People's Democratic Republic of Algeria
Ministry of Higher Education and Scientific Research

University of Saida – Dr. Moulay Tahar
Faculty of Mathematics, Computer Science,
and Telecommunications
Department of Mathematics

This Thesis was Submitted in Partial Fulfillment of the Requirements the degree of

Academic Master

Discipline: MATHEMATICS
Specialty: Statistical Stochastic Analysis of Processes and Applications

by

ALLALI **H**afsa[1]

Under the supervision of

Dr. ROUANE Rachida

Theme:

Neural Regression Models
Theory and Applications

Defended On June 14, 2026 in front of the committee

Dr. F. MOKHTARI	University of Saida Dr. Moulay Tahar	President
Dr. R. ROUANE	University of Saida Dr. Moulay Tahar	Supervisor
Dr. F. BENZIADI	University of Saida Dr. Moulay Tahar	Examiner

Academic Year: 2025/2026

[1]e-mail: hafsabatouta12@gmail.com

Contents

List of Figures

List of Tables

Abstract

This thesis examines the theoretical foundations and practical applications of two regression approaches: Multiple Linear Regression (MLR) and Neural Network Regression (NNR). The study begins by establishing a robust mathematical framework for MLR, emphasizing the Ordinary Least Squares (OLS) estimation. However, recognizing the limitations of linear models in capturing complex, high-dimensional patterns, the research shifts toward Neural Networks (NNs) as a flexible alternative.

The core of this work investigates the architecture of Neural Networks (NNs) and the mathematical rigor of the Backpropagation algorithm. By situating neural networks within a regression context, we demonstrate their capacity to function as universal approximators for non-linear phenomena. A comparative analysis is conducted to evaluate the predictive performance and stability of both frameworks. The results suggest that while linear regression remains indispensable for interpretability and structural inference, neural regression models provide superior accuracy in handling non-linearities and large datasets, offering a powerful tool for predictive modeling in contemporary data science.

Dedication

To my beloved mother, whose love, prayers, and endless support have always been my greatest strength.

To my dear father, my role model and source of inspiration.

To my beloved siblings and family, for their constant encouragement, love, and support throughout this journey.

This work is dedicated to all of you with love and gratitude.

Acknowledgements

First and foremost, all praise and thanks are to Allah, who granted me patience, strength, and determination and guided me to complete this work.

I would like to express my sincere gratitude to all the teachers and professors who taught and guided me throughout my university journey and contributed to my academic development. My special thanks and deepest appreciation go to my supervisor, Dr. ROUANE Rachida, for her guidance, patience, encouragement, and trust in my abilities. Her advice and support helped me greatly throughout this work and taught me many valuable lessons.

I would also like to express my deepest gratitude to my beloved mother, whose prayers, love, sacrifices, and endless support have always been my greatest source of strength throughout this journey. Without her prayers and support, I would never have reached this stage.

My sincere thanks also go to my dear father, Naimi, who taught me the meaning of responsibility, hard work, and perseverance, and who will always remain my role model and inspiration in life.

To my beloved siblings, my companions from the first step to the last, thank you for your endless moral support, encouragement, and presence in my life: Hakim, Imane, Yasser, Abdelsalam, Israa, Anfel, and Siradj.

To my mother's family, from my beloved grandparents to the youngest member of the family, thank you for your sincere love and support and for always being part of my life.

To the friends who accompanied me throughout this journey, especially Hafsa and Ferial, thank you for your support, encouragement, and all the beautiful moments we shared together.

Finally, my sincere gratitude goes to everyone who taught me even a single letter since the beginning of my educational journey.

General Introduction

Regression analysis occupies a central position in statistics, econometrics, and machine learning. Its origins date back to the early 19th century, with the independent work of Adrien-Marie Legendre (1805) and Carl Friedrich Gauss (1809) on the method of least squares, initially applied to astronomical observations. For more than two centuries, linear regression has remained an essential tool for modeling relationships between variables, thanks to its simplicity and interpretability, as thoroughly presented in the book *Introduction to Linear Regression Analysis* by Montgomery, Peck, and Vining [3].

For many years, the linear regression framework, and more specifically Multiple Linear Regression (MLR), has been the method of choice in many scientific fields, from economics to engineering. The Gauss–Markov theorem, detailed by Kuo in *Econometric Theory*, guarantees that the ordinary least squares estimator is the best linear unbiased estimator (BLUE) under a set of well-defined assumptions. However, the last twenty years have witnessed an explosion in the volume, variety, and velocity of data. The advent of big data, fueled by digitalization, the development of the internet, including social media, and the proliferation of environmental sensors, has profoundly altered the nature of prediction problems, as explained by Hastie, Tibshirani, and Friedman in their reference book *The Elements of Statistical Learning* [9].

In this new context, classical linear models often reach their limits. The assumptions of MLR (linearity, independence of errors, homoscedasticity, and absence of multicollinearity) are frequently violated when dealing with high-dimensional, noisy, or strongly non-linear data. Moreover, MLR requires the user to explicitly specify interactions between variables, which becomes impractical when the number of predictors is large.

In parallel, advances in computational power and the availability of massive datasets have triggered a renaissance of artificial neural networks. What is now called "deep learning" has its roots in the perceptron introduced by Rosenblatt (1958) and the backpropagation algorithm developed by Rumelhart, Hinton and Williams (1986). However, it is really the convergence of three factors that has propelled it to the forefront of modern data science: large annotated datasets, powerful parallel computing, and the development of flexible neural architectures, as described in the book *Deep Learning* by Goodfellow, Bengio and Courville [8].

Deep learning methods, and particularly feedforward neural networks, have achieved remarkable results in fields as diverse as computer vision, natural language processing, speech recog-

nition, and, more recently, forecasting. Unlike linear models, neural networks are "universal approximators": under mild conditions, a network with a single hidden layer can approximate any continuous function on a compact set to arbitrary precision, a result first proved by Cybenko in 1989 in *Mathematics of Control, Signals and Systems* [6]. This theoretical property, combined with their practical effectiveness, explains why neural networks are now systematically considered for complex regression tasks.

Indeed, MLR remains a powerful tool for inference, confidence intervals, and hypothesis testing, but its predictive capacity is limited by its linear nature. By contrast, as emphasized by Hastie et al. in *The Elements of Statistical Learning*, neural networks capture complex interactions automatically, require no distributional assumptions, and scale favorably with large datasets [9].

Consequently, the comparison between these two families of models is not simply a matter of choosing the “best” method; it is a matter of understanding the trade-offs between interpretability, predictive accuracy, and computational cost.

The main objectives of this work are threefold:

1. To recall the theoretical foundations of Multiple Linear Regression (matrix formulation, OLS estimation, and assumptions) and to illustrate its application.
2. To present the mathematical framework of Neural Network Regression (perceptron, feedforward architecture, activation functions, gradient descent, backpropagation, and the universal approximation theorem).
3. To apply both models to a real dataset of photovoltaic and meteorological measurements from Laghouat (Algeria) and to compare their predictive performances using appropriate metrics (R^2, RMSE, MAE).

The thesis is organized into three chapters:

- **Chapter 1** is devoted to Multiple Linear Regression. After introducing the model and its matrix notation, we discuss the underlying assumptions, the ordinary least squares estimator, its properties (unbiasedness, variance, Gauss–Markov theorem), the ANOVA decomposition, and hypothesis testing.
- **Chapter 2** presents the theory of Neural Network Regression. Starting from the perceptron, feedforward networks, the role of activation functions, gradient descent optimization, and the backpropagation algorithm. We also recall the universal approximation theorem and discuss the bias-variance trade-off.
- **Chapter 3** is dedicated to the numerical application. We describe the Laghouat PV dataset (source, variables, preprocessing), implement both models in Python, present the results, and compare the performances of MLR and NNR using graphs and error metrics. A final discussion highlights the advantages and limitations of each approach.

Finally, we have presented a clear comparison between MLR and NNR, explained the mathematical foundations of neural network regression, and highlighted the importance of each model according to its theoretical properties and practical use. This work thus opens up new perspectives for further mathematical studies in the field of regression analysis and deep learning.

Chapter 1

Multiple linear regression

Multiple Linear Regression is one of the most classical and widely used tools in statistical modeling. It extends simple linear regression by allowing the response variable to depend on several explanatory variables, making it especially useful for studying complex relationships in real data. Because of its mathematical simplicity, interpretability, and broad range of applications, MLR remains a fundamental method in statistics, econometrics, engineering, and machine learning. In this chapter, we introduce the basic theory of Multiple Linear Regression, including its model formulation, main assumptions, estimation by ordinary least squares, and essential inferential results.

1.1 Multiple Regression Model

Multiple linear regression, denoted by **MLR**, is a generalization of simple linear regression and remains one of the most important statistical models because of its interpretability and mathematical simplicity. Its goal is to study and model the relationship between an explained variable Y and several explanatory variables. $X_1, X_2, \cdots, X_p$. It is defined as follows:

$$y_i = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_p x_{ip} + \varepsilon_i, \quad \forall i \in \{1, \cdots, n\} \tag{1.1}$$

- $p \leq n$.
- i: $(1, \cdots, n)$ corresponds to the number of observations.
- y_i: Observed value of dependent variable.
- β_0: Intercept constant.
- x_{ij}: Value of the j-th independent variable for individual i.
- β_j: Regression coefficient associated with x_j.
- ε_i: Random error (noise).

1.1.1 Example

The regression model $y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \varepsilon$ represents a plane in the three-dimensional space of y, x_1, and x_2. Figure 1.1 a depicts the regression plane for the model $y = 50 + 10x_1 + 7x_2$. [13]

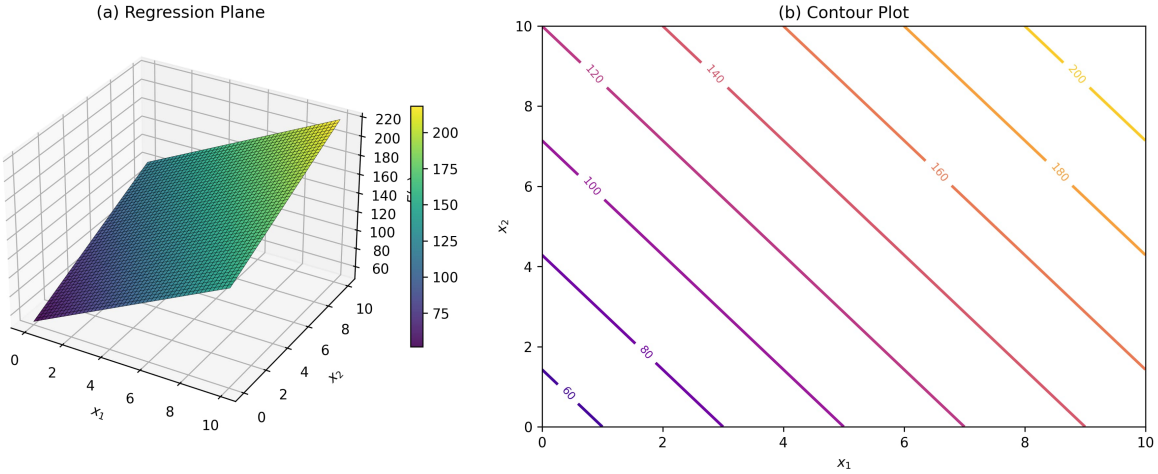

Figure 1.1: (a): The regression plane for the model $y = 50 + 10x_1 + 7x_2$
(b): The contour plot

1.1.2 Matrix notation of MLR

The regression model can also be written in matrix form as ([3]):

$$\begin{cases} y_1 = \beta_0 + \beta_1 x_{11} + \beta_2 x_{12} + \cdots + \beta_p x_{1p} + \varepsilon_1 \\ y_2 = \beta_0 + \beta_1 x_{21} + \beta_2 x_{22} + \cdots + \beta_p x_{2p} + \varepsilon_2 \\ \vdots \\ y_n = \beta_0 + \beta_1 x_{n1} + \beta_2 x_{n2} + \cdots + \beta_p x_{np} + \varepsilon_n \end{cases}$$

In matrix form:

$$\begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix} = \begin{pmatrix} 1 & x_{11} & \cdots & x_{1p} \\ 1 & x_{21} & \cdots & x_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n1} & \cdots & x_{np} \end{pmatrix} \begin{pmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_p \end{pmatrix} + \begin{pmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \vdots \\ \varepsilon_n \end{pmatrix} \tag{1.2}$$

Equivalently, we write:

$$\boxed{Y = X\beta + \varepsilon} \tag{1.3}$$

- Y: is the value to be explained of size $(n \times 1)$.
- X: is the matrix of size $(n \times (p+1))$ that contains all observations on exogenous variables, with a first column consisting of the value 1, indicating that we include the constant 0 in the equation, where p is the number of real explanatory variables.
- β: is the parameter vector of size $((p+1) \times 1)$.
- ε: is the vector of errors of size $(n \times 1)$.

1.2 Assumptions of the multiple linear regression model

As in simple regression, the assumptions make it possible to determine the properties of the estimators (bias, convergence) and their distribution laws (for interval estimates and hypothesis tests). There are two main categories of assumptions:

1.2.1 Stochastic assumptions

- **H1:** The X_j are determined without errors, $j = 1, \cdots, p$.
- **H2:** $\forall i, \mathbb{E}(\varepsilon_i) = 0$, the model is well specified on average.
- **H3:** $\text{Var}(\varepsilon_i) = \sigma_\varepsilon^2$, $\forall i$, homoscedasticity of errors (constant variance).
- **H4:** $\forall i \neq j$, $\text{Cov}(\varepsilon_i, \varepsilon_j) = 0$, absence of autocorrelation (i.e., the errors associated with the different observations are uncorrelated with each other).
- **H5:** $\text{Cov}(\varepsilon_i, x_i) = 0$, the errors are linearly independent of the independent variables.

1.2.2 Structural hypotheses:

- **H6:** Absence of collinearity between independent variables (i.e., X^tX is regular, $\det(X^tX) \neq 0$ and $(X^tX)^{-1}$ exists).
- **H7:** $\frac{1}{n}X^tX$ tends towards a finite non-singular matrix.
- **H8:** $n > p+1$ The number of observations is strictly greater than the number of variables + 1 (the constant). If they were equal, the number of equations would be equal to the number of unknowns β_j, the regression line would pass through all the points, and we would be faced with a linear interpolation problem.
- **H9:** Variance–covariance matrix form:$\text{Var}(\varepsilon) = \text{Var}(Y) = \sigma_\varepsilon^2 I_n$. This implies:
  - homoscedasticity
  - absence of autocorrelation
  - diagonal structure

Remark 1.2.1. $\mathbb{E}(Y) = X\beta$.

Proof.

$$\begin{aligned}\mathbb{E}(Y) &= \mathbb{E}(X\beta + \varepsilon) \\ &= \mathbb{E}(X\beta) + \mathbb{E}(\varepsilon) \\ &= X\beta + 0 = X\beta\end{aligned}$$

□

Remark 1.2.2. *Homoscedasticity means that the error terms have constant variance σ_ε^2 for all observations, ensuring a uniform spread of residuals around the regression function.*

1.3 Estimation by Ordinary Least Squares

Definition 1.3.1. [3] Linear regression using the least squares method (denoted by OLS) is a statistical technique that fits a straight line to data to minimize the differences between observed and predicted values (i.e., the goal of OLS estimators is to minimize the sum of the squares of residuals). Form of MLR:

$$\underbrace{Y}_{(n\times 1)} = \underbrace{X}_{(n\times(p+1))}\ \underbrace{\beta}_{((p+1)\times 1)} + \underbrace{\varepsilon}_{(n\times 1)}. \tag{1.4}$$

Fitted values under the Ordinary Least Squares (OLS) method:

$$\hat{Y} = X\hat{\beta} \tag{1.5}$$

$\hat{Y}$: predicted value, the Ordinary Least Squares (OLS) estimator $\hat{\beta}$ is defined as follows:

$$\hat{\beta} = \arg\min_{\beta\in\mathbb{R}^{p+1}} \sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij} \right)^2. \tag{1.6}$$

Theorem 1.3.1. *[9][3](Estimator of β by OLS)*

$$\hat{\beta} = (X^t X)^{-1}(X^t Y) \tag{1.7}$$

is the estimator that minimizes the sum of the squares of the residuals.

