MINISTRY OF EDUCATION AND TRAINING
VIETNAM ACADEMY OF SCIENCE AND TECHNOLOGY
GRADUATE UNIVERSITY OF SCIENCE AND TECHNOLOGY

Le Linh Dan

DEEP LEARNING AND TIME SERIES

MASTER'S THESIS IN MATHEMATICS

Major: Applied Mathematics

Code: 8 46 01 12

SCIENCE SUPERVISORS:
Assoc. Prof. Dr. Tran Nam Trung

Hanoi - 2025

BỘ GIÁO DỤC
VÀ ĐÀO TẠO

VIỆN HÀN LÂM KHOA HỌC
VÀ CÔNG NGHỆ VIỆT NAM

HỌC VIỆN KHOA HỌC VÀ CÔNG NGHỆ

Lê Linh Đan

HỌC SÂU VÀ CHUỖI THỜI GIAN

LUẬN VĂN THẠC SĨ TOÁN HỌC

Ngành: Toán ứng dụng

Mã số: 8 46 01 12

NGƯỜI HƯỚNG DẪN KHOA HỌC :
PGS.TS Trần Nam Trung

Hà Nội - 2025

INTRODUCTION

1. Research Context and Rationale

In the era of big data, time series forecasting has emerged as a critical decision-making tool across various domains, ranging from financial market analysis to supply chain management and transportation planning. The ability to accurately predict future values based on historical patterns allows organizations to optimize resources and mitigate risks.

For decades, classical statistical models, particularly the Autoregressive Integrated Moving Average (ARIMA) and its seasonal extension (SARIMA), have been the industry standard. These models are celebrated for their mathematical robustness and interpretability, especially when dealing with linear relationships and distinct seasonal patterns [cite: 341-342]. However, the increasing complexity of real-world data often exposes the limitations of these linear models, as they may fail to capture intricate nonlinear dynamics or complex interactions.

This limitation has driven the rise of Deep Learning approaches, such as Long Short-Term Memory (LSTM) networks, which are designed to model long-term dependencies and nonlinear structures[cite: 846]. Furthermore, the concept of "Hybrid models"—which decompose a time series into linear and nonlinear components to leverage the strengths of both ARIMA and LSTM—has become a prominent research direction.

2. Research Problem

While Hybrid models are theoretically promising, a critical gap remains in understanding their practical efficiency relative to their complexity. The central research problem addressed in this thesis is: *"Does the increased complexity of Deep Learning and Hybrid models necessarily translate to better forecasting performance compared to well-tuned statistical models, particularly for univariate datasets with strong seasonality and limited sample size?"*

To answer this, the research focuses on a comparative study using the classic **AirPassengers dataset** (1949–1960)[cite: 339, 344]. This dataset serves as the primary research object due to its non-stationarity, clear upward trend, and multiplicative seasonality, making it an ideal benchmark for testing the capabilities of different modeling families.

3. Significance of the Study

This study holds significant practical and theoretical value:

- **Methodological Evaluation:** It provides a rigorous empirical comparison of four distinct modeling approaches: ARIMA, SARIMA, LSTM, and Hybrid SARIMA-LSTM.
- **Guidance on Model Selection:** The findings offer evidence-based insights for practitioners, highlighting that model selection should be context-dependent. It challenges the assumption

that complex deep learning models are always superior, demonstrating that for specific data types (small scale, strong linearity), traditional methods like SARIMA may still be the optimal choice.

- **Implementation Framework:** The thesis provides a reproducible framework for implementing these models using Python, detailing the process from data preprocessing to performance evaluation.

4. Structure of the Thesis

To address the research objectives systematically, the thesis is organized into two main chapters:

Chapter 1: Theoretical Framework

This chapter establishes the mathematical and conceptual foundations of the study. It begins by defining the fundamental components of time series (trend, seasonality, noise) and the concept of stationarity. It then details the mechanisms of statistical models (ARIMA, SARIMA), explaining how differencing and lag operators work. Subsequently, it introduces the architecture of Artificial Neural Networks, specifically focusing on the LSTM cell structure and its gating mechanisms. Finally, it presents the theoretical methodology for constructing a Hybrid model by combining linear and nonlinear techniques.

Chapter 2: Experiment and Results

This chapter presents the empirical core of the research. It details the experimental setup using Python libraries (Statsmodels, TensorFlow/Keras). The AirPassengers dataset is analyzed and preprocessed to achieve stationarity. The chapter then walks through the training and evaluation of each model (ARIMA, SARIMA, LSTM, and Hybrid) using metrics such as Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), and Mean Absolute Percentage Error (MAPE). The chapter concludes with a comparative discussion, analyzing why the SARIMA model outperformed the Hybrid model in this specific context.

DECLARATION

This thesis has been written on the basis of my own research conducted at the Institute of Mathematics, Vietnam Academy of Science and Technology, under the supervision of Assoc. Prof. Dr. Tran Nam Trung. The results presented in this thesis are original and have not been submitted for any other degree or qualification.

Furthermore, the thesis makes use of certain comments and evaluations from other authors, all of which have been properly cited.

Hanoi, Octorber, 2025

Student

ACKNOWLEDGEMENTS

This work would not have been possible without the generous support and guidance of many individuals and institutions.

First and foremost, I am deeply grateful to the Graduate University of Science and Technology and the Institute of Mathematics, Vietnam Academy of Science and Technology, including the Board of Directors, the Training Office, and the functional departments, for providing me with the opportunity and the favorable conditions to study in a rigorous academic environment and to complete this thesis.

I would also like to acknowledge the invaluable financial support from the VinIF Foundation of Vingroup. The scholarship has enabled me to fully dedicate myself to my studies and research. I am sincerely thankful for this support and will continue to strive to meet the trust placed in me.

My profound gratitude goes to my supervisor, Assoc. Prof. Dr. Trần Nam Trung, whose dedicated guidance, insightful advice, and profound knowledge have been indispensable throughout this research journey. His encouragement and mentorship have inspired me to pursue this topic with passion and perseverance.

Last but not least, I am deeply indebted to my family and friends for their unwavering support, encouragement, and understanding, which have been a constant source of strength in overcoming the challenges of academic and personal life.

Hanoi, Octorber, 2025

Student

CONTENTS

INTRODUCTION **I**

Page

DECLARATION **III**

ACKNOWLEDGEMENTS **IV**

LIST OF FIGURES **VIII**

LIST OF SYMBOLS AND ABBREVIATIONS **VIII**

LIST OF TABLES **X**

1 Time series **1**

1.1 Definition of Time Series and its components 1
1.1.1 Definition of Time series 1
1.1.2 Components of Time Series 1
1.1.3 Time series decomposition 3

1.2 Time series classification 5
1.2.1 Univariate and multivariate time series 5
1.2.2 Stationary and Non-stationary 5

1.3 Transformations of Time Series 8
1.3.1 Purpose of Transformations 8
1.3.2 Logarithmic Transformation 8
1.3.3 Square Root Transformation 9
1.3.4 Power Transformation 9
1.3.5 Box–Cox Transformation 9
1.3.6 Differencing as a Transformation 9
1.3.7 Standardization and Normalization 10

1.4 ARIMA model 12
1.4.1 Overview 12
1.4.2 Autoregressive (AR) Component 13
1.4.3 Integrated (I) Component 14
1.4.4 Moving Average (MA) Component 15
1.4.5 Characteristic and applications 16

1.5 SARIMA Model . . . 16
1.5.1 Overview of Seasonality in Time Series . . . 16
1.5.2 Definition of SARIMA . . . 16
1.5.3 Identification of Seasonal Orders . . . 17
1.5.4 Illustrative Example: Airline Passengers Data . . . 17
1.5.5 Applications and Limitations . . . 18
1.5.6 Comparison: ARIMA vs SARIMA . . . 18

1.6 Forecasting Model Evaluation Metrics . . . 19
1.6.1 Mean Absolute Error (MAE) . . . 19
1.6.2 Root Mean Squared Error (RMSE) . . . 19
1.6.3 Mean Absolute Percentage Error (MAPE) . . . 19
1.6.4 Coefficient of Determination (R^2) . . . 20
1.6.5 Symmetric Mean Absolute Percentage Error (SMAPE) . . . 20

2 Deep learning and its application in time series forecasting 21

2.1 Artificial Neural Networks (ANN) . . . 21
2.1.1 Neurons . . . 21
2.1.2 Neural Networks . . . 22

2.2 Training artificial neural networks . . . 23
2.2.1 Loss function . . . 24
2.2.2 Backpropagation algorithm . . . 24
2.2.3 Optimization Algorithms (Optimizers) . . . 27

2.3 Convolutional Neural Networks (CNN) for Time Series . . . 29
2.3.1 From Spatial to Temporal Feature Learning . . . 29
2.3.2 Mathematical Foundation: 1D Convolution . . . 29
2.3.3 Core Architectural Components . . . 30
2.3.4 Advantages for Time Series Forecasting . . . 31
2.3.5 Limitations and the Case for Hybrid Models . . . 31
2.3.6 Illustrative Example: Detecting Weekly Patterns in Electricity Demand . . . 31

2.4 Recurrent Neural Network (RNN) . . . 32
2.4.1 Concept . . . 32
2.4.2 Basic Structure . . . 32
2.4.3 Advantages of RNN . . . 33
2.4.4 Limitations of RNN . . . 33
2.4.5 Significance in Time Series Forecasting . . . 33

2.5 Long Short-Term Memory Network Model (LSTM) . . . 33
2.5.1 Overview . . . 33
2.5.2 Operating Principle . . . 34
2.5.3 Key Features of LSTM . . . 35

2.6 Hybrid Models in Time Series Forecasting . . . 35
2.6.1 Introduction . . . 35
2.6.2 ARIMA–LSTM Hybrid Model . . . 35

2.6.3 CNN–LSTM Hybrid Model 37
2.6.4 Other Hybrid Models 38
2.6.5 Practical Implementation 38
2.6.6 Advantages and Challenges 38
2.6.7 Conclusion 39
2.7 Application in Number of Air Passengers Forecasting 39
2.7.1 Programming Language and Libraries Used 39
2.7.2 Application in Air Passenger Forecasting 40

LIST OF SYMBOLS AND ABBREVIATIONS

Symbol	**Meaning**
X_t	Value of the time series at time t
L	Lag (backshift) operator defined by $LX_t = X_{t-1}$
ΔX_t	First-order difference: $\Delta X_t = X_t - X_{t-1}$
$\Delta^d X_t$	d^{th}-order differenced series
ϕ_i	Autoregressive coefficient of order i in AR model
θ_i	Moving average coefficient of order i in MA model
ε_t	White noise error term
p, d, q	Non-seasonal ARIMA parameters (AR, differencing, MA orders)
P, D, Q, s	Seasonal ARIMA parameters and seasonal period
T_t, S_t, C_t, I_t	Trend, Seasonal, Cyclic, and Irregular components of a time series
μ	Mean of the series
σ^2	Variance of the series
$\rho(k)$	Autocorrelation coefficient at lag k
$\Phi_P(L^s), \Theta_Q(L^s)$	Seasonal AR and MA polynomials in SARIMA
$Y_t, \hat{Y}_t$	Actual and forecasted values at time t
MAE	Mean Absolute Error
$RMSE$	Root Mean Squared Error
$MAPE$	Mean Absolute Percentage Error
$SMAPE$	Symmetric Mean Absolute Percentage Error
R^2	Coefficient of determination
$\mathbf{a}^l, \mathbf{z}^l, W^l, \mathbf{b}^l$	Activation vector, pre-activation vector, weight matrix, and bias vector at layer l
$\sigma(x)$	Activation function (e.g., ReLU, Sigmoid, Tanh)
softmax$(\cdot)$	Softmax activation function for multi-class output
δ^k	Backpropagation error term at layer k
f_t, i_t, o_t, C_t, h_t	Forget gate, input gate, output gate, cell state, and hidden state in LSTM
λ	Power parameter in Box–Cox transformation

Abbreviation	**Full meaning**
AR	Autoregressive
MA	Moving Average
ARMA	Autoregressive Moving Average
ARIMA	Autoregressive Integrated Moving Average
SARIMA	Seasonal Autoregressive Integrated Moving Average
ADF	Augmented Dickey–Fuller (stationarity test)
KPSS	Kwiatkowski–Phillips–Schmidt–Shin (stationarity test)
ACF	Autocorrelation Function
PACF	Partial Autocorrelation Function

DWT	Discrete Wavelet Transform
ANN	Artificial Neural Network
CNN	Convolutional Neural Network
RNN	Recurrent Neural Network
LSTM	Long Short-Term Memory
SGD	Stochastic Gradient Descent
ReLU	Rectified Linear Unit
MSE	Mean Squared Error
BIC	Bayesian Information Criterion
AIC	Akaike Information Criterion
MAE	Mean Absolute Error
RMSE	Root Mean Squared Error
MAPE	Mean Absolute Percentage Error
SMAPE	Symmetric Mean Absolute Percentage Error
API	Application Programming Interface

List of Tables

1.1 Comparison of Additive, Multiplicative, and Wavelet Decomposition Methods 4
1.2 Types of nonstationary time series 6
1.3 Compare stationary and nonstationary time series 7
1.4 Comparison between ARIMA and SARIMA 18

2.1 Comparison of ARIMA, SARIMA, LSTM, and Hybrid SARIMA–LSTM models 45

List of Figures

1.1 Components of a time series 2
1.2 Raw synthetic time series with exponential growth and multiplicative oscillations. The variance increases as the series level grows. 11
1.3 Logarithmic transformation reduces variance and converts multiplicative effects into additive effects. 11
1.4 Box–Cox transformation with $\lambda = 0.3$ provides flexible variance stabilization, with $\lambda = 0$ recovering the logarithmic case. 12
1.5 Distribution of monthly passenger counts 17
1.6 Time series of monthly passengers 17

2.1 Artificial neuron. 22
2.2 Basic neural network. 23
2.3 Sigmoid function and its derivative 26
2.4 Optimization algorithms in Deep Learning 28
2.5 Basic structure of RNN 32
2.6 ARIMA–LSTM hybrid model workflow 37
2.7 Distribution of monthly passenger counts 40
2.8 Time series of monthly passengers 41
2.9 ACF plot 41
2.10 12-month forecast with ARIMA model 42
2.11 12-month forecast with SARIMA model 43
2.12 12-month forecast with LSTM model 44
2.13 Forecasting with Hybrid SARIMA–LSTM model 45

Abstract

Time series forecasting is a fundamental problem in applied mathematics and data science, with critical implications in domains such as finance, energy, and transportation. Classical statistical approaches, including Autoregressive Integrated Moving Average (ARIMA) and its seasonal extension (SARIMA), provide a rigorous framework for modeling stationary linear processes with trend and seasonality. Their reliance on differencing, autoregressive (AR) terms, and moving average (MA) components allows for efficient parameterization; however, they exhibit limited ability to capture nonlinear dynamics and long-range temporal dependencies.

In contrast, Deep learning architectures such as Long Short-Term Memory (LSTM) networks are specifically designed to mitigate vanishing gradient issues in recurrent neural networks by introducing memory cells and gating mechanisms. This enables effective modeling of nonlinear and long-horizon dependencies in complex time series. Nevertheless, LSTMs require large-scale data, careful hyperparameter optimization, and computational resources that may limit their applicability in small-sample settings.

This thesis presents a comparative study of ARIMA, SARIMA, LSTM, and a hybrid SARIMA–LSTM model for univariate time series forecasting. The hybrid model is constructed by decomposing the series into linear and nonlinear components, where SARIMA captures the stochastic linear structure and LSTM is applied to the residual series to approximate nonlinearities. Empirical experiments are conducted using benchmark datasets, including the AirPassengers dataset [2], with model performance evaluated through Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), and Mean Absolute Percentage Error (MAPE).

The research contributes a critical assessment of the methodological trade-offs between statistical and deep learning models. The results highlight that while Hybrid models are theoretically robust, their empirical success is highly context-dependent. Specifically, in datasets with strong linear seasonality and limited sample size, classical models like SARIMA may outperform complex hybrid architectures. This provides a foundational framework for selecting appropriate forecasting strategies based on data characteristics.

Keywords: Time Series Forecasting, ARIMA, SARIMA, LSTM, Hybrid Models, Deep Learning.

Chapter 1

Time series

In many research areas and real-world applications, data are frequently collected in chronological order. These datasets, known as time series, are essential for analysis, forecasting, and informed decision-making [1]. In finance, time series are particularly valuable for examining stock price movements, interest rates, market indices, and a wide range of economic indicators. Gaining a solid understanding of the characteristics and components of time series is a critical first step in choosing the right forecasting method—whether it is a traditional model like ARIMA [2] or a modern approach based on Deep Learning [3].

