UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat d'Informàtica de Barcelona

FIB

A COMPREHENSIVE STUDY OF COUNTER PROPAGATION NEURAL NETWORKS: THEORICAL AND EXPERIMENTAL EVALUATION

RUBEN CATALAN RUA

Thesis supervisor
LUIS ANTONIO BELANCHE MUÑOZ (Department of Computer Science)

Degree
Bachelor's Degree in Informatics Engineering (Computing)

Bachelor's thesis

Facultat d'Informàtica de Barcelona (FIB)

Universitat Politècnica de Catalunya (UPC) - BarcelonaTech

Acknowledgements

Abstract

In the realm of machine learning architectures, certain ones are more prevalent than others. Those that are not as well-known or researched may simply not be up to today's industry standards (MLPs, CNNs...), while others have perhaps been forgotten over time.

This bachelor thesis aims to give new life to an almost-forgotten network architecture, the Counter Propagation Neural Network. Three variants are implemented: two following the original author's prescription, and one enhanced with a modern twist to boost performance.

[AÑADIR AQUÍ UN PÁRRAFO DE RESULTADOS]

[AÑADIR AQUÍ UN PÁRRAFO DE CONCLUSIONES]

Resumen

EN ESPAÑOL

Resum

EN CATALÀ

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation of the project

Most, if not all modern neural network architectures use **backpropagation** as the backbone of their learning process. This is not without reason: it is a proven and efficient application of the chain rule for **updating the parameters of a network**, yielding better models.

The development and evolution of the algorithm is understandable, However, this lead to other options or architectures to be obsolete or plainly **forgotten** in the development of new and improved networks.

The Counter Propagation Neural Network is one of these forgotten architectures. Originally proposed in 1986 by Robert Hecht-Nielsen [1], it is a different approach in the training process of a network: featuring three layers, it performs a mapping of input-output data by using a **hybrid learning** approach (unsupervised → supervised).

1.1.1 The problem to solve

Due to their low relevance in today's world, **almost no research has been made to improve or implement this technology** in place of algorithms such as backpropagation. We believe this to be an opportunity to present this architecture in a modern language, with today's tools, and add onto the original idea with proven methods to improve its performance, while keeping the spirit of the original network intact, that is, it's hybrid learning, which is not applied in current networks. Furthermore, the non-existence of actual *state-of-the-art* models allows for us to, not only implement the original and improved versions of the network, but also to benchmark and compare them to current proved architectures, such as the Multi Layer Perceptron (MLP).

1.2 Objectives

There are two main objectives for this project. The first is to **implement** the two types of the counterpropagation algorithm: **'forward-only'** and **'full counterpropagation'**, and the second one is to **evaluate** its performance across different types of problems.

To achieve these, we can outline the steps we will take and what they entail:

1. **Literature Review and Theoretical Analysis:**

   - Conduct an extensive review of the historical development, underlying principles, and previous applications of counterpropagation networks.
   - Establish a solid theoretical foundation and contextualize the relevance of the CPNN in modern AI.

2. **Design and Implementation:**

   - Seamlessly integrate both the unsupervised and supervised layer into a network.
   - Carry out the implementation using a contemporary programming framework that supports modular and scalable development.

3. **Experimental Evaluation:**

   - Apply the implemented model to a range of classification datasets (the CPNN is a mapping network by nature).
   - Assess key performance metrics including accuracy and convergence speed.

4. **Comparative Analysis:**

   - Benchmark the performance of the counterpropagation network against more traditional neural network architectures (e.g., those employing backpropagation).
   - This analysis aims to highlight the unique strengths and potential limitations of the CPNN approach.

1.3 Structure of the project

What follows is a brief summary of the thesis, so that the reader may better understand each point in it and its purpose.

Chapter 2 explains in detail the architecture of the CPNN and its two types, while making sure that its differences with current networks are understood. Chapter 3 shows the implementation and design choices of the networks, and why they are as such. The experimentation process and the evaluation of results are done in chapters 4 and 5, respectively. Finally, the conclusions on the project are given in chapter 6, with a discussion of future developments in chapter 7, and an addendum with the experiment sheets and the project management at the end.

