得到未来 96 小时的预测。
5. **输出层**：投影到 1 维，用于预测共享单车租借数量 cnt。

2.2.3 Transformer 伪代码

```
TransformerTimeSeries:
    input_fc: Linear(input_size=9 -> d_model=64)
    pos_encoder: PositionalEncoding(d_model=64)
    transformer: nn.Transformer(d_model=64, nhead=4, ...)
    output_fc: Linear(d_model=64 -> 1)

forward(src, tgt):
    # src: (batch, I=96, 9)
    # tgt: (batch, O=96, 9)  (或全零)

    src_embed = pos_encoder( input_fc(src) )  # -> (batch, I, d_model)
    tgt_embed = pos_encoder( input_fc(tgt) )  # -> (batch, O, d_model)

    out = transformer(src_embed, tgt_embed)   # -> (batch, O, d_model)
    out = output_fc(out)                      # -> (batch, O, 1)
    return out.squeeze(-1)                    # -> (batch, O)

# 与 LSTM 类似，只是模型换成 TransformerTimeSeries，并且decoder需要一个 tgt 输入，这里用全0
initialize model TransformerTimeSeries(...) with hyperparams
define loss_function = MSELoss()
define optimizer = Adam(model.parameters(), lr=1e-3)

function generate_tgt_zeros(batch_size, seq_len, input_dim=9):
    return zeros(batch_size, seq_len, input_dim)

for epoch in [1..epochs]:
    model.train()
    train_mse_sum, train_mae_sum = 0, 0
    total_train_samples = 0

    for (X_batch, y_batch) in train_loader:
        # X_batch: (batch, 96, 9)
        # y_batch: (batch, 96)

        tgt_zeros = generate_tgt_zeros(batch_size, 96, 9)

        optimizer.zero_grad()
        pred = model(X_batch, tgt_zeros)  # (batch, 96)
        loss = MSELoss(pred, y_batch)

        loss.backward()
        optimizer.step()

        # accumulate MSE & MAE
        ...

    # compute train_mse, train_mae
    # do validation in similar manner
    print epoch log

```

2.2.4 Transformer 训练与评估

1. **位置编码：**

- 为了让 Transformer 知道输入序列中每个时间步的位置，输入增加正余弦位置编码。

2. **Decoder 输入：**

- 采用“全零”序列+96 大小的特征维度来让 Transformer Decoder 解码 O=96 步。

3. **多步预测：**

- 和 LSTM 相同，最终输出(batch, O)。

4. 训练与验证：

- 同样使用 MSE、MAE、Loss 评估。

5. 测试集评估：

- 同 LSTM，把滑动窗口应用到测试数据，每个样本调用 model(src, tgt_zeros)，预测出 96 小时和 240 小时的序列并反归一化得到实际 cnt，计算 MAE、MSE 并画出曲线图。

6. 稳定性测试：

- 同 LSTM，多次执行测试评估模型再测试集上预测结果的 MAE、MS 以及稳定性。

2.3 新模型：多尺度卷积 + LSTM + 注意力 = MSCLA

下面提出一种融合多尺度卷积（Multi-Scale CNN）、LSTM 和注意力机制 Attention 的混合模型 MSCLA，用于多步时间序列预测：给定过去 I=96 小时的特征数据：温度、湿度、天气、节假日、工作日、总租借量等，预测未来 O=96 或 O=240 小时的共享单车租借量。

2.3.1 MSCLA 原理

在长序列预测中，单纯的 LSTM 可能无法充分利用不同时间尺度下的局部信息，尤其是当数据本身存在季节性、周期性或突发峰值等复杂模式时。可以借鉴 Temporal Convolutional Network[1]和 WaveNet [2]提出的扩大卷积思路，在输入层并行若干条不同膨胀率 dilation 或不同卷积核大小的卷积分支，提取多尺度的时序特征，然后再将多尺度卷积输出拼接后输入到 LSTM 以捕捉长时依赖；最后可以在 LSTM 输出序列上施加注意力机制，让模型自动学习关键时间步对预测的贡献度。

具体流程如下：多尺度卷积层通过并行的不同膨胀率（dilation）或不同卷积核大小的卷积层，提取序列中的多尺度局部特征。多条并行的 1D 卷积分支，每条分支具有不同的膨胀率或卷积核大小。每条分支输出固定数量的特征通道。最终将所有分支的输出在通道维度上拼接，形成融合后的多尺度特征表示。LSTM 编码对多尺度卷积层提取的特征进行时序建模，捕捉长时依赖关系。通过注意力机制，强调对预测最为关键的时间步的影响，提高模型对重要时刻的响应能力。将注意力机制生成的上下文向量映射到预测目标的维度，实现多步预测。

此外我观察上面的 LSTM 和 Transformer，发现他们在预测峰值时，通常会有点保守，

原因可能时这样：“峰值”出现频次往往较低，而且可能与常规数据分布差异明显。模型在最小化整体 Loss 时，更关注数据中占多数的中等或低值，而对高值错误并不特别敏感，于是模型会趋向于在峰值处更保守，因为大部分时间的预测做好了就能显著降低平均误差。于是我进一步调整了 Loss 函数，采用加权的 Loss，针对超过一定阈值的值赋予更多的权重。

