《时间序列分析》课程报告

题目：循环神经网络模型的应用比较研究

成员：陈奕昕（3227042017）（组长）
李林吉（3217042035）
李　坦（3227042001）
邵俊泽（3227042005）
练锴雯（3227042022）

应用(生物)统计学专业 2022 级

完成时间：2024 年 12 月 27 日

授课教师：李 骊

目录

摘要

背景 时间序列数据因其数据点间的时间依赖性而在金融市场分析、气象预测等多个领域中占据核心地位。尽管传统时间序列分析方法（如 AR、MA、ARMA、ARIMA 等）在特定情况下表现不俗，但它们多基于数据关系的线性假设，难以捕捉数据中的复杂非线性依赖。随着深度学习技术的进步，循环神经网络（RNN）已成为时间序列建模的新利器。**目的** 尽管深度学习方法在众多预测任务中展现出卓越性能，但目前对于不同深度学习模型的适用性尚缺乏明确指导。本研究旨在对比分析几种主流循环神经网络模型（包括 RNN、LSTM、GRU、CW-RNN 和 ESN 等），评估它们在多元时间序列建模中的优势与局限，并探索如何整合深度学习方法与传统模型的优势，以提高预测的精确度和稳定性。**方法** 本文通过模拟数据和实例数据的对比验证，对 ARIMA 模型、RNN、LSTM、CW-RNN 和 GRU 模型进行了深入分析。评价模型性能的指标包括均方误差（MSE）、均方根误差（RMSE）、平均绝对误差（MAE）和平均绝对百分比误差（MAPE）。**结论** 在模拟数据和实例分析中，GRU 模型在多个评价指标上均优于其他模型，尤其在预测准确性方面表现突出。尽管 LSTM 模型广泛使用，但在本研究中并未显示出预期的优势，只在样本量较大的情况展示出较好的拟合效果。相比之下，ARIMA 模型在所有评价指标上均不如深度学习模型。因此建议在实际应用中优先考虑 GRU 模型，并建议对 LSTM 模型进行进一步优化，同时对 CW-RNN 和 ARIMA 模型进行更深入的数据预处理或模型调整，以期获得更贴近实际的预测结果。

关键词：时间序列分析；循环神经网络；数据模拟；股票价格预测

Abstract

Background Time series data plays a central role in various fields such as financial market analysis and meteorological forecasting due to the temporal dependencies among data points. Although traditional time series analysis methods (such as AR, MA, ARMA, ARIMA, etc.) perform well in certain situations, they are mostly based on the linear assumption of data relationships and struggle to capture the complex nonlinear dependencies within the data. With the advancement of deep learning technology, Recurrent Neural Networks (RNNs) have become a new powerful tool for time series modeling.**Objective** Despite the outstanding performance of deep learning methods in numerous forecasting tasks, there is currently a lack of clear guidance on the applicability of different deep learning models. This study aims to compare and analyze several mainstream recurrent neural network models (including RNN, LSTM, GRU, CW-RNN, and ESN, etc.), assess their strengths and limitations in multivariate time series modeling, and explore how to integrate the advantages of deep learning methods with traditional models to improve the accuracy and stability of forecasts. **Method** This paper conducts an in-depth analysis of ARIMA models, RNN, LSTM, CW-RNN, and GRU models through comparative verification with simulated and empirical data. The performance metrics for evaluating the models include Mean Squared Error (MSE), Root Mean Squared Error (RMSE), Mean Absolute Error (MAE), and Mean Absolute Percentage Error (MAPE). **Conclusion** In both simulated data and empirical analysis, the GRU model outperforms other models across multiple evaluation metrics, particularly in terms of predictive accuracy. Although the LSTM model is widely used, it did not show the expected advantage in this study, only demonstrating better fitting effects in situations with larger sample sizes. In contrast, the ARIMA model is inferior to deep learning models in all evaluation metrics. Therefore, it is recommended to prioritize the GRU model in practical applications, and further optimization of the LSTM model is suggested. Additionally, more in-depth data preprocessing or model adjustments for CW-RNN and ARIMA models are recommended to achieve more realistic forecasting results.

Keywords：Time Series Analysis；Recurrent Neural Networks；Data Simulation, Stock Price Forecasting

1.介绍

1.1 背景

时间序列数据广泛存在于各个领域，包括金融市场分析、气象预测、健康监测、销售预测等。时间序列数据的核心特点是数据点之间存在时间依赖性，即当前时刻的状态与历史状态密切相关。因此如何有效地建模时间序列的时序依赖性，成为解决这些实际问题的关键挑战之一。传统的时间序列分析方法，如自回归模型（AR）、移动平均模型（MA）以及其组合形式（ARMA、ARIMA）等，虽然在某些情况下能够取得较好的效果，但这些方法一般假设数据的关系是线性且具有固定的时间跨度，并且通常无法有效捕捉数据中复杂的非线性依赖关系。随着计算能力的提升和深度学习技术的发展，循环神经网络逐渐成为时间序列建模领域的重要工具。

Hochreiter, S 等[1]首次提出了 LSTM 模型，并说明了其在长短期记忆网络的结构和优势，同时其也讨论了 LSTM 在序列数据处理时间序列数据中的优势。Chen, Xue 等 Chimmula 等[2]通过将 LSTM 应用在医学上用于预测新型冠状病毒爆发上的应用，进一步说明 LSTM 的价值。Kim, Taewook 等[3]将 LSTM 应用到股票价格的预测上，进一步说明其在股票价格预测上的预测优势。Torres, Jose F 等[4]在论文中回顾了 RNN 在时间序列预测中的应用，并说明其在不同领域中的应用。Fournier, Claudia 等[5] 提出了一种改进的 LSTM 模型，用于多变量时间序列预测，其结合了多种数据预处理技术和 LSTM 网络优化方法，展示了该方法在多种实际应用场景中的优势。此外，Sheng, Ziyu 等[6]在其论文中回顾性的研究了多种循环神经网络模型，并且概括了其在时间序列数据预测上的应用。

1.2 目的

尽管这些深度学习方法在许多预测任务中展示了卓越的性能，特别是在处理复杂、非线性关系和大规模数据时，它们仍面临一些挑战，最主要的问题之一是缺乏对不同深度学习模型适用情况的明确界定，包括它们在不同场景下的适用范围、优劣性以及性能表现的差异。目前，许多研究虽然集中于某一特定模型在特定任务中的表现，但对这些模型在多元时间序列预测中的全面比较仍较为稀缺。

因此本文旨在对比几种主流的循环神经网络模型（包括 RNN、LSTM、GRU 和 CW-RNN），评估它们在时间序列建模中的优势和局限性。本文将揭示不同模型在处理复杂依赖的时间序列数据时的表现差异，并探讨它们在不同数据规模、噪声水平以及非线性关系下的适用性。此

外考虑到传统统计模型（如 ARIMA）在处理较小数据集和关系明确的时间序列时，仍能保持较高的预测精度和效率，本文还将探讨深度学习方法对比传统模型的优势。

1.3 方法

1.3.1 Arima

ARIMA 模型，即自回归积分滑动平均模型（AutoRegressive Integrated Moving Average Model），是时间序列分析中常用的一种预测模型。它结合了自回归（AR）模型、差分（I）和滑动平均（MA）模型的特点，特别适用于分析和预测单变量时间序列数据。下面是对 ARIMA 模型的简要介绍：

自回归（AR）部分：模型考虑了时间序列过去值对其未来值的影响。如果一个时间序列的当前值与其前几期的值相关，那么这个时间序列就具有自回归性。

差分（I）部分：为了使非平稳时间序列变得平稳，可以通过对原始序列进行差分来实现。差分的次数称为差分阶数（d）。

滑动平均（MA）部分：模型考虑了时间序列的当前值与过去预测误差的关系。如果一个时间序列的当前值与其前几期的预测误差相关，那么这个时间序列就具有滑动平均性。

1.3.2 RNN

RNN（Recurrent Neural Network）循环神经网络[7]是一种常用于处理序列数据的神经网络模型，其最大的特点是具有循环连接，能够将前一时刻的输出信息反馈到网络中，对于时序数据和多变量时间序列数据非常适用。RNN 通过隐藏状态（hidden state）在时间步骤间传播信息，使得模型能够记住之前时刻的输入信息，从而捕捉序列的时间依赖性。RNN 的基本结构包括输入层、隐藏层以及输出层，输入层即每个时刻输入一个特征向量，通常是时间序列的一个切片；隐藏层通过递归计算，传递前一时刻的隐藏状态和当前时刻的输入信息，计算当前的隐藏状态；输出层则根据当前隐藏状态生成输出，可以是分类或回归任务的预测结果。其构成图如下所示。

