时间序列分析期末大作业实验报告

田浩乐 PB20010379 王昊然 PB20010382 孙守恒 PB20010376

2024 年 1 月 23 日

1 简介

1.1 实验摘要

本实验旨在构建一个模型，用于预测期货/期权合约的每日结算价。通过分析中金所官网提供的数据，我们利用近月 IO 期权认购 Call 和认沽 Put 的行权价、IF 近月合约结算价、距离行权日的天数，以及期权每日结算价等金融数据。我们采用 LSTM 和 XGBoost 算法构建预测模型，以期达到通过 IF 近月合约的行权价，IO 近月合约与 IF 价差，距离行权日的天数拟合出 IO 的 Call 和 Put 结算价。

1.2 近月合约、两个期权的结算价的相关性、结算价和距离行权日期的关系

本部分深入探讨了近月合约、认购（Call）和认沽（Put）期权的结算价之间的相关性，以及这些价格与距离行权日期的关系。近月合约指的是距离当前日期最近的交割月份的合约。在本实验中，我们关注的是近月 IO 期权合约及其相对于 IF 近月合约的行权价差异。

首先，我们分析了认购和认沽期权的结算价与 IF 近月合约结算价之间的关系。期权的内在价值和时间价值共同影响期权的结算价。对于认购期权而言，当 IF 合约价高于行权价时，其内在价值增加，从而提高了认购期权的结算价。相反，对于认沽期权，当 IF 合约价低于行权价时，其结算价则会增加。

其次，我们考察了结算价与距离行权日天数的关系。随着行权日的临近，期权的时间价值逐渐降低，这通常会对其结算价产生影响。特别是在行权日临近时，这种影响更为显著。此外，市场波动性、利率和股息等因素也会影响期权价值的时间衰减。

2 数据处理

2.1 爬取数据

数据爬取是本实验的关键初步步骤，旨在从中金所官网（`http://www.cffex.com.cn/rtj/`）获取必要的期货和期权合约数据。我们主要关注的是每日结算价等关键金融指标。

具体的爬取过程遵循以下步骤：

1. ** 网站访问与解析 **：首先，我们编写一个自动化脚本来访问中金所的相关网页。使用 Python 的 requests 库来发送 HTTP 请求，获取页面内容。随后，利用 BeautifulSoup 库解析 HTML 页面，以提取所需数据。

2. ** 数据定位 **：在网页的 HTML 结构中定位到包含期货/期权合约每日结算价等数据的特定部分。这通常涉及寻找包含数据的表格或其他 HTML 元素。

3. ** 数据提取 **：从定位的元素中提取所需的数据，如 IF 近月合约结算价、IO 期权的行权价、结算价和到期日等信息。这一步可能需要处理不同格式的数据，例如，将字符串格式的日期转换为 Python 日期对象。

4. ** 数据存储 **：提取的数据被存储在适当的格式中，例如 CSV 或数据库，以便于后续的数据清洗和分析。确保数据的完整性和准确性是此步骤的关键。

2.2 获取相关数据

为了建立有效的期权结算价推导模型，我们首先需要获取相关数据。这包括期权的行权日、IF 近月合约的每日结算价、IO 行权价与 IF 合约价的差额，以及期权的每日结算价。

使用 Python 编写的脚本自动化了数据获取过程。我们定义了两个主要的函数：‘*get_strike_day_calendar*‘ 和 ‘*get_IO_C*‘。‘*get_strike_day_calendar*‘ 函数生成一个日期范围内所有期权的行权日列表。它使用了 pandas 的 $WeekOfMonth$ 来获取每个月的第三个星期五，即期权的标准行权日。然后，结合 ‘*chinese_calendar*‘ 库来检查是否有任何行权日与中国的法定假日冲突，如有冲突，则相应地调整日期。

get_IO_C 函数用于从指定的文件中提取和处理期权数据。它首先读取包含期权合约信息的 CSV 文件，然后提取特定日期的 IF 近月合约数据。对于认购（Call）或认沽（Put）期权，它计算行权价与 IF 合约价的差额、距离行权日的天数，并将这些数据与期权的每日结算价结合起来，形成模型所需的输入数据集。

2.3 数据清洗

数据清洗是确保模型准确性的关键步骤。在我们的案例中，数据清洗包括以下几个方面：

1. ** 剔除节假日数据 **：根据中国的法定假日日历，我们从数据集中剔除了所有节假日的记录。这是因为在节假日，金融市场不开市，因此相关数据可能不具代表性或缺失。