2.3.2 MSCLA 结构

MSCLA 结构如下：

1. **输入层形状：**(batch,seq_len=I,input_size=9)，其中 9 表示特征列数（包括温度、湿度、天气、节假日、工作日、windspeed、cnt 等）。
2. **多尺度卷积层（Multi-Scale CNN）：**若干并行卷积分支，例如 dilation = 1, 2, 4；每个分支输出维度 cnn_out_channels；将并行分支输出在通道维度 concat，得到 (batch,I,cnn_out_dim)。
3. **LSTM 层：**隐藏单元数（hidden_size）= 64，能处理长度 I=96 的序列；输出序列形状为 (batch,I,64)，每个时间步都有一个隐藏状态。
4. **注意力机制（Attention）**以一个可学习的查询向量或多头注意力，对所有时间步的隐藏状态做加权聚合；输出一个上下文向量 (batch,64)。
5. **全连接层（FC）：**将上述上下文向量映射到 output_size=96 或 output_size=240，一次性预测对应时长的租借量曲线。

2.3.3 伪代码

```
# PSEUDOCODE
# 1. Prepare dataset
X_train, y_train, X_val, y_val = sliding_window(dataset, I=96, O=96)  # or O=240
scaler.fit_transform(...)
batchify(...)

# 2. Initialize model
model = MSCLANet(...)
optimizer = Adam(model.parameters(), lr=1e-3)
loss_fn = MSELoss()  # or MAE, or Weighted MSE

for epoch in range(num_epochs):
    model.train()
    for X_batch, y_batch in train_loader:
        pred = model(X_batch)  # shape: (batch, O)
        loss = loss_fn(pred, y_batch)
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

    # Evaluate on val
    model.eval()
    val_loss = 0
    for X_val_b, y_val_b in val_loader:
        with no_grad:
            val_pred = model(X_val_b)
            val_loss += loss_fn(val_pred, y_val_b)
    val_loss /= len(val_loader)

    print("Epoch {}, val_loss={}".format(epoch, val_loss))

# 3. Final test
# use best model on test_data
# compute MSE, MAE
# plot predictions
```

2.3.4 训练与评估过程

1. **数据预处理：**构建滑动窗口；使用 MinMaxScaler 将 temp, atemp, hum, windspeed, cnt 等特征值映射到 [0,1] 区间。
2. **批量训练：**DataLoader：将(X,y)分批次输入模型，其中 X 为(batch,96,9)；采用合适损失函数，使用 Adam 优化器进行反向传播更新网络参数。
3. **验证集性能：**每个 epoch 结束后，在验证集上做前向预测，计算 MSE、MAE 并观察曲线，选择合适 Epoch。
4. **最终测试：**使用在验证集上性能较好的模型，在 test_data 上执行相同的滑动窗口预测；计算其 MSE、MAE，并可视化预测曲线。
5. **多次训练和测试：**由于深度学习模型在不同随机初始化下可能存在一定波动，可以多次训练并统计均值与标准差；

三、 结果与分析

3.1 LSTM

3.1.1 训练过程

首先是 96 小时

通过观察验证集上的预测性能，差不多 30 左右指标不降反升，所以最后选择 30 的 Epoch

训练过程的输出结果：

```
E:\Program\Python\Python313\python.exe E:\Coding\PythonPrograms\machine_learning_homework_2024\data\model_lstm.py
训练样本数: 12020
验证样本数: 3005
[Epoch 1/30] Train MSE: 0.023238, Train MAE: 0.114749 | Val MSE: 0.038866,   Val MAE: 0.130905 | Loss: 0.013445
[Epoch 2/30] Train MSE: 0.013065, Train MAE: 0.080527 | Val MSE: 0.027084,   Val MAE: 0.111499 | Loss: 0.010813
[Epoch 3/30] Train MSE: 0.011113, Train MAE: 0.075019 | Val MSE: 0.026259,   Val MAE: 0.109581 | Loss: 0.010832
[Epoch 4/30] Train MSE: 0.010594, Train MAE: 0.072813 | Val MSE: 0.024953,   Val MAE: 0.110122 | Loss: 0.010011
[Epoch 5/30] Train MSE: 0.010239, Train MAE: 0.071089 | Val MSE: 0.022354,   Val MAE: 0.103790 | Loss: 0.008972
[Epoch 6/30] Train MSE: 0.009921, Train MAE: 0.069720 | Val MSE: 0.021844,   Val MAE: 0.103266 | Loss: 0.009564
[Epoch 7/30] Train MSE: 0.009737, Train MAE: 0.068940 | Val MSE: 0.023510,   Val MAE: 0.107280 | Loss: 0.009666
[Epoch 8/30] Train MSE: 0.009467, Train MAE: 0.067947 | Val MSE: 0.020808,   Val MAE: 0.100869 | Loss: 0.009353
```

最后输出本次的结果和一段预测曲线（曲线见下一小节）

```
[Epoch 30/30] Train MSE: 0.007176, Train MAE: 0.057887 | Val MSE: 0.017181,   Val MAE: 0.092758 | Loss: 0.007282
test_data行数: 2160
X_test_all shape: (1969, 96, 9)
y_test_all shape: (1969, 96)
Test MSE: 20298.7736, Test MAE: 92.0590
```

