الجمهورية الجزائرية الديمقراطية الشعبية

Ministry of Higher Education and Scientific Research

UNIVERSITY OF SAAD DAHLAB BLIDA
Faculty of Sciences

Department of Mathematics

MASTER's THESIS
In mathematics

Option: Statistical and Stochastic Modeling

Time series prediction with a combined GARCH (Generalized Autoregressive Conditional Heteroskedasticity) and ANN (Artificial Neural Network) Model

---

Realised by

MEROUCHE Abdelkader LANDJAS Fatiha

Jury Members **:**

President :	Mr. REDOUNE BOUDJEMAA	M.C.A	U.S.D.BLIDA 1
Examiner :	Mr. MOHAMED BOUKHARI	M.A.A	U.S.D.BLIDA 1
Supervisor :	Mr. REDOUANE FRIHI	M.C.B	U.S.D.BLIDA 1

4 July 2024

Dedicace

I dedicate this thesis to...

My Almighty Allah, the Most Gracious, who guided me to the right path, enlightened my way, and helped me overcome all the hardships I encountered throughout this study.

To my dear parents, who always picked me up at the right time and encouraged me to embark on every adventure, especially this one. To my sister, brother, and aunts who were with me throughout my academic journey, guiding me with the light of hope and support, encouraging me to keep moving forward.

To all the people who gave me the strength to continue.

FATIHA LANDJAS

Dedicace

I dedicate this thesis to...

My Almighty Allah, the Most Gracious, who guided me to the right path, enlightened my way, and helped me overcome all the hardships I encountered throughout this study.

To my dear parents, who always picked me up at the right time and encouraged me to embark on every adventure, especially this one. To my sisters, brothers, and their children who were with me throughout my academic journey, guiding me with the light of hope and support, encouraging me to keep moving forward.

To all the people who gave me the strength to continue.

ABDELKADER MEROUCHE

Acknowledgements

First and foremost, all praise is due to Allah, the Almighty, who granted us the ability to stand here with our dissertation.

We would like to thank Mr. REDOUANE FRIHI for his guidance, assistance, patience, and continuous support, which have played a crucial role in shaping this research.

We are also grateful to Mr. REDOUNE BOUDJEMAA and Mr. MOHAMED BOUKHARI for the honor of accepting to preside over and review this work.

Finally, we thank all the people who contributed in any way to the development of this work.

ملخص

نقدم في هذه الأطروحة دراسة حول نماذج التنبؤ بالتقلبات المالية باستخدام نماذج $GARCH$ والشبكات العصبية الاصطناعية (ANN) والتهجين بينهما. ثم طبقناها على بيانات مالية حقيقية لتقييم أدائها.

تهدف الدراسة إلى تحليل أداء نموذج $GARCH(1,1)$ وتقييم دقته التنبؤية مقارنة بالنماذج الأخرى. وأخيرًا، نقدم نموذجًا هجينًا يجمع بين مكونات متعددة من النماذج السابقة بهدف تحسين الأداء التنبؤي بشكل أكبر.

تساهم هذه الدراسة في تقديم فهم أعمق لكيفية تحسين دقة التنبؤ بالتقلبات المالية من خلال استخدام النماذج المتقدمة والمختلطة.

الكلمات المفتاحية : $GARCH$; الشبكات العصبية الاصطناعية (ANN) ; التهجين بين النتائج ; التنبؤ بالتقلبات المالية ; دقة التنبؤ ; نموذج $GARCH(1,1)$; $ANN\ Linear$; $ANN\ Tanh$; النماذج الهجينة ; النماذج المالية المتقدمة.

Abstract

GARCH models, artificial neural networks (ANN), and their hybridization.Then we applied it to real financial data to evaluate its performance.
The study aims to analyze the performance of the $GARCH(1,1)$ model and assess its predictive accuracy compared to other models.
Finally, we propose a hybrid model that combines multiple components of the previous models to further enhance predictive performance.
This study contributes to a deeper understanding of how to improve the accuracy of financial volatility forecasts by using advanced and hybrid models.

Keywords: GARCH; Artificial Neural Networks (ANN); Model Hybridization; Financial Volatility Forecasting; Predictive Accuracy; $GARCH(1,1)$ Model; ANN Linear; ANN Tanh; Hybrid Models; Advanced Financial Models.

Résumé

Dans cette mémoire, nous présentons une étude sur les modèles de prévision de la volatilité financière en utilisant les modèles GARCH, les réseaux de neurones artificiels (ANN) et leur hybridation.Ensuite, nous l'avons appliqué à des données financières réelles pour évaluer ses performances.
L'étude vise à analyser la performance du modèle $GARCH(1,1)$ et à évaluer sa précision prédictive par rapport aux autres modèles.
Enfin, nous proposons un modèle hybride qui combine plusieurs composants des modèles précédents afin d'améliorer davantage la performance prédictive.
Cette étude contribue à une compréhension plus approfondie de la manière d'améliorer la précision des prévisions de volatilité financière en utilisant des modèles avancés et hybrides.

Mots clés: GARCH; Réseaux de Neurones Artificiels (ANN); Hybridation des Modèles; Prévision de la Volatilité Financière: Précision Prédictive; Modèle GARCH(1,1); ANN Linéaire; ANN Tanh; Modèles Hybrides; Modèles Financiers Avancés.

Contents

List of Figures

List of Tables

Abbreviations

GARCH	Gneralized Autoregressive Conditional Heteroskedasticity.
ARCH	Autoregressive Conditional Heteroskedasticity.
EGARCH	Exponential Generalized Autoregressive Conditional Heteroskedasticity.
TGARCH	Threshold GARCH.
GJR-GARCH	Glosten-Jagannathan-Runkle Generalized Autoregressive Conditional Heteroskedasticity.
AIC	Akaike Information Criterion.
BIC	Schwarz (Bayesian) Information Criterion.
MAE	Mean Absolute Error.
MSE	Mean Squared Error.
RMSE	Root Mean Squared Error.
MAPE	Mean Absolute Percentage Error.
VaR	Value-at-Risk.
ANN	Artificial neural network.
RBF	Radial Basis Function.
RNN	Recurrent Neural Network.
MLP	Multilayer Perceptron.

General Introduction

Time series are sequences of data collected or recorded at successive time points at regular intervals. These data represent values of a specific variable over time, and the time intervals can vary, such as hours, days, weeks, months, or years.
Time series are characterized by the presence of temporal dependency between successive values, making their analysis different from that of non-time-dependent data.

Time series analysis involves several key concepts. A trend refers to a long-term increase or decrease in the data, indicating the overall direction of the series over a period of time. Seasonality encompasses regular, repeating fluctuations in the data that occur at specific intervals, such as daily, monthly, or yearly cycles. Cyclical patterns are fluctuations that happen at irregular intervals, often influenced by economic conditions and other external factors.
Lastly, noise represents random variations that do not follow any discernible pattern, adding a layer of unpredictability to the data.

Time series analysis is vital for forecasting in various fields. It helps predict stock prices, interest rates, and economic indicators in economics and finance.
In business and marketing, it aids in sales forecasting, demand planning, and inventory management. It is used to predict weather conditions and climate change patterns, track and predict disease outbreaks, and monitor patients in healthcare.
The energy sector utilizes it for forecasting electricity demand and optimizing power generation, while in manufacturing, it supports predictive maintenance and production schedule optimization.

Time series analysis offers several advantages. It utilizes historical data to predict future trends, enhancing the reliability of forecasts.
By identifying underlying patterns in the data, it helps in understanding the behavior of the system being studied.
Additionally, it detects unusual patterns or outliers, which may indicate significant changes or events. These capabilities make time series analysis a powerful tool for various applications.

Time series analysis utilizes several techniques for effective data analysis and forecasting. Moving averages and exponential smoothing smooth data to identify trends, with the latter giving more weight to recent observations.
The ARIMA model combines autoregression, differencing, and moving averages. Seasonal Decomposition of Time Series (STL) breaks down data into trend, seasonal, and residual components. Fourier Transform analyzes frequency components, while machine learning models like neural networks and RNNs handle complex forecasting.

These techniques collectively enhance the understanding and prediction of time series data.

Time series analysis has diverse applications across various fields. In economics, it aids in understanding economic cycles and making policy decisions based on economic forecasts. In finance, investment strategies and risk management rely heavily on time series forecasts of stock prices and market trends.
In supply chain management, it drives efficient inventory management and logistics planning through accurate demand forecasting.
In healthcare, time series analysis supports predictive analytics in patient care, resource allocation, and managing public health crises.
Additionally, in energy management, it optimizes the use of renewable energy sources and manages the supply-demand balance in power grids.

In Chapter, one of our study on time series analysis, we delve into the application of linear models, specifically the Generalized Autoregressive Conditional Heteroskedasticity (GARCH) model.
This model is adept at capturing volatility patterns and persistence within financial and economic data, providing valuable insights into market dynamics.
In Chapter Two, we transition to exploring the capabilities of Artificial Neural Network (ANN) models. These nonlinear models offer flexibility in capturing intricate relationships within time series data, making them particularly useful for forecasting in complex and dynamic environments.
Moving to Chapter Three, we introduce hybrid models that combine the strengths of both GARCH and ANN models. These hybrid approaches leverage the predictive power of neural networks while incorporating the volatility dynamics captured by GARCH, resulting in enhanced forecasting accuracy.
Finally, in Chapter Four, we apply these models to real financial data. By utilizing the GARCH model, ANN model, and hybrid model, we conduct comprehensive forecasting analyses, providing actionable insights and predictive capabilities for decision-making in financial markets. Through this structured approach, our study aims to showcase the efficacy of these modeling techniques and their practical applications in real-world scenarios.

In concluding our study, we summarize our insights on time series analysis, encompassing the application of linear models such as GARCH, the adaptability of ANN models, and the effectiveness of hybrid approaches.

Chapter 1

GARCH Models

1.1 Introduction

In the world of financial markets, understanding how prices and returns change is crucial for investors and financial analysts.
Fluctuations mean the movement and changes in financial values like stocks and currencies, which are very important for assessing investment risks and identifying profit opportunities.
Therefore, being able to predict and understand these fluctuations is essential for successful investment strategies.

In this context, the ARCH (Autoregressive Conditional Heteroskedasticity) model and the GARCH (Generalized Autoregressive Conditional Heteroskedasticity) model emerge as important tools for analyzing financial market fluctuations.
The main goal of these models is to estimate and forecast price or return changes using past data. Providing accurate estimates of fluctuation levels is necessary for making sound investment decisions and achieving financial goals (see,e.g., [1, 7]).

It's important to note that while the ARCH model focuses on estimating fluctuations, the GARCH model adds the ability to analyze different effects that may occur due to price changes. These different effects mean that fluctuations may respond differently to unexpected losses compared to unexpected gains.

Through this analysis, it becomes clear that the GARCH model plays an important role in understanding the behavior of financial markets.
Consequently, it helps in achieving investment goals and effectively managing risks.
Introducing the GARCH model highlights the importance of understanding financial fluctuations and using appropriate models to analyze and predict them (see e.g., [1, 5, 6, 7, 8]).

1.2 Autoregressive Conditional Heteroskedasticity (ARCH) Model

The basic idea behind ARCH is to capture the time-varying volatility or heteroskedasticity in financial returns.
In financial markets, volatility tends to cluster, meaning periods of high volatility are followed by more periods of high volatility and vice versa. ARCH models provide a framework to model

this phenomenon (see, e.g, [1]).

Definition 1 *The ARCH(p) model allows for generating episodes of high volatility followed by episodes of lower volatility.*

$$Y_t = \mu + \varepsilon_t$$

Where

$$\varepsilon_t / I_{t-1} \sim N(0, h_t)$$

and $I_t = \sigma(X_{t-s})_{s \leq t}$ *denotes the sigma-algebra generated by the* X_{t-s} *in ARCH modeling.*

$$\varepsilon = \eta_t h_t$$

$$h_t = \sqrt{\omega + \sum_{i=1}^{p} \alpha_i \varepsilon_{t-i}^2}$$

and η_t *an independent white noise.*

$$\eta_t \sim N(0, 1)$$

α_i *the autoregressive coefficients.*

1.2.1 ARCH(1)

Definition 2 *When* $p = 1$ *the ARCH(1) model is defined according to the following formula:*

$$\varepsilon_t = \eta_t h_t \tag{1.2.1}$$

where the conditional variance $h_t^2 = E[\varepsilon_t^2 | F_{t-1}]$ *is satisfied for all* $t \in \mathbb{R}^+$ *et* $\omega > 0, \alpha_1 \geq 0$, *constants given:*

$$h_t = \sqrt{\omega + \alpha_1 \varepsilon_{t-1}^2}$$

where $(\eta_t)_t$ *is white noise: a sequence of random variables i.i.d, centered and standardized. It is often assumed that the variables* η_t *are independent of the filtration* F_{t-1} *and that* h_t *depends on* ε_t.

Property : The ARCH(1) model has the following properties:

- $Var(\varepsilon_t | F_{t-1}) = \omega + \alpha_1 \varepsilon_{t-1}^2 = h_t^2$
- $Var(\varepsilon_t) = \omega + \alpha_1 Var(\varepsilon_{t-1})$
- $E(\eta_t | F_{t-1}) = 0$

1.3 Generalized AutoRegressive Conditional Heteroskedasticity (GARCH) Model

GARCH models, introduced by (see, e.g, [1]), extend the ARCH framework by allowing for a more complex structure of volatility dynamics.
GARCH models not only incorporate lagged squared residuals but also lagged conditional variances.

GARCH models are widely used in financial econometrics for modeling and forecasting volatility, especially in asset pricing, risk management, and derivative pricing.
They provide a flexible and powerful framework for capturing the time-varying nature of volatility in financial markets.

Definition 3 *Regarding an autoregressive model, expressed in the following format:*

$$Y_t = \mu + \varepsilon_t \quad \forall t \in \mathbb{R}^+,$$

ε_t *is a weak white noise that satisfies the property* $E[\varepsilon_t / I_{t-1}] = 0$

$$\varepsilon_t = \eta_t h_t$$

$$h_t^2 = \sigma_t^2 = \omega + \sum_{i=1}^{p} \alpha_i \varepsilon_{t-1}^2 + \sum_{j=1}^{q} \beta_j h_{t-j}^2 \tag{1.3.1}$$

Or $\eta_t \sim N(0, \sigma^2)$, η_t *is a weak white noise such as* $\omega > 0$, $\alpha_i > 0$ *for* $i = 1, 2, ..., p$ *and* $\beta_j > 0$ *for* $j = 1, 2, ..., q$ *satisfying to ensure the positivity of* $\omega + \sum_{i=1}^{p} \alpha_i \varepsilon_{t-1}^2 + \sum_{j=1}^{q} \beta_j h_{t-j}^2$

1.3.1 GARCH(1,1) model

Empirical financial market data is often modeled using the GARCH(1,1) error model. It is given by the equation:

$$Y_t = \mu + \varepsilon_t$$

with $\varepsilon_t = \eta_t h_t$,

and $h_t = \sqrt{\omega + \alpha_1 \varepsilon_{t-1}^2 + \beta_1 h_{t-1}^2}$

with $\omega > 0, \alpha_1 > 0$ and $\beta_1 > 0$, the squares of the residuals follow an ARMA(1,1) process.

$$\varepsilon_t^2 = \omega + (\alpha_1 + \beta_1)\varepsilon_{t-1}^2 + \beta_1 v_{t-1} + v_t$$

It is stationary for $\omega > 0$, $\alpha_1 > 0$, and $\beta_1 > 0$ or $_t = \varepsilon_t^2 - h_t^2$ is an innovation process for ε_t^2. Under the second-order stationarity condition, the unconditional variance of the process ε_t exists and remains constant over time.

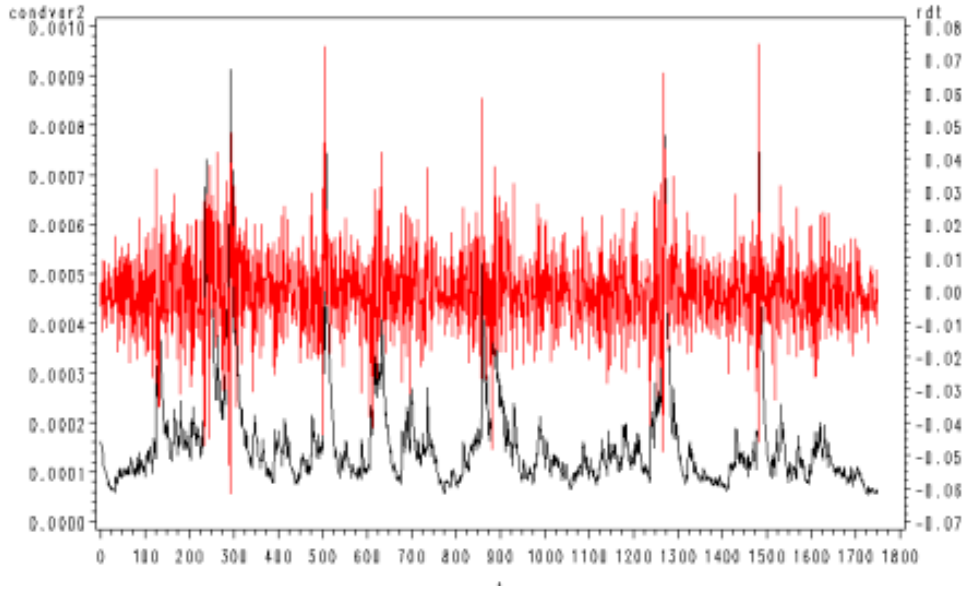


Figure 1.1: Conditional variance and estimated returns example GARCH(1, 1)

Given that $V(\varepsilon_t) = E(\varepsilon_t^2)$, it is sufficient, starting from the $ARMA(1,1)$ form on ε_t^2, to define the variance of the process:

$$V(\varepsilon_t) = \frac{\omega}{1 - (\alpha_1 + \beta_1)}$$

kurtosis exists if

$$3\alpha_1^2 + 2\alpha_1\beta_1 + \beta_1^2 < 1$$

and is given by:

$$\begin{aligned} K_u &= \frac{E[\varepsilon_t^4]}{E[\varepsilon_t^2]} \\ &= 3\frac{1-(\alpha_1+\beta_1)^2}{1-(\alpha_1+\beta_1)^2-2\alpha_1^2} \end{aligned} \tag{1.3.2}$$

It is always greater than three. Thus, if α_1 tends towards zero, the heteroscedasticity disappears and the value of kurtosis tends towards three. Finally, it can be shown that for a GARCH process, the kurtosis is directly related to conditional heteroscedasticity.

Consider the case of kurtosis associated with the unconditional distribution in a conditionally Gaussian GARCH process, such that $\eta_t \sim N(0,1)$.

In this case, the conditional moments of order 2 and 4 of the process ε_t are related :

$$E[\varepsilon_t^4/I_{t-1}] = 3[E(\varepsilon_t^2/I_{t-1})]^2$$

Indeed, it is recalled that if a centered variable X_t follows a centered normal distribution, then

$$E(X_t^4) = 3(Var(X_t))^2 = 3(E(X_t^2))^2$$

If we apply the expectation operator to both sides of the previous equation, it becomes

$$\begin{aligned} E(\varepsilon_t^4) &= E(E[\varepsilon_t^4/I_{t-1}]) \\ &= 3E([E(\varepsilon_t^2/I_{t-1})]^2) \\ &\geq 3E([E(\varepsilon_t^2/I_{t-1})])^2 \\ &= 3E^2(\varepsilon_t^2) \end{aligned} \tag{1.3.3}$$

We can calculate the kurtosis as follows:

$$\begin{aligned} K_u &= \frac{E[\varepsilon_t^4]}{E^2[\varepsilon_t^2]} \\ &= \frac{3E([E(\varepsilon_t^2/I_{t-1})]^2)}{E^2[\varepsilon_t^2]} \\ &= 3\frac{E^2[\varepsilon_t^2]}{E^2[\varepsilon_t^2]} + \frac{3}{E^2[\varepsilon_t^2]}(E([E(\varepsilon_t^2/I_{t-1})]^2) - E^2[\varepsilon_t^2] \\ &= 3 + \frac{3}{E^2[\varepsilon_t^2]}(E([E(\varepsilon_t^2/I_{t-1})]^2) - E^2[E(\varepsilon_t^2/I_{t-1})]) \\ &= 3 + 3\frac{Var[E\varepsilon_t^2/I_{t-1}]}{E^2[\varepsilon_t^2]} > 3 \end{aligned} \tag{1.3.4}$$

The kurtosis is thus linked to a measure of conditional heteroscedasticity.

1.4 Classification of GARCH processes

Since 1985, various specifications for GARCH models have been developed, and Drost and Nijman classified them in a paper presented at the ARCH process conference held in Paris in June 1990.(see, e.g, [3])

1.4.1 Strong GARCH

It is said that the process is strong $GARCH(p,q)$ in the case of a semi-strong GARCH such that the standardized innovation v_t is a strong white noise (a sequence of independent variables with the same law) and $\eta_t \sim N(0,1)$.
To motivate the introduction of GARCH processes, we can rewrite equation (1.3.1) using operators $\alpha(.)$ and $\beta(.)$. In this new context, these operators are defined by :

$$\alpha(L) = \alpha_1 L + \alpha_2 L^2 + ... + \alpha_p L_p$$

and

$$\beta(L) = \beta_1 L + \beta_2 L^2 + ... + \beta_p L_p$$

So, we can write:

$$\varepsilon_t = \eta_t \sqrt{\omega + \alpha(L)\varepsilon_t^2 + \beta(L)h_t^2}$$

So, we can write: L is the lag operator.

So, we have:

$$h_t^2 = \omega + \alpha(L)\varepsilon_t^2 + \beta(L)h_t^2$$

Similar to the ARCH model, we can express the process ε_t^2 as an ARMA process through inversion, defined by the innovation:

$$v_t = \varepsilon_t^2 - h_t^2$$

v_t: represents the innovation or unexpected shock at time t.
The innovation v_t can be interpreted as the difference between the observed squared value ε_t^2 and the expected conditional variance h_t^2.

If all the roots of $1 - \beta(L)$ are outside the unit circle, then we have:

$$\varepsilon_t^2 - v_t = \omega + \sum_{i=1}^{p} \alpha_i \varepsilon_{t-i}^2 + \sum_{j=1}^{q} \beta_j (\varepsilon_{t-j}^2 - v_{t-j})$$

Hence, we conclude that:

$$\varepsilon_t^2 = \omega + \sum_{i=1}^{p} (\alpha_i + \beta_i)\varepsilon_{t-i}^2 - \sum_{j=1}^{q} \beta_j v_{t-j} + v_t, \ t \in \mathbb{Z}$$

$n = max(p, q)$

With the convention: $\alpha_i = 0 \ \forall i > p$, $\beta_j = 0 \ \forall j > q$

1.4.2 Semi-strong GARCH

When the innovation process v_t and ε_t^2 is even assumed to be weak white noise, they call GARCH semi-strong GARCH. The same process ε_t when it comes to a martingale difference with an innovation process v_t that is itself a martingale difference. The semi-strong GARCH processes thus defined coincide well with the initial idea of Engle and Bollerslev, since it is clear conversely that if we assume that v_t is a martingale difference, we deduce that :

$$v_t = \varepsilon_t^2 - h_t^2$$

where h_t^2 is indeed the conditional variance of ε_t given past information.

1.4.3 Weak GARCH

A semi-strong GARCH is a weak GARCH, but the converse is not true.

The ARCH and GARCH processes have led to multiple studies and the development of other conditional variance processes of autoregressive form; we will present the main ones.