2. ** 处理缺失值 **：我们检查了数据集中的缺失值。对于缺失的数据，我们采取了适当的方法进行处理，例如使用前一交易日的数据填补或者根据周围数据进行插值。

3. ** 数据类型转换和格式化 **：确保所有数据列的类型正确，例如将日期字符串转换为日期类型，确保数值类型正确等。

3 理论背景和数学原理

3.1 LSTM

LSTM 网络是一种特殊类型的递归神经网络（RNN），专门用于解决 RNN 在处理长序列数据时的短期记忆问题。LSTM 通过其独特的结构单元（称为 LSTM 单元）来维护和更新单元状态。一个标准的 LSTM 单元包括遗忘门（forget gate）、输入门（input gate）和输出门（output gate）。

每个门的运算可以用以下公式表示：

1. 遗忘门（Forget Gate）：

$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$ 其中，f_t 是在时间步 t 的遗忘门的输出，σ 是 sigmoid 激活函数，W_f 是遗忘门的权重矩阵，h_{t-1} 是前一个时间步的隐藏状态，x_t 是当前时间步的输入，b_f 是遗忘门的偏置项。

2. 输入门（Input Gate）：

$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$ $\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$ 其中，i_t 是输入门的输出，$\tilde{C}_t$ 是候选状态值，W_i 和 W_C 分别是输入门和状态更新的权重矩阵，b_i 和 b_C 是相应的偏置项。

3. 单元状态更新:

$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$ 这里，C_t 是当前时间步的单元状态，C_{t-1} 是前一个时间步的单元状态。

4. 输出门（Output Gate）：

$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$ $h_t = o_t * \tanh(C_t)$ 在这里，o_t 是输出门的输出，h_t 是当前时间步的隐藏状态，W_o 是输出门的权重矩阵，b_o 是偏置项。

在这些公式中，σ 代表 sigmoid 函数，$*$ 代表元素级别的乘法，tanh 是双曲正切激活函数。权重矩阵 W_f, W_i, W_C, W_o 的大小依赖于输入特征的维度和隐藏层的维度。

LSTM 通过这些门控制机制有效地允许信息随时间流动，同时避免梯度消失或爆炸的问题。

3.2 XGBoost

XGBoost（eXtreme Gradient Boosting）是一种先进的机器学习算法，用于回归和分类问题。它基于梯度提升决策树（GBDT）的原理，通过迭代地添加决策树来最小化预测误差。

XGBoost 的目标函数是损失函数和正则化项的和：

$$\text{Obj}(\Theta) = \sum_{i=1}^{n} l(y_i, \hat{y}_i) + \sum_{k=1}^{K} \Omega(f_k)$$

其中，l 是损失函数，衡量预测值 $\hat{y}_i$ 和真实值 y_i 之间的差距；n 是样本数量；f_k 是第 k 棵树；K 是树的总数；Ω 是正则化项。

正则化项 $\Omega(f)$ 由树的复杂度决定：

$$\Omega(f) = \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2$$

其中，T 是树的叶子节点数，w_j 是叶子节点 j 的权重，γ 和 λ 是正则化系数。γ 控制树的数量，λ 控制叶子节点权重的影响。

在每次迭代中，XGBoost 为损失函数添加一个新的树 f_t，以最小化以下目标（残差 + 正则项）：

$$\text{Obj}^{(t)} = \sum_{i=1}^{n} l(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)) + \Omega(f_t)$$

其中，$\hat{y}_i^{(t-1)}$ 是第 $t-1$ 次迭代后的预测。

XGBoost 使用二阶泰勒展开来近似损失函数，使得优化更加准确和快速：

$$\text{Obj}^{(t)} \approx \sum_{i=1}^{n} [l(y_i, \hat{y}_i^{(t-1)}) + g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i)] + \Omega(f_t)$$

其中，g_i 和 h_i 分别是损失函数关于 $\hat{y}_i^{(t-1)}$ 的一阶和二阶导数。

XGBoost 通过优化这个近似目标函数来训练每一棵树，进而实现了模型性能的不断提升。

3.3 ResNet

ResNet（残差网络）是一种深度神经网络架构，它通过引入残差连接来解决深度神经网络训练中的退化问题。

- **残差连接的数学原理和公式:** 在传统的神经网络中，第 l 层的输出 $H(x)$ 直接传递到第 $l+1$ 层。在 ResNet 中，引入了残差连接，其数学表达为：

$$F(x, \{W_l\}) = H(x) - x$$

其中，F 表示残差函数，$H(x)$ 是第 $l+1$ 层的输出，而 x 是第 l 层的输入。残差连接允许梯度直接流过网络，有助于减少梯度消失的问题，并且能够更容易地训练更深的网络。

