2024 年苏州大学研究生机器学习课程报告

院　　别：计算机科学与技术学院　类　　别：硕士

学生姓名：何欣娜（独立完成）　学　　号：20244227084

研究领域：社交网络　授课老师：蔡蔚然

日期：2025 年 1 月 18 日

1. 问题介绍

共享单车系统为城市交通带来了不少便利，但是运营效率和服务质量仍有优化的空间，准确预测共享单车的租赁数量能够助力运营商合理规划车辆调度。因此本研究借助机器学习模型，考虑历史特征，对未来的共享单车租赁数量进行预测，这将为租赁公司合理调度车辆、提升服务效率提供有力支持。

本研究实质上是一个时间序列预测问题，旨在基于过去 96 小时的数据，分别构建短期（未来 96 小时间）和长期（未来 240 小时间）的租赁数量变化曲线预测模型。通过分别训练这两个模型，力求精准把握不同时间维度下的租赁数量变化趋势，为共享单车运营的科学决策提供坚实依据。

本研究将尝试三种方法来解决该问题：

1. 使用 LSTM 模型进行预测；
2. 使用 Transformer 模型进行预测；
3. 使用改进模型进行预测

2. 模型

2.1 数据预处理

根据观察数据集，可以发现共有 16 个特征（['instant', 'dteday', 'season', 'yr', 'mnth', 'hr', 'holiday', 'weekday', 'workingday', 'weathersit', 'temp', 'atemp', 'hum', 'windspeed', 'casual', 'registered']）和 1 个预测标签 cnt

Field	instant	dteday	season	yr	mnth	hr	holiday	weekday
Description	Index Number	date	Season Code	years	month	Hour	Holiday logo	Day of the week

Field	workingday	weathersit	temp	atemp	hum	windspeed	casual	registered	cnt
Description	Working day logo	Weather Condition Code	Actual temperature	Feeling temperature	humidity	Wind speed	Non-registered user - Vehicle	Registered User - Vehicle	Total rental quantity

特征处理：因 dteday 为日期数据类型，这里将其进行提取保留 day 部分，因为 yr，mnth 已经保留了月份和年份信息。

```
data['dteday'] = pd.to_datetime(data['dteday'])
data['dteday'] = data['dteday'].dt.day
```

处理异常值和空白值:这里将处理后得新数据保存到本地。

```
data = pd.read_csv(filepath)
```

```
features = ['instant', 'dteday', 'season', 'yr', 'mnth', 'hr', 'holiday
', 'weekday', 'workingday', 'weathersit', 'temp', 'atemp', 'hum', 'wind
speed', 'casual', 'registered']
label = 'cnt'
data['dteday'] = pd.to_datetime(data['dteday'])
data['dteday'] = data['dteday'].dt.day
iso_forest = IsolationForest(contamination=0.01)
outliers = iso_forest.fit_predict(data)
data = data[outliers != -1]
data.fillna(data.mean(), inplace=True)
data.to_csv("processed_data_train.csv", index=False)
```

归一化：用 MinMaxScaler 对特征进行归一化处理，确保数据在相同尺度（0，1）中。

```
if scaler_X is None:
   scaler_X = MinMaxScaler(feature_range=(0, 1))
   X_scaled = scaler_X.fit_transform(X)
else:
   X_scaled = scaler_X.transform(X)
```

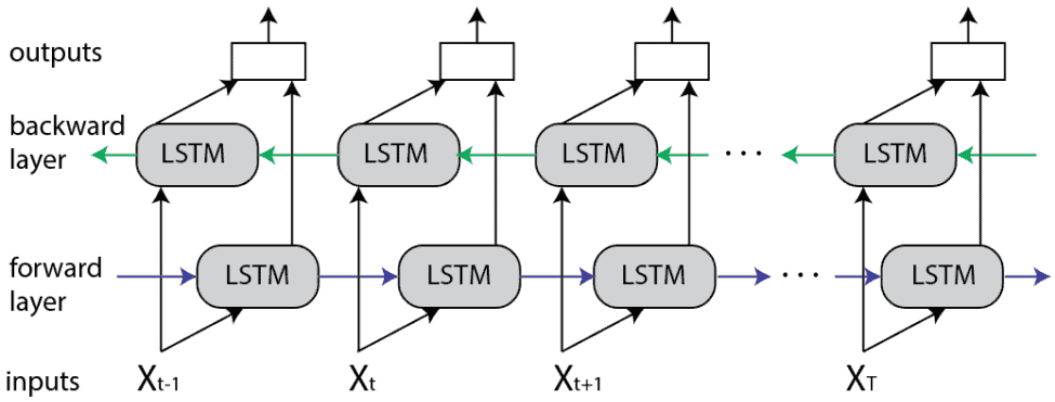
序列构建：将历史数据转换为固定长度（96）的序列，分别切割出长度为 96 的历史序列和未来长度为 96 和 240 的预测标签序列，分别记为（X_short， y_short），（X_long， y_long）。

