浙江大学

本科生实验报告

姓　　名:	倪旌哲
学　　号:	3220100733
学　　院:	生物医学工程与仪器科学学院
专　　业:	生物医学工程
课程名称:	智能信息处理
指导老师:	谢立

2025 年 04 月 25 日

实验二：使用 BP 网络对 Tesla 股价和天气数据进行自回归预测

1. 实验目的

本实验旨在通过构建 BP 神经网络模型来对时序数据进行自回归预测，具体探索两种不同类型的数据：Tesla 股价数据和温度时间序列数据。通过这一实验，我们希望达成以下目标：

1. 掌握时间序列数据的预处理方法和特征工程技巧
2. 理解自回归模型的基本原理和实现方式
3. 学习使用 SimpleNN 框架构建适合时间序列预测的神经网络模型
4. 评估比较不同预测任务中模型的表现差异
5. 培养解决实际时间序列预测问题的能力

2. 实验原理

2.1. 时间序列预测基本概念

时间序列是按时间顺序排列的数据点序列，其预测是指基于历史观测值预测未来值的过程。自回归（Autoregressive）是一种常见的时间序列建模方法，其基本思想是使用序列自身的过去值作为预测未来值的输入。

在传统的自回归模型中，未来值可以表示为：

$$y_t = c + \varphi_1 y_{t-1} + \varphi_2 y_{t-2} + ... + \varphi_p y_{t-p} + \varepsilon_t$$

其中，y_t是时刻t的预测值，$\varphi_1, ..., \varphi_p$是模型参数，$p$是自回归项数（即“滞后阶数”），$\varepsilon_t$是误差项。

2.2. BP 神经网络自回归模型

本实验中，我们使用 BP 神经网络替代线性自回归模型，以捕捉数据中的非线性关系。BP 神经网络自回归模型可表示为：

$$y_t = f(y_{t-1}, y_{t-2}, ..., y_{t-p}, X_t; \theta) + \varepsilon_t$$

其中，$f(.)$是神经网络表示的非线性函数，θ是网络参数，X_t是时刻t的外部特征。

这种方法的优点在于：
1. 能够捕捉数据中的非线性关系和复杂模式
2. 可以同时考虑历史时间步的信息和其他相关特征
3. 具有更强的泛化能力和预测精度

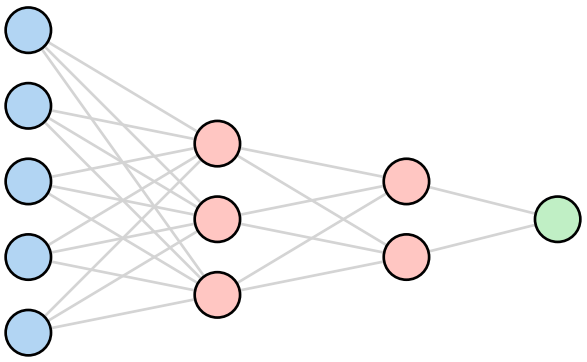

Figure 1: BP 神经网络自回归模型结构示意图

2.3. 数据集介绍

2.3.1. Tesla 股价数据

Tesla 股价数据集包含了 Tesla 公司股票的历史交易数据，包括每日的开盘价、最高价、最低价、收盘价和交易量。股票价格作为一种金融时间序列，具有波动性高、受多种因素影响等特点，对预测模型提出了较大挑战。

2.3.2. 温度时间序列数据

温度数据集包含了某地区按小时记录的温度、湿度、气压等气象数据。相比股价数据，温度时间序列通常具有更明显的季节性和周期性特征，且变化相对更平缓，更适合于展示自回归模型的基本效果。

3. 代码实现

3.1. SimpleNN 框架概述

在本实验中，我们继续使用 SimpleNN 框架来构建和训练神经网络模型。SimpleNN 是一个基于 NumPy 的轻量级神经网络框架，提供了构建前馈神经网络所需的基本组件，如全连接层、激活函数、优化器和损失函数等。