- **为什么要进行残差连接:** 残差连接允许网络学习对输入的微小调整，而不是完全依赖于栈式层的输出。这意味着网络可以通过学习零映射（即不改变输入）来保持性能，同时仍然有能力学习更复杂的功能，从而减少了训练深层网络时的梯度退化问题。

3.3.1 MLP based ResNet

MLP（多层感知机）基于的 ResNet 在其隐藏层之间加入残差连接，以提高网络训练的效率和稳定性。

3.3.2 LSTM based ResNet

在 LSTM（长短期记忆网络）基于的 ResNet 中，残差连接被添加到 LSTM 层之间。这有助于保持信息在长序列中的传递，并减少梯度消失的问题。

3.3.3 Conv1d LSTM based ResNet

Conv1d LSTM 结合了一维卷积神经网络（Conv1D）和 LSTM 的特性，适用于处理时间序列数据。

- **Conv1D 的数学原理和公式:** 一维卷积操作可以表示为:

$$(f * g)(t) = \sum_{\tau=-\infty}^{\infty} f(\tau)g(t-\tau)$$

其中，f 是输入信号，g 是卷积核，t 是时间维度。在实际应用中，卷积核的尺寸是有限的，因此上述求和只在卷积核覆盖的范围内进行。

- **Conv1D 的 input 和 output 的 shape:** 假设输入的 shape 为 (N, L_{in}, C_{in})，其中 N 是批次大小，L_{in} 是序列长度，C_{in} 是输入通道数。对于一个具有 C_{out} 个输出通道和长度为 K 的卷积核的 Conv1D 层，输出的 shape 将是 (N, L_{out}, C_{out})，其中 L_{out} 可以根据卷积核大小、步长和填充来计算。注意：这里的通道数可以理解为每一个样本在每一个时间切片上的 num of features