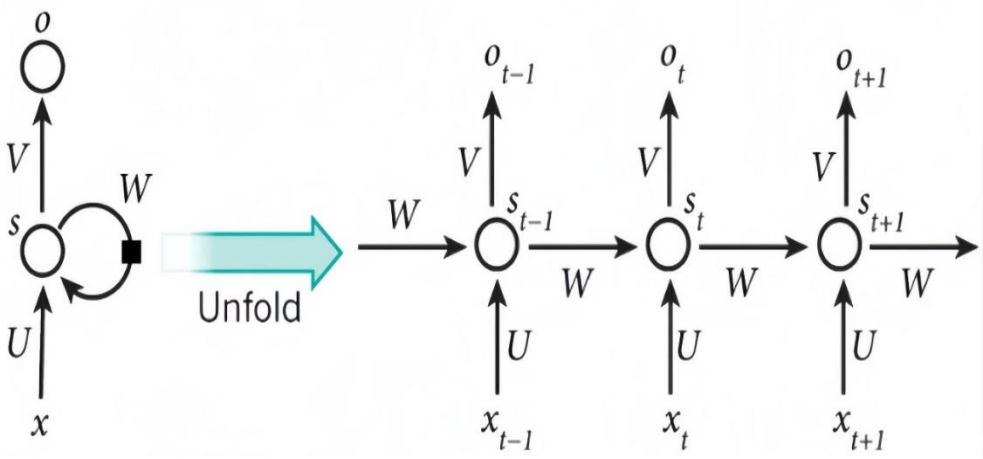

图 1　RNN 构成图

对于隐藏状态的更新，

$$h_t = \sigma\left(W_{xh}x_t + W_{hh}h_{t-1} + b_h\right) \tag{1}$$

其中，h_t是当前时刻的隐藏状态，x_t是当前时刻的输入，h_{t-1} 是上一时刻的隐藏状态（递归连接），W_{xh}是输入到隐藏层的权重矩阵，W_{hh}是隐藏状态到隐藏状态的权重矩阵，b_h是隐藏层的偏置项，σ是激活函数，它对输入值进行非线性变换，从而帮助网络捕捉非线性的关系。

对于输入层的计算，若 RNN 用于回归任务，输出 y_t则是与当前隐藏状态 h_t直接相关的预测值，

$$y_t = W_{hy}h_t + b_y \tag{2}$$

其中，W_{hy}是从隐藏层到输出层的权重矩阵，b_y是输出层的偏置项。

1.3.3 LSTM

LSTM (Long Short Term Memory) 长短期记忆网络[2, 5, 8-11]是一种针对时间序列数据设计的深度学习模型，广泛应用于多种时间序列预测任务。由于其结构中特有的门控机制，LSTM 能够有效捕捉和记忆长期的时间依赖关系，克服了传统 RNN 在处理长期依赖时遇到的梯度消失问题，因此在许多时间序列分析任务中表现出了卓越的性能。

LSTM 通过引入一种叫做记忆单元（memory cell）的结构来解决长短期依赖问题。每个 LSTM 单元包含三个重要的门控机制，分别是 输入门、遗忘门和输出门，这使得它可以有效地选择和更新它的记忆。对于遗忘门（Forget Gate），其决定了哪些信息需要丢弃。在每个时间步，遗忘门通过一个 Sigmoid 激活函数决定应该“忘记”多少来自前一状态的记忆，这个值越接近 0 表示完全忘记，值越接近 1 表示完全保留，遗忘门可以表示为，

$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] + b_f\right) \tag{3}$$

其中，f_t是遗忘门的输出，h_{t-1}是前一时刻的隐藏状态，x_t是当前时刻的输入，W_f是权重矩阵，b_f是偏置项，σ是激活函数。

输入门（Input Gate）即决定了哪些新的信息可以被存储在记忆单元中。输入门首先通过一个 Sigmoid 函数来决定哪些信息将被更新，然后通过一个 Tanh 函数生成新的候选值，将其与输入门的输出结合，更新记忆单元，即基于遗忘门和输入门的输出，更新当前的记忆单元状态。通过将遗忘门的输出与上一时刻的记忆单元状态相乘，再加上输入门的更新信息，来得到新的记忆状态，输入门、记忆单元以及记忆单元的更新可以表示为，

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \tag{4}$$

$$\tilde{C}_t = \tan\mathrm{h}(W_C \cdot [h_{t-1}, x_t] + b_C) \tag{5}$$

$$C_t = f_t \cdot C_{t-1} + i_t \cdot \tilde{C}_t \tag{6}$$

其中，i_t是输入门的输出，$\tilde{C}_t$是候选记忆信息，C_{t-1}是前一时刻的记忆单元状态，C_t 是当前时刻的记忆单元状态。

对于输出门（Output Gate），其决定当前时刻的输出。首先通过 Sigmoid 激活函数生成一个输出门，它决定了哪些部分的记忆需要输出。然后记忆单元状态通过 Tanh 函数进行压缩，并与输出门的结果相乘，得到当前时刻的输出，输出门可以表示为，

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \tag{7}$$

其中，o_t为输出门的输出，W_o为权值矩阵，h_{t-1}为前一时刻的隐藏状态，x_t是当前时刻的输入，b_o为偏置项。

最终的隐藏状态h_t可以表示为

$$h_t = o_t \odot \tan\mathrm{h}(C_t) \tag{8}$$

其中，o_t为输出门的结果，C_t为当前记忆单元状态。

对于最终的预测结果$\hat{y}$可以表示为

$$\hat{y}_t = W_h \cdot h_t + b_h \tag{9}$$

其中，W_h为权值矩阵，b_h为偏置项。

1.3.4 CW-RNN

CW-RNN (Clockwork Recurrent Neural Network) [12]是一种用于处理时间序列数据的神经网络模型，其主要特点是通过多个递归神经网络（RNN）模块在不同的时间尺度上工作，从而能

够捕捉时间序列数据中的多尺度动态特征。CW-RNN 模型适用于具有多层次时间依赖关系的任务，对长时间序列预测和复杂动态系统建模有较好的性能。

CW-RNN 的核心思想是在标准的 RNN 模型中加入时间尺度的分解，将时间序列划分为不同的时间粒度进行处理。它通过分层的时序更新机制使得每个 RNN 模块在不同的时间间隔更新，从而能够同时学习多种时间尺度上的依赖模式。对于 CW-RNN 的构成，其是由多个 RNN 模块组成，每个模块按照不同的时间间隔更新其状态。通过将多个 RNN 模块组成可以将长时间尺度和短时间尺度的依赖同时建模。每个模块的隐藏状态更新频率不同，模块间的时间步长通常呈倍数关系。通过这种方式 CW-RNN 能够在不同的时间粒度上捕捉到数据的动态信息，从而在多尺度的依赖关系上进行建模。对于每个 RNN 模块，其不仅仅依赖于自己的状态，也依赖于前一个模块的输出，从而实现不同时间尺度的交互和信息流动，其构成图如下所示。

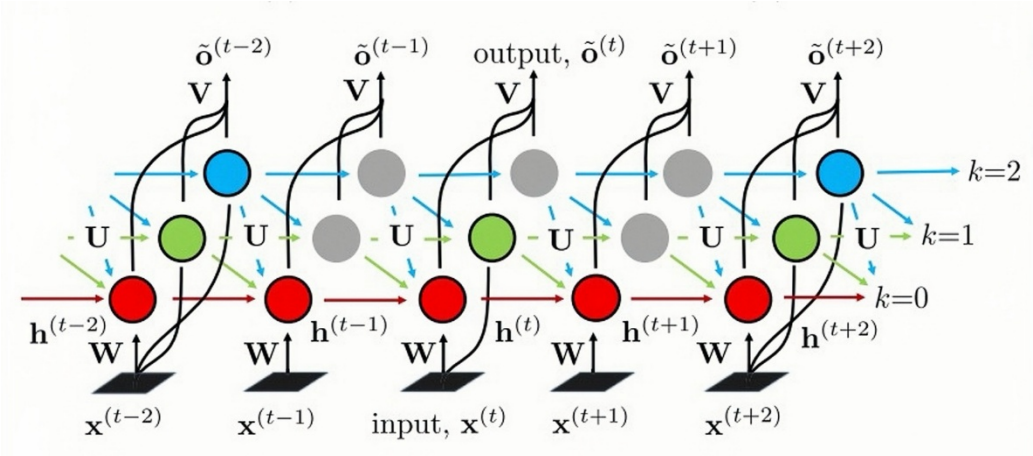

图 2　CW-RNN 构成图

假设我们有一个时间序列数据$\{x_t\}_{t=1}^{T}$，对于隐藏状态更新（时间尺度分解）每个 RNN 模块都有自己的隐藏状态，假设每个模块的隐藏状态为$h_i(t)$，其中i是模块的索引,t是时间步。对于每个模块，隐藏状态的更新可以表现为，