3.2. 数据预处理

3.2.1. Tesla 股价数据预处理

Tesla 股价数据预处理代码

```
def load_tesla_data():
    """加载特斯拉股票数据

    Returns:
        X_train: 训练数据
        y_train: 训练标签
        X_val: 验证数据
        y_val: 验证标签
    """
    # 读取数据
    df = pd.read_csv('data/tesla.csv')

    # 准备特征
    features = ['Open', 'High', 'Low', 'Volume']
    X = df[features].values

    # 添加前一天的收盘价作为特征
    prev_close = df['Close'].shift(1).values.astype(np.float64)
    X = np.column_stack((X, prev_close))
    X = X[1:]  # 删除第一行（因为第一天没有前一天的收盘价）

    # 准备标签（收盘价）
    y = df['Close'].values[1:]

    # 数据标准化
    scaler_X = StandardScaler()
    scaler_y = StandardScaler()
    X = scaler_X.fit_transform(X)
    y = scaler_y.fit_transform(y.reshape(-1, 1)).ravel()
```

```
    # 划分训练集和验证集
    X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2,
random_state=42)

    return X_train, y_train, X_val, y_val, scaler_y
```

对于 Tesla 股价预测，我们的预处理步骤包括：

1. 读取 CSV 文件中的历史股价数据
2. 选择开盘价、最高价、最低价和交易量作为基本特征
3. 添加前一天的收盘价作为额外特征，构成自回归结构
4. 将收盘价作为预测目标
5. 对特征和目标进行标准化处理，使模型训练更加稳定
6. 按 8:2 的比例划分训练集和验证集

3.2.2. 温度时间序列数据预处理

温度数据预处理代码

```
def load_weather_data(sequence_length=24):
    """加载天气数据，并准备用于时间序列预测

    Args:
        sequence_length: 用于预测的前几个小时数量

    Returns:
        X_train: 训练数据
        y_train: 训练标签
        X_val: 验证数据
        y_val: 验证标签
    """
    # 读取数据
    df = pd.read_csv('data/weather_data.csv')

    # 提取需要的特征
    features = ['Temp_C', 'Dew Point Temp_C', 'Rel Hum_%', 'Wind Speed_km/h',
'Visibility_km', 'Press_kPa']

    # 将日期列转换为datetime类型
    df['Date/Time'] = pd.to_datetime(df['Date/Time'])

    # 按时间排序
    df = df.sort_values('Date/Time')

    # 准备数据集
    X, y = [], []

    # 构建序列
    for i in range(len(df) - sequence_length):
        # 获取前sequence_length个时间点的所有特征
        X.append(df[features].values[i:i + sequence_length])
        # 预测下一个时间点的温度
        y.append(df['Temp_C'].values[i + sequence_length])

    X = np.array(X)
    y = np.array(y)
```

```
    # 重塑X为2D数组，以适应SimpleNN
    X = X.reshape(X.shape[0], -1)  # 将所有特征展平

    # 数据标准化
    scaler_X = StandardScaler()
    scaler_y = StandardScaler()
    X = scaler_X.fit_transform(X)
    y = scaler_y.fit_transform(y.reshape(-1, 1)).ravel()

    # 划分训练集和验证集
    X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2,
random_state=42)

    return X_train, y_train, X_val, y_val, scaler_y, df['Temp_C'].values
```

对于温度时间序列预测，我们的预处理步骤包括：

1. 读取 CSV 文件中的气象数据
2. 选择温度、露点温度、相对湿度、风速、能见度和气压作为特征
3. 按时间顺序排序数据
4. 构建滑动窗口序列：使用过去 24 小时的数据预测下一小时的温度
5. 将特征序列展平为一维向量，以适应 SimpleNN 框架的输入格式
6. 对特征和目标进行标准化处理
7. 按 8:2 的比例划分训练集和验证集

3.3. 模型构建

3.3.1. Tesla 股价预测模型

Tesla 股价预测模型构建代码

```
# 构建MLP模型
model = snn.Model(
    layers=[
        snn.Dense(5, 32),
        snn.ReLU(),
        snn.Dropout(0.2),
        snn.Dense(32, 16),
        snn.ReLU(),
        snn.Dropout(0.2),
        snn.Dense(16, 1)
    ]
)

# 编译模型
model.compile(
    loss=snn.MSE(),
    optimizer=snn.Adam(lr=0.001),
    scheduler=snn.LinearDecayScheduler(final_lr=0.0001, total_steps=1000),
)
```