Proof. We aim to minimize the sum of squared residuals:

$$\min \sum_{i=1}^{n} \sigma_\varepsilon^2 = \min e^t e = \min (Y - X\beta)^t (Y - X\beta).$$

Taking the derivative with respect to β and setting it to zero:

$$\frac{\partial}{\partial \beta}(Y - X\beta)^t (Y - X\beta) = 0\cdot$$

$$\frac{\partial}{\partial \beta}\left(Y^t Y - Y^t X\beta - \beta^t X^t Y + \beta^t X^t X\beta\right) = 0\cdot$$

Since $Y^t X\beta$ is a scalar, we have $Y^t X\beta = (Y^t X\beta)^t = \beta^t X^t Y$. Thus:

$$\frac{\partial}{\partial \beta}\left(Y^t Y - 2\beta^t X^t Y + \beta^t X^t X\beta\right) = 0\cdot$$

Computing the derivatives:

$$-2X^t Y + 2X^t X\beta = 0\cdot$$

$$2X^t X\beta = 2X^t Y \quad \Rightarrow \quad X^t X\beta = X^t Y\cdot$$

Assuming $X^t X$ is invertible (no perfect multicollinearity), we multiply both sides by $(X^t X)^{-1}$:

$$\hat{\beta} = (X^t X)^{-1} X^t Y\cdot$$

This is the ordinary least squares estimator that minimizes the sum of squared residuals. □

1.3.1 Geometric Interpretation of OLS

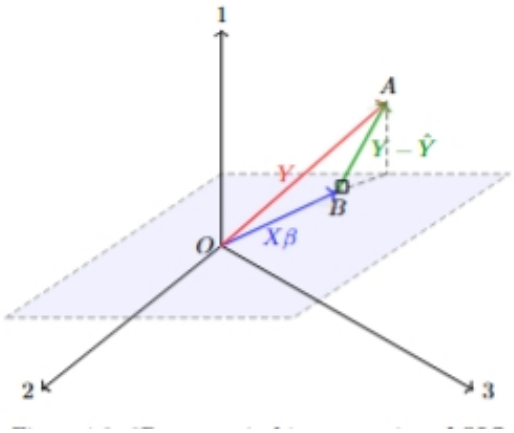

Figure 1.2: 3D geometrical interpretation of OLS

An intuitive geometric interpretation of least squares is often helpful. We may think of the data vector

$$Y^t = \begin{bmatrix} Y_1 & Y_2 & \cdots & Y_n \end{bmatrix}$$

as defining a vector from the origin to a point A in Figure 1.2. The components $Y_1, Y_2, \ldots, Y_n$ represent the coordinates of a point in an n-dimensional sample space. In Figure 1.2, this space is illustrated in three dimensions for simplicity.

The matrix X consists of p column vectors of dimension $(n \times 1)$, namely,

$$\mathbf{1}, X_1, X_2, \ldots, X_p$$

where $\mathbf{1}$ denotes the column vector of ones. Each column of X defines a vector from the origin in the sample space. These p vectors span a p-dimensional subspace of $\mathbb{R}^n$, called the *estimation space*. For example, when $p = 2$, the estimation space is a two-dimensional plane embedded in $\mathbb{R}^n$.

Any point in the estimation space can be expressed as a linear combination of the columns of X. Hence, any point in this subspace is of the form $X\beta$.

Let $X\beta$ determine the point B in Figure 1.2. The squared distance between point A and point B is given by

$$S(\beta) = (Y - X\beta)^t (Y - X\beta)\cdot$$

To minimize the squared distance between point A, defined by the observation vector Y, and the estimation space, we must determine the point in the estimation space that is closest to A.

This minimum occurs when the point in the estimation space is the orthogonal projection of A onto that space. This projected point is denoted C in Figure 1.2 and is defined by

$$\hat{Y} = X\hat{\beta}$$

because the residual vector

$$Y - \hat{Y} = Y - X\hat{\beta}$$

is orthogonal to the entire column space of X, it follows that

$$X^t(Y - X\hat{\beta}) = 0\cdot$$

Equivalently

$$X^tY = X^tX\hat{\beta}\cdot$$

These equations are known as the least-squares normal equations [13].In the remainder of this section, we will present the properties of the estimator $\hat{\beta}$.

Properties 1.3.1. *[12] OLS estimators are* ***unbiased****.*

Proof. According to (1.3) and by the hypotheses, we have:

$$\begin{aligned}\mathbb{E}(\varepsilon) &= 0\\ \mathbb{E}(Y) &= \mathbb{E}(X\beta + \varepsilon)\\ &= \mathbb{E}(X\beta) + \mathbb{E}(\varepsilon)\\ &= X\beta + 0 = X\beta\end{aligned}$$

and we have (1.3).

It must be proven that: $\mathbb{E}(\hat{\beta}) = \beta$

$$\begin{aligned}\mathbb{E}(\hat{\beta}) &= \mathbb{E}[(X^tX)^{-1}(X^tY)]\\ &= (X^tX)^{-1}X^t\mathbb{E}(Y)\\ &= (X^tX)^{-1}(X^tX)\beta\\ &= \beta\cdot\end{aligned}$$

Consequently, this proves that the estimator $\hat{\beta}$ is unbiased. The expected value of $\hat{\beta}$ is β. □

Properties 1.3.2. *[12] Variance-covariance matrix of $\hat{\beta}$:*
According to (1.3), (1.7), $\mathbb{E}(\varepsilon) = 0$ and $\text{var}(\varepsilon) = \sigma_\varepsilon^2 I$ we have:

$$\text{Var}(\hat{\beta}) = \text{Var}((X^tX)^{-1}X^tY)\cdot$$

We know that:
X is not random , $\text{Var}(\alpha Y) = \alpha\text{Var}(Y)\alpha^t$, $(AB)^t = B^tA^t$ and $[X^tX]^{-1} = [(X^tX)^{-1}]^t$
From this:

$$\begin{aligned}
\text{Var}(\hat{\beta}) &= (X^tX)^{-1}X^t\text{Var}(Y)[(X^tX)^{-1}X^t]^t \\
&= (X^tX)^{-1}X^t\sigma_\varepsilon^2 IX[(X^tX)^{-1}]^t \\
&= (X^tX)^{-1}X^t\sigma_\varepsilon^2 IX(X^tX)^{-1} \\
&= \sigma_\varepsilon^2(X^tX)^{-1}X^tX(X^tX)^{-1} \\
&= \sigma_\varepsilon^2(X^tX)^{-1}
\end{aligned}$$

σ_ε^2 is a diagonal matrix

$$\mathbb{E}(\varepsilon\varepsilon^t) = \begin{pmatrix}
\mathbb{E}(\varepsilon_1,\varepsilon_1) & \mathbb{E}(\varepsilon_1,\varepsilon_2) & \cdots & \mathbb{E}\varepsilon_1,\varepsilon_n) \\
\mathbb{E}(\varepsilon_2,1) & \mathbb{E}(\varepsilon_2,\varepsilon_2) & \cdots & \mathbb{E}(\varepsilon_2,\varepsilon_n) \\
\mathbb{E}(\varepsilon_3,\varepsilon_1) & \mathbb{E}(\varepsilon_3,\varepsilon_2) & \cdots & \mathbb{E}\varepsilon_3,\varepsilon_n) \\
\vdots & \vdots & \ddots & \vdots \\
\mathbb{E}(\varepsilon_n,\varepsilon_1) & \mathbb{E}(\varepsilon_n,\varepsilon_2) & \cdots & \mathbb{E}(\varepsilon_n,\varepsilon_n)
\end{pmatrix}$$

$$\begin{pmatrix}
\sigma_\varepsilon^2 & 0 & 0 & \cdots & 0 \\
0 & \sigma_\varepsilon^2 & 0 & \cdots & 0 \\
0 & 0 & \ddots & \cdots & 0 \\
\vdots & \vdots & \vdots & \ddots & 0 \\
0 & 0 & 0 & 0 & \sigma_\varepsilon^2
\end{pmatrix}$$

σ_ε^2 is the variance of the model error term ε.

Theorem 1.3.2. ***Gauss-Markov:*** *[11] The Gauss-Markov theorem says that, under certain conditions, the ordinary least squares (OLS) estimator of the coefficients of a linear regression model is the best linear unbiased estimator (BLUE), that is, the estimator that has the smallest variance among those that are unbiased and linear in the observed output variables.*

Proof. Let $\tilde{\beta}$ be an arbitrary linear unbiased estimator of β. Since it is linear, we can write $\tilde{\beta} = CY$ in the model $Y = X\beta + \varepsilon$. The unbiasedness condition implies:

$$\mathbb{E}(\tilde{\beta}) = C\mathbb{E}(Y) = CX\beta = \beta,$$

which holds for all β if and only if $CX = I$, where I is the identity matrix.

The variance of $\tilde{\beta}$ is:

$$\mathrm{Var}(\tilde{\beta}) = \mathrm{Var}(CY) = C\,\mathrm{Var}(Y)C^t = \sigma_\varepsilon^2 CC^t.$$

Let $P_X = X(X^tX)^{-1}X^t$ be the projection matrix onto the column space of X. Since P_X is idempotent and symmetric, we have $CC^t \geq CP_XC^t$. Indeed,

$$CC^t - CP_XC^t = C(I - P_X)C^t \geq 0,$$

because $I - P_X$ is positive semidefinite.

Thus,

$$\mathrm{Var}(\tilde{\beta}) \geq \sigma_\varepsilon^2 CP_XC^t = \sigma_\varepsilon^2 CX(X^tX)^{-1}X^tC^t.$$

But $CX = I$, so:

$$\mathrm{Var}(\tilde{\beta}) \geq \sigma_\varepsilon^2 (X^tX)^{-1}.$$

The right-hand side is exactly the variance of the OLS estimator $\hat{\beta}_{OLS} = (X^tX)^{-1}X^tY$. Therefore, the OLS estimator has the smallest variance among all linear unbiased estimators; it is the Best Linear Unbiased Estimator (BLUE). □

1.4 Estimation of the variance σ_ε^2

As in simple linear regression, we may develop an estimator of σ_ε^2 from the residual sum of squares [3].

$$\mathbf{RSS} = \sum_{i=1}^{n} \sigma_\varepsilon^2 = \sum_{i=1}^{n} (Y_i - \hat{Y}_i)^2 = e^te. \tag{1.8}$$

Substituting $e = Y - \hat{Y} = Y - X\hat{\beta}$, we have:

$$\begin{aligned}\mathbf{RSS} &= (Y - X\hat{\beta})^t(Y - X\hat{\beta}) \\ &= Y^tY - \hat{\beta}^tX^tY - Y^tX\hat{\beta} + \hat{\beta}^tX^tX\hat{\beta} \\ &= Y^tY - 2\hat{\beta}^tX^tY + \hat{\beta}^tX^tX\hat{\beta}.\end{aligned}$$

Since $X^tX\hat{\beta} = X^tY$, this last equation becomes

$$\mathbf{RSS} = Y^tY - \hat{\beta}^tX^tY. \tag{1.9}$$

We have shown in (1.3) that the residual sum of squares has $n - p - 1$ degrees of freedom associated with it, since $p + 1$ parameters are estimated in the regression model. The residual mean square (**RMS**) is

$$\mathbf{RMS} = \frac{\mathbf{RSS}}{n - p - 1}, \tag{1.10}$$

also shows that the expected value of MS Residual is σ^2_ε, so an **unbiased** estimator of σ^2_ε is given by:

$$\hat{\sigma}^2_\varepsilon = \mathbf{RMS}. \tag{1.11}$$

Remark 1.4.1. *As noted in the simple linear regression case, this estimator of σ^2_ε is model-dependent.*

1.5 Sampling Distributions

The estimated regression coefficients have a multivariate normal distribution due to the fact that they are equal to linear combinations of the normally distributed responses:

$$\hat{\beta} \sim \mathcal{N}_{p+1}(\beta, \sigma^2_\varepsilon(X^tX)^{-1}). \tag{1.12}$$

This means the least squares estimator is normally distributed, unbiased, and has a covariance matrix $\sigma^2_\varepsilon(X^tX)^{-1}$.
The variance estimator σ^2_ε (properly scaled) has a chi-squared distribution:

$$\frac{[n - (p + 1)]\hat{\sigma}^2_\varepsilon}{\sigma^2_\varepsilon} \sim \mathcal{X}^2_{(n-p-1)}. \tag{1.13}$$

Furthermore, the least squares estimator is independent of the variance estimator, which implies the studentized estimated coefficients are t-distributed:

$$\frac{\hat{\beta}_j - \beta_j}{\sqrt{\hat{\sigma}^2_\varepsilon(X^tX)^{-1}_{j,j}}} \sim t_{n-p-1}. \tag{1.14}$$

1.6 Confidence Intervals

Proposition 1.6.1. *[3] Confidence Intervals and Confidence Regions*

1. *For every $j \in \{1, \dots, p\}$, a $(1-\alpha)$ confidence interval for β_j is*

$$\left[\hat{\beta}_j - t_{n-p-1}^{1-\alpha/2}\, s_{\hat{\beta}_j} \;;\; \hat{\beta}_j + t_{n-p-1}^{1-\alpha/2}\, s_{\hat{\beta}_j}\right] \tag{1.15}$$

where $t_{n-p-1}^{1-\alpha/2}$ denotes the $(1-\alpha/2)$ quantile of a Student distribution with $n-p-1$ degrees of freedom, and

$$s_{\hat{\beta}_j} = \sqrt{\hat{\sigma}_\varepsilon^2 (X^T X)_{jj}^{-1}}$$

is the standard error of $\hat{\beta}_j$.

2. *A $(1-\alpha)$ confidence interval for σ_ε^2 is*

$$\left[\frac{(n-p-1)\hat{\sigma}_\varepsilon^2}{\chi_{n-p-1}^{1-\alpha/2}} \;;\; \frac{(n-p-1)\hat{\sigma}_\varepsilon^2}{\chi_{n-p-1}^{\alpha/2}}\right] \tag{1.16}$$

where χ_{n-p-1}^{γ} denotes the γ quantile of a chi-square distribution with $n-p-1$ degrees of freedom.

1.7 Variance Decomposition and ANOVA Table

As in the simple linear regression model, one can verify that the deviation between Y_i and the sample mean $\bar{Y}$ can be obtained by adding and subtracting the fitted value $\hat{Y}_i$ from the regression model. This procedure yields a decomposition into two components (see [12] and [13]):

$$Y_i - \bar{Y} = (Y_i - \hat{Y}_i) + (\hat{Y}_i - \bar{Y})\cdot \tag{1.17}$$

By squaring and summing over $i = 1, \dots, n$, we obtain the variance decomposition:

$$\sum_{i=1}^{n}(Y_i - \bar{Y})^2 = \sum_{i=1}^{n}(\hat{Y}_i - \bar{Y})^2 + \sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2. \tag{1.18}$$

Equivalently

$$\underbrace{\sum_{i=1}^{n}(Y_i - \bar{Y})^2}_{\textbf{TSS}} = \underbrace{\sum_{i=1}^{n}(\hat{Y}_i - \bar{Y})^2}_{\textbf{ESS}} + \underbrace{\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2}_{\textbf{RSS}} = \underbrace{\sum_{i=1}^{n}(\hat{Y}_i - \bar{Y})^2}_{\textbf{ESS}} + \underbrace{\sum_{i=1}^{n}\hat{e}_i^2}_{\textbf{RSS}}. \tag{1.19}$$

Where:

- **TSS** denotes the total sum of squares (centered),
- **ESS** denotes the explained sum of squares (centered),
- **RSS** denotes the residual sum of squares.

The **ANOVA** (analysis of variance) table for **MLR** is given by:

Source	**Degrees of Freedom**	**Sum of Squares**	**Mean Square**
Explained (Regression)	p	ESS	EMS = ESS/p
Residual (Error)	$n-p-1$	RSS	RMS = RSS/$(n-p-1)$
Total	$n-1$	TSS	

Table 1.1: ANOVA table for MLR

Remark 1.7.1. *The ANOVA decomposition can be written in matrix form as*

$$\underbrace{(Y-\bar{Y})^t(Y-\bar{Y})}_{\text{TSS}} = \underbrace{(\hat{Y}-\bar{Y})^t(\hat{Y}-\bar{Y})}_{\text{ESS}} + \underbrace{\hat{e}^t\hat{e}}_{\text{RSS}} \tag{1.20}$$

where $\bar{Y}$ denotes the vector in $\mathbb{R}^n$ whose components are all equal to the sample mean $\bar{Y}$, that is:

$$\bar{Y} = (\bar{Y}, \ldots, \bar{Y})^t.$$

1.8 Statistical Inference and Hypothesis Testing

After estimating the model parameters, we must evaluate their statistical significance by relying on the assumption of normally distributed errors. The standard hypothesis tests used for this purpose, the individual t-test, the overall F-test and the partial F-test are detailed below.