2.2 LSTM 模型

本小节使用长短期记忆网络（LSTM）模型预测共享单车租赁数量 cnt。

在这里我构建了一个包含双向 LSTM 层、Dropout 层、Batch Normalization 层和全连接层的 LSTM 模型。模型使用 Adam 优化器，并使用 MSE、MAE 作为损失函数，用方差来评估实验稳定性。

这里使用的是 BiLSTM，其结构大致如下，由一个前向传导的 LSTM 链和后向传导的 LSTM 链组成，来捕捉其上下文中的局部的邻居之间的局部特征。



LSTM 模型的构造参数如下

```
# LSTM 模型构建
```

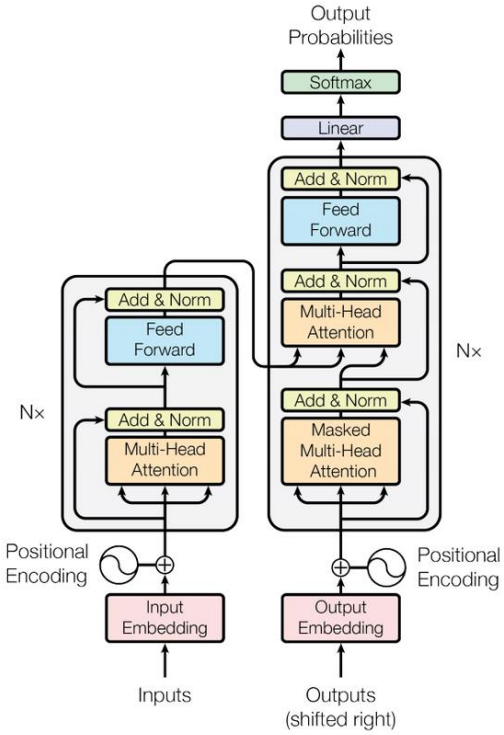
```
def create_lstm_model(input_shape, predict_length):
    model = Sequential()
    model.add(Bidirectional(LSTM(128, return_sequences=True, input_shape=input_shape)))
    model.add(Dropout(0.2))
    model.add(BatchNormalization())
    model.add(Bidirectional(LSTM(64, return_sequences=True)))
    model.add(Dropout(0.2))
    model.add(Bidirectional(LSTM(32)))
    model.add(Dropout(0.2))
    model.add(Dense(64, activation='relu'))  # 增加全连接层
    model.add(Dense( predict_length))  # 输出预测值
    model.compile(optimizer=Adam(learning_rate=0.001), loss='mse', metrics=['mae'])
    return model
```

通过使用 Dropout 和 Adam,限制学习率和随机局部丢弃来防止过拟合问题。

2.3 Transformer 模型

本节用 Transformer 模型进行短期和长期预测，分别定义了两个类：TransformerBlock 和 TransformerModel 。TransformerBlock 类实现了 Transformer 模型中的一个块，包括多头自注意力（MultiHeadAttention）和前馈网络（ffn）。TransformerModel 类则使用多个 TransformerBlock 来构建完整的 Transformer 模型，最后在输出层，通过全连接层输出预测的租赁数量。

经典的 Transformer 框架如下，由左侧的编码器（Encoder）和右侧的解码器（Decoder）组合而成，同时需要将位置信息进行编码作为 PE 与输入嵌入一起做加法放入编码器。



这里对 Transformer 模型做简单调整以适应本任务，由一层多头注意力层，一层 Dropout 层，一层归一化层，再放入前馈网络层和 Dropout 层，最后归一化输出作为一个块，如下为 Transformer Block 的结构：

```
# 定义 Transformer 模型
class TransformerBlock(tf.keras.layers.Layer):
    def __init__(self, embed_dim, num_heads, ff_dim, rate=0.1):
        super(TransformerBlock, self).__init__()
        self.att = tf.keras.layers.MultiHeadAttention(num_heads=num_hea
ds, key_dim=embed_dim)
        self.ffn = tf.keras.Sequential([
            Dense(ff_dim, activation="relu"),
            Dense(embed_dim),
        ])
        self.layernorm1 = LayerNormalization(epsilon=1e-6)
        self.layernorm2 = LayerNormalization(epsilon=1e-6)
        self.dropout1 = Dropout(rate)
        self.dropout2 = Dropout(rate)
    def call(self, inputs, training):
        attn_output = self.att(inputs, inputs)
        attn_output = self.dropout1(attn_output, training=training)
        out1 = self.layernorm1(inputs + attn_output)
        ffn_output = self.ffn(out1)
        ffn_output = self.dropout2(ffn_output, training=training)
        return self.layernorm2(out1 + ffn_output)
```