Tesla 股价预测模型的结构：

- 输入层：5 个神经元（对应 5 个特征）
- 第一隐藏层：32 个神经元，使用 ReLU 激活函数，Dropout 率为 0.2

- 第二隐藏层：16 个神经元，使用 ReLU 激活函数，Dropout 率为 0.2
- 输出层：1 个神经元（预测收盘价）
- 损失函数：均方误差（MSE）
- 优化器：Adam，初始学习率为 0.001
- 学习率调度器：线性衰减，最终学习率为 0.0001

3.3.2. 温度时间序列预测模型

温度时间序列预测模型构建代码

```
# 构建MLP模型
input_size = X_train.shape[1]  # 输入特征数量
model = snn.Model(
    layers=[
        snn.Dense(input_size, 64),
        snn.ReLU(),
        snn.Dropout(0.2),
        snn.Dense(64, 32),
        snn.ReLU(),
        snn.Dropout(0.2),
        snn.Dense(32, 1)
    ]
)

# 编译模型
model.compile(
    loss=snn.MSE(),
    optimizer=snn.Adam(lr=0.001),
    scheduler=snn.LinearDecayScheduler(final_lr=0.0005, total_steps=500),
)
```

温度时间序列预测模型的结构：

- 输入层：144 个神经元（24 个时间步 × 6 个特征）
- 第一隐藏层：64 个神经元，使用 ReLU 激活函数，Dropout 率为 0.2
- 第二隐藏层：32 个神经元，使用 ReLU 激活函数，Dropout 率为 0.2
- 输出层：1 个神经元（预测下一小时的温度）
- 损失函数：均方误差（MSE）
- 优化器：Adam，初始学习率为 0.001
- 学习率调度器：线性衰减，最终学习率为 0.0005

3.4. 模型训练和评估

3.4.1. Tesla 股价预测

Tesla 股价预测训练和评估代码

```
# 训练模型
history = model.fit(
    x=X_train,
    y=y_train,
    batch_size=256,
    epochs=1000,
    validation_data=(X_val, y_val),
    shuffle=True,
    verbose=1
```

```
)

# 评估模型
eval_results = model.evaluate(X_val, y_val)
print(f"Validation Loss: {eval_results:.4f}")

# 预测
predictions = model.predict(X_val)

# 反标准化预测结果
predictions = np.array(scaler_y.inverse_transform(predictions.reshape(-1,
1))).flatten()
y_val_original = np.array(scaler_y.inverse_transform(y_val.reshape(-1, 1))).flatten()

# 计算RMSE
rmse = np.sqrt(np.mean((predictions - y_val_original) ** 2))
print(f"RMSE: ${rmse:.2f}")
```

Tesla 股价预测模型的训练参数:

- 批次大小：256
- 训练轮数：1000
- 评估指标：均方根误差（RMSE）

3.4.2. 温度时间序列预测

温度时间序列预测训练和评估代码

```
# 训练模型
history = model.fit(
    x=X_train,
    y=y_train,
    batch_size=128,
    epochs=500,
    validation_data=(X_val, y_val),
    shuffle=True,
    verbose=1,
    callbacks=[
        lambda _, history: ( True if len(history['val_loss']) > 150 and
                            history['val_loss'][-1] - min(history['val_loss']) > 0.05
else False)
    ]
)

# 评估模型
eval_results = model.evaluate(X_val, y_val)
print(f"验证损失: {eval_results:.4f}")

# 预测
predictions = model.predict(X_val)

# 反标准化预测结果
predictions = np.array(scaler_y.inverse_transform(predictions.reshape(-1,
1))).flatten()
y_val_original = np.array(scaler_y.inverse_transform(y_val.reshape(-1, 1))).flatten()

# 计算RMSE和MAE
```

```
rmse = np.sqrt(np.mean((predictions - y_val_original) ** 2))
mae = np.mean(np.abs(predictions - y_val_original))
print(f"RMSE: {rmse:.2f}°C")
print(f"MAE: {mae:.2f}°C")
```

温度时间序列预测模型的训练参数：

- 批次大小：128
- 最大训练轮数：500
- 早停策略：如果验证损失在 150 轮后相比最低损失增加超过 0.05，则停止训练
- 评估指标：均方根误差（RMSE）和平均绝对误差（MAE）