1.8.1 Test of Individual Regression Coefficients (t-test)

To determine if a specific predictor X_j significantly contributes to the model, we perform the following tests:

$$\begin{cases} H_0 : \beta_j = 0 \\ H_1 : \beta_j \neq 0 \end{cases}$$

The test statistic is:

$$t = \frac{\hat{\beta}_j}{\sqrt{\hat{\sigma}_\varepsilon^2 (X^t X)^{-1}_{jj}}}$$

where

$$\hat{\sigma}_\varepsilon^2 = \frac{\mathbf{RSS}}{n-p-1}.$$

Under the null hypothesis (H_0) and the normality assumption, the statistic follows a Student's t-distribution with $n-p-1$ degrees of freedom, ($t \sim t_{n-p-1}$).

Decision:

If the p-value is less than the significance level α, we reject H_0, which means that X_j has a statistically significant effect on the response variable.

1.8.2 Overall Significance of the Model (F-test)

It is also called the ANOVA F-test. To test if the regression model is globally significant, we test the following:

$$\begin{cases} H_0 : \beta_1 = \beta_2 = \cdots = \beta_p = 0 \\ H_1 : \exists! \quad \beta_j \neq 0 \end{cases}$$

The test statistic is:

$$F = \frac{\mathbf{ESS}/(p)}{\mathbf{RSS}/(n-p-1)} = \frac{\mathbf{EMS}}{\mathbf{RMS}}.$$

Using quadratic theory:

- $\mathbf{ESS}/\sigma_\varepsilon^2 \sim \mathcal{X}_p^2$
- $\mathbf{RSS}/\sigma_\varepsilon^2 \sim \mathcal{X}_{n-p-1}^2$
- They are independent

Thus

$$F \sim F_{(p),(n-p-1)}$$

Decision:

If the p-value is less than the significance level α, we reject H_0, indicating that the regression model is statistically significant and that at least one explanatory variable affects the response variable. Otherwise, we fail to reject H_0.

1.8.3 Partial F-test (General Linear Hypothesis)

The partial F-test is used to compare a reduced model with a full model.
Let:

- $\mathbf{RSS_R}$ = residual sum of squares (reduced model).
- $\mathbf{RSS_F}$ = residual sum of squares (full model).
- q = number of restrictions.

The test statistic is:

$$F = \frac{(\mathbf{RSS_R} - \mathbf{RSS_F})/q}{\mathbf{RSS_F}/(n-p-1)}$$

Thus:

$$F \sim F_{q,(n-p-1)}$$

Decision:

If the p-value is less than the significance level α, we reject H_0, indicating that the group of variables under consideration contributes significantly to the model. Otherwise, we fail to reject H_0.

1.9 Example: Agricultural Production Analysis

Consider a study examining the relationship between crop yield Y_i (quintals/hectare), the amount of fertilizer X_{i1} (kg/hectare), and irrigation water X_{i2} (m^3/hectare). The following data are collected from $n = 5$ agricultural plots:

Plot	**Yield (Y_i)**	**Fertilizer (X_{i1})**	**Water (X_{i2})**
1	40	50	200
2	50	60	250
3	35	40	180
4	55	70	300
5	45	55	220

Table 1.2: Agricultural data for MLR example

We postulate the linear model:

$$Y_i = \beta_0 + \beta_1 X_{i1} + \beta_2 X_{i2} + \varepsilon_i, \quad i = 1, \ldots, 5$$

Matrix Formulation

The model in matrix form is $Y = X\beta + \varepsilon$, where:

$$Y = \begin{pmatrix} 40 \\ 50 \\ 35 \\ 55 \\ 45 \end{pmatrix}, \quad X = \begin{pmatrix} 1 & 50 & 200 \\ 1 & 60 & 250 \\ 1 & 40 & 180 \\ 1 & 70 & 300 \\ 1 & 55 & 220 \end{pmatrix}, \quad \beta = \begin{pmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \end{pmatrix}, \quad \varepsilon = \begin{pmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \varepsilon_3 \\ \varepsilon_4 \\ \varepsilon_5 \end{pmatrix}.$$

OLS Estimation

The OLS estimator is given by $\hat{\beta} = (X^t X)^{-1} X^t Y$.

First, compute $X^t X$ and $X^t Y$:

$$X^t X = \begin{pmatrix} 5 & 275 & 1150 \\ 275 & 15625 & 65300 \\ 1150 & 65300 & 273300 \end{pmatrix}, \quad X^t Y = \begin{pmatrix} 225 \\ 12725 \\ 53200 \end{pmatrix}.$$

Solving the normal equations $(X^t X)\hat{\beta} = X^t Y$ yields:

$$\hat{\beta} = \begin{pmatrix} \hat{\beta}_0 \\ \hat{\beta}_1 \\ \hat{\beta}_2 \end{pmatrix} = \begin{pmatrix} 27.51 \\ 0.33 \\ -0.00032 \end{pmatrix}$$

Thus, the fitted regression equation is:

$$\hat{Y}_i = 27.51 + 0.33 X_{i1} - 0.00032 X_{i2}$$

Interpretation of Coefficients

- $\hat{\beta}_1 = 0.33$: Each additional kg/hectare of fertilizer increases the expected crop yield by 0.33 quintals/hectare, holding irrigation water constant.
- $\hat{\beta}_2 = -0.00032$: Each additional m^3/hectare of irrigation water increases the expected crop yield by -0.00032 quintals/hectare, holding fertilizer constant.
- $\hat{\beta}_0 = 27.51$: The intercept represents the expected yield when both fertilizer and water are zero (which may not be practically meaningful).

Goodness of Fit

The predicted values $\hat{Y}_i$ are:

$$\hat{Y} = X\hat{\beta} = \begin{pmatrix} 27.51 + 0.33(50) + (-0.00032)(200) \\ 27.51 + 0.33(60) + (-0.00032)(250) \\ 27.51 + 0.33(40) + (-0.00032)(180) \\ 27.51 + 0.33(70) + (-0.00032)(300) \\ 27.51 + 0.33(55) + (-0.00032)(220) \end{pmatrix} = \begin{pmatrix} 43.946 \\ 47.23 \\ 40.6524 \\ 50.514 \\ 45.5896 \end{pmatrix}$$

The residuals $e_i = Y_i - \hat{Y}_i$ are:

$$e = \begin{pmatrix} 40 - 43.95 \\ 50 - 47.23 \\ 35 - 40.65 \\ 55 - 50.51 \\ 45 - 45.59 \end{pmatrix} = \begin{pmatrix} -3.95 \\ 2.77 \\ -5.65 \\ 4.49 \\ -0.59 \end{pmatrix}$$

The residual sum of squares is:

$$\text{RSS} = \sum_{i=1}^{5} \sigma_\varepsilon^2 = (-3.95)^2 + (2.77)^2 + (-5.65)^2 + (4.49)^2 + (-0.59)^2$$

$$\text{RSS} = 15.60 + 7.67 + 31.92 + 20.16 + 0.35 = 75.7061$$

With $n - p = 5 - 3 = 2$ degrees of freedom, the unbiased estimator of σ^2 is:

$$\hat{\sigma}_\varepsilon^2 = \frac{\text{RSS}}{n - p} = \frac{75.70}{2} = 37.85$$

The total sum of squares is:

$$\text{TSS} = \sum_{i=1}^{5} (Y_i - \bar{Y})^2 = (40 - 45)^2 + (50 - 45)^2 + (35 - 45)^2 + (55 - 45)^2 + (45 - 45)^2$$

$$\text{TSS} = 25 + 25 + 100 + 100 + 0 = 250$$

The coefficient of determination is:

$$R^2 = 1 - \frac{\text{RSS}}{\text{TSS}} = 1 - \frac{75.70}{250} = 1 - 0.3028 = 0.6972$$

Thus, approximately 69.72% of the variability in crop yield is explained by the linear relationship with fertilizer and irrigation water.

Hypothesis Testing for Fertilizer

To test the significance of fertilizer ($H_0 : \beta_1 = 0$):

$$t = \frac{\hat{\beta}_1}{\sqrt{\hat{\sigma}_\varepsilon^2 (X^t X)^{-1}_{11}}} = \frac{0.33}{0.085} = 3.88$$

For $n - p - 1 = 2$ degrees of freedom at $\alpha = 0.05$, the critical value is $t_{0.025,2} = 4.303$. Since $3.88 < 4.303$, we fail to reject H_0 at the 5% significance level, indicating that fertilizer does not have a statistically significant effect on crop yield in this small sample.

Summary of Results

$$\hat{Y} = 27.51 + 0.33X_1 - 0.00032X_2, \quad R^2 = 69.72\%$$

Plot	Y_i	$\hat{Y}_i$	e_i	σ_ε^2
1	40	43.95	-3.95	15.60
2	50	47.23	2.77	7.67
3	35	40.65	-5.65	31.92
4	55	50.51	4.49	20.16
5	45	45.59	-0.59	0.35
Total				**75.70**

Table 1.3: Observed values, predicted values, residuals, and squared residuals

Chapter 2

Neural Network Regression

Neural networks (NNs) are systems inspired by biological neurons and primarily used for tasks such as big-data analysis, weather prediction, and biometric recognition. Although neural networks are widely known for use in deep learning and modeling complex problems such as image recognition, they are easily adapted to regression problems. Any class of statistical models can be termed a neural network if they use adaptive weights and can approximate non-linear functions of their inputs. Thus, neural network regression is suited to problems where a more traditional regression model (MLR) cannot fit a solution, and this is what we will delve into in detail in this chapter. The theoretical foundations of NNs are presented in Hastie et al. (2009) [9] and Bishop (2006) [2].

2.1 The Perceptron as a Mathematical Model

2.1.1 Historical Background

The idea of artificial neural networks goes back to 1940, when Walter Pitts and Warren McCulloch discovered that neurons in the human brain essentially perform combinations of logical operations and have binary outputs that depend on a specific threshold: active or not active. On this basis, mathematical models were built and generated great interest in the Artificial Intelligence Community (AIC). At the time, however, computers were already invented and were not close to handling algorithms of this complexity. Over the following years, scientists lost interest in neural networks due to the lack of progress in the field and other, more promising methods at the time. The scientist Frank Rosenblatt created perceptrons in the 1950s and 1960s. Since then, neural networks have been a topic of uprising in artificial intelligence.

2.1.2 Mathematical Formulation

A perceptron is an **Artificial Neuron** (AN). It is the simplest possible **Neural Network**. It defines a linear decision function used for binary classification. It makes choices by combining

inputs (x_i) with weights (w_i) and using an activation function ($\sigma(.)$). It is mostly used to solve problems with two choices. It is the basic part of many deep learning models. [14]

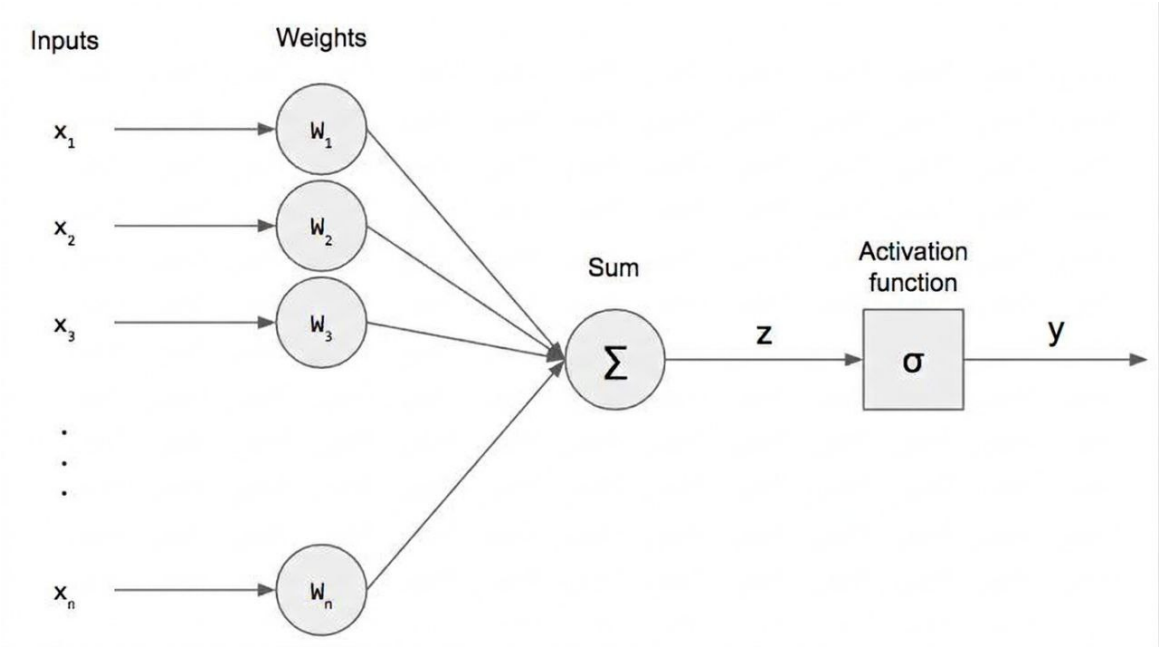

Figure 2.1: The simplest neural network known as the Perceptron

Definition 2.1.1. [4] Let the vector of input $X \in \mathbb{R}^n$, the vector of weights $W \in \mathbb{R}^n$, and $b \in \mathbb{R}$ biased term. On one hand, z is the result of processing the n-inputs x_i throughout the weights $(w_1, \cdots, w_n)$.

$$z = (w_1, w_2, \cdots, w_n) \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} + b = \sum_{i=1}^{n} w_i x_i + b = W^t X + b\cdot \tag{2.1}$$

On the other hand, the function $\sigma(\cdot)$, which is responsible for the nonlinearity of the whole process, is known as the activation function. This is chosen depending on the learning task and the desired properties of the model. Hence, each artificial neuron may be regarded as a mathematical function that results in an output y by chaining both the transfer function (linear) and the activation function (nonlinear) on the inputs X.
In general form, an artificial neuron network is

$$y = \sigma(z) = \sigma(W^t \cdot X + b)\cdot \tag{2.2}$$

The activation function of a perceptron is:

$$\sigma(z) = sign(z) \tag{2.3}$$

where

$$sign(z) = \begin{cases} +1 \quad if \quad z \geq 0 \\ -1 \quad if \quad z < 0 \end{cases}$$

2.1.3 Geometric interpretation

A linear binary classifier on $\mathbb{R}^d$ since $W^t \cdot X = -b$ is a hyperplane in $\mathbb{R}^d$. This hyperplane divides the space into two half-spaces and assigns the value 0 to one half and 1 to the other. The weight vector W is orthogonal to the separating hyperplane (W it sets the direction, and b: it sets the translation).

Remark 2.1.1. *It must be known*

- *Straight line if* $d = 2$.
- *Plan if* $d = 3$.
- *Hyperplane if* $d > 3$.

2.1.4 Link with Linear Regression

A perceptron is usually used for classification, but if the activation function is linear (identity function), it can also perform a linear regression function whose objective is to minimize the mean squared error of the model (MSE).

The Mean Squared Error (MSE) is defined as:

$$\mathcal{L}(W, b) = \frac{1}{2n} \sum_{i=1}^{n} (y^{(i)} - \hat{y}^{(i)})^2 \tag{2.4}$$

where n is the number of samples.

Perceptron	Linear Regression
Binary output	Continuous output
Used for classification	Used for regression
Sign activation	Identity activation

Table 2.1: Main Difference

Remark 2.1.2. *Perceptrons are often used for linear binary classification.*

2.2 Feedforward neural network model

A Feedforward Neural Network (FNN) is one of the simplest ANNs and an extension of the perceptron by introducing one or more hidden layers, allowing the network to learn nonlinear relationships in which information flows in one direction from the input layer to the output layer through intermediate hidden layers. It is often used in regression (for more details see [7]).

2.2.1 Single Hidden Layer Network

A single hidden-layer feedforward neural network (SLFN) is a fundamental neural network with an input layer, one intermediate hidden layer, and an output layer. Information flows in one direction, where neurons are fully connected, applying weighted sums and nonlinear activation functions.

Definition 2.2.1. [8] Let the hidden layer contain m neurons. Each hidden neuron computes a weighted linear combination of the input variables, followed by a nonlinear activation function. $X = (x_1, x_2, \cdots, x_n)^t \in \mathbb{R}^n$ is vector of inputs. For the j-th hidden neuron, we have the linear combination of the inputs:

$$z_j = W_j^t X + b_j \tag{2.5}$$

- $W_j \in \mathbb{R}^n$: Vector of weights associated with neuron j
- $b_j \in \mathbb{R}$: The bias term

Definition 2.2.2. [4] Let $j = 1, \cdots, m$ and the activation function $\sigma(.)$, the output of this neuron:

$$y_j = \sigma(z_j) = \sigma(W_j^t X + b_j) \tag{2.6}$$

$Y = (y_1, y_2, \cdots, y_m)^T$ is a vector that represents the output of the hidden layer.

