
Łódź University of Technology

International Faculty of Engineering

BSc in Computer Science

Course: Numerical Methods

*Lecturer: Prof. Paolo Di Barba*

*Academic Year: 2023-24*

# Decision Trees vs Deep Feedforward Neural Networks in Regression Problems

*Candidate: Mateusz Cieśliński*
*Student Number: 246927*

# Contents



# 1 Introduction

In the past decade, artificial intelligence (AI) has witnessed significant advancements, accelerating and streamlining solutions across various domains, from automation to the pace of work, and more recently, even in chemical discovery. A pivotal driver behind this progress has been the modern approach to function approximation through neural networks. These sophisticated computational models have revolutionized our ability to generalize function values, particularly in scenarios involving numerous v ariables that pose challenges for human comprehension and solution. Consequently,

neural networks have become instrumental in deploying technologies for tasks such as image segmentation, sound recognition, and image classification.

However, the efficacy of neural networks comes at a cost – notably, extensive training periods required to optimize model structures. Consequently, it prompts us to question how neural network models perform in comparison to conventional tabular data, particularly when juxtaposed against established methods like decision trees, renowned for their adeptness in handling such data formats.

This introduction sets the stage for a comprehensive examination and comparison between decision trees and deep feedforward neural networks in the context of regression problems. Through this comparative analysis, we aim to discern the strengths and limitations of each approach, shedding light on their respective suitability for diverse regression tasks.

## 1.1 Goals

The main objectives of the following article are:

1. To present the differences in the principle of operation of decision trees and neural networks.
2. To present the differences in the results on different datasets.
3. To present the amount of time required to obtain a certain result.

## 1.2 Environment

The experiments will be implemented in Python using [1] Anaconda environment and [2] Jupyter Notebook. The [3] scikit-learn library will be utilized for decision tree implementation, while [4] PyTorch framework will be used for deep learning models.

# 2 Basic Concepts

## 2.1 What is Machine Learning?

Machine learning, a subset of artificial intelligence (AI) and computer science, involves the utilization of data and algorithms to replicate the human learning process,

enhancing its precision over time. It encompasses the development and examination of statistical algorithms capable of learning from data, making predictions, and adapting to novel scenarios without explicit instructions. [1] [2]

## 2.2 Deep Learning

### 2.2.1 What is Deep Learning?

Deep learning, a subset of machine learning, employs neural networks consisting of three or more layers. These networks, inspired by the human brain, albeit far from replicating its capabilities, aim to emulate learning processes by analyzing extensive datasets. While a single-layer neural network can provide approximate predictions, additional hidden layers enhance accuracy through optimization and refinement. Deep learning teaches computers to learn from examples, mirroring human learning patterns. It serves as a fundamental technology behind driverless cars, enabling tasks such as stop sign recognition and pedestrian detection. Moreover, it powers voice control in consumer electronics like smartphones, tablets, and smart speakers. With its remarkable capabilities, deep learning has garnered significant attention, as it achieves unprecedented results previously deemed unattainable. [3] [4]

### 2.2.2 Where Deep Learning is used?

Deep learning is currently utilized in various fields including computer vision for tasks like object detection and facial recognition, natural language processing for translation and sentiment analysis, and speech recognition for virtual assistants and transcription services. It's also applied in medical imaging, autonomous vehicles, and quality control in manufacturing, among other areas. [5]

### 2.2.3 Feedforward Neural Networks

In this section, we will delve into the construction of a simple deep neural network using feedforward neural networks. Below, you will find a comprehensive explanation of the theory and mathematics behind feedforward neural networks.

#### Intuition and Mathematics

[1] https://www.ibm.com/topics/machine-learning
[2] https://en.wikipedia.org/wiki/Machine_learning
[3] https://www.ibm.com/topics/deep-learning
[4] https://www.mathworks.com/discovery/deep-learning.html
[5] https://bernardmarr.com/what-is-deep-learning-ai-a-simple-guide-with-8-practical-examples/

**Definition** According to [6], a feedforward neural network is one of the two broad types of artificial neural networks, characterized by the direction of the flow of information between its layers. Its flow is uni-directional, meaning that information in the model flows in only one direction—from the input nodes, through the hidden nodes (if any), and to the output nodes—without any cycles or loops.