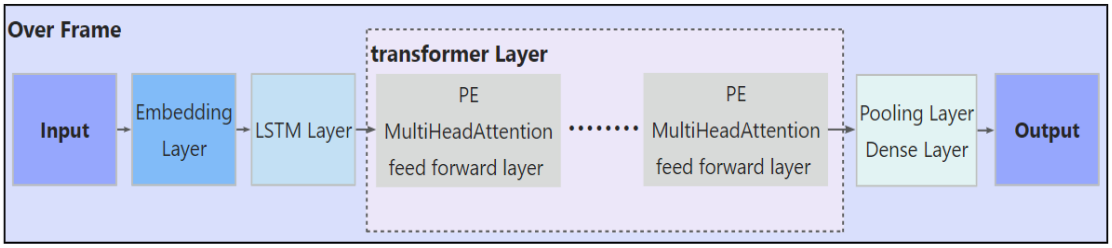
将多个 Transformer Block 进行堆叠，放入嵌入前要将特征嵌入与位置嵌入

做加法运算，再放入编码器，在编码其中通过堆叠的多个 Transformer Block 传递嵌入，最后放入池化层，再放入全连接层将嵌入大小调整为指定的规模后输出。

```
class TransformerModel(Model):
    def __init__(self, sequence_length, feature_dim, embed_dim, num_heads, ff_dim, num_layers, output_dim):
        super(TransformerModel, self).__init__()
        self.embed_layer = Dense(embed_dim)
        self.transformer_blocks = [TransformerBlock(embed_dim, num_heads, ff_dim) for _ in range(num_layers)]
        self.global_pool = tf.keras.layers.GlobalAveragePooling1D()
        self.output_layer = Dense(output_dim)
    def call(self, inputs, training=False):
        pos_encoding = positional_encoding(sequence_length, 256)
        pos_encoding = tf.expand_dims(pos_encoding, axis=0)
        x = self.embed_layer(inputs)
        x = x + pos_encoding
        for transformer_block in self.transformer_blocks:
            x = transformer_block(x, training=training)
        x = self.global_pool(x)
        return self.output_layer(x)
```

2.4 Him-model

第三问我使用了 LSTM + Transformer 混合模型结合对特征进行嵌入处理。我将其命名为 Hi-model，它的框架结构如下：



这里对 Him-model 框架进行简单描述，它由以下几部分构成：

1. 输入：16 项特征 data[features]（X）与标签 data[label]（y），predict_length。
2. Embedding 层：首先用 Dense 对特征进行学习，自动化学习特征表示。
3. LSTM 层：用双向 LSTM 层来提取时间序列中的局部时序依赖。
4. Transformer 层：通过多层 Transformer 层提取的全局时序特征，在每一层 Transformer 前添加位置编码，再用自注意力机制（MultiHeadAttention）和前馈神经网络增强特征表示。
5. 输出：放入池化层和全连接层，输出指定 predict_length 长度的预测序列。

LSTM 层较第一问的 solution 更加简单，主要是为了捕捉邻居节点之间的关联

```
 # LSTM 层：捕捉局部时间依赖
 x = layers.Bidirectional(layers.LSTM(128, return_sequences=True))(x)
 x = layers.Dropout(0.2)(x)
 x = layers.BatchNormalization()(x)
```

transformer block 不另外封装，加入 PE，堆叠多层注意力机制和前馈网络与全连接层进行组织，同时对齐形状。

```
# Transformer 层：捕捉全局依赖
    for _ in range(num_layers):
        # 添加位置编码
        pos_encoding = positional_encoding(sequence_length, 256)
        pos_encoding = tf.expand_dims(pos_encoding, axis=0)  # 位置编码维度扩展(1, seq_len, embed_dim)
        x = x + pos_encoding  # 将位置编码加到输入的 x
        attn_output = layers.MultiHeadAttention(num_heads=num_heads, key_dim=embed_dim)(x, x)
        attn_output = layers.Dropout(0.2)(attn_output)
        attn_output = layers.Dense(256)(attn_output)
        x = layers.LayerNormalization(epsilon=1e-6)(x + attn_output)
        ffn_output = layers.Dense(ff_dim, activation='relu')(x)
        ffn_output = layers.Dense(256)(ffn_output)  # 确保维度为 256
        ffn_output = layers.Dropout(0.2)(ffn_output)
       x =layers.LayerNormalization(epsilon=1e-6)(x+ffn_output)#残差连接
```