$$h_i(t) = \sigma\big(W_i x(t) + U_i h_i(t-1)\big) \tag{10}$$

其中，$h_i(t)$是第i个模块在时间步t的隐藏状态，

σ是非线性激活函数，W_i和U_i分别是输入和隐藏状态到当前隐藏状态的权重矩阵，$x(t)$是当前输入的向量。

在 CW-RNN 中，每个模块的更新频率不同。第i个模块每隔2^i个时间步更新一次隐藏状态。

对于最后的输出即所有模块的加权组合，假设输出$y(t)$是由最后一个模块的隐藏状态和一些全连接层计算得到的，即$y(t)$表示为，

$$y(t) = W_{\text{out}} h_n(t) \tag{11}$$

其中，$h_n(t)$是最后一个模块（即第n个模块）的隐藏状态，W_{out} 是输出层的权重矩阵。

1.3.5 GRU

GRU（Gated Recurrent Unit）[13]是一种基于循环神经网络（RNN）的改进型神经网络结构，是循环神经网络的一种，旨在解决传统 RNN 在处理长时间序列数据时遇到的梯度消失和梯度爆炸问题。GRU 通过引入门控机制，使得模型能够有效捕捉时间序列中的长期依赖关系，同时在计算效率上比 LSTM 更具优势。GRU 主要包含两个核心部分,更新门（update gate）和重置门（reset gate），其简化结构图如下所示。

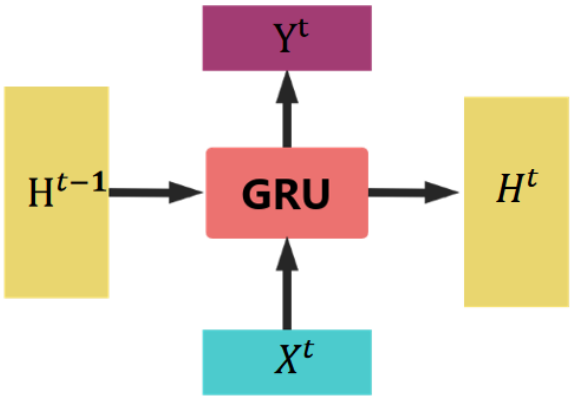

图 3　GRU 简化构成图

假设时间序列的每个时间步输入为x_t，上一时刻的隐藏状态为h_{t-1}。则 GRU 的更新门如下所示， 其用于控制前一时刻的状态信息被带入到当前状态中的程度，也就是更新门帮助模型决定到底要将多少过去的信息传递到未来，即用于更新记忆，

$$z_t = \sigma(W_z x_t + U_z h_{t-1} + b_z) \tag{12}$$

其中，z_t决定了上一时刻h_{t-1}对当前状态的影响程度，σ表示 sigmoid 激活函数，W_z是当前时刻输入x_t的权重矩阵。U_z是上一时刻隐藏状态 h_{t-1}的权重矩阵，b_z是更新门的偏置项。

上述公式即表示为，若z_t越接近 1，代表“记忆”下来的数据越多即保留的信息越多；而z_t越接近 0 则代表“遗忘”的越多即丢弃的信息越多。

重置门如下所示，其决定了如何将新的输入信息与前面的记忆相结合，

$$r_t = \sigma(W_r x_t + U_r h_{t-1} + b_r) \tag{13}$$

其中，r_t控制当前输入对候选隐藏状态$\widetilde{h_t}$的影响，W_r是当前时刻输入x_t的权重矩阵，U_r是上一时刻隐藏状态 h_{t-1}的权重矩阵，b_r是重置门的偏置项。

上述公式即表示为，若r_t的值越小，它与h_{t-1}经过哈达玛积计算的矩阵数值则越小，再通过与权值矩阵相差得到的值就越小，即$(r_t \odot h_{t-1})W_r$的值越小，表示上一个时刻的信息需要遗忘的越多，丢弃的越多。反正，$(r_t \odot h_{t-1})W_r$值越大，说明需要记住上时刻的信息就越多，新输入的信息x_t与前面记忆结合就越多。当r_t值接近 0 时，说明上一个时刻的内容就需要全部丢弃，只保留当前时刻。当r_t的值接近 1 时，表示保留上一时刻的隐藏状态。

对于隐藏状态$\widetilde{h_t}$的计算，

$$\widetilde{h_t} = \tan \mathrm{h}(W_h x_t + U_h(r_t . h_{t-1}) + b_h) \tag{14}$$

其中，tanh 表示双曲正切激活函数，$\widetilde{h_t}$是当前时间步的候选隐藏状态，它基于当前输入和经过重置门调整的过去隐藏状态计算。

最终隐藏状态

$$h_t = (1 - z_t) \cdot h_{t-1} + z_t \cdot \tilde{h}_t \tag{15}$$

其中，h_t是当前时刻的输出状态。更新门z_t控制着上一时刻的隐藏状态h_{t-1}与当前候选状态$\tilde{h}_t$的加权和。

h_t忘记传递下来的 h_{t-1}中的某些信息，并加入当前节点输入的某些信息，从而形成了最终的记忆h_t。其总的构成图如下所示。

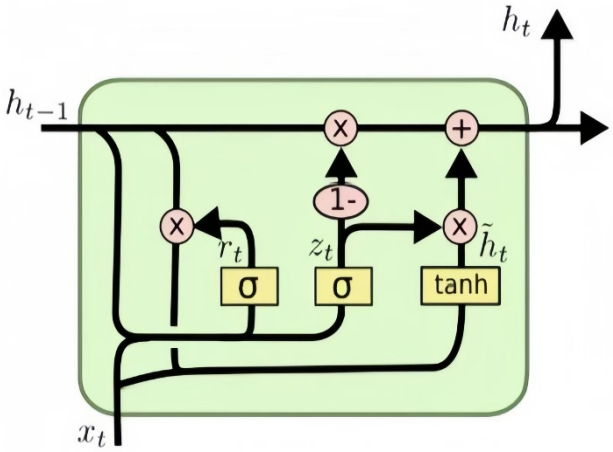

图 4　GRU 构成图

2. 模拟分析

2.1 评价指标

本文的模型的对比我们应用的指标有均方误差、均方根误差、平均绝对误差、平均绝对误差以及平均绝对百分比误差。

均方误差（MSE）：均方误差是预测值与实际值之差的平方和的平均值。它能够量化模型预测的误差，并且对较大的误差给予更大的惩罚（因为误差是平方的）。MSE 的计算公式如下：

$$MSE = \frac{\sum_{i=1}^{n} (f(x) - y)^2}{n} \tag{16}$$

其中，y是实际值，$f(x)$是预测值，n是样本数量。

均方根误差（RMSE）：均方根误差是 MSE 的平方根，它与原始数据在同一量纲上，因此更易于解释。RMSE 的计算公式如下：

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n} (y_i - \hat{y}_i)^2} \tag{17}$$

平均绝对误差（MAE）：平均绝对误差是预测值与实际值之差的绝对值的平均值。与 MSE 和 RMSE 相比，MAE 对异常值的敏感度较低。MAE 的计算公式如下：

$$RMSE = \frac{1}{n}\sum_{i=1}^{n} |y_i - \hat{y}_i| \tag{18}$$

平均绝对百分比误差（MAPE）：平均绝对百分比误差是预测误差的绝对值与实际值之比的平均值，以百分比形式表示。它能够反映预测值与实际值之间的相对误差。MAPE 的计算公式如下：

$$RMSE = \frac{100\%}{n}\sum_{i=1}^{n} \left|\frac{y_i - \hat{y}_i}{y_i}\right| \tag{19}$$