**Linear Neural Network** Primarily in this experiment, we will utilize a linear neural network to construct our deep learning model, enabling a comparison with ensemble methods.

**Perceptron Algorithm** The perceptron algorithm is founded on the concept of a linear classifier, which can be represented by a linear equation in the form of $y = wx + b$. Here, $w$ represents the weights or coefficients of the input features, $x$ represents the input features, and $b$ represents the bias term.

The primary objective of the perceptron is to determine the optimal values of $w$ and $b$ that can effectively classify input data into one of two classes, represented by 1 or -1. To achieve this, the perceptron algorithm employs an iterative process to adjust the weights and bias based on the error between the predicted output and the true output.

The fundamental steps of the perceptron algorithm are as follows:

1. Initialize the weights and bias to random values.
2. For each training example in the dataset, make a prediction using the current weights and bias.
3. Calculate the error between the predicted output and the true output.
4. Update the weights and bias based on the error and a learning rate.
5. Repeat steps 2–4 until the model converges or a maximum number of iterations is reached.

The update step is where the math comes into play. To update the weights and bias, we use the following equations:

$$w_{\text{new}} = w_{\text{old}} + \text{learning\_rate} \times \text{error} \times x \tag{1}$$

$$b_{\text{new}} = b_{\text{old}} + \text{learning\_rate} \times \text{error} \tag{2}$$

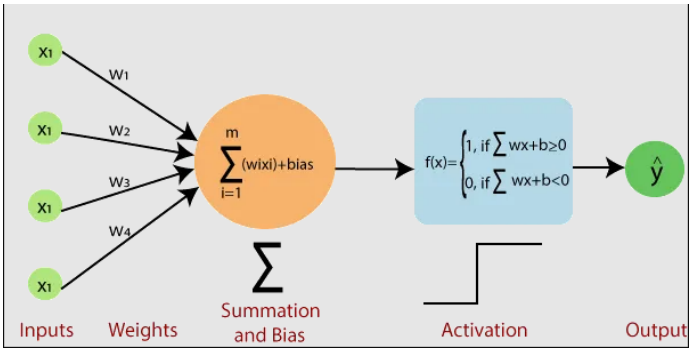


Figure 1: Perceptron Algorithm Update Step

In these equations, the learning rate is a hyperparameter that determines the size of the update step. A smaller learning rate will result in a slower convergence, but a higher learning rate may result in the model overshooting the optimal weights and bias.
[6]

## 2.3 Decision Trees

### 2.3.1 What is Decision Tree?

Decision Trees (DTs) represent a non-parametric approach in supervised learning, serving for both classification and regression tasks. The primary objective is to construct a predictive model capable of estimating the target variable's value by discerning straightforward decision rules derived from the features within the dataset. In essence, a decision tree can be visualized as an approximation that discretizes the data into distinct segments characterized by constant values. [7] [8]

### 2.3.2 Where Decision Trees are used?

Decision trees are widely employed across various domains for intuitive decision-making based on data analysis. They find application in daily decision-making, healthcare diagnostics, financial analysis, customer relationship management, quality control in manufacturing, fraud detection, and recommendation systems for per-

[6] https://medium.com/@ayush260201/maths-behind-neural-network-simplified-5f49594def0d
[7] https://scikit-learn.org/stable/modules/tree.html
[8] https://en.wikipedia.org/wiki/Decision_tree

sonalized content delivery. [9]

### 2.3.3 RandomForestRegressor

The *RandomForestRegressor*, a variant of decision trees, plays a pivotal role in our experimental setup. Below, we delve into the mathematical formulations and key concepts crucial for a thorough understanding, essential for the successful execution of our experiments. All explanations provided are directly related to the implementation in the Scikit-Learn Library.