1.1. Definition of Time Series and its components

1.1.1. Definition of Time series

Definition 1.1.1. A time series is a sequence of data points recorded or measured at successive points in time, denoted as X_t where t indexes time.

$$\{X_t\}_{t=1}^{T} = (X_1, X_2, \ldots, X_T)$$

where X_t is the observed value at time t, and T is the total number of observations. [1, 2]

Time series data can be collected at regular intervals (e.g., daily, weekly, monthly) or irregular intervals. This thesis focuses on data that are continuous in value but discrete in time. Unlike cross-sectional data, the properties of a time series depend on time, the present value depends on or correlated with its past values.

Next we define the lag operator which is used to simplify the notation of all models presented later.

Definition 1.1.2. The lag operator L, also called the back-shift operator, is an operator that shifts a time series one period backward:

$$LX_t = X_{t-1}.$$

Remark 1.1.3. By using the lag operator shifts a time series for k periods backward, we obtain

$$L^k X_t = X_{t-k}.$$

1.1.2. Components of Time Series

A time series is typically influenced by several underlying components that explain different types of variation in the data. The classical decomposition framework distinguishes four main components: trend, seasonality, cyclic fluctuations, and irregular variations [1, 2]. Each of these components contributes to the observed series in different ways.

The *trend* represents the long-term direction of a time series, which may be upward, downward, or relatively stable. It captures the persistent movement of the data over an extended period and can take various forms such as linear, exponential, or nonlinear growth. A simple linear trend can be expressed as

$$T_t = \beta_0 + \beta_1 t + \epsilon_t,$$

where β_0 is the intercept, β_1 is the slope, and ϵ_t represents random noise. An illustrative example of a trend is the steady increase in the average height of adolescents between ages 13 and 16, which reflects biological growth over time.

The *seasonal component* refers to short-term, repetitive, and predictable fluctuations that occur at fixed intervals such as daily, monthly, or yearly. These variations are caused by recurring external factors such as climate conditions, holidays, or cultural events. A purely seasonal component is denoted by S_t. For instance, monthly ice cream sales typically peak during summer months and fall during winter, reflecting the influence of temperature on consumer behavior.

The *cyclic component*, denoted by C_t, reflects long-term oscillations that recur but not at fixed or predetermined intervals. Unlike seasonality, which has a fixed calendar length, cycles vary in duration and are often linked to broader macroeconomic or business dynamics [1]. For example, economic boom-and-bust cycles typically last several years but do not have a fixed period.

Irregular Variations

The *irregular component*, denoted by I_t, encompasses unpredictable and erratic fluctuations that cannot be attributed to trend, seasonality, or cycles. These variations are typically short-lived and result from unforeseen events or external shocks. Examples include sudden drops in production due to natural disasters such as floods or earthquakes, or sharp market disruptions triggered by wars or pandemics. Since these events lack systematic patterns, they are often treated as random noise in modeling and forecasting.

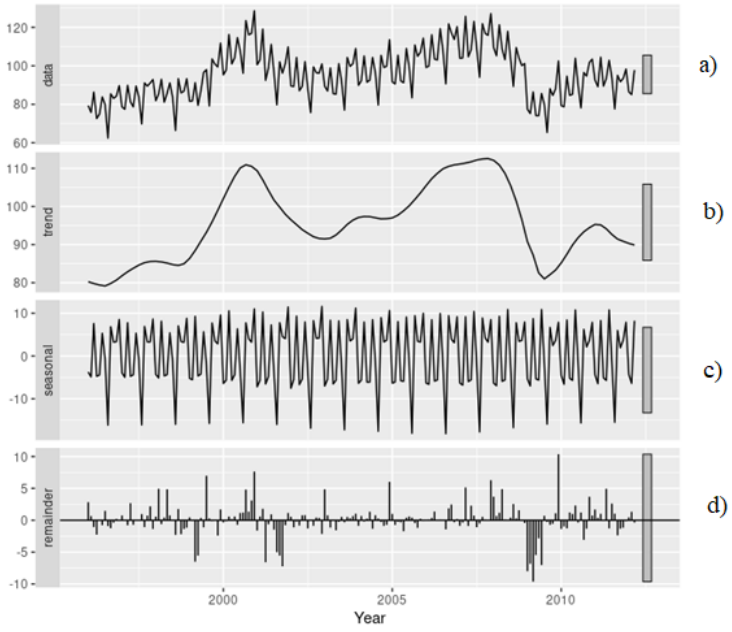


Figure 1.1: Components of a time series

In figure 1.1, the graph a) describes a time series that collects the number of orders of electrical devices. Graph (b) shows a significant increasing trend from 2000 to 2001 and between 2005 and 2007. The graph c) presents a regular change in the data. Lastly, the graph d) describes the irregular component (or noise) remaining after the trend and seasonal components have been removed.

The four classical components (trend, seasonality, cyclic, irregular) are essential but often inadequate when the underlying process exhibits non-linear dynamics. To address this, Wavelet decomposition provides a powerful tool for multi-resolution analysis [4, 5].

1.1.3. Time series decomposition

Additive Decomposition

In the additive framework, the observed time series is assumed to be the sum of its underlying components: trend, seasonality, cyclic fluctuations, and irregular noise. The model is written as

$$X_t = T_t + S_t + C_t + I_t,$$

where T_t denotes the trend, S_t the seasonal variation, C_t the cyclic component, and I_t the irregular element. Additive decomposition is most appropriate when the magnitude of seasonal fluctuations and irregular variations remains constant over time, regardless of the level of the series. For example, in monthly sales data of a small retail shop, the additional number of customers during holiday months is roughly constant each year, independent of the long-term growth trend. On a daily scale, consider the number of passengers boarding a city bus: there may be small but regular increases during weekdays compared to weekends, and these increments remain fairly stable over the year.

Multiplicative Decomposition

The multiplicative model assumes that the components interact in a proportional way, such that the observed time series is expressed as

$$X_t = T_t \times S_t \times C_t \times I_t.$$

In this case, the seasonal and cyclic effects scale with the overall level of the series, meaning that higher trend values are accompanied by larger absolute seasonal swings. Multiplicative decomposition is commonly applied in economic and financial data where relative (percentage) changes are more meaningful than absolute differences. For instance, if summer sales of air conditioners are consistently 20% higher than the yearly average, then the seasonal effect grows in absolute magnitude as the overall market expands. On a daily level, electricity demand often follows a multiplicative structure: during peak trend periods such as winter heating season, daily fluctuations between daytime and nighttime usage become proportionally larger compared to off-season months.

Wavelet Decomposition

In addition to the four classical components (trend, seasonality, cyclic, and irregular), modern time series analysis often employs wavelet methods to capture both short-term fluctuations and long-term structures simultaneously. The *Discrete Wavelet Transform* (DWT) provides a multi-resolution decomposition of the series, enabling the separation of smooth trends from high-frequency variations

[4, 5]. A time series X_t can be expressed as

$$X_t = \sum_k a_{j_0,k}\,\phi_{j_0,k}(t) \;+\; \sum_{j=j_0}^{J}\sum_k d_{j,k}\,\psi_{j,k}(t),$$

where $\phi_{j_0,k}(t)$ denotes the scaling functions at a coarse resolution, capturing the long-term approximation of the series, while $\psi_{j,k}(t)$ represents the wavelet functions at scale j, which describe localized fluctuations. The coefficients $a_{j_0,k}$ and $d_{j,k}$ correspond to approximation and detail components, respectively. This decomposition allows practitioners to analyze non-stationary signals by isolating structural shifts from noise across multiple frequency bands, making wavelets particularly effective for financial, biomedical, and climatological time series.

Table 1.1: Comparison of Additive, Multiplicative, and Wavelet Decomposition Methods

Aspect	**Additive**	**Multiplicative**	**Wavelet**
Assumption	Seasonal and irregular effects remain constant over time.	Seasonal and irregular effects scale with the series level.	Series contains both short- and long-term frequency variations.
Typical data	Stable seasonal amplitude (e.g., small retail sales).	Proportional seasonal amplitude (e.g., financial or energy data).	Non-stationary or noisy signals (e.g., biomedical or climate data).
Advantages	Simple, interpretable, and easy to implement.	Captures proportional or percentage-based changes.	Effective for non-stationary, multi-scale patterns.
Limitations	Fails with heteroscedastic variance.	Requires positive data; may need log transformation.	Computationally complex and harder to interpret.

Table 1.1 summarizes the conceptual differences among the three major decomposition approaches. The additive and multiplicative models belong to the classical decomposition family, assuming the existence of global trend and seasonal structures that can be isolated by algebraic operations [10]. The additive form is suitable when seasonal fluctuations have a constant magnitude, whereas the multiplicative model is appropriate when these fluctuations grow with the level of the series, such as in economic and demographic data. In contrast, the wavelet-based approach extends beyond these static frameworks by decomposing a signal into localized time–frequency components, enabling a simultaneous view of both transient and persistent patterns [4,5]. This flexibility makes wavelet methods especially valuable for modern applications involving complex, non-stationary, or nonlinear time series.

1.2. Time series classification

1.2.1. Univariate and multivariate time series

Univariate time series

An univariate time series is a sequence of data that observes only one characteristic of a phenomenon and then expresses it as a variable denoted as

$$X_t, \quad t = 1, 2, \ldots, T$$

where X_t is the variable value at moment t and T is the number of time points.

Example 1.2.1. A daily example for an univariate time series is a series recording the temperature of Hanoi in a day.

Multivariate time series

Definition 1.2.1. A multivariate time series is a set of data points indexed by time, where each data point consists of multiple related variables recorded simultaneously.

Mathematically, a multivariate time series can be formulated as a sequence of vector as

$$\mathbf{X}_t = \begin{bmatrix} X_{1,t} \\ X_{2,t} \\ \vdots \\ X_{n,t} \end{bmatrix}, \quad t = 1, 2, \ldots, T$$

where $X_{i,t}$ is the value of i^{th} feature at the time t and n is the number of features examined.

Example 1.2.2. Daily Multivariate COVID-19 Time Series. An example of a multivariate time series can be found in daily epidemiological records during the COVID-19 pandemic. Each observation consists of several related variables such as the number of new confirmed cases, hospitalizations, tests conducted, and vaccinations administered. Together, these variables form a sequence of daily vectors that evolve over time, capturing both temporal dynamics and inter-variable relationships.

Unlike univariate time series, multivariate time series deal with both the temporal dynamics of each individual variable and the correlations among variables. This dual perspective improves explanatory power and improves the accuracy of forecasting models, especially when dealing with complex phenomena such as economics, finance, or meteorology. Such as forecasting stock prices with many economic indicators.

1.2.2. Stationary and Non-stationary

Stationary is a key characteristic of modeling time series. A time series is stationary if its statistical properties such as: mean, variance, auto-covariance,... do not change over time [2].

Stationary time series

Definition 1.2.2. A time series $\{X_t\}$ is strictly stationary if the joint distribution of $(X_{t_1}, \ldots, X_{t_k})$ is invariant under time shift for all $t_1, \ldots, t_k$. [6]

A weaker form, weak (or second-order) stationarity [2], requires:

1. Constant mean: $\mathbb{E}[X_t] = \mu \quad \forall t$
2. Constant variance: $\mathrm{Var}(X_t) = \sigma^2 \quad \forall t$
3. Auto-covariance depends only on lag h : $\gamma(h) = \mathrm{Cov}(X_t, X_{t+h})$

Based on the definition, weak stationarity excludes stochastic trend; deterministic seasonality must be removed (seasonal dummies or differencing). When a series is stationary, the model can reliably learn relationships between past and future values without worrying about shifting averages, changing variability, or evolving correlations. This stability makes the model parameters consistent and improves the accuracy of forecast. If the data is non-stationary (e.g., has trends or seasonality), it often needs to be transformed — through differencing, de-trending, or removing seasonality — to become stationary before modeling.

Nonstationary time series

Definition 1.2.3. A nonstationary time series is a series whose statistical properties change over time.

There are some typical types of nonstationary time series which are listed in the following table:

Table 1.2: Types of nonstationary time series

Type of Non-stationarity	**Symptom**	**How to detect it**	**Example**
Mean (trend)	Increasing/decreasing mean	Plot, ACF slow decay, ADF test	Stock price, GDP
Variance	Changing volatility over time	Plot, rolling variance, ARCH test	Financial returns
Seasonal	Repeated cycles	Plot, ACF spikes at seasonal lags	Sales, electricity demand
Structural break	Sudden changes in level/trend	CUSUM, Chow test	Exchange rate after crisis

Example 1.2.3. A simple and intuitive example of a nonstationary time series in daily life is the daily average temperature in a city across several years.

In such a series, the mean and variance change over time — temperatures tend to rise during summer months and fall during winter, exhibiting a clear seasonal pattern and often a long-term warming trend due to climate change. This violates the assumption of stationarity because the statistical properties (mean, variance, autocorrelation) are not constant throughout the year.

Comparison of stationary and nonstationary time series

Table 1.3: Compare stationary and nonstationary time series

Properties	**Nonstationary**	**Stationary**
Stability	Mean and variance changes over times; the autocorrelation functions tends to fluctuate.	Mean and variance do not change over times; the autocorrelation functions depends only on lag, not on time.
Forecasting	Handle nonstationary such as trend and variance (by differencing, de-trending, or seasonal adjustment).	Apply simple models such as ARMA and ARIMA.

Checking stationarity

In practice, to check the properties of stationary, we often use the Augmented Dickey-Fuller test (ADF) [7]. The main idea of this test is to test the hypothesis:

$$H_0 : \text{Nonstationary} \quad \text{compare with} \quad H_1 : \text{Stationary}.$$

The mathematical formula for an ADF test is as follows:

$$\Delta X_t = \alpha + \beta t + \gamma X_{t-1} + \sum_{i=1}^{p} \delta_i \Delta X_{t-i} + \varepsilon_t$$

where:

- $\Delta X_t = X_t - X_{t-1}$ is the first difference.
- t is time variable.
- p is the degree of lag operator.
- ε_t is white noise.

Reject H_0 (unit root) if the ADF test statistic is less than the critical value; otherwise fail to reject. The result of the test decides whether or not we need to make the data stationary before building and applying models.

Other tests include KPSS [8] and Phillips–Perron, which provide complementary evidence of stationarity.

Method to achieve stationarity

Most of the time series collected in daily life is nonstationary. To apply time series traditional models such as ARMA, the series need to be transform to stationary one. Some widely used methods can be listed as follows:

1. **Differencing** is a method to eliminate seasonality or trend in the series. The d^{th} differencing is defined as:

$$\Delta^d X_t = (1 - L)^d X_t$$

where $LX_t = X_{t-1}$ as in Definition 1.1.2. Differencing is typically applied at lag 1 or 2 as:

$$\Delta X_t = X_t - X_{t-1}$$
$$\Delta^2 X_t = \Delta(\Delta X_t) = (X_t - X_{t-1}) - (X_{t-1} - X_{t-2}).$$

2. **De-trending** is used for series that has linear or nonlinear trend to estimate the trend then eliminate it as $X_t^* = X_t - T_t$ where T_t is estimated by:
   - Linear regression: $T_t = \alpha + \beta t$.
   - Multiple regression.
   - Smoothing such as moving average.
3. **Variance Stabilization.** is applied for time series that is heteroskedasticity. This method take square root or logarithm of all values in the series to stabilize the variance as:
$$Y_t = \log(X_t) \quad \text{or} \quad Y_t = \sqrt{X_t}.$$
4. **Deseasonalization** is used for series whose trend repeats periodically. Normally, its formula is
$$X_t^* = X_t - S_t$$
where S_t is estimated by moving average or time series decomposition methods.

1.3. Transformations of Time Series

Transformations are applied to time series data in order to stabilize variance, linearize relationships, and prepare the data for modeling. In practice, many real-world time series exhibit heteroscedasticity, multiplicative seasonality, or nonlinear dynamics, which make raw data unsuitable for direct application of classical forecasting methods. By applying an appropriate transformation, we can achieve a stationary and well-behaved series, which simplifies both interpretation and modeling.

1.3.1. Purpose of Transformations

The main objectives of transformations are to stabilize the variance over time, convert multiplicative effects into additive effects, make the distribution of the series closer to normal, and enhance the interpretability of model coefficients. In summary, transformations act as a bridge between the raw series and the statistical assumptions required by forecasting models.

1.3.2. Logarithmic Transformation

Definition 1.3.1. The logarithmic transformation is defined as

$$Y_t = \log(X_t),$$

where $X_t > 0$. [2]

The logarithmic transformation compresses large values more strongly than small ones, which reduces the effect of extreme observations and stabilizes the variance. It also converts multiplicative seasonality into additive seasonality, making it easier to apply linear models. This is especially useful when the magnitude of fluctuations grows proportionally with the level of the series.

Example 1.3.1. Monthly sales that grow proportionally with the demand can be stabilized by log transformation, making seasonal peaks appear constant in absolute terms.

1.3.3. Square Root Transformation

Definition 1.3.2. The square root transformation is defined as

$$Y_t = \sqrt{X_t}, \quad X_t \geq 0.$$

This transformation is weaker than the logarithm but still reduces the impact of large values. It is particularly suitable for count data such as arrivals, number of claims, or disease incidence, where the variance is approximately proportional to the mean. By compressing the scale of high observations, it balances the fluctuations and helps improve model fit.