It is said that (ε_t) is a semi-strong GARCH(p, q) process if

- $E(\varepsilon_t/\varepsilon_{t-1}) = 0, \quad t \in \mathbb{Z}$
- There exist constants $w,\ \alpha_i,\ i = 1, ..., p$ and $\beta_j\ ,\ j = 1, ..., q$ such that

$$\sigma_t^2 = Var(\varepsilon_t/\varepsilon_{t-1}) = \omega + \sum_{i=1}^{p} \alpha_i \varepsilon_{t-i}^2 + \sum_{j=1}^{q} \beta_j \sigma_{t-j,}^2 \quad t \in \mathbb{R}^+. \tag{1.4.1}$$

1.5 Types of GARCH processes

Here are some common types of GARCH processes, and there are more versions and modifications available to meet various modeling needs in the field of financial economics and time series analysis.(see, e.g, [4])

1.5.1 EGARCH model

The EGARCH model introduces an additional source of nonlinearity by replacing the conditional variance with its logarithm, allowing to avoid constraints on the positivity of coefficients α_i and β_j. The model in which the EGARCH term is used is represented as follows:

Definition 4 *A process X_t satisfies an $EGARCH(p, q)$ representation if and only if:*

$$X_t = \varepsilon_t \sqrt{h_t} \tag{1.5.1}$$

$$\log(h_t) = \omega + \sum_{i=1}^{p} \alpha_i g(\varepsilon_{t-i}) + \sum_{j=1}^{q} \beta_j \log(h_{t-j})$$

where the normalized residual z_t is weak white noise and the function $g(.)$ satisfies:

$$g(\varepsilon_{t-i}) = \theta\varepsilon_{t-i} + \gamma(|\varepsilon_{t-i}| - E|\varepsilon_{t-i}|)$$

If we set $a_i = \theta\alpha_i$ and $b_i = \gamma\alpha_i$, the conditional variance of X_t can be rewritten as:

$$\log(h_t) = \omega + \sum_{i=1}^{p} a_i \varepsilon_{t-i} + \sum_{i=1}^{p} b_i(|\varepsilon_{t-i}| - E|\varepsilon_{t-i}|) + \sum_{j=1}^{q} \beta_j \log(h_{t-j}) \tag{1.5.2}$$

1.5.2 TGARCH model

The TGARCH (Threshold GARCH) model is a type of GARCH model that accounts for threshold effects in volatility.
This model allows positive and negative shocks to have different impacts on the volatility of the time series, helping to capture the asymmetric changes in financial volatility.

Definition 5 *A $TGARCH(p, q)$ process is expressed as:*

$$\begin{aligned} X_t &= \varepsilon_t h_t \\ h_t^2 &= \omega + \textstyle\sum_{i=1}^{p} (\alpha_i^+ X_{t-i}^+ - \alpha_i^- X_{t-i}^-) + \sum_{j=1}^{q} \beta_j h_{t-j}^2 \\ &= \omega + \alpha^+(L) X_t^+ - \alpha^-(L) X_t^- + \beta(L) h_t^2 \end{aligned}$$

where:

$$\begin{aligned} X_t^+ &= \max(X_t, 0) \\ X_t^- &= \min(X_t, 0) \end{aligned}$$

Since the specification is not based on a square but on the conditional standard deviation, it is possible to remove the constraints of positivity on the coefficients.
Removing these constraints allows for the consideration of the asymmetry phenomena previously described regarding volatility.

1.5.3 GJR-GARCH model

The Glosten-Jagannathan-Runkle Generalized Autoregressive Conditional Heteroskedasticity (GJR-GARCH) model is an extension of the GARCH model that accounts for the different impacts of positive and negative shocks on volatility.
It is used to model financial time series where asymmetric volatility effects are significant.
In the GJR-GARCH model, an additional term is included to allow the impact of negative shocks to differ from that of positive shocks, helping to capture the asymmetrical effects in volatility.

Definition 6 *The GJR-GARCH model is represented by the expression*

$$\zeta_t^2 = \omega + \sum_{i=1}^{p} \alpha_i \epsilon_{t-i}^2 + \sum_{j=1}^{q} \beta_j \sigma_{t-j}^2 + \sum_{i=1}^{p} \gamma_i I_{t-i} \epsilon_{t-i}^2 \tag{1.5.3}$$

where:

$$I_{t-i} = \begin{cases} 1 & \text{if } \epsilon_{t-i} < 0 \\ 0 & \text{if } \epsilon_{t-i} \geq 0 \end{cases}$$

1.6 Properties of GARCH processes

The theoretical properties of GARCH processes can be inferred using the same approach used to develop the properties of ARCH processes(see, e.g, [1]).

Proposition 1 *The process ε_t is a white noise if $E(\varepsilon_t^2) < \infty$.*

$E[\varepsilon_t] = E[E(\varepsilon_t / I_{t-1})]$
and

$$\begin{aligned} Cov(\varepsilon_t, \varepsilon_{t-k}) &= E(\varepsilon_t \varepsilon_{t-k}) \\ &= E[\varepsilon_{t-k} E(\varepsilon_t / I_{t-1)]})] = 0, \forall k > 0 \end{aligned} \tag{1.6.1}$$

Proposition 2 *A necessary condition for the existence of the variance of a GARCH(p, q) process is*

$$\sum_{i=1}^{p} \alpha_i + \sum_{j=1}^{q} \beta_j < 1$$

Remark 1 *If this condition is met along with the non-negativity constraints given above, it is also sufficient. Thus, the GARCH process is weakly stationary or second-order stationary.*

In the case where the previous inequality is saturated, i.e.,

$\sum_{i=1}^{p} \alpha_i + \sum_{j=1}^{q} \beta_j = 1$

In such a case, we say that the GARCH process is integrated, and we refer to it as an IGARCH (Integrated GARCH) process.

Proposition 3 *The process ε_t^2 with a GARCH(p,q) representation can be expressed as an ARMA(max(p,q),q) process defined within an innovation.*

$v_t = \varepsilon_t^2 - h_t^2$ *Such that:*

$\varepsilon_t^2 = \omega + \sum_{i=1}^{n} (\alpha_i + \beta_i)\varepsilon_t^2 + \sum_{j=1}^{p} \beta_j v_{t-j} + v_t$

$n = max(p, q)$

With the conversion $\alpha_i = 0$, if $i > p$ and $\beta_j = 0$ if $j > q$

1.7 Parameter Estimation of GARCH Model

1.7.1 Yule-Walker estimator

Based on the representation of $GARCH$ processes, parameter estimation can be conveniently carried out by utilizing existing Yule-Walker equations.
We first outline the idea of Yule-Walker estimation with the $ARCH(q)$ model as a special case of $GARCH(p, q)$ where $p = 0$
. Consider the stationary $ARCH(q)$ process $\bar{\varepsilon}_t$ defined by.

$$\sigma_t^2 = \omega + \alpha_1 \bar{\varepsilon}_{t-1}^2 + \ldots + \alpha_q \bar{\varepsilon}_{t-q}^2$$

whose representation in $\bar{\varepsilon}_t^2$ is:

$$\bar{\varepsilon}_t^2 - \alpha_1 \bar{\varepsilon}_{t-1}^2 - \ldots - \alpha_q \bar{\varepsilon}_{t-q}^2 = \omega + \eta_t \tag{1.7.1}$$

We will now explain the main steps in conducting the Yule-Walker estimation procedure in greater detail:

Step 1. Transform a nonzero mean process to a zero-mean one. For a stationary process we shall use the notation $\mu := E(\bar{\varepsilon}_t^2)$. Taking the expectation on both sides of (1.7.1), we have

$$\mu = \omega \left(1 - \sum_{i=1}^{q} \alpha_i\right)^{-1}$$

Now we denote $\bar{\varepsilon}_t^2 - \mu$ as ε_t^2 to obtain a zero-mean, stationary process satisfying

$$\varepsilon_t^2 - \alpha_1 \varepsilon_{t-1}^2 - \ldots - \alpha_q \varepsilon_{t-q}^2 = \eta_t \tag{1.7.2}$$

Step 2. Express ε_t^2 in terms of $\eta_s, s \leq t$. then

$$\varepsilon_t^2 = \sum_{j=0}^{\infty} \psi_j \eta_{t-j}, \quad \psi_0 = 1, \quad \psi_j = \sum_{k=1}^{q} \alpha_k \psi_{j-k} \text{ for } j \geq 1 \tag{1.7.3}$$

where $\psi_j = 0$ for $j < 0$.

Step 3. Derive the Yule-Walker equations. Multiplying each side of (1.7.2) by $\varepsilon_{t-k}^2, k = 0, 1, \ldots, q$, taking expectations and using (1.7.3) to evaluate the right-hand side of (1.7.2), we obtain the Yule-Walker equations:

$$\gamma(k) - \alpha_1 \gamma(k-1) - \ldots - \alpha_q \gamma(k-q) = \begin{cases} \sigma^2, & k = 0 \\ 0, & 0 < k \leq q \end{cases} \tag{1.7.4}$$

where $\gamma(k) = \operatorname{cov}\left(\varepsilon_t^2, \varepsilon_{t-k}^2\right)$ and $\sigma^2 = \operatorname{Var}(\eta_t)$.

Step 4. Solve for the Yule-Walker estimators $\hat{\alpha}_1, \ldots, \hat{\alpha}_q$ and $\hat{\sigma}^2$, by calculating the $q+1$ linear equations (1.7.4) with sample estimates:

$$\hat{\gamma}(k) = n^{-1} \sum_{t=k+1}^{n} \left(\varepsilon_t^2 - \hat{\mu}\right)\left(\varepsilon_{t-k}^2 - \hat{\mu}\right) \tag{1.7.5}$$

where $\hat{\mu} = n^{-1} \sum_{t=1}^{n} \varepsilon_t^2$.

Remark 2 *Stationarity is an important part required in the Yule-Walker estimation. It is necessary for the existence of the appropriate representation of ϵ_t^2 in Step 2. It also guarantees that we can consistently estimate $\gamma(k)$ in Step 4.*

We extend the previous result to a more general case where $p > 0$. Since the steps involved are analogous, we skip some details in the subsequent description and concentrate on the key points.

We start with the transformation of (1.3.1) :

$$\varepsilon_t^2 - \sum_{i=1}^{m} \gamma_i \varepsilon_{t-i}^2 = \eta_t - \sum_{j=1}^{p} \beta_j \eta_{t-j}$$

We express ε_t^2 as $\varepsilon_t^2 = \sum_{j=0}^{\infty} \psi_j \eta_{t-j}$, where Continuing as in Step 3, we find the Yule-Walker equations.

$$\gamma(k) - \gamma_1 \gamma(k-1) - \ldots - \gamma_m \gamma(k-m) = \sigma^2 \sum_{j=k}^{p} (-\beta_j) \psi_{j-k} \tag{1.7.6}$$

for $0 \leq k \leq m+p$, from which the unknown coefficients can be solved with the sample covariances.
Although the Yule-Walker estimation can be adapted to ARMA models, the corresponding equations are nonlinear in the unknown coefficients, as the following example reveals.

Example 1 *The Yule-Walker equations obtained from (1.7.6) for $k = 0, 1, 2$ are:*

$$\begin{aligned}
\hat{\gamma}(0) - \left(\hat{\alpha}_1 + \hat{\beta}_1\right)\hat{\gamma}(1) &= \left(1 - \hat{\alpha}_1\hat{\beta}_1\right)\hat{\sigma}^2 \\
\hat{\gamma}(1) - \left(\hat{\alpha}_1 + \hat{\beta}_1\right)\hat{\gamma}(0) &= -\hat{\beta}_1\hat{\sigma}^2 \\
\hat{\gamma}(2) - \left(\hat{\alpha}_1 + \hat{\beta}_1\right)\hat{\gamma}(1) &= 0
\end{aligned}$$

Simple algebra shows that

$$\hat{\beta}_1^2 - \frac{\rho_2^2 - 2\rho_1\rho_2 + 1}{\rho_2 - \rho_1}\hat{\beta}_1 + 1 = 0, \quad \hat{\alpha}_1 = \rho_2 - \hat{\beta}_1$$

where $\rho_1 := \hat{\gamma}(1)/\hat{\gamma}(0)$ and $\rho_2 := \hat{\gamma}(2)/\hat{\gamma}(1)$.

We solve the quadratic equation for $\hat{\beta}_1$, which would require much computing time and lead to possible nonexistence and nonuniqueness of solution.

1.7.2 Maximum likelihood estimator

A more common approach to parameter estimation is Maximum Likelihood method. Here we review the general estimation strategy of the technique. For a given set of observations $(x_1, x_2, \ldots, x_n)$ drawn from a probability distribution associated with a known density function f parameterized by θ, the likelihood function $L(\theta)$ is

$$L\left(\theta \mid (x_1, x_2, \ldots, x_n)\right) = f\left((x_1, x_2, \ldots, x_n) \mid \theta\right)$$

viewed as a function of θ with $x_1, x_2, \ldots, x_n$ fixed. An estimator of θ is then defined as

$$\hat{\theta} = \arg\max_{\theta \in \Theta} L(\theta) \tag{1.7.7}$$

In other words, the estimator is the value of θ that maximizes the probability of the observed sample.
For likelihood maximization one frequently uses numerical iterative methods, where a likelihood function is often replaced by its natural logarithm for computational convenience, which of course does not change the location of the maximum. Many algorithms exist for solving such problem, here we briefly recall score algorithm:

Let θ_j denote the parameter estimates after the j th iteration. θ_{j+1} is then calculated from

$$\theta_{j+1} = \theta_j + J^{-1}\left(\theta_j\right)\nabla L\left(\theta_j\right) \tag{1.7.8}$$

with the gradient

$$\nabla L = \frac{\partial L}{\partial \theta} \tag{1.7.9}$$

and the Fisher Information matrix

$$J = E\left(-\frac{\partial^2 L}{\partial\theta\partial\theta^T}\right) \tag{1.7.10}$$

Let us now look at the application of estimation in the case $p = 0$ and then, as before, we turn to the $GARCH(p, q)$ model with $p > 0$.

For the $ARCH(q)$ model defined by $\sigma_t^2 = \omega + \sum_{i=1}^{q} \alpha_i \varepsilon_{t-i}^2$, assuming that ε_t is conditionally normally distributed, we have

$$P\left(\varepsilon_t \mid F_{t-1}\right) = \frac{1}{\sqrt{2\pi}\sigma_t} \exp\left\{-\frac{1}{2}\frac{\varepsilon_t^2}{\sigma_t^2}\right\}$$

The log-likelihood function L of parameter vector $\theta = (\omega, \alpha_1, \ldots, \alpha_q)^T$ can be written as:

$$L(\theta) = \sum_{t=q+1}^{n} l_t(\theta) \tag{1.7.11}$$

where:

$$\begin{aligned} l_t(\theta) &= \log P\left(\varepsilon_t \mid F_{t-1}\right) \\ &= -\frac{1}{2}\log(2\pi) - \frac{1}{2}\log\sigma_t^2 - \frac{1}{2}\frac{\varepsilon_t^2}{\sigma_t^2} \end{aligned}$$

Therefore, the derivatives of l_t are given by:

$$\begin{aligned} \frac{\partial l_t}{\partial \theta} &= \frac{1}{2\sigma_t^2}\frac{\partial \sigma_t^2}{\partial \theta}\left(\frac{\varepsilon_t^2}{\sigma_t^2} - 1\right) \\ \frac{\partial^2 l_t}{\partial \theta \partial \theta^T} &= \frac{1}{2\sigma_t^2}\left(\frac{\varepsilon_t^2}{\sigma_t^2} - 1\right)\frac{\partial^2 \sigma_t^2}{\partial \theta \partial \theta^T} + \frac{1}{\sigma_t^4}\left(\frac{1}{2} - \frac{\varepsilon_t^2}{\sigma_t^2}\right)\frac{\partial \sigma_t^2}{\partial \theta}\frac{\partial \sigma_t^2}{\partial \theta^T} \end{aligned}$$

where

$$\frac{\partial \sigma_t^2}{\partial \theta} = \left(1, \varepsilon_{t-1}^2, \ldots, \varepsilon_{t-q}^2\right)^T$$

This allows us to construct the gradient

$$\nabla L = \frac{1}{2}\sum_{t=q+1}^{n} \frac{1}{\sigma_t^2}\frac{\partial \sigma_t^2}{\partial \theta}\left(\frac{\varepsilon_t^2}{\sigma_t^2} - 1\right) \tag{1.7.12}$$

and the Fisher Information matrix

$$J = \frac{1}{2}\sum_{t=q+1}^{n} E\left(\frac{1}{\sigma_t^4}\frac{\partial \sigma_t^2}{\partial \theta}\frac{\partial \sigma_t^2}{\partial \theta^T}\right) \tag{1.7.13}$$

employed in the updating scheme (1.7.8).

Example 2 *As an application, let us consider an ARCH (1) model with* $\sigma_t^2 = \omega + \alpha_1 \varepsilon_{t-1}^2$. *For the optimization of the log-likelihood function* $L(\omega, \alpha_1)$, *we carry out the iterations (1.7.8) with*

$$\nabla L = \frac{1}{2}\sum_{t=2}^{n} \frac{1}{\sigma_t^2}\left(\frac{\varepsilon_t^2}{\sigma_t^2} - 1\right)\begin{bmatrix} 1 \\ \varepsilon_{t-1}^2 \end{bmatrix}$$

and

$$J = \frac{1}{2}\sum_{t=2}^{n} E\left(\frac{1}{\sigma_t^4}\right)\begin{bmatrix} 1 & \varepsilon_{t-1}^2 \\ \varepsilon_{t-1}^2 & \varepsilon_{t-1}^4 \end{bmatrix}$$

where $E\left(\sigma_t^{-4}\right)$ *is estimated by* $(n-1)^{-1}\sum_{t=2}^{n}\left(\omega + \alpha_1\varepsilon_{t-1}^2\right)^{-2}$.

Having made this introduction, we continue to the GARCH(p,q) *model, which could be handled in a similar way.*
Under the assumption of conditionally normal distribution, we write the log*-likelihood function* L *with the same notation as in (1.7.11) of parameter vector* $\theta = (\omega, \alpha_1, \ldots, \alpha_q, \beta_1, \ldots, \beta_p)^T$.
When computing the estimator, the score-algorithm (1.7.8) is still valid with the gradient (1.7.12) and the Fisher Information matrix (1.7.13), however, in which

$$\frac{\partial\sigma_t^2}{\partial\theta} = v_t + \sum_{j=1}^{p}\beta_j\frac{\partial\sigma_{t-j}^2}{\partial\theta} \tag{1.7.14}$$

where

$$v_t = \left(1, \varepsilon_{t-1}^2, \ldots, \varepsilon_{t-q}^2, \sigma_{t-1}^2, \ldots, \sigma_{t-p}^2\right)^T$$

The only difference from the $ARCH(q)$ *regression model is the inclusion of the recursive part in (1.7.14).*

Example 3 *Now consider* $GARCH(1,1)$ *process with* $\sigma_t^2 = \omega + \alpha_1\varepsilon_{t-1}^2 + \beta_1\sigma_{t-1}^2$, *to estimate the unknown coefficients* $\theta = (\omega, \alpha_1, \beta_1)^T$, *we use iteration (1.7.8) with* J *and* ∇L *as defined above, where*

$$\frac{\partial\sigma_t^2}{\partial\theta} = v_t + \beta_1\frac{\partial\sigma_{t-1}^2}{\partial\theta}, \quad v_t = \left(1, \varepsilon_{t-1}^2, \sigma_{t-1}^2\right)^T \tag{1.7.15}$$

Remark 3 *From (see, e.g, [1]) it follows that, under the conditions:*

$$E\left(Z_t \mid F_{t-1}\right) = 0, \quad E\left(Z_t^2 \mid F_{t-1}\right) = 1, \quad E\left(Z_t^4 \mid F_{t-1}\right) < \infty \tag{1.7.16}$$

and strict stationarity of ε_t, *the estimator* $\hat{\theta}$ *is strongly consistent and asymptotically normally distributed:*

$$\sqrt{n}(\hat{\theta} - \theta) \to N_{p+q+1}\left(0, J^{-1}IJ^{-1}\right)$$

with

$$I = E\left(\frac{\partial l_t}{\partial\theta}\frac{\partial l_t}{\partial\theta^T}\right), \quad J = -E\left(\frac{\partial^2 l_t}{\partial\theta\partial\theta^T}\right)$$

Moreover replacing (1.7.16) with a stronger condition

$$Z_t \sim N(0,1)$$

as proved in (see, e.g, [1]) that $I = J$, *the asymptotic variance can be simplified to* J^{-1}, *that is,*

$$\sqrt{n}(\hat{\theta} - \theta) \to N_{p+q+1}\left(0, J^{-1}\right)$$

1.8 Information Criteria

Information criteria are used to compare $GARCH(p, q)$ models with different p and q values. Lower values indicate better models.

1.8.1 Akaike Information Criterion (AIC)

$$AIC(p, q) = \ln(\sigma^2) + \frac{2(p + q)}{T} \tag{1.8.1}$$

where the first term represents the model fit, while the second term is a penalty for the number of parameters.

1.8.2 Schwarz (Bayesian) Information Criterion (BIC)

$$BIC(p, q) = \ln(\sigma^2) + \frac{\ln T \cdot 2(p + q)}{T} \tag{1.8.2}$$

where the penalty is related to the sample size.

1.8.3 Final Prediction Error (FPE)

$$FPE(p) = \sigma^2 \left(\frac{n + p}{n - p} \right) \tag{1.8.3}$$

1.9 Point Forecasts

Point forecasts involve predicting future values using the assumed GARCH model. We can generate forecasts for future time steps using the GARCH model.

$$\hat{X}_{t+h|t} = E(X_{t+h}|X_t, X_{t-1}, \ldots, X_1) \tag{1.9.1}$$

Here, $\hat{X}_{t+h|t}$ represents the forecasted value at time $t + h$ based on information available up to time t.

1.10 Forecast Intervals

Forecast intervals provide a range within which future observations are expected to fall with a certain probability, taking into account the uncertainty in the forecasts. The forecast interval is typically given by:

$$\hat{X}_{t+h|t} \pm z \times \sigma_{\hat{X}} \tag{1.10.1}$$

where z (for example, if $\alpha = 0.05$, $z = 1.96$) is the critical value from the standard normal distribution corresponding to the desired confidence level, and $\sigma_{\hat{X}}$ is the standard error of the forecast.

1.11 Evaluating Forecast Accuracy

1.11.1 Metrics

Evaluating the accuracy of forecasts is crucial. Common metrics include:

- Mean Absolute Error (MAE):

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|X_i - \hat{X}_i| \tag{1.11.1}$$

- Mean Squared Error (MSE):

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(X_i - \hat{X}_i)^2 \tag{1.11.2}$$

- Root Mean Squared Error (RMSE):

$$RMSE = \sqrt{MSE} \tag{1.11.3}$$

- Mean Absolute Percentage Error (MAPE):

$$MAPE = 100\% \cdot \frac{1}{n}\sum_{i=1}^{n}\left|\frac{X_i - \hat{X}_i}{X_i}\right| \tag{1.11.4}$$

1.11.2 Visualizing Forecasts

Visualizing forecasts alongside actual values can provide insights into the model's performance. Common visualizations include:

- Line Plots: Plotting actual vs. forecasted values over time.
- Prediction Intervals: Including confidence intervals in the plots to show the uncertainty in forecasts.

1.12 Case Studies and Practical Examples

1.12.1 Real-world Applications of GARCH

GARCH models are widely used in various fields for time series forecasting. Examples include:

- Stock Market Predictions: Forecasting stock prices and indices.
- Weather Forecasting: Predicting temperature, precipitation, and other weather-related variables.
- Sales Forecasting: Predicting future sales for inventory management and planning.

1.12.2 Industry-specific Forecasting Examples

Different industries have specific requirements and use cases for forecasting. Some examples include:

- Finance: Risk management, portfolio optimization, and economic forecasting.
- Retail: Demand forecasting, inventory management, and supply chain optimization.
- Healthcare: Patient admission rates, resource allocation, and disease outbreak prediction.