然后是 240 小时

还是选择 30 个 Epoch

训练过程的输出结果：

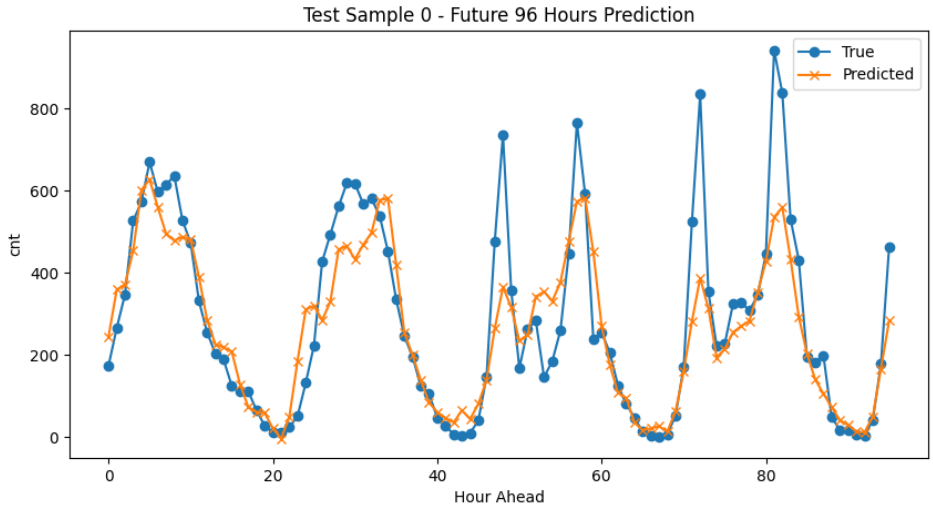
```
E:\Program\Python\Python313\python.exe E:\Coding\PythonPrograms\machine_learning_homework_2024\data\model_lstm.
训练样本数: 11904
验证样本数: 2977
[Epoch 1/30] Train MSE: 0.022827, Train MAE: 0.113067 | Val MSE: 0.036169,   Val MAE: 0.127000 | Loss: 0.015614
[Epoch 2/30] Train MSE: 0.013988, Train MAE: 0.084435 | Val MSE: 0.028595,   Val MAE: 0.113005 | Loss: 0.014301
[Epoch 3/30] Train MSE: 0.012415, Train MAE: 0.079894 | Val MSE: 0.028360,   Val MAE: 0.112617 | Loss: 0.012634
[Epoch 4/30] Train MSE: 0.011934, Train MAE: 0.078191 | Val MSE: 0.024316,   Val MAE: 0.106086 | Loss: 0.011262
```

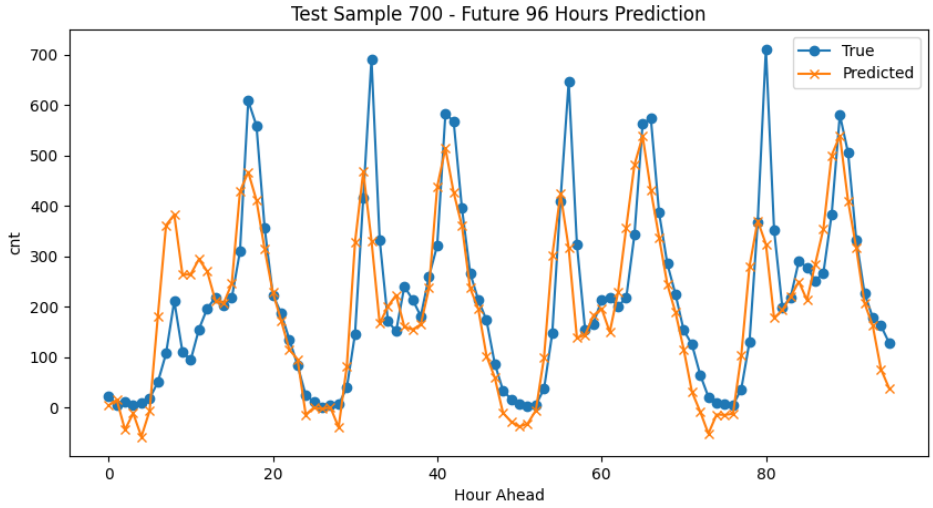
最后输出本次的结果和一段预测曲线

```
[Epoch 30/30] Train MSE: 0.009266, Train MAE: 0.066809 | Val MSE: 0.019995,   Val MAE: 0.098287 | Loss: 0.009868
test_data行数: 2160
X_test_all shape: (1825, 96, 9)
y_test_all shape: (1825, 240)
Test MSE: 28400.3586, Test MAE: 110.8539
```