1.3.4. Power Transformation

Definition 1.3.3. A power transformation raises the series to a power λ:

$$Y_t = X_t^{\lambda}, \quad \lambda \in \mathbb{R}.$$

Power transformations form a family of transformations where λ controls the degree of compression. If $\lambda < 1$, the transformation reduces the effect of large values (for example, $\sqrt{X_t}$ when $\lambda = 0.5$). If $\lambda > 1$, the transformation expands large values. This flexibility allows the researcher to adjust the scale of the data depending on the degree of heteroscedasticity.

1.3.5. Box–Cox Transformation

Definition 1.3.4. The Box–Cox transformation generalizes power and log transformations as follows:

$$Y_t = \begin{cases} \dfrac{X_t^{\lambda} - 1}{\lambda}, & \lambda \neq 0, \\ \log(X_t), & \lambda = 0, \end{cases}$$

where $X_t > 0$ and λ are the transformation parameters. [9]

The Box–Cox transformation is a flexible tool that automatically selects the best value of λ based on maximum likelihood estimation. This unifies several transformations in a single framework, where $\lambda = 1$ corresponds to the identity, $\lambda = 0$ corresponds to the logarithm, and intermediate values represent different levels of variance stabilization. The Box–Cox transformation is widely used because it adapts to the distribution of the data rather than requiring a fixed choice.

Example 1.3.2. The variance of airline passenger counts, which increases over time, can be stabilized using a Box–Cox transformation with an estimated $\lambda \approx 0.3$.

1.3.6. Differencing as a Transformation

Although differencing is primarily associated with the integrated part of ARIMA, it can also be regarded as a transformation.

Definition 1.3.5. The first-order difference is given by

$$\Delta X_t = X_t - X_{t-1}.$$

Definition 1.3.6. The second-order difference is defined as

$$\Delta^2 X_t = \Delta(\Delta X_t) = (X_t - X_{t-1}) - (X_{t-1} - X_{t-2}).$$

Differencing removes trends or seasonal components by eliminating systematic patterns in the data. For example, first-order differencing removes linear trends, while seasonal differencing $\Delta_s X_t = X_t - X_{t-s}$ removes recurring seasonal patterns. As a result, differencing transforms a nonstationary series into a stationary one, which is required for ARMA-type models.

1.3.7. Standardization and Normalization

Definition 1.3.7. Standardization rescales data to have zero mean and unit variance:

$$Z_t = \frac{X_t - \mu}{\sigma}.$$

Definition 1.3.8. Normalization rescales data into a fixed range $[0, 1]$:

$$Z_t = \frac{X_t - \min(X)}{\max(X) - \min(X)}.$$

Standardization ensures that the data are centered and scaled, which is important for models that rely on distance or assume normally distributed residuals. Normalization preserves the relative scale of the series but constrains it to a fixed interval, which is particularly useful in neural networks where activation functions (such as sigmoid or tanh) operate within bounded ranges. Both methods enhance numerical stability and convergence during training.

Example 1.3.3. To illustrate the role of transformations, we generate a synthetic time series of length $T = 100$ with exponential growth and oscillatory components. The raw series is defined as

$$X_t = e^{0.03t} \times (1 + 0.30\sin(0.20t) + 0.15\sin(1.30t) + 0.05\cos(0.55t)), \quad t = 1, 2, \ldots, 100.$$

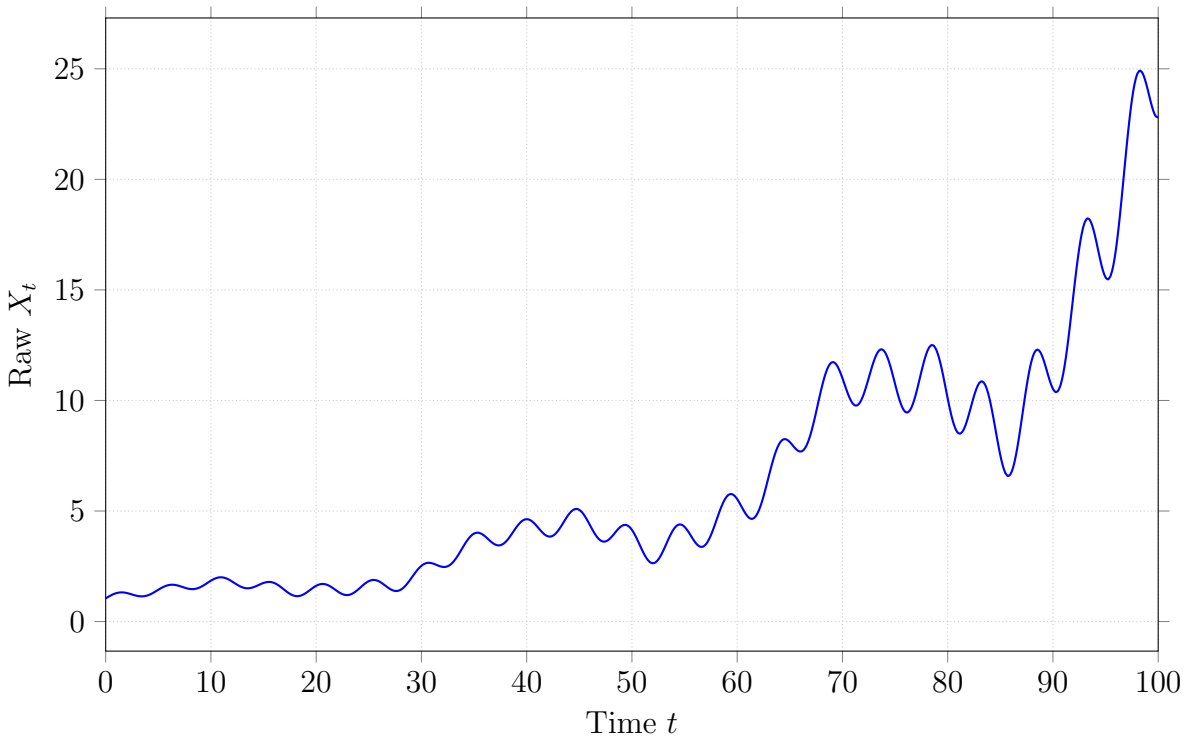


Figure 1.2: Raw synthetic time series with exponential growth and multiplicative oscillations. The variance increases as the series level grows.

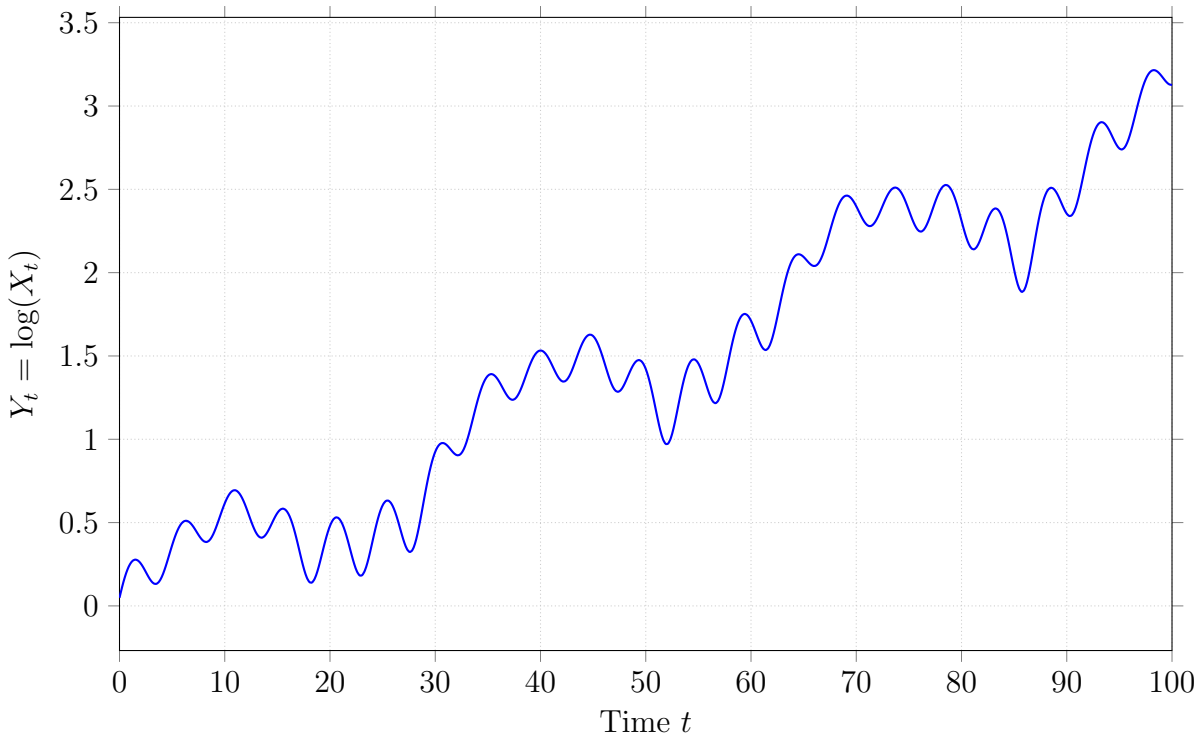


Figure 1.3: Logarithmic transformation reduces variance and converts multiplicative effects into additive effects.

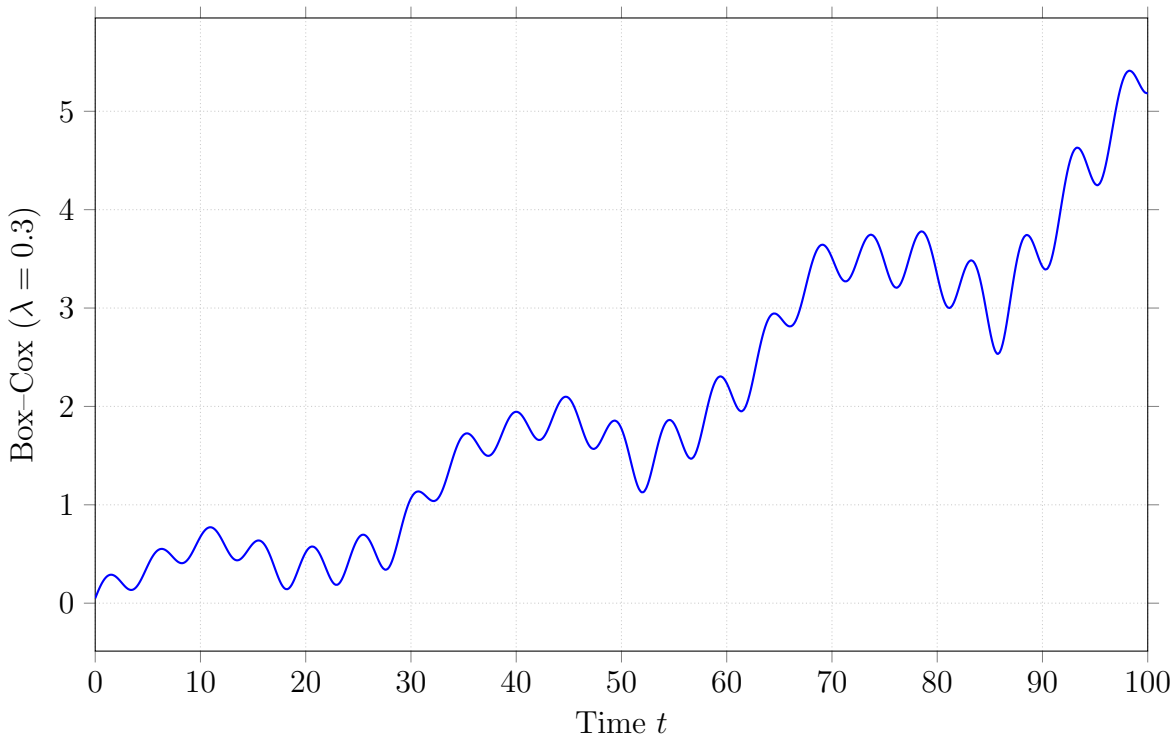


Figure 1.4: Box–Cox transformation with $\lambda = 0.3$ provides flexible variance stabilization, with $\lambda = 0$ recovering the logarithmic case.

This construction represents a process with long-term growth and multiplicative seasonal fluctuations, where the variance increases as the level of the series rises. Figure 1.2 shows the original data. Applying a logarithmic transformation (Figure 1.3) stabilizes the variance and converts multiplicative patterns into additive ones. Finally, the Box–Cox transformation with parameter $\lambda = 0.3$ (Figure 1.4) demonstrates flexible variance stabilization, with $\lambda = 0$ reducing to the logarithmic case. This example highlights how transformations make the data more suitable for subsequent modeling by ARIMA, SARIMA, or deep learning methods.

Summary. Transformations are a crucial pre-processing step in time series analysis. They not only stabilize variance and remove multiplicative effects, but also help to achieve stationarity and normalize the distribution of values. Choosing the right transformation depends on the characteristics of the series and the requirements of the forecasting method. In practice, it is common to experiment with multiple transformations (log, Box–Cox, differencing) and select the one that yields the best model performance.

1.4. ARIMA model

1.4.1. Overview

The model Autoregressive Integrated Moving Average (ARIMA) is a crucial and widely used tool in forecasting time series [2, 10]. The main idea of ARIMA is a combination of 3 fundamental blocks:

- Autoregressive (AR) component represents present value in terms of past values of a variable,

- Integrated (I) component transfer a nonstationary series to a stationary one.
- Moving average (MA) shows the relationship between present value and past white noises.
- p is the degree of AR.
- d is the degree of integrated part.
- q is the degree of MA.

1.4.2. Autoregressive (AR) Component

AR Model

The autoregressive (AR) model assumes that the current observation of a time series can be expressed as a weighted sum of its past values plus a random shock. This captures the intuition that the past influences the present, with the impact of older observations diminishing over time.

Definition 1.4.1. An AR(p) process is given by:

$$X_t = \phi_1 X_{t-1} + \phi_2 X_{t-2} + \cdots + \phi_p X_{t-p} + \varepsilon_t,$$

where ε_t is white noise and $\phi_1, \ldots, \phi_p \in \mathbb{R}$. [2]

For instance, an AR(1) model with $\phi_1 = 0.7$ expresses that 70% of the current value is explained by the previous observation, while the remaining variation comes from random noise. This is a natural way to model time series with short-term persistence, such as daily temperature fluctuations.

Theorem 1.4.2. *For* AR(1)*:*

$$X_t = \phi_1 X_{t-1} + \varepsilon_t,$$

stationarity requires $|\phi_1| < 1$*. Then*

$$E[X_t] = 0, \qquad \mathrm{Var}(X_t) = \frac{\sigma^2}{1 - \phi_1^2}.$$

[6]

This condition ensures that the influence of shocks decays over time, preventing the process from exploding.

Theorem 1.4.3. *For* AR(p)*, the process is stationary if all zeros of*

$$1 - \phi_1 z - \phi_2 z^2 - \cdots - \phi_p z^p = 0$$

lie outside the unit circle. [6]

Parameter estimation can be done with the Yule–Walker equations, OLS regression, or Maximum Likelihood Estimation (MLE). Each method has trade-offs: Yule–Walker is computationally efficient but less accurate in small samples, while MLE is statistically efficient but requires stronger assumptions.

Partial Autocorrelation Function (PACF)

The PACF measures the *direct* correlation between X_t and X_{t-k} after removing the effect of intermediate lags. This makes PACF particularly suitable for identifying AR order, because it tells us how many direct past lags matter for prediction.

Definition 1.4.4. The partial autocorrelation at lag k, denoted ϕ_{kk}, is defined as the correlation between X_t and X_{t-k} after removing effects of $X_{t-1}, \ldots, X_{t-k+1}$. [2]

We list here some properties of PACF:

- PACF values range in $[-1, 1]$.
- For an AR(p) process, PACF *cuts off* sharply after lag p.
- In practice, PACF plots include confidence bands $\pm 1.96/\sqrt{n}$ to judge significance.

Example 1.4.1. If we simulate an AR(2) with coefficients $(0.6, 0.2)$, its PACF plot shows two clear spikes at lags 1 and 2, while all later lags fall within the confidence bounds. This pattern immediately suggests $p = 2$.

Tools to Find Best Order p

Beyond visual inspection of PACF, statistical criteria are widely used:

$$AIC(p) = -2\ln(\hat{L}) + 2p, \quad BIC(p) = -2\ln(\hat{L}) + p\ln(n).$$

Example 1.4.2. Suppose we fit AR(1), AR(2), and AR(3) to a monthly sales dataset. If AR(2) yields the smallest BIC, we conclude that two lags suffice, balancing fit and parsimony. In practice, BIC tends to favor simpler models, while AIC leans toward more complex ones.

1.4.3. Integrated (I) Component

Many real-world time series are not stationary: they contain trends or changing variance. To apply AR or MA models, we transform them by differencing.