1.13 Advantages and disadvantages of GARCH Model

1.13.1 Advantages

The GARCH model is distinguished by its ability to model changes in volatility over time, aiding in understanding financial market dynamics and risk management.
It is flexible in handling various volatility patterns and provides accurate forecasts of future volatility. Models like EGARCH can capture asymmetric effects in volatility.
These models are more statistically efficient than ARCH models and are widely used in estimating Value-at-Risk (VaR), evaluating risks, and developing mitigation strategies.(see, e.g, [9])

1.13.2 Disadvantages

Despite its numerous advantages, the GARCH model has some limitations. Results can be sensitive to chosen model specifications, making it difficult to determine the optimal model.
The model struggles to accurately predict extreme events or market shocks. Estimating complex models can be computationally intensive and time-consuming.
Additionally, GARCH models may not fully capture the complex nonlinearities present in financial data..(see, e.g,[6])

Chapter 2

Artificial Neural Networks

2.1 Introduction

Artificial Neural Networks (ANN) consist of a set of tools and computational methods. They are characterized by their ability to learn, generalize, memorize, classify, adapt, and make decisions.
They are applied in various fields, such as pattern recognition, control, and robotics. In industrial maintenance, neural networks are used to solve diagnostic problems through automatic classification of signals and shapes corresponding to the different states of normal and abnormal machine operations.
In this chapter, we present some concepts regarding the types and characteristics of artificial neural networks(see, e.g, [10]).

2.1.1 Historical

The history of Artificial Neural Networks (ANN) spans several decades, marked by significant milestones and advancements. Here's a brief historical overview:

- **1940s-1950s:** The foundational work on artificial neural networks began with the development of the McCulloch-Pitts neuron model by Warren McCulloch and Walter Pitts in 1943.
  In 1949, Donald Hebb introduced the concept of Hebbian learning, which laid the groundwork for associative learning in neural networks.

- **1950s-1960s:** Frank Rosenblatt developed the perceptron, a type of artificial neural network capable of binary classification, in the late 1950s.
  However, the limitations of perceptrons in solving complex problems led to a decline in interest in neural network research during the late 1960s.

- **1980s-1990s**: Neural network research experienced a resurgence in the 1980s, fueled by advancements in computing technology and the development of new learning algorithms. The backpropagation algorithm, proposed by Paul Werbos in 1974 and independently rediscovered by multiple researchers in the 1980s, allowed for efficient training of multilayer neural networks.
  This period also saw the emergence of various neural network architectures and applications, leading to increased interest from both academia and industry.

- **2000s-present:** The 21st century has seen significant advancements in neural network research due to the availability of large datasets, increased computational power, and the

development of deep learning techniques.
Deep neural networks have achieved remarkable success in various fields and have become a fundamental part of machine learning and artificial intelligence research, with applications in image and speech recognition, autonomous vehicles, and medical diagnosis.

Overall, the history of artificial neural networks reflects a journey of discovery, innovation, and continuous advancement, with each milestone building upon the achievements of previous generations of researchers.

2.2 Biological Neurons

A neuron is a cell that can transmit information to other neurons through its various connections (synapses). The human brain is the best model of an extremely fast multifunctional machine(see, e.g, [11]).

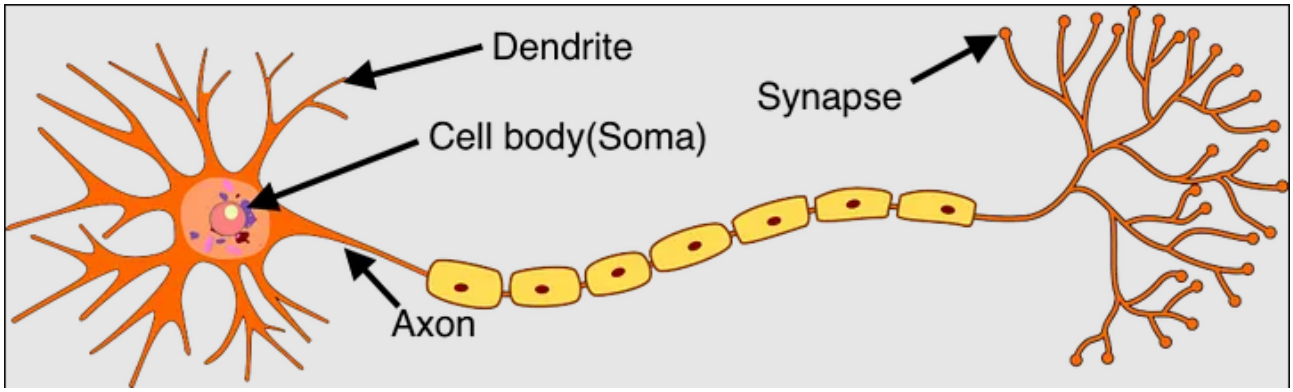


Figure 2.1: Schematic representation of a biological neuron.

2.2.1 Characteristics

In their general organization and biochemical system, neurons share many similarities with other cells. Here are the characteristics of biological neurons:

- Receiving signals from neighboring neurons.
- Integrating these signals.
- Generating a nerve impulse (nerve message).
- Conducting it.
- Transmitting it to another neuron capable of receiving it.

2.2.2 Structure

A neuron consists of three parts:

- **Dendrites:** receive messages.
- **Cell body:** generates the action potential (the response).
- **Axon:** transmits the signal to the next cells.
- **Synapse:** allows cells to communicate with each other, and it also plays a role in modulating the signals that pass through the nervous system.

2.3 Mathematical modeling of the biological neuron

2.3.1 The artificial neuron

The first systematic study of artificial neurons came from the neuropsychiatrist McCulloch and the logician Pitts, who were inspired by the study of biological neurons(see, e.g,[12]).

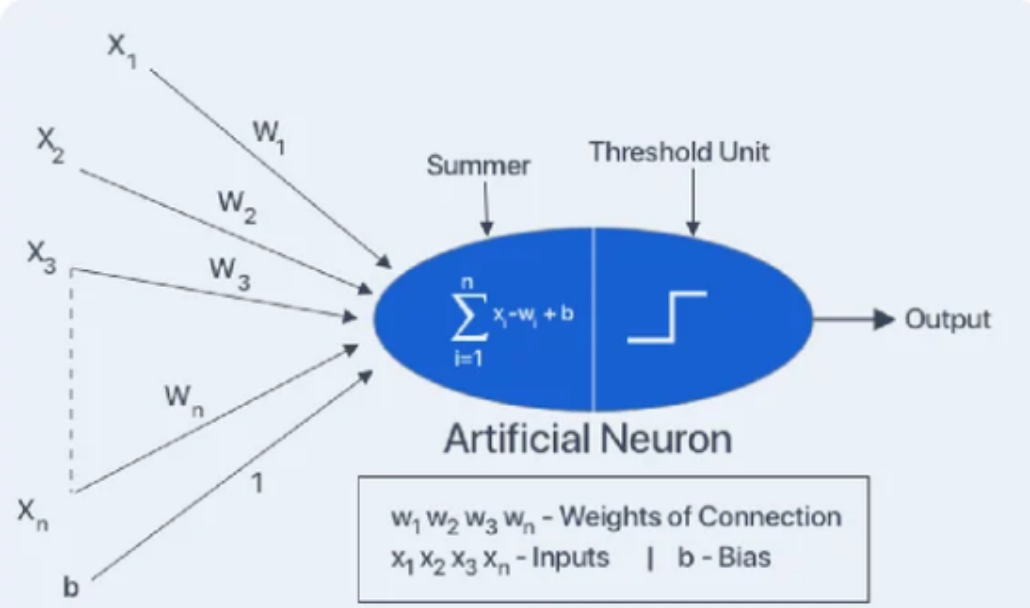


Figure 2.2: Model of an artificial neuron

It computes the weighted sum of the inputs :

$$Z = \sum w_i x_i - \theta \tag{2.3.1}$$

$$Y = f(Z) \tag{2.3.2}$$

- Z : The potential of the neuron.
- Y : The output of the neuron.
- x_i : The input signal i. (Inputs can be boolean, binary (0,1), bipolar (-1,1), or real).
- w_i : Weight of the connection to input i.
- θ : Bias.
- f : The activation function.

2.3.2 Principles of an artificial neuron

Each artificial neuron is a basic processor. It receives a variable number of inputs from upstream neurons or sensors comprising the machine to which it belongs.
Each input is associated with a weight representing the strength of the connection.
Each basic processor has a single output, which then branches out to feed a variable number of downstream neurons.

To each connection is associated a weight.

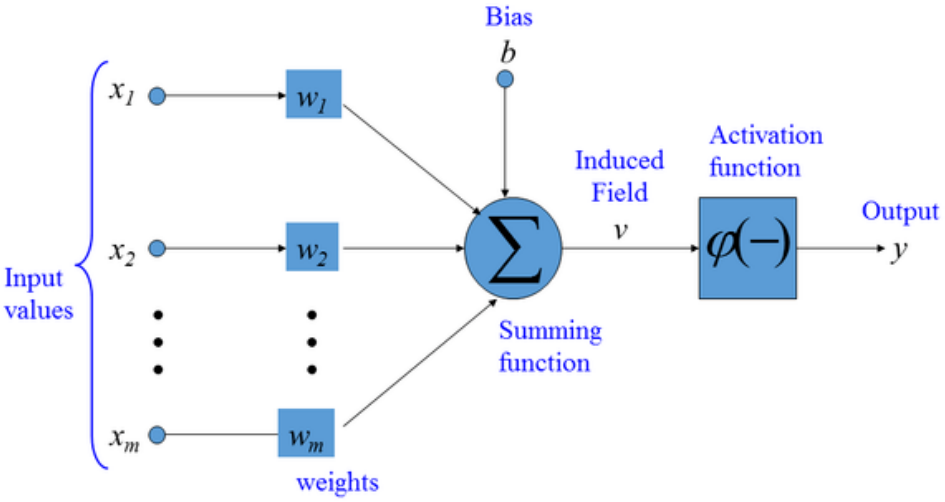


Figure 2.3: Nonlinear model of a neuron.

There is an obvious analogy (**Table**2.1) with biological neurons.

Biological neurons	Artificial neurons
Synapses	Weighted connections
Axons	Outputs
Dendrites	Inputs
summation	activation function

Table 2.1: The analogy between biological neurons and artificial neurons.

2.4 Activation functions

There are many possible forms for the activation function. The most common ones are(see, e.g, [13]):

2.4.1 Threshold (Hard Limit) Transfer Function

The Heaviside function (step function, unit step function) is a discontinuous function H that takes the value 0 for all negative real numbers and the value 1 everywhere else :

$$F(x) = \begin{cases} 0 & if \quad x < 0 \\ 1 & if \quad x \geq 0 \end{cases} \tag{2.4.1}$$

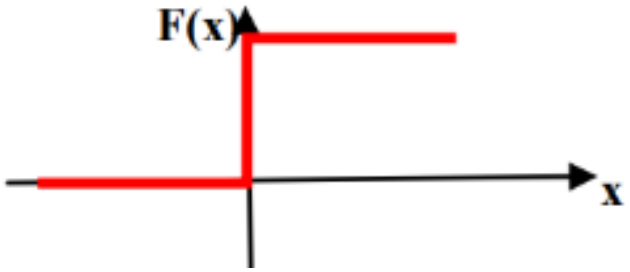


Figure 2.4: Heaviside function

2.4.2 Symmetric Hard Limit Transfer Function

The hard limiter function is the mostly used in classification of patterns, the characteristics of this function is shown in **Figure (2.5)** and its mathematical description is :

$$F(x) = \begin{cases} 1 & if \quad x < 0 \\ -1 & if \quad x \geq 0 \end{cases} \tag{2.4.2}$$

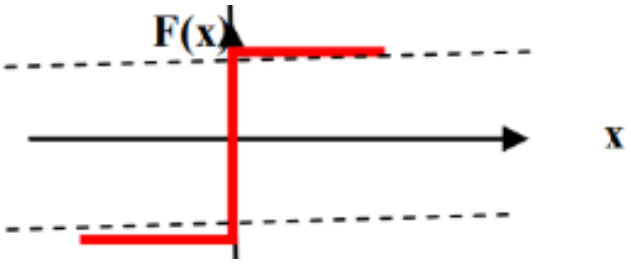


Figure 2.5: Sign function

This function is not differentiable. Therefore it cannot be used for continuation type of applications.

2.4.3 Linear Transfer Function

The Linear function characteristics are shown in **Figure (2.6)** and its mathematical description is :

$$F(x) = x, \quad x \in \mathbb{R} \tag{2.4.3}$$

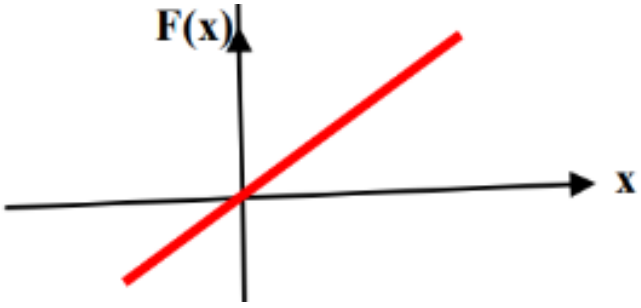


Figure 2.6: Linear Function

It is differentiable and is mostly used for output nodes of the networks.

2.4.4 Logistic (Log-Sigmoid) Transfer Function

Logistic function is a standard sigmoid function and is defined by

$$F(x) = \frac{1}{1 + e^{-x}} \tag{2.4.4}$$

The derivative of F is defined by $f(x) = F(x)(1 - F(x))$

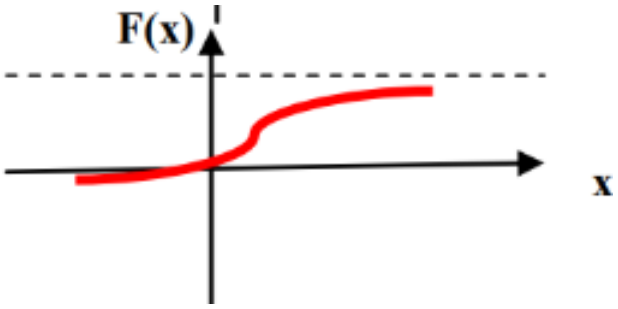


Figure 2.7: Sigmoid function

The log-sigmoid transfer function is commonly used in multilayer networks that are trained using the backpropagation algorithm.

2.4.5 Hyperbolic Tangent Transfer Function

The hyperbolic tangent is a sigmoid function and is defined by

$$F(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{2.4.5}$$

The derivative f of F is defined by

$$f(x) = (1 - F^{2}(x))$$

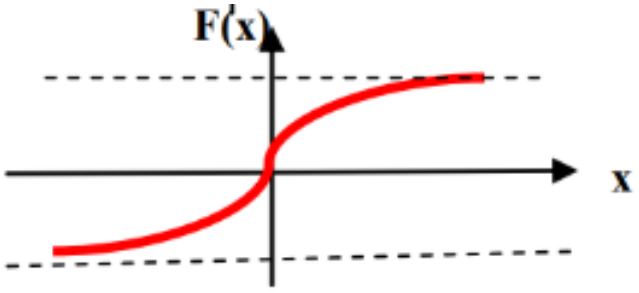


Figure 2.8: Hyperbolic Tangent

Since $\frac{\tan(\frac{x}{2})+1}{2} = \frac{1}{1+e^{-x}}$ then using the tanh function instead of the logistic one is equivalent. The tanh function has the advantage of being symmetrical with respect to the origin.

2.4.6 Saturating Linear Transfer Function

The saturating linear transfer function characteristics is shown in **Figure** (2.9) and its mathematical description is

$$F(x) = \begin{cases} 0 & if \quad x < 0 \\ x & if \quad 0 \leq x \leq 1 \\ 1 & if \quad x > 1 \end{cases} \tag{2.4.6}$$

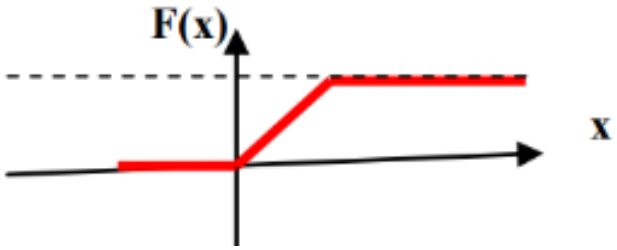


Figure 2.9: Saturation linear

2.4.7 Symmetric Saturating Linear Transfer Function

The symmetric saturating linear transfer function characteristics is shown in **Figure (2.10)** and its mathematical description is

$$F(x) = \begin{cases} -1 & if \quad x < -1 \\ x & if \quad -1 \leq x \leq 1 \\ 1 & if \quad x > 1 \end{cases} \tag{2.4.7}$$

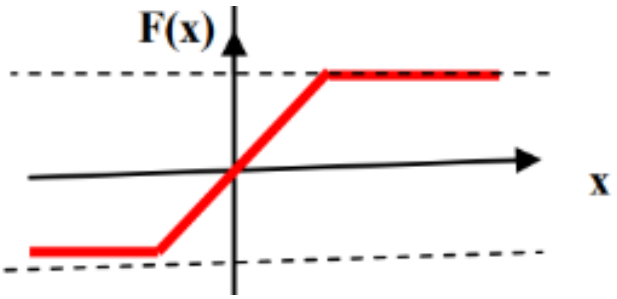


Figure 2.10: Saturated symmetric linear

2.4.8 Positive Linear Transfer Function

The positive linear transfer function characteristics is shown in **Figure(2.11)** and its mathematical description is

$$F(x) = \begin{cases} 0 & if \quad x < 0 \\ x & if \quad x \geq 0 \end{cases} \tag{2.4.8}$$

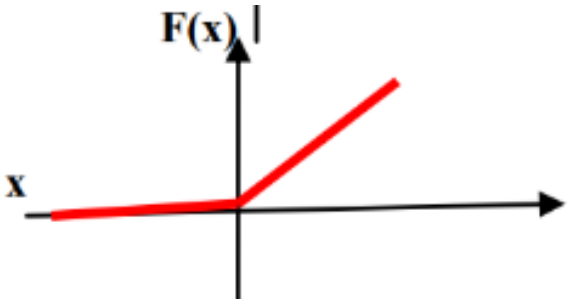


Figure 2.11: Positive linear

2.4.9 Radial Basis Transfer Function

The radial basis transfer function characteristics is shown in **Figure(2.12)** and its mathematical description is

$$F(x) = e^{-x^2} \tag{2.4.9}$$

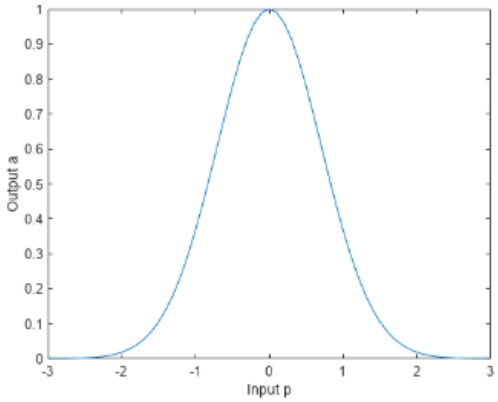


Figure 2.12: Radial Basis Function

2.5 Network Architectures

The connections between the neurons that make up the network describe the topology of the model. There are several network architectures(see, e.g, [14]), including:

2.5.1 Multilayer network

Neurons are arranged in multiple layers, with connections between neurons within each layer restricted, without any connections between neurons in the same layer, while connections occur only with neurons in the subsequent layers.

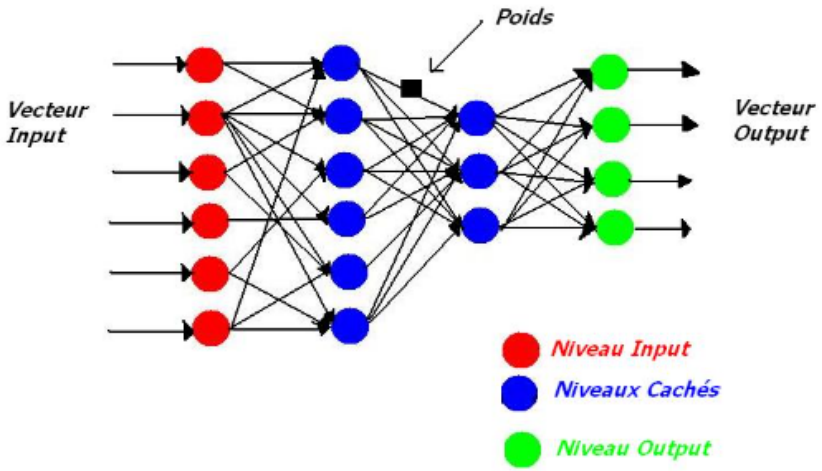


Figure 2.13: Classical multilayer network.

2.5.2 Local connection network

It's a multilayer structure. However, like the retina, it maintains a certain topological structure.
Each neuron maintains contact with a small number of local neurons in the downstream layer.

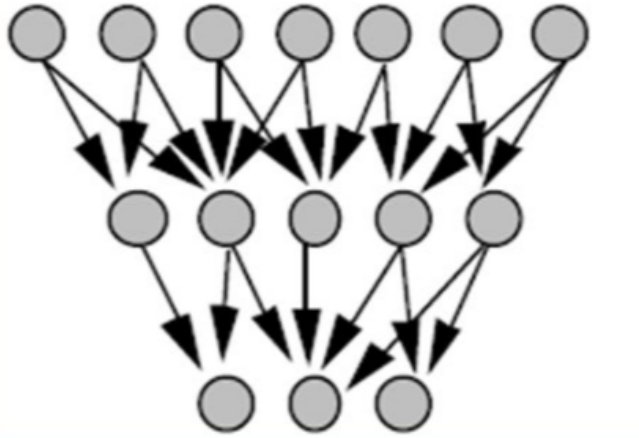


Figure 2.14: Locally connected network.

2.5.3 Recurrent Neural Network (RNN)

These are networks in which information propagates from layer to layer with possible backward feedback.

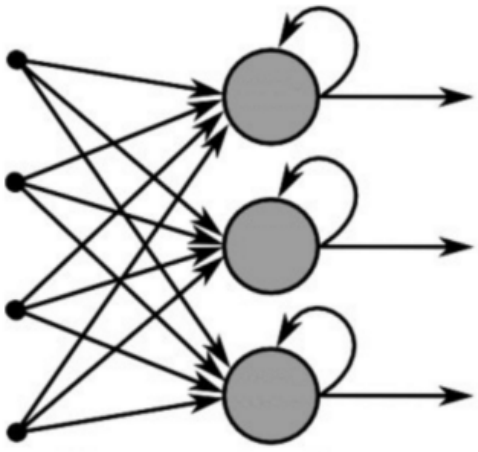


Figure 2.15: Recurrently connected network.

2.5.4 Fully connected network

This is the most general interconnection structure where each neuron is connected to every other neuron in the network.

Figure 2.16: Fully connected network.

2.6 Neural Network Learning

The learning of a neural network can be defined as the phase during which its various characteristic parameters are updated until they enable the network to best approximate the function it is intended to perform.
Depending on the application in which the network will be integrated, the function to be approximated may be known or unknown analytically.

In most current algorithms, the variables modified during learning are the weights of the connections.
Learning involves modifying the network weights to align the network's response with the examples and experience(see, e.g, [15]).

2.6.1 The type of learning in neural networks

There are two types of learning: supervised learning and unsupervised learning.

1. **Supervised learning**

   Supervised learning involves adjusting the synaptic coefficients of the network so that the network output corresponds to the desired output in each case.
   Supervised learning is the most common type of learning. Whenever you want to adjust your weights,

   - Every time you try, the error is calculated.
   - The weight is replaced by the smallest error, if there is one.