The network's final output is obtained by calculating the weighted sum of the output of the hidden layer. The network output can then be written as

$$f(X) = \sum_{j=1}^{m} \alpha_j y_j + c \tag{2.7}$$

- α_j denotes the weight connecting the hidden neuron j to the output neuron.
- c The bias of the output neuron.

We substitute y_j with its expression

$$f(X) = \sum_{j=1}^{m} \alpha_j \sigma(W_j^t X + b_j) + c\cdot \tag{2.8}$$

This formulation (2.7) describes the functional representation of a single hidden-layer neural network. The activation function allows the network to approximate complex nonlinear mappings between inputs and outputs, thereby overcoming the limitations of linear models such as the perceptron.

2.2.2 Matrix formulation

The single hidden-layer neural network equation can be expressed in a compact matrix form. This form simplifies the notation and facilitates the analysis of neural network models.
Let the input vector $X \in \mathbb{R}^n$

$$X = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix}.$$

Assume that the hidden layer contains m neurons. The matrix of weights $W \in \mathbb{R}^{m \times n}$

$$W = \begin{pmatrix} w_{11} & w_{12} & \cdots & w_{1n} \\ w_{21} & w_{22} & \cdots & w_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ w_{m1} & w_{m2} & \cdots & w_{mn} \end{pmatrix}$$

and the bias vector $b \in \mathbb{R}^m$

$$B = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{pmatrix}.$$

The output hidden layer can be written in matrix form as

$$Z = \sigma(WX + B)\cdot$$

And therefore

$$Y = \sigma \left[\begin{pmatrix} w_{11} & w_{12} & \cdots & w_{1n} \\ w_{21} & w_{22} & \cdots & w_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ w_{m1} & w_{m2} & \cdots & w_{mn} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} + \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{pmatrix} \right]. \tag{2.9}$$

Expanding the matrix product gives

$$Y = \sigma \begin{pmatrix} w_{11}x_1 & w_{12}x_2 & \cdots & w_{1n}x_n + b_1 \\ w_{21}x_1 & w_{22}x_2 & \cdots & w_{2n}x_n + b_2 \\ \vdots & \vdots & \ddots & \vdots \\ w_{m1}x_1 & w_{m2}x_2 & \cdots & w_{mn}x_n + b_m \end{pmatrix}. \tag{2.10}$$

This corresponds to the computation performed by all neurons in the hidden layer simultaneously.

2.2.3 Extension to Multi-layer case

The multi-layer feedforward network is known as a multilayer perceptron (MLP), which contains several hidden layers between the input and the output, allowing the modeling of nonlinear relationships. The information in MLP sequentially goes from the input layer to the hidden layers and finally to the output layer (known as forward propagation in FNN). Let $x \in \mathbb{R}^n$ be the input vector, and denote $Y^{(0)} = X$. Each layer's output is obtained by applying a nonlinear activation function after a linear transformation. The transformation of the first hidden layer is written as

$$Y^{(1)} = \sigma(W^{(1)}X + B^{(1)})$$

where $W^{(1)}$ is the weight matrix connecting the input layer to the first hidden layer, and $B^{(1)}$ is the corresponding bias vector.
For the second hidden layer, the output becomes

$$Y^{(2)} = \sigma(W^{(2)}Y^{(1)} + B^{(2)})$$

more generally, for the l-th layer ($l = 1, 2, \ldots, L$), we have

$$Y^{(l)} = \sigma(W^{(l)}Y^{(l-1)} + B^{(l)}) \tag{2.11}$$

where $W^{(1)}$ and $B^{(l)}$ denote the weight matrix and bias vector associated with layer l.
Finally, the output layer computes

$$Y = \sigma(W^{(L)}Y^{(L-1)} + B^{(L)}) \tag{2.12}$$

- L: The total number of layers.
- $\sigma(\cdot)$: The activation function used in the output layer.

In light of what has been presented, the latter is the most suitable for conducting the regression study.

2.3 Activation function

In neural networks, the activation function is a mathematical function that uses a neuron's input to calculate its output. It is a function that should "activate" the neuron, as the name implies. The activation function determines whether convolutional or recurrent neural networks will be used. They receive input values and produce matching output values, acting as transfer functions. The vast majority of activation functions are non-linear, and for good reason. Neural networks are made more complex by nonlinear activation functions, which also allow them to "learn" to mimic a much wider range of functions. Neural networks would only be able to learn linear and affine functions if it weren't for nonlinear activation functions, as the layers would merely be a glorified affine function and would be linearly dependent on one another (for more details see [19], [21] and [8]).

2.3.1 The mechanism of activation functions

It's a good idea to have a firm grasp of how activation functions work in a neural network before talking about contemporary and popular activation functions. An activation function would take the values produced by a particular network layer (in a fully connected network, this would be the sum of weights and biases) and apply a certain transformation to them in order to map them to a particular range, regardless of the network architecture. The output of the given neuron is obtained by passing the weighted sum of the inputs plus the bias ($W_1X_1 + W_2X_2 + \cdots + W_nX_n + b$), to an activation function σ, In this instance, W_i represents our neuron's weights assigned to each input X, and each of the X_i values represents the output of a neuron from the preceding layer. The role of the activation function in the neural network can be explained by Figure 2.1.

2.3.2 Differentiability of activation function

Why should an activation function be differentiable? Differentiability is the most crucial characteristic that the activation function should possess. **Backpropagation** is an algorithm used by artificial neural networks to learn. In essence, this technique makes advantage of the model's inaccurate predictions to modify the network so that it becomes less inaccurate, hence enhancing the network's capacity for prediction. Differentiation is used to accomplish this.

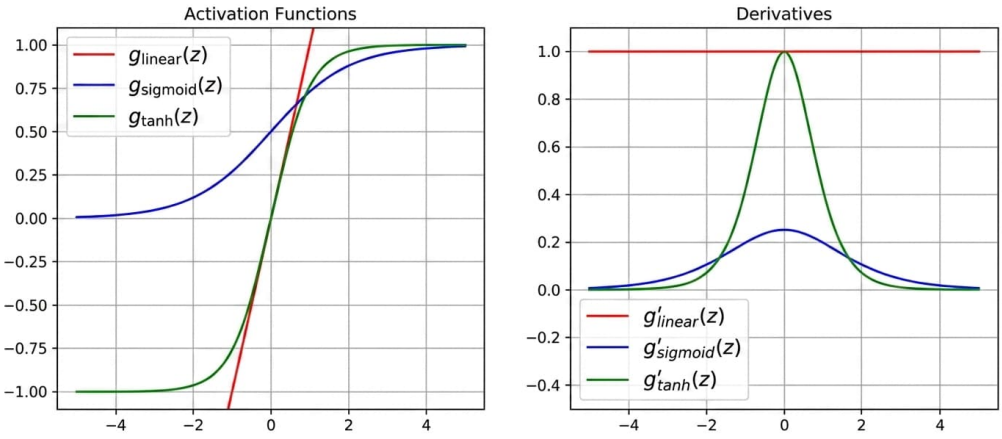

Figure 2.2: Activation functions and their derivatives

2.3.3 Sigmoid/Logistic activation function

The sigmoid activation function, often called the logistic activation function, is a common nonlinear activation function. It transforms input data to an output range between 0 and 1. The sigmoid activation function is shown as follows:

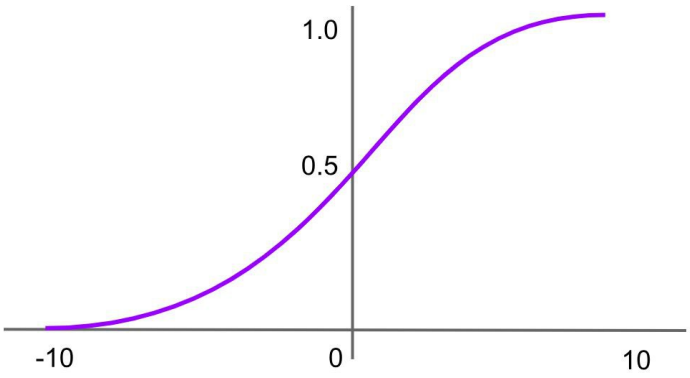

Figure 2.3: Sigmoid activation function

The sigmoid function formula is

$$\text{Sigmoid} = \sigma(x) = \frac{1}{(1 + \exp(-x))}. \tag{2.13}$$

For models where the probability must be predicted as an output, sigmoid functions are helpful. For instance, we want the outcome of binary classification issues to be represented as a probability between 0 and 1. Nevertheless, there is a vanishing gradient issue with the sigmoid. The network learns by updating weights during backpropagation, which results in very small sigmoid gradients and slow learning for deeper layers. The Python code for the sigmoid function:

```
import numpy as np
def sigmoid_function(x):
 z=(1/(1+np.exp(-x)))
 return z
```

2.3.4 Hyperbolic tangent activation function

One of the issues with the sigmoid function is resolved by the tanh activation function, which is similar to the sigmoid in that it translates the input values to an S-shaped curve. In this instance, the output range is (-1, 1) and is centered at 0. Tanh is an acronym for the hyperbolic tangent, which, like the conventional tangent, is simply the hyperbolic sine divided by the hyperbolic cosine.

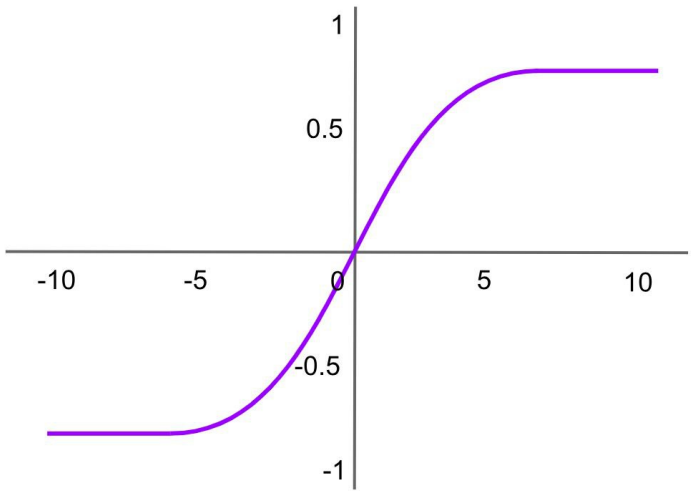

Figure 2.4: Tanh activation function

The Tanh function formula is

$$\tanh(x) = \sigma(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}. \tag{2.14}$$

Or

$$\tanh(x) = \sigma(x) = [2/(1 + \exp(-2x))] - 1 = 2 \ \ \text{sigmoid}(2x) - 1. \tag{2.15}$$

Even while the tanh activation function has the potential to be more efficient than the sigmoid activation function, it still has the same challenges with backpropagation as the sigmoid function. The derivative of the tanh function approaches zero at extremely large or very tiny values, making it more difficult to train the neural network. It is computationally expensive because it is an exponential function. To send better input values to the subsequent hidden

layer, the tanh function is a useful activation function that can be used in the hidden layers. The Python code for the latter is:

```
def tanh_function(x):
 z = (2/1+np.exp(-2*x)))-1
 return z
```

2.3.5 Rectified linear unit (ReLU) activation function

The ReLU activation function is a more modern and widely used activation function. It stands for Rectified Linear Unit and looks like this: The beauty of the ReLU activation function lies in its simplicity. As you can see, it replaces the negative value by 0 and keeps the positive value as is. This avoids the problem of "killing" large-value gradients and is faster to calculate because it involves simpler mathematical operations. In addition, the neuronal network using the ReLU tends to converge approximately six times faster than the sigmoid and tanh neural networks.

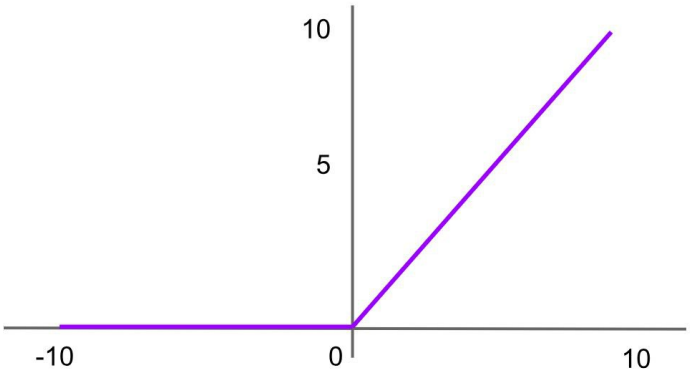

Figure 2.5: ReLU activation function

The formula for the ReLU function is:

$$ReLU = \sigma(x) = \max(0, x)\cdot \tag{2.16}$$

However, ReLU is still facing some problems. It is not first the neural; it is not 0-centered, which can cause problems during training. However, the most important thing is that it does not deal particularly meaningfully with negative inputs. During the reverse propagation process, the neural network updates the weights with gradients. Some neurons with negative input values will have zero gradients, and some neurons may not be updated. Some neurons with negative input values will not be updated during the entire process of training neural networks. These neurons are called "dead" neurons. Modern activation functions tend to take ReLU and try to fix these problems. Many variations of the ReLU function have been developed to avoid

this problem in neural network model training. Python code for ReLU:

```
def  relu_function(x):
  if x < 0:
     return 0
  else:
     return x
```

2.3.6 Leaky ReLU activation function

The Leaky ReLU activation function is a variation of the standard ReLU. It allows for a small, non-zero gradient when the input is negative. This design helps neurons keep learning, preventing them from becoming permanently inactive.

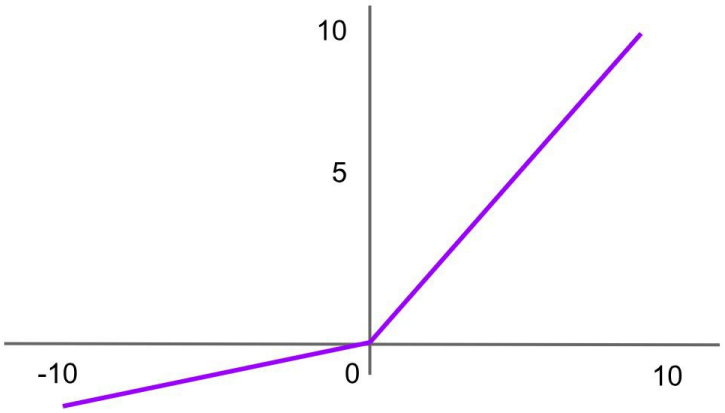

Figure 2.6: Leaky ReLU activation function

Mathematical formula

$$Leaky \;\; ReLU = \sigma(x) = \max(\alpha x, x) \tag{2.17}$$

- α is a minor positive constant (e.g., 0.01) utilized to guarantee that the neuron outputs a small negative value rather than zero for negative inputs.

The Python code for this last function

```
  def leaky_relu_function(x):
     if x < 0:
        return 0.01*x
     else:
        return x
```

2.3.7 Activation Function Selection in Neural Networks

The choice of the activation function depends on the nature of the prediction problem. In neural networks, different activation functions are used according to the role of each layer and the type of output required.

- **For binary classification:** the sigmoid activation function is commonly used in the output layer, since it maps values into the interval $(0, 1)$ and can be interpreted as a probability.
- **For hidden layers:** the ReLU activation function is often preferred due to its simplicity, computational efficiency, and ability to reduce the vanishing gradient problem.
- **For regression:** the output layer is often linear; that is, no activation function is applied in order to allow the network to produce unbounded real-valued outputs.

Remark 2.3.1. *Without an activation function, the NN would just be a linear model.*

2.4 Neural Network Regression Framework

Neural Network Regression extends beyond linear models like multiple linear regression, which we previously discussed in the first chapter (see 1.3), and may be inadequate for explaining relationships in complex or nonlinear data. Using nonlinear transformations of input variables via hidden layers, it offers a flexible approach to approximate intricate functional relationships and capture complex patterns between predictors and the response variable. This is what we will formulate in this section.

2.4.1 Regression Function

In general, the regression function is

$$m(x) = \mathbb{E}[Y \mid X] \tag{2.18}$$

The objective is to employ neural network regression to construct an approximate function that assists in determining the optimal value for Y (i.e, $f_\theta(x) \approx f(x)$). The model for NNR with one hidden layer:

$$f_\theta(x) = V^t \sigma(WX + B) + c\cdot \tag{2.19}$$

2.4.2 Empirical Risk Minimization (ERM)

The empirical risk minimization is defined as follows:

$$\mathcal{L}(\theta) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{2.20}$$

where $y_i = f(x_i)$ is the true value and $\hat{y}_i = f_\theta(x_i)$ is the predicted value.