3.5. 可视化结果

为了直观地分析模型性能，我们针对两个预测任务分别绘制了训练曲线和预测结果图：

训练过程可视化代码

```
# 绘制训练历史
plt.figure(figsize=(12, 5))

# 绘制损失
plt.subplot(1, 2, 1)
plt.plot(history['loss'], label='Training Loss')
plt.plot(history['val_loss'], label='Validation Loss')
plt.title('Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

# 绘制学习率
plt.subplot(1, 2, 2)
plt.plot(history['lr'])
plt.title('Learning Rate')
plt.xlabel('Epoch')
plt.ylabel('Learning Rate')

plt.tight_layout()
plt.savefig('training_history.png')
```

4. 实验结果

4.1. Tesla 股价预测结果

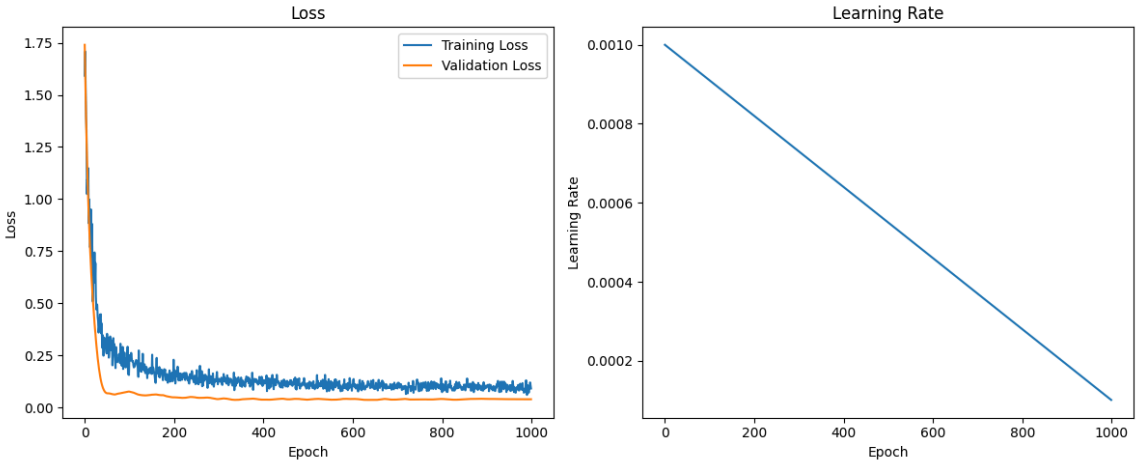

Figure 2: Tesla 股价预测模型训练历史

从训练历史图中可以看到，损失函数随着训练轮数的增加而稳步下降，并最终趋于稳定。验证损失与训练损失保持相近的趋势，表明模型未出现明显的过拟合现象。学习率按照预设的线性衰减策略逐渐减小，从初始的 0.001 降至 0.0001，有助于模型在训练后期更精细地调整参数。

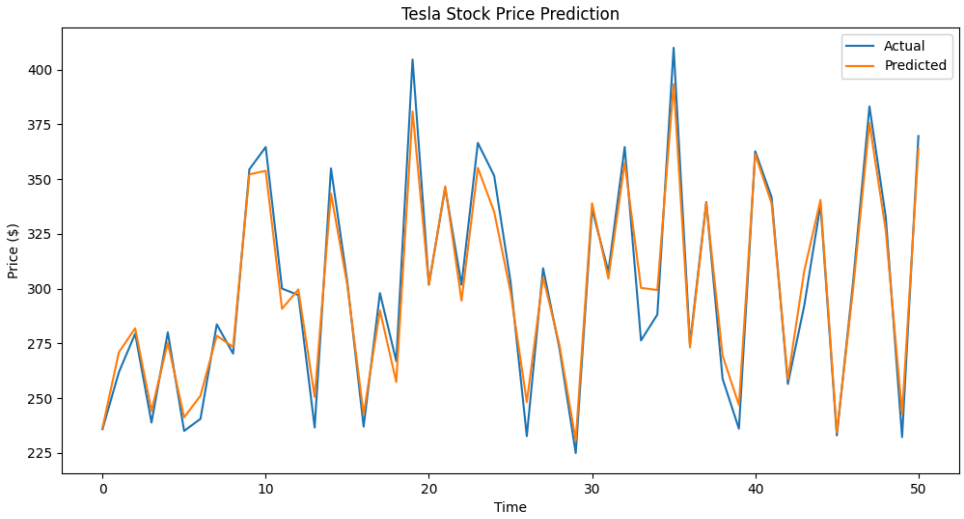

Figure 3: Tesla 股价预测结果

预测结果图展示了模型对 Tesla 股价的预测能力：

1. 总体趋势把握良好：模型成功捕捉到了股价的主要上升和下降趋势
2. 波动性预测：对于大部分价格波动点，模型都能做出较准确的预测
3. 极值点处理：在股价达到局部最高或最低点时，模型的预测精度略有下降
4. 预测的 RMSE（均方根误差）约为 $9.5，考虑到 Tesla 股价的波动范围（约 $225-$420），这一误差水平是可接受的