这里将 PE 位置编码函数构造得更加复杂一些，将技术位置和偶数位置分别用正弦和余弦函数编码，再简单拼接作为 PE。

```
def positional_encoding(seq_len, embed_dim):
    position = tf.range(seq_len, dtype=tf.float32)[:, tf.newaxis]
    div_term = tf.exp(tf.range(0, embed_dim, 2, dtype=tf.float32) * -(tf.math.log(10000.0) / embed_dim))  # (embed_dim // 2)
    pe = tf.zeros((seq_len, embed_dim), dtype=tf.float32)
    # 对偶数位置用正弦
    sin_values = tf.sin(position * div_term)  # (seq_len, embed_dim//2)
    # 对奇数位置用余弦
    cos_values = tf.cos(position * div_term)  # (seq_len, embed_dim//2)
    # 拼接
    pe = tf.concat([sin_values, cos_values], axis=-1)  # 拼接成 (seq_len, embed_dim)
    return pe
```

3. 结果与分析

实验设置：
三个模型的超参数设置:

超参数	值	描述
embed_dim	64	位置编码和 Transformer 中头的维度
num_heads	4	Transformer 中的多头注意力数目
ff_dim	128	前馈神经网络的隐藏层维度
num_layers	2	Transformer 中的层数
batch_size	32	训练时的批次大小
num_layers	2	堆叠注意力层数
num_epochs	10	训练轮次
num_trials	11	训练次数（用于计算方差）

总计：这里以 Him-model 的参数量为例：

Total params: 1014240 (3.87 MB) ；Trainable params: 1013728 (3.87 MB)；Non-trainable params: 512 (2.00 KB)

评估指标：均方误差（MSE）、平均绝对误差（MAE）
实验结果：
短期预测前 5 轮 MSE 和 MAE 值为例（每个训了 10 轮）：

模型	MSE	MAE	AVE MSE	AVE MAE	STD MSE	STD MAE
	0.0053	0.0509				
	0.0040	0.0443				
LSTM	0.0034	0.0412	0.0204	0.0158	0.0005	0.0018
	0.0031	0.0392				
	0.0028	0.0378				
Transformer	0.0138	0.0851	0.0204	0.0991	0.0034	0.0106
	0.0064	0.0561				
	0.0051	0.0495				
	0.0043	0.0454				
	0.0039	0.0433				

Him-model	0.0098	0.0694	0.0269	0.1151	0.0005	0.0009
	0.0097	0.0688				
	0.0098	0.0690				
	0.0096	0.0683				
	0.0096	0.0683				

同时也计算了 10 次实验的 MSE 和 MAE 的平均值和标准差。实际的实验结果如下：

短期预测

LSTM

```
In [188]: model_short = create_lstm_model((sequence_length, 16),short_predict_length)
          lstm_model_short, mse_mean_short, mse_std_short, mae_mean_short, mae_std_short = train( model_short, X_train_short, y_train_short, X_test_
              sequence_length, epochs=10, batch_size=32, num_trials=10)
          print("short prediction:")
          print(f"Average MSE: {mse_mean_short} ± {mse_std_short}")
          print(f"Average MAE: {mae_mean_short} ± {mae_std_short}")
          # 模型预测
          y_pred_short = lstm_model_short.predict(X_test_short)

          # 反归一化预测值与真实值
          y_pred_short = scaler_y_short.inverse_transform(y_pred_short.reshape(-1, 1))
          y_test_short = scaler_y_short.inverse_transform(y_test_short.reshape(-1, 1))
```

```
376/376 [==============================] - 67s 179ms/step - loss: 0.0024 - mae: 0.0345 - val_loss: 0.0138 - val_mae: 0.0797
Epoch 5/10
376/376 [==============================] - 62s 166ms/step - loss: 0.0024 - mae: 0.0346 - val_loss: 0.0123 - val_mae: 0.0749
Epoch 6/10
376/376 [==============================] - 64s 170ms/step - loss: 0.0024 - mae: 0.0346 - val_loss: 0.0124 - val_mae: 0.0759
Epoch 7/10
376/376 [==============================] - 64s 170ms/step - loss: 0.0024 - mae: 0.0345 - val_loss: 0.0129 - val_mae: 0.0775
Epoch 8/10
376/376 [==============================] - 62s 166ms/step - loss: 0.0023 - mae: 0.0343 - val_loss: 0.0126 - val_mae: 0.0764
Epoch 9/10
376/376 [==============================] - 66s 175ms/step - loss: 0.0023 - mae: 0.0344 - val_loss: 0.0120 - val_mae: 0.0745
Epoch 10/10
376/376 [==============================] - 64s 172ms/step - loss: 0.0023 - mae: 0.0342 - val_loss: 0.0135 - val_mae: 0.0789
(1969, 96, 16) (1969, 96, 1)
62/62 [==============================] - 4s 71ms/step - loss: 0.0160 - mae: 0.0824
short prediction:
Average MSE: 0.015824700891971587 ± 0.0005833404408137774
Average MAE: 0.08339446261525155 ± 0.0018017039114383267
62/62 [==============================] - 8s 63ms/step
```