2.2 数据生成

模拟分析部分使用正态分布与累加算法结合生成，使其具有增长趋势，并且符合一般数据的波动趋势，在同样在后期也使数据出现了一定的较大波动，符合突发事件发生的情况。

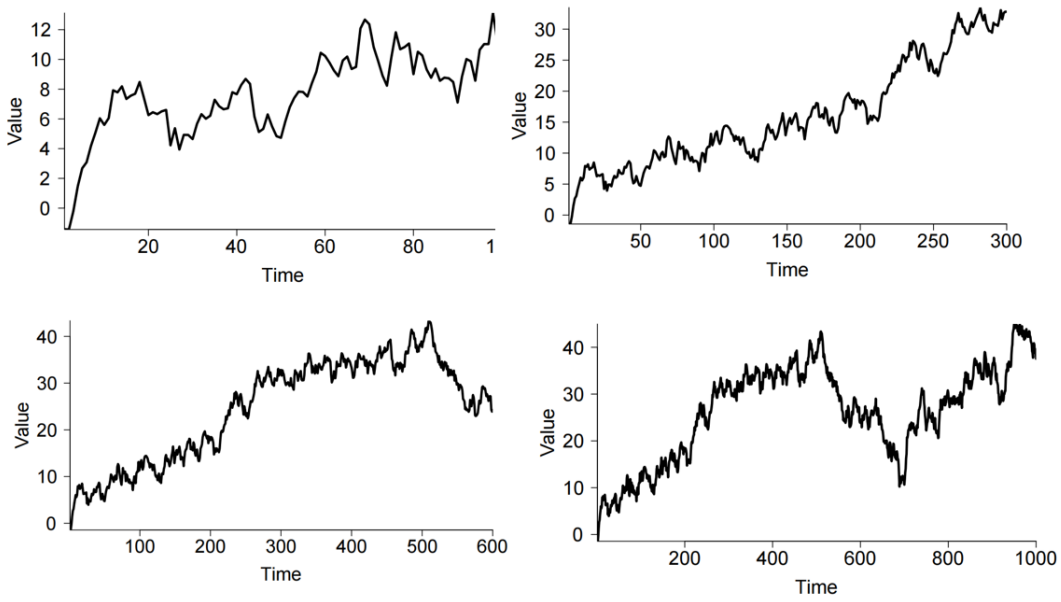

图 5　不同样本量下的模拟数据时序图

3.2 模拟比较

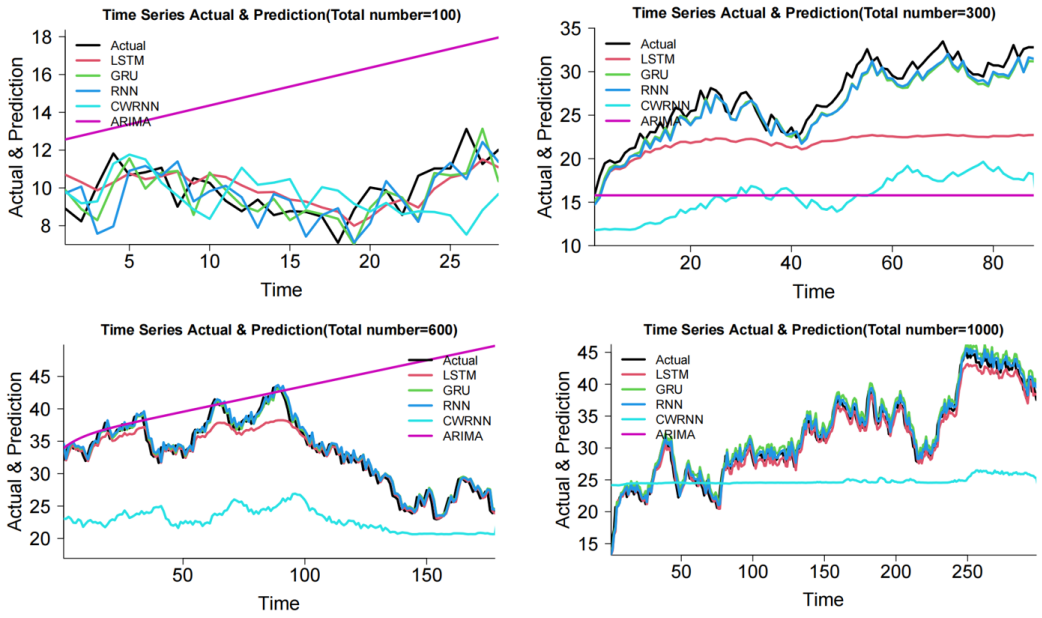

图 6　不同样本量模型拟合曲线图

从拟合效果图中均可看出 ARIMA、CWRNN 模型的拟合效果在多种数据规模下均不理想，且 ARIMA 模型在预测样本数据大时愈发偏移，而 CWRNN 模型在十分有限解释波动程度的同时与真实值相差极大，故在真实数据的情况下，建议使用者谨慎考虑此二类方法。而

LST 模型虽然在低样本量时效果并不理想，但在数据量足够大时效果会得到显著提升。与三者相反，RNN、GRU 模型效果相近，且在不同数据规模时其效果均较为优异，但仅凭图像无法粗略做出诊断其优劣，需使用其它评价指标进行佐证区分。

表 1　模拟数据模型评价指标汇总表

Model	Number	MSE	RMSE	MAE	MAPE
LSTM	100	1.3542	1.1637	0.9807	0.9807
LSTM	300	38.3946	6.1963	5.4313	5.4313
LSTM	600	2.6862	1.639	1.2266	1.2266
LSTM	1000	1.7286	1.3148	1.0611	1.0611
GRU	100	1.5317	1.2376	1.0188	0.1018
GRU	300	2.2787	1.5095	1.269	0.0464
GRU	600	1.1361	1.0659	0.8618	0.0269
GRU	1000	2.3338	1.5277	1.2625	0.0409
RNN	100	2.3501	1.533	1.2367	0.1262
RNN	300	2.2701	1.5067	1.2482	0.0465
RNN	600	1.2234	1.1061	0.874	0.0271
RNN	1000	1.3683	1.1698	0.9425	0.0309
CWRNN	100	3.4643	1.8561	1.4731	0.1402
CWRNN	300	136.5547	11.4588	11.1851	0.4082
CWRNN	600	123.337	10.9988	10.5402	0.3098
CWRNN	1000	81.7107	8.9583	7.4089	0.2124
ARIMA	100	34.5238	5.8757	5.5365	0.5824
ARIMA	300	142.9257	11.9552	11.1368	0.3956
ARIMA	600	161.0502	12.6906	9.4449	0.3326
ARIMA	1000	31500.0477	177.4825	156.3277	4.591

可以明显看出 RNN 和 GRU 在预测准确性方面的表现优于其他模型。它们在 MSE、RMSE、MAE 和 MAPE 这四个关键性能指标上都展现出了较小的误差。表格中展示的是在样本量 100 时的情况，尽管部分情况下，RNN 在指标上都达到了最低的误差值，可出其在处理特定数据

集时的高效性和准确性。但 GRU 模型在更多时刻稳定且优秀，综合来看，GRU 比 RNN 优势更大。与二者相比，LSTM 虽然是一种常用的深度学习模型，但在这次比较中并未展现出预期的优势，其性能未能超越 RNN 和 GRU，但其只有在样本量极大时效果较好。此外，CWRNN 和 ARIMA 在所有指标上的表现都不如深度学习模型，尤其是在 MSE 和 MAPE 上，这可能意味着这些传统模型在处理该数据集时存在局限性，或者需要进一步的优化和调整。总体而言，RNN 和 GRU 在这次性能比较中表现最佳，推荐在类似任务中优先考虑这两种模型。同时，也建议对 LSTM 进行进一步的优化，以及对 CWRNN 和 ARIMA 进行更多的数据预处理或模型调整，以期提高它们的预测性能。

3.实例分析

3.1 数据介绍

本文的数据来源于中兴集团， 2023 年 9 月 19 日到 2024 年 12 月 21 日交易日的每日股票的开盘价、收盘价、高点、低点和交易量等市场数据指标信息，部分数据及结构如下表所示。

表 2　股票数据部分信息

日期	收盘	开盘	高	低	交易量	涨跌幅
2023/9/19	32.32	32.6	32.74	32.08	33.91	-0.95%
2023/9/20	32.43	32.35	32.86	32.3	29.85	0.34%
2023/9/21	33.21	32.53	34.09	32.45	87.93	2.41%
2023/9/22	34.76	33.14	34.8	32.88	113.65	4.67%
2023/9/25	33.77	34.54	34.65	33.68	65.7	-2.85%
…	…	…	…	…	…	…

在得到中兴集团的股票历史数据后，本文将收盘价序列确定为目标变量，进而对其进行可视化，其时序图如图 6 所示。

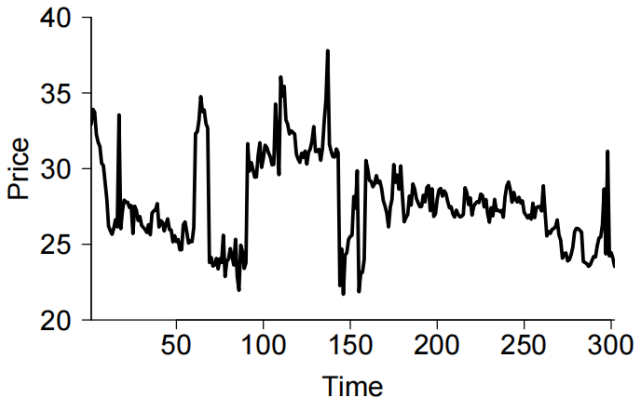

图 7　收盘价时序图

通过对股票历史数据变化的观察可以发现，数据序列呈平稳的趋势。为了进一步验证其平稳性，通过单位根（ADF）检验，获得的 P 值大于 0.05，即该序列为非平稳。其 ACF 图和 PACF 图如下所示。