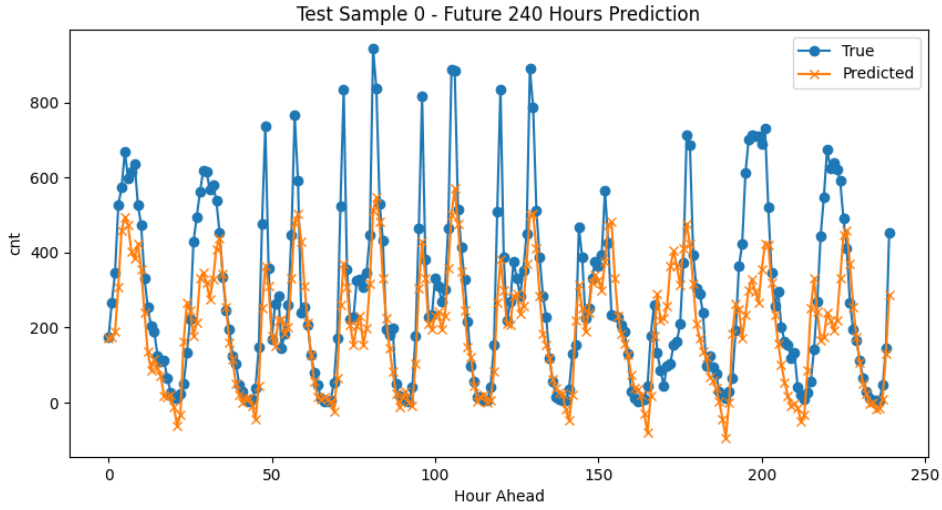
3.1.2 预测结果曲线

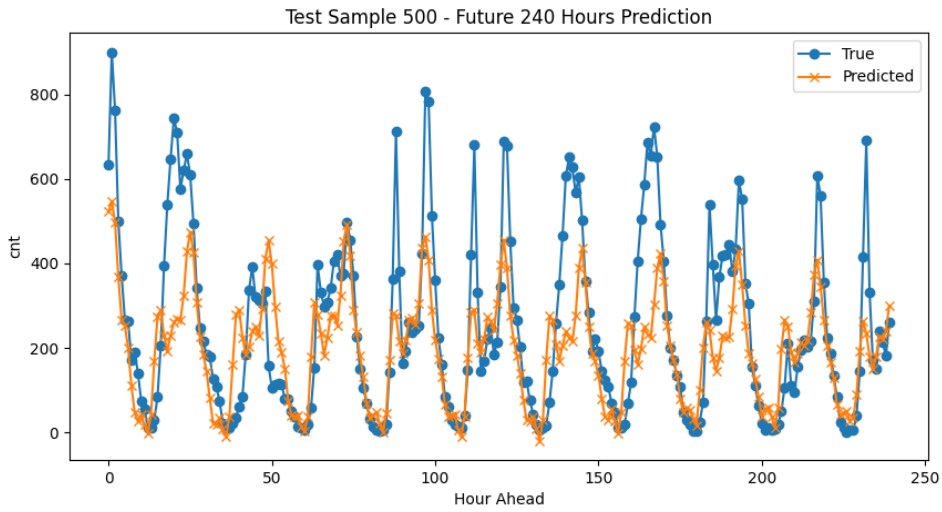
预测 96 小时，随机选取了其中两个窗口的预测结果绘制了如下的曲线





下面是预测 240 小时选择的两个窗口





3.1.3 模型稳定性

预测 96 小时的五轮的训练 MSE、MAE 以及标准差如下：

轮次	MSE	MAE
1	20298.7736	92.0590
2	21957.5224	96.4458
3	20447.1815	93.1595
4	20642.4489	92.3302
5	19893.1344	91.7428
标准差 std	699.5696930050955	1.7149297660254212

预测 240 小时的五轮的训练 MSE、MAE 以及标准差如下：

轮次	MSE	MAE
1	28400.3586	110.8539
2	29443.7118	113.2525
3	28447.2535	112.0095
4	28089.4154	110.2895
5	28593.1775	112.3442
标准差 std	455.11424871441073	1.059620268586817

3.1.4 分析

从 96 小时到 240 小时，MSE 和 MAE 都显著增加，说明预测范围越长，误差累积越明显。可能是 LSTM 在长序列预测中记忆能力有限，使得预测准确性下降。尽管 240 小时的预测误差更大，但标准差占均值的比例更低，表明 240 小时预测结果更稳定。总体上看 LSTM 能完成这个预测任务。96 小时预测误差较低但标准差相对稍高，适合短期预测任务。240 小时预测误差较高但标准差较小，预测效果更稳定但需改善长期预测性能。

3.2 Transformer

3.2.1 训练过程

预测 96 小时还是选择用 30 个 Epoch

训练过程的输出结果：

```
E:\Program\Python\Python313\python.exe E:\Coding\PythonPrograms\machine_learning_homework_2024\data\model_tansformer.py
train_data 行数: 15216
X_all: (15025, 96, 9)
y_all: (15025, 96)
训练样本数: 12020
验证样本数: 3005
[Epoch 1/30] Train MSE: 0.036778, MAE: 0.144963 | Val MSE: 0.060921, MAE: 0.191106 | Loss: 0.022899
[Epoch 2/30] Train MSE: 0.023705, MAE: 0.119238 | Val MSE: 0.055338, MAE: 0.187666 | Loss: 0.025810
[Epoch 3/30] Train MSE: 0.022533, MAE: 0.116081 | Val MSE: 0.055116, MAE: 0.187851 | Loss: 0.020240
[Epoch 4/30] Train MSE: 0.022264, MAE: 0.115271 | Val MSE: 0.055511, MAE: 0.187467 | Loss: 0.021207
[Epoch 5/30] Train MSE: 0.022141, MAE: 0.114877 | Val MSE: 0.056212, MAE: 0.187509 | Loss: 0.026137
```

最后输出本次的结果和一段预测曲线（曲线见下一小节）

```
[Epoch 30/30] Train MSE: 0.007008, MAE: 0.057494 | Val MSE: 0.019239, MAE: 0.094250 | Loss: 0.007386
X_test_all: (1969, 96, 9)
y_test_all: (1969, 96)
Test MSE: 21280.7590, Test MAE: 93.2311
```

预测 240 小时

训练过程的过程：

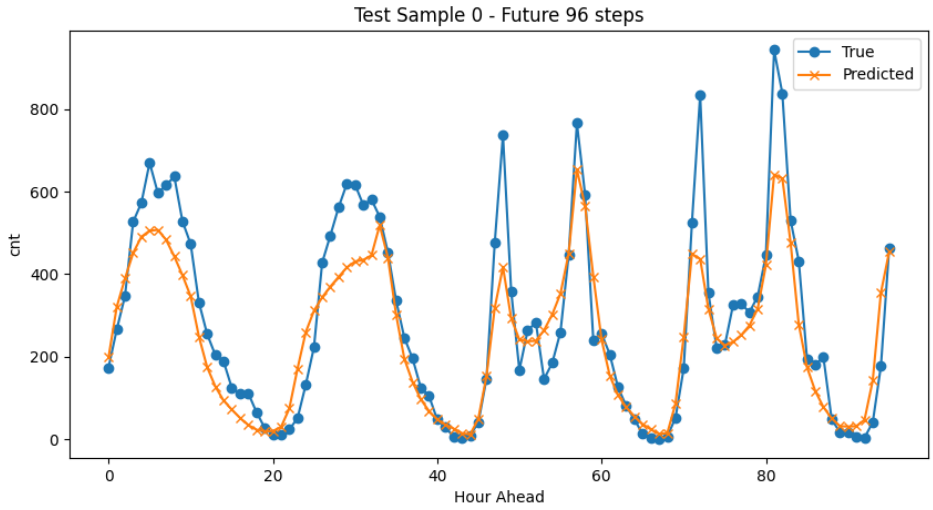
```
E:\Program\Python\Python313\python.exe E:\Coding\PythonPrograms\machine_learning_homework_2024\data\model_tansformer.py
train_data 行数: 15216
X_all: (14881, 96, 9)
y_all: (14881, 240)
训练样本数: 11904
验证样本数: 2977
[Epoch 1/30] Train MSE: 0.054889, MAE: 0.166941 | Val MSE: 0.063156, MAE: 0.193568 | Loss: 0.028702
[Epoch 2/30] Train MSE: 0.025938, MAE: 0.125282 | Val MSE: 0.059695, MAE: 0.189854 | Loss: 0.025495
[Epoch 3/30] Train MSE: 0.023904, MAE: 0.120023 | Val MSE: 0.055535, MAE: 0.187098 | Loss: 0.025259
```