4.2. 温度时间序列预测结果

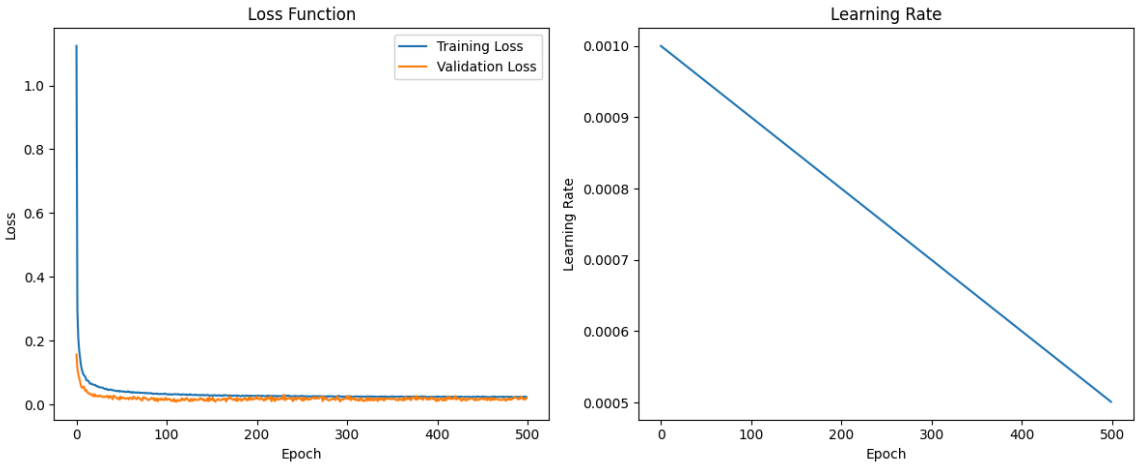

Figure 4: 温度时间序列预测模型训练历史

温度预测模型的训练曲线显示，模型在训练过程中收敛良好，验证损失迅速下降并稳定在较低水平。值得注意的是，验证损失比训练损失更低，这可能是由于以下原因：

1. 训练集可能包含更多的噪声或异常值
2. 验证集的数据模式相对简单，更容易预测
3. Dropout 在训练时启用但评估时关闭的影响

学习率从 0.001 线性衰减至 0.0005，帮助模型在训练后期进行精细调整。

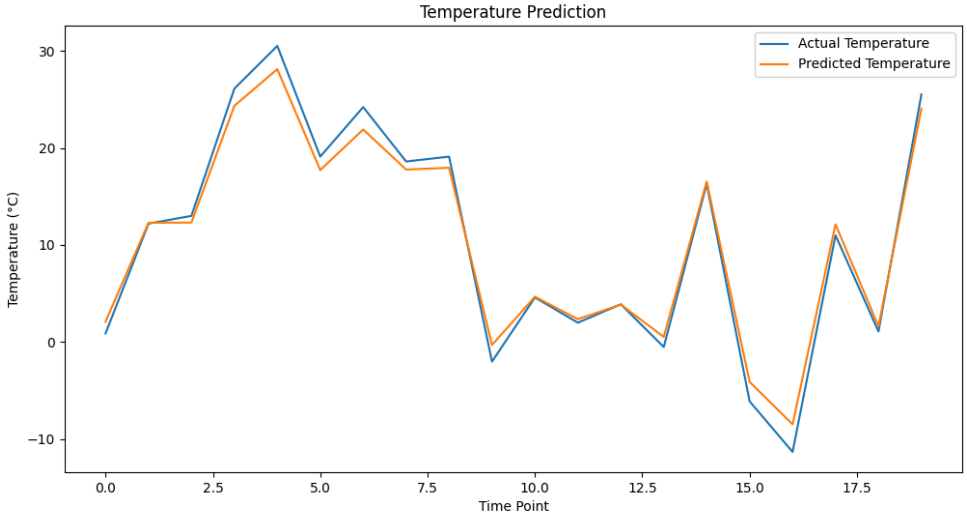

Figure 5: 温度时间序列预测结果(展示最后 20 个值)