The goal is to determine the optimal parameters that minimize the empirical risk function. This optimization problem can be expressed as:

$$\theta^* = \arg\min_\theta \mathcal{L}(\theta) \tag{2.21}$$

Figure 2.7 illustrates the geometrical interpretation of the Empirical Risk Minimization process in neural regression. Unlike linear regression, the function $f_\theta(x)$ defines a complex manifold that adaptively adjusts to minimize the vertical distances, thereby capturing nonlinear patterns [9].

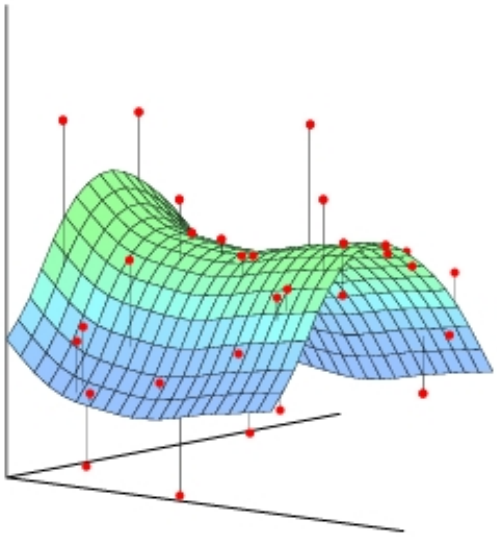

Figure 2.7: Geometric interpretation of empirical risk minimization.

2.5 Optimization and Backpropagation

2.5.1 Gradient Descent

Gradient descent is a well-known optimization algorithm that we have previously studied in linear algebra. Gradient descent can be viewed as a local search algorithm to locate a function's minimum. Think of a search space whose dimension is determined by the neural network's weight count. Finding a weight combination that minimizes the overall error on the training instances is our goal. We will utilize the gradient, or partial derivative, to decide on the

magnitude and direction of each step we wish to take. To minimize the loss function defined in the previous section (2.20), which measures how far off predicted values are from true values, we use gradient descent. Based on this loss function, the parameters are updated iteratively using the gradient descent rule:

$$\theta^{(T+1)} = \theta^{(T)} - \eta \nabla \mathcal{L}(\theta^{(T)}). \tag{2.22}$$

Where:

- θ: represents the set of model parameters (weights and biases).
- η: is the learning rate, controlling the step size of the update.
- $\mathcal{L}(\theta)$: is the loss function.
- $\nabla \mathcal{L}(\theta)$: denotes the gradient of the loss function with respect to the parameters.
- T: is the iteration index.

Remark 2.5.1. *The gradient descent algorithm, used to minimize the loss function, is described in detail in Goodfellow, Bengio and Courville (2016, Chapter 8 [8]).*

2.5.2 The Chain Rule

The chain rule provides a way to calculate the derivative of composite functions, functions nested within one another. This is exactly the situation we have in a neural network.

Definition 2.5.1. [8]**Derivatives of Composite Function**
Let's review the fundamental concept. Suppose we have a simple composition of functions. If a variable y depends on a variable v, written as $y = f(v)$, and v itself depends on another variable x, written as $v = g(x)$, then y indirectly depends on x through $v : y = f(g(x))$. The chain rule tells us how to find the rate of change of y with respect to x, denoted as $\frac{dy}{dx}$. It states that this derivative is the product of the derivatives of the "outer" function with respect to its input and the "inner" function with respect to its input:

$$\frac{dy}{dx} = \frac{dy}{dv} \times \frac{dv}{dx}.$$

We can extend this to longer chains. If $y = f(z)$, $z = h(x)$ and $x = g(u)$, then y depend on u through this chain. The derivative of y with respect to u is found by multiplying the derivatives along the path:

$$\frac{dy}{du} = \frac{dy}{dz} \times \frac{dz}{dx} \times \frac{dx}{du}.$$

Chain Rule for Neural Networks

Consider a simple feedforward neural network consisting of:

- one input neuron with input x,
- one hidden neuron with activation h,
- one output neuron producing the prediction $\hat{y}$.

The forward propagation through this network is defined by the following sequence of computations:

1. Pre-activation of the hidden neuron:

$$z_1 = w_1 x + b_1$$

2. Activation of the hidden neuron (using a non-linear activation function σ):

$$h = \sigma(z_1)$$

3. Pre-activation of the output neuron:

$$z_2 = w_2 h + b_2$$

4. Output (prediction) of the network:

$$\hat{y} = \sigma(z_2)$$

5. Computation of the loss (e.g., mean squared error):

$$\mathcal{L} = \frac{1}{2}(y - \hat{y})^2$$

   where y denotes the true target value.

To train the network, we need to compute the gradient of the loss with respect to the weight w_1, i.e., $\dfrac{\partial \mathcal{L}}{\partial w_1}$. Applying the chain rule of calculus, we obtain:

$$\frac{\partial \mathcal{L}}{\partial w_1} = \frac{\partial \mathcal{L}}{\partial \hat{y}} \times \frac{\partial \hat{y}}{\partial z_2} \times \frac{\partial z_2}{\partial h} \times \frac{\partial h}{\partial z_1} \times \frac{\partial z_1}{\partial w_1}.$$

Each term is computed as follows:

- $\dfrac{\partial \mathcal{L}}{\partial \hat{y}} = -(y - \hat{y})$ for the squared error loss.

- $\frac{\partial \hat{y}}{\partial z_2} = \sigma'(z_2)$, the derivative of the output activation function.
- $\frac{\partial z_2}{\partial h} = w_2$, since $z_2 = w_2 h + b_2$.
- $\frac{\partial h}{\partial z_1} = \sigma'(z_1)$, the derivative of the hidden activation function.
- $\frac{\partial z_1}{\partial w_1} = x$, since $z_1 = w_1 x + b_1$.

Thus, the complete gradient expression becomes:

$$\frac{\partial \mathcal{L}}{\partial w_1} = -(y - \hat{y}) \times \sigma'(z_2) \times w_2 \times \sigma'(z_1) \times x.$$

This derivation illustrates how the chain rule enables the propagation of error signals backward through the network, forming the foundation of the backpropagation algorithm. The same principle applies to deeper architectures, where gradients are computed layer by layer from the output to the input.

Dependencies get more complicated in multi-layer networks with many neurons per layer (a neuron's output impacts numerous neurons in the next layer), including sums of derivatives. The fundamental idea is still the same, though: by propagating derivative information backward through the network, layer by layer, the chain rule enables us to compute gradients. The backpropagation algorithm, which we shall describe in the sections that follow, accomplishes this methodical application of the chain rule. The first important step in comprehending how neural networks learn is to grasp the chain rule (for more details, see [1] and [18]).

2.5.3 Backpropagation Algorithm

A neural network undergoes training by adjusting its weights and biases to minimize an error function, a process frequently accomplished through gradient descent. A fundamental requirement for gradient descent is the computation of the error function's gradient with respect to each network parameter, encompassing both weights and biases. Directly calculating these gradients for a deep network comprising millions of parameters would entail a computationally prohibitive cost. This is precisely where the backpropagation algorithm becomes instrumental. It is not an optimization algorithm in itself, but rather a highly efficient methodology for deriving the necessary gradients. Backpropagation judiciously applies the chain rule of calculus, propagating error signals backward from the final error computation through the network's layers to quantify each parameter's contribution to the overall error (see [15]).

The Two Backpropagation Passes

The algorithm operates through two distinct passes:

1. **Forward Pass:** This pass constitutes the conventional procedure of propagating an input X through the network, layer by layer. During this phase, the weighted sums ($z^{(l)}$) and activations ($Y^{(l)}$) are calculated for each layer until the final output activation ($Y^{(L)}$) is obtained and the error $\mathcal{L}$ is computed. Throughout this pass, it is imperative to retain the intermediate values, specifically the weighted sums $z^{(l)}$ and activations $Y^{(l)}$ for each layer l, as these are subsequently utilized in the backward pass.

$$Z^{(l)} = W^{(l)}X^{(l-1)} + B^{(l)}$$

$$Y^{(l)} = \sigma(Z^{(l)})$$

2. **Backward Pass:** This phase represents the central mechanism of backpropagation. It commences at the output layer and propagates the gradient of the error ($\mathcal{L}$) iteratively backward through the network structure.

   - **Output Layer (Layer L):** Initially, the gradient of the error with respect to the output activations $\frac{\partial \mathcal{L}}{\partial z^{(L)}}$, $\frac{\partial \mathcal{L}}{\partial Y^{(L)}}$ is determined. Its specific mathematical form is contingent upon the chosen error function. By applying the chain rule and leveraging the stored $z^{(L)}$ value, the error gradient can be derived in relation to the pre-activation value $z^{(L)}$ for the output layer. This term is conventionally denoted as the "error" $\delta^{(L)}$ for the output layer:

$$\delta^{(L)} = \frac{\partial \mathcal{L}}{\partial z^{(L)}} = \frac{\partial \mathcal{L}}{\partial Y^{(L)}} \odot \sigma'(z^{(L)}) \cdot \tag{2.23}$$

     Here, σ' denotes the derivative of the activation function applied in the output layer, and the symbol $\odot$ denotes the Hadamard product (element-wise multiplication), where corresponding elements of two vectors or matrices are multiplied individually. This operation is appropriate given that both $\frac{\partial \mathcal{L}}{\partial Y^{(L)}}$ and $\sigma'(z^{(L)})$ are vectors sharing the same dimensionality as the output layer.

   - **Gradients for Output Layer Parameters:** Given $\delta^{(L)}$, the gradients for the weights $W^{(L)}$ and biases $B^{(L)}$ can be computed as follows:

$$\frac{\partial \mathcal{L}}{\partial W^{(L)}} = \delta^{(L)}(Y^{(L-1)})^t,$$

$$\frac{\partial \mathcal{L}}{\partial B^{(L)}} = \delta^{(L)}.$$

- **Propagating the Error Backward:** Let $\mathcal{L}$ represent the loss function. To facilitate further backward propagation of the error, it is necessary to determine the sensitivity of $\mathcal{L}$ with respect to the activations of the preceding layer, $Y^{(L-1)}$. Applying the chain rule, and utilizing the relationship $z^{(L)} = W^{(L)}Y^{(L-1)} + b^{(L)}$, yields:

$$\frac{\partial \mathcal{L}}{\partial Y^{(L-1)}} = \left(W^{(L)}\right)^t \delta^{(L)}. \tag{2.24}$$

- **Hidden Layers (Layer l):** The procedure is then generalized for an arbitrary hidden layer l, progressing backward from $L = 1$ down to 1. It is presumed that the error propagated from the layer $l+1$ has been previously computed. For the current layer, the error $\delta^{(l)}$ is calculated as:

$$\delta^{(l)} = \frac{\partial \mathcal{L}}{\partial z^{(l)}} = \frac{\partial \mathcal{L}}{\partial Y^{(l)}} \odot \sigma'(z^{(l)}) = ((W^{(l+1)})^t \delta^{(l+1)}) \odot \sigma'(z^{(l)}). \tag{2.25}$$

  This equation illustrates the mechanism by which the error originating from the subsequent layer ($\delta^{(l+1)}$), scaled by the synaptic weights ($W^{(l+1)}$), integrates with the local gradient of the activation function ($\sigma'(z^{(l)})$) to establish the error for the current layer.

- **Gradients for Hidden Layer Parameters:** Utilizing $\delta^{(l)}$, the gradients for the parameters of the layer l are computed as follows:

$$\frac{\partial \mathcal{L}}{\partial W^{(l)}} = \delta^{(l)} (Y^{(l-1)})^t,$$

$$\frac{\partial \mathcal{L}}{\partial B^{(l)}} = \delta^{(l)} .$$

- **Continue Backward:** This process is iteratively applied, calculating $\delta^{(l)}$, $\frac{\partial \mathcal{L}}{\partial W^{(l)}}$, and $\frac{\partial \mathcal{L}}{\partial B^{(l)}}$ for each layer l, progressing from $L-1$ down to 1. Each iteration in this sequence relies upon the error $\delta^{(l+1)}$ derived during the preceding step, which corresponds to layer $l+1$.

Workflow of the Backpropagation Algorithm

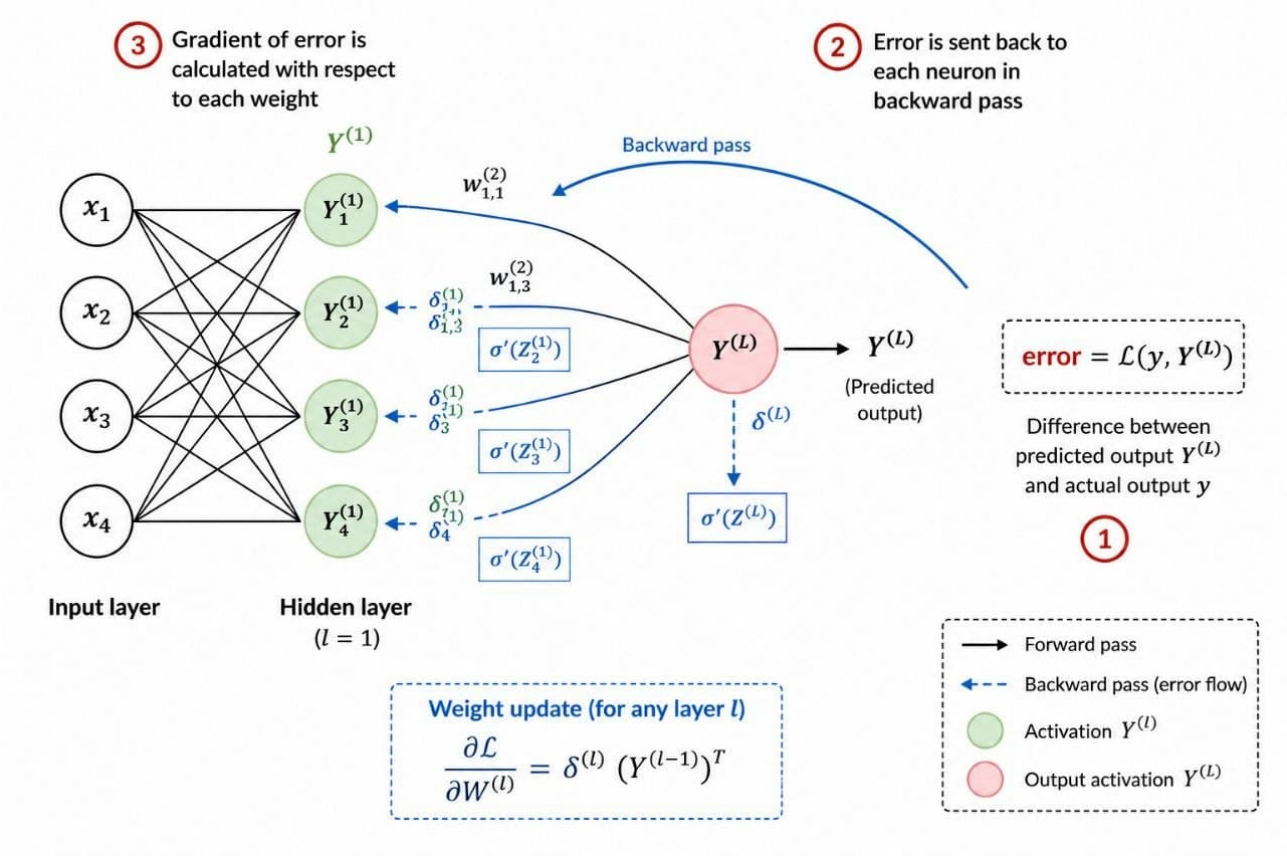

Figure 2.8: Backpropagation process in a neural network showing forward pass, error computation, and backward propagation using gradient descent.

The accompanying figure depicts the backpropagation algorithm as applied within a neural network. In the forward propagation phase, input data, denoted as X, traverses the network sequentially, layer by layer. Each successive layer l generates an output $Y^{(l)}$, culminating in the network's final predicted output, $Y^{(L)}$. Subsequently, the discrepancy between this predicted output and the true value is quantified via a designated loss function, thereby yielding a measure of the network's error. During the backward propagation phase, this calculated error is then systematically disseminated backward through the network, commencing from the output layer and proceeding towards the preceding layers. For each individual layer, **the gradient** of the loss function with respect to the associated weights is determined. This computation relies on the application of **the chain rule** and the derivative of the layer's activation function, represented as σ'. These derived gradients provide an indication of each weight's contribution to the overall error. Consequently, they are instrumental in adjusting the model's parameters, a process undertaken with the objective of minimizing the accumulated loss. Through this iterative optimization procedure, the neural network acquires the capacity to learn and progressively refine its predictive accuracy over subsequent iterations.