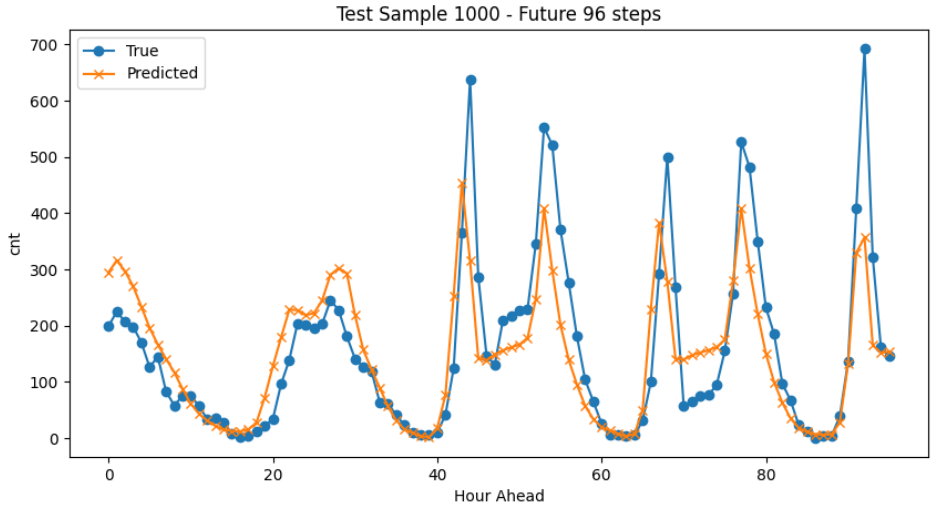
结果：

```
[Epoch 30/30] Train MSE: 0.010751, MAE: 0.073808 | Val MSE: 0.021386, MAE: 0.101717 | Loss: 0.010820
X_test_all: (1825, 96, 9)
y_test_all: (1825, 240)
Test MSE: 25757.1039, Test MAE: 106.1716
```