Transformer

```
# 输出平均值和标准差
print(f"Average MSE: {np.mean(mse_results):.4f} ± {np.std(mse_results):.4f}")
print(f"Average MAE: {np.mean(mae_results):.4f} ± {np.std(mae_results):.4f}")
```

```
1_loss: 0.0250 - val_mean_squared_error: 0.0250 - val_mean_absolute_error: 0.1078
Epoch 6/10
376/376 [==============================] - 29s 76ms/step - loss: 0.0028 - mean_squared_error: 0.0028 - mean_absolute_error: 0.0369 - va
1_loss: 0.0280 - val_mean_squared_error: 0.0280 - val_mean_absolute_error: 0.1143
Epoch 7/10
376/376 [==============================] - 29s 77ms/step - loss: 0.0028 - mean_squared_error: 0.0028 - mean_absolute_error: 0.0369 - va
1_loss: 0.0260 - val_mean_squared_error: 0.0260 - val_mean_absolute_error: 0.1099
Epoch 8/10
376/376 [==============================] - 29s 77ms/step - loss: 0.0028 - mean_squared_error: 0.0028 - mean_absolute_error: 0.0370 - va
1_loss: 0.0266 - val_mean_squared_error: 0.0266 - val_mean_absolute_error: 0.1116
Epoch 9/10
376/376 [==============================] - 29s 78ms/step - loss: 0.0028 - mean_squared_error: 0.0028 - mean_absolute_error: 0.0371 - va
1_loss: 0.0254 - val_mean_squared_error: 0.0254 - val_mean_absolute_error: 0.1090
Epoch 10/10
376/376 [==============================] - 29s 76ms/step - loss: 0.0028 - mean_squared_error: 0.0028 - mean_absolute_error: 0.0368 - va
1_loss: 0.0266 - val_mean_squared_error: 0.0266 - val_mean_absolute_error: 0.1115
62/62 [==============================] - 2s 25ms/step
Average MSE: 0.0204 ± 0.0034
Average MAE: 0.0991 ± 0.0106
```

Him-model

```
Epoch 7/10
376/376 [==============================] - 97s 259ms/step - loss: 0.0096 - mean_absolute_error: 0.0682 - val_loss: 0.0269 - val_mean_ab
solute_error: 0.1138 - lr: 5.0000e-04
Epoch 8/10
376/376 [==============================] - ETA: 0s - loss: 0.0096 - mean_absolute_error: 0.0681Restoring model weights from the end of
the best epoch: 3.
376/376 [==============================] - 99s 265ms/step - loss: 0.0096 - mean_absolute_error: 0.0681 - val_loss: 0.0269 - val_mean_ab
solute_error: 0.1139 - lr: 5.0000e-04
Epoch 8: early stopping
62/62 [==============================] - 5s 80ms/step - loss: 0.0271 - mean_absolute_error: 0.1149
```

```
In [11]: # 输出平均值和标准差
         print(f"Average MSE: {np.mean(him_model_mse_results_short):.4f} ± {np.std(him_model_mse_results_short):.4f}")
         print(f"Average MAE: {np.mean(him_model_mae_results_short):.4f} ± {np.std(him_model_mae_results_short):.4f}")

         Average MSE: 0.0269 ± 0.0005
         Average MAE: 0.1151 ± 0.0009
```

分析：在短期预测中 LSTM 模型的表现最好，其次是 Transformer 模型，根据 STD 分析其稳定性 Him-model 较另外两个模型而言更好，但是 MSE 和 MAE 稍逊，LSTM 很擅长捕捉局部的邻居之前的时序关系，而解决本问题提出的任务自行车租赁数量更多地依赖相邻的时间步的特征。

长期预测

LSTM

```
Epoch 8/10
372/372 [==============================] - 50s 133ms/step - loss: 0.0041 - mae: 0.0448 - val_loss: 0.0192 - val_mae: 0.0959
Epoch 9/10
372/372 [==============================] - 48s 129ms/step - loss: 0.0041 - mae: 0.0449 - val_loss: 0.0180 - val_mae: 0.0926
Epoch 10/10
372/372 [==============================] - 48s 130ms/step - loss: 0.0041 - mae: 0.0449 - val_loss: 0.0189 - val_mae: 0.0946
(1825, 96, 16) (1825, 240, 1)
58/58 [==============================] - 3s 44ms/step - loss: 0.0218 - mae: 0.0991
long prediction:
Average MSE: 0.02142066601663828 ± 0.0004988090330530718
Average MAE: 0.09839885085821151 ± 0.0015828749628815318
```