Definition 1.4.5. The differencing operator Δ is defined as

$$\Delta X_t = X_t - X_{t-1}, \qquad \Delta^2 X_t = X_t - 2X_{t-1} + X_{t-2}.$$

For example, a stock price series is often a random walk. Its first difference corresponds to daily returns, which are usually stationary and better suited for ARMA modeling.

Tools to Find Best Order d

The appropriate differencing order d is determined by unit root tests:

- **Augmented Dickey–Fuller (ADF)**: tests H_0 of a unit root.
- **KPSS**: opposite null hypothesis (H_0: stationarity).
- **Phillips–Perron (PP)**: robust alternative to ADF.

Example 1.4.3. If ADF p-value = 0.99, we fail to reject a unit root $\rightarrow$ apply first differencing. If KPSS test then accepts stationarity, we conclude $d = 1$ is sufficient.

1.4.4. Moving Average (MA) Component

The MA model explains the series using current and past shocks.

Definition 1.4.6. An MA(q) process is

$$X_t = \mu + \varepsilon_t + \theta_1 \varepsilon_{t-1} + \cdots + \theta_q \varepsilon_{t-q}.$$

[2]

Example 1.4.4. If today's electricity demand is unexpectedly high, the MA term allows yesterday's shock (forecast error) to influence today's value.

In order to measure overall correlation with past values, we introduce the following concept.

Definition 1.4.7. The autocorrelation function (ACF for short) is

$$\rho(k) = \frac{\mathbb{E}[(X_t - \mu)(X_{t-k} - \mu)]}{\sigma^2}.$$

[2]

We list here some properties of ACF:

- For MA(q), ACF cuts off after lag q.
- PACF tails off gradually.

Example 1.4.5. An MA(1) process has ACF with only one significant spike at lag 1. This pattern is distinctive and provides a quick identification rule.

Tools to Find Best Order q

- Visual inspection of ACF: cut-off lag $\to q$.
- AIC, BIC, HQ for model comparison.
- Residual checks: ACF of residuals should resemble white noise.

Example 1.4.6. In modeling airline passenger series, after differencing, the ACF shows significant spikes at lag 12 (annual seasonality). This suggests including seasonal MA terms in addition to non-seasonal ones.

Practical Role in ARIMA Identification

The Box–Jenkins methodology combines these tools:

- PACF $\to$ AR order p.
- Unit root tests $\to$ differencing order d.
- ACF $\to$ MA order q.

In practice, these are only initial guesses. The final model is chosen using information criteria (AIC/BIC), validated with out-of-sample forecasting, and confirmed with residual diagnostics.

1.4.5. Characteristic and applications

- ARIMA is specifically designed for time series data, and is commonly applied to economic, financial, meteorological, and production phenomena.
- The model assumes that the series is stationary (or can be made stationary after differencing).
- The parameters p, d, q are selected based on stationarity tests (ADF, KPSS), ACF/PACF plots, and information criteria (AIC, BIC).
- Once the model is constructed, ARIMA can forecast future values and analyze the structure of historical data.

1.5. SARIMA Model

1.5.1. Overview of Seasonality in Time Series

Many real-world time series exhibit regular seasonal fluctuations in addition to long-term trends [2]. While ARIMA models can handle nonstationary trends through differencing, they cannot directly account for repeating seasonal patterns. This motivates the use of the **Seasonal ARIMA (SARIMA)** model, which extends ARIMA by incorporating seasonal autoregressive, differencing, and moving average components.

1.5.2. Definition of SARIMA

Definition 1.5.1. A seasonal ARIMA model is denoted as

$$\text{SARIMA}(p, d, q) \times (P, D, Q)_s,$$

and is defined by:

$$\Phi_p(L)\,\Phi_P(L^s)\,(1-L)^d\,(1-L^s)^D y_t = \Theta_q(L)\,\Theta_Q(L^s)\,\varepsilon_t + \mu,$$

where:

- (p, d, q) are the non-seasonal ARIMA orders,
- (P, D, Q) are the seasonal ARIMA orders,
- s is the seasonal period (e.g. $s = 12$ for monthly data with annual seasonality),
- $\Phi_p(L)$ and $\Theta_q(L)$ are non-seasonal AR and MA polynomials,
- $\Phi_P(L^s)$ and $\Theta_Q(L^s)$ are seasonal AR and MA polynomials,
- $(1-L)^d$ is the non-seasonal differencing operator,
- $(1-L^s)^D$ is the seasonal differencing operator,
- ε_t is white noise and μ is a constant.

SARIMA reduces to ARIMA when the seasonal orders (P, D, Q) are set to zero.

1.5.3. Identification of Seasonal Orders

The process of identifying SARIMA orders extends the Box–Jenkins methodology:

1. **Visual inspection:** seasonal cycles can often be observed directly in time plots.
2. **Seasonal differencing:** apply $(1 - L^s)^D$ to remove repeating seasonal patterns. For example, with monthly data, $(1 - L^{12})y_t = y_t - y_{t-12}$ removes annual effects.
3. **ACF and PACF:**
   - Seasonal spikes at lags $s, 2s, \ldots$ suggest seasonal dependence.
   - Seasonal AR terms (P) are identified via PACF at multiples of s.
   - Seasonal MA terms (Q) are identified via ACF at multiples of s.
4. **Model selection:** fit candidate models and compare using AIC, BIC, or HQ.

1.5.4. Illustrative Example: Airline Passengers Data

The monthly `AirPassengers` dataset (1949–1960) provides a classic example of seasonality.

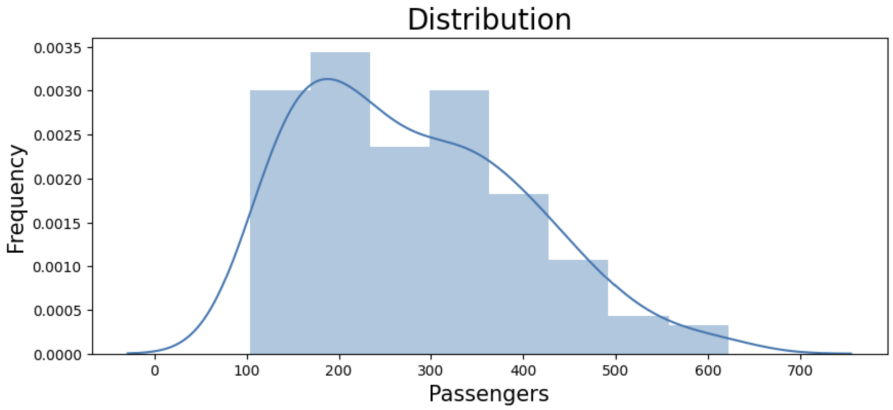


Figure 1.5: Distribution of monthly passenger counts

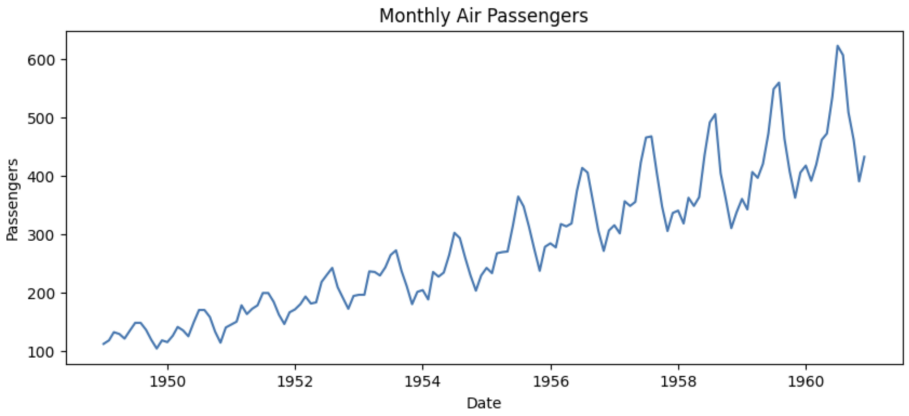


Figure 1.6: Time series of monthly passengers

The graphs of data show both an upward trend and a repeating yearly pattern:

- After applying first differencing ($d = 1$) and seasonal differencing with $s = 12$ ($D = 1$), the series becomes approximately stationary.
- The ACF shows a strong spike at lag 12, suggesting a seasonal MA(1) term.
- The PACF indicates additional short-term autoregressive effects.
- A well-fitting specification is:

$$\text{SARIMA}(0, 1, 1) \times (0, 1, 1)_{12}.$$

This model successfully captures both the short-term autocorrelation and the annual seasonal cycle, producing accurate forecasts.

1.5.5. Applications and Limitations

SARIMA models are widely applied in practice:

- **Economics:** quarterly GDP, monthly inflation, retail sales.
- **Transportation and tourism:** airline passengers, hotel bookings.
- **Energy:** seasonal demand for electricity, gas, or water.

Limitations of SARIMA include:

- Seasonal period s must be known in advance.
- Large parameter sets (high p, q, P, Q) can lead to overfitting and instability.
- SARIMA assumes linear dependence; nonlinear dynamics may require advanced models such as Seasonal GARCH or deep learning methods.

Despite these drawbacks, SARIMA remains one of the most widely used seasonal models due to its interpretability, flexibility, and solid theoretical foundation. Model performance will later be evaluated using metrics such as MAE, RMSE, and MAPE (see Section 1.5).

1.5.6. Comparison: ARIMA vs SARIMA

Table 1.4: Comparison between ARIMA and SARIMA

Aspect	**ARIMA**	**SARIMA**
Seasonality	Cannot directly model	Explicitly includes seasonal terms
Parameters	(p, d, q)	$(p, d, q) \times (P, D, Q)_s$
Use Case	Short-memory, non-seasonal	Monthly, quarterly, or periodic data
Complexity	Fewer parameters	More parameters, risk of overfitting

1.6. Forecasting Model Evaluation Metrics

In time series forecasting problems in general, the selection of appropriate evaluation metrics is crucial [10] to accurately reflect both the predictive performance and the generalization capability of a model. Below are some commonly used evaluation metrics.

1.6.1. Mean Absolute Error (MAE)

Definition: The Mean Absolute Error (MAE) measures the average magnitude of errors in a set of forecasts, without considering their direction. It is formally defined as:

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i| \tag{1.1}$$

where y_t denotes the actual value at time t, $\hat{y}_t$ is the forecasted value, and n is the total number of observations. [10]

Interpretation: MAE indicates, on average, how many units the predictions deviate from the actual values. Since it uses absolute differences, MAE does not distinguish between positive and negative errors, and it is less sensitive to outliers compared to squared-error metrics. This makes it one of the most widely used measures for evaluating the overall accuracy of forecasting models.

1.6.2. Root Mean Squared Error (RMSE)

Definition: The Root Mean Squared Error (RMSE) is the square root of the average squared differences between the actual values and the predicted values. It is defined as:

$$RMSE = \sqrt{\frac{1}{n}\sum_{t=1}^{n}(y_t - \hat{y}_t)^2}.1001[10]$$

Interpretation: RMSE provides a measure of the average magnitude of the prediction errors, but places a higher weight on larger errors due to the squaring process. This makes RMSE more sensitive to outliers compared to MAE. In many forecasting applications, RMSE is considered a standard indicator of predictive accuracy.

1.6.3. Mean Absolute Percentage Error (MAPE)

Definition: The Mean Absolute Percentage Error (MAPE) expresses the prediction accuracy as a percentage, calculated as:

$$MAPE = \frac{100}{n}\sum_{t=1}^{n}\left|\frac{y_t - \hat{y}_t}{y_t}\right|.1001[10]$$

Interpretation: MAPE indicates the average percentage error between forecasts and actual values, making it scale-independent and easy to interpret. However, it has limitations when actual values y_t are close to zero, which can result in extremely large percentage errors.

1.6.4. Coefficient of Determination (R^2)

Definition: The coefficient of determination, denoted R^2, measures the proportion of variance in the dependent variable that is explained by the model. It is defined as:

$$R^2 = 1 - \frac{\sum_{t=1}^{n}(y_t - \hat{y}_t)^2}{\sum_{t=1}^{n}(y_t - \bar{y})^2},$$

where $\bar{y}$ is the mean of the actual values.

Interpretation: R^2 ranges between 0 and 1, where higher values indicate a better fit of the model to the data. An R^2 value close to 1 implies that the model explains most of the variability in the observed data, whereas values near 0 suggest poor explanatory power.

1.6.5. Symmetric Mean Absolute Percentage Error (SMAPE)

SMAPE overcomes the division-by-zero issue in MAPE:

$$SMAPE = \frac{100}{n}\sum_{t=1}^{n}\frac{|y_t - \hat{y}_t|}{(|y_t| + |\hat{y}_t|) : 2}.$$

SMAPE is especially useful in financial forecasting where relative errors matter.

Example 1.6.1. If $y_t = 100, \hat{y}_t = 90$,

$$SMAPE = \frac{10}{95} \times 100 = 10.53\%.$$

Chapter 2

Deep learning and its application in time series forecasting

Deep Learning is a branch of Machine Learning based on Artificial Neural Networks (ANN) with multiple hidden layers [3]. The main idea of Deep Learning is to model complex and non-linear relationships in data by learning features at different levels of abstraction.

While traditional Machine Learning often requires *feature engineering* designed by humans, Deep Learning is capable of automatically learning features from raw data, thereby reducing dependence on domain experts. For example, in a time series problem, instead of manually designing statistical indicators (moving average, volatility, etc.), Deep Learning can directly learn from raw data such as value sequences over time.

2.1. Artificial Neural Networks (ANN)

In this section, we explore the theory and background of neural networks, especially feed-forward neural networks.

2.1.1. Neurons

Definition 2.1.1. An artificial neuron (or neuron for short) is a function $f\colon \mathbb{R}^d \to \mathbb{R}$ of the form

$$f(\mathbf{x}) = \phi(\mathbf{w} \cdot \mathbf{x} + b)$$

where $\mathbf{w} \in \mathbb{R}^d$ is the weighted vector and $b \in \mathbb{R}$ is the bias of f, and $\sigma\colon \mathbb{R} \to \mathbb{R}$ is an activation function. [3]

Here is a schematic representation of an artificial neuron where $\Sigma = \mathbf{w} \cdot \mathbf{x} + b$. This means that the bias b is a attribute of the perceptron.

Several activation functions can be considered.

- The identity function
$$\sigma(x) = x.$$
- The sigmoid function (or logistic)
$$\sigma(x) = \frac{1}{1 + e^{-x}}.$$
- The hyperbolic tangent function (or tanh)
$$\sigma(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}.$$
- The Rectified Linear Unit (ReLU) activation function
$$\sigma(x) = \max\{0, x\}.$$

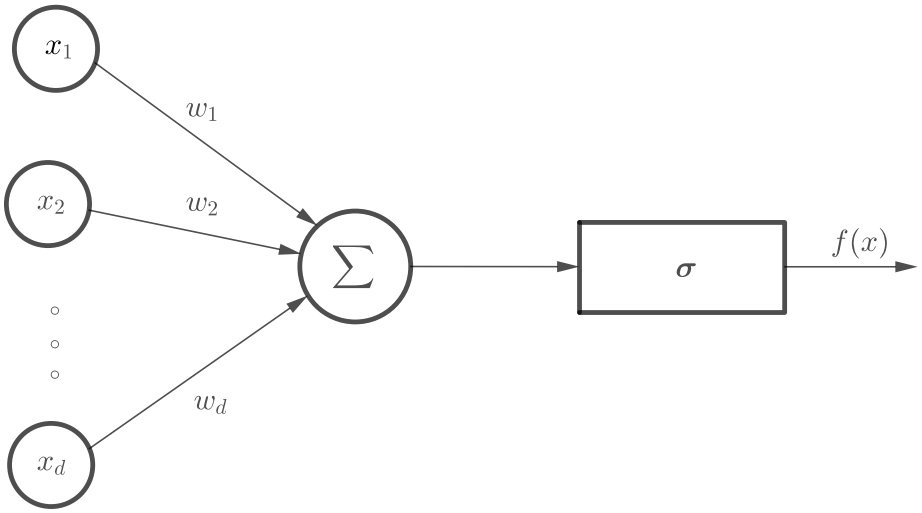


Figure 2.1: Artificial neuron.

2.1.2. Neural Networks

Definition 2.1.2 (Neural Network)**.** A neural network is a structure composed by several layers of neurons where the output of a neuron of a layer becomes the input of a neuron of the next layer. Moreover, the output of a neuron can also be the input of a neuron of the same layer or of neuron of previous layers. The first layer is called the input layer and if the input $\mathbf{x} \in \mathbb{R}^d$ then the layer has d neurons. The final layer is called the output layer and if $\mathbf{y} \in \mathbb{R}^K$ then the layer has K neurons. In between the two aforementioned layers are some number of the hidden layers each with some number of neurons. We define the architecture of the neural network by the number of nodes in each layer. [3]

Definition 2.1.3 (Feed-Forward Neural Network)**.** If a neuron network, every output of a layer is only the input for the next layer, then it is called the a feed-forward neural network. [3]

The Figure 2.2 represents a feed-forward neural network with three input variables, one output variable, and two hidden layers.