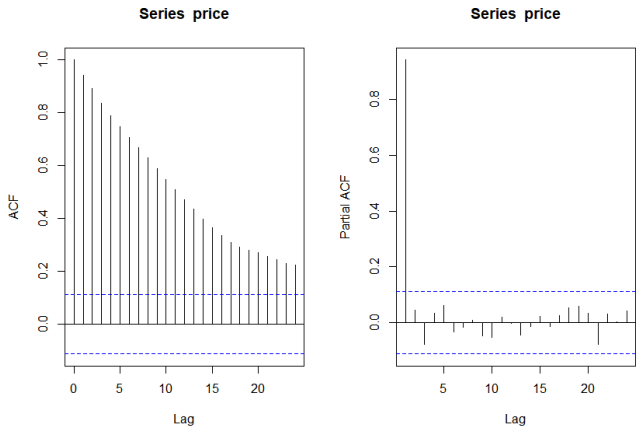

图 8　收盘价 ACF 和 PACF 图

3.2 模型构建

本文对于神经网络模型的构建，均将时间步长设置为 10，即下一个交易日的股价受前 10 个交易日的影响，程序均通过构建滑动窗口来实现。为了得到更加可信的模型，我们将股票数据集划分为训练集和验证集,将 70%的数据作为训练集，30%的数据作为验证集。本文的模型构建均基于 R 语言的 Keras 包。

RNN 模型的构建，本文基于 Keras 包设置的各种神经网络中的神经元均为 200 个，激活函数均设置为 Relu 函数。

$$f(x) = max(0, x) \tag{20}$$

通过 Relu 函数将负值输出为 0，正值则保持不变，加速训练的过程并且减少梯度消失的

问题。损失函数设置为均方误差函数。

$$MSE = \frac{\sum_{i=1}^{n} (f(x) - y)^2}{n} \tag{21}$$

优化器均设置为 Adam 优化器，即通过计算梯度的一阶矩估计和二阶矩估计来调整每个参数的学习率，其可以通过自动调整每个参数的更新的步长，从而实现更高效的网络训练。对于模型的训练，本文将 epochs 设置为 200，即训练数据进行 200 次的迭代，每一个迭代则成为一个 epoch，在每次迭代结束后，模型将自动调参，然后减少损失函数的值，其中在每次训练时本文均将数据划分为 32 个样本组进行训练，从而加快训练速度。

3.3 结果分析

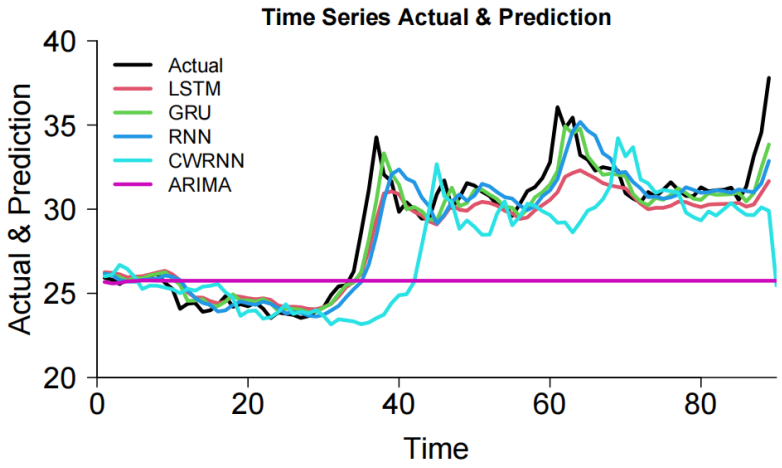

图 9　实例数据拟合图

表 3　实例数据模型评价指标汇总表

指标	LSTM	GRU	RNN	CWRNN	ARIMA
MSE	2.27087879	1.1318195	2.07214379	17.4373455	22.9655293
RMSE	1.50694364	1.0638701	1.43949446	4.0835684	4.7922364
MAE	0.99202239	0.6779366	0.93997656	3.2367163	3.9498072
MAPE	0.03210797	0.0222148	0.03067236	0.1040159	0.1278345

从评价指标来看，GRU 模型在多个维度上表现均是最优的，尤其在 MSE（1.1318）、RMSE（1.0639）和 MAE（0.6779）上明显优于其他模型，说明其预测精度和误差控制方面表现突出。此外，GRU 的 MAPE 为 0.0222，显示出其在相对误差上的优势，表明该模型在实际应用中能够提供较为精确的预测结果。相比之下，RNN 虽然在 MAPE（0.0307）和 MSE（2.0721）上与

GRU 接近，但在 RMSE（1.4395）和 MAE（0.9400）上的表现稍逊，说明其尽管结构简单，适用于一些任务，但在复杂数据的预测中略显不足。LSTM 虽然在 MAPE（0.0321）和 MAE（0.9920）上表现稳定，但其计算开销较大，效率较低，因此在需要较高计算资源时可能不如 GRU。在 CWRNN 和 ARIMA 的表现上，两者在 MSE 和 RMSE 上明显高于其他模型，分别为 17.4373 和 22.9655，表明它们在捕捉时间序列规律和减小误差方面的能力较弱，尤其在 MAPE（CWRNN 为 0.1040，ARIMA 为 0.1278）上，误差较大，可能导致实际应用中的预测不够准确。因此，综合来看，GRU 是最优选择，尤其适用于需要高精度预测的时间序列任务，而 RNN 和 LSTM 也能在特定场景下提供有效的预测，CWRNN 和 ARIMA 则更适用于误差容忍度较高的简单任务。

4.讨论

本文通过对循环神经网络模型（RNN、LSTM、GRU、CW-RNN）和传统统计模型（ARIMA）在时间序列分析中的应用进行了深入的比较研究。由于时间序列数据在多个领域的重要性以及传统方法在处理非线性依赖关系时的局限性，且随着深度学习技术的发展，特别是循环神经网络模型，已成为时间序列建模的重要工具，故本文通过模拟数据和实例数据的对比分析，评估不同模型在多元时间序列建模中的优势和局限性，并探讨如何整合深度学习方法与传统模型的优势，以提高预测的精确度和稳定性。

本文的研究结果表明，在模拟数据和实例分析中，GRU 模型在多个评价指标上要优于其他模型，尤其在预测准确性方面表现突出。LSTM 模型虽然被广泛使用，但在本研究中并未显示出预期的优势，只有在样本量较大的时候展现出较好的拟合效果。CW-RNN 和 ARIMA 模型在所有评价指标上的表现均不如深度学习模型，这可能意味着这些传统模型在处理特定数据集时存在局限性，或者需要进一步的优化和调整。因此，建议在实际应用中优先考虑 GRU 模型，并建议对 LSTM 进行进一步优化，同时对 CW-RNN 和 ARIMA 进行更多的数据预处理或模型调整。

对于本文拟合的 LSTM 模型效果不佳的原因可能有几个方面。首先，LSTM 是一个复杂的模型，容易发生过拟合，并且本文的数据量较小导致训练数据不足，从而导致过拟合的发生。其次，LSTM 需要大量的训练数据来有效学习长期依赖关系，而本文的数据量较小，模型的预测效果可能会受此影响。另一个原因是 LSTM 擅长处理具有长期依赖性的序列数据，而对于

短期波动较多或不具长期依赖性的任务，LSTM 的复杂性可能反而导致表现不佳，因为本文应用的股票数据且其在短期内震荡较大，而导致的模型拟合效果不好。

同时本文存在一定的缺陷，其一是本文的实例分析仅使用了中兴集团的股票数据，可能无法全面代表所有类型的时间序列数据。未来的研究可以考虑更多种类的数据集，以增强模型的泛化能力。其二是本文的设定的模型参数较少，虽然对模型进行了基本的训练和测试，但可能存在更优的参数设置和网络结构，本文未能进行全面的参数优化。其三是深度学习模型通常被认为是“黑箱”模型，本文未能深入探讨模型的解释性，即为何某些模型在特定数据集上表现更好。在未来的研究中可以包括更多种类的时间序列数据，如气象数据、健康监测数据等，以测试和验证模型的泛化能力。同时，可以通过自动化的超参数优化技术，如网格搜索、随机搜索或贝叶斯优化，来寻找最优的模型参数。此外，还可以探索深度学习模型与传统统计模型更深层次的结合方式，以利用各自的优势，提高预测的准确性和稳定性。

参考文献

[1] Waqas M, Humphries U W. A critical review of RNN and LSTM variants in hydrological time series predictions[J]. MethodsX, 2024,13:102946.