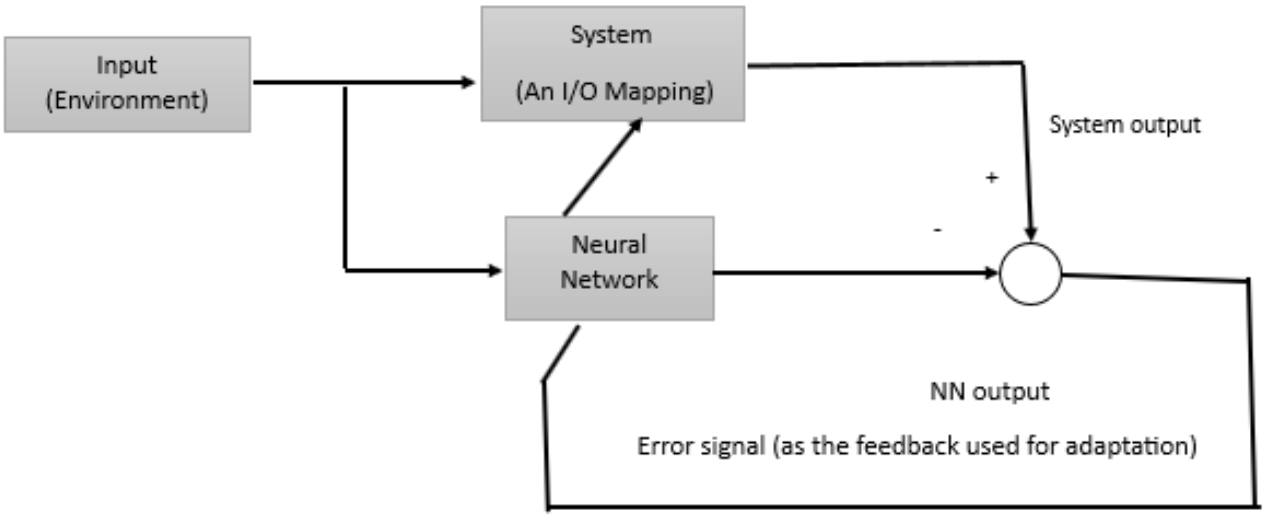


Figure 2.17: The block diagram of supervised learning in a neural network.

2. **Unsupervised learning**

   There is no prior knowledge of the outputs for given inputs. In fact, it's learning by exploration where the learning algorithm adjusts the weights of the connections between neurons to maximize the quality of input classification.

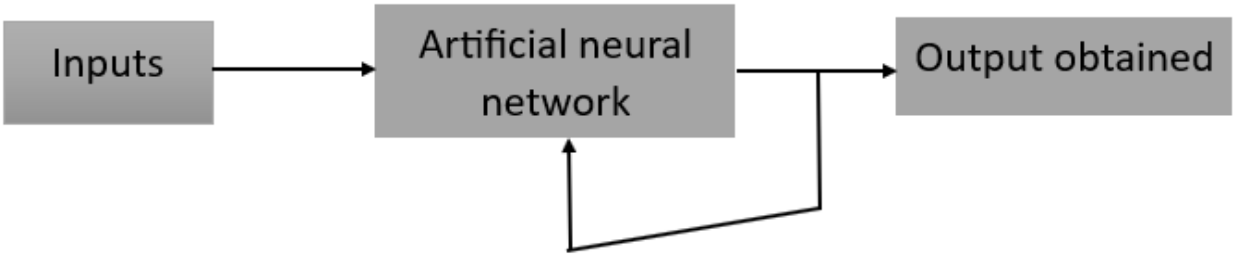


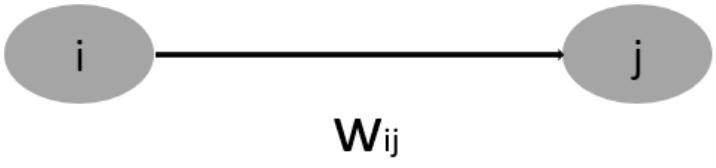
Figure 2.18: Block diagram of unsupervised learning in a neural network.

3. **Semi-supervised learning**

   In this learning context, the neural network hasn't developed a certain fundamental behavior yet.
   The network can determine qualitative indicators (true/false) for network performance.

2.6.2 Learning rule

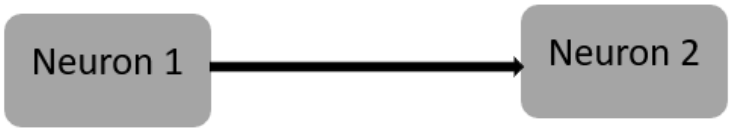
The purpose of learning is to modify the weight of connections between neurons.



There are several modification rules:

1. **Hebb's Rule**

   According to the results of neurobiological observation experiments: Neurons that fire together, wire together.
   The synaptic coefficients of neurons whose activities are synchronized are higher. When two connected units are operational at the same time, the strength of the connection increases.
   The following equations can be used to model Hebb's rule.



   Hebb's rule can be modeled by the following equations:

$$w_{ij}(t+1) = w_{ij}(t) + \Delta w_{ij} \tag{2.6.1}$$

$$\Delta w_{ij} = \varepsilon \cdot x_i \cdot x_j \tag{2.6.2}$$

   - x_j: The output value of the neuron j.
   - x_i: The output value of the neuron i.
   - ε: It is a positive constant representing the learning rate (epsilon) or decay.

2. **Widrow-Hoff rule of adaline (Delta rule**)

   This law is also a modified version of Hebb's law. Using the principle of error correction, it guides some artificial neural network learning algorithms.

$$E = d_i - x_i$$

   If the output is less than the desired response, for example, the weight of the connection should be increased, assuming that unit j is excitable (equal to 1). This rule can be expressed as follows:

$$\Delta w_{ij} = \varepsilon(d_i - x_i)x_j \tag{2.6.3}$$

   with:
   Output x_i and input x_j
   d_i: Desired response by the human expert.

3. **Learning algorithm**

   - Initialize weights and S-score using random (small) values.
   - Present an input $E_l = (e_1, ...e_n)$ from the training set.
   - For this input, compute the output sequence x.

$$a = \sum(w_i \cdot e_i) - s \tag{2.6.4}$$

$$x = \text{signe(a)} : \begin{cases} if\ a > 0 \text{ then } x = +1 \\ if\ a \leq 0 \text{ then } x = -1 \end{cases} \tag{2.6.5}$$

- If the output x is different from the desired output d_l for this input example E_l, then modify the weights:

$$w_{ij}(t+1) = \Delta w_{ij}(t) + u \cdot (x_i \cdot x_j) \tag{2.6.6}$$

- Once all examples in the training set (i.e., weight adjustments) are processed correctly, please return to step 2.

2.7 Perceptron

The perceptron is a model of an artificial neuron and can be considered one of the simplest forms of artificial neural networks.
It takes several binary inputs, applies weights to these inputs, sums the provided products, and then produces a binary output based on the result of this sum.
The perceptron is typically used for binary classification and can be trained using a supervised learning algorithm, such as the Widrow-Hoff rule(see, e.g, [16]).

2.7.1 Perceptron with a single layer

Due to its ability to learn simple pattern recognition, this network has attracted significant interest: it is the first of three networks that can be used with binary or continuous inputs.

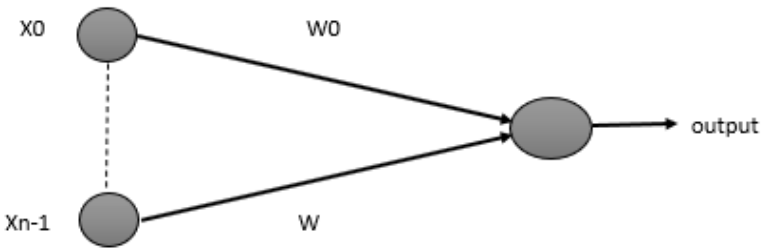


Figure 2.19: Perceptron with a single layer

2.7.2 Multilayer Perceptron (MLP)

The advancement of new learning algorithms has made this network widely used, consisting of non-recurrent networks with one or more layers of neurons between the input and output layers.
These additional layers contain hidden units or neurons that are not directly connected to the neurons of the input and output layers.
A perceptron with three layers including two hidden layers is illustrated in **Figure**2.20.

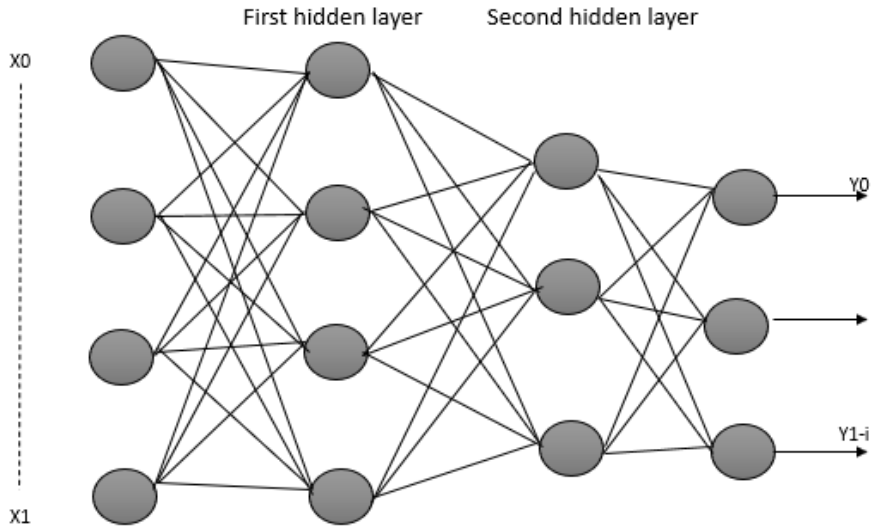


Figure 2.20: Multilayer Perceptron.

2.8 Radial Basis Function (RBF)

The Radial Basis Function (RBF) network is a model of connectivity that is easy to implement and understand. The RBF network typically consists of three layers:

- The input layer, which is a linear function that simply introduces the input signal into the hidden layer.
- The hidden layer, which is a processing unit that executes basic radial functions.
- The output layer, which is the neural output layer with a linear activation function.

RBF neural networks are widely used in areas such as function approximation, pattern classification, and prediction. They are particularly effective in representing nonlinear relationships and robust against noise in data(see, e.g, [14]).

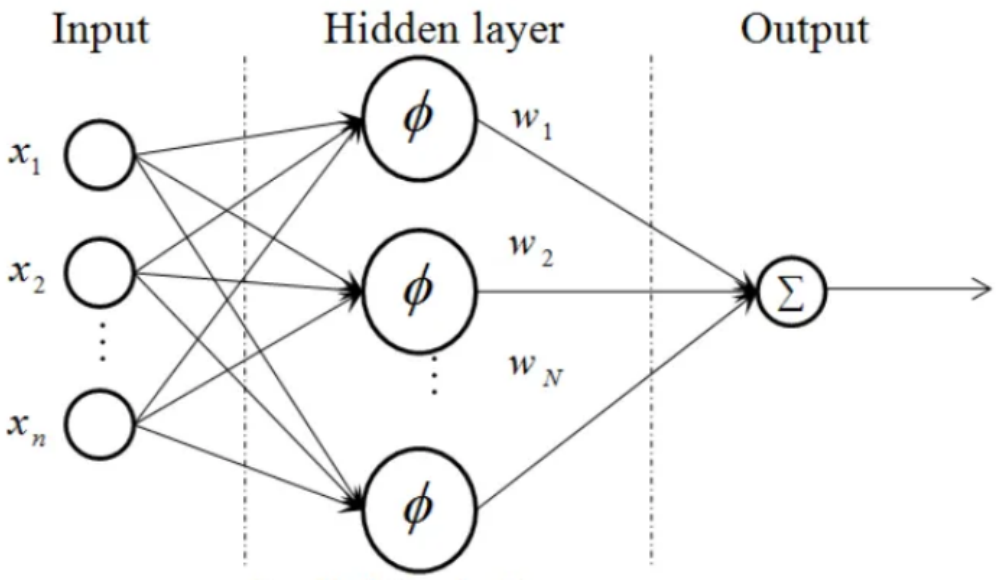


Figure 2.21: Radial Basis Function (RBF) neural networks.

$$\widehat{y} = \sum_{i=1}^{l} w_i e^{(-\frac{v^2}{\sigma^2})} \tag{2.8.1}$$

$$v_j(x) = ||c_j - x|| = \sqrt{\sum_{i=-1}^{n} (x_j - c_{ji})^2} \tag{2.8.2}$$

$v(x)$ The distance between the centers of the neurons and their input vectors

$$\sigma = \frac{v}{\sqrt{2s}} \tag{2.8.3}$$

σ: The standard deviation.

2.8.1 Training of RBF networks

- Calculate the center c_j using the classification algorithm.
- Calculate the dispersion coefficient using the method of mean distances.
- Calculate w_{ij} using the least squares method.

2.9 Training

In artificial neural networks (ANNs), optimizing the loss function involves finding the set of parameters (weights and biases) that minimize the difference between the predicted outputs of the network and the actual target values in the training data.

Commonly used loss functions in neural networks include:

- **The mean squared error (MSE)** for regression tasks

$$L(y, \hat{y}) = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2,$$

- **Binary cross-entropy** for binary classification tasks

$$L(y, \hat{y}) = -\frac{1}{n} \sum_{i=1}^{n} [y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i)],$$

- **Categorical cross-entropy for multi-class classification** tasks, Sparse categorical cross-entropy for multi-class classification with integer labels

$$L(y, \hat{y}) = -\frac{1}{n} \sum_{i=1}^{n} \sum_{c=1}^{C} y_{i,c} \log(\hat{y}_{i,c}),$$

- **the Kullback-Leibler divergence** for measuring the difference between probability distributions

$$L(P \| Q) = \sum_{i} P(i) \log \left(\frac{P(i)}{Q(i)} \right).$$

Here's a step-by-step guide on how loss function optimization is typically performed in ANNs:

1. **Choose a Loss Function**: Select an appropriate loss function based on the nature of the problem you're trying to solve. Common loss functions include mean squared error (MSE) for regression tasks, binary cross-entropy for binary classification tasks, and categorical cross-entropy for multi-class classification tasks.

2. **Choose an Optimization Algorithm**: Select an optimization algorithm to minimize the loss function. Popular optimization algorithms include gradient descent, stochastic gradient descent (SGD), Adam, RMSprop, and Adagrad. Each algorithm has its own advantages and is suitable for different scenarios.

3. **Compute Gradients**: Compute the gradients of the loss function with the respect to the network parameters (weights and biases). This is typically done using backpropagation, which efficiently computes the gradients of the loss function with the respect to each parameter in the network.

4. **Update Parameters**: Use the computed gradients to update the parameters of the network in the direction that minimizes the loss function. The magnitude of the update is determined by the learning rate, which controls how much the parameters are adjusted in each iteration.

5. **Repeat**: Repeat steps 3 and 4 for multiple iterations (epochs) or until a stopping criterion is met. This could be a maximum number of epochs, reaching a certain level of performance on a validation set, or convergence of the loss function.

6. **Evaluate Performance**: After training, evaluate the performance of the trained model on a separate test dataset to assess its generalization ability. This involves computing metrics such as accuracy, precision, recall, or mean squared error, depending on the task.

7. **Adjust Hyperparameters**: Fine-tune hyperparameters such as learning rate, batch size, and network architecture based on the performance on the validation set. This process may involve experimenting with different values for these hyperparameters to find the optimal configuration.

8. **Regularization**: Optionally, apply regularization techniques such as L1 or L2 regularization, dropout, or early stopping to prevent overfitting and improve the generalization performance of the model. By iteratively following these steps, the loss function of the ANN is optimized, leading to a model that accurately predicts the target values for new input data

Algorithms for updating hyperparameters (weights and biases) in neural networks, here are some commonly used ones:

1. Gradient Descent: Update weights by moving in the opposite direction of the gradient of the loss function.

$$w_{t+1} = w_t - \alpha \nabla L(w_t)$$

Stochastic Gradient Descent (SGD): Update weights using the gradient of the loss function computed on a subset of the training data (mini-batch).

$$w_{t+1} = w_t - \alpha \nabla L(w_t, x_i, y_i)$$

2. Adam: An adaptive learning rate optimization algorithm that computes adaptive learning rates for each parameter.

- $m_{t+1} = \beta_1 m_t + (1 - \beta_1) \nabla L(w_t)$

- $v_{t+1} = \beta_2 v_t + (1 - \beta_2)(\nabla L(w_t))^2$
- $\hat{m}_{t+1} = \frac{m_{t+1}}{1-\beta_1^{t+1}}$
- $\hat{v}_{t+1} = \frac{v_{t+1}}{1-\beta_2^{t+1}}$
- $w_{t+1} = w_t - \alpha \frac{\hat{m}_{t+1}}{\sqrt{\hat{v}_{t+1}}+\epsilon}$

3. RMSprop: Another adaptive learning rate optimization algorithm that divides the learning rate by an exponentially decaying average of squared gradients.

$$v_{t+1} = \beta v_t + (1 - \beta)(\nabla L(w_t))^2$$
$$w_{t+1} = w_t - \frac{\alpha}{\sqrt{v_{t+1}} + \epsilon} \nabla L(w_t)$$

4. Adagrad: An adaptive learning rate optimization algorithm that scales the learning rate for each parameter based on the historical gradients.

$$G_{t+1} = G_t + (\nabla L(w_t))^2 w_{t+1} = w_t - \frac{\alpha}{\sqrt{G_{t+1}} + \epsilon} \nabla L(w_t)$$

5. L-BFGS (Limited-memory Broyden-Fletcher-Goldfarb-Shanno): A quasi-Newton optimization algorithm that approximates the inverse Hessian matrix.

6. Each of these algorithms has its advantages and is suitable for different types of neural network architectures and problem domains. The choice of algorithm often depends on factors such as the dataset size, network architecture, computational resources, and convergence speed.

Where

1. w_t represents the weight vector at time step t.
2. α denotes the learning rate.
3. $\nabla L(w_t)$ is the gradient of the loss function L with respect to the weights w_t.
4. β_1 and β_2 are exponential decay rates.
5. m_t and v_t are moment estimates.
6. $\hat{m}_{t+1}$ and $\hat{v}_{t+1}$ are bias-corrected moment estimates.
7. G_t represents the sum of the squares of past gradients up to time step t.
8. ϵ is a small constant to prevent division by zero.

2.10 Advantages and disadvantages of neural networks

2.10.1 Advantages

Neural networks have several capabilities that make them effective in processing complex data. They can learn from data and adapt to non-linear patterns.
They exhibit fault tolerance by handling noisy or incomplete data through their ability to generalize patterns.
Their massive parallelism allows for fast data processing across many neurons.
Neural networks can be used for a variety of tasks, including classification, prediction, and pattern recognition. Some types, like self-organizing maps, can organize data in an unsupervised manner(see, e.g, [16]).

2.10.2 Disadvantages

Neural networks have several limitations. Their complexity makes them difficult to understand and interpret.
Training them is computationally intensive and time-consuming, especially with large datasets and complex architectures.
They are prone to overfitting, which leads to poor performance on new data. Large amounts of labeled data are often required, which may not always be available. They are vulnerable to adversarial attacks and require powerful, costly hardware like GPUs.
Additionally, tuning hyperparameters is challenging and time-consuming(see, e.g, [16]).

Chapter 3

Hybrid Model GARCH-ANN

3.1 Introduction

The hybrid model is an approach that combines multiple methods or different models in data analysis or prediction. This is typically done by integrating different features from various models to achieve a stronger and more accurate model.

The hybrid model between the GARCH model and Artificial Neural Network (ANN) combines the strengths of both models to enhance the prediction of volatility in financial or economic data.
The GARCH model analyzes and predicts volatility in time-series data, while the Artificial Neural Network is used to learn nonlinear relationships in the data(see, e.g, [17]).

Generally, the hybrid model is applied in a similar manner to how each of the GARCH model and Artificial Neural Network is individually applied, using financial or economic data to train the model and evaluate its performance.
Integrating the advantages of each model and their variations in the data can lead to improved accuracy in predicting volatility, thus enhancing risk management and decision-making in financial or economic markets(see, e.g, [19]).

3.2 Methodology

Due to their successful application in forecasting the volatilities of economic and financial variables, GARCH-type models are considered as part of the proposed hybrid models.

EGARCH and GJR-GARCH models are used for constructing these hybrid models, as they allow for leverage effects, where a large decrease in the price of financial assets can have a greater impact on volatility than a large increase in price(see, e.g, [1]).

Moreover, due to the complex non-linear correlation structures among financial variables, more flexible models are required to approximate the features of these variables, such as Artificial Neural Network (ANN) models.
The greatest advantage of ANN models lies in their ability to model complex nonlinear relationships without making prior assumptions about the nature of the relationship. They can relate a set of input variables to one or more output target variables in a nonlinear manner, providing significant flexibility.

Therefore, ANN models are employed to construct the proposed hybrid models(see, e.g, [6]).

There are several training methods for neural networks, with Back-propagation being the most common.
In this method, the parameters of the model are updated iteratively according to the quadratic loss function during the model estimation process, allowing for the achievement of the lowest error possible(see, e.g, [20]).

3.3 Hybrid Models

The purpose of this study is to propose two types of hybrid ANN and GARCH-type models and to compare the forecast performance of volatility for both hybrid models. Type I model is established by inputting the outcome of the preferred GARCH-type models into ANN, called ANN-GARCH model.
On the other hand, Type II model is built by considering the output layer of ANN as a variable of GARCH-type models, called GARCH-ANN model, so that the augmented GARCH-type models can behavior better on forecasting volatility.
Both types of models are expressed as follows:

3.3.1 Type *I*: ANN-GARCH model

Based on the preferred EGARCH model, $\beta_1 \log(h_{t-1})$, $\alpha_1 \left| \frac{\varepsilon_{t-1}}{\sqrt{h_{t-1}}} \right|$, and $\gamma_1 \frac{\varepsilon_{t-1}}{\sqrt{h_{t-1}}}$ are chosen as the endogenous explanatory variables, which are regarded as the input variables in the ANN. Similar to EGARCH, $\beta_1 h_{t-1}$, $\alpha_1 \varepsilon_{t-1}^2$, and $\gamma \varepsilon_{t-1}^2 I_{t-1}$ of the preferred GJR-GARCH model are chosen as the input variables in the ANN.

3.3.2 Type *II*: GARCH-ANN model

$$EGARCH-ANN : \log(h_t) = \omega + \sum_{j=1}^{q} \beta_j \log(h_{t-j}) + \sum_{i=1}^{p} \alpha_i \left| \frac{\varepsilon_{t-i}}{\sqrt{h_{t-i}}} \right| + \sum_{k=1}^{r} \text{and } \gamma_k \frac{\varepsilon_{t-k}}{\sqrt{h_{t-k}}} + \sum_{h}^{s} \xi_h \psi(z_t \lambda_h) \tag{3.3.1}$$

$$GJR-EGARCH-ANN : h_t = \omega + \sum_{m=1}^{p} \alpha_m \varepsilon_{t-m}^2 + \gamma \varepsilon_{t-1}^2 I_{t-1} \sum_{n=1}^{q} \beta_n h_{t-n} + \sum_{h}^{s} \xi_h \psi(z_t \lambda_h) \tag{3.3.2}$$

$$\psi(z_t \lambda_h) = \left[1 + \exp\left(\lambda_{h,d,w} + \sum_{d=1}^{v} \left[\sum_{w=1}^{m} (\lambda_{h,d,w} z_{t-d}^{w}) \right] \right) \right]^{-1} \tag{3.3.3}$$

where: $z_{t-d} = \left[\varepsilon_{t-d} - E(\varepsilon) \right] / \sqrt{E(\varepsilon^2)}$. $\frac{1}{2} \lambda_{h,d,w} \sim uniform\left[-1, +1 \right]$.

3.4 Forecast Encompassing

To evaluate the characteristics of the forecast series, two forecast encompassing tests are performed, based on Cook's (2012) methodology. The first test is the Fair and Shiller (1989) test, which can be derived from the following regression(see, e.g, [22, 24]):

$$RV_t = c + \lambda_1 f_{1,t} + \lambda_2 f_{2,t} + u_t \tag{3.4.1}$$

where RV_t is the realized volatility, c is a constant, $f_{1,t}$ is the volatility forecast from model 1, and $f_{2,t}$ is the volatility forecast from model 2. When $\lambda_1 = 0$ and $\lambda_2 \neq 0$, the forecast from model 2 encompasses (outperforms) the forecast from model 1. Conversely, the forecast from model 1 encompasses the forecast from model 2 if $\lambda_1 \neq 0$ and $\lambda_2 = 0$. If both $\lambda_1 \neq 0$ and $\lambda_2 \neq 0$, then we fail to reject the null hypothesis, indicating that both forecasts contain independent information.