Transformer

```
transformer_model_long= TransformerModel(sequence_length, X_train_long.shape[2], embed_dim, num_heads, ff_dim, num_layers, output_dim)
for i in range(num_trials):
    print(f"Trial {i}")
    mse, mae = train_and_evaluate(transformer_model_long, (X_train_long, y_train_long), (X_test_long, y_test_long), num_epochs, batch_size)
    mse_results.append(mse)
    mae_results.append(mae)

# 输出平均值和标准差
print(f"Average MSE: {np.mean(mse_results):.4f} ± {np.std(mse_results):.4f}")
print(f"Average MAE: {np.mean(mae_results):.4f} ± {np.std(mae_results):.4f}")
```

```
al_loss: 0.0228 - val_mean_squared_error: 0.0228 - val_mean_absolute_error: 0.1085
Epoch 6/10
372/372 [==============================] - 53s 143ms/step - loss: 0.0041 - mean_squared_error: 0.0041 - mean_absolute_error: 0.0446 - v
al_loss: 0.0243 - val_mean_squared_error: 0.0243 - val_mean_absolute_error: 0.1121
Epoch 7/10
372/372 [==============================] - 54s 144ms/step - loss: 0.0041 - mean_squared_error: 0.0041 - mean_absolute_error: 0.0446 - v
al_loss: 0.0234 - val_mean_squared_error: 0.0234 - val_mean_absolute_error: 0.1100
Epoch 8/10
372/372 [==============================] - 53s 143ms/step - loss: 0.0041 - mean_squared_error: 0.0041 - mean_absolute_error: 0.0445 - v
al_loss: 0.0237 - val_mean_squared_error: 0.0237 - val_mean_absolute_error: 0.1109
Epoch 9/10
372/372 [==============================] - 53s 143ms/step - loss: 0.0041 - mean_squared_error: 0.0041 - mean_absolute_error: 0.0448 - v
al_loss: 0.0231 - val_mean_squared_error: 0.0231 - val_mean_absolute_error: 0.1092
Epoch 10/10
372/372 [==============================] - 53s 143ms/step - loss: 0.0041 - mean_squared_error: 0.0041 - mean_absolute_error: 0.0445 - v
al_loss: 0.0225 - val_mean_squared_error: 0.0225 - val_mean_absolute_error: 0.1079
58/58 [==============================] - 4s 47ms/step
Average MSE: 0.0393 ± 0.0017
Average MAE: 0.1475 ± 0.0042
```

Him-model

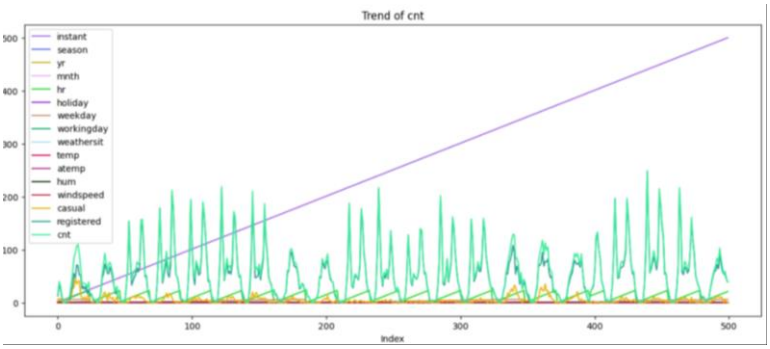
```
# 输出平均值和标准差
print(f"Average MSE: {np.mean(him_model_mse_results_long):.4f} ± {np.std(him_model_mse_results_long):.4f}")
print(f"Average MAE: {np.mean(him_model_mae_results_long):.4f} ± {np.std(him_model_mae_results_long):.4f}")# 模型预测
y_pred_long_him_model = him_model_long.predict(X_test_long)

# 反归一化预测值与真实值
y_pred_long_him_model = scaler_y_long.inverse_transform(y_pred_long_him_model.reshape(-1, 1))
y_test_long_him_model = scaler_y_long.inverse_transform(y_test_long.reshape(-1, 1))
```

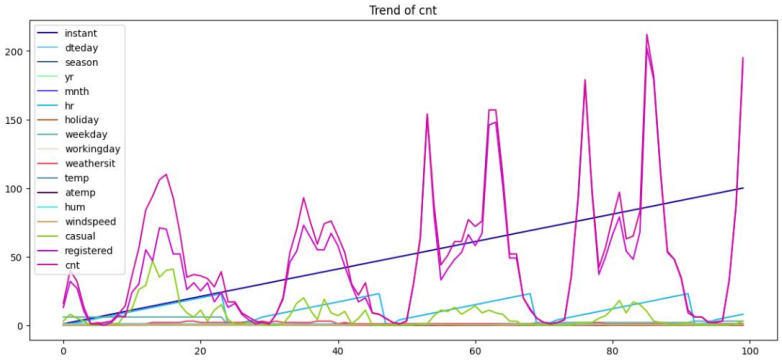
```
Epoch 5/10
372/372 [==============================] - ETA: 0s - loss: 0.0091 - mean_absolute_error: 0.0677
Epoch 5: ReduceLROnPlateau reducing learning rate to 0.0005000000237487257.
372/372 [==============================] - 86s 231ms/step - loss: 0.0091 - mean_absolute_error: 0.0677 - val_loss: 0.0220 - val_mean_ab
solute_error: 0.1085 - lr: 0.0010
Epoch 6/10
372/372 [==============================] - 86s 231ms/step - loss: 0.0090 - mean_absolute_error: 0.0673 - val_loss: 0.0221 - val_mean_ab
solute_error: 0.1087 - lr: 5.0000e-04
Epoch 7/10
372/372 [==============================] - ETA: 0s - loss: 0.0090 - mean_absolute_error: 0.0673Restoring model weights from the end of
the best epoch: 2.
372/372 [==============================] - 87s 234ms/step - loss: 0.0090 - mean_absolute_error: 0.0673 - val_loss: 0.0221 - val_mean_ab
solute_error: 0.1088 - lr: 5.0000e-04
Epoch 7: early stopping
58/58 [==============================] - 4s 70ms/step - loss: 0.0267 - mean_absolute_error: 0.1134
Average MSE: 0.0269 ± 0.0002
Average MAE: 0.1136 ± 0.0008
58/58 [==============================] - 5s 66ms/step
```