Efficiency of backpropagation

Backpropagation significantly improves the efficiency of training neural networks. Instead of computing gradients independently for each parameter, which would be computationally expensive, backpropagation applies the chain rule to reuse intermediate results during the backward pass. This allows all gradients to be computed efficiently using only one forward pass and one backward pass through the network. As a result, the computational cost is greatly reduced, making it feasible to train deep neural networks with a large number of parameters.

2.6 Theoretical Properties

2.6.1 Universal Approximation

The Universal Approximation Theorem (UAT) serves as the theoretical foundation for the efficacy of neural networks in regression tasks. Formally, the theorem states that a feedforward neural network with at least one hidden layer and a finite number of neurons can approximate any continuous function $f : \mathbb{R}^n \to \mathbb{R}$ on a compact subset of the input space to any desired degree of accuracy, provided that the activation function is non-polynomial (Cybenko, 1989 [6]; Hornik, 1991 [10]).
In the context of regression, this implies that a neural network can model complex, non-linear mappings between input features and continuous target variables, regardless of the underlying data distribution. While the theorem guarantees the existence of a network configuration that minimizes the approximation error to an arbitrary threshold ε, it does not explicitly dictate the optimal number of neurons or the specific architectural design required for a given dataset. Consequently, deep learning practice relies on empirical hyperparameter optimization and iterative training to leverage this universal approximation capability effectively for high-dimensional predictive modeling.

Theorem 2.6.1 (Universal Approximation)**.** *[6] Let $I_n = [0,1]^n$ be the n-dimensional unit hypercube and $C(I_n)$ be the space of continuous functions on I_n. Given any function $f \in C(I_n)$ and any $\varepsilon > 0$, there exists a finite number of neurons M, weights $w_i \in \mathbb{R}^n$, biases $b_i \in \mathbb{R}$, and output weights $v_i \in \mathbb{R}$ such that the neural network function $F(x)$, defined as:*

$$F(x) = \sum_{i=1}^{M} v_i \cdot \sigma(w_i^t x + b_i),$$

satisfies the following condition:

$$\sup_{x \in I_n} |F(x) - f(x)| < \varepsilon \tag{2.26}$$

where $\sigma(\cdot)$ is a non-polynomial, continuous, and bounded activation function.

Proof. The proof, based on the work of Cybenko (1989) [6], relies on the Hahn-Banach theorem from functional analysis:

1. Define $\mathcal{S}$ as the set of all finite linear combinations of the form
$$\sum_{i=1}^{M} v_i \sigma(w_i^t x + b_i) \cdot$$
2. Assume $\mathcal{S}$ is not dense in $C(I_n)$. By the Hahn-Banach theorem, there exists a non-zero, bounded linear functional L such that $L(\mathcal{S}) = 0$.
3. By the Riesz representation theorem, L corresponds to a finite signed Borel measure μ, such that:
$$\int_{I_n} \sigma(w^t x + b) d\mu(x) = 0, \quad \forall w \in \mathbb{R}^n, b \in \mathbb{R} \cdot \tag{2.27}$$
4. Since σ is discriminatory, this implies $\mu = 0$, which contradicts the assumption that $L \neq 0$. Thus, $\mathcal{S}$ must be dense in $C(I_n)$.

□

2.6.2 Model Complexity and the Bias-Variance Trade-off

In the context of regression analysis, model complexity refers to the capacity of a mathematical model to capture the underlying functional relationships within the data. While neural networks are universal approximators (as demonstrated in Section 2.7), this flexibility introduces a fundamental challenge known as the *Bias-Variance Trade-off*. For more details, see [20] and [5].

Mathematical Formulation

Let $f(x)$ be the true underlying function and $f_\theta(x)$ be the estimator produced by the neural network with parameters θ. The expected prediction error at a point x_0 can be decomposed as follows:

$$\mathbb{E}[(Y - f_\theta(x_0))^2] = \text{Bias}^2(f_\theta(x_0)) + \text{Var}(f_\theta(x_0)) + \sigma_\varepsilon^2, \tag{2.28}$$

where:

- $\text{Bias}^2(f_\theta(x_0))$: Measures the error resulting from erroneous assumptions in the learning algorithm (e.g., using a model that is too simple).
- $\text{Var}(f_\theta(x_0))$: Measures the sensitivity of the model to small fluctuations in the training set, which increases with model complexity (overfitting).
- σ_ε^2: Represents the irreducible noise inherent in the data.

2.6.3 Complexity Control

In neural network regression, model complexity is mainly determined by the number of hidden layers, the number of neurons in each layer, and the magnitude of the weight parameters. To reduce overfitting, regularization techniques are commonly applied.

One of the most widely used methods is L2 regularization (weight decay), which modifies the loss function by adding a penalty term associated with large weight values:

$$\mathcal{L}_{reg}(\theta) = \mathcal{L}(\theta) + \lambda \sum_{l=1}^{L} \left\| W^{(l)} \right\|_F^2 \tag{2.29}$$

where:

- $\mathcal{L}(\theta)$ is the original loss function.
- $\mathcal{L}_{reg}(\theta)$ is the regularized loss function.
- λ is the regularization hyperparameter.
- $W^{(l)}$ denotes the weight matrix of layer l.
- $\| \cdot \|_F$ denotes the Frobenius norm.

This regularization technique constrains the magnitude of the weights and helps the neural network generalize better by reducing overfitting.

2.7 Example: Nonlinear Agricultural Production

We consider the same agricultural problem, but now the relationship between inputs (fertilizer x_1 and water x_2) and crop yield y exhibits nonlinear behavior. We will use a single hidden-layer feedforward neural network to model this relationship.

Dataset

We have $n = 4$ observations:

Plot	**Fertilizer** (x_1)	**Water** (x_2)	**Yield** (y)
1	50	200	42
2	60	250	48
3	40	180	36
4	70	300	54

Table 2.2: Agricultural data for NNR example

Neural Network Architecture

We consider a feedforward neural network with:

- Input layer: 2 neurons (fertilizer x_1, water x_2).
- Hidden layer: 2 neurons with activation function $\sigma(z) = \max(0, z)$ (ReLU).
- Output layer: 1 neuron with linear activation (for regression).

The network function is:

$$f_\theta(x) = v_1\sigma(w_{11}x_1 + w_{12}x_2 + b_1) + v_2\sigma(w_{21}x_1 + w_{22}x_2 + b_2) + c\cdot$$

Initial Parameters

Suppose we initialize the parameters as follows:

$$W^{(1)} = \begin{pmatrix} 0.2 & 0.1 \\ 0.3 & 0.4 \end{pmatrix}, \quad B^{(1)} = \begin{pmatrix} 0.5 \\ 0.5 \end{pmatrix}, \quad V = \begin{pmatrix} 0.6 \\ 0.6 \end{pmatrix}, \quad c = 0.2$$

Forward Pass for Plot 1 ($x_1 = 50$, $x_2 = 200$)

Hidden layer computation:

$$z_1 = w_{11}x_1 + w_{12}x_2 + b_1 = 0.2(50) + 0.1(200) + 0.5 = 10 + 20 + 0.5 = 30.5$$

$$z_2 = w_{21}x_1 + w_{22}x_2 + b_2 = 0.3(50) + 0.4(200) + 0.5 = 15 + 80 + 0.5 = 95.5$$

Activation (ReLU):

$$h_1 = \sigma(z_1) = \max(0, 30.5) = 30.5$$

$$h_2 = \sigma(z_2) = \max(0, 95.5) = 95.5$$

Output layer:

$$\hat{y} = v_1h_1 + v_2h_2 + c = 0.6(30.5) + 0.6(95.5) + 0.2 = 18.3 + 57.3 + 0.2 = 75.8$$

The true yield is $y = 42$, so the prediction error is $75.8 - 42 = 33.8$.

Loss Function (MSE)

For a single observation, the squared error loss is:

$$\mathcal{L} = \frac{1}{2}(y - \hat{y})^2 = \frac{1}{2}(42 - 75.8)^2 = \frac{1}{2}(-33.8)^2 = \frac{1142.44}{2} = 571.22$$

Backpropagation (Gradient Computation)

We compute the partial derivatives using the chain rule. First, the derivative of the loss with respect to the prediction is:

$$\frac{\partial \mathcal{L}}{\partial \hat{y}} = -(\hat{y} - y) = 75.8 - 42 = 33.8$$

Gradient with respect to v_1

$$\frac{\partial \mathcal{L}}{\partial v_1} = \frac{\partial \mathcal{L}}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial v_1} = (\hat{y} - y) \cdot h_1 = 33.8 \times 30.5 = 1030.9$$

Gradient with respect to w_{11}

$$\frac{\partial \mathcal{L}}{\partial w_{11}} = \frac{\partial \mathcal{L}}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial h_1} \cdot \frac{\partial h_1}{\partial z_1} \cdot \frac{\partial z_1}{\partial w_{11}}$$

We have:

- $\frac{\partial \hat{y}}{\partial h_1} = v_1 = 0.6$
- $\frac{\partial h_1}{\partial z_1} = 1$ (since $z_1 > 0$ for ReLU)
- $\frac{\partial z_1}{\partial w_{11}} = x_1 = 50$

Thus:

$$\frac{\partial \mathcal{L}}{\partial w_{11}} = 33.8 \times 0.6 \times 1 \times 50 = 1014$$

Parameter Update (Gradient Descent)

With learning rate $\eta = 0.01$:

$$v_1^{\text{new}} = v_1 - \eta \frac{\partial \mathcal{L}}{\partial v_1} = 0.6 - 0.01 \times 1030.9 = 0.6 - 10.309 = -9.709$$

$$w_{11}^{\text{new}} = w_{11} - \eta \frac{\partial \mathcal{L}}{\partial w_{11}} = 0.2 - 0.01 \times 1014 = 0.2 - 10.14 = -9.94$$

Remark 2.7.1. *In the same way, we calculate the other plots.*

Interpretation

After training over many epochs (iterations), the neural network learns to approximate the true nonlinear relationship between agricultural inputs and crop yield. Unlike linear regression, the neural network can capture complex interactions such as:

- Diminishing returns: additional fertilizer increases yield, but the marginal gain decreases at high levels.
- Synergy between water and fertilizer: the combined effect is greater than the sum of individual effects.

2.8 Theoretical Comparison: Neural Regression vs. MLR

While Multiple Linear Regression (MLR) and Neural Regression (NNR) both aim to model the relationship between variables, they differ fundamentally in their theoretical approach and structural flexibility.

Criterion	MLR	NNR
Functional form	$Y = X\beta + \varepsilon$ (linear)	$f_\theta(X) = \sigma(W^{(L)}\sigma(\cdots\sigma(W^{(1)}X + b^{(1)})\cdots) + b^{(L)})$ (non-linear)
Parameter estimation	Ordinary Least Squares (OLS): $\hat{\beta} = (X^tX)^{-1}X^tY$	Gradient descent + backpropagation
Interpretability	High: coefficients β_j have direct meaning	Low: black-box model, weights are hard to interpret
Feature interactions	Must be explicitly specified (e.g., $X_i \cdot X_j$)	Learned automatically via hidden layers
Assumptions	Normality, homoscedasticity, independence of errors, no multicollinearity	No distributional assumptions
Sensitivity to outliers	High (outliers affect OLS significantly)	Moderate (depends on activation and regularization)
Data requirement	Works well with small to medium datasets ($n > p$)	Requires large datasets to generalize well
Computational cost	Low: $O(np^2)$	High: $O(n \cdot \text{epochs} \cdot \text{parameters})$
Theoretical guarantee	Gauss-Markov theorem (BLUE)	Universal Approximation Theorem
Regularization	Ridge, Lasso, ElasticNet	Weight decay, Dropout, Early stopping

Table 2.3: Theoretical comparison between Multiple Linear Regression (MLR) and Neural Network Regression (NNR)

Key Differentiators

- **Representational Power:** MLR is restricted to hyperplanes in $\mathbb{R}^n$. In contrast, the Universal Approximation Theorem (2.6.1) guarantees that NNR can model highly non-linear manifolds, which is essential when the relationship between X and Y exhibits complex curvature.
- **Feature Interaction:** In MLR, interactions between predictors must be explicitly defined (e.g., $X_i \cdot X_j$). NNR learns these interactions automatically through hidden layers, which serve as feature extractors.
- **Statistical Inference:** MLR provides clear statistical inference (t-tests, confidence intervals), allowing for the quantification of uncertainty. NNR is primarily a predictive framework, often treated as a "black box," where interpreting individual weights is significantly more challenging than interpreting regression coefficients.

Chapter 3

Numerical Applications

In this chapter, we will present the practical application of the regression models that were studied previously.

3.1 Dataset Description

This study used a dataset that consists of 4,368 hourly records of meteorological and photovoltaic production data collected from a solar energy system located in Laghouat (Algeria) from February 17 to August 17, 2022 [16].

The main objective is to use this dataset to analyze and predict photovoltaic power generation using two different regression approaches, Multiple Linear Regression (MLR) and Neural Network Regression (NNR).

The dataset contains several explanatory variables related to weather conditions and atmospheric properties that may influence solar energy production. In addition, the dataset includes output variables representing the electrical power generated by photovoltaic panels under different conditions.

Table 3.1 presents the first 30 observations, including all 14 variables.

Table 3.1: First 30 observations of the Laghouat PV dataset (all variables).

Datetime	$\mathbf{Dust_{meas}}$ (g/m²)	$\mathbf{Dust_{recon}}$ (g/m²)	$\mathbf{T_{amb}}$ (°C)	$\mathbf{T_{cell}}$ (°C)	G (W/m²)	RH (%)	Wind (m/s)	Cloud (%)	Precip (mm)	$\mathbf{P_{clean}}$ (W)	$\mathbf{P_{dusty}}$ (W)	$\mathbf{P^{norm}_{clean}}$	$\mathbf{P^{norm}_{dusty}}$
17/02/2022 00:00		0.0000	9.00	9.00	0.00	26	4.70	1	0.00	0.00	0.00	0.0000	0.0000
17/02/2022 01:00		0.0000	8.10	8.10	0.00	33	4.50	0	0.00	0.00	0.00	0.0000	0.0000
17/02/2022 02:00		0.0000	7.10	7.10	0.00	37	3.60	29	0.00	0.00	0.00	0.0000	0.0000
17/02/2022 03:00		0.0000	5.50	5.50	0.00	44	2.60	31	0.00	0.00	0.00	0.0000	0.0000
17/02/2022 04:00		0.0000	4.60	4.60	0.00	49	2.40	15	0.00	0.00	0.00	0.0000	0.0000
17/02/2022 05:00		0.0000	4.00	4.00	0.00	52	2.60	15	0.00	0.00	0.00	0.0000	0.0000
17/02/2022 06:00		0.0000	3.80	3.80	0.00	53	2.30	17	0.00	0.00	0.00	0.0000	0.0000
17/02/2022 07:00		0.0000	4.20	4.20	0.00	50	2.00	0	0.00	0.00	0.00	0.0000	0.0000
17/02/2022 08:00		0.0000	3.50	3.70	7.88	51	2.20	1	0.00	0.77	0.45	0.0096	0.0056
17/02/2022 09:00		0.0000	6.80	8.10	40.14	40	2.00	6	0.00	3.68	3.45	0.0458	0.0430
17/02/2022 10:00		0.0000	10.10	12.40	73.94	32	1.50	0	0.00	6.59	6.92	0.0821	0.0862
17/02/2022 11:00		0.0000	12.50	40.70	902.20	29	1.40	1	0.00	70.81	72.13	0.8819	0.8984
17/02/2022 12:00		0.0000	14.80	44.50	948.87	27	0.40	0	0.00	72.97	75.12	0.9088	0.9356
17/02/2022 13:00		0.0000	17.20	45.70	913.26	24	3.30	0	0.00	70.36	70.52	0.8763	0.8783
17/02/2022 14:00	0.00	0.0000	18.40	43.40	801.08	23	3.90	0	0.00	62.54	63.79	0.7789	0.7945
17/02/2022 15:00		0.0000	19.10	38.60	623.56	22	3.80	0	0.00	49.05	51.09	0.6109	0.6363
17/02/2022 16:00		0.0001	19.40	31.90	398.54	22	3.80	0	0.00	33.11	33.63	0.4124	0.4189
17/02/2022 17:00		0.0001	19.20	24.00	155.15	23	4.10	0	0.00	13.25	13.53	0.1650	0.1685
17/02/2022 18:00		0.0002	18.50	18.80	10.14	24	4.00	0	0.00	0.88	0.84	0.0110	0.0105
17/02/2022 19:00		0.0003	16.40	16.40	0.00	28	2.40	0	0.00	0.00	0.00	0.0000	0.0000
17/02/2022 20:00		0.0005	14.60	14.60	0.00	32	2.10	0	0.00	0.00	0.00	0.0000	0.0000
17/02/2022 21:00		0.0006	13.80	13.80	0.00	33	3.20	0	0.00	0.00	0.00	0.0000	0.0000
17/02/2022 22:00		0.0008	12.30	12.30	0.00	37	2.90	0	0.00	0.00	0.00	0.0000	0.0000
17/02/2022 23:00		0.0010	10.80	10.80	0.00	42	2.50	0	0.00	0.00	0.00	0.0000	0.0000
18/02/2022 00:00		0.0013	9.90	9.90	0.00	44	1.70	0	0.00	0.00	0.00	0.0000	0.0000
18/02/2022 01:00		0.0015	12.10	12.10	0.00	41	0.60	0	0.00	0.00	0.00	0.0000	0.0000
18/02/2022 02:00		0.0018	11.10	11.10	0.00	43	1.20	0	0.00	0.00	0.00	0.0000	0.0000
18/02/2022 03:00		0.0022	9.40	9.40	0.00	48	1.50	0	0.00	0.00	0.00	0.0000	0.0000
18/02/2022 04:00		0.0025	8.10	8.10	0.00	53	1.50	0	0.00	0.00	0.00	0.0000	0.0000
18/02/2022 05:00		0.0029	6.90	6.90	0.00	57	1.70	0	0.00	0.00	0.00	0.0000	0.0000

Note: $\text{Dust}_{\text{meas}}$ = measured dust density, $\text{Dust}_{\text{recon}}$ = reconstructed dust density.