3.4 数据可视化

展示近 6 个月的 $call\_stl\_price$, put_stl_price, if_stl_price 的图表，分析模型预测结果的趋势和特点。

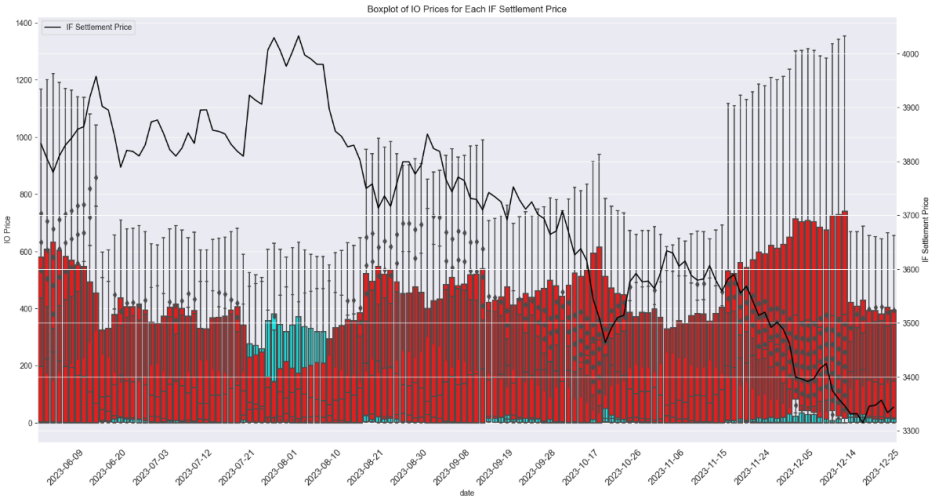

图 1: 近 6 个月的 IO put stl price; IO call stl price IF stl price

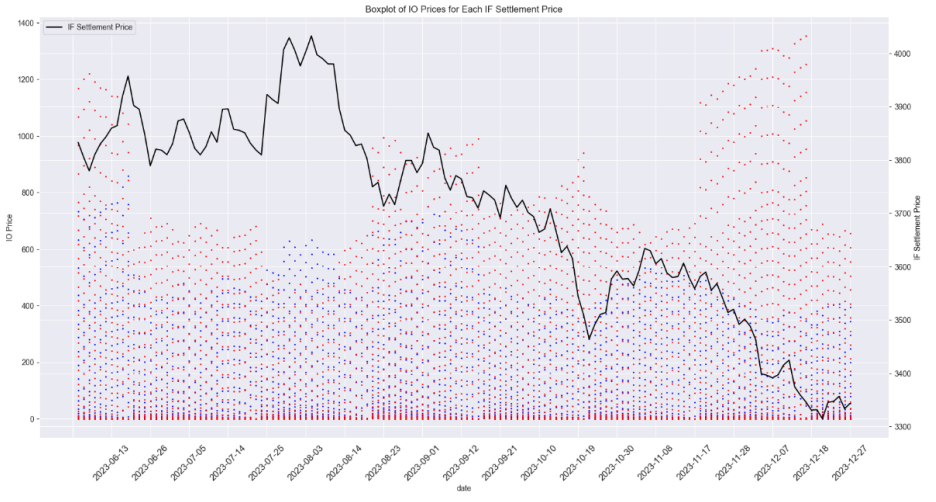

图 2: 近 6 个月的 IO put stl price; IO call stl price IF stl price

4 实验结果

各个模型的实验结果图大如下图所示：

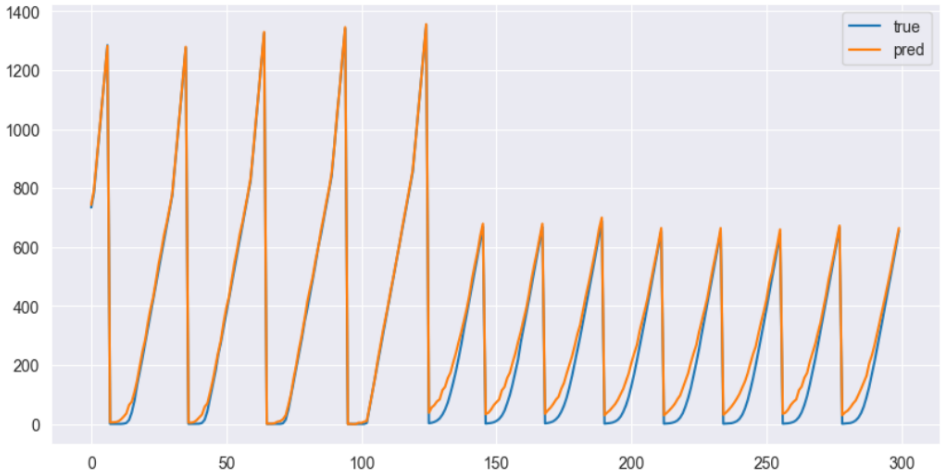

图 3: xgboost

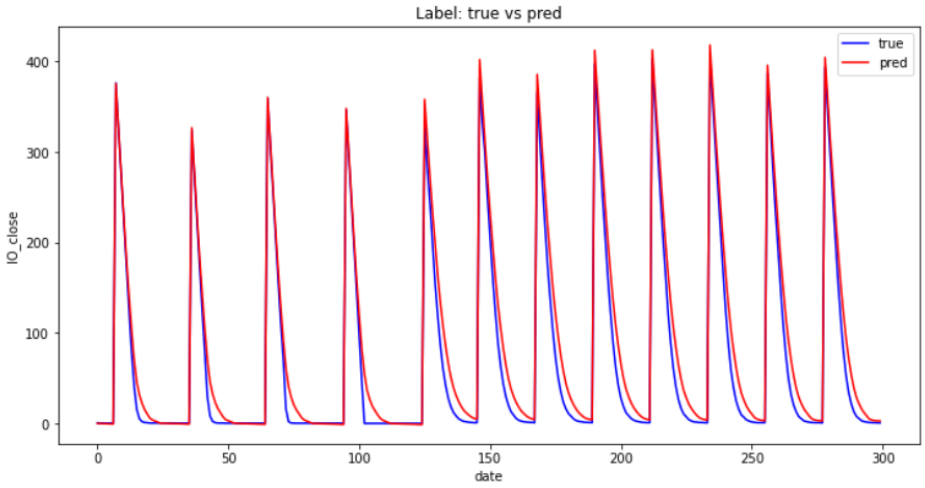

图 4: LSTM

5 讨论

在本研究中，我们通过多个模型对期权结算价数据进行了预测，包括 XGBoost、LSTM、MLP based ResNet 和 Conv1d LSTM based ResNet。以下是对这些模型性能的分析，可能的改进方向，以及在实际应用中可能面临的挑战。