分析：在长期预测中 Him-model 模型的表现最好，其次是 LSTM 模型，根据 STD 分析其稳定性 Him-model 较另外两个模型而言更好，Him-model 不仅能捕捉局部的邻居之前的时序关系，也能考虑到全局的联系，对于较长的时序预测更加准确，表现最好。

数据分析

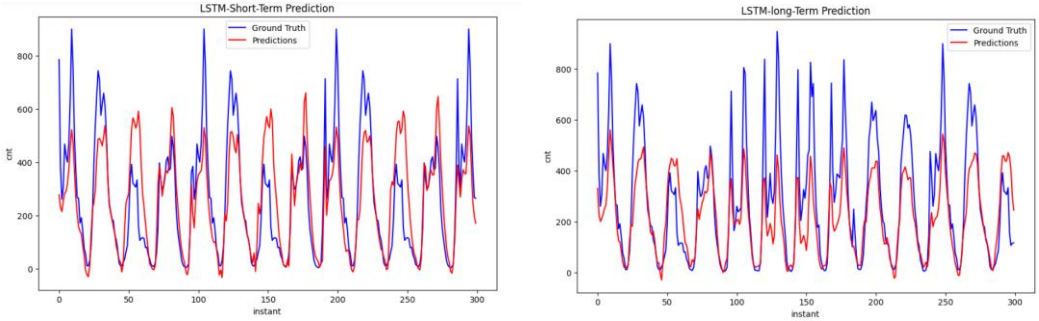


通过折线图的形式绘制各特征之间的联系，可以看到数据呈现出周期性的变化与 weekday 特征相符，也就是说，每周工作日要较非工作日租赁数量更多。season 的变化似乎有一些波动，但整体上对数量的影响不如其他因素显著。Yr 也就是年份的变化呈现出轻微的增长趋势，可能暗示着随着时间的推移，数量有所增加。



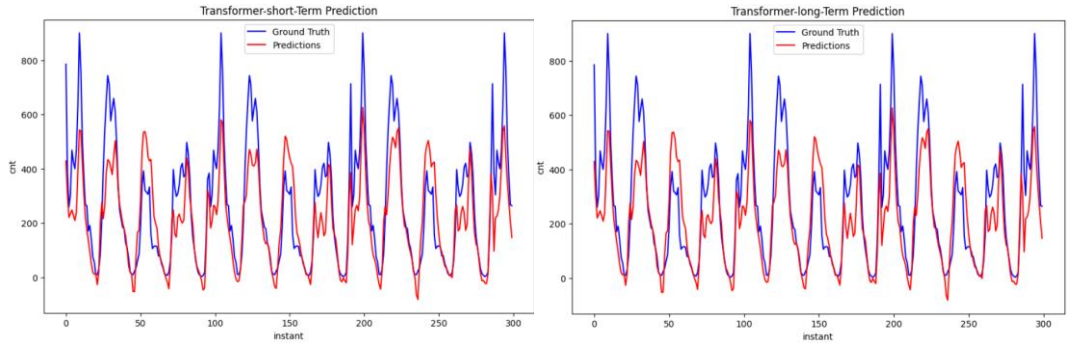
同时，租赁数量由注册用户租赁数量和游客用户租赁数量相加组成，注册用户租赁数量占总租赁数量的比重非常大，所以自行车租赁公司可以在注册用户上多发布广告和优惠政策来鼓励租赁，效果更好。而 weather 天气状况的波动往往导致租赁数量的大波动，这可能是因为恶劣天气导致的活动减少或其他原因。Temp 与 atemp 相似，温度的变化有一定的波动性，高温时数量可能会下降。其他因素不明显。