3.2 Data preprocessing

To ensure meaningful predictions, only **daytime hours** are retained, i.e., records with global tilted irradiation $G > 10\,W/m^2$. This filtering step reduces the dataset to **2262 observations**. We select **5 input variables** (solar irradiation, ambient temperature, relative humidity, wind speed, and dust density) and one target variable, which is the power output of the soiled panel P_{dusty} (see Table 3.6).

Variable	Description
Datetime	Date and time of the observation recording.
G (W/m^2)	Global solar irradiance measured on the plane of the photovoltaic panels. It represents the amount of solar energy received per square meter.
T_{amb} (°C)	Ambient temperature surrounding the photovoltaic system.
RH (%)	Relative humidity of the air expressed as a percentage.
Wind (m/s)	Wind speed measured in meters per second.
Dust (g/m^2)	Dust density accumulated on the photovoltaic panel surface expressed in grams per square meter.
P_{dusty} (W)	Electrical power generated by the dusty photovoltaic panel measured in watts. This variable is considered as the target variable in this study.

Table 3.2: Description of the variables used in the photovoltaic dataset

3.2.1 Train-Test Split

The dataset was divided into two subsets: a training set and a testing set.

- 80% of the data represent the training set, which was used to train the models (1809 observations).
- 20% was allocated to evaluate its predictive performance on unseen data (453 observations).

3.2.2 Data Normalization for NNR

Neural networks are sensitive to the scale of input features. Features with large numerical ranges may dominate those with smaller ranges, leading to unstable training and slow convergence. To avoid this issue, we apply **standardization** to all input variables and the target variable.

Mathematical Formula

For each feature x, the standardized value x' is:

$$x' = \frac{x - \mu}{\sigma}$$

where:

- μ is the mean of the feature over the training set,
- σ is the standard deviation of the feature over the training set.

This transformation ensures that each feature has a mean of 0 and a standard deviation of 1. To avoid data leakage, we apply the same mean and standard deviation to the training set.

Implementation

In this study, we use the `StandardScaler` from the `sklearn.preprocessing` module. The scaler is fitted on the training data only and then applied to both training and test sets:

```
from sklearn.preprocessing import StandardScaler

scaler_X = StandardScaler()
scaler_y = StandardScaler()

X_train_scaled = scaler_X.fit_transform(X_train)
X_test_scaled  = scaler_X.transform(X_test)

y_train_scaled = scaler_y.fit_transform(y_train.reshape(-1,1)).flatten
    ()
y_test_scaled  = scaler_y.transform(y_test.reshape(-1,1)).flatten()
```

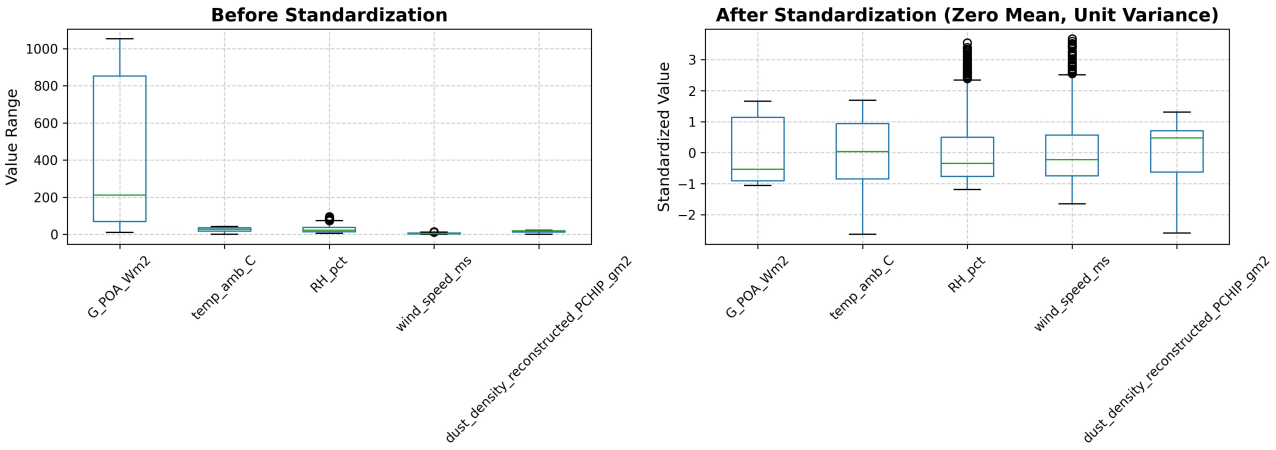

Figure 3.1: The distribution of input features before and after standardization.

Remark 3.2.1. *The input variables are standardized before training the neural network regression model to improve the learning process. Additionally, data augmentation helps the gradient descent algorithm converge more efficiently and improves the behavior of activation functions during training.*

3.3 Multiple Linear Regression Model

The MLR model can be written as follows:

$$P_{\text{dusty}} = \beta_0 + \beta_1 G + \beta_2 T_{\text{amb}} + \beta_3 RH + \beta_4 Wind + \beta_5 Dust + \varepsilon$$

The main objective is to estimate the power generated by the photovoltaic panel (Y) using several explanatory variables: solar irradiation (X_1), ambient temperature (X_2), relative humidity (X_3), wind speed (X_4), and dust density (X_5).

3.3.1 Python Code of MLR

We model, in **Python**, the MLR model using the `LinearRegression` function from the `Scikit-learn` library.

```
# Train MLR Model
mlr = LinearRegression()
mlr.fit(X_train_scaled, y_train_scaled)

# MLR Predictions (in original scale)
y_pred_mlr_scaled = mlr.predict(X_test_scaled)
y_pred_mlr = scaler_y.inverse_transform(y_pred_mlr_scaled.reshape
    (-1, 1)).flatten()

# MLR Evaluation
mse_mlr = mean_squared_error(y_test, y_pred_mlr)
rmse_mlr = np.sqrt(mse_mlr)
mae_mlr = mean_absolute_error(y_test, y_pred_mlr)
r2_mlr = r2_score(y_test, y_pred_mlr)
```

3.3.2 Performance Evaluation of the MLR Model

MSE	11.8821
RMSE	3.4470
MAE	2.7085
R²	0.9780

Table 3.3: MLR results

These results show that the MLR model is capable of capturing the general linear relationship. However, due to the nonlinear nature of the data, some prediction errors remain significant.

Figure 3.2 illustrates the comparison between the real and the predicted values. The blue color represents real value, and the red color represents predicted value.

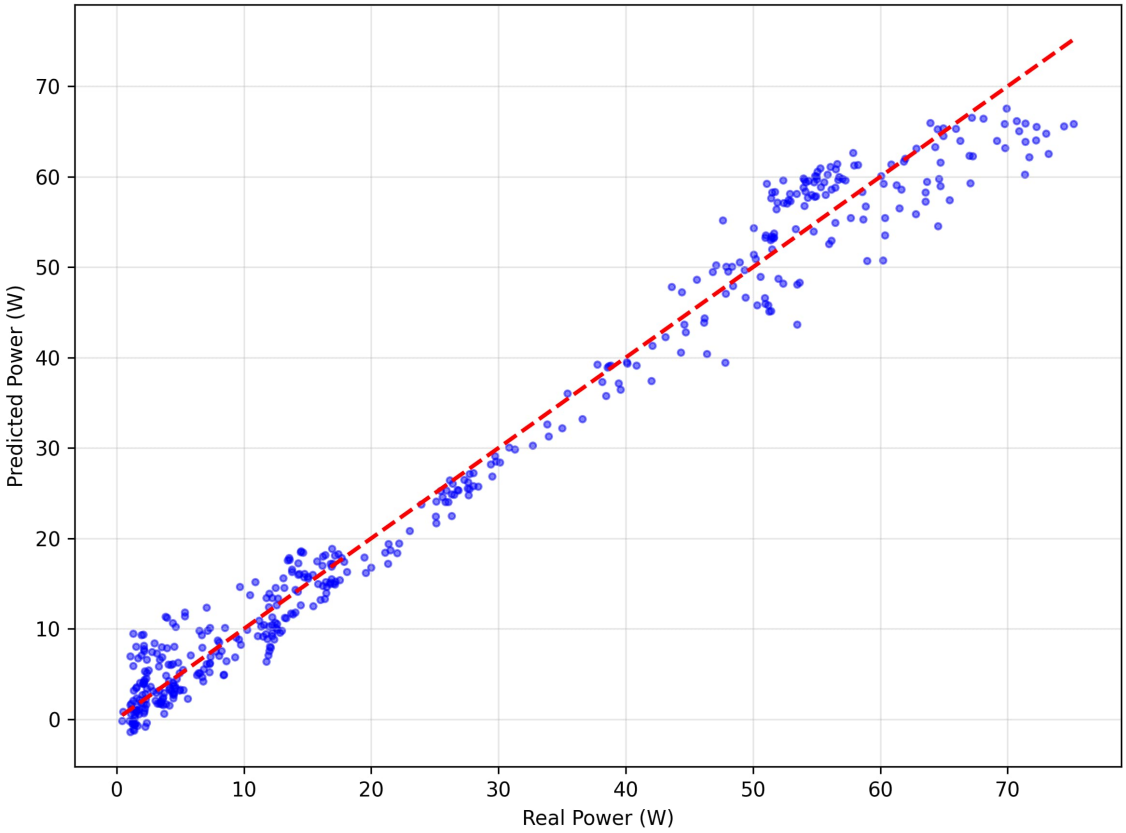

Figure 3.2: MLR: Real vs Predicted Power Output

Interpretation of coefficient

Regression Coefficients:

	Variable	Coefficient
0	**G**	0.993734
1	**Temp**$_{amb}$	−0.098180
2	**RH**	−0.056999
3	**Wind**	0.009706
4	**Dust**	−0.086355

Here, we explain that as the temperature (T_{amb}) increases, it leads to a decrease in photovoltaic power (P_{dusty}) output. Dust also negatively affects the results, while solar irradiance (G) increases production.

3.4 Neural Network Regression Model

Neural Network Regression (NNR) was used to model the nonlinear relationship between meteorological variables and photovoltaic power output.

3.4.1 Architecture of NN

In this study, we used a neural network consisting of an input layer, two hidden layers, and one output layer, where the weather variables are received by the input layer, while the hidden layers perform nonlinear transformations using activation functions. The output layer predicts the photovoltaic power output. In the hidden layers, we used the Rectified Linear Unit (ReLU) activation function due to its efficiency and good performance in regression problems.

3.4.2 Training Procedure

We trained the neural network using the Adam optimizer and the Mean Squared Error (MSE) loss function. The training process was performed over several epochs using the training dataset after standardization.

Figure 3.3 showed a gradual decrease in both the training and validation loss curves, indicating that the neural network learned the relationship between the input variables and photovoltaic power generation. The validation loss remained close to the training loss, suggesting that the model generalized well without overfitting significantly.

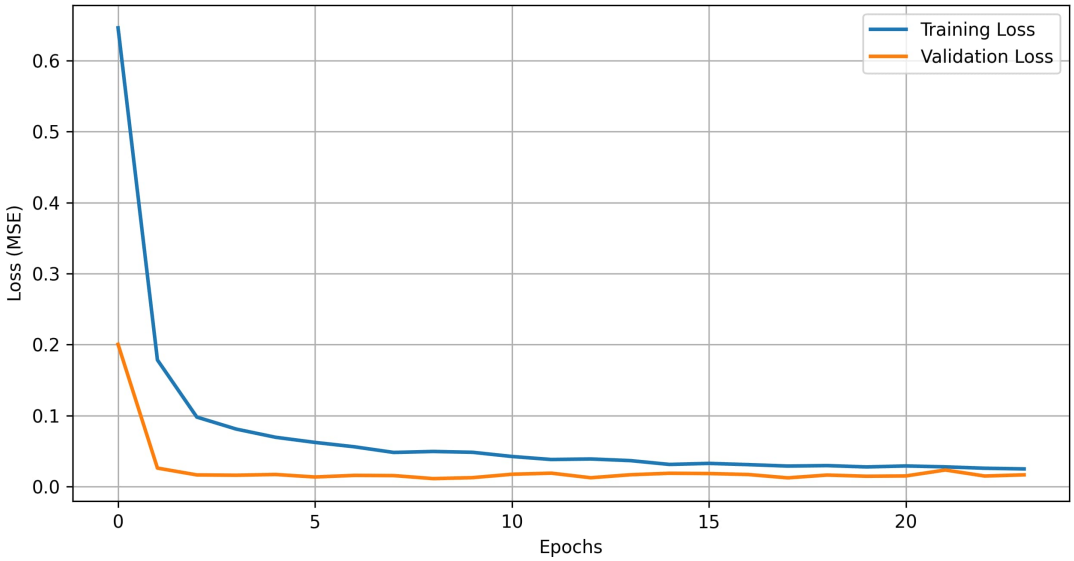

Figure 3.3: Training and Validation Loss

3.4.3 Python Code of NNR

We model NNR in Python as follows:

```
  nnr = Sequential([
    Input(shape=(X_train.shape[1],)),
    Dense(64, activation='relu'),
    Dropout(0.2),
    Dense(32, activation='relu'),
    Dropout(0.2),
    Dense(16, activation='relu'),
    Dense(1)
  ])

# Compile Model
   nnr.compile(
    optimizer='adam',
    loss='mse',
    metrics=['mae']
   )
   nnr.summary()

# Early Stopping Callback
   early_stop = EarlyStopping(
    monitor='val_loss',
    patience=15,
    restore_best_weights=True
   )

# Train Model
   history = nnr.fit(
    X_train_scaled,
    y_train_scaled,
    epochs=200,
    batch_size=32,
    validation_split=0.2,
    callbacks=[early_stop],
    verbose=1
   )
```

Remark 3.4.1. *During training, there is something called an* ***Epoch****, which corresponds to one complete pass of the training dataset through the neural network during the learning process (used in this study: 200 epochs). During each epoch, the model updates its weights in order to minimize the prediction error.*

3.4.4 Results of the NNR Model

MSE	7.8577
RMSE	2.8032
MAE	2.2589
R^2	0.9855

Table 3.4: NNR results

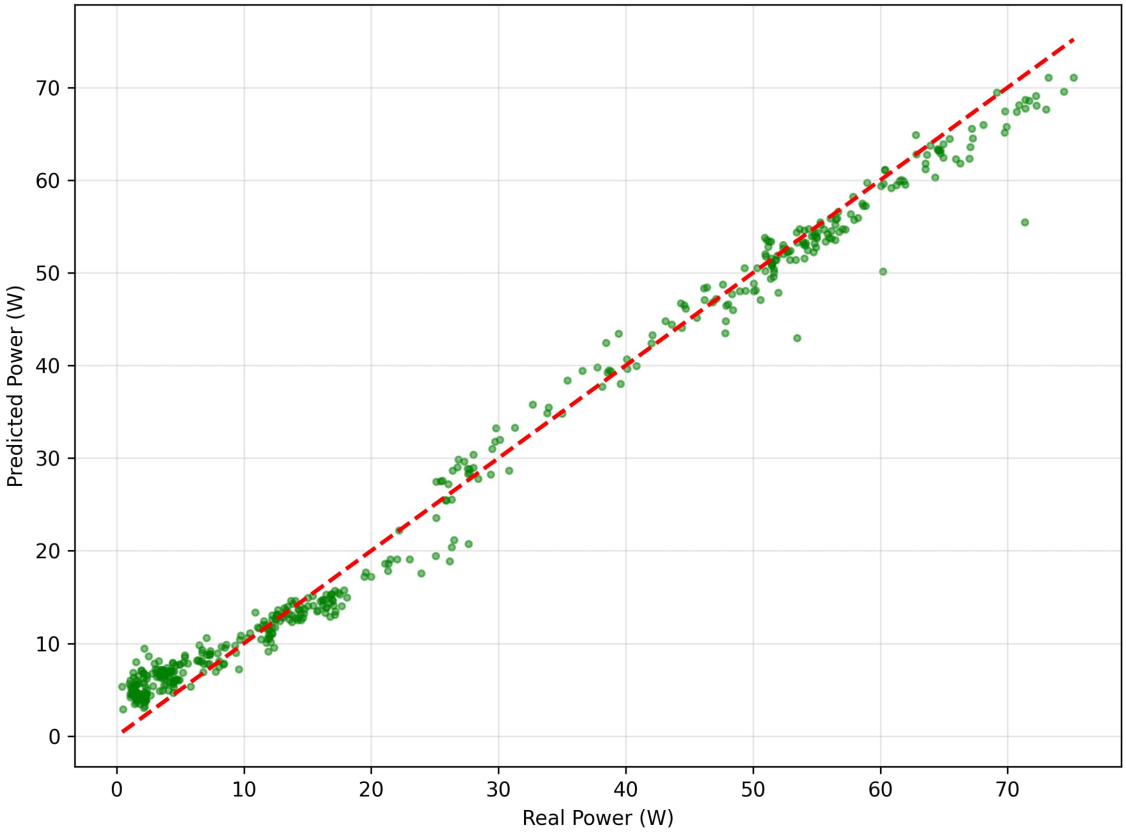

Figure 3.4: NNR: Real vs Predicted Power Output

These results show that the neural network model successfully captured the nonlinear relationships between the meteorological variables and the photovoltaic power generation.