The second test evaluates how well one forecast explains the error of another. Forecast error represents the information a forecast missed. Letting the forecast error be $e_t = RV_t - f_t$, the Chong and Hendry (1986) test uses the following regressions(see, e.g, [23]):

$$e_{1,t} = \lambda_2 f_{2,t} + v_{1,t} \tag{3.4.2}$$

$$e_{2,t} = \lambda_1 f_{1,t} + v_{2,t} \tag{3.4.3}$$

If the forecast error of model 1 ($e_{1,t}$) is not related to the forecast of model 2 ($f_{2,t}$), meaning $\lambda_2 = 0$, then forecast 1 can be used on its own.
However, if the forecast error is influenced by the other forecast, a combined forecast using both $f_{1,t}$ and $f_{2,t}$ should be created.

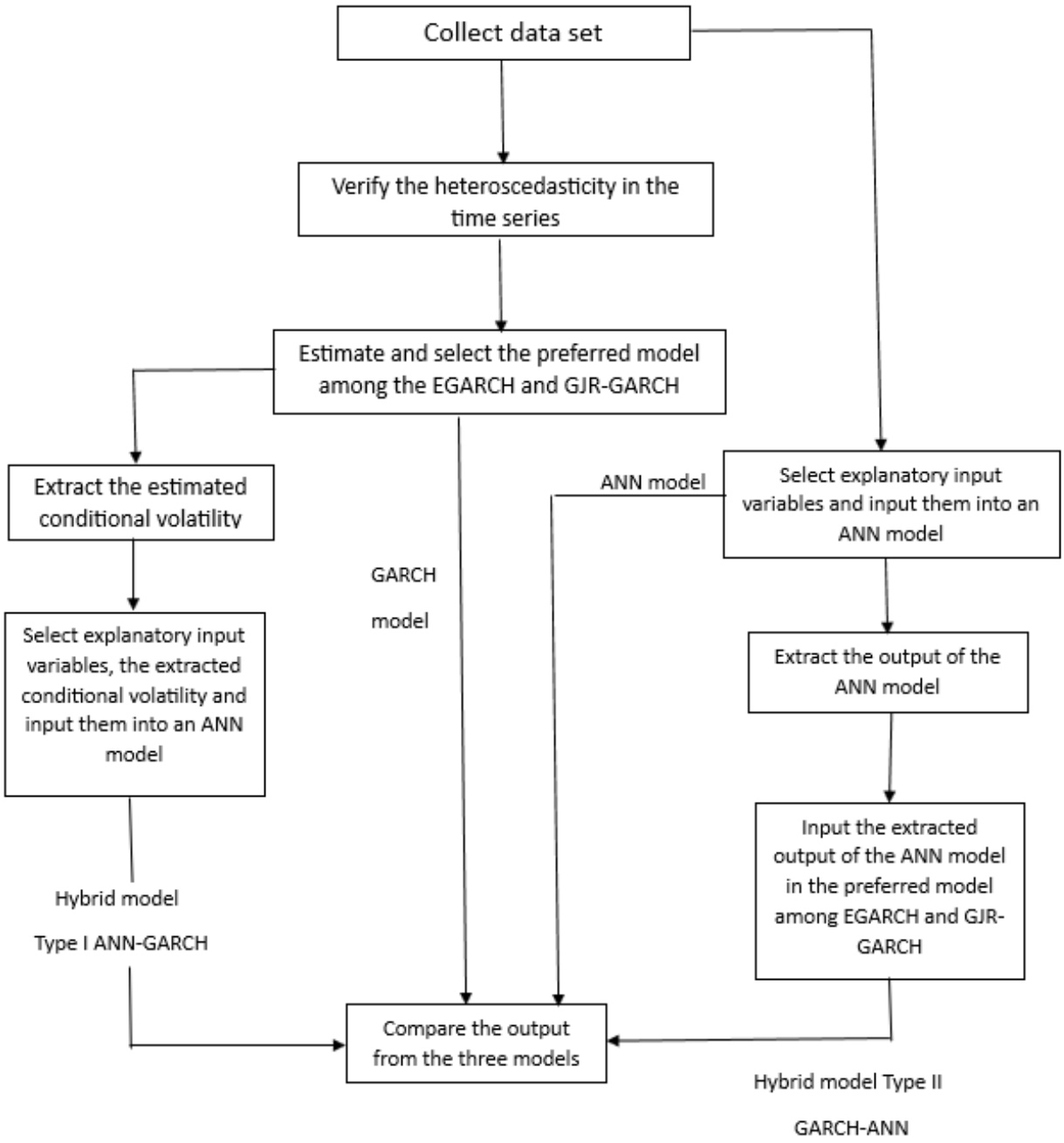


Figure 3.1: Flowchart of the modelling process.

The flowchart in (**Figure3.1**)illustrates the process of collecting and analyzing data using GARCH, ANN, and hybrid models. Here is an explanation of each part of the flowchart:

- **Collect data set**: This is the first step where the required data for analysis is collected.
- **Verify the heteroscedasticity in the time series:** Non-homogeneity of variance in time series data is verified using tests such as the ARCH test.
- **Estimate and select the preferred model among the EGARCH and GJR-GARCH:** The preferred model between EGARCH and GJR-GARCH models is estimated and selected.
- **Extract the estimated conditional volatility:** The estimated conditional variance is extracted from the GARCH model.

- **Select explanatory input variables, the extracted conditional volatility and input them into an ANN model:** The explanatory variables and the estimated conditional variance are selected and input into the Artificial Neural Network (ANN) model.
- **Select explanatory input variables and input them into an ANN model:** The explanatory variables are selected and input into the ANN model.
- **Extract the output of the ANN model:** The outputs of the ANN model are extracted.
- **Input the extracted output of the ANN model in the preferred model among EGARCH and GJR-GARCH:** The extracted outputs of the ANN model are input into the preferred model between EGARCH and GJR-GARCH.
- **Hybrid model Type I ANN-GARCH:** Creating a hybrid model of the first type that combines ANN and GARCH.
- **Hybrid model Type II GARCH-ANN:** Creating a hybrid model of the second type that combines GARCH and ANN.
- **Compare the output from the four models:** The outputs of the four models (GARCH, ANN, and the hybrid models) are compared to determine the best model.

3.5 Advantages and disadvantages of Hybrid Models

3.5.1 Advantages

A hybrid model combines the strengths of different models to boost prediction accuracy and performance.
It excels in predicting volatility and adapts well to diverse data challenges, including unstable or noisy conditions. By integrating various models, it provides comprehensive insights into complex data, enhancing understanding and forecasting capabilities significantly.
Thus, the hybrid model stands out as a versatile and effective approach in predictive analytics and decision-making processes.

3.5.2 Disadvantages

Hybrid models have several drawbacks, including increased complexity, higher computational resource requirements, and the need for specialized expertise.
They are more challenging to interpret, have a higher risk of overfitting, face integration challenges between different model components, and require more time and computational resources compared to simpler models.

Chapter 4

Application

4.1 Introduction

This study aims to provide a comprehensive comparison of GARCH, ANN, and hybrid models for time series forecasting. By leveraging both simulated and real-world data, we seek to highlight the strengths and limitations of each approach and demonstrate the potential benefits of hybrid models in capturing both linear and nonlinear dynamics in time series data. The findings of this study will contribute to the growing body of knowledge in time series forecasting and provide practical insights for researchers and practitioners in the field.

4.2 Simulation

To simulate data using a GARCH model for forecasting, we first generate a time series that exhibits characteristics commonly observed in real-world financial data, such as volatility clustering.
The GARCH (Generalized Autoregressive Conditional Heteroskedasticity) model is particularly suited for this purpose as it models the variance of the time series as a function of past variances and past forecast errors.
Specifically, we simulate the time series data by specifying parameters for the mean equation (e.g., an ARIMA process) and the variance equation (GARCH process), ensuring that the generated data reflects realistic patterns of changing volatility over time.
We will apply simulations for the GARCH model using the Python programming language to program and execute tests and benchmarks that measure the system's performance.

4.2.1 Steps for GARCH Model Simulations

- Using the parameters for the $(\omega = 0.002, \alpha = 0.234, \beta = 0.74345))$ model for a time series of 100 samples of size 500 and 1000.
- Calculate daily returns using the formula $r_t = \mu + \varepsilon_t = \eta_t h_t + \mu$, $\mu = -0.0003$
- Calculate implied volatility using the formula $h_t = \sqrt{\alpha_0 + \alpha_1 \varepsilon_{t-1}^2 + \beta_1 h_{t-1}^2}$
- Extract parameters for the sets of each series.
- Calculate the Mean Squared Error (MSE) to compare the extracted values with the values taken in the simulation. To calculate the Mean Squared Error (MSE):

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2$$

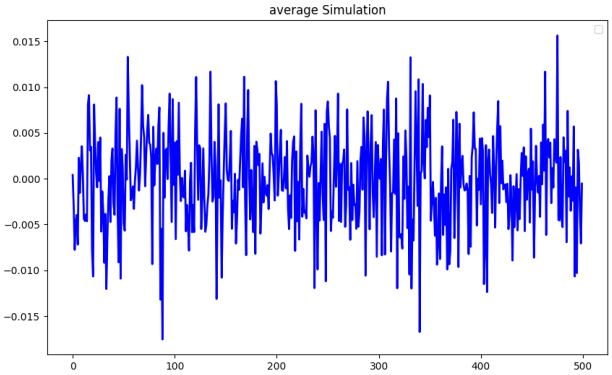
Where:

- n is the number of points in the time series.
- Y_i is the actual value of point i.
- $\hat{Y}_i$ is the predicted value of point i.

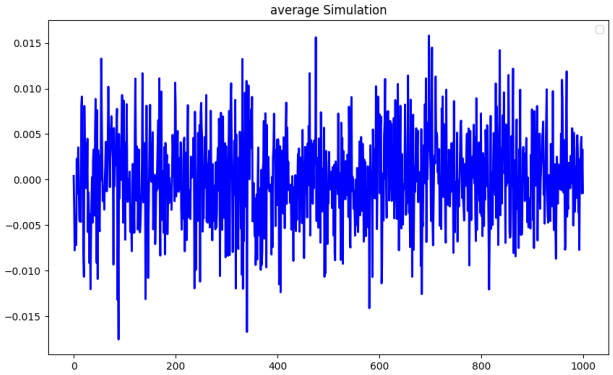
- Divide the data into two sets: a training set 97% and a test set 3%, then use the training set to build the model and the test set to evaluate the model.
- Use the GARCH model to predict the variances for the test set and compare them with the implied volatility.
- Calculate the Mean Squared Error (MSE) to measure the accuracy of the model in predicting the variance.
- Calculate the mean of the test sets and the mean of the prediction sets.

4.2.2 GARCH(1,0) or ARCH(1)

We will perform a simulation of the $ARCH(1)$ model using the parameters ($\omega = 0.002, \alpha = 0.234$).
The simulation will generate a curve representing the average daily returns across different groups, as well as a curve illustrating the average implied volatility across the $ARCH(1)$ groups. Additionally, we will display the parameter values extracted from these groups to further analyze the model's behavior.

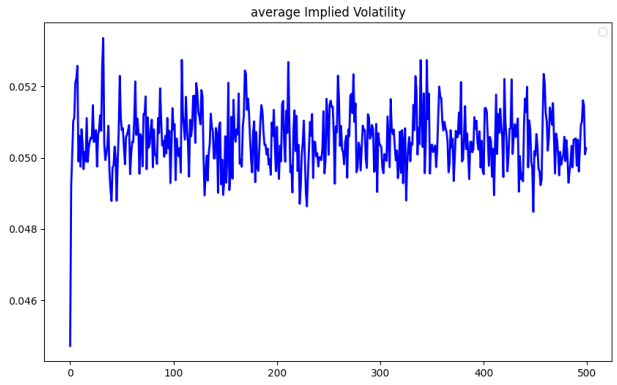


(a) Sample size 500

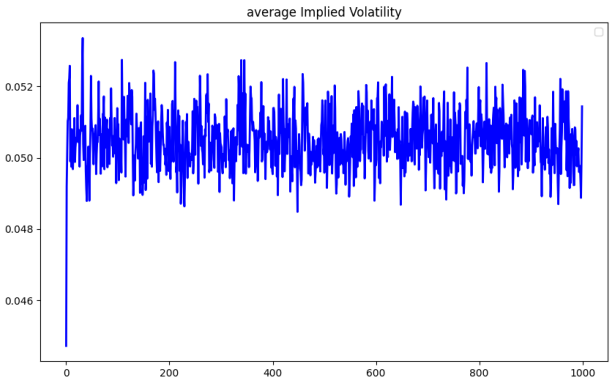


(b) Sample size 1000

Figure 4.1: The curves represent the average of daily returns groups.

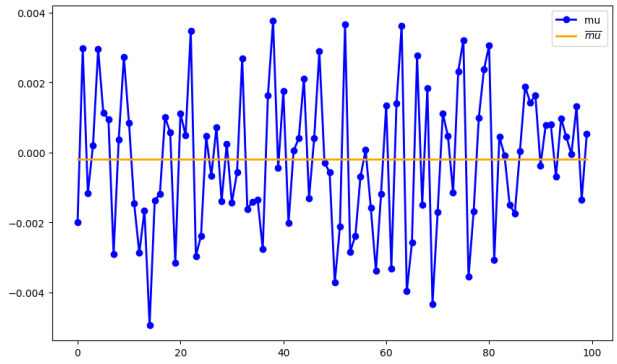


(a) Sample size 500

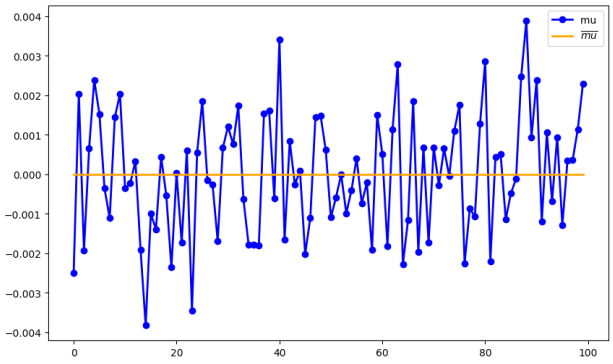


(b) Sample size 1000

Figure 4.2: The curves represent the average implied volatility across $ARCH(1)$ groups.

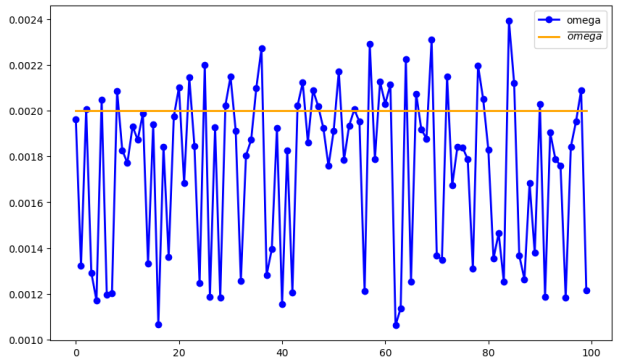


(a) Sample size 500

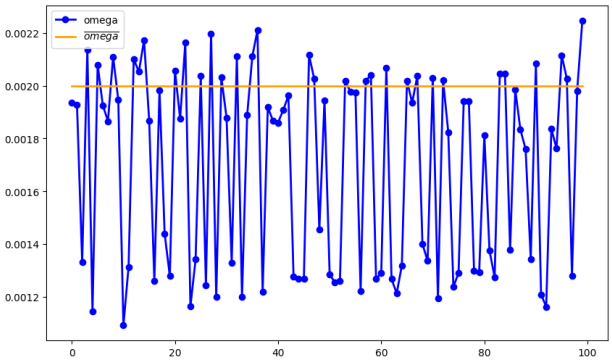


(b) Sample size 1000

Figure 4.3: The curves represent the values of the μ coefficient

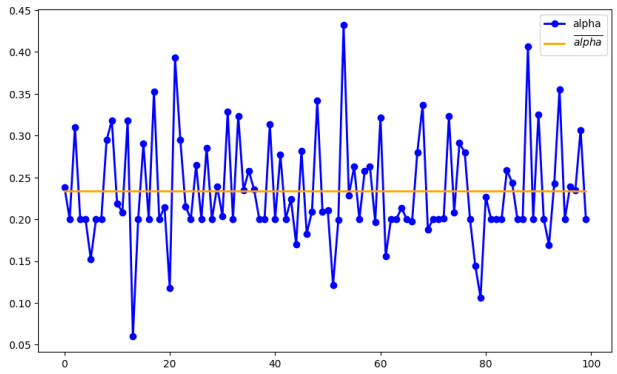


(a) Sample size 500

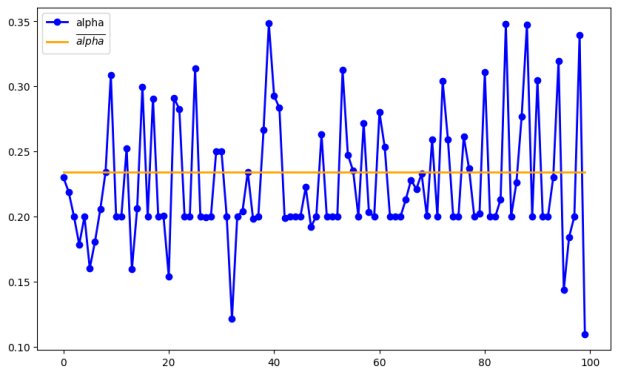


(b) Sample size 1000

Figure 4.4: The curves represent the values of the ω coefficient



(a) Sample size 500



(b) Sample size 1000

Figure 4.5: The curves represent the values of the α coefficient

	Average of Parameters
μ	-0.000196
ω	0.001745
α	0.234086

Sample size 500

	Average of Parameters
μ	0.000005
ω	0.001699
α	0.226668

Sample size 1000

Table 4.1: The tables represent the average parameters.

	MSE
μ	4.105774e-06
ω	1.976593e-07
α	0.004013

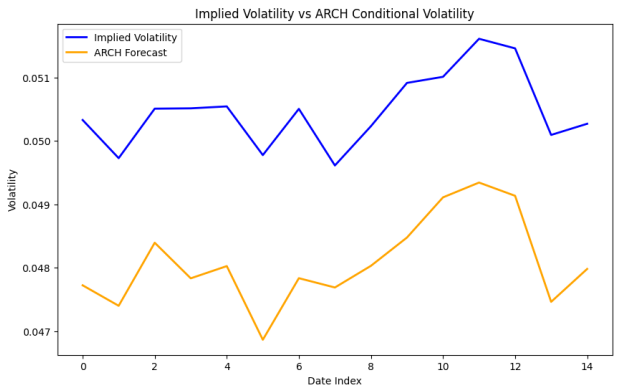
Sample size 500

	MSE
μ	2.3379e-06
ω	2.2478e-07
α	0.002387

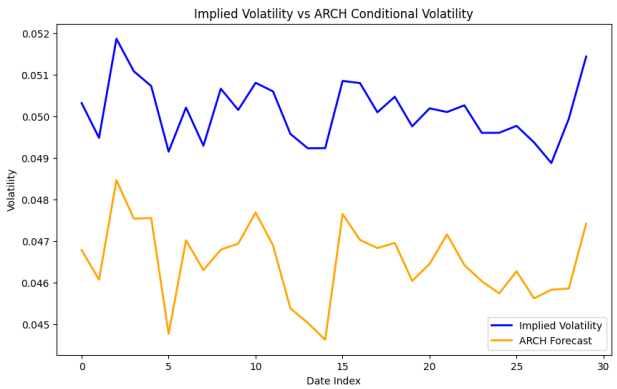
Sample size 1000

Table 4.2: The tables represent the mean squared error (MSE) for the models.

From the results of parameter estimates and (MSE), it can be concluded that the performance of the ARCH simulations was good.



(a) Sample size 500

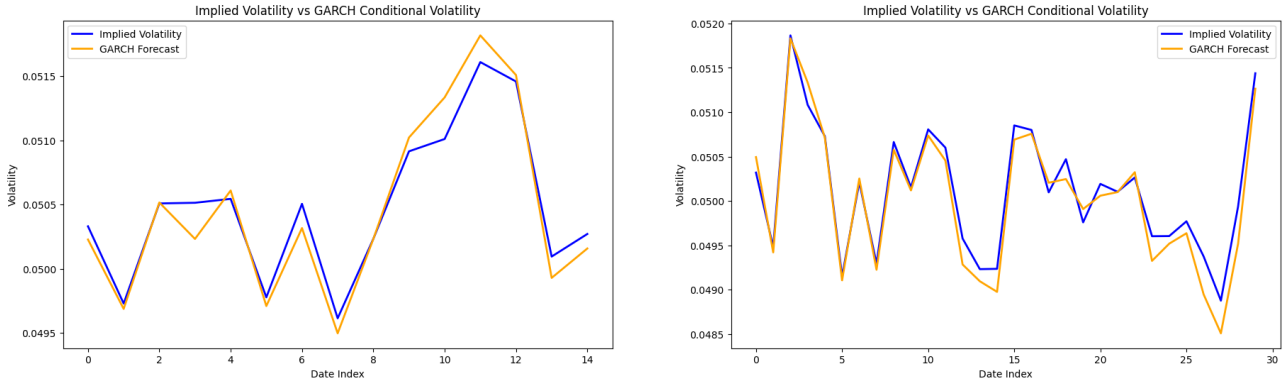


(b) Sample size 1000

Figure 4.6: The curves represent the forecast results of the $ARCH(1)$ model

	RMSE
$ARCH(1)$ 500	0.002402
$ARCH(1)$ 1000	0.003639

Table 4.3: The table represents the root mean square error (RMSE) for the $ARCH(1)$ model.



(a) Sample size 500 **(b)** Sample size 1000

Figure 4.7: The curves represent the forecast results of the $GARCH(1,1)$ model.

	RMSE
$GARCH(1,1)$ 500	0.0001534
$GARCH(1,1)$ 1000	0.000189

Table 4.4: The table represents the root mean square error (RMSE) for the $GARCH(1,1)$ model.

4.2.3 Effect of Sample Size

We observe that the values increase with the increase in sample size from 500 to 1000 in both models (ARCH and GARCH). This may indicate that increasing the sample size reduces random fluctuations and makes the estimates more stable and less influenced by the specific sample.

4.2.4 Comparison between ARCH and GARCH

Overall, we notice that the values for $GARCH(1,1)$ are much lower than those for $ARCH(1)$ for the same sample size.
This suggests that the GARCH model is more accurate in estimation and less volatile compared to the ARCH model.

The GARCH model accounts for both conditional variance and lagged effects, whereas the ARCH model only considers conditional variance.
This makes GARCH more effective in capturing the volatility dynamics in financial data.

4.2.5 Accuracy and Stability

The smaller values obtained from the $GARCH(1,1)$ model compared to the $ARCH(1)$ model indicate that GARCH provides more stable and accurate volatility estimates, making it

preferable for financial data analysis.

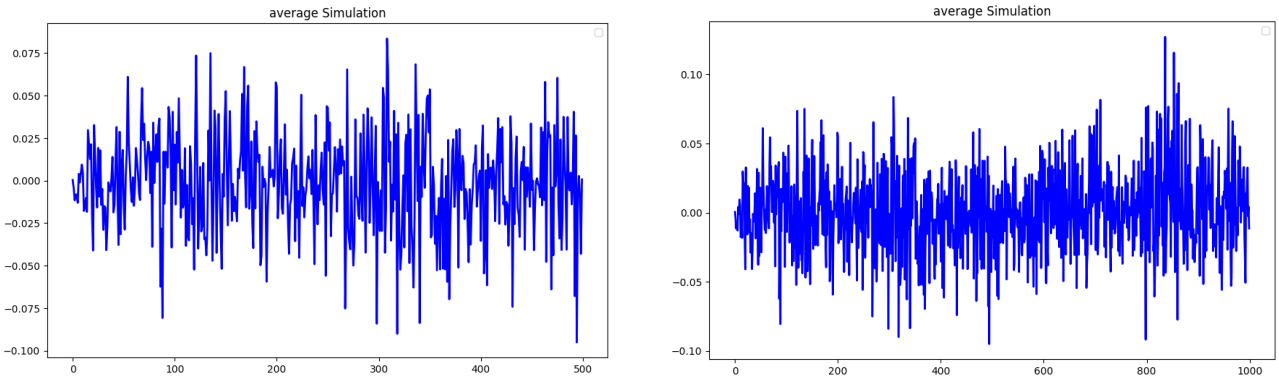
4.3 Overall Conclusion

The $GARCH(1,1)$ model appears to be more effective and stable in estimating volatility compared to the $ARCH(1)$ model. Additionally, increasing the sample size improves the stability of estimates in both models.