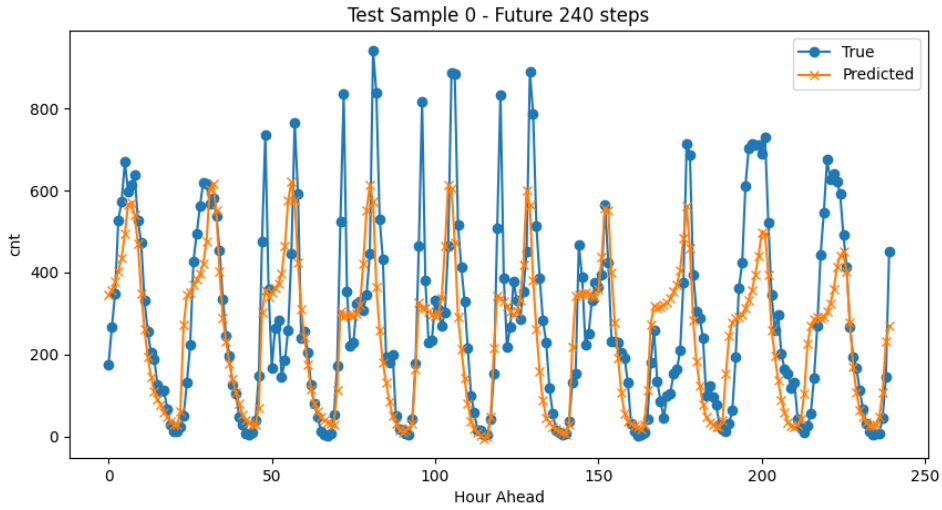
3.2.2 预测结果

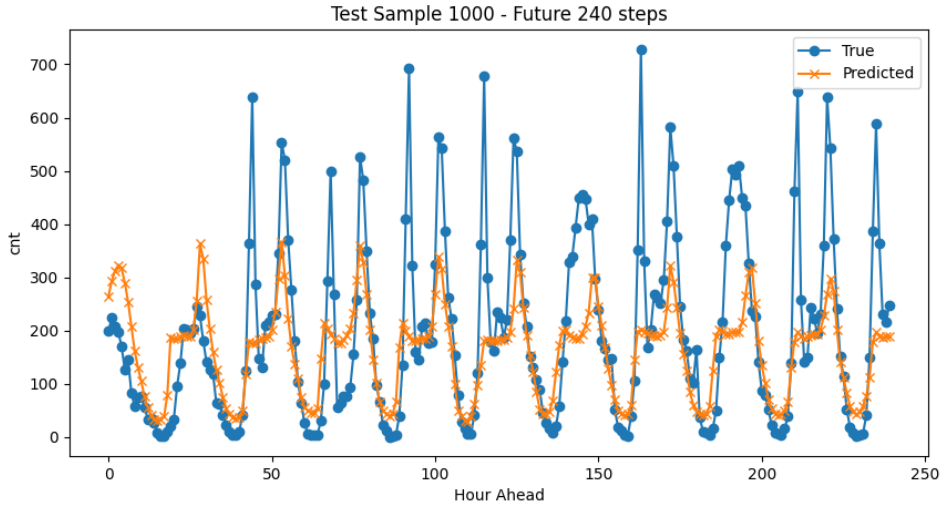
预测 96 小时，随机选取了其中两个窗口的预测结果绘制了如下的曲线





预测 240 小时，随机选取了其中两个窗口的预测结果绘制了如下的曲线





3.2.3 模型稳定性

96 小时预测

轮次	MSE	MAE
1	20546.1682	92.2503
2	22781.6179	96.7133
3	21280.7590	93.2311
4	20616.1581	91.7433
5	20516.5902	91.9139
标准差 std	863.720755606532	1.8448748232874785

240 小时预测

轮次	MSE	MAE
1	25757.1039	106.1716
2	28628.3153	111.7106
3	26425.2853	106.2438
4	27164.8546	109.4246
5	28750.3871	112.7591
标准差 std	1185.052943959389	2.717053275591043

3.2.4 分析

从 96 小时到 240 小时，MSE 和 MAE 都显著增加，说明预测范围越长，在 Transformer 中的误差累积也是越来越明显。LSTM 更适合稳定性要求高、短时间范围内的预测任务。Transformer 更适合捕捉长期依赖关系的复杂任务，但需要优化以提高训练结果的稳定性。Transformer 的 240 小时 MSE 和 MAE 明显低于 LSTM，表明其在长期预测中的效果更好。然而，Transformer 的标准差较大，说明在多次训练中，Transformer 的表现容易受到随机初始化、超参数选择的影响，模型的表现波动较大。

3.3 MSCLA

3.3.1 训练过程

预测 96 小时还是选择用 30 个 Epoch

训练过程的输出结果：

```
E:\Program\Python\Python313\python.exe E:\Coding\PythonPrograms\machine_learning_homework_2024\data\my_model.py
X_train: (12020, 96, 9) y_train: (12020, 96)
X_val: (3005, 96, 9) y_val: (3005, 96)
[Epoch 1/30] Train MSE: 0.024673, MAE: 0.120556 | Val MSE: 0.056923, MAE: 0.188071 | Loss: 0.021965
[Epoch 2/30] Train MSE: 0.018889, MAE: 0.103713 | Val MSE: 0.034760, MAE: 0.129110 | Loss: 0.017736
[Epoch 3/30] Train MSE: 0.012705, MAE: 0.080304 | Val MSE: 0.026247, MAE: 0.113271 | Loss: 0.010476
[Epoch 4/30] Train MSE: 0.011083, MAE: 0.074599 | Val MSE: 0.028433, MAE: 0.117218 | Loss: 0.012282
```

最后输出本次的结果和一段预测曲线

```
[Epoch 30/30] Train MSE: 0.006395, MAE: 0.054319 | Val MSE: 0.019032, MAE: 0.096759 | Loss: 0.006082
X_test_all: (1969, 96, 9) y_test_all: (1969, 96)
Test MSE: 23229.2520, Test MAE: 99.8355
```

预测 240 小时

训练过程的过程：

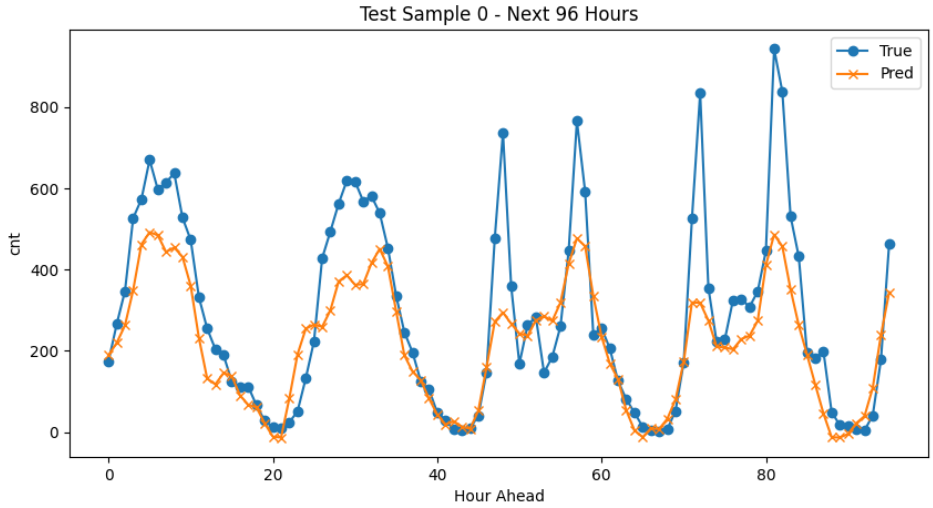
```
E:\Program\Python\Python313\python.exe E:\Coding\PythonPrograms\machine_learning_homework_2024\data\my_model.py
X_train: (11904, 96, 9) y_train: (11904, 240)
X_val: (2977, 96, 9) y_val: (2977, 240)
[Epoch 1/30] Train MSE: 0.025429, MAE: 0.122882 | Val MSE: 0.055437, MAE: 0.187301 | Loss: 0.027076
[Epoch 2/30] Train MSE: 0.020397, MAE: 0.109712 | Val MSE: 0.036111, MAE: 0.133092 | Loss: 0.015785
[Epoch 3/30] Train MSE: 0.013715, MAE: 0.084789 | Val MSE: 0.028692, MAE: 0.116543 | Loss: 0.013930
[Epoch 4/30] Train MSE: 0.012085, MAE: 0.079005 | Val MSE: 0.025503, MAE: 0.109230 | Loss: 0.013700
```