The parameters of the architecture are the number of hidden layers and of neurons in each layer. The activation functions are also to choose by the user.

For the output layer the activation function is generally different from the one used on the hidden layers. In the case of regression, we apply no activation function on the output layer.

For binary classification, the output gives a prediction of $P(y = 1 \mid \mathbf{x})$ since this value is in $[0, 1]$, the sigmoid activation function is generally considered.

For multi-class classification, the output layer contains one neuron per class i, giving a prediction of $P(y = i \mid \mathbf{x})$. The sum of all these values has to be equal to 1. The multi-dimensional function *softmax* is generally used:

$$\text{softmax}(\mathbf{z})_i = \frac{\exp(z_i)}{\sum_{j=1}^{K} \exp(z_j)},$$

where K is the number of classes.

In the remaining of the chapter, we only consider feed-forward neural networks. Assume that the network has $L + 1$ layers that index from 0 to L, so that the 0-th layer is the input layer and the L-th layer is the output layer. Let w_{ij}^l denote the weight for the connection from the j-th neuron in the $(l-1)$-th layer to the i-th neuron in the l-th layer.

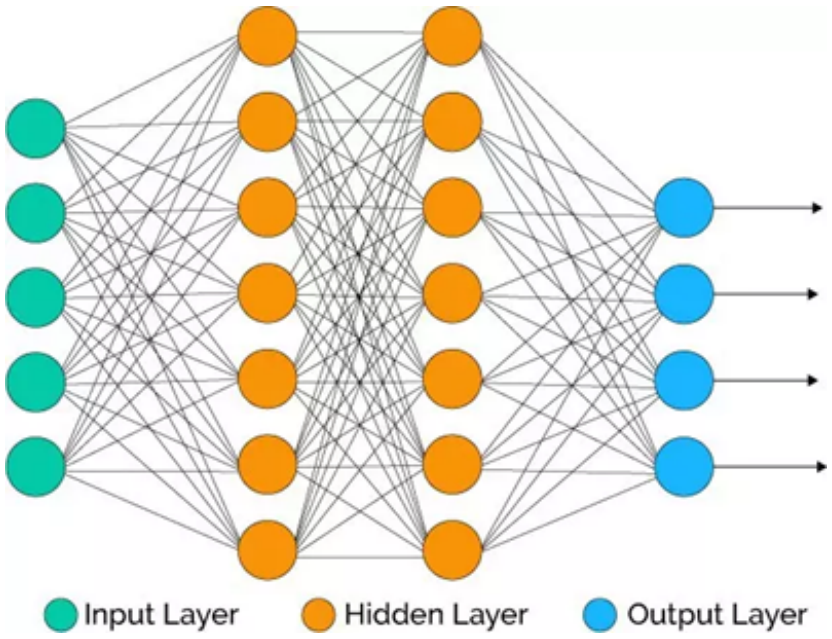


Figure 2.2: Basic neural network.

At the l-layer, we use b_i^l for the bias of the i-th neuron, and a_i^l is the activation of the i-th neuron, i.e. its output. Let σ_l be the activation function of the l-layer. Then, we have

$$a_i^l = \sigma_l \left(\sum_j w_{ij}^l a_j^{l-1} + b_i^l \right) \tag{2.1}$$

where the sum is over all neurons i in the $(l-1)$-th layer.

To rewrite this expression in a matrix form we define a weight matrix W^l for each layer, l. The entries of the weight matrix W^l are just the weights connecting to the l-th layer of neurons, that is, the entry in the i-th row and j-th column is w_{ij}^l. Similarly, for each layer l we define a bias vector, $\mathbf{b}^l$, where the components of the bias vector are just the values b_i^l, one component for each neuron in the l-th layer. And finally, we define an activation vector $\mathbf{a}^l$ whose components are the activations a_i^l.

Denote $\theta = (W^1, \mathbf{b}^1, \ldots, W^L, \mathbf{b}^L)$, which we call the parameters of the network.

Let us summarize the mathematical formulation of a feed-forward neural network with $L+1$ layers.

- We set $\mathbf{a}^0(\mathbf{x}) = \mathbf{x}$.
- For $k = 1, \ldots, L$,

$$\mathbf{z}^k(\mathbf{x}) = \mathbf{b}^k + W^k \mathbf{a}^{k-1}(\mathbf{x}), \tag{2.2}$$

$$\mathbf{a}^k(\mathbf{x}) = \sigma_k(\mathbf{z}^k(\mathbf{x})). \tag{2.3}$$

2.2. Training artificial neural networks

In order to minimize the loss functions for neural networks, a *stochastic gradient descent algorithm* is used. To update the values of the parameters in neural networks the algorithm called

backpropagation is typically used. Backpropagation is an efficient algorithm for computing gradients on neural networks using the chain rule.

2.2.1. Loss function

In order to estimate the parameters θ, we use a training set

$$DTS = \{(\mathbf{x}_i, \mathbf{y}_i) \mid i = 1, \dots, n\},$$

and we minimize the empirical loss

$$L_n(\theta) = \frac{1}{n} \sum_{i=1}^{n} \ell(f(\mathbf{x}_i, \theta), \mathbf{y}_i),$$

and it is associated to a loss function ℓ.

Eventually we add a regularization term, so this leads to minimize the penalized empirical risk

$$L(\theta) = \frac{1}{n} \sum_{i=1}^{n} \ell(f(\mathbf{x}_i, \theta), \mathbf{y}_i) + \lambda \Omega(\theta).$$

We comsider ℓ_p-regularization. Using the same notation as follows:

$$\Omega(\theta) = \sum_k \sum_i \sum_j |w_{ij}^k|^p = \sum_k \|W^k\|_p^p.$$

Note that only the weights are penalized, the biases are not penalized.

We now consider three common types of loss functions:

1. *Binary classification*: In a classification task with two classes, it is standard to use a neural network architecture with a single logistic output unit and the cross-entropy loss function:

$$L(\theta) = -\frac{1}{n} \sum_{i=1}^{n} y_i \log(f(\mathbf{x}_i, \theta) + (1 - y_i)(\log(1 - f(\mathbf{x}_i, \theta)).$$

2. *Multiclass classification*: When a classification task has more than two classes, it is standard to use a softmax output layer. The softmax function provides a way of predicting a discrete probability distribution over the classes. We again use the cross-entropy error function, but it takes a slightly different form. The cross entropy error function for multi-class output is

$$L(\theta) = -\frac{1}{n} \sum_{i=1}^{n} \sum_{j=1}^{K} y_{ij} \log(f(\mathbf{x}_i, \theta)_j).$$

3. *Regression*: We perform regression with the mean squared error loss function

$$L(\theta) = \frac{1}{n} \sum_{i=1}^{n} \|\mathbf{y}_i - f(\mathbf{x}_i, \theta)\|^2 = \frac{1}{n} \sum_{i=1}^{n} \sum_{j=1}^{k} (y_{ij} - f(\mathbf{x}_i, \theta)_j)^2.$$

2.2.2. Backpropagation algorithm

For each data point $(\mathbf{x}_s, y_s)$, for simplicity we denote $R_s = R_s(\theta) = \ell(\mathbf{y}_s, f(\mathbf{x}_s, \theta))$, and hence

$$L_n(\theta) = \frac{1}{n} \sum_{s=1}^{n} R_s.$$

Now we compute $\partial R_s/\partial w_{ij}^k$ and $\partial R_s/\partial b_j^k$. The main idea is first we compute for the output layer, and then by using the chain rule we compute for the previous layer, an so on.

For each k, j, let

$$\mathbf{a}^k = \mathbf{a}^k(\mathbf{x}_s), \mathbf{z}^k = \mathbf{z}^k(\mathbf{x}_s), \text{ and } \delta_j^k = \frac{\partial R_s}{\partial z_j^k}.$$

Observe that for each $k = 1, \ldots, L$, the function R_s is determined by $\mathbf{z}^k$. It follows that for $k = 1, \ldots, L-1$ we have $R_s = R_s(\mathbf{z}^{k+1})$ and

$$\begin{aligned}\delta_j^k = \frac{\partial R_s}{\partial z_j^k} = \frac{\partial R_s(\mathbf{z}^{k+1})}{\partial z_j^k} = \sum_i \frac{\partial R_s}{\partial z_i^{k+1}} \frac{\partial z_i^{k+1}}{\partial z_j^k} = \sum_i \frac{\partial R_s}{\partial z_i^{k+1}} \cdot \frac{\partial}{\partial z_j^k} \left(\sum_t w_{it}^{k+1} \sigma_k(z_t^k) \right) \\ = \sum_i \frac{\partial R_s}{\partial z_i^{k+1}} w_{ij}^{k+1} \sigma_k'(z_t^k) = \sum_i \delta_i^{k+1} w_{ij}^{k+1} \sigma_k'(z_j^k) = \left(\sum_i \delta_i^{k+1} w_{ij}^{k+1} \right) \sigma_k'(z_j^k).\end{aligned}$$

The vector form is

$$\delta^k = ((W^{k+1})^T \delta^{k+1}) \odot \sigma_k'(\mathbf{z}^k). \tag{2.4}$$

By using the vector δ^k, we can compute $\partial R_s/\partial w_{ij}^k$ and $\partial R_s/\partial b_j^k$. Note that

$$\frac{\partial R_s}{\partial w_{ij}^k} = \sum_t \frac{\partial R_s}{\partial z_t^k} \frac{\partial z_t^k}{\partial w_{ij}^k} = \frac{\partial R_s}{\partial z_i^k} \frac{\partial z_i^k}{\partial w_{ij}^k}$$

because because $\partial z_t^k/\partial w_{ij}^k = 0$ if $t \neq i$ by Formula (2.2). Together with Formula (2.3), it yields

$$\frac{\partial R_s}{\partial w_{ij}^k} = \frac{\partial R_s}{\partial z_i^k} \frac{\partial}{\partial w_{ij}^k} \left(\sum_t w_{it}^k a_t^{k-1} + b_i^k \right) = \delta_i^k a_j^{k-1}. \tag{2.5}$$

By the same way,

$$\frac{\partial R_s}{\partial b_i^k} = \frac{\partial R_s}{\partial z_i^k} \frac{\partial}{\partial b_i^k} \left(\sum_t w_{it}^k a_t^{k-1} + b_i^k \right) = \delta_i^k. \tag{2.6}$$

By Formulas (2.4)-(2.6), in order to compute the $\frac{\partial R_s}{\partial w_{ij}^k}$'s and $\frac{\partial R_s}{\partial b_i^k}$'s, it suffices to compute δ_j^L for the output layer.

Here are three common types of output layers:

1. *Binary classification*: in a classification task with two classes, we have

   $$R_s = R_s(\theta) = -y_s \log(a_1^L) - (1 - y_s) \log(1 - a_1^L),$$

   where $a_1^L = \sigma(z_1^L)$ with the sigmoid function σ (see Figure 2.3). Since $\sigma'(u) = \sigma(u)(1 - \sigma(u)$ for $u \in \mathbb{R}$, we have

   $$\frac{\partial R_s}{\partial a_1^L} \frac{\partial a_1^L}{\partial z_1^L} = \left(-\frac{y_s}{a_1^L} - -\frac{y_s}{a_1^L} \right) a_1^L (1 - a_1^L) = a_1^L - y_s,$$

   so that

   $$\delta_1^L = a_L^1 - y_s. \tag{2.7}$$

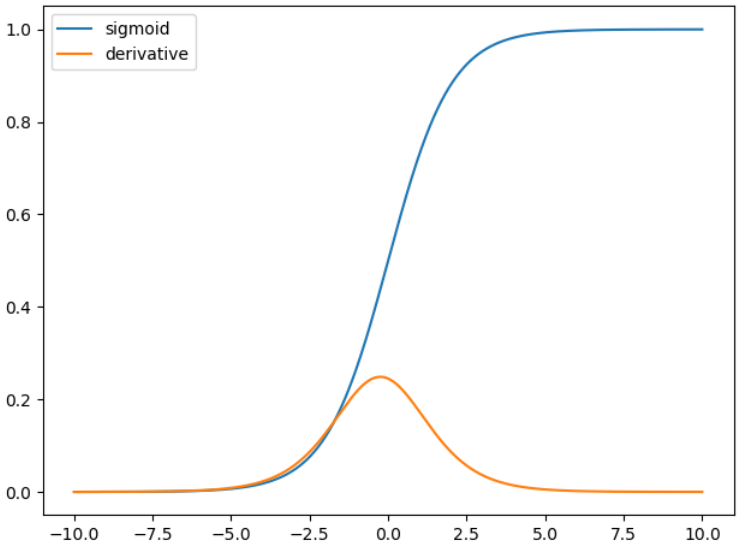


Figure 2.3: Sigmoid function and its derivative .

2. *Multi-class classification*: In this learning task,

$$R_s = R_s(\theta) = -\sum_{i=1}^{K} y_{si} \log(a_i^L) = -\sum_{i=1}^{K} y_{si} \log(\text{softmax}(\mathbf{z}^L)_i).$$

Note that $\sum_i y_{si} = 1$, so

$$\begin{aligned} R_s(\theta) &= -\sum_{i=1}^{K} y_{si} \log(\text{softmax}(\mathbf{z}^L)_i) = -\sum_i y_{si} \log\left(\frac{e^{z_i^L}}{\sum_t e^{z_t^L}}\right) \\ &= -\sum_i y_{si} z_j^L + \left(\sum_i y_{si}\right) \log\left(\sum_t e^{z_t^L}\right) \\ &= -\sum_i y_{si} z_j^L + \log\left(\sum_t e^{z_t^L}\right) \end{aligned}$$

It follows that

$$\begin{aligned} \delta_j^L = \frac{\partial R_s}{\partial z_j^L} &= -y_{sj} + \frac{e^{z_j^L}}{\sum_t e^{z_t^L}} \\ &= -y_{sj} + \text{softmax}(\mathbf{z}^L)_j = -y_{sj} + a_j^L. \end{aligned}$$

Thus,

$$\delta^L = \mathbf{a}^L - \mathbf{y}_s. \tag{2.8}$$

3. *Regression*: In this case we have

$$R_s = \|\mathbf{y}_s - \mathbf{a}^L\|^2 = \sum_{i=1}^{K} (y_{si} - a_i^L)^2.$$

It follows that

$$\delta_j^L = -2(y_{sj} - a_j^L)\frac{\partial a_j^L}{\partial z_j^L} = -2(y_{sj} - a_j^L)\frac{\partial a_j^L}{\partial z_j^L} = -2(y_{sj} - a_j^L)\sigma_L'(z_j^L).$$

Thus,

$$\delta^L = 2(\mathbf{a}^L - \mathbf{y}_s) \odot \sigma_L'(\mathbf{z}^L). \tag{2.9}$$

We can now summarize the backpropagation algorithm.

- **Forward pass**: we fix the value of the current weights

  $$\theta = (W^1, \mathbf{b}^1, \ldots, W^L, \mathbf{b}^L),$$

  and we compute all the intermediate values

  $$\{\mathbf{z}^k(\mathbf{x}_s), \mathbf{a}^k(\mathbf{x}_s) = \sigma_L(\mathbf{z}^k(\mathbf{x}_s)\}_{1 \leqslant k \leqslant L}.$$

  Note that the predicted value $f(\mathbf{x}_s, \theta)$ is $\mathbf{a}^L(\mathbf{x}_s)$.

- **Backpropagation algorithm**:
  - At the output layer L,
    (a) Compute δ^L that depends on the learning task.
    (b) Compute $\partial R_s / \partial w_{ij}^L$ by using Formula (2.5).
    (c) Compute $\partial R_s / \partial b_j^L$ by using Formula (2.6).
  - For $k = L-1, \ldots, 1$,
    (a) Compute δ^k by using Formula (2.4) and the computed vector δ^{k+1}.
    (b) Compute $\partial R_s / \partial w_{ij}^k$ by using Formula (2.5).
    (c) Compute $\partial R_s / \partial b_j^k$ by using Formula (2.6).

2.2.3. Optimization Algorithms (Optimizers)

The goal of optimization algorithms is to find the set of weights and parameters that *minimize the loss function* during the training process of the network. These algorithms determine the convergence speed and the quality of the model.

- **Gradient Descent (GD):** This is the most basic algorithm, based on calculating the first derivative of the loss function with respect to the parameters and updating them in the direction of descent. Weight update rule:

  $$\theta_{t+1} = \theta_t - \eta \cdot \nabla_\theta L(\theta_t)$$

  where η is the learning rate. [13]
  - Advantages: Simple, easy to understand.
  - Disadvantages: Computationally expensive on large datasets, prone to local minima.

- **Stochastic Gradient Descent (SGD):** Instead of using the whole dataset, SGD updates the weights after each sample (or mini-batch).

  $$\theta_{t+1} = \theta_t - \eta \cdot \nabla_\theta L_i(\theta_t)$$

  where L_i is the loss function of a single sample or a mini-batch.
  - Advantages: Much faster than GD, suitable for large datasets.