[2] Chimmula V K R, Zhang L. Time series forecasting of COVID-19 transmission in Canada using LSTM networks[J]. Chaos Solitons Fractals, 2020,135:109864.

[3] Kim T, Kim H Y. Forecasting stock prices with a feature fusion LSTM-CNN model using different representations of the same data[J]. PLoS One, 2019,14(2):e0212320.

[4] Torres J F, Hadjout D, Sebaa A, et al. Deep Learning for Time Series Forecasting: A Survey[J]. Big Data, 2021,9(1):3-21.

[5] Fournier C, Fernandez-Fernandez R, Cires S, et al. LSTM networks provide efficient cyanobacterial blooms forecasting even with incomplete spatio-temporal data[J]. Water Res, 2024,267:122553.

[6] Sheng Z, Wen S, Feng Z, et al. A Survey on Data-Driven Runoff Forecasting Models Based on Neural Networks[J]. IEEE TRANSACTIONS ON EMERGING TOPICS IN COMPUTATIONAL INTELLIGENCE, 2023,7(4):1083-1097.

[7] Wang J, Zhang W, Yang H, et al. Visual Analytics for RNN-Based Deep Reinforcement Learning[J]. IEEE Trans Vis Comput Graph, 2022,28(12):4141-4155.

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

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

[10] Usmani S, Shamsi J A. LSTM based stock prediction using weighted and categorized financial news[J]. PLoS One, 2023,18(3):e0282234.

[11] Greff K, Srivastava R K, Koutnik J, et al. LSTM: A Search Space Odyssey[J]. IEEE Trans Neural Netw Learn Syst, 2017,28(10):2222-2232.

[12] Koutník J, Greff K, Gomez F, et al. A Clockwork RNN[J]. 2014.

[13] Dey R, Salem F M. Gate-Variants of Gated Recurrent Unit (GRU) Neural Networks[J]. 2017.