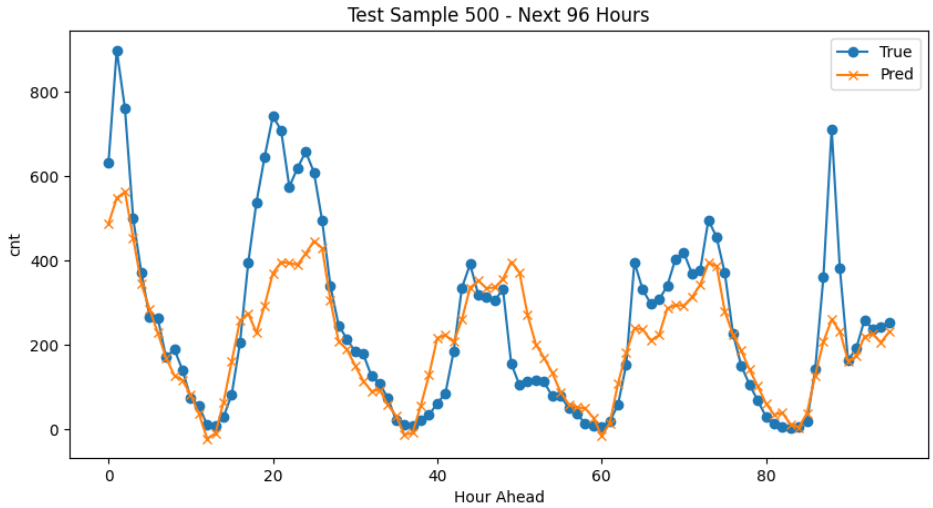
结果：

```
[Epoch 30/30] Train MSE: 0.008488, MAE: 0.063781 | Val MSE: 0.021836, MAE: 0.102634 | Loss: 0.008131
X_test_all: (1825, 96, 9) y_test_all: (1825, 240)
Test MSE: 28952.8251, Test MAE: 113.0817
```

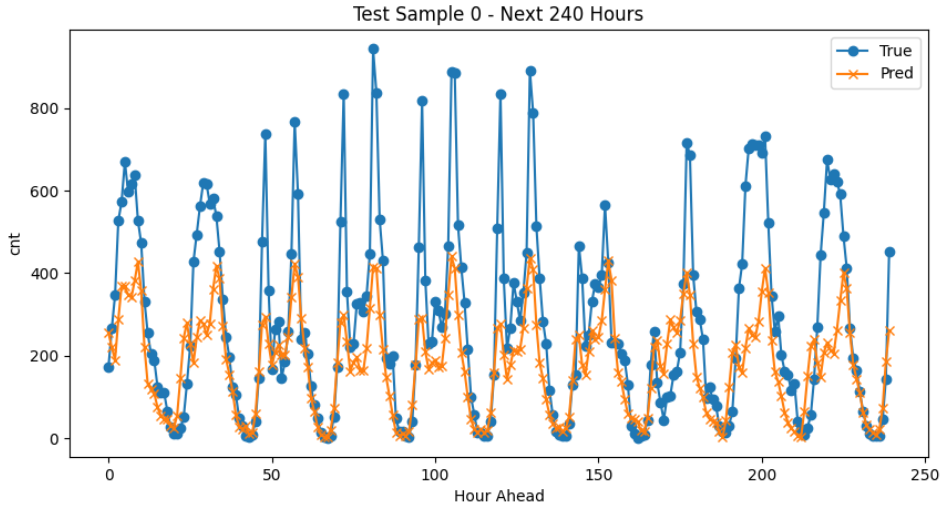
3.3.2 预测结果

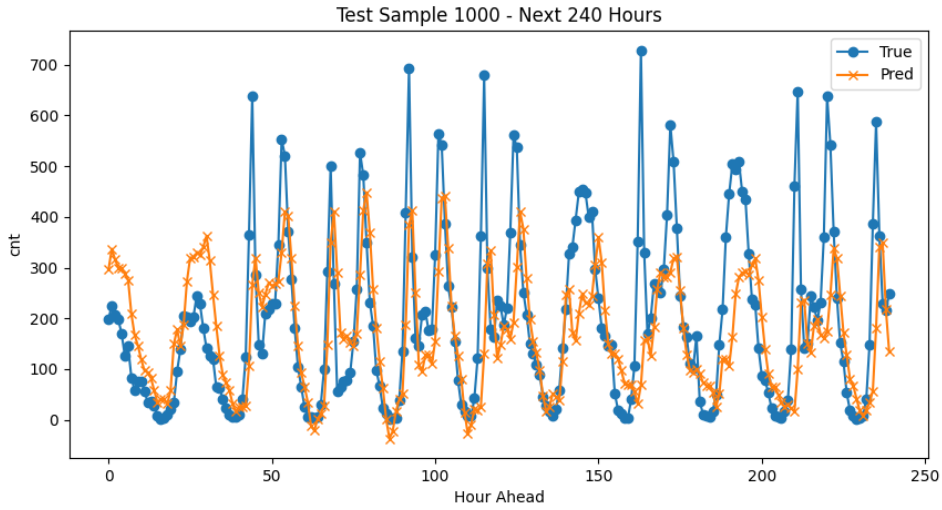
预测 96 小时，随机选取了其中两个窗口的预测结果绘制了如下的曲线





预测 240 小时，随机选取了其中两个窗口的预测结果绘制了如下的曲线





3.3.3 模型稳定性

96 小时预测

轮次	MSE	MAE
1	22741.1016	97.9898
2	22867.0336	97.6864
3	26185.6055	108.4054
4	25253.4535	104.0220
5	25531.4830	107.7608
标准差 std	1430.522987041666	4.606844067862509