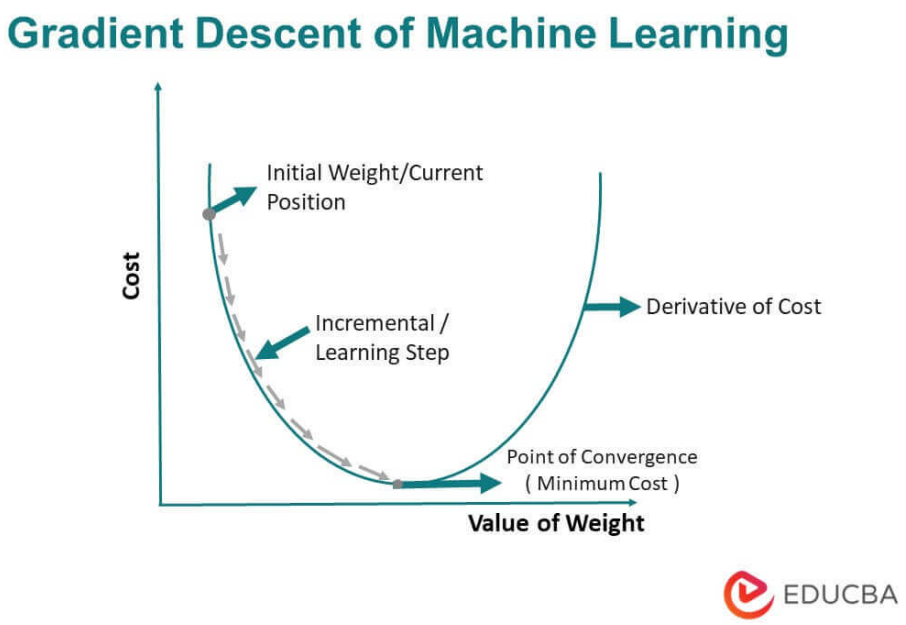


Figure 2.4: Optimization algorithms in Deep Learning

  - Disadvantages: Convergence process oscillates, less stable.

- **Momentum:** Developed to overcome the limitations of SGD. The idea is to add "inertia" to the update process:

$$v_t = \gamma v_{t-1} + \eta \nabla_\theta L(\theta_t)$$

$$\theta_{t+1} = \theta_t - v_t$$

  where γ is the momentum coefficient ($0 < \gamma < 1$).

  - Advantages: Reduces oscillations, helps escape local minima faster.

- **RMSProp (Root Mean Square Propagation):** RMSProp adjusts the learning rate for each parameter by dividing the gradient by the square root of the exponentially decaying average of squared gradients:

$$E[g^2]_t = \beta E[g^2]_{t-1} + (1-\beta) g_t^2$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} g_t$$

  - Advantages: Suitable for non-stationary data, particularly effective in Deep Learning.

- **Adam (Adaptive Moment Estimation):** Adam combines the advantages of Momentum and RMSProp. The algorithm estimates the *first moment* (the moving average of gradients) and the *second moment* (the moving average of squared gradients):

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t$$

$$v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2$$

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t}, \quad \hat{v}_t = \frac{v_t}{1-\beta_2^t}$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon}\hat{m}_t$$

[14]

- Advantages: Fast convergence, effective on large-scale and complex nonlinear data.
- Disadvantages: May converge to suboptimal solutions in some cases.

2.3. Convolutional Neural Networks (CNN) for Time Series

2.3.1. From Spatial to Temporal Feature Learning

While Convolutional Neural Networks (CNNs) achieved groundbreaking success in computer vision by learning spatial hierarchies in images [11], their core principles are equally powerful for sequential data. A CNN applied to time series performs **temporal convolution**, sliding filters across the time dimension to automatically extract relevant features such as trends, seasonal patterns, shapes, and anomalous spikes. This capability to learn salient patterns directly from raw data eliminates the need for extensive manual feature engineering, which is often required by traditional statistical models.

2.3.2. Mathematical Foundation: 1D Convolution

The fundamental operation in a temporal CNN is the one-dimensional convolution.

Definition 2.3.1 (One-Dimensional Discrete Convolution)**.** Given a discrete time series

$$X = (x_1, x_2, \ldots, x_T)$$

and a filter (or kernel) $W = (w_1, w_2, \ldots, w_k)$ of length k, the convolution operation at time t is defined as:

$$s(t) = (X * W)(t) = \sum_{\tau=1}^{k} w_\tau \cdot x_{t-\tau+b}$$

where b is an index shift parameter that depends on the type of padding used. This operation produces a new sequence $S = (s_1, s_2, \ldots)$, known as a feature map. [11]

In practice, multiple filters are used in parallel, each learning to detect a different type of temporal motif. For instance, one filter might activate for sharp increases, while another might respond to periodic dips.

Key Hyperparameters of the Convolutional Layer

The behavior of the convolution is controlled by several critical hyperparameters:

- **Kernel Size (k):** The length of the filter. A small kernel (e.g., $k = 3$) captures short-term, high-frequency patterns, while a large kernel (e.g., $k = 12$ for monthly data) can learn seasonal cycles.
- **Number of Filters (F):** The number of unique kernels in the layer. Each filter produces one feature map. A higher F allows the network to learn a richer set of features but increases computational cost.

- **Stride (s):** The step size with which the kernel slides across the input. A stride of 1 is standard; a larger stride downsamples the output.
- **Padding (p):** Adding zeros to the beginning and end of the input sequence. 'Same' padding preserves the input length, while 'Valid' padding does not add any zeros, resulting in a shorter output.

The length of the output feature map T' can be calculated as:

$$T' = \left\lfloor \frac{T + 2p - k}{s} \right\rfloor + 1$$

2.3.3. Core Architectural Components

A standard CNN architecture for time series forecasting consists of a stack of the following layers:

Convolutional Layers

These are the feature extraction engines. A convolutional layer applies a set of learnable filters to the input sequence. Each neuron in a convolutional layer is connected only to a local region (a time window) of the previous layer's output, a concept known as **local connectivity**. This dramatically reduces the number of parameters compared to a fully connected layer and enforces a translation-invariant learning of patterns.

Activation Functions

After convolution, a non-linear activation function is applied element-wise to the feature map. The **Rectified Linear Unit (ReLU)**, defined as $f(x) = \max(0, x)$, is most common. It introduces non-linearity, allowing the network to learn complex relationships, and is computationally efficient.

Pooling Layers

Pooling layers perform a downsampling operation along the temporal dimension. **Max Pooling** is the most prevalent technique, which takes the maximum value within a sliding window (e.g., of size 2).

Definition 2.3.2 (Max Pooling)**.** For a feature map S and a pooling size p, the max pooling operation is:

$$P_t = \max(s_t, s_{t+1}, \ldots, s_{t+p-1})$$

[11]

The purposes of pooling are twofold:

1. It reduces the computational load for subsequent layers by decreasing the sequence length.
2. It provides a form of translation invariance, making the model more robust to small temporal shifts in the input patterns.

Fully Connected (Dense) Layers

After alternating convolutional and pooling layers, the final feature maps are **flattened** into a single vector. This vector is then fed into one or more fully connected layers. The role of these layers is to perform the high-level reasoning based on all the extracted features and produce the final forecast (e.g., a single value for one-step-ahead prediction).

2.3.4. Advantages for Time Series Forecasting

CNNs offer several distinct advantages over other models for time series analysis:

- **Automatic Feature Learning:** CNNs automatically discover the relevant features from raw time series data, bypassing the need for manual creation of statistical features (e.g., moving averages, volatility).
- **Computational Efficiency:** Due to parameter sharing (the same filter is used across all time steps), CNNs are typically faster to train than RNNs/LSTMs, especially on long sequences.
- **Resistance to Overfitting:** The combination of local connectivity and parameter sharing acts as a strong regularizer, reducing the risk of overfitting compared to fully connected networks.
- **Parallelization:** Unlike RNNs, which process data sequentially, convolutions can be parallelized over the time dimension, leading to faster training on modern hardware (GPUs).

2.3.5. Limitations and the Case for Hybrid Models

The primary limitation of a standard CNN is its inherent **local perspective**. A filter with a receptive field of size k can only look back k time steps. This makes it challenging for CNNs to capture long-term dependencies that span hundreds or thousands of time steps, a task at which RNNs/LSTMs excel.

This limitation motivates the development of **hybrid models** (e.g., CNN-LSTM), where the CNN acts as a feature extractor for the raw series, and the LSTM models the long-term dynamics of the resulting feature maps.

2.3.6. Illustrative Example: Detecting Weekly Patterns in Electricity Demand

Consider the task of forecasting hourly electricity demand. The data exhibits strong **daily** and **weekly** seasonality.

- A CNN can be designed with a first layer containing filters of size $k = 24$ (to capture daily cycles). These filters will learn patterns associated with daytime high demand and nighttime low demand.
- A subsequent layer could use filters of size $k = 7$ (on the downsampled output of the first layer) to learn weekly patterns, such as the difference between weekday and weekend consumption profiles.

- The final dense layers then combine these hierarchically extracted features—the daily shape and the weekly context—to make an accurate forecast for the next hour or day.

This example illustrates how CNNs can build a multi-scale representation of time series data, from short-term patterns to longer-term structures, making them a powerful and flexible tool for modern forecasting.

2.4. Recurrent Neural Network (RNN)

2.4.1. Concept

A Recurrent Neural Network (RNN) is an artificial neural network architecture specifically designed to process sequential data or time series. Unlike a Feedforward Neural Network, RNN has a *recurrent loop* mechanism that allows information from previous steps to be stored and used in subsequent steps. This enables RNN to capture temporal dependencies in data, such as text, speech, or stock price sequences.

2.4.2. Basic Structure

At each time step t, the RNN receives an input X_t and combines it with the hidden state from the previous step h_{t-1} to compute the current hidden state h_t and the output y_t. The formulas are:

$$h_t = f(W_{xh}X_t + W_{hh}h_{t-1} + b_h)$$

$$y_t = g(W_{hy}h_t + b_y)$$

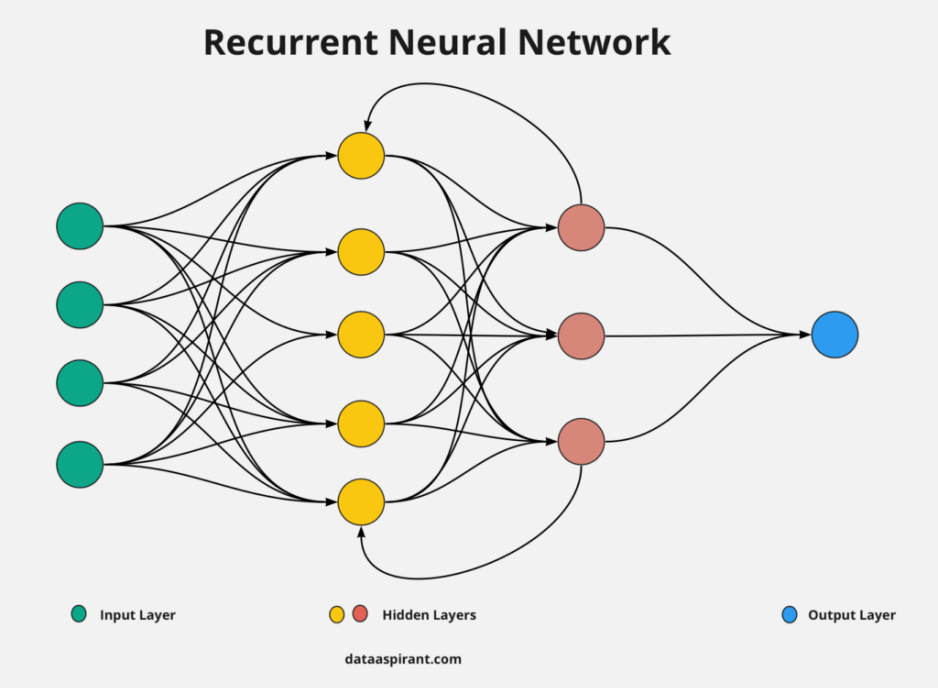


Figure 2.5: Basic structure of RNN

Where:

- X_t: input at time t.
- h_t: hidden state at time t.
- W_{xh}, W_{hh}, W_{hy}: weight matrices.
- f: nonlinear activation function, typically tanh or ReLU.
- g: output function, which can be softmax or sigmoid depending on the task.

2.4.3. Advantages of RNN

- Context memory: RNN can retain information from previous steps, making it suitable for sequential data.
- Flexibility: Effectively applied in many fields such as natural language processing, speech recognition, machine translation, and time series forecasting.

2.4.4. Limitations of RNN

- **Vanishing Gradient:** When training with long sequences, the gradient becomes very small, making it difficult for the model to learn long-term dependencies.
- **Exploding Gradient:** Conversely, gradients can become too large, causing instability during training.
- Limited memory capacity: Typically effective only for short-term dependencies, while time series data such as stock prices require consideration of long-term factors.

2.4.5. Significance in Time Series Forecasting

RNN is a foundational model for processing sequential and time series data. However, due to its limitations in long-term memory, improved variants such as **LSTM (Long Short-Term Memory)** and **GRU (Gated Recurrent Unit)** have been developed. Among them, LSTM is considered a major breakthrough, effectively addressing the vanishing gradient problem and enabling the model to learn long-term dependencies.

2.5. Long Short-Term Memory Network Model (LSTM)

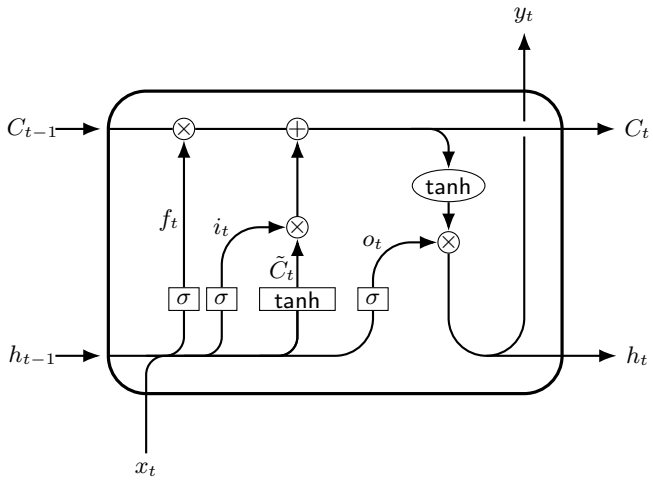
2.5.1. Overview

LSTM is a special variant of recurrent neural networks (RNN), introduced by Hochreiter and Schmidhuber (1997) [12]. Unlike RNN, which only has one mechanism to pass information from one time step to the next, LSTM adds an additional mechanism for information management based on the **cell state** and **gates**. Thanks to this, LSTM can store, maintain, and discard information over a long period of time, allowing the model to learn *long-term dependencies* in sequential data.

Practical significance: LSTM networks are particularly useful in **time series forecasting** tasks (e.g., stock price prediction, energy demand, network traffic), as well as in natural language processing applications (machine translation, semantic analysis).

2.5.2. Operating Principle

An LSTM cell consists of four main components: the **cell state**, **forget gate**, **input gate**, and **output gate**.



- **Cell State**: This is the "long-term memory" of the LSTM, carrying information across many time steps. It allows the network to retain or discard unnecessary information during training. In essence, the cell state helps LSTM maintain the global context of the sequence (e.g., upward/downward trends in a time series).

- **Forget Gate**: Computed by a sigmoid layer, it decides which parts of the previous state should be discarded.

$$f_t = \sigma(W_f \cdot [h_{t-1}, X_t] + b_f)$$

  If f_t is close to 0, the information will be forgotten; if close to 1, the information will be retained. Meaning: The forget gate allows the model to flexibly remove noise or irrelevant information for prediction.

- **Input Gate**: Determines which new information will be added to the cell state. It consists of two steps:

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

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, X_t] + b_C)$$

  Here, i_t is the level of information added, while $\tilde{C}_t$ is the candidate new state. Meaning: The input gate allows the network to update with new knowledge from data, combined with previous knowledge.

- **Cell State Update**: The new state of the cell is calculated as follows:

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

  This step decides which past information to keep and which new information to add.

- **Output Gate**: Determines which part of the cell state will be output as the LSTM's output.

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

$$h_t = o_t * \tanh(C_t)$$

Where h_t is the output at time step t. Meaning: The output gate controls which information is important to produce predictions at the current step.

2.5.3. Key Features of LSTM

- **Ability to Learn Long-Term Dependencies:** Thanks to its gating mechanism, LSTM can remember distant information in sequences without suffering from gradient loss.
- **Addressing the Vanishing Gradient Problem:** The cell state enables gradients to propagate through many time steps without vanishing.
- **Flexibility:** The model can be tuned with hyperparameters such as number of LSTM layers, number of neurons, and learning rate to fit specific applications.
- **Wide Applications:** From time series forecasting, natural language processing, sentiment analysis, to speech recognition.