According to [5], Random forests (RF) create many individual decision trees during training. Predictions from all trees are aggregated to make the final prediction; for regression, this typically involves averaging the predictions from all trees. As they utilize a collection of results to make a final decision, they are referred to as ensemble techniques.

#### Intuition and Mathematics

**Feature Importance** Feature importance is determined by assessing the reduction in node impurity, which is then weighted by the probability of reaching that node. This probability is computed by dividing the number of samples reaching the node by the total number of samples. A higher probability indicates greater feature importance.

For each decision tree, Scikit-Learn calculates a node's importance using Gini Importance, assuming only two child nodes (binary tree):

$$ni_j = w_j C_j - w_{left(j)} C_{left(j)} - w_{right(j)} C_{right(j)} \tag{3}$$

Where:

- $ni_j$: Importance of node $j$
- $w_j$: Weighted number of samples reaching node $j$
- $C_j$: Impurity value of node $j$
- $left(j)$: Child node from left split on node $j$
- $right(j)$: Child node from right split on node $j$

[9] https://medium.com/@diehardankush/why-practical-applications-of-decision-trees-ae09e04b2b16

The importance for each feature on a decision tree is then calculated as:

$$fi_i = \frac{\sum_{j \,:\, node\ j\ splits\ on\ feature\ i} ni_j}{\sum_{k \,\in\, all\ nodes} ni_k} \tag{4}$$

Where:

- $fi_i$: Importance of feature $i$
- $ni_j$: Importance of node $j$

These values can then be normalized to a range between 0 and 1 by dividing by the sum of all feature importance values. The final feature importance at the Random Forest level is the average over all the trees:

$$RFfi_i = \frac{\sum_{j \,\in\, all\ trees} normfi_{ij}}{T} \tag{5}$$

Where:

- $RFfi_i$: Importance of feature $i$ calculated from all trees in the Random Forest model
- $normfi_{ij}$: Normalized feature importance for $i$ in tree $j$
- $T$: Total number of trees

# 3 Datasets

This section introduces the datasets utilized in our research. Data preprocessing steps are excluded from this discussion as they are not within the scope of this paper.

## 3.1 Concrete Compressive Strength Dataset

**Description:** The Concrete Compressive Strength dataset provides crucial insights into the behavior of concrete, the primary material in civil engineering. Comprising 1030 instances with 8 quantitative input variables and 1 quantitative output variable, this dataset facilitates the study of concrete compressive strength—a nonlinear function of age and ingredient proportions.

**Source and Features:** `https://www.kaggle.com/datasets/maajdl/yeh-concret-data`

### 3.2 Crab Age Dataset

**Description:** The dataset for this competition (both train and test) was generated from a deep learning model trained on the Crab Age Prediction dataset. Feature distributions are close to, but not exactly the same, as the original.

**Source and Features:** `https://www.kaggle.com/competitions/linear-regression-winter-2overview`

## 4 Methodology

Our experiment will follow a systematic approach where we conduct five tests on each dataset. In each test, we increment the number of epochs to train our deep learning architecture. The models will be trained using a batch size of 32 and will be trained for 10, 50, 100, 200, and 500 epochs. Subsequently, we compare the results based on various criteria such as time taken and the mean squared error (MSE). The MSE is calculated using the following mathematical formula:

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

Where:

- $n$ is the total number of observations,
- $y_i$ is the actual value of the target variable for the $i^{th}$ observation,
- $\hat{y}_i$ is the predicted value of the target variable for the $i^{th}$ observation.

### 4.1 Model Architecture

The architecture of our deep learning model is as follows:

```
Linear
ReLU
Linear
ReLU
Linear
ReLU
Linear
ReLU
```

```
Linear
ReLU
Linear
ReLU
Linear
```

This architecture consists of several fully connected layers (Linear) followed by rectified linear unit activation functions (ReLU).