Chapter 2

Architecture of the CPNN

The Counter Propagation Neural Network (CPNN), originally introduced by Robert Hecht-Nielsen in 1986, stands as a distinct neural network architecture that combines both unsupervised and supervised learning paradigms. Unlike traditional feed-forward networks that primarily rely on backpropagation for weight adjustments, CPNNs employ a hybrid learning approach across their distinct layers. This chapter will detail the general architecture of the CPNN, differentiating between its two primary forms: the forward-only CPNN and the full counterpropagation CPNN.

2.1 General CPNN Architecture

A typical Counter Propagation Neural Network consists of three layers:

1. **Input Layer:** This layer receives the external input data $\mathbf{X}$.
2. **Kohonen Layer (or Competitive Layer):** This is the hidden layer, where unsupervised learning takes place. It functions similarly to a Self-Organizing Map (SOM) and is responsible for clustering the input patterns. Each neuron in this layer, often referred to as a Kohonen unit, has a weight vector that serves as a prototype for a specific cluster of input patterns.
3. **Grossberg Layer (or Associative Layer):** This is the output layer, where supervised learning occurs. It associates the activated Kohonen units with the desired output patterns $\mathbf{Y}$.

The core principle behind the CPNN is its ability to learn a mapping from input patterns to output patterns by first categorizing the inputs into distinct clusters (Kohonen layer) and then associating these clusters with corresponding outputs (Grossberg layer).

2.2 Forward-Only Counter Propagation Neural Network

The forward-only CPNN represents the simpler and more commonly implemented variant of the architecture. It is primarily used for pattern classification or function approximation, performing a direct mapping from input to output.

2.2.1 Kohonen Layer Operation (Unsupervised Learning)

The Kohonen layer is the first processing stage. Its learning is entirely unsupervised, meaning it does not use target outputs during its training.

- **Input Presentation:** An input vector $\mathbf{x}$ is presented to the Kohonen layer.
- **Winner-Take-All (WTA) Competition:** For each input $\mathbf{x}$, the Euclidean distance between $\mathbf{x}$ and the weight vector $\mathbf{w}_j$ of every Kohonen neuron j is calculated. The neuron with the minimum distance is declared the "winner" or Best-Matching Unit (BMU): $j^* = \arg\min_j \|\mathbf{x} - \mathbf{w}_j\|^2$
- **Weight Update:** Only the weight vector of the winning neuron j^* is updated. The update rule pulls the weights closer to the input vector. This update is typically governed by a learning rate $\eta(t)$

  $$\Delta \mathbf{w}_j = \eta(t)(\mathbf{x} - \mathbf{w}_j)$$

  Optionally, if we want to go closer to how a typical SOM works, we can add a neighborhood function to influence not only the winning neuron, but its topological neighbors as well. The neighborhood function is as follows:

  $$h_{j^*,j}(t)$$

  Then the update function looks like this:

  $$\Delta \mathbf{w}_j = \eta(t) h_{j^*,j}(t) * (\mathbf{x} - \mathbf{w}_j)$$

  The learning rate $\eta(t)$ and the neighborhood size $\sigma(t)$ (which influences $h_{j^*,j}(t)$) usually decrease monotonically over time (epochs). This decay allows for broad self-organization initially and finer tuning of the map later. Common neighborhood functions include Gaussian, triangular, or rectangular, which define the extent of influence on neighboring neurons. This neighborhood size and

  For example, the Gaussian neighborhood function is given by:

  $$h_{j^*,j}(t) = \exp\left(-\frac{\|\mathbf{p}_{j^*} - \mathbf{p}_j\|^2}{2\sigma(t)^2}\right)$$

  where $\mathbf{p}_{j^*}$ and $\mathbf{p}_j$ are the topological positions of the winning neuron and neuron j, respectively. After the update, the Kohonen weights are often normalized to maintain unit length, which helps in maintaining the stability of the distance calculations.

The output of the Kohonen layer to the Grossberg layer is typically a one-hot vector, where only the BMU's activation is 1 and all others are 0.

2.2.2 Grossberg Layer Operation (Supervised Learning)

The Grossberg layer is an associative memory that learns to map the activated Kohonen units to the desired output patterns. Its learning is supervised, meaning it uses target outputs $\mathbf{y}$ during its training.