3.5 Comparison Between MLR and NNR

This section is important as we will compare the predictive performance of the MLR model and the NNR model using several evaluation metrics. The goal of this is to determine which model provides more accurate predictions for photovoltaic power generation.

Metric	MLR	NNR
MSE	11.8821	7.8577
RMSE	3.4470	2.8032
MAE	2.7085	2.2589
R^2	0.9780	0.9855

Table 3.5: Performance comparison between MLR and NNR models

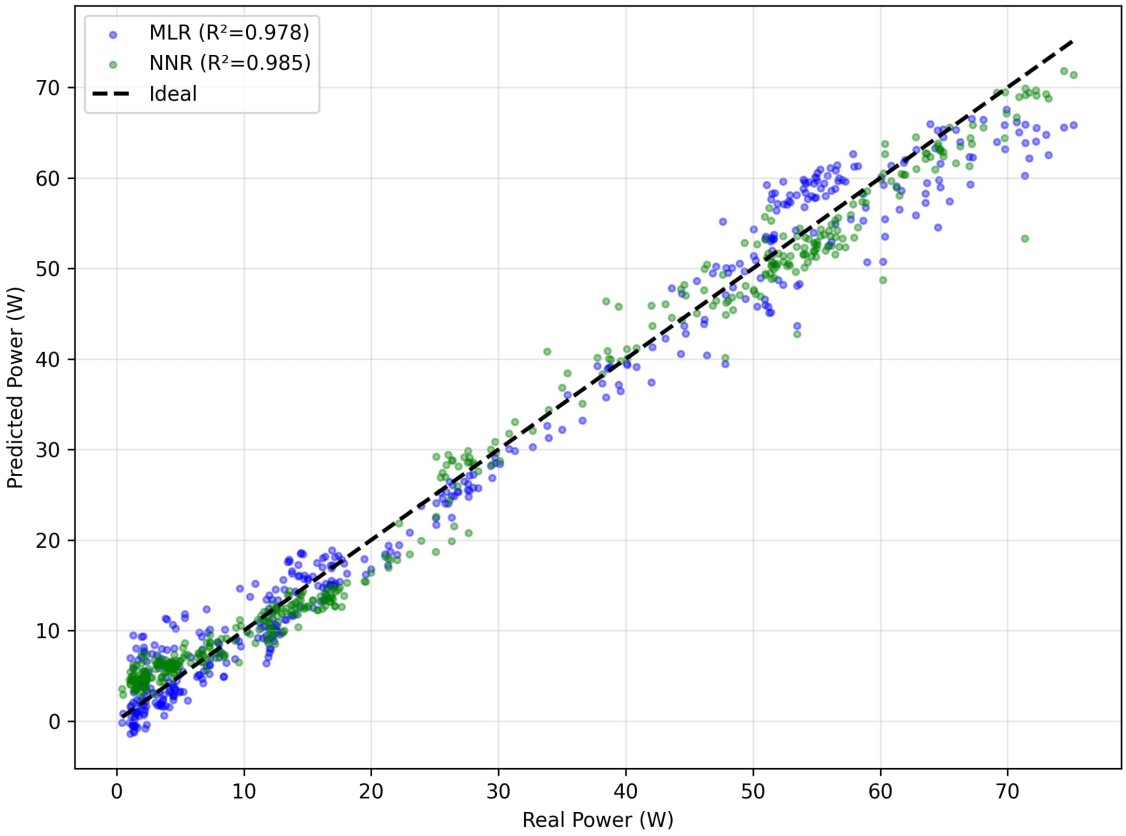

Figure 3.5: Comparison Between MLR and NNR

The comparison results show that the NNR model achieved better predictive performance than the MLR model. The NNR model produced lower error values in terms of MSE, RMSE, and MAE, while achieving a higher coefficient of determination (R^2).
These results indicate that the neural network was more effective in capturing the complex nonlinear relationships between the meteorological variables and the photovoltaic power output.

In contrast, the Multiple Linear Regression model is limited to linear relationships, which reduces its prediction accuracy for this type of data.
Although the MLR model remains simple, interpretable, and computationally efficient, the NNR model provides more accurate and reliable predictions for photovoltaic energy generation.

In conclusion, the results of these comparisons between the two models showed that NNR is more suitable for modeling photovoltaic energy production due to its strong ability to learn nonlinear patterns from meteorological data. [17]

Table 3.6: The first 27 observations of the Laghouat PV dataset were used for MLR and NNR regression (daytime hours only, $G > 10\,W/m^2$).

Datetime	**G (W/m²)**	**T_{amb} (°C)**	**RH (%)**	**Wind (m/s)**	**Dust (g/m²)**	**P_{dusty} (W)**
17/02/2022 09:00	40.14	6.80	40	2.00	0.0000	3.45
17/02/2022 10:00	73.94	10.10	32	1.50	0.0000	6.92
17/02/2022 11:00	902.20	12.50	29	1.40	0.0000	72.13
17/02/2022 12:00	948.87	14.80	27	0.40	0.0000	75.12
17/02/2022 13:00	913.26	17.20	24	3.30	0.0000	70.52
17/02/2022 14:00	801.08	18.40	23	3.90	0.0000	63.79
17/02/2022 15:00	623.56	19.10	22	3.80	0.0000	51.09
17/02/2022 16:00	398.54	19.40	22	3.80	0.0001	33.63
17/02/2022 17:00	155.15	19.20	23	4.10	0.0001	13.53
18/02/2022 09:00	40.90	7.70	54	1.70	0.0046	3.83
18/02/2022 10:00	74.78	10.70	44	1.20	0.0051	7.08
18/02/2022 11:00	905.61	13.20	37	1.00	0.0056	72.02
18/02/2022 12:00	952.10	15.70	33	2.00	0.0062	74.40
18/02/2022 13:00	916.26	17.60	29	2.50	0.0068	70.71
18/02/2022 14:00	803.80	18.80	26	3.00	0.0074	62.78
18/02/2022 15:00	625.92	19.50	24	2.90	0.0080	51.57
18/02/2022 16:00	400.48	19.80	23	2.60	0.0087	34.14
18/02/2022 17:00	156.51	19.60	24	2.60	0.0093	13.57
19/02/2022 09:00	41.67	8.40	48	1.40	0.0239	3.96
19/02/2022 10:00	75.64	11.10	40	1.60	0.0250	7.12
19/02/2022 11:00	885.83	14.20	30	3.10	0.0262	69.63
19/02/2022 12:00	470.25	16.20	28	4.90	0.0274	39.43
19/02/2022 13:00	283.36	17.30	27	5.00	0.0286	24.96
19/02/2022 14:00	164.03	18.10	26	4.80	0.0298	14.45
19/02/2022 15:00	141.13	18.70	25	4.40	0.0311	12.43
19/02/2022 16:00	105.08	18.90	25	4.40	0.0324	9.47
19/02/2022 17:00	59.62	18.50	25	4.90	0.0337	5.06

Note: Only daytime hours ($G > 10\,W/m^2$) are retained for regression modeling.

General Conclusion

This work focused on the study and application of Multiple Linear Regression (MLR) and Neural Network Regression (NNR) for photovoltaic power prediction using meteorological data.

First, the theoretical foundations of both regression models were presented, including their mathematical formulations, assumptions, and main evaluation methods. Then, a practical application was carried out using real data in order to compare the predictive performance of the two approaches.

The obtained results showed that both models were able to predict photovoltaic power generation with satisfactory accuracy. However, the Neural Network Regression model achieved better predictive performance, particularly in modeling the nonlinear relationships between the meteorological variables and the photovoltaic power output.

Although Multiple Linear Regression remains simple and interpretable, Neural Network Regression proved to be more effective for this type of application. Overall, this study demonstrates the importance of regression methods in photovoltaic energy prediction and highlights the efficiency of neural network models for improving prediction accuracy.

Nomenclature

Abbreviations

Abbreviation	Definition
MLR	Multiple Linear Regression
OLS	Ordinary Least Squares
BLUE	Best Linear Unbiased Estimator
ANOVA	Analysis of Variance
MSE	Mean Squared Error
RMSE	Root Mean Squared Error
MAE	Mean Absolute Error
RSS	Residual Sum of Squares
ESS	Explained Sum of Squares
TSS	Total Sum of Squares
R^2	Coefficient of Determination
ANN	Artificial Neural Network
FNN	Feedforward Neural Network
SLFN	Single Hidden-Layer Feedforward Neural Network
MLP	Multilayer Perceptron
NNR	Neural Network Regression
ReLU	Rectified Linear Unit
UAT	Universal Approximation Theorem
ERM	Empirical Risk Minimization
Adam	Adaptive Moment Estimation

Table 3.7: List of abbreviations used in this thesis

Mathematical Symbols

Symbol	Definition
n	Number of observations
p	Number of explanatory variables
Y_i	Dependent variable (response) for observation i
X_{ij}	j-th explanatory variable for observation i
β_0	Intercept (constant term)
β_j	Regression coefficient associated with X_j
ε_i	Random error (noise) for observation i
Y	Dependent variable vector $(n \times 1)$
X	Explanatory variables matrix $(n \times (p+1))$
X^t	Transposed matrix
β	Parameter vector $((p+1) \times 1)$
ε	Random error vector $(n \times 1)$
$\hat{\beta}$	OLS estimator of β
$\tilde{\beta}$	Arbitrary linear unbiased estimator of β
$\hat{Y}$	Predicted (fitted) values
σ_ε^2	Error variance
$\hat{\sigma}_\varepsilon^2$	Unbiased estimator of σ_ε^2
e_i	Residual for observation i $(e_i = Y_i - \hat{Y}_i)$
RSS	Residual sum of squares $(\sum e_i^2)$
ESS	Explained sum of squares
TSS	Total sum of squares
R^2	Coefficient of determination
MSE	Mean squared error
RMSE	Root mean squared error
MAE	Mean absolute error
$\mathbb{E}(\cdot)$	Expected value
$\mathrm{Var}(\cdot)$	Variance
$\mathrm{Cov}(\cdot,\cdot)$	Covariance
$t_{n-p-1}^{1-\alpha/2}$	Student's t quantile with $n-p-1$ degrees of freedom
χ_{n-p-1}^{γ}	Chi-square quantile with $n-p-1$ degrees of freedom
$F_{(p),(n-p-1)}$	Fisher-Snedecor distribution
H_0	Null hypothesis
H_1	Alternative hypothesis

Table 3.8: List of mathematical symbols used in chapter 01

Symbol	Definition
X	Input vector ($\mathbb{R}^n$)
W	Weight vector ($\mathbb{R}^n$)
b	Bias term ($\mathbb{R}$)
z	Pre-activation value ($z = W^t X + b$)
$\sigma(\cdot)$	Activation function
y	Output of a neuron ($y = \sigma(z)$)
m	Number of neurons in hidden layer
$W^{(l)}$	Weight matrix at layer l
$B^{(l)}$	Bias vector at layer l
$H^{(l)}$	Output vector of layer l (activation values)
$Z^{(l)}$	Pre-activation vector at layer l
L	Total number of layers
α_j	Weight connecting hidden neuron j to output neuron
c	Bias of output neuron
$f_\theta(x)$	Neural network function with parameters θ
$\mathcal{L}(\theta)$	Loss function (empirical risk)
$\nabla\mathcal{L}(\theta)$	Gradient of the loss function
η	Learning rate
$\theta^{(T)}$	Parameters at iteration T
$\delta^{(l)}$	Error term at layer l (backpropagation)
$\odot$	Hadamard product (element-wise multiplication)
λ	Regularization hyperparameter
$\|W^{(l)}\|_F^2$	Frobenius norm squared of weight matrix
$m(x)$	Regression function ($m(x) = \mathbb{E}[Y\|X = x]$)
$\text{ReLU}(z)$	Rectified Linear Unit ($\max(0, z)$)
$\tanh(z)$	Hyperbolic tangent activation function
$\text{Sigmoid}(z)$	Logistic sigmoid function ($1/(1 + e^{-z})$)

Table 3.9: List of mathematical symbols used in chapter 02

References

[1] APXML. Calculus and the chain rule (backpropagation & advanced optimization). APXML Course, 2026. Accessed: 2026-06-08.

[2] Christopher M. Bishop. *Pattern recognition and machine learning by Christopher M. Bishop*, volume 350. Springer Science+ Business Media, LLC Berlin, Germany:, 2006.

[3] Douglas C and Peck, Montgomery, Elizabeth A and Vining, G Geoffrey. *Introduction to linear regression analysis*. John Wiley & Sons, 2021.

[4] Garcia Cabello, Julia. Mathematical neural networks. *Axioms*, 11(2):80, 2022.

[5] Geman, Stuart and Bienenstock, Elie and Doursat, René. Neural networks and the bias/-variance dilemma. *Neural Computation*, 4(1):1–58, 1992.

[6] George, Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989.

[7] Glorot, Xavier and Bengio, Yoshua. Understanding the difficulty of training deep feedforward neural networks. *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pages 249–256, 2010.

[8] Goodfellow, Ian and Bengio, Yoshua and Courville, Aaron. *Deep Learning*. MIT Press, Cambridge, MA, 2016.

[9] Hastie, Trevor and Tibshirani, Robert and Friedman, Jerome H and Friedman, Jerome H. *The elements of statistical learning: data mining, inference, and prediction*, volume 2. Springer, 2009.

[10] Hornik, Kurt. Approximation capabilities of multilayer feedforward networks. *Neural networks*, 4(2):251–257, 1991.

[11] Kuo, Biing-Shen. Asymptotics of ml estimator for regression models with a stochastic trend component. *Econometric Theory*, 15(1):24–49, 1999.

[12] Kutner, M. H. and Nachtsheim, C. J. and Neter, J. *Applied Linear Regression Models*. McGraw-Hill, 2004.

[13] Robert V. Hogg and Allen T. Craig. *Introduction to Mathematical Statistics*. Macmillan, New York, 4th edition, 1978.

[14] Rosenblatt, Frank. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

[15] Rumelhart, David E and Hinton, Geoffrey E and Williams, Ronald J. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

[16] Mohammed Seghir. Hourly PV power and dust accumulation dataset – Laghouat, Algeria, February–August 2022, 2026. Published: 11 May 2026.

[17] Soteris A. Kalogirou. Greek long-term energy consumption prediction using artificial neural networks. *Energy*, 34(3):287–294, 2009.

[18] Stanford CS231n Course Staff. Backpropagation, intuitions. https://cs231n.github.io/optimization-2/, 2019. Course Notes, Accessed: 2026-05-03.

[19] SuperAnnotate. Activation functions in neural networks, 2023. Accessed: 2026-03-11.

[20] Vapnik, Vladimir N. *The Nature of Statistical Learning Theory*. Springer-Verlag, 1995.

[21] Zilliz. Activation functions, 2024. Accessed: 2026-03-12.