1. ** 爬取数据 **：爬取数据的过程涉及从网页上获取.csv 文件。这通常需要分析网页结构，确定.csv 文件的网址格式。一种常见的挑战是网页结构的变化，可能导致爬

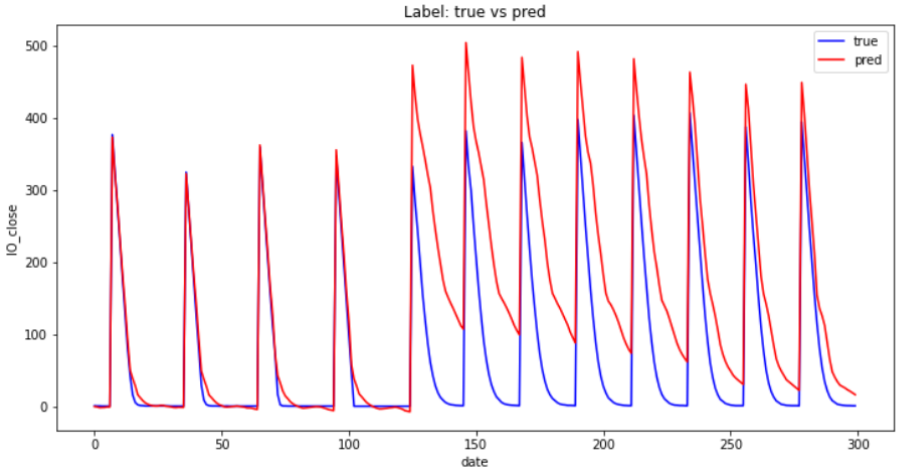

图 5: MLP based ResNet

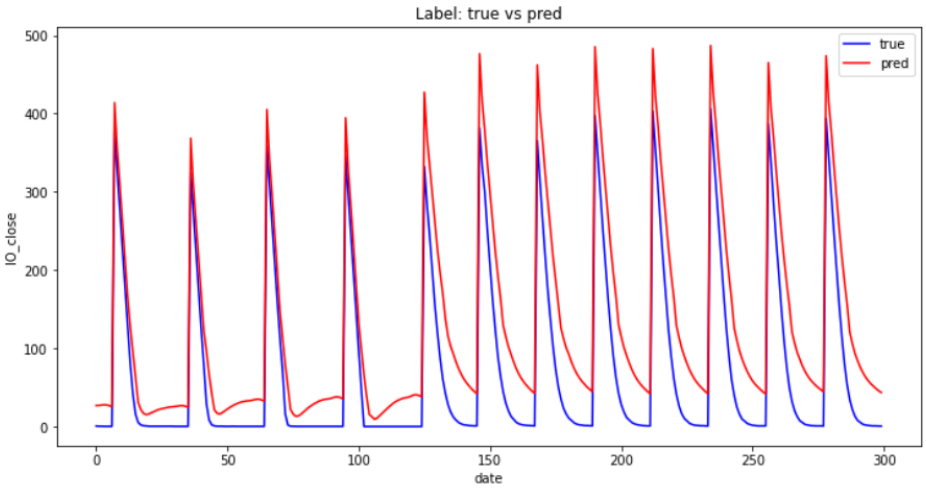

图 6: LSTM based ResNet

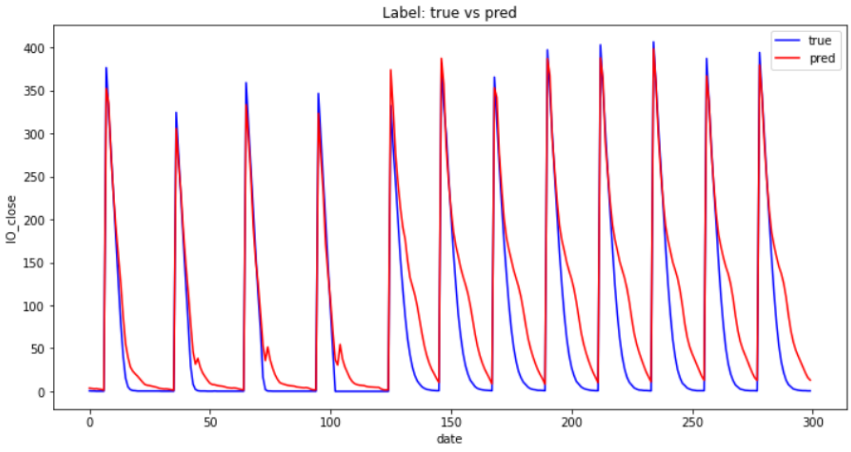

图 7: Conv1D LSTM based ResNet

虫脚本失效。此外，我们需要确保抓取的数据格式符合预期，这可能涉及对.csv 文件的内容进行校验。在实际操作中，需要注意网站更新或数据发布格式的变化，以确保数据的连续性和准确性。