附录

```
##########################################
# 项目名称:时间序列分析--实例分析
# 软件版本:R 4.4.1
# 攥写日期:2024.12.18
##########################################

#安装并加载相关包

#install.packages("tensorflow");install.packages("reticulate")
#install.packages("keras");install.packages("tidyverse");install.packages("rio")
#install.packages("forecast");install.packages("tseries")
library(tensorflow);library(reticulate);library(keras) # 用于进行深度学习
library(rio) # 用于数据导入
library(tidyverse) # 用于数据处理
library(forecast) # ARIMA 模型的预测
library(tseries) # 时间序列预处理
library(aTSA) # ARIMA 模型显著性检验


###################
# Part 1
#
# 数据导入与预处理
###################

data <- import("000063 历史数据.csv")
data <- data[order(data$日期),]
data <- data$收盘

# 定义时间步长
time_steps <- 8

create_dataset <- function(data,look_back) {
  x <- matrix(0,length(data)-look_back,look_back)
  y <- numeric(length(data)-look_back)

  for (i in 1:(length(data)-look_back)) {
    x[i, ] <- data[i:(i + look_back - 1)]
    y[i] <- data[i + look_back]
  }
```

```
  return(list(x = x, y = y))
}
dataset <- create_dataset(data,time_steps)
x <- dataset$x
y <- dataset$y

# 划分训练集和验证集(7:3)
set.seed(20241218)
train_size <- floor(0.7 * length(y))
X_train <- as.matrix(x[1:train_size,])
y_train <- y[1:train_size]
X_test <- as.matrix(x[(train_size + 1):length(y),])
y_test <- y[(train_size + 1):length(y)]

# 深度学习模型需要三维输入数据,格式为(样本数, 时间步长, 特征数)
# 将 X_train 和 X_test 转换为三维数组,特征数为 1
X_train <- array(X_train,dim = c(dim(X_train)[1],dim(X_train)[2],1))
X_test <- array(X_test,dim = c(dim(X_test)[1],dim(X_test)[2],1))


####################
# Part 2
#
# 构建训练 LSTM 模型
####################

# 构建 LSTM 模型
LSTM <- keras_model_sequential() %>%
  layer_lstm(units = 200,input_shape = c(time_steps,1),
             return_sequences = FALSE) %>%
  layer_dense(units = 1)

# 编译模型
# 指定损失函数为 mean_squared_error(均方误差),优化器为 adam
LSTM %>% compile(
  loss = 'mean_squared_error',  # 均方误差
  optimizer = 'adam'  # 使用 Adam 优化器
)

# 训练模型
LSTM %>% fit(X_train,y_train,epochs = 200,batch_size = 32,
             validation_data = list(X_test,y_test))
```

```
####################
# Part 3
#
# 构建训练 GRU 模型
####################

# 构建 GRU 模型
GRU <- keras_model_sequential() %>%
  layer_gru(units = 200,activation = 'relu',input_shape=c(time_steps,1),
            return_sequences = TRUE) %>%
  layer_gru(units = 200,activation = 'relu') %>%  # 添加第二层 GRU
  layer_dropout(rate = 0.2) %>%  # 添加 Dropout 层来防止过拟合
  layer_dense(units = 1)  # 输出一个单一的预测值

# 编译模型,设置更小的学习率
GRU %>% compile(
  loss = 'mean_squared_error',  # 使用均方误差作为损失函数
  optimizer = optimizer_adam(lr = 0.001)  # 调整学习率
)

# 训练模型
GRU %>% fit(
  X_train, y_train,
  epochs = 200,  # 增加训练轮数
  batch_size = 32,
  validation_data = list(X_test, y_test),
  verbose = 1
)


####################
# Part 4
#
# 构建训练 RNN 模型
####################

# 构建 RNN 模型
RNN <- keras_model_sequential() %>%
  layer_simple_rnn(units = 200,activation = 'relu',
                   input_shape = c(time_steps,1)) %>%
  layer_dense(units = 1)  # 输出一个预测值
```

```
# 编译模型
RNN %>% compile(
  loss = 'mean_squared_error',  # 使用均方误差作为损失函数
  optimizer = 'adam'  # 使用 Adam 优化器
)

# 训练模型
RNN %>% fit(
  X_train,y_train,
  epochs = 200,
  batch_size = 32,
  validation_data = list(X_test, y_test)
)


############################
# Part 5
#
# 构建训练 CW-RNN 模型
############################

# 构建 CW-RNN 模型（使用耦合的 RNN 结构）
CWRNN <- keras_model_sequential() %>%
  # 第一层 RNN（GRU 或 LSTM）层
  layer_simple_rnn(units = 200,activation = 'relu',input_shape = c(time_steps,1),
                   return_sequences = TRUE) %>%
  # 第二层 RNN(GRU 或 LSTM)层,耦合权重
  layer_simple_rnn(units = 200,activation = 'relu',return_sequences = TRUE) %>%
  # 输出层,预测单一的数值
  layer_dense(units = 1)

# 编译模型
CWRNN %>% compile(
  loss = 'mean_squared_error',  # 使用均方误差作为损失函数
  optimizer = 'adam'  # 使用 Adam 优化器
)

# 训练模型
CWRNN %>% fit(
  X_train, y_train,
  epochs = 100,  # 训练的轮数
  batch_size = 32,
  validation_data = list(X_test, y_test)
```

```
)


########################
# Part 6
#
# Arima 模型的建立与检验
########################

#  划分训练集和验证集(7:3)
set.seed(20241218)
train_size <- floor(0.7 * length(data))
traindata <- data[1:train_size]
testdata <- data[(train_size+1):length(data)]

# ADF 检验平稳性
adf <- adf.test(data)
print(adf) # p-value > 0.05,序列不平稳,需要差分
#  进行差分
diff_data <- diff(data)
adf_diff <- adf.test(diff_data)
#  白噪声检验
for (k in 1:3) print (Box.test(diff_data,lag = 6*k,type = "Ljung-Box"))

#  自动拟合 ARIMA 模型
ARIMA <- auto.arima(traindata)
summary(ARIMA)


########################
# Part 7
#
#  测试各模型的预测效果
########################

predictions_LSTM <- LSTM %>% predict(X_test)
predictions_GRU <- GRU %>% predict(X_test)
predictions_RNN <- RNN %>% predict(X_test)
predictions_CWRNN <- CWRNN %>% predict(X_test)
predictions_ARIMA <- as.numeric(
   forecast::forecast(ARIMA,h=length(testdata))$mean)

par(xaxs="i",yaxs="i",mar=c(5,5,2,2))
```

```
plot(y_test,lwd=3,type="l",bty="l",col=1,lty=1,xaxt="n",yaxt="n",
     xlab="",ylab="",ylim=c(20,40),xlim=c(0,90))
axis(1,las=1,cex.axis=1.5,tcl=-0.4,padj=-0.3,lwd=1)
axis(2,las=1,cex.axis=1.5,tcl=-0.4,hadj=0.9,lwd=1)
lines(predictions_LSTM,lwd=3,col=2)
lines(predictions_GRU,lwd=3,col=3)
lines(predictions_RNN,lwd=3,col=4)
lines(predictions_CWRNN,lwd=3,col=5)
lines(predictions_ARIMA,lwd=3,col=6)
title(xlab = "Time",font.lab=7,cex.lab=1.5,line=2.6)
title(ylab = "Actual & Prediction",font.lab=7,cex.lab=1.5,line=2.6)
title(main = "Time Series Actual & Prediction")
legend(0,40,c("Actual","LSTM","GRU","RNN","CWRNN","ARIMA"),lwd=3,col=c(1:6),
       bty="n",xpd=TRUE)


#########################
# Part 8
#
# 各模型的预测评价指标
#########################

# 各类模型的预测误差
error_LSTM <- predictions_LSTM-y_test
error_GRU <- predictions_GRU-y_test
error_RNN <- predictions_RNN-y_test
error_CWRNN <- predictions_CWRNN-y_test
error_ARIMA <- predictions_ARIMA-testdata

# 均方误差(MSE)
MSE_LSTM <- LSTM %>% evaluate(X_test, y_test)
MSE_GRU <- GRU %>% evaluate(X_test, y_test)
MSE_RNN <- RNN %>% evaluate(X_test, y_test)
MSE_CWRNN <- CWRNN %>% evaluate(X_test, y_test)
MSE_ARIMA <- mean((error_ARIMA)^2)
cat("各类模型的均方误差(MSE)为","\n",
    "LSTM:",MSE_LSTM,"\n",
    "GRU:",MSE_GRU,"\n",
    "RNN:",MSE_RNN,"\n",
    "CWRNN:",MSE_CWRNN,"\n",
    "ARIMA:",MSE_ARIMA)

# 均方根误差(RMSE)
```

```
RMSE_LSTM <- sqrt(mean(error_LSTM^2))
RMSE_GRU <- sqrt(mean(error_GRU^2))
RMSE_RNN <- sqrt(mean(error_RNN^2))
RMSE_CWRNN <- sqrt(mean(error_CWRNN^2))
RMSE_ARIMA <- sqrt(mean(error_ARIMA^2))
cat("各类模型的均方根误差(RMSE)为","\n",
      "LSTM:",RMSE_LSTM,"\n",
      "GRU:",RMSE_GRU,"\n",
      "RNN:",RMSE_RNN,"\n",
      "CWRNN:",RMSE_CWRNN,"\n",
      "ARIMA:",RMSE_ARIMA)

# 平均绝对误差(MAE)
MAE_LSTM <- mean(abs(error_LSTM))
MAE_GRU <- mean(abs(error_GRU))
MAE_RNN <- mean(abs(error_RNN))
MAE_CWRNN <- mean(abs(error_CWRNN))
MAE_ARIMA <- mean(abs(error_ARIMA))
cat("各类模型的平均绝对误差(MAE)为","\n",
      "LSTM:",MAE_LSTM,"\n",
      "GRU:",MAE_GRU,"\n",
      "RNN:",MAE_RNN,"\n",
      "CWRNN:",MAE_CWRNN,"\n",
      "ARIMA:",MAE_ARIMA)

# 平均绝对误差百分比(MAPE)
MAPE_LSTM <- mean(abs(error_LSTM/y_test))
MAPE_GRU <- mean(abs(error_GRU/y_test))
MAPE_RNN <- mean(abs(error_RNN/y_test))
MAPE_CWRNN <- mean(abs(error_CWRNN/y_test))
MAPE_ARIMA <- mean(abs(error_ARIMA/testdata))
cat("各类模型的平均绝对偏差百分比(MAPE)为","\n",
      "LSTM:",MAPE_LSTM,"\n",
      "GRU:",MAPE_GRU,"\n",
      "RNN:",MAPE_RNN,"\n",
      "CWRNN:",MAPE_CWRNN,"\n",
      "ARIMA:",MAPE_ARIMA)

# 汇总评价指标
MSE <- c(MSE_LSTM,MSE_GRU,MSE_RNN,MSE_CWRNN,MSE_ARIMA)
RMSE <- c(RMSE_LSTM,RMSE_GRU,RMSE_RNN,RMSE_CWRNN,RMSE_ARIMA)
MAE <- c(MAE_LSTM,MAE_GRU,MAE_RNN,MAE_CWRNN,MAE_ARIMA)
MAPE <- c(MAPE_LSTM,MAPE_GRU,MAPE_RNN,MAPE_CWRNN,MAPE_ARIMA)
```

```
matrix(rbind(MSE,RMSE,MAE,MAPE),nrow = 4,ncol = 5,
       dimnames = list(c("MSE","RMSE","MAE","MAPE"),
                       c("LSTM","GRU","RNN","CWRNN","ARIMA")))


##########################################
# 项目名称:时间序列分析--模拟分析
# 软件版本:R 4.4.1
# 攥写日期:2024.12.24
##########################################

#安装并加载相关包