Based on these results, it is recommended to use the $GARCH(1,1)$ model for analyzing financial volatility due to its accuracy and ability to capture complex dynamics in financial data.

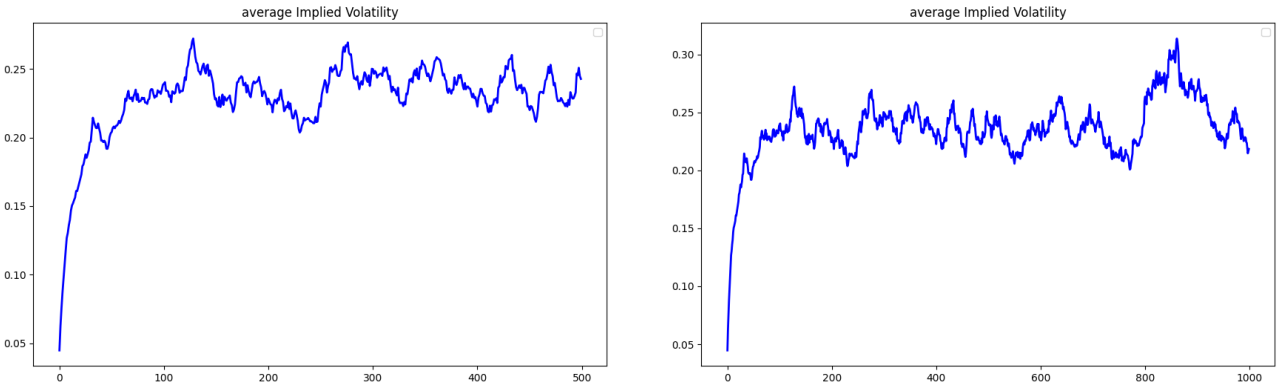
4.3.1 GARCH(1,1)

We will simulate the $GARCH(1,1)$ model using the parameters ($\omega = 0.002, \alpha = 0.234, \beta = 0.74345$).
We will display a curve for the average of daily returns across groups and a curve for the average of implied volatility across $GARCH(1,1)$ groups.
Additionally, we will display the values of the parameters extracted from the groups.



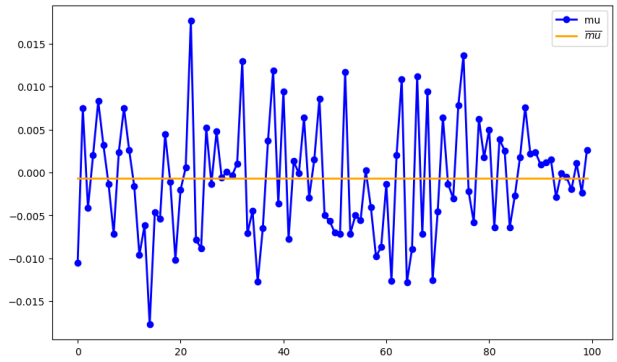
(a) Sample size 500 **(b)** Sample size 1000

Figure 4.8: The curves represent the average of daily returns groups.

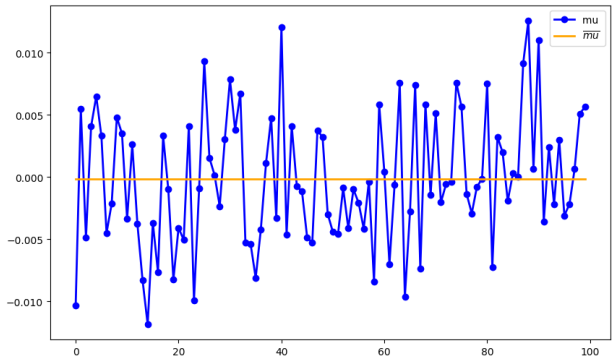


(a) Sample size 500 **(b)** Sample size 1000

Figure 4.9: The curves represent the average implied volatility across $GARCH(1,1)$ groups.

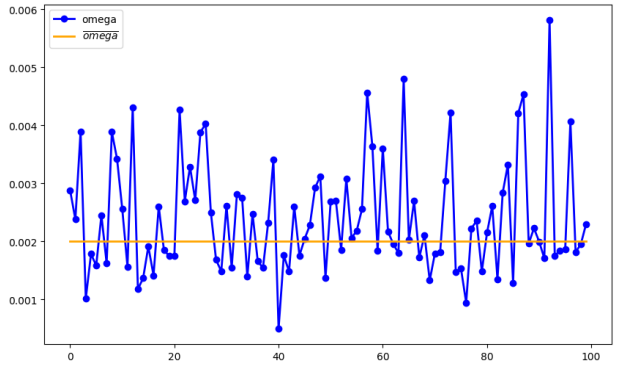


(a) Sample size 500

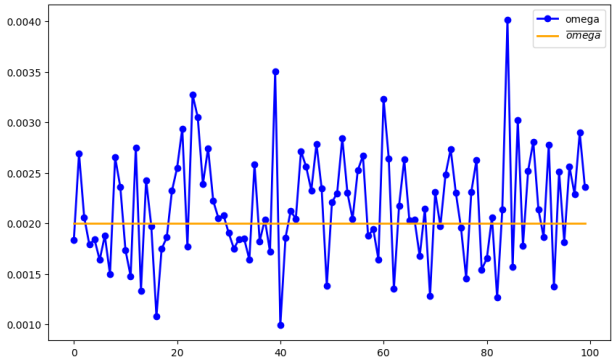


(b) Sample size 1000

Figure 4.10: The curves represent the values of the μ coefficient

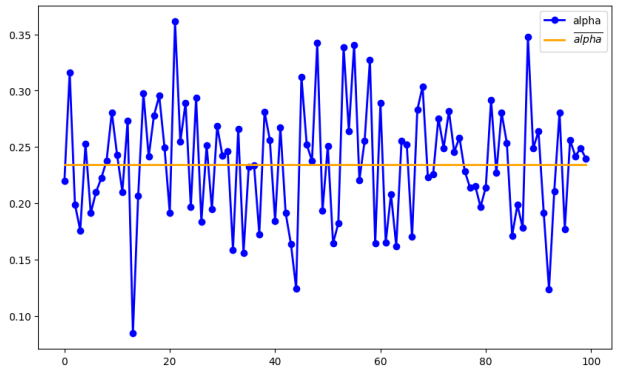


(a) Sample size 500

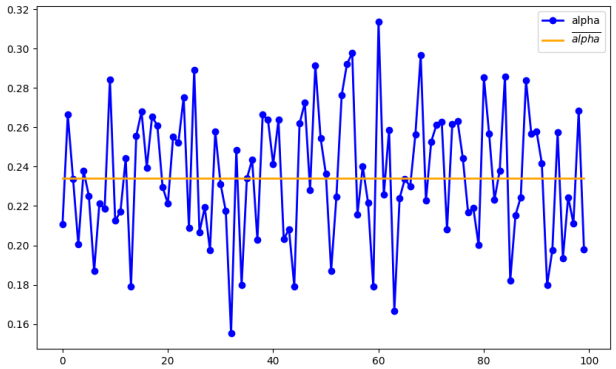


(b) Sample size 1000

Figure 4.11: The curves represent the values of the ω coefficient

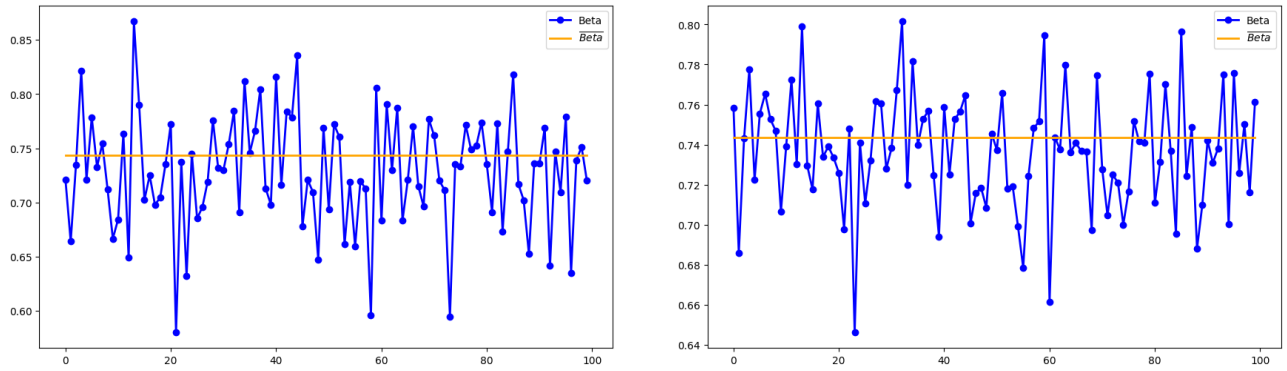


(a) Sample size 500



(b) Sample size 1000

Figure 4.12: The curves represent the values of the α coefficient



(a) Sample size 500 **(b)** Sample size 1000

Figure 4.13: The curves represent the values of the β coefficient

	Average of Parameters
μ	-0.000691
ω	0.0024
α	0.235406
β	0.7288

Sample size 500

	Average of Parameters
μ	-0.000169
ω	0.002165
α	0.23558
β	0.736577

Sample size 1000

Table 4.5: The tables represent the average parameter.

	MSE
μ	4.5313e-05
ω	1.1161e-06
α	0.00273
β	0.002939

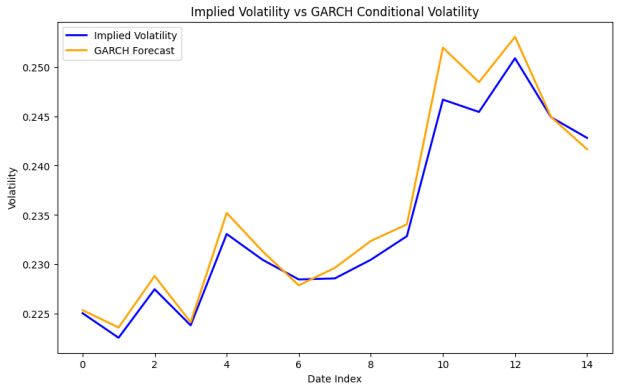
Sample size 500

	MSE
μ	2.7775e-05
ω	3.1719e-07
α	0.001082
β	0.001082

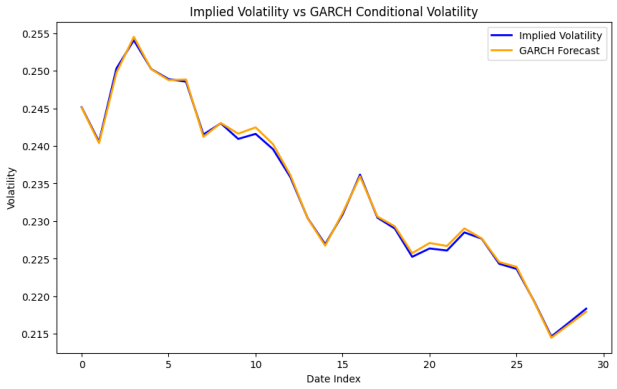
Sample size 1000

Table 4.6: The tables represent the mean squared error (MSE) for the models.

From the results of parameter estimates and (MSE), it can be concluded that the performance of the GARCH simulations was good.



(a) Sample size 500

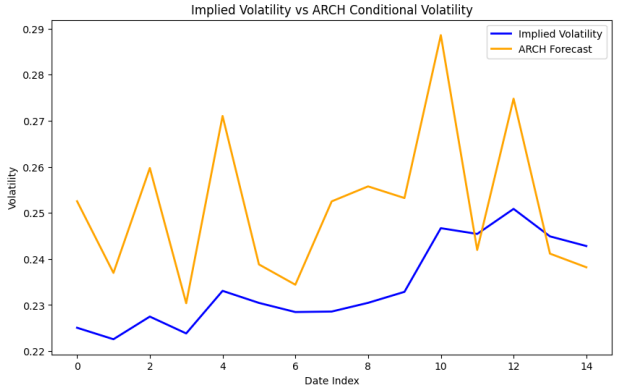


(b) Sample size 1000

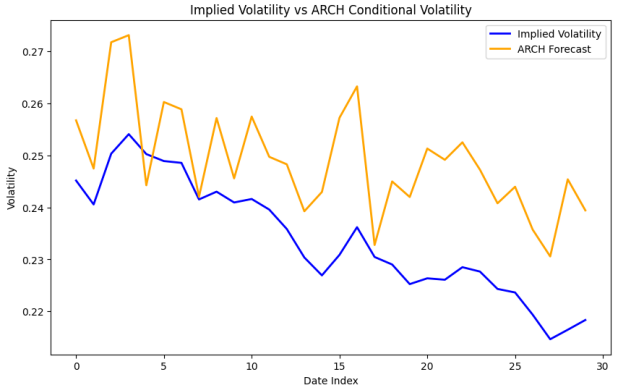
Figure 4.14: The curves represent the forecast results of the $GARCH(1,1)$ model

	RMSE
GARCH(1,1) 500	0.0019611
GARCH(1,1) 1000	0.0003916

Table 4.7: The table represents the root mean square error (RMSE) for the $GARCH(1,1)$ model.



(a) Sample size 500



(b) Sample size 1000

Figure 4.15: The curves represent the forecast results of the $ARCH(1)$ model

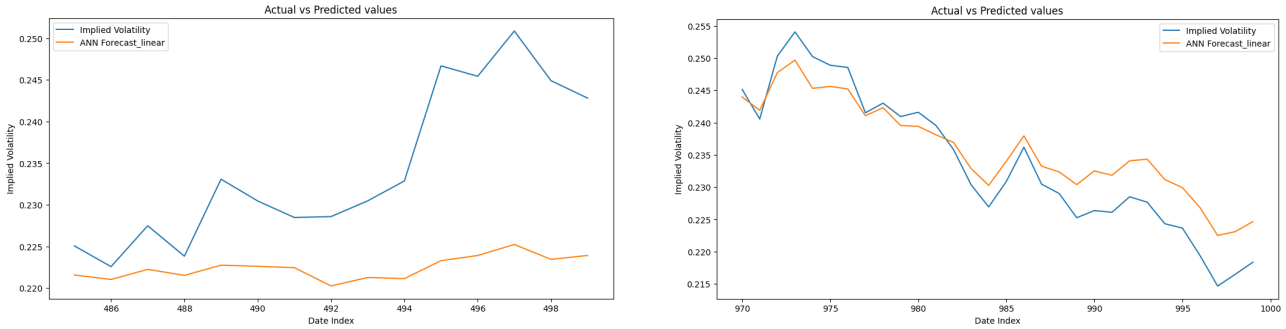
	RMSE
ARCH(1) 500	0.022492
ARCH(1) 1000	0.017214

Table 4.8: The table represents the root mean square error (RMSE) for the $ARCH(1)$ model.

4.4 Artificial Neural Network

We will use simulated data from a $GARCH(1,1)$ model to test an ANN model using two transformation functions: linear and tanh. The test results will be as follows:

1. **Linear transfer function**



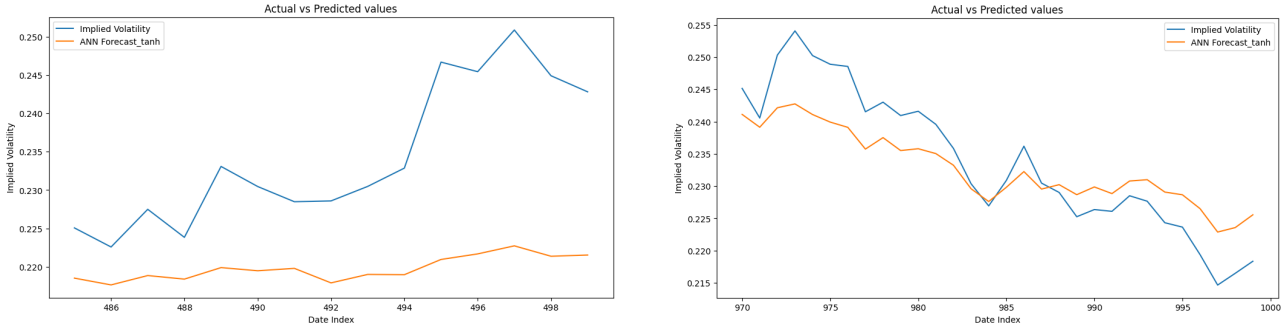
(a) Sample size 500 **(b)** Sample size 1000

Figure 4.16: The curves represent the forecast results of the ANN linear model

	RMSE
ANN linear 500	0.014204
ANN linear 1000	0.004448

Table 4.9: The table represents the root mean square error (RMSE) for the ANN linear model

2. **Tanh transfer function**



(a) Sample size 500 **(b)** Sample size 1000

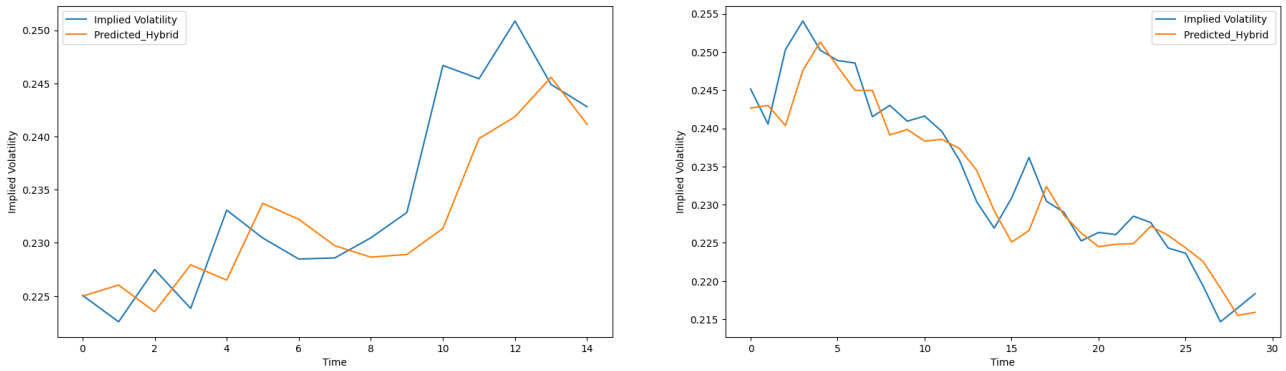
Figure 4.17: The curves represent the forecast results of the ANN tanh model

	RMSE
ANN tanh 500	0.016340
ANN tanh 1000	0.005643

Table 4.10: The table represents the root mean square error (RMSE) for the ANN tanh model

4.5 Hybrid Models

We will use simulated data from a $GARCH(1,1)$ model to test the hybrid model, and the results are as follows:



(a) Sample size 500 **(b)** Sample size 1000

Figure 4.18: The curves represent the forecast results of the hybrid model.

	RMSE
Hybrid 500	0.005675
Hybrid 1000	0.003743

Table 4.11: the table represents the root mean square erro (RMSE) for the hybrid model

4.6 Conclusion

Based on the analysis of the presented results, we conclude that:

- The $GARCH(1,1)$ model shows significant superiority in predicting volatility with increased sample size, making it a reliable model for long-term volatility analysis.
- $The ARCH(1)$ model shows less effective performance compared to the GARCH model, indicating that it may not be the optimal choice for predicting financial volatility.
- Both ANN models (linear and tanh) show significant improvement in performance with increased sample size, indicating their effectiveness in analyzing financial data when large amounts of data are available.
- The Hybrid model provides strong and stable performance across different sample sizes, making it a reliable and effective choice for analyzing financial volatility.

Based on these results, it is recommended to use the $GARCH(1,1)$ model or the Hybrid model to achieve the highest accuracy in predicting financial volatility, while considering increasing the sample size to improve prediction accuracy.

4.7 Real Data

We aim to experiment with models in U.S. equity derivatives, so we have decided to work with the $S\&P500$ index due to the availability of daily market data.
We will attempt to forecast the implied volatility of the $S\&P500$ between 01-01-2015 and 01-03-2022.
This period allows us to investigate volatility models in bullish markets as well as periods of

high volatility and financial crises.

The $S\&P500$, or Standard & Poor's 500, is a stock market index that measures the performance of 500 large companies listed on stock exchanges in the United States. It is widely regarded as one of the best gauges of the overall performance of the U.S. equity market.
The index includes companies from various industries and is used by investors as a benchmark for the U.S. stock market.
Our target data will be the Chicago Board Options Exchange Volatility Index (VIX). "The VIX Index is a calculation designed to produce a measure of the 30-day expected volatility of the U.S. stock market.

" We will import the adjusted closing price of the VIX directly from Yahoo Finance, Represented in the following charts.:

Figure 4.19: The curve represents the closing prices.

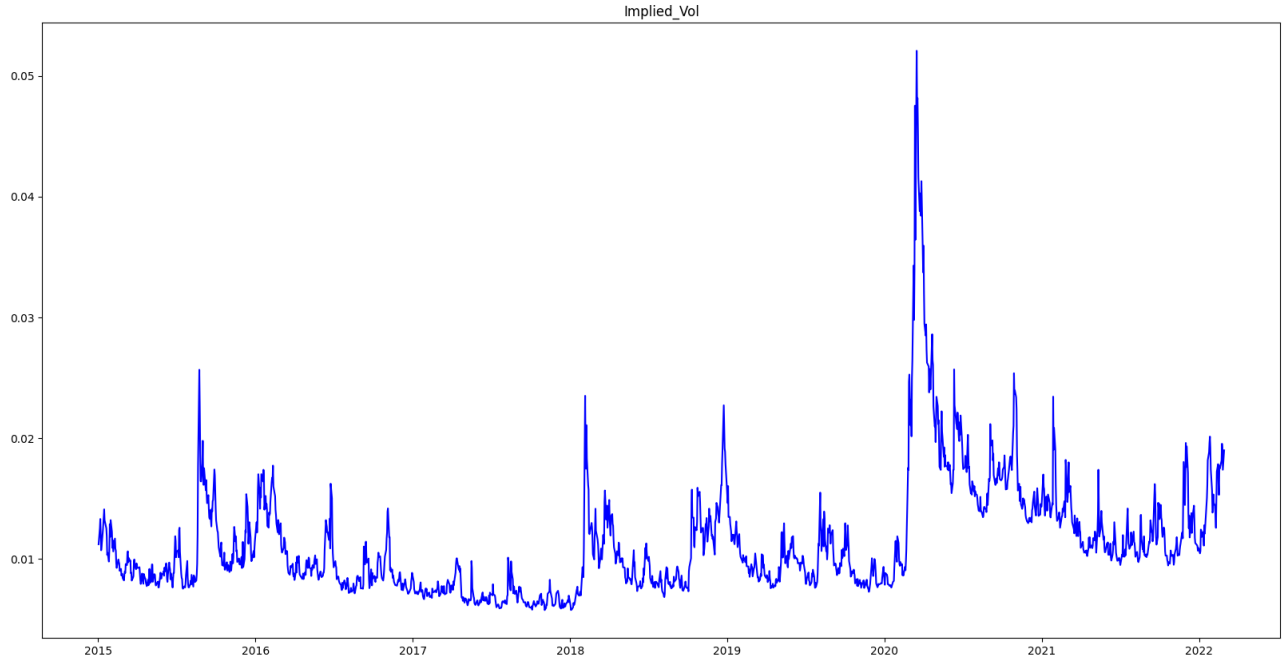


Figure 4.20: The curve of the implied volatility.