2. ** 数据处理 **：在数据处理阶段，尤其是在近月合约数据的获取方面，存在一定的挑战。近月合约的确定需要考虑合约的到期日和当前日期，这可能涉及复杂的日期计算和节假日的考虑。一个潜在的改进方向是开发更加健壮的方法来自动识别和处理这些情况，从而减少手动干预和潜在的错误。

3. ** 模型输入数据的形状 **: LSTM 和 Conv1D 处理的输入数据形状不同。LSTM 通常期望输入数据的形状为 [批大小, 时间步长, 特征数]，而 Conv1D 则期望 [批大小, 特征数, 时间步长] 的形状。这种差异意味着在模型设计和数据预处理时必须特别注意。为了改进模型性能，我们可以探索不同的数据重塑策略，以最适合每种模型的方式提供输入数据。

4. ** 挑战与实际应用 **：在实际应用中，模型可能面临从理论到实践的转换挑战，如数据质量问题、实时数据处理需求和模型部署问题。此外，市场条件的变化可能导致模型性能下降，因此需要定期重新训练和调整模型。另一个关键挑战是确保模型的解释性和透明度，特别是在对金融决策有影响的应用中。

总的来说，虽然我们的模型在实验中表现出了一定的效果，但在将这些模型应用到实际场景中时，我们需要考虑数据的实时性、模型的稳定性和可维护性等多方面因素。

6 结论

本研究通过对多种模型在特定数据集上的表现进行了比较和分析，包括 XGBoost、LSTM、MLP based ResNet 以及 Conv1d LSTM based ResNet。我们主要关注的评估指标是均方误差（MSE），它衡量的是预测值与真实值之间的平均差异。以下是对每个模型的结果分析：

1. **XGBoost**：XGBoost 模型在所有模型中表现出中等水平的 MSE。作为一个基于决策树的集成学习方法，XGBoost 通常在处理结构化数据时表现良好。然而，它可能在处理具有复杂时间依赖性的数据时不如基于神经网络的模型。

2. **LSTM**：LSTM 模型在本次实验中表现最佳。这一结果表明，LSTM 强大的时间序列数据处理能力使其在捕捉时间依赖性方面特别有效。LSTM 能够处理长期依赖信息，这可能是其表现优异的主要原因。

3. **MLP based ResNet**：MLP based ResNet 模型表现最差。这可能是因为虽然残差连接有助于缓解深层网络的训练问题，但 MLP 本身不擅长处理时间序列数据，特别是当数据具有复杂的时间依赖性时。

4. **Conv1d LSTM based ResNet**: 这个模型的效果比 LSTM based ResNet 略好，但仍然不如纯 LSTM 模型。Conv1d LSTM based ResNet 结合了一维卷积层和 LSTM 层，理论上能够同时捕捉时间序列的局部特征和长期依赖。然而，这种组合可能增加了

模型的复杂度，从而影响了模型的整体性能。

综上所述，虽然残差连接和一维卷积层在某些情况下能够提高模型性能，但在处理具有复杂时间依赖性的时间序列数据时，纯 LSTM 模型可能是一个更合适的选择。此外，这些结果也提示我们，在选择模型时需要考虑到数据的特性和问题的具体需求，没有一种模型能够在所有情况下都是最优的。

本次大作业由三人共同完成，分工出力没有贵贱之分

实验代码请见：Final PJ

参考文献

[1] Sepp Hochreiter, Jürgen Schmidhuber, *Long Short-Term Memory*, Neural Computation, 9(8):1735-1780, 1997.

[2] Tianqi Chen, Carlos Guestrin, *XGBoost: A Scalable Tree Boosting System*, Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2016.

[3] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun, *Deep Residual Learning for Image Recognition*, Proceedings of the IEEE conference on computer vision and pattern recognition, 2016.