## 4.2 Rectified Linear Unit (ReLU)

ReLU is an activation function commonly used in deep neural networks. It introduces non-linearity to the network, allowing it to learn complex patterns in the data. ReLU function is defined as:

$$f(x) = \begin{cases} 0 & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases}$$

ReLU has several advantages, including computational efficiency and avoiding the vanishing gradient problem. It is widely used in practice due to its simplicity and effectiveness. [10]

## 4.3 Performance Measurement

The performance of our models will be measured using two main criteria: the mean squared error (MSE) and the training time. MSE provides a measure of the average squared difference between the predicted and actual values, giving us insight into the accuracy of our models. Additionally, we will measure the time taken to train each model, allowing us to assess the computational efficiency of different training procedures.

[10] `https://machinelearningmastery.com/rectified-linear-activation-function-for-deep-learning-neural`

# 5 Results

Table 1: Comparison of Training Time and Accuracy On Concrete Strength Dataset

| **Model** | **Training Time (seconds)** | **Mean Squared Error (MSE)** |
|---|---|---|
| Neural Network (10 epochs) | 0.55 | 1543.86 |
| Neural Network (50 epochs) | 2.67 | 165.45 |
| Neural Network (100 epochs) | 5.35 | 153.00 |
| Neural Network (200 epochs) | 10.85 | 79.56 |
| Neural Network (500 epochs) | 27.13 | 47.3 |
| Random Forest Regressor | 0.63 | 31.29 |

Table 2: Comparison of Training Time and Accuracy On Crab Age Dataset

| **Model** | **Training Time (seconds)** | **Mean Squared Error (MSE)** |
|---|---|---|
| Neural Network (10 epochs) | 46.44 | 4.45 |
| Neural Network (50 epochs) | 227.11 | 11.25 |
| Neural Network (100 epochs) | 435.26 | 4.27 |
| Neural Network (200 epochs) | 843.87 | 4.22 |
| Neural Network (500 epochs) | 2202.20 | 9.94 |
| Random Forest Regressor | 43.62 | 4.49 |

# 6 Conclusions

Based on the performance measurements on two datasets, namely the Concrete Strength Dataset and the Crab Age Dataset, two main conclusions can be drawn:

1. **Training Time:** The neural network models consistently took longer to train compared to the Random Forest Regressor on both datasets. This is expected given the complexity of neural networks and their iterative training process. Additionally, as the number of epochs increased, the training time of neural networks significantly escalated, especially evident in the Crab Age Dataset.

2. **Mean Squared Error (MSE):** Despite the longer training time, neural networks achieved competitive performance in terms of mean squared error, particularly as the number of epochs increased. However, Random Forest Regressor demonstrated relatively lower MSE across both datasets.

In summary, while neural networks offer potential for improved accuracy, particularly with increased training epochs, Random Forest Regressor presents a compelling alternative for tasks where computational efficiency is a priority without significant sacrifice in predictive performance. Furthermore, it is advisable to experiment with various architectures and models to achieve the best results.
[11]

[11]The code and experiments can be found at: `https://github.com/PyMati/EnsembleMethodsVsDeepLearning`

# References


[1] Inc. Anaconda. Conda: Package, dependency and environment management for any language. `https://docs.conda.io/en/latest/`, 2022. Version 4.12.0.

[2] Thomas Kluyver, Benjamin Ragan-Kelley, Fernando Pérez, Brian Granger, Matthias Bussonnier, Jonathan Frederic, Kyle Kelley, Jessica Hamrick, Jason Grout, Sylvain Corlay, Paul Ivanov, Damián Avila, Safia Abdalla, and Carol Willing. Jupyter notebooks – a publishing format for reproducible computational workflows. In F. Loizides and B. Schmidt, editors, *Positioning and Power in Academic Publishing: Players, Agents and Agendas*, pages 87 – 90. IOS Press, 2016.

[3] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[4] Facebook AI Research. PyTorch: An open source deep learning platform. `https://pytorch.org/`, 2022. Version 1.11.0.

[5] Stacey Ronaghan. The mathematics of decision trees, random forest and feature importance in scikit-learn and spark. *Towards Data Science*, 2018.

[6] Wikipedia. Feedforward neural network. *Wikipedia*, 2011.