拟合情况可视化

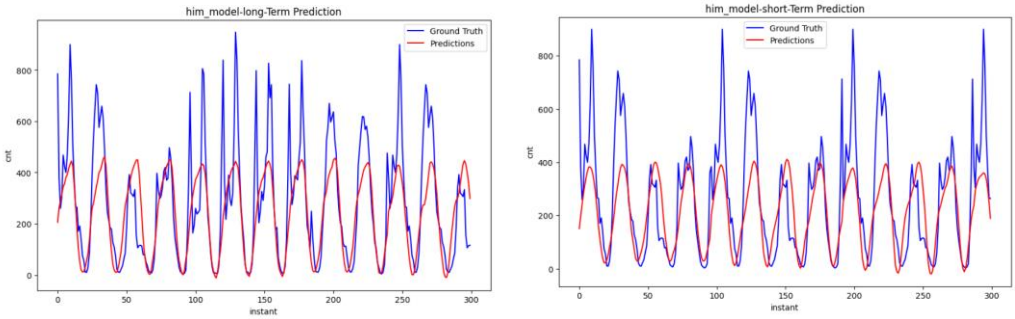


对于 LSTM 模型，它能较好地拟合短期预测中的峰值，对于波动的捕捉能力

很好，基本拟合，但是对于长期预测则只表现出周期性的波动，峰值拟合比较差，也就是说他很难捕捉更长周期间那些跨越月，年的特征波动，但是对于一周内工作日间的预测则很准确，这也是 LSTM 自身固有的优势。



对于 Tranformer 模型，在拟合表现中发现它所能预测到的低谷峰值要较真实值更低，同时它的高峰值也基本较真实值更低，但是对于长期预测的捕捉能力要较 LSTM 好很多，能做到基本拟合，这是 Transformer 对于全局特征的捕捉能力的帮助。



对于改良后的 Him-model,要较前两者更加稳定，但是表现出更明显的周期变化，周期间的波动差异小，没能捕捉到小区间内的差异，有可能是过拟合了，但是从侧面突出，改良后的 Him-model 实质上很适合去除噪音来捕捉底层的规律，因为租赁数量就是与一天内的白天租赁数量更多，但是没能描绘出一天中上班和下班两个高峰期。

4.讨论与总结

本研究对于自行车租赁数量预测问题使用三种方法（LSTM,Transformer，Him-model）进行解决，其中 Him-model 为自主提出的改良模型，综合了嵌入层，LSTM 层和 Transformer 层来串行堆叠而成，既考虑了短期预测中对于邻居上下文的依赖，也考虑到全局的信息，实验证明其具有较好的拟合能力，能捕捉到底层规律。其次，本研究对三种方法实现了 96 小时的短期预测和 240 小时的长期预测，讨论模型的优缺点，并提出改进方向,将三个模型进行比较，分析模型在不同预测时间范围（短期和长期）上的表现。同时探讨了模型性能的影响因素，对数据进行了分析和处理，将模型预测结果与真实值进行可视化对比，分析模型的预测精度和误差来源发现 LSTM 对于局部信息捕捉能力最好，Him-model 在长期预测中表现最好，其次是 Transformer，这两个模型更能捕捉长跨度的信息。

本研究提出的 Him-model 是通过串行堆叠实现的，可以考虑并行联合学习，

采纳更好的一方，给多个模型附以权重来综合多个模型的能力来拟合。

代码：已上传至 GitHub 仓库：https://github.com/Harzerr/machine-learning-final-work.git ，如有不足，欢迎指正！

参考文献

[1] Yu Y , Si X , Hu C ,et al.A Review of Recurrent Neural Networks: LSTM Cells and Network Architectures[J].Neural Computation, 2019, 31(7):1235-1270.DOI:10.1162/neco_a_01199.

[2] Staudemeyer R C , Rothstein Morris E .Understanding LSTM -- a tutorial into Long Short-Term Memory Recurrent Neural Networks[J].arXiv, 2019.DOI:10.48550/arXiv.1909.09586.

[3] Vaswani A , Shazeer N , Parmar N ,et al.Attention Is All You Need[J].arXiv, 2017.DOI:10.48550/arXiv.1706.03762.

[4] Wang Y , Huang M , Zhu X ,et al.Attention-based LSTM for Aspect-level Sentiment Classification[C]//Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing.2016.DOI:10.18653/v1/D16-1058.

[5] Teng Y , Liu J , Wu K .Time Series Prediction Based on LSTM and High-Order Fuzzy Cognitive Map with Attention Mechanism[J].Neural Processing Letters, 2024, 56(5).DOI:10.1007/s11063-024-11666-1.

[6] Hu Z , Gao Y , Ji S ,et al.Improved multistep ahead photovoltaic power prediction model based on LSTM and self-attention with weather forecast data[J].Applied Energy, 2024, 359.DOI:10.1016/j.apenergy.2024.122709.