4.7.1 GARCH(p,q)

To calibrate the Generalized Autoregressive Conditional Heteroskedasticity (GARCH) model, we calculate the daily returns of closing prices using the following formula:

$$r_t = \frac{P_t - P_{t-1}}{P_{t-1}}.$$

The results will be represented in the following:

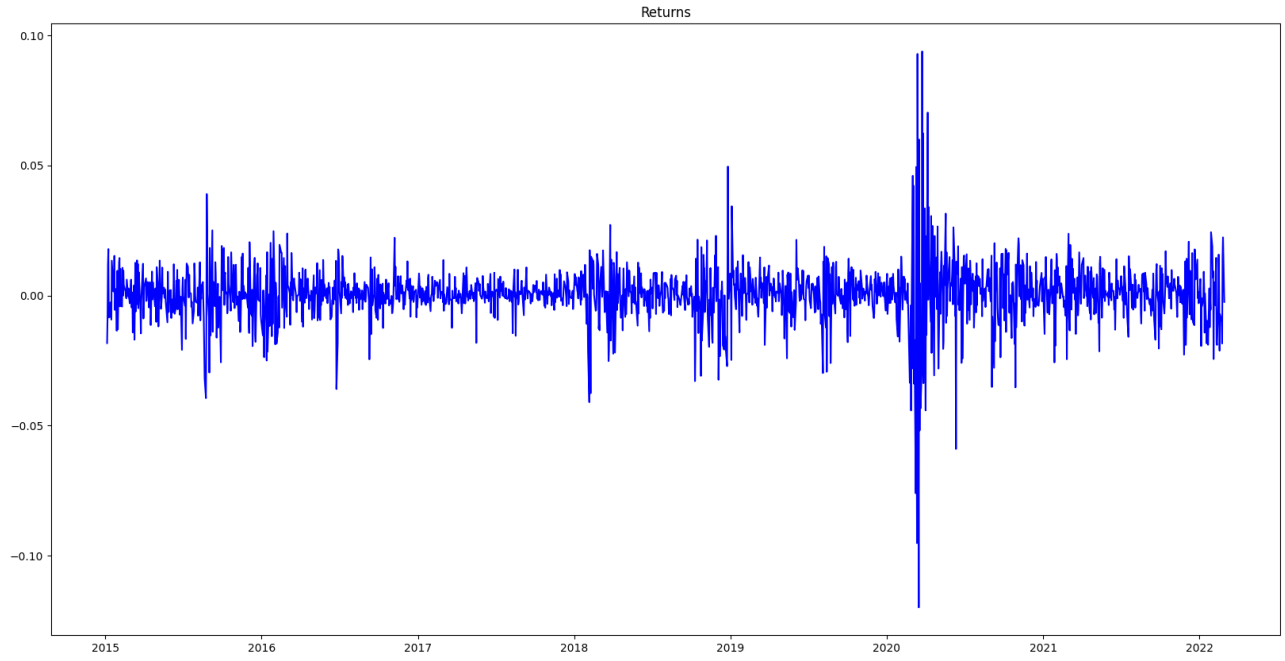


Figure 4.21: The curve of daily returns groups.

We will calculate (AIC), (BIC), and (FBE) to choose the best pair (p, q).

p	q	AIC
1	0	-5990.9366
1	1	-6304.0255
1	2	-6285.9205
1	3	-6271.7718
2	0	-6084.4499
2	1	-6294.8106
2	2	-6283.3808
2	3	-6268.4512
3	0	-6090.4102
3	1	-6279.6925
3	2	-6265.2388
3	3	-6253.2122

Table 4.12: AIC

p	q	BIC
1	0	-5976.3047
1	1	-6284.5164
1	2	-6261.5341
1	3	-6242.5081
2	0	-6064.9407
2	1	-6270.4241
2	2	-6254.1170
2	3	-6234.3102
3	0	-6066.0238
3	1	-6250.4287
3	2	-6231.0977
3	3	-6214.1938

Table 4.13: BIC

p	q	FPE
1	0	0.7012
1	1	0.8764
1	2	1.0523
1	3	1.2292
2	0	0.8763
2	1	1.0525
2	2	1.2290
2	3	1.4060
3	0	1.0523
3	1	1.2291
3	2	1.4059
3	3	1.5829

Table 4.14: FPE

From this, we conclude that the best pair is $GARCH(1,1)$.

The $GARCH(1,1)$ coefficients will be as follows:

Dep. Variable:	Close	**R-squared:**	0.000
Mean Model:	Constant Mean	**Adj. R-squared:**	0.000
Vol Model:	GARCH	**Log-Likelihood:**	3239.31
Distribution:	Normal	**AIC:**	-6470.63
Method:	Maximum Likelihood	**BIC:**	-6451.00
		No. Observations:	1000
		Df Residuals:	999
		Df Model:	1

	coef	**std err**	**t**	**P> |t|**	**95.0% Conf. Int.**
mu	9.7222e-04	1.222e-05	79.577	0.000	[9.483e-04,9.962e-04]

	coef	**std err**	**t**	**P> |t|**	**95.0% Conf. Int.**
omega	3.6012e-06	1.155e-12	3.117e+06	0.000	[3.601e-06,3.601e-06]
alpha[1]	0.2000	4.112e-02	4.864	1.152e-06	[0.119, 0.281]
beta[1]	0.7800	3.094e-02	25.208	3.285e-140	[0.719, 0.841]

The table represents the parameter values for the GARCH(1,1) model.

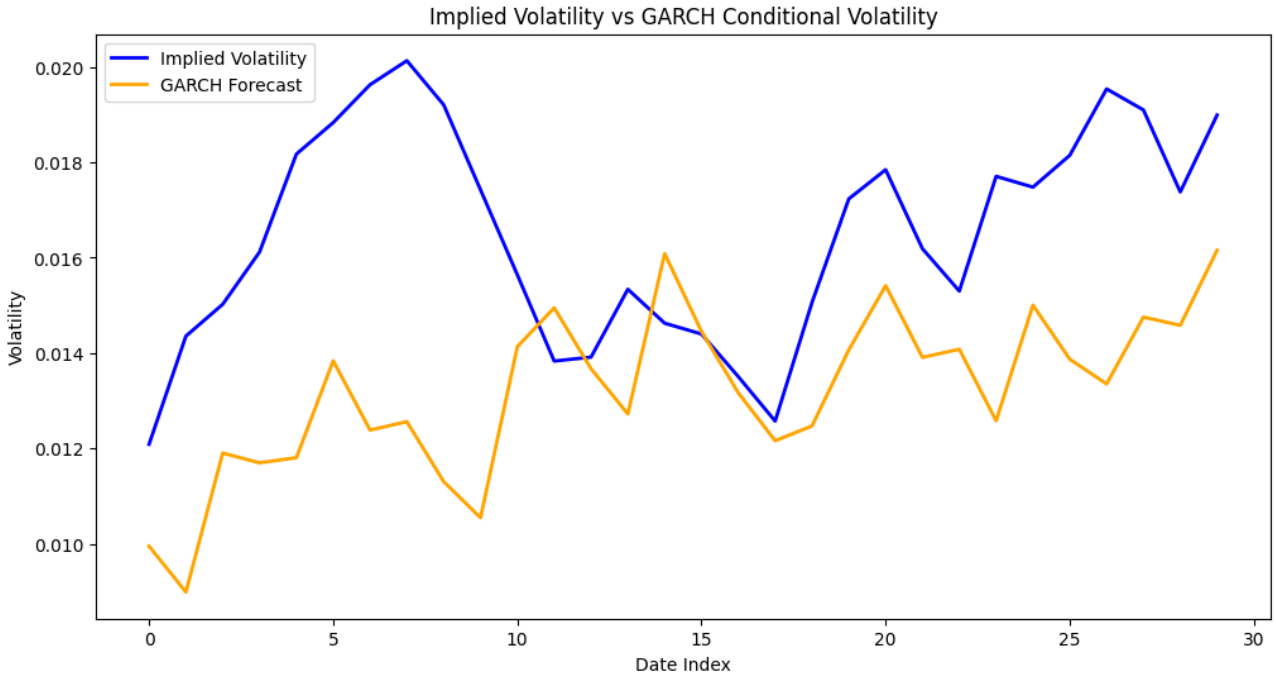


Figure 4.22: The plot represents the forecast results of the $GARCH(1,1)$ model.

The Root Mean Squared Error for the GARCH model is: 0.004137

4.7.2 Artificial Neural Network (ANN) model

1. **Linear transfer function**

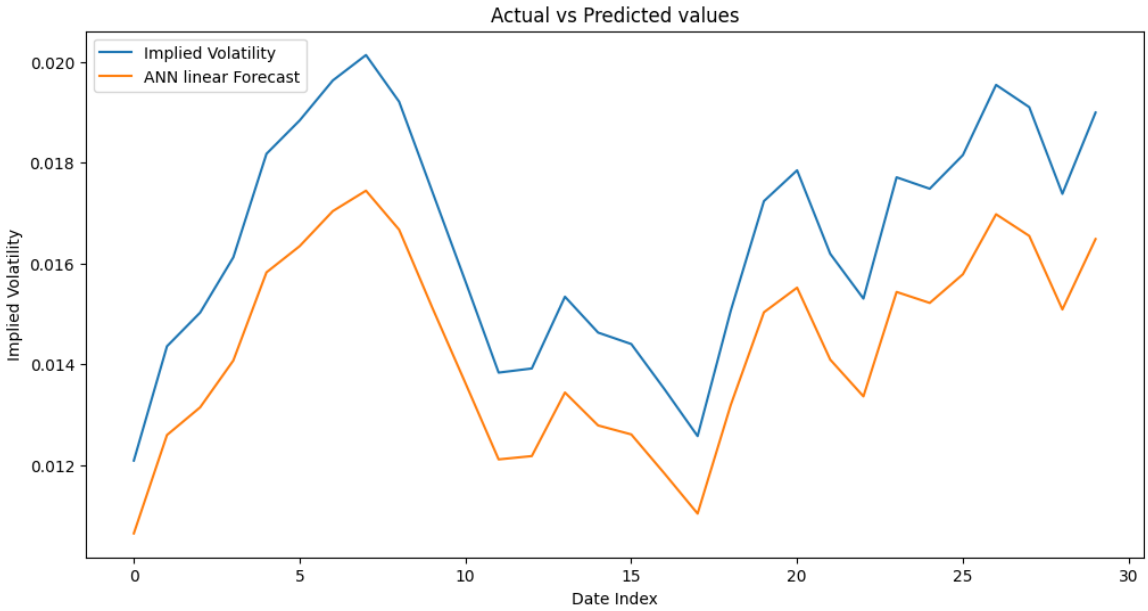


Figure 4.23: The plot represents the forecast results of the ANN linear model

The Root Mean Squared Error for the Neural Network model is: 0.002147

2. **Tanh transfer function**

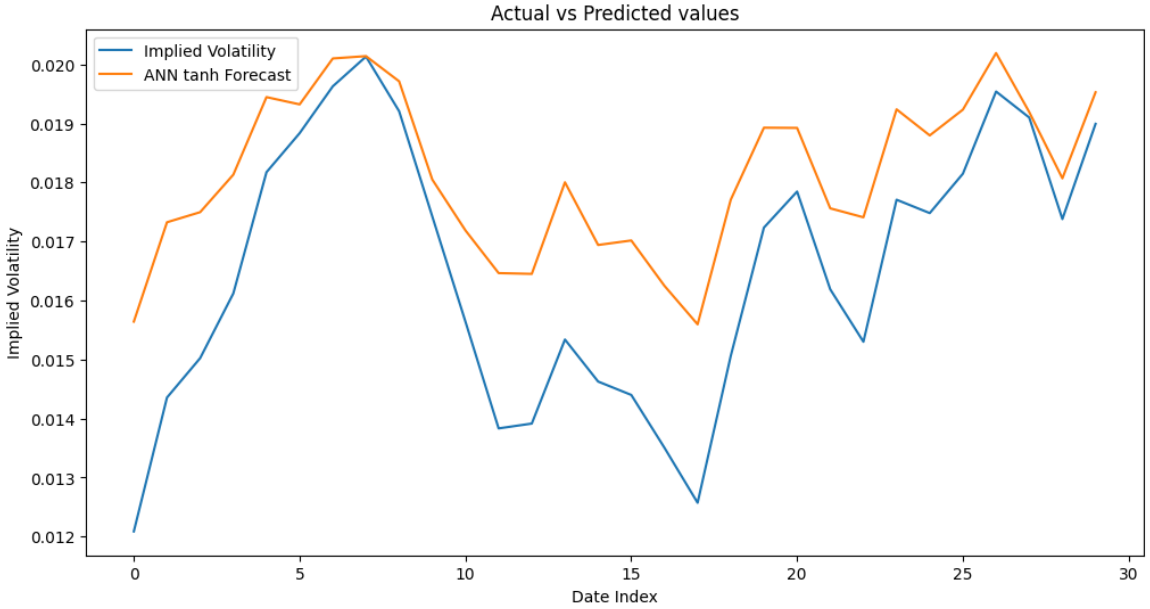


Figure 4.24: The plot represents the forecast results of the ANN model with tanh as a transfer function

The Root Mean Squared Error for the Neural Network model is: 0.0019135

4.7.3 Hybrid models

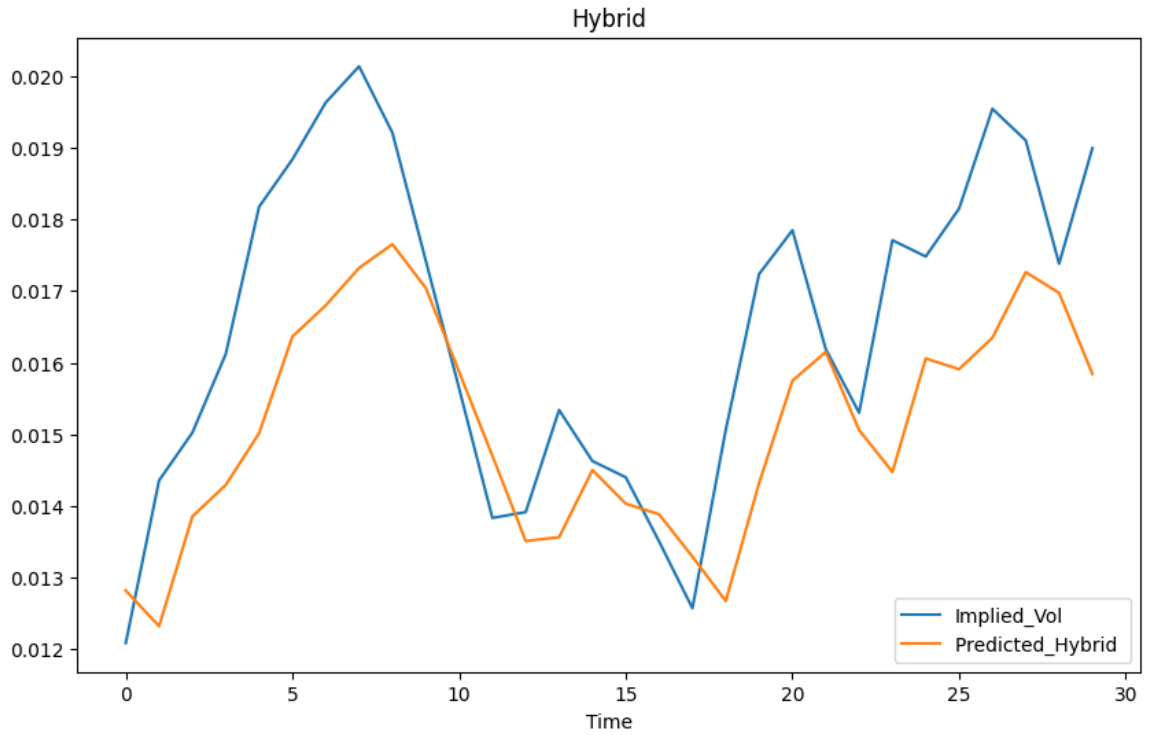


Figure 4.25: The plot represents the forecast results of the hybrid model

The Root Mean Squared Error for the hybrid is 0.001903

4.8 Results

4.8.1 Results and Analysis

We report the out-of-sample performance metrics for each of the models covered. The performance metrics will be root mean squared error (RMSE)

	RMSE
GARCH(1,1)	0.004137
ANN linear	0.002147
ANN tanh	0.001913
Hyprid	0.001903

Table 4.15: The table represents the root mean square error (RMSE) for each model.

We can draw the following conclusions:

1. **Performance of the $GARCH(1,1)$ Model**
The $GARCH(1,1)$ model has the highest root mean squared error (RMSE) among all the models studied, with an (RMSE) of approximately 0.004137. This indicates that its performance is the worst compared to the other models.

2. **Performance of the ANN Linear Model**
The artificial neural network linear (ANN Linear) model achieved a significant improvement compared to the $GARCH(1,1)$ model, with an (RMSE) of approximately 0.002147. This indicates that the (ANN Linear) model performs significantly better than the GARCH model.

3. **Performance of the ANN Tanh Model**
The model using the Tanh activation function (ANN Tanh) achieved a better (RMSE) than the (ANN Linear) model, with an (RMSE) of approximately 0.001913. This indicates a further improvement in performance.

4. **Performance of the Hybrid Model**
The hybrid model (Hybrid) demonstrated the best performance among all the models, achieving the lowest (RMSE) of approximately 0.0019034. This indicates that the hybrid model is the most accurate in prediction among all the tested models.

4.9 Overall Conclusion

It is evident that the models based on artificial neural networks significantly outperform the $GARCH(1,1)$ model in predictive performance, as reflected in the lower (RMSE) values. The hybrid model is the best among the tested models, suggesting that combining multiple components into a single model can significantly improve predictive accuracy.

General Conclusion

In this Master's thesis, volatility prediction models were studied using three main types of models: the GARCH model, Artificial Neural Networks (ANN), and hybrid models that combine both. The main objective was to analyze the performance of each model and evaluate its predictive accuracy compared to the others.

We applied the $GARCH(1,1)$ model to assess its performance in predicting financial volatility, comparing its performance with other models to evaluate its predictive accuracy.

Next, we evaluated artificial neural networks using two types of ANN models: ANN Linear and ANN Tanh. The extent of improvement in predictive accuracy using these models was determined in comparison to the GARCH model.

One of the most significant contributions of this thesis is the introduction of the hybrid model that combines components of the GARCH model with neural algorithms. The performance of the hybrid model was evaluated to determine the extent of improvement it offers in predicting financial volatility.

The study demonstrated that artificial neural network (ANN) models significantly outperform the GARCH model in terms of predictive accuracy. Furthermore, the hybrid model combining GARCH and ANN showed the best performance in predicting financial volatility, indicating that integrating these models can substantially improve prediction accuracy.

The study recommends using hybrid and advanced models to enhance the accuracy of financial volatility predictions, which can greatly assist in financial and strategic decision-making in financial markets.

This study contributes to a deeper understanding of how advanced and hybrid models can be utilized to improve financial forecasts, thereby enhancing the ability of investors and analysts to make more accurate decisions.

Overall, with these considerations in mind, the GARCH-ANN hybrid model can be a powerful tool for financial forecasting, providing more accurate and robust predictions than traditional models alone.

Bibliographic References

[1] Engle, R. F. (1982). Autoregressive Conditional Heteroskedasticity with Estimates of the Variance of United Kingdom Inflation. Econometrica, 50(4), 987-1007.

[2] Bollerslev, T. (1986). Generalized Autoregressive Conditional Heteroskedasticity. Journal of Econometrics, 31(3), 307-327.

[3] Drost, F.C., and Nijman, T.E. (1993). Temporal aggregation of GARCH processes. Econometrica, 61(4), 909-927.

[4] Hull, J. (2012). Options, Futures, and Other Derivatives. Prentice Hall.

[5] Glosten, L. R., Jagannathan, R., and Runkle, D. E. (1993). On the relation between the expected value and the volatility of the nominal excess return on stocks. The Journal of Finance, 48(5), 1779-1801.

[6] Nelson, D. B. (1991). Conditional heteroskedasticity in asset returns: A new approach. Econometrica, 59(2), 347-370.

[7] Taylor, S. J. (1986). Modeling Financial Time Series. John Wiley and Sons.

[8] Zakoian, J. M. (1994). Threshold heteroskedastic models. Journal of Economic Dynamics and Control, 18(5), 931-955.

[9] Engle, R. F. (2001). GARCH 101: The Use of ARCH/GARCH Models in Applied Econometrics. Journal of Economic Perspectives, 15(4), 157-168.

[10] Haykin, S. (2009). Neural Networks and Learning Machines (3rd ed.). Prentice Hall.

[11] Kandel, E. R., Schwartz, J. H., and Jessell, T. M. (2000). Principles of Neural Science (4th ed.). McGraw-Hill.

[12] Dayan, P., and Abbott, L. F. (2001). Theoretical Neuroscience: Computational and Mathematical Modeling of Neural Systems. MIT Press.

[13] Nielsen, M. A. (2015). Neural Networks and Deep Learning. Determination Press.

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

[15] Bishop, C. M. (2006). Pattern Recognition and Machine Learning. Springer.

[16] Aggarwal, C. C. (2018). Neural Networks and Deep Learning: A Textbook. Springer.

[17] Nakamoto, S. Bitcoin: A Peer-to-Peer Electronic Cash System; ReadLiberty.Org: Chicago, IL, USA, 2008.

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

[19] Stavroyiannis, S.; Babalos, V.; Bekiros, S.; Lahmiri, S.; Uddin, G.S. The high frequency multifractal properties of Bitcoin. Physica A 2019, 520, 62–71

[20] Heston, S.L. (1993), A closed-form solution for options with stochastic volatility with applications to bond and currency options. The Review of Financial Studies, 6(2), 327-343.

[21] Sahin, S., Tolun, M.R., Hassanpour, R. (2012), Hybrid expert systems: A survey of current approaches and applications. Expert Systems with Applications, 39(4), 4609-4617.

[22] Cook, S. (2012), An historical perspective on the forecasting performance of the treasury model: Forecasting the growth in UK consumers' expenditure. Applied Economics, 44(5), 555-563.

[23] Chong, Y.Y., Hendry, D.F. (1986), Econometric evaluation of linear macro-economic models. The Review of Economic Studies, 53(4), 671-690.

[24] Fair, R.C., Shiller, R.J. (1989), The informational content of ex ante forecasts. The Review of Economics and Statistics, 71(2), 325-331.