2.6. Hybrid Models in Time Series Forecasting

2.6.1. Introduction

Time series forecasting involves predicting future values based on historical data, a critical task in domains like finance, energy, transportation, and meteorology. Traditional statistical models, such as ARIMA, excel at capturing linear patterns and are computationally efficient, but they struggle with complex nonlinear relationships and long-term dependencies. Deep learning models, such as Long Short-Term Memory (LSTM) networks and Convolutional Neural Networks (CNNs), are adept at modeling nonlinear patterns and long-term dependencies but require large datasets and significant computational resources. Hybrid models combine the strengths of both approaches to achieve superior forecasting accuracy and robustness, particularly for complex time series with mixed linear and nonlinear components.

This section provides a comprehensive theoretical and practical overview of hybrid models in time series forecasting, focusing on the ARIMA–LSTM and CNN–LSTM hybrids, and exploring other combinations of traditional and deep learning methods. Each model is explained with its components, mathematical foundations, practical considerations, and applications, supported by illustrative diagrams.

2.6.2. ARIMA–LSTM Hybrid Model

The ARIMA–LSTM hybrid model integrates the Autoregressive Integrated Moving Average (ARIMA) model with LSTM networks to capture both linear and nonlinear components of a time series, making it highly effective for complex datasets like air passenger numbers or stock prices.

ARIMA Component

The ARIMA model, introduced by Box and Jenkins (1970), is a cornerstone of statistical time series forecasting. It comprises three components:

- **Autoregressive (AR):** Models the current observation as a linear combination of past observations:
$$X_t = c + \phi_1 X_{t-1} + \phi_2 X_{t-2} + \cdots + \phi_p X_{t-p} + \varepsilon_t \tag{2.10}$$
where ϕ_i are AR coefficients, p is the AR order, and ε_t is white noise.

- **Integrated (I):** Applies differencing to make the time series stationary, removing trends or seasonality. For first-order differencing:
$$X'_t = X_t - X_{t-1} \tag{2.11}$$
where d is the order of differencing.

- **Moving Average (MA):** Models the current observation as a linear combination of past forecast errors:
$$X_t = \mu + \varepsilon_t + \theta_1 \varepsilon_{t-1} + \theta_2 \varepsilon_{t-2} + \cdots + \theta_q \varepsilon_{t-q} \tag{2.12}$$
where θ_i are MA coefficients and q is the MA order.

ARIMA is denoted as $ARIMA(p, d, q)$, where p, d, q are hyperparameters determined through techniques like ACF/PACF analysis or grid search.

LSTM Component

LSTM networks, introduced by Hochreiter and Schmidhuber (1997), are a type of recurrent neural network (RNN) designed to address the vanishing gradient problem in traditional RNNs. LSTMs use a cell state and three gates to manage information flow:

- Forget gate:
$$f_t = \sigma(W_f \cdot [h_{t-1}, X_t] + b_f) \tag{2.13}$$

- Input gate:
$$i_t = \sigma(W_i \cdot [h_{t-1}, X_t] + b_i), \quad \tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, X_t] + b_C) \tag{2.14}$$

- Cell state update:
$$C_t = f_t \cdot C_{t-1} + i_t \cdot \tilde{C}_t \tag{2.15}$$

- Output gate:
$$o_t = \sigma(W_o \cdot [h_{t-1}, X_t] + b_o), \quad h_t = o_t \cdot \tanh(C_t) \tag{2.16}$$

Hybrid Approach

The ARIMA–LSTM hybrid model leverages ARIMA's ability to model linear trends and LSTM's strength in capturing nonlinear residuals. The workflow:

1. Fit ARIMA and generate predictions $\hat{Y}_{t,ARIMA}$ or residuals $e_t = Y_t - \hat{Y}_{t,ARIMA}$.
2. Train LSTM on residuals to produce nonlinear corrections $\hat{Y}_{t,LSTM}$.
3. Final prediction:
$$\hat{Y}_t = \hat{Y}_{t,ARIMA} + \hat{Y}_{t,LSTM} \tag{2.17}$$

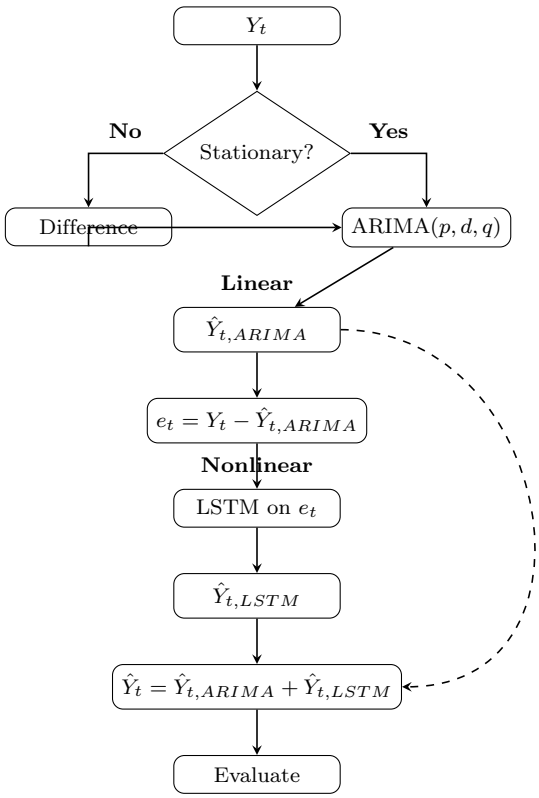


Figure 2.6: ARIMA–LSTM hybrid model workflow

Practical Considerations

- Stationarity: ARIMA requires preprocessing (differencing, detrending, log transforms).
- Hyperparameter tuning: ACF/PACF or auto-ARIMA for (p, d, q); grid/Bayesian search for LSTM.
- Data requirements: Hundreds of observations typically needed for LSTM.
- Computational complexity: More demanding than standalone models.
- Applications: Stock prices, energy demand, air passengers.

Example Application: Air Passenger Forecasting

For the classic Air Passengers dataset (1949–1960):

- ARIMA(1,1,1) captures linear trend/seasonality.
- LSTM (50 units, 2 layers) models residuals.
- Combined forecasts reduce MAE and RMSE.

2.6.3. CNN–LSTM Hybrid Model

The CNN–LSTM hybrid combines CNN feature extraction with LSTM sequence modeling.

CNN Component

A 1D CNN applies filters across time windows:

$$Z_t = \sigma(W \cdot X_{t:t+k-1} + b) \tag{2.18}$$

LSTM Integration

Features extracted by CNN are fed into LSTM, producing the final forecast:

$$\hat{Y}_t = LSTM(CNN(X_t)) \tag{2.19}$$

Practical Considerations

- Filter size: small ($k = 3$) for fine patterns; large ($k = 7$) for broader trends.
- Requires normalization (e.g., Min-Max scaling).

Example: Air Passenger Forecasting

- CNN: 32 filters of size 3 extract local patterns.
- LSTM: 2 layers with 50 units each.
- Outperforms standalone LSTM.

2.6.4. Other Hybrid Models

- SARIMA–LSTM: Seasonal ARIMA + LSTM on residuals.
- GARCH–LSTM: GARCH models volatility, LSTM captures nonlinear patterns.
- Prophet–LSTM: Prophet decomposition + LSTM residual modeling.
- Wavelet–LSTM: Wavelet transform + LSTM for multi-scale signals.
- ETS–LSTM: Exponential smoothing + LSTM for nonlinear residuals.

2.6.5. Practical Implementation

1. Data preprocessing (normalization, train/test split).
2. ARIMA/SARIMA/GARCH modeling with Statsmodels.
3. LSTM/CNN in Keras/TensorFlow.
4. Evaluation: MAE, RMSE, MAPE.
5. Visualization with Matplotlib/Seaborn.

2.6.6. Advantages and Challenges

Advantages

- Combines interpretability of statistics with deep learning flexibility.
- Improved accuracy for mixed linear/nonlinear series.

Challenges

- High computational demands.
- Sensitive to preprocessing and hyperparameters.

2.6.7. Conclusion

Hybrid models (ARIMA–LSTM, CNN–LSTM, SARIMA–LSTM, GARCH–LSTM, Prophet–LSTM, Wavelet–LSTM, ETS–LSTM) offer powerful solutions for complex time series. By leveraging both statistical and deep learning methods, they provide superior accuracy and robustness across domains such as finance, energy, and transportation.

2.7. Application in Number of Air Passengers Forecasting

2.7.1. Programming Language and Libraries Used

This study uses Python with several open-source libraries, including Statsmodels [18], Scikit-learn [19], TensorFlow [20], and Keras [21]. Python is one of the most popular programming languages in the fields of data science and artificial intelligence thanks to its simple, readable syntax, along with a rich ecosystem of libraries. Python provides strong support for data processing, visualization, and the development of machine learning as well as deep learning models.

Some important libraries used in this research include:

- **NumPy**: Provides scientific computing tools, especially operations on arrays and matrices, enabling fast and efficient data processing.
- **Pandas**: A powerful library for tabular data manipulation and analysis (DataFrame), particularly useful for time series data such as financial or transportation data.
- **Matplotlib and Seaborn**: Two popular libraries for data visualization. Matplotlib provides basic plotting capabilities, while Seaborn offers more visually appealing and easy-to-analyze statistical charts.
- **Statsmodels**: A specialized library for statistical analysis and time series modeling, supporting tools such as ARIMA, ADF tests, and autocorrelation analysis.
- **Scikit-learn**: The standard library for machine learning. In this research, it is used for data preprocessing (normalization, train/test split) and evaluation metrics such as MAE and MAPE.
- **TensorFlow/Keras**: A deep learning framework used to build and train deep neural network models. Keras provides a user-friendly interface, making it easy to design architectures such as LSTM networks for time series forecasting.

The combination of Python with the above libraries enables fast, efficient, and scalable research. This is one of the key factors ensuring the accuracy and practical applicability of forecasting models.

2.7.2. Application in Air Passenger Forecasting

Overview of Example

This example applies multiple forecasting methods on the well-known **AirPassengers** dataset, which records monthly airline passenger totals from 1949 to 1960. The dataset is an excellent benchmark because of its strong upward trend and pronounced seasonality. We compare the performance of ARIMA, SARIMA, Long Short-Term Memory (LSTM), and a Hybrid SARIMA–LSTM model, all trained on the historical series and tested on a holdout set.

Data Description

The dataset consists of **144 monthly observations** from January 1949 to December 1960 [2]. It contains two variables:

- **Month:** Observation date (monthly frequency).
- **#Passengers:** Total number of international airline passengers.

Descriptive statistics show that the number of passengers ranges from about **112,000** (early 1949) to nearly **622,000** (mid-1960). The series demonstrates both a clear long-term growth trend and strong seasonal cycles with peaks in summer months.

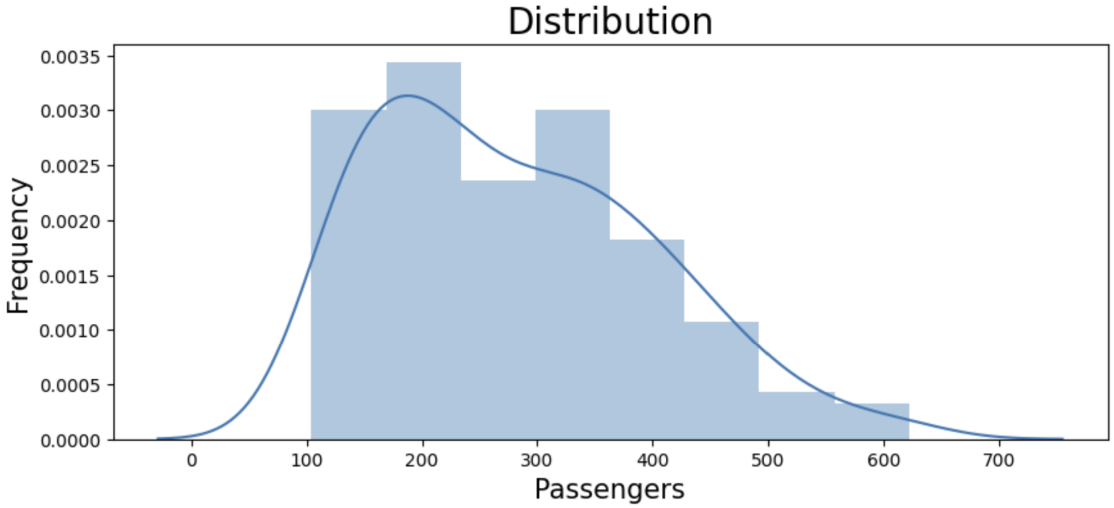


Figure 2.7: Distribution of monthly passenger counts

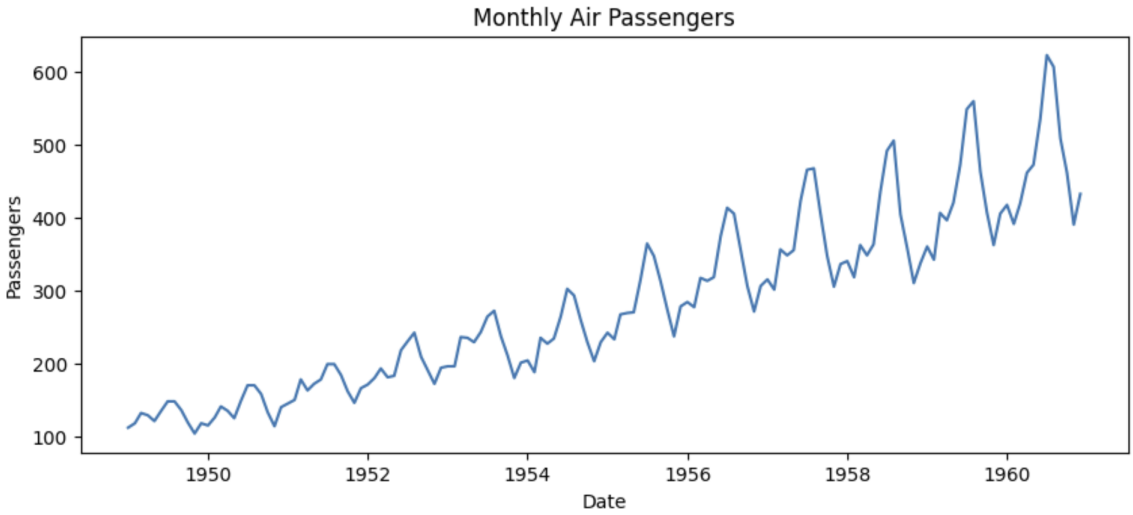


Figure 2.8: Time series of monthly passengers

Stationarity Test

The Augmented Dickey-Fuller (ADF) test was performed to assess stationarity:

- **Before differencing:** ADF Statistic = 0.8154, p-value = 0.9919 ⇒ Fail to reject H_0, series is **non-stationary**.
- **After first differencing:** ADF Statistic = -2.8293, p-value = 0.0542 ⇒ Still borderline non-stationary at the 5% level.

After calculating and plotting graph of Autocorrelation function, seasonal differencing with lag 12 was also applied, leading to a stationary transformed series suitable for ARIMA and SARIMA modeling.

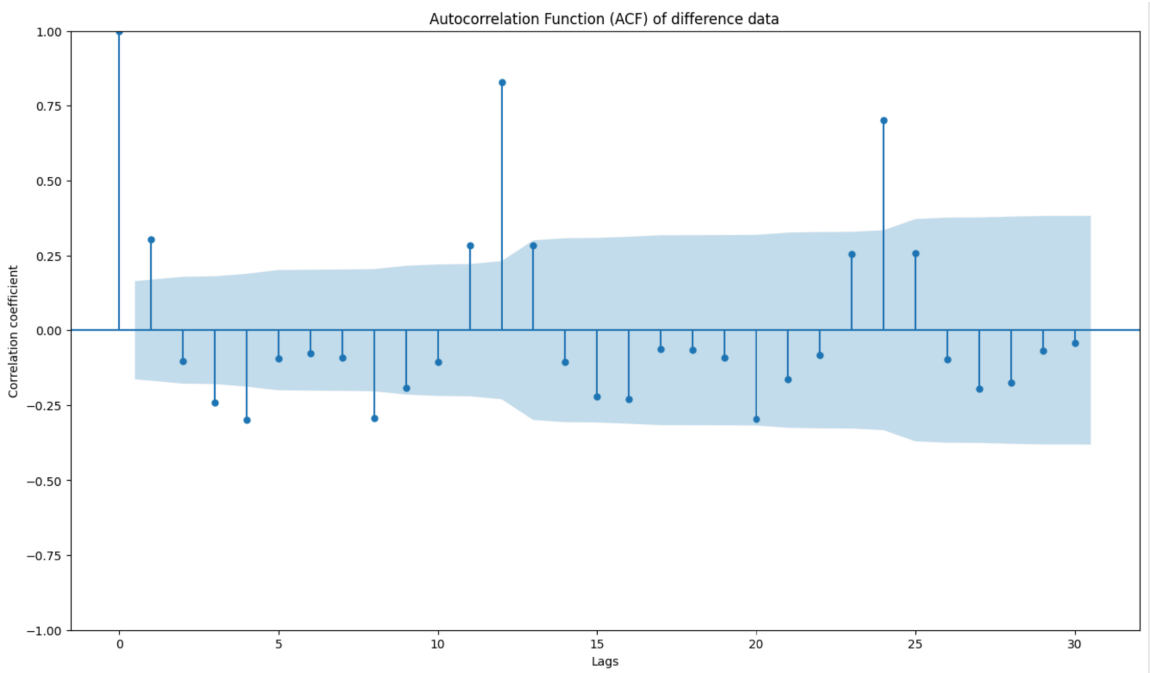


Figure 2.9: ACF plot

ARIMA Model Application

A non-seasonal ARIMA model was fitted with automatic parameter search. The selected model is:

$$\text{ARIMA}(1, 1, 2)$$

with the best AIC = **871.46**.