#install.packages("tensorflow");install.packages("reticulate")
#install.packages("keras");install.packages("tidyverse");install.packages("rio")
#install.packages("forecast");install.packages("tseries")
library(tensorflow);library(reticulate);library(keras) # 用于进行深度学习
library(rio) # 用于数据导入
library(tidyverse) # 用于数据处理
library(forecast) # ARIMA 模型的预测
library(tseries) # 时间序列预处理
library(aTSA) # ARIMA 模型显著性检验


####################
# Part 1
#
# 数据导入与预处理
####################

set.seed(20241224)
time_series_data <- data.frame(time = 1:300, value = cumsum(rnorm(300)))
ts <- ts(time_series_data$value)
par(xaxs="i",yaxs="i",mar=c(5,5,2,2))
plot(ts,lwd = 3,type="l",bty="l",lty=1,xaxt="n",yaxt="n",xlab="",ylab="")
axis(1,las=1,cex.axis=1.5,tcl=-0.4,padj=-0.3,lwd=1)
axis(2,las=1,cex.axis=1.5,tcl=-0.4,hadj=0.9,lwd=1)
title(xlab = "Time",font.lab=7,cex.lab=1.5,line=2.6)
title(ylab = "Value",font.lab=7,cex.lab=1.5,line=2.6)
data <- as.numeric(ts(time_series_data$value))


# 定义时间步长
```

```
time_steps <- 8

create_dataset <- function(data,look_back) {
  x <- matrix(0,length(data)-look_back,look_back)
  y <- numeric(length(data)-look_back)

  for (i in 1:(length(data)-look_back)) {
    x[i, ] <- data[i:(i + look_back - 1)]
    y[i] <- data[i + look_back]
  }
  return(list(x = x, y = y))
}
dataset <- create_dataset(data,time_steps)
x <- dataset$x
y <- dataset$y

# 划分训练集和验证集(7:3)
train_size <- floor(0.7 * length(y))
X_train <- as.matrix(x[1:train_size,])
y_train <- y[1:train_size]
X_test <- as.matrix(x[(train_size + 1):length(y),])
y_test <- y[(train_size + 1):length(y)]

# 深度学习模型需要三维输入数据,格式为(样本数, 时间步长, 特征数)
# 将 X_train 和 X_test 转换为三维数组,特征数为 1
X_train <- array(X_train,dim = c(dim(X_train)[1],dim(X_train)[2],1))
X_test <- array(X_test,dim = c(dim(X_test)[1],dim(X_test)[2],1))


####################
# Part 2
#
# 构建训练 LSTM 模型
####################

# 构建 LSTM 模型
LSTM <- keras_model_sequential() %>%
  layer_lstm(units = 200,input_shape = c(time_steps,1),
             return_sequences = FALSE) %>%
  layer_dense(units = 1)

# 编译模型
# 指定损失函数为 mean_squared_error(均方误差),优化器为 adam
```

```
LSTM %>% compile(
  loss = 'mean_squared_error',  # 均方误差
  optimizer = 'adam'  # 使用 Adam 优化器
)

# 训练模型
LSTM %>% fit(X_train,y_train,epochs = 200,batch_size = 32,
             validation_data = list(X_test,y_test))


####################
# Part 3
#
# 构建训练 GRU 模型
####################

# 构建 GRU 模型
GRU <- keras_model_sequential() %>%
  layer_gru(units = 200,activation = 'relu',input_shape=c(time_steps,1),
            return_sequences = TRUE) %>%
  layer_gru(units = 200,activation = 'relu') %>%  # 添加第二层 GRU
  layer_dropout(rate = 0.2) %>%  # 添加 Dropout 层来防止过拟合
  layer_dense(units = 1)  # 输出一个单一的预测值

# 编译模型,设置更小的学习率
GRU %>% compile(
  loss = 'mean_squared_error',  # 使用均方误差作为损失函数
  optimizer = optimizer_adam(lr = 0.001)  # 调整学习率
)

# 训练模型
GRU %>% fit(
  X_train, y_train,
  epochs = 200,  # 增加训练轮数
  batch_size = 32,
  validation_data = list(X_test, y_test),
  verbose = 1
)


####################
# Part 4
#
```

```
# 构建训练 RNN 模型
####################

# 构建 RNN 模型
RNN <- keras_model_sequential() %>%
  layer_simple_rnn(units = 200,activation = 'relu',
                   input_shape = c(time_steps,1)) %>%
  layer_dense(units = 1)  # 输出一个预测值

# 编译模型
RNN %>% compile(
  loss = 'mean_squared_error',  # 使用均方误差作为损失函数
  optimizer = 'adam'  # 使用 Adam 优化器
)

# 训练模型
RNN %>% fit(
  X_train,y_train,
  epochs = 200,
  batch_size = 32,
  validation_data = list(X_test, y_test)
)


####################
# Part 5
#
# 构建训练 CW-RNN 模型
####################

# 构建 CW-RNN 模型（使用耦合的 RNN 结构）
CWRNN <- keras_model_sequential() %>%
  # 第一层 RNN（GRU 或 LSTM）层
  layer_simple_rnn(units = 200,activation = 'relu',input_shape = c(time_steps,1),
                   return_sequences = TRUE) %>%
  # 第二层 RNN(GRU 或 LSTM)层,耦合权重
  layer_simple_rnn(units = 200,activation = 'relu',return_sequences = TRUE) %>%
  # 输出层,预测单一的数值
  layer_dense(units = 1)

# 编译模型
CWRNN %>% compile(
  loss = 'mean_squared_error',  # 使用均方误差作为损失函数
```

```
  optimizer = 'adam'   # 使用 Adam 优化器
)

# 训练模型
CWRNN %>% fit(
  X_train, y_train,
  epochs = 100,   # 训练的轮数
  batch_size = 32,
  validation_data = list(X_test, y_test)
)


#########################
# Part 6
#
# Arima 模型的建立与检验
#########################

# 划分训练集和验证集(7:3)
set.seed(20241218)
train_size <- floor(0.7 * length(data))
traindata <- data[1:train_size]
testdata <- data[(train_size+1):length(data)]

# ADF 检验平稳性
adf <- adf.test(data)
print(adf) # p-value > 0.05,序列不平稳,需要差分
# 进行差分
diff_data <- diff(data)
adf_diff <- adf.test(diff_data)
# 白噪声检验
for (k in 1:3) print (Box.test(diff_data,lag = 6*k,type = "Ljung-Box"))

# 自动拟合 ARIMA 模型
ARIMA <- auto.arima(traindata)
summary(ARIMA)


########################
# Part 7
#
# 测试各模型的预测效果
########################
```

```
predictions_LSTM <- LSTM %>% predict(X_test)
predictions_GRU <- GRU %>% predict(X_test)
predictions_RNN <- RNN %>% predict(X_test)
predictions_CWRNN <- CWRNN %>% predict(X_test)
predictions_ARIMA <- as.numeric(
   forecast::forecast(ARIMA,h=length(testdata))$mean)

par(xaxs="i",yaxs="i",mar=c(5,5,2,2))
plot(y_test,lwd=3,type="l",bty="l",col=1,lty=1,xaxt="n",yaxt="n",
        xlab="",ylab="",ylim=c(10,40))
axis(1,las=1,cex.axis=1.5,tcl=-0.4,padj=-0.3,lwd=1)
axis(2,las=1,cex.axis=1.5,tcl=-0.4,hadj=0.9,lwd=1)
lines(predictions_LSTM,lwd=3,col=2)
lines(predictions_GRU,lwd=3,col=3)
lines(predictions_RNN,lwd=3,col=4)
lines(predictions_CWRNN,lwd=3,col=5)
lines(predictions_ARIMA,lwd=3,col=6)
title(xlab = "Time",font.lab=7,cex.lab=1.5,line=2.6)
title(ylab = "Actual & Prediction",font.lab=7,cex.lab=1.5,line=2.6)
title(main = "Time Series Actual & Prediction")
legend("topleft",c("Actual","LSTM","GRU","RNN","CWRNN","ARIMA"),lwd=3,col=c(1:6),
           bty="n",xpd=TRUE)

########################
# Part 8
#
# 各模型的预测评价指标
########################

# 各类模型的预测误差
error_LSTM <- predictions_LSTM-y_test
error_GRU <- predictions_GRU-y_test
error_RNN <- predictions_RNN-y_test
error_CWRNN <- predictions_CWRNN-y_test
error_ARIMA <- predictions_ARIMA-testdata

# 均方误差(MSE)
MSE_LSTM <- LSTM %>% evaluate(X_test, y_test)
MSE_GRU <- GRU %>% evaluate(X_test, y_test)
MSE_RNN <- RNN %>% evaluate(X_test, y_test)
MSE_CWRNN <- CWRNN %>% evaluate(X_test, y_test)
MSE_ARIMA <- mean((error_ARIMA)^2)
```

```
cat("各类模型的均方误差(MSE)为","\n",
    "LSTM:",MSE_LSTM,"\n",
    "GRU:",MSE_GRU,"\n",
    "RNN:",MSE_RNN,"\n",
    "CWRNN:",MSE_CWRNN,"\n",
    "ARIMA:",MSE_ARIMA)

# 均方根误差(RMSE)
RMSE_LSTM <- sqrt(mean(error_LSTM^2))
RMSE_GRU <- sqrt(mean(error_GRU^2))
RMSE_RNN <- sqrt(mean(error_RNN^2))
RMSE_CWRNN <- sqrt(mean(error_CWRNN^2))
RMSE_ARIMA <- sqrt(mean(error_ARIMA^2))
cat("各类模型的均方根误差(RMSE)为","\n",
    "LSTM:",RMSE_LSTM,"\n",
    "GRU:",RMSE_GRU,"\n",
    "RNN:",RMSE_RNN,"\n",
    "CWRNN:",RMSE_CWRNN,"\n",
    "ARIMA:",RMSE_ARIMA)

# 平均绝对误差(MAE)
MAE_LSTM <- mean(abs(error_LSTM))
MAE_GRU <- mean(abs(error_GRU))
MAE_RNN <- mean(abs(error_RNN))
MAE_CWRNN <- mean(abs(error_CWRNN))
MAE_ARIMA <- mean(abs(error_ARIMA))
cat("各类模型的平均绝对误差(MAE)为","\n",
    "LSTM:",MAE_LSTM,"\n",
    "GRU:",MAE_GRU,"\n",
    "RNN:",MAE_RNN,"\n",
    "CWRNN:",MAE_CWRNN,"\n",
    "ARIMA:",MAE_ARIMA)

# 平均绝对误差百分比(MAPE)
MAPE_LSTM <- mean(abs(error_LSTM/y_test))
MAPE_GRU <- mean(abs(error_GRU/y_test))
MAPE_RNN <- mean(abs(error_RNN/y_test))
MAPE_CWRNN <- mean(abs(error_CWRNN/y_test))
MAPE_ARIMA <- mean(abs(error_ARIMA/testdata))
cat("各类模型的平均绝对偏差百分比(MAPE)为","\n",
    "LSTM:",MAPE_LSTM,"\n",
    "GRU:",MAPE_GRU,"\n",
    "RNN:",MAPE_RNN,"\n",
```

```
    "CWRNN:",MAPE_CWRNN,"\n",
    "ARIMA:",MAPE_ARIMA)

# 汇总评价指标
MSE <- c(MSE_LSTM,MSE_GRU,MSE_RNN,MSE_CWRNN,MSE_ARIMA)
RMSE <- c(RMSE_LSTM,RMSE_GRU,RMSE_RNN,RMSE_CWRNN,RMSE_ARIMA)
MAE <- c(MAE_LSTM,MAE_GRU,MAE_RNN,MAE_CWRNN,MAE_ARIMA)
MAPE <- c(MAPE_LSTM,MAPE_GRU,MAPE_RNN,MAPE_CWRNN,MAPE_ARIMA)
matrix(rbind(MSE,RMSE,MAE,MAPE),nrow = 4,ncol = 5,
       dimnames = list(c("MSE","RMSE","MAE","MAPE"),
                       c("LSTM","GRU","RNN","CWRNN","ARIMA")))
```