Appendix

Code for simulating a GARCH(1,1) model

```
import pandas as pd
from random import gauss
from random import seed
from arch import arch_model
import numpy as np
import matplotlib.pyplot as plt

# Seed the random number generator for reproducibility
# (you can uncomment the following line to use it)
# seed(0)

# Define GARCH(1, 1) process parameters
a0 = 0.002  # Constant term
a1 = 0.234  # Coefficient for lagged squared residuals
b1 = 0.74345  # Coefficient for lagged conditional variance
mu1=0.003
# Number of time points in the simulation
n = 1001
# Number of simulations
g = 100


# DataFrames to store simulated volatilities and GARCH parameters
vol = pd.DataFrame(index=range(n), columns=range(g))
garch_parameters = pd.DataFrame(index=t, columns=['mu', 'omega', 'alpha', '
    beta'])
sim_df = pd.DataFrame(index=range(n), columns=range(g))

# Loop over the number of simulations
for j in range(g):
    # Set the random seed for each simulation for reproducibility
    np.random.seed(j)
    # Generate random normal values
    w = np.random.normal(size=n)
    # Initialize arrays for residuals (eps) and conditional variances (sigsq
   )
    eps = np.zeros_like(w)
    sigsq = np.zeros_like(w)

    # Simulate the GARCH(1, 1) process
    for i in range(1, n):
        sigma[i] = a0 + a1 * (eps[i-1]**2) + b1 * sigsq[i-1]  # Update
   conditional variance
        eps[i] = w[i] * np.sqrt(sigsq[i]) + mu1 # Update residuals
```

```
    # Store the simulated volatilities
    vol[j] = np.sqrt(sigma)
    # Store the simulated residuals
    sim_df[j] = eps

    # Fit a GARCH(1, 1) model to the simulated data
    model = arch_model(eps)
    model_fit = model.fit(disp='off')
    model_fit_params = model_fit.params

    # Extract the estimated parameters
    mu = model_fit_params['mu']
    omega = model_fit_params['omega']
    alpha = model_fit_params['alpha[1]']
    beta = model_fit_params['beta[1]']

    # Store the estimated parameters in the DataFrame
    garch_parameters.loc[j] = [mu, omega, alpha, beta]

# Remove the first row (initial values) and reset index
vol = vol.iloc[1:].reset_index(drop=True)
sim_df = sim_df.iloc[1:].reset_index(drop=True)

# Print the estimated GARCH parameters for each simulation
print(garch_parameters)

# Calculate the mean of each column in garch_parameters
column_means = garch_parameters.mean()
print(column_means)

# Function to calculate Mean Squared Error (MSE)
def calculate_mse(column, custom_mean_value):
    mse = ((column - custom_mean_value) ** 2).mean()
    return mse

# DataFrame containing GARCH parameters
df = garch_parameters

# Custom mean values for MSE calculation
custom_mean_values = {'mu': column_means[0], 'omega': a0, 'alpha': a1, 'beta
    ': b1}

# Calculate MSE for each column in the DataFrame
mse_values = {column: calculate_mse(df[column], custom_mean_values[column])
    for column in df.columns}
print(mse_values)

# Plotting mu values
y1 = garch_parameters['mu']
y2_value = column_means[0]
y2 = [y2_value] * len(y1)
plt.figure(figsize=(10, 6))
plt.plot(range(len(y1)), y1, label='mu', color='blue', linestyle='-',
    linewidth=2, marker='o')
plt.plot(range(len(y1)), y2, label=r'$\overline{mu}$', color='orange',
    linestyle='-', linewidth=2)
plt.legend()
plt.show()
```

```
# Plotting omega values
y1 = garch_parameters['omega']
y2_value = a0
y2 = [y2_value] * len(y1)
plt.figure(figsize=(10, 6))
plt.plot(range(len(y1)), y1, label='omega', color='blue', linestyle='-',
    linewidth=2, marker='o')
plt.plot(range(len(y1)), y2, label=r'$\overline{omega}$', color='orange',
    linestyle='-', linewidth=2)
plt.legend()
plt.show()

# Plotting alpha values
y1 = garch_parameters['alpha']
y2_value = a1
y2 = [y2_value] * len(y1)
plt.figure(figsize=(10, 6))
plt.plot(range(len(y1)), y1, label='alpha', color='blue', linestyle='-',
    linewidth=2, marker='o')
plt.plot(range(len(y1)), y2, label=r'$\overline{alpha}$', color='orange',
    linestyle='-', linewidth=2)
plt.legend()
plt.show()

# Plotting beta values
y1 = garch_parameters['beta']
y2_value = b1
y2 = [y2_value] * len(y1)
plt.figure(figsize=(10, 6))
plt.plot(range(len(y1)), y1, label='beta', color='blue', linestyle='-',
    linewidth=2, marker='o')
plt.plot(range(len(y1)), y2, label=r'$\overline{beta}$', color='orange',
    linestyle='-', linewidth=2)
plt.legend()
plt.show()

# Plotting the average of simulated data
y1 = sim_df.mean(axis=1)
plt.figure(figsize=(10, 6))
plt.plot(range(len(y1)), y1, color='blue', linestyle='-', linewidth=2)
plt.title('Average Simulation')
plt.legend()
plt.show()
# Assuming vol is a DataFrame containing implied volatilities
y1 = vol.mean(axis=1)

plt.figure(figsize=(10, 6))
plt.plot(range(len(y1)), y1, color='blue', linestyle='-', linewidth=2)
plt.title('Average Implied Volatility')
plt.xlabel('Time')
plt.ylabel('Implied Volatility')
plt.legend(['Average'])
plt.grid(True)
plt.show()

returns = sim_df
Implied_Vol = vol

# Splitting data into train and test sets
split_index = int(0.97 * len(returns))
```

```
returns_train = returns.iloc[:split_index]
returns_test = returns.iloc[split_index:]

Implied_Vol_train = Implied_Vol[:round(0.97 * len(returns))]
Implied_Vol_test = Implied_Vol[round(0.97 * len(returns)):]

# DataFrame to store GARCH forecasted volatilities
garch_forecasted_volatilities = pd.DataFrame(index=returns_test.index)

# Loop over columns (simulations)
for column in returns.columns:
    series = returns[column]
    forecasted_volatilities = []

    # Loop over test set
    for i in range(len(returns_test)):
        train = series[:-(len(returns_test) - i)]
        model = arch_model(train, p=1, q=1)
        model_fit = model.fit(disp='off')
        pred = model_fit.forecast(horizon=1)
        forecasted_volatilities.append(np.sqrt(pred.variance.values
   [-1,:][0]))

    # Store forecasted volatilities in DataFrame
    garch_forecasted_volatilities[column] = forecasted_volatilities

# Calculate average implied volatility
average_implied_volatility = Implied_Vol_test.mean(axis=1)

# Calculate average forecasted volatility
average_forecasted_volatility = garch_forecasted_volatilities.mean(axis=1)

# Calculate GARCH MSE and RMSE
GARCH_MSE = np.mean((average_implied_volatility -
   average_forecasted_volatility) ** 2)
GARCH_RMSE = np.sqrt(GARCH_MSE)

# Print or use GARCH_MSE and GARCH_RMSE as needed
print(f"GARCH MSE: {GARCH_MSE}")
print(f"GARCH RMSE: {GARCH_RMSE}")
# Plotting graph to compare actual IV against GARCH conditional volatilities
    for test set
x = range(len(average_forecasted_volatility1))
y1 = average_implied_volatility
y2 = average_forecasted_volatility1

plt.figure(figsize=(10, 6))
plt.plot(x, y1, label='Implied Volatility', color='blue', linestyle='-',
   linewidth=2)
plt.plot(x, y2, label='GARCH Forecast', color='orange', linestyle='-',
   linewidth=2)
plt.xlabel('Date Index')
plt.ylabel('Volatility')
plt.title('Implied Volatility vs GARCH Conditional Volatility')
plt.legend()

plt.show()
```

Artificial Neural Network code

```
# Import necessary libraries and modules
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense
from sklearn.model_selection import train_test_split



# Iterate over columns in returns
for i in returns.columns:
    # Create a DataFrame with relevant data for the current column
    data = {
        'returns': returns[i].squeeze().values,
        'Implied_Vol_LAG1': Implied_Vol[i].values,
        'Implied_Vol_LAG2': Implied_Vol[i].values,
        'Implied_Vol_LAG3': Implied_Vol[i].values,
        'Implied_Vol': Implied_Vol[i].values,
    }
    df = pd.DataFrame(data)

    # Scale input variables using Min-Max scaler
    scaler = MinMaxScaler()
    scaled_data = scaler.fit_transform(df.iloc[:, :-1])
    scaled_df = pd.DataFrame(scaled_data, columns=df.columns[:-1])


    d_1 = 64  # Output dimension
    d_2 = 64  # Output dimension
    K = 1  # Output dimension of the last layer
    N_epochs = 5
    N_batch_size = 64

    # Split data into training and test sets
    X = scaled_df.iloc[:, :].values
    y = df.iloc[:, -1].values
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size
   =0.03, shuffle=False)

    # Reshape data for ANN input tensors
    X_train = X_train.reshape(X_train.shape[0], 1, X_train.shape[1])
    X_test = X_test.reshape(X_test.shape[0], 1, X_test.shape[1])

    # Define the ANN model architecture
    model = Sequential([
        LSTM(d_1, input_shape=(X_train.shape[1], X_train.shape[2]),
   return_sequences=True),
        LSTM(d_2),
        Dense(32, activation='linear'),
        Dense(K, activation='linear')
    ])

    # Compile the model
    model.compile(optimizer='adam', loss='mean_squared_error')
```

```
    # Train the model on the training dataset
    history = model.fit(X_train, y_train, epochs=N_epochs, batch_size=
   N_batch_size, validation_data=(X_test, y_test))

    # Predict implied volatility using the ANN model
    ANN_predictions = model.predict(X_test)
    Ann_forecasted_volatilities_linear[i] = ANN_predictions
    y_test1[i] = y_test

# Calculate average predictions and actual values
ANN_predictions_linear = Ann_forecasted_volatilities_linear.mean(axis=1)
y_test = y_test1.mean(axis=1)

# Calculate RMSE
ANN_mse = np.mean((y_test - ANN_predictions_linear) ** 2)
ANN_rmse = np.sqrt(ANN_mse)
print("The Root Mean Squared Error for the Neural Network model is: ",
    ANN_rmse)

# Plot actual vs predicted values
plt.figure(figsize=(12, 6))
plt.plot(y_test, label='Implied Volatility')
plt.plot(ANN_predictions_linear, label='ANN Forecast_linear')
plt.xlabel('Date Index')
plt.ylabel('Implied Volatility')
plt.title('Actual vs Predicted values')
plt.legend()

plt.show()
```

Hybrid code

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.neural_network import MLPRegressor
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error
from sklearn.impute import SimpleImputer

forecasted_volatilities = pd.DataFrame()
y_test1 = pd.DataFrame()

for i in returns.columns:
    data = {
        'returns': returns[i].squeeze().values,
        'Implied_Vol': Implied_Vol[i].values,
    }
    df = pd.DataFrame(data)

    # Calculate returns to make the data stationary

    # Create sequences for your analysis
    sequence_length = 10
    X, y = [], []

    for j in range(len(df) - sequence_length):
```

```
        X.append(df['Implied_Vol'].iloc[j:j + sequence_length].values)
        y.append(df['Implied_Vol'].iloc[j + sequence_length])

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

    # Split the data into training and testing sets
    train_size = int(0.97 * len(X))

    X_train, X_test = X[:train_size], X[train_size:]
    y_train, y_test = y[:train_size], y[train_size:]

    scaler = MinMaxScaler()
    X_train_shape = X_train.shape
    X_test_shape = X_test.shape
    X_train = scaler.fit_transform(X_train.reshape(-1, X_train.shape[-1])).
reshape(X_train_shape)
    X_test = scaler.transform(X_test.reshape(-1, X_test.shape[-1])).reshape(
X_test_shape)

    # Create an imputer to fill missing values with mean or other strategies
    imputer = SimpleImputer(strategy='mean')
    X_train = imputer.fit_transform(X_train)
    X_test = imputer.transform(X_test)

    # Define a parameter grid for GridSearchCV
    param_grid = {
        'hidden_layer_sizes': [(50, 50), (100, 50), (100, 100)],
        'alpha': [0.0001, 0.001, 0.01, 0.1],
        'activation': ['relu', 'tanh'],
        'learning_rate': ['constant', 'adaptive']
    }

    # Create the MLPRegressor model
    model = MLPRegressor(random_state=42, max_iter=1000)

    # Create GridSearchCV
    grid_search = GridSearchCV(estimator=model, param_grid=param_grid,
scoring='neg_mean_squared_error', cv=3, n_jobs=-1, verbose=2)

    # Fit the model to find the best hyperparameters
    grid_search.fit(X_train.reshape(X_train.shape[0], -1), y_train)

    # Get the best hyperparameters from the grid search
    best_hidden_layer_size = grid_search.best_params_['hidden_layer_sizes']
    best_alpha = grid_search.best_params_['alpha']
    best_activation = grid_search.best_params_['activation']
    best_learning_rate = grid_search.best_params_['learning_rate']

    print(f"Best Hidden Layer Size: {best_hidden_layer_size}, Best Alpha: {
best_alpha}, Best Activation: {best_activation}, Best Learning Rate: {
best_learning_rate}")

    # Train the best model with the selected hyperparameters
    best_model = MLPRegressor(hidden_layer_sizes=best_hidden_layer_size,
alpha=best_alpha, activation=best_activation, learning_rate=
best_learning_rate, random_state=42, max_iter=1000)
    best_model.fit(X_train.reshape(X_train.shape[0], -1), y_train)

    # Make predictions on the test set
    y_pred = best_model.predict(X_test.reshape(X_test.shape[0], -1))
```

```
    forecasted_volatilities[i] = y_pred
    y_test1[i] = y_test

predictions_Hybrid = forecasted_volatilities.mean(axis=1)
y_test = y_test1.mean(axis=1)

# Calculate RMSE on the test set
test_rmse = np.sqrt(mean_squared_error(y_test, predictions_Hybrid))
print(f"Test RMSE: {test_rmse}")

# Plotting actual vs. predicted returns
plt.figure(figsize=(10, 6))
plt.plot(y_test, label='Implied Volatility')
plt.plot(predictions_Hybrid, label='Predicted Hybrid')
plt.xlabel('Time')
plt.ylabel('Implied Volatility')
plt.legend()
plt.grid(True)  # Optionally add grid lines
plt.savefig('hybrid_plot.pdf')  # Save the plot as PDF (or use 'hybrid_plot.
   png' for PNG)
plt.show()
```

Code for retrieving and preprocessing financial data

```
import numpy as np
import pandas as pd
import yfinance as yf
import matplotlib.pyplot as plt
from datetime import datetime

# Define start and end dates
start = datetime(2015, 1, 1)
end = datetime(2022, 3, 1)

# Ticker symbols for S&P 500 and VIX
tckr = '^GSPC'
Implied_Volatility = '^VIX'

# Retrieve historical data for Implied Volatility (VIX)
Implied_Volatility = yf.Ticker(Implied_Volatility)
Implied_Volatility = Implied_Volatility.history(start=start, end=end,
   interval="1d")

# Retrieve historical data for S&P 500 (^GSPC)
ticker = yf.Ticker(tckr)
ticker_historical = ticker.history(start=start, end=end, interval="1d")

# Plotting S&P 500 closing prices
plt.figure(figsize=(15, 7))
plt.plot(ticker_historical.Close)
plt.title('S&P 500 Closing Prices')
plt.xlabel('Date')
plt.ylabel('Price')
plt.grid(True)
plt.show()

# Calculate daily percentage returns of S&P 500
returns = ticker_historical.Close.pct_change().dropna()
```

```
# Plotting S&P 500 returns
fig, ax1 = plt.subplots(figsize=(20, 10))
ax1.plot(returns, color='blue')
ax1.set_title('S&P 500 Returns')
ax1.set_xlabel('Date')
ax1.set_ylabel('Returns')
plt.grid(True)
plt.show()

# Calculate annualized Implied Volatility from VIX
Implied_Vol = Implied_Volatility.Close / (100 * np.sqrt(252))

# Plotting Implied Volatility (VIX)
fig, ax1 = plt.subplots(figsize=(20, 10))
ax1.plot(Implied_Vol, color='blue')
ax1.set_title('Implied Volatility (VIX)')
ax1.set_xlabel('Date')
ax1.set_ylabel('Implied Volatility')
plt.grid(True)
plt.show()

# Splitting data into 95:5 train and test datasets
returns = returns[-1000:]
Implied_Vol = Implied_Vol[-1000:]

returns_train = returns[:round(0.97 * len(returns))]
returns_test = returns[round(0.97 * len(returns)):]

Implied_Vol_train = Implied_Vol[:round(0.97 * len(returns))]
Implied_Vol_test = Implied_Vol[round(0.97 * len(returns)):]
```

GARCH model code

Below is the Python code for estimating a GARCH model, selecting the best model based on BIC, forecasting volatility, and evaluating performance:

```
import numpy as np
import pandas as pd
from arch import arch_model
import matplotlib.pyplot as plt

# Define the parameters for grid search
a = 1
p = 4
b = 0
q = 4

# Function to calculate Final Prediction Error (FPE)
def calculate_fpe(results):
    n_params = len(results.params)
    n_data = len(results.resid)
    sse = (results.resid ** 2).sum()
    fpe = (n_data * (n_params + 1)) / (n_data - n_params - 1) * sse
    return fpe

# Grid search for best GARCH model based on BIC, AIC, and FPE
p_values = range(a, p)
q_values = range(b, q)
```

```
bic_results = []
aic_results = []
fpe_results = []

for p in p_values:
    for q in q_values:
        model = arch_model(returns_train, vol='GARCH', p=p, q=q)
        results = model.fit(disp='off')

        bic = results.bic
        aic = results.aic
        fpe = calculate_fpe(results)

        bic_results.append((p, q, bic))
        aic_results.append((p, q, aic))
        fpe_results.append((p, q, fpe))

# Convert results to DataFrames
bic_df = pd.DataFrame(bic_results, columns=['p', 'q', 'BIC'])
aic_df = pd.DataFrame(aic_results, columns=['p', 'q', 'AIC'])
fpe_df = pd.DataFrame(fpe_results, columns=['p', 'q', 'FPE'])

# Print and find the best model based on BIC
print("BIC:")
print(bic_df)
best_model = bic_df.loc[bic_df['BIC'].idxmin()]
best_model_p = int(best_model['p'])
best_model_q = int(best_model['q'])
print("Best GARCH Model (based on BIC):")
print(best_model)

# Fit the best GARCH model
best_model_fit = arch_model(returns, vol='GARCH', p=best_model_p, q=
    best_model_q).fit(disp='off')

# Forecast volatility using the best GARCH model
garch_forecasted_volatilities = []
for i in range(len(returns_test)):
    train = returns.iloc[:-(len(returns_test)-i)]
    model = arch_model(train, p=best_model_p, q=best_model_q)
    model_fit = model.fit(disp='off')
    pred = model_fit.forecast(horizon=1)
    garch_forecasted_volatilities.append(np.sqrt(pred.variance.values[-1,
    :][0]))

# Calculate RMSE for GARCH model
GARCH_MSE = np.mean((Implied_Vol_test - garch_forecasted_volatilities) ** 2)
GARCH_RMSE = np.sqrt(GARCH_MSE)
print("The Root Mean Squared Error for the GARCH model is: ", GARCH_RMSE)

# Plotting graph to compare actual IV against GARCH conditional volatilities
    for test set
x = range(len(Implied_Vol_test))
y1 = Implied_Vol_test
y2 = garch_forecasted_volatilities

plt.figure(figsize=(12, 6))
plt.plot(x, y1, label='Implied Volatility', color='blue', linestyle='-',
    linewidth=2)
```

```
plt.plot(x, y2, label='GARCH Forecast', color='orange', linestyle='-',
    linewidth=2)
plt.xlabel('Date Index')
plt.ylabel('Volatility')
plt.title('Implied Volatility vs GARCH Conditional Volatility')
plt.legend()
plt.grid(True)
plt.show()
```

Artificial Neural Network code

Below is the Python code for training a neural network for forecasting implied volatility:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error

# Define input data and target variables in a dataframe
data = {
    'returns': returns.squeeze().values,
    'Implied_Vol_LAG1': Implied_Vol,  # Assuming Implied_Vol is already
    lagged
    'Implied_Vol_LAG2': Implied_Vol   # You can add more lagged variables if
    needed
}
df = pd.DataFrame(data)

# Scale input variables using MinMaxScaler
scaler = MinMaxScaler()
scaled_data = scaler.fit_transform(df.iloc[:, :-1])
scaled_df = pd.DataFrame(scaled_data, columns=df.columns[:-1])


d_1 = 64       # Output dimension
d_2 = 64       # Output dimension
K = 1          # Output dimension of the last layer (assuming one output)
N_epochs = 2   # Number of epochs for training
N_batch_size = 64  # Batch size for training

# Prepare X and y for training and testing
X = scaled_df.iloc[:, :].values
y = df.iloc[:, -1].values
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.03,
    shuffle=False)

# Reshape X into suitable LSTM input tensors
X_train = X_train.reshape(X_train.shape[0], 1, X_train.shape[1])
X_test = X_test.reshape(X_test.shape[0], 1, X_test.shape[1])

# Create a neural network with 2 LSTM layers and 2 dense layers
model = Sequential([
    LSTM(d_1, input_shape=(X_train.shape[1], X_train.shape[2]),
    return_sequences=True),
    LSTM(d_2),
```

```
    Dense(32, activation='linear'),
    Dense(K, activation='linear')
])

# Compile the model
model.compile(optimizer='adam', loss='mean_squared_error')

# Train the model on the training dataset
history = model.fit(X_train, y_train, epochs=N_epochs, batch_size=
    N_batch_size, validation_data=(X_test, y_test))

# Predict implied volatility using the trained ANN model
ANN_linear_predictions = model.predict(X_test)

# Calculate RMSE
ANN_mse = np.mean((y_test - ANN_linear_predictions.flatten()) ** 2)
ANN_linear_rmse = np.sqrt(ANN_mse)
print("The Root Mean Squared Error for the Neural Network model is: ",
    ANN_linear_rmse)

# Plot the true values and predicted values
plt.figure(figsize=(12, 6))
plt.plot(y_test, label='Implied Volatility')
plt.plot(ANN_linear_predictions, label='ANN Linear Forecast')
plt.xlabel('Date Index')
plt.ylabel('Implied Volatility')
plt.title('Actual vs Predicted Values')
plt.legend()
plt.grid(True)
plt.show()
```

Hybrid model Code

Below is the Python code for training a hybrid model for forecasting implied volatility:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.neural_network import MLPRegressor
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error
from sklearn.impute import SimpleImputer

# Define input data and target variables in a dataframe
data = {
    'returns': returns.squeeze().values,
    'Implied_Vol': Implied_Vol.values,
}
df = pd.DataFrame(data)

# Create sequences for analysis (adjust sequence length as needed)
sequence_length = 1
X, y = [], []

for j in range(len(df) - sequence_length):
    X.append(df['Implied_Vol'].iloc[j:j + sequence_length].values)
    y.append(df['Implied_Vol'].iloc[j + sequence_length])
```

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

# Split data into training and testing sets
train_size = int(0.97 * len(X))
X_train, X_test = X[:train_size], X[train_size:]
y_train, y_test = y[:train_size], y[train_size:]

# Scale input variables using MinMaxScaler
scaler = MinMaxScaler()
X_train_shape = X_train.shape
X_test_shape = X_test.shape
X_train = scaler.fit_transform(X_train.reshape(-1, X_train.shape[-1])).
    reshape(X_train_shape)
X_test = scaler.transform(X_test.reshape(-1, X_test.shape[-1])).reshape(
    X_test_shape)

# Fill missing values with mean using SimpleImputer
imputer = SimpleImputer(strategy='mean')
X_train = imputer.fit_transform(X_train)
X_test = imputer.transform(X_test)

# Define parameter grid for GridSearchCV
param_grid = {
    'hidden_layer_sizes': [(50, 50), (100, 50), (100, 100)],
    'alpha': [0.0001, 0.001, 0.01, 0.1],
    'activation': ['relu', 'tanh'],
    'learning_rate': ['constant', 'adaptive']
}

# Create MLPRegressor model
model = MLPRegressor(random_state=2, max_iter=1000)

# Create GridSearchCV
grid_search = GridSearchCV(estimator=model, param_grid=param_grid, scoring='
    neg_mean_squared_error', cv=3, n_jobs=-1, verbose=2)

# Fit model to find best hyperparameters
grid_search.fit(X_train.reshape(X_train.shape[0], -1), y_train)

# Get best hyperparameters from grid search
best_hidden_layer_size = grid_search.best_params_['hidden_layer_sizes']
best_alpha = grid_search.best_params_['alpha']
best_activation = grid_search.best_params_['activation']
best_learning_rate = grid_search.best_params_['learning_rate']

print(f"Best Hidden Layer Size: {best_hidden_layer_size}, Best Alpha: {
    best_alpha}, Best Activation: {best_activation}, Best Learning Rate: {
    best_learning_rate}")

# Train best model with selected hyperparameters
best_model = MLPRegressor(hidden_layer_sizes=best_hidden_layer_size, alpha=
    best_alpha, activation=best_activation, learning_rate=best_learning_rate,
     random_state=2, max_iter=1000)
best_model.fit(X_train.reshape(X_train.shape[0], -1), y_train)

# Make predictions on test set
y_pred = best_model.predict(X_test.reshape(X_test.shape[0], -1))

# Calculate RMSE on test set
test_rmse = np.sqrt(mean_squared_error(y_test, y_pred))
```

```
print(f"Test RMSE: {test_rmse}")

# Plot actual vs. predicted returns
plt.figure(figsize=(10, 6))
plt.plot(y_test, label='Implied Volatility')
plt.plot(y_pred, label='Predicted Hybrid Model')
plt.xlabel('Time')
plt.ylabel('Implied Volatility')
plt.title('Actual vs. Predicted Implied Volatility')
plt.legend()
plt.grid(True)
plt.show()
```