Forecast accuracy on the 12-month test set:

- MAE = 99.67
- RMSE = 122.10
- MAPE = 21.40%

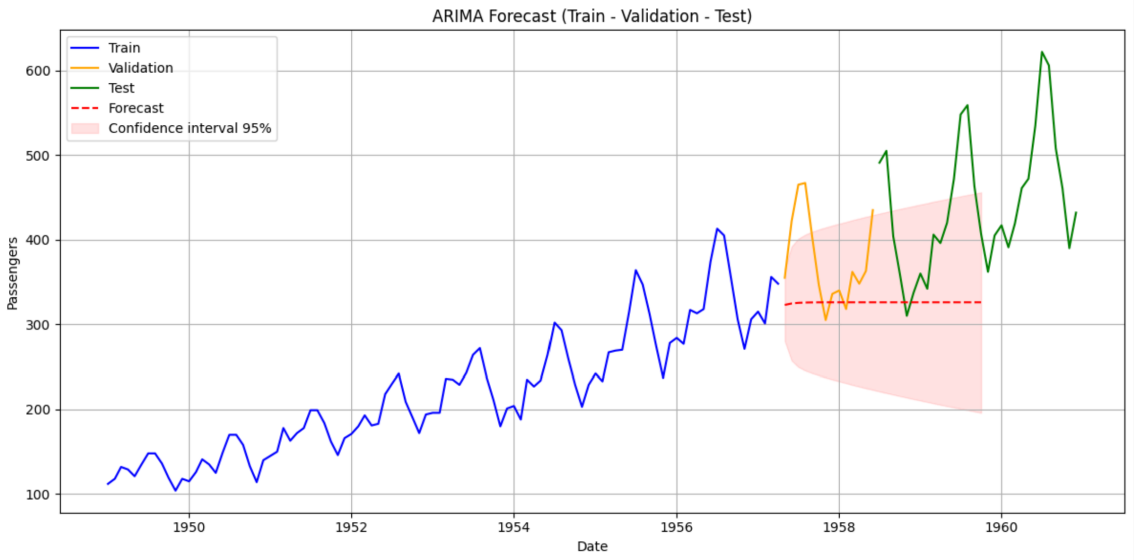


Figure 2.10: 12-month forecast with ARIMA model

SARIMA Model Application

To better capture both the trend and the seasonality of the AirPassengers dataset, a Seasonal ARIMA (SARIMA) model was also applied.

The selected model was:

$$\text{SARIMA}(0, 1, 2) \times (1, 0, 1)_{12}$$

with the lowest AIC value of **605.38**, indicating a strong balance between model complexity and goodness of fit.

Forecast accuracy:

- Validation set: MAE = 18.53, RMSE = 24.66, MAPE = 5.05%.
- Test set: MAE = 20.72, RMSE = 24.76, MAPE = 4.74%.

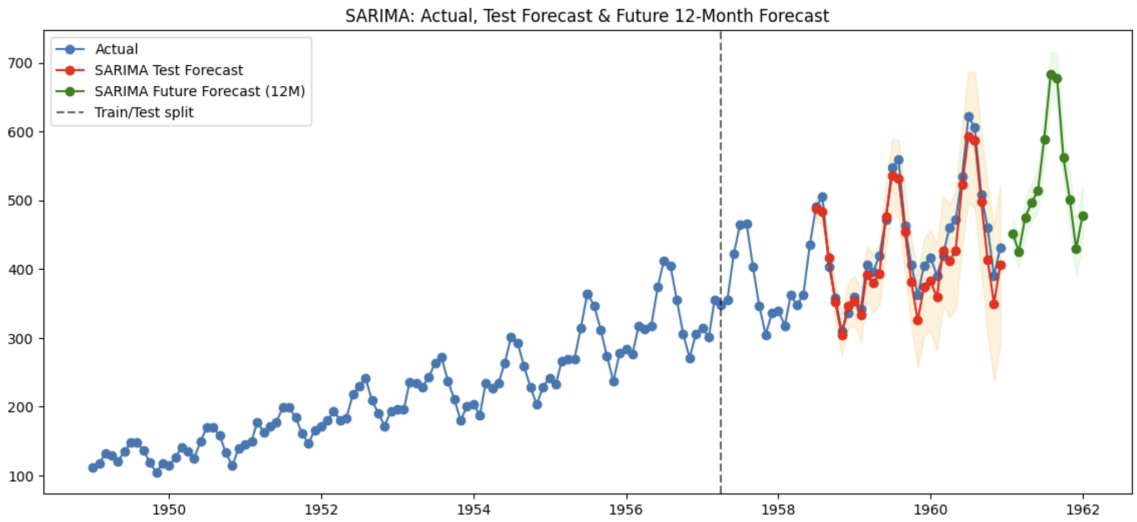


Figure 2.11: 12-month forecast with SARIMA model

Residual diagnostics confirm that the SARIMA model adequately captures the time series dynamics: the Ljung–Box test ($p = 0.96$) shows no significant autocorrelation in the residuals, and the Jarque–Bera test ($p = 0.83$) indicates approximate normality.

LSTM Model Application

To capture nonlinear and long-term dependencies, an LSTM model was built with the following architecture:

- **LSTM layer:** 64 units
- **Dense output layer:** 1 unit

This network has approximately 16,961 parameters.

Evaluation on the test set:

- MAE = 37.99
- RMSE = 47.16
- MAPE = 8.36%

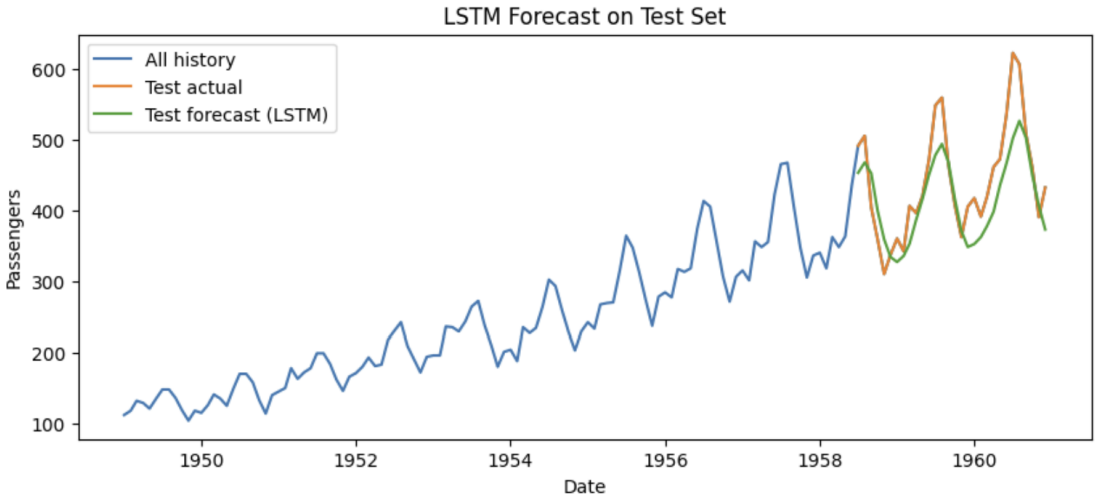


Figure 2.12: 12-month forecast with LSTM model

Hybrid SARIMA–LSTM Model Application

To leverage the strengths of both statistical and deep learning models, a hybrid SARIMA–LSTM model was implemented based on the methodology of Zhang (2003) [17]. The procedure follows a **two-stage approach**:

1. The SARIMA model $(0, 1, 2) \times (1, 0, 1)_{12}$ was first fitted to the original passenger time series. This component captures the linear trend and seasonal structure.

2. The residuals from the SARIMA model, representing nonlinear components not explained by the seasonal linear structure, were then used to train an LSTM network. The LSTM was trained on residual sequences with input shape $(86, 12, 1)$ and output shape $(86, 1)$.

Dataset split:

- Training set: 100 months
- Validation set: 14 months
- Test set: 30 months

Forecast accuracy on the test set:

- MAE = 72.38
- RMSE = 85.81
- MAPE = 16.15%

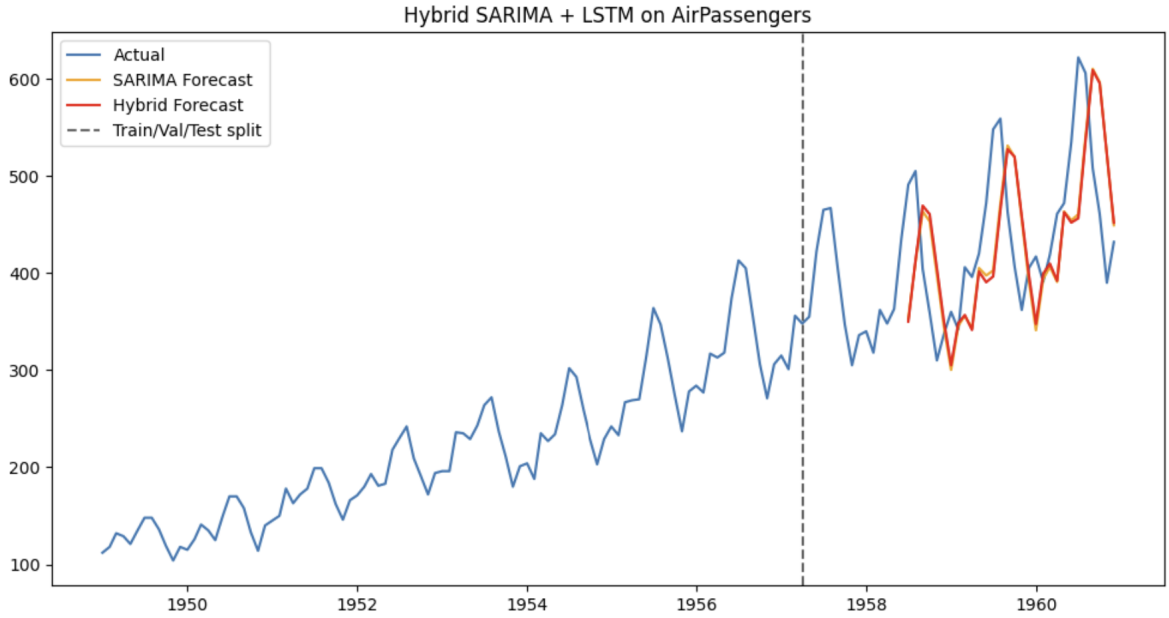


Figure 2.13: Forecasting with Hybrid SARIMA–LSTM model

The hybrid SARIMA-LSTM model underperformed compared to the individual SARIMA and LSTM models. This is likely due to the model's increased complexity leading to overfitting on the relatively small dataset of 144 observations. Since SARIMA already captures the strong seasonal-linear pattern effectively, the residual noise contains limited nonlinear structure for the LSTM to learn.

Overall Comparison of Models

The performance of the four forecasting approaches — ARIMA, SARIMA, LSTM, and Hybrid SARIMA–LSTM — on the AirPassengers dataset is summarized in Table 2.1.

Table 2.1: Comparison of ARIMA, SARIMA, LSTM, and Hybrid SARIMA–LSTM models

Model	**MAE**	**RMSE**	**MAPE (%)**
ARIMA	99.67	122.10	21.40
SARIMA	**20.72**	**24.76**	**4.74**
LSTM	37.99	47.16	8.36
Hybrid (SARIMA–LSTM)	72.38	85.81	16.15

Analytical Discussion of Results

From the empirical experiments, a clear hierarchy of forecasting performance emerges among the four models for this specific dataset.

The simple ARIMA(1,1,2) model, although capable of capturing basic linear autocorrelation, fails to represent the strong seasonal fluctuations in the AirPassengers dataset. Its relatively high MAPE of 21.40% confirms that ignoring seasonality severely deteriorates forecast accuracy.

By contrast, the SARIMA$(0, 1, 2) \times (1, 0, 1)_{12}$ model substantially improves accuracy, reducing the MAPE to below 5%. This improvement highlights the advantage of explicitly modeling seasonal

autoregressive and moving average components for data with clear calendar cycles. Residual diagnostics further reinforce this conclusion: the Ljung–Box test ($p = 0.96$) suggests the residuals are uncorrelated.

The LSTM model, trained using a 12-step sliding window, offers an alternative perspective. It successfully captures both short-term temporal patterns and nonlinear growth without explicit seasonal differencing. Its test performance (MAPE = 8.36%) is better than ARIMA but inferior to SARIMA. This suggests that while LSTM excels in representing nonlinearities, it benefits less from small datasets such as AirPassengers (144 points), where the sample size limits the generalization capability of Deep Learning models.

The hybrid SARIMA–LSTM model integrates these two paradigms. However, in this case, the hybrid model does not outperform its components (MAPE = 16.15%). The most plausible explanation is that the SARIMA model already explains most of the variance (over 95%), leaving residuals that are essentially random noise. Consequently, the LSTM component tends to overfit this noise rather than uncovering meaningful hidden nonlinear dynamics.

This result underlines an important methodological conclusion: Hybrid models are not universally superior. Their effectiveness is most pronounced when the time series exhibits complex nonlinear patterns that standard statistical models fail to capture. For small, strongly seasonal, and linear datasets like AirPassengers, a well-tuned SARIMA model remains the most robust and efficient choice.

REFERENCES

[1] Chatfield, C. (2000). *Time-Series Forecasting*. Chapman & Hall/CRC.

[2] Box, G.E.P., Jenkins, G.M., Reinsel, G.C., and Ljung, G.M. (2015). *Time Series Analysis: Forecasting and Control*. John Wiley & Sons.

[3] Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press.

[4] Mallat, S. (1989). A theory for multiresolution signal decomposition: the wavelet representation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 11(7), 674–693.

[5] Daubechies, I. (1992). *Ten Lectures on Wavelets*. SIAM.

[6] Hamilton, J.D. (1994). *Time Series Analysis*. Princeton University Press.

[7] Dickey, D.A., and Fuller, W.A. (1979). Distribution of the estimators for autoregressive time series with a unit root. *Journal of the American Statistical Association*, 74(366), 427–431.

[8] Kwiatkowski, D., Phillips, P.C.B., Schmidt, P., and Shin, Y. (1992). Testing the null hypothesis of stationarity against the alternative of a unit root. *Journal of Econometrics*, 54(1–3), 159–178.

[9] Box, G.E.P., and Cox, D.R. (1964). An analysis of transformations. *Journal of the Royal Statistical Society: Series B (Methodological)*, 26(2), 211–243.

[10] Hyndman, R.J., and Athanasopoulos, G. (2021). *Forecasting: Principles and Practice* (3rd ed.). OTexts.

[11] LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11), 2278–2324.

[12] Hochreiter, S., and Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, 9(8), 1735–1780.

[13] Robbins, H., and Monro, S. (1951). A stochastic approximation method. *The Annals of Mathematical Statistics*, 22(3), 400–407.

[14] Kingma, D.P., and Ba, J. (2015). Adam: A method for stochastic optimization. *International Conference on Learning Representations (ICLR)*.

[15] Rumelhart, D.E., Hinton, G.E., and Williams, R.J. (1986). Learning representations by back-propagating errors. *Nature*, 323(6088), 533–536.

[16] Brownlee, J. (2017). *Deep Learning for Time Series Forecasting: Predict the Future with MLPs, CNNs and LSTMs in Python*. Machine Learning Mastery.

[17] Zhang, G.P. (2003). Time series forecasting using a hybrid ARIMA and neural network model. *Neurocomputing*, 50, 159–175.

[18] Seabold, S., and Perktold, J. (2010). Statsmodels: Econometric and statistical modeling with Python. *Proceedings of the 9th Python in Science Conference*, 57–61.

[19] Pedregosa, F., et al. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12, 2825–2830.

[20] Abadi, M., et al. (2016). TensorFlow: Large-scale machine learning on heterogeneous systems. *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*.

[21] Chollet, F. (2015). Keras: Deep learning library for Theano and TensorFlow. *GitHub repository*, https://github.com/fchollet/keras.