240 小时预测

轮次	MSE	MAE
1	30550.4088	115.6367
2	28941.9955	113.9501
3	31117.4247	119.7808
4	29221.3833	114.0961
5	30351.8528	116.4369
标准差 std	823.9299875022033	2.118597002169121

3.3.4 分析

从上面可以看出，新模型其实并没有表现出非常好的效果，可能是中间参数以及 Epoch 数量还有优化空间，此外从图中可以产出新加入的加权 Loss 应该是起到了一定的作用，高点相比较 LSTM 和 Transformer 可能好了一点，但是也不明显，可能还是要更加细致的方案，而直接超过阈值就增加权重，可能并不太合适。模型的稳定性正常相比 LSTM 和 Transformer 都区别不大。存在的问题还有中间 Epoches 的选择，有时选择小，模型不充分，选择大了容易过拟合效果反而下降，但是模型训练过程又有一定的随机性，所以大小有时也不太能通过训练时输出的指标精确判断，只有大致的范围。

四、 讨论和总结

本次实验完成了 LSTM 和 Transformer 的时间序列预测任务。

LSTM 能够有效地捕捉长时间序列中的长期依赖关系。它通过引入记忆单元和门控机制（输入门、遗忘门、输出门）来解决传统 RNN 在处理长序列时遇到的梯度消失问题。它的记忆单元能够在每个时间步骤中传递重要的信息，同时遗忘无关的部分，从而使网络能够在较长的时间跨度上进行有效学习。

Transformer 是一种基于自注意力机制的深度学习模型，广泛应用于自然语言处理任务。Transformer 完全摆脱了顺序计算的限制，采用了自注意力机制来并行处理序列中的所有元素，显著提高了计算效率。它通过多头注意力机制对输入的每个位置进行加权平均，捕捉序列中各个部分之间的关系。它具有并行处理能力并且对长范围依赖的建模能力更好。

在共享单车租赁数量预测的这个问题中，租赁数量受多种因素的影响，包括天气、季节、工作日非工作日、节假日等，而这些因素随时间变化是典型的时间序列预测问题。LSTM 在处理周期性和季节性数据时表现较好，特别是通过滑动窗口方式构建的序列数据。而 Transformer 在捕捉周期性和季节性变化方面也很适用，注意力机制可以帮助模型集中注意力于过去的关键时间点。这两个都能很好处理此类预测问题。

通过 96 小时和 240 小时这两个的预测结果可以看出 LSTM 在短期预测中能够依赖相对较近的历史数据进行训练，捕捉短期的趋势变化。其记忆单元能够有效存储并处理最近时间段的数据，帮助模型在短期内做出较准确的预测。而 Transformer 模型在处理更长时间序列时，能够通过并行处理并捕捉时间序列中较远距离的依赖关系，因此适合用于较长时间跨度的预测。其自注意力机制能够处理长时间依赖，并且在长时间预测任务中，能够减少传统 LSTM 在长期依赖中可能遇到的梯度消失问题。

在 epoch 的选择上如果 epoch 设置过小，模型可能无法充分学习到数据的规律，导致欠拟合。如果 epoch 设置过大，模型可能会开始对训练数据过度拟合，导致过拟合，此时训练误差继续降低，但验证误差开始上升。可以通过监控训练误差和验证误差的变化来判断。当验证误差开始上升时，表示模型可能已经学习到了训练集的噪声，开始出现过拟合，此时就应该停止训练。还可以使用早停，允许验证误差在若干轮训练内没有显著改善后才停止，可以在每一轮训练中保存验证误差最小的模型，还可以交叉验证在不同的数据集划分上训练和评估模型的泛化能力。但是本次实验没有使用特殊的方法，直接通过观察训练过程中的验证误差和损失函数的变化趋势，确定的 Epoch 大致为 30，当然这种方法并不准确，可能会由

于模型随机性不是很稳定。

提出的新模型多尺度卷积 + LSTM + 注意力的 MSCLA 模型在这个预测任务中也完成了任务。通过结合多尺度卷积提取不同时间尺度的局部特征、LSTM 捕捉长时依赖关系以及注意力机制强调关键时间步的影响，MSCLA 模型在捕捉复杂时序模式和应对峰值预测方面展现出一定的性能。此外，还通过自定义加权损失函数的策略，进一步提升了模型对高值区域的预测能力。遗憾的是实际效果可能跟其他模型拉不开差距，可能还需要进行进一步模型调优。

最后再回到这个预测任务上，通过查看数据集，我觉得还是有很多信息可以挖掘的，目前模型以及能准确反映出同一天内的波动情况，正确描述一天内的变化趋势，而对于周、月、年来说是否也隐含一些关系；还有题中给出的 casual 和 registered 两类租赁分开，是否他们两个对于某些日期也有不一样的关系，或者他们之间是否也有一定的比例联系，这些关系可能都可以拿出来单独分析。

【参考文献】

[1] Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. Neural computation, 9(8), 1735-1780.

[2] Vaswani A. Attention is all you need[J]. Advances in Neural Information Processing Systems, 2017.

[3] Bai S, Kolter J Z, Koltun V. An empirical evaluation of generic convolutional and recurrent networks for sequence modeling[J]. arXiv preprint arXiv:1803.01271, 2018.

[4] Van Den Oord A, Dieleman S, Zen H, et al. Wavenet: A generative model for raw audio[J]. arXiv preprint arXiv:1609.03499, 2016, 12.

【代码】

见 GitHub

Blank-Sky/machine_learning_homework_2024