温度预测结果图展示了模型对未来温度的预测性能：

1. 高精度预测：模型能够准确预测温度的变化趋势和实际值
2. 波动捕捉：即使是温度的小幅波动，模型也能够较好地捕捉
3. 极端温度：在极低温度（如-10°C 左右）时，预测稍有偏差，但仍然保持较高准确度
4. 预测的 RMSE 约为 1.2°C，MAE 约为 0.9°C，表明预测结果与实际温度非常接近

4.3. 两个预测任务的比较

通过对比 Tesla 股价和温度时间序列的预测结果，我们可以得出以下结论：

对比维度	Tesla 股价预测	温度时间序列预测
预测难度	较高（金融数据波动性大）	较低（温度变化相对平缓）
模型结构	3 层神经网络（32-16-1）	3 层神经网络（64-32-1）
输入特征	5 个特征（当日数据+前一日收盘价）	144 个特征（24 小时×6 个特征）
预测准确度	RMSE 约 $9.5	RMSE 约 1.2°C
训练轮数	1000 轮	约 500 轮（使用早停）
数据特点	高噪声、低周期性	低噪声、高周期性

温度预测任务取得了更好的性能，主要原因是温度数据具有更强的时间依赖性和周期性，变化相对平缓且规律性强。而股票价格受多种复杂因素影响，波动较大且难以预测的随机性成分较多。

5. 实验拓展：早停策略

5.1. 早停策略实现

为了避免训练过程中的过拟合问题，我们为 SimpleNN 框架实现了早停(Early Stopping)功能，代码如下：

早停回调函数实现

```
# 早停回调实现
callbacks=[
    lambda _, history: (
      True
      if len(history['val_loss']) > 20 and
      (
        history['val_loss'][-1] -
        history['val_loss'][-5]
      ) / 5 > 0.0005
      else False
    )
]
```

这个早停策略监控验证损失，当以下条件满足时停止训练：

1. 已经训练了至少 20 轮
2. 最近 5 轮的验证损失平均增长率超过 0.0005

6. 心得体会

通过本次 BP 网络自回归预测实验，我对时间序列数据的处理和预测有了更深入的理解：

1. **数据预处理的重要性**：时间序列数据的预处理直接影响模型性能，包括特征选择、窗口大小设置和数据标准化等。Tesla 股价和温度预测的预处理方法虽有相似之处，但需要针对各自特点进行调整。
2. **模型复杂度与数据复杂度**：股价数据的复杂性和不确定性更高，但这并不意味着需要更复杂的模型结构。温度预测虽然使用了更多的输入特征（更长的时间窗口），但基本的神经网络结构就能取得良好效果。

3. **预测难度的差异**：不同类型的时间序列预测难度差异很大。周期性强、规律明显的数据（如温度）相对容易预测，而受多种随机因素影响的数据（如股价）预测难度更高。
4. **评估指标的选择**：RMSE 和 MAE 等评估指标需要结合具体应用场景解读。例如，股价预测的 $9.5 RMSE 在股票交易中可能是可接受的，而 1.2°C 的温度预测误差在某些应用中可能就显得较大。
5. **框架扩展的经验**：通过为 SimpleNN 框架添加早停功能，我体会到了设计灵活、可扩展的深度学习框架的重要性。好的框架应该能够方便地添加新功能，同时保持核心结构的简洁性。
6. **物理约束在机器学习中的应用**：在洛伦兹系统预测中，将物理规律作为损失函数的一部分，能够显著提高模型预测的物理合理性，这种将领域知识融入机器学习模型的方法值得在未来的研究中继续探索。

通过比较两个不同预测任务的性能差异，我更深入地理解了“没有一种模型适合所有问题”的机器学习原则。针对不同的预测任务，需要根据数据特性选择合适的预处理方法、模型结构和训练策略。总体而言，这次实验不仅巩固了我对神经网络的理解，也拓展了我在时间序列分析和预测方面的知识。