- **Output Computation:** When a Kohonen unit j^* is activated (i.e., it is the BMU for a given input $\mathbf{x}$), its corresponding weight vector $\mathbf{W}\_\text{Grossberg}, j^*$ in the Grossberg layer is used to produce the output $\mathbf{y}_{\text{pred}}$. If the Kohonen layer output is a one-hot vector $\mathbf{a}_{\text{WTA}}$, then:

$$\mathbf{y}_{\text{pred}} = \mathbf{a}_{\text{WTA}} \cdot \mathbf{W}_{\text{Grossberg},j^*}^T$$

- **Weight Update:** The Grossberg weights are updated only for the winning Kohonen unit j^*. The update rule is a form of Hebbian learning, adjusting the weights to associate the winning Kohonen unit with the presented target output $\mathbf{y}$. The update rule for the weights connecting the winning Kohonen unit j^* to the output units is:

$$\Delta \mathbf{W}_{\text{Grossberg},j^*} = \eta_{\text{gr}}(\mathbf{y} - \mathbf{W}_{\text{Grossberg},j^*})$$

  where η_{gr} is the Grossberg learning rate. This rule effectively moves the Grossberg weights associated with the winning Kohonen unit towards the average of the target outputs that have activated that specific Kohonen unit.

The training process for a forward-only CPNN typically involves alternating between Kohonen layer updates (unsupervised) and Grossberg layer updates (supervised) for each training epoch.

```
procedure TrainForwardOnlyCPNN(X, Y, epochs)
    initialize KohonenWeights randomly (0,1)
    initialize GrossbergWeights randomly (0,1)

    for epoch in 1 to epochs do
        for each (x, y) in (X, Y) do
            // Kohonen Layer: Winner-Take-All
            j* := argmin_j ||x - w_j||^2

            // Update Kohonen Weights
            for each neighbor j of j* do
                inf := neighborhood_function(j*, j, epoch)
                w_j := w_j + η_kh * inf * (x - w_j)
                normalize w_j

            // Grossberg Layer: Hebbian Learning
            W_grossberg[j*] := W_grossberg[j*] + η_gr * (y -
               W_grossberg[j*])
        end for
    end for
end procedure

function PredictForwardOnlyCPNN(x)
    j* := argmin_j ||x - w_j||^2
    return W_grossberg[j*]
end function
```

Listing 2.1: Forward-Only Counter Propagation Neural Network

2.3 Full Counterpropagation Neural Network

The full counterpropagation CPNN extends the forward-only model by allowing information to flow in both directions through the Grossberg layer, enabling it to learn bidirectional mappings. This means the network can not only map inputs to outputs but also outputs back to inputs, or even complete partial patterns.

2.3.1 Bidirectional Grossberg Layer

In the full counterpropagation model, the Grossberg layer is designed to be bidirectional. Instead of simply mapping Kohonen units to outputs, it learns to associate Kohonen units with pairs of input and output patterns $(\mathbf{x}, \mathbf{y})$.

- **Combined Input-Output Space:** During training, a concatenated vector $[\mathbf{x}, \mathbf{y}]$ is often used as the target for the Grossberg layer. The Grossberg weights $\mathbf{W}_{\text{Grossberg}}$ are trained to store these combined patterns.

- **Training Phase:**

  1. An input $\mathbf{x}$ is presented to the Kohonen layer, and the BMU j^* is identified.
  2. The Kohonen weights are updated as in the forward-only model.
  3. Simultaneously, the Grossberg weights connected to j^* are updated to learn the association with the pair $(\mathbf{x}, \mathbf{y})$. The Grossberg layer essentially learns to store the average of all input-output pairs that activate a particular Kohonen unit. The update rule for the Grossberg weights connecting the winning Kohonen unit j^* to the combined input-output space is:
  $$\Delta \mathbf{W}_{\text{Grossberg},j^*} = \eta_{\text{gr}}([\mathbf{x}, \mathbf{y}] - \mathbf{W}_{\text{Grossberg},j^*})$$

- **Interpolation/Recall Phase:** After training, the full counterpropagation network can be used for various tasks:

  1. **Forward Mapping (Input to Output):** An input $\mathbf{x}$ is presented. The Kohonen layer identifies the BMU j^*. The Grossberg layer then recalls the associated output $\mathbf{y} * \text{pred}$ from its learned weights $\mathbf{W}_{\text{Grossberg},j^*}$.
  2. **Backward Mapping (Output to Input):** An output $\mathbf{y}$ is presented. The Grossberg layer can be used in reverse to activate the Kohonen units, and then the Kohonen weights can be used to reconstruct the associated input $\mathbf{x}_{\text{pred}}$. This often involves finding the Kohonen unit whose Grossberg weights are closest to the presented output.
  3. **Pattern Completion:** If a partial pattern (e.g., only a part of $\mathbf{x}$ or $\mathbf{y}$) is presented, the network can attempt to complete the missing parts by iteratively activating layers and recalling associated patterns.

The full counterpropagation CPNN offers more versatility than its forward-only counterpart, making it suitable for tasks requiring associative memory and pattern completion in addition to classification. Its bidirectional nature, however, also adds complexity to its training and application.

```
procedure TrainFullCPNN(X, Y, epochs)
    initialize KohonenWeights randomly (0,1)
    initialize GrossbergWeights randomly (0,1)

    for epoch in 1 to epochs do
        for each (x, y) in (X, Y) do
            z := concatenate(x, y)

            // Kohonen Layer: Winner-Take-All
            j* := argmin_j ||x - w_j||^2

            // Update Kohonen Weights
            for each neighbor j of j* do
                inf := neighborhood_function(j*, j, epoch)
                w_j := w_j + η_kh * inf * (x - w_j)
                normalize w_j

            // Update Grossberg Weights with concatenated
               input-output
            W_grossberg[j*] := W_grossberg[j*] + η_gr * (z -
               W_grossberg[j*])
        end for
    end for
end procedure

function ForwardRecall(x)
    j* := argmin_j ||x - w_j||^2
    z := W_grossberg[j*]
    return extract_y(z) // returns the output part of the
       concatenated vector
end function

function BackwardRecall(y)
    j* := argmin_j ||y - extract_y(W_grossberg[j])||^2
    z := W_grossberg[j*]
    return extract_x(z) // returns the input part of the
       concatenated vector
end function
```

Listing 2.2: Full Counterpropagation Neural Network

Chapter 3

Implementation

The implementation of the network was carried out in PyTorch, a very well-known Python framework for neural networks. It was chosen over others such as TensorFlow due to its dynamic computational graph and low-level API. This means that, while it is harder to build a network, it allows for way more control over it, such as the way in which data passes through the network, or the manual calculation of the gradient vector, which were key in developing the CPNN.

I will now go over the decisions during development. These will be separated between design choices (such as the loss function, schedulers, etc) and hyperparameters (and why choose these). A separate explanation will be provided for each of the types of CPNN implemented if required due to changes.

3.1 Design choices

3.1.1 Common Design Choices

- **Device Agnosticism:** The models are designed to run on either a CUDA-enabled GPU (if available) or the CPU. This is handled by `self.device = torch.device("cuda"if torch.cuda.is_available()else "cpu")`, ensuring portability and leveraging hardware acceleration when possible.
- **Input Flattening:** An `nn.Flatten()` layer is included at the beginning of the `forward` pass. This ensures that inputs of varying shapes (e.g., images with multiple channels and dimensions) are converted into a 1D vector compatible with the Kohonen layer's weight vectors.
- **Weight Initialization:**
  - **Kohonen Weights:** Both `kohonen_weights` and `grossberg_weights` (for `BaseCPNN`) are initialized using a normal distribution (`.normal_(0, 1)`), which is a common practice to break symmetry and allow effective learning.
  - **Weight Normalization:** Crucially, after initialization and after every update step, the Kohonen weights are normalized using `F.normalize(..., p=2, dim=1)`. This ensures that the weight vectors remain on the unit hypersphere, which is fundamental for distance-based learning in SOMs (Kohonen layers) where the closest prototype is determined by Euclidean distance. Normalization prevents weights from growing indefinitely and helps maintain stable learning.

- **Loss Function:** For evaluating the supervised component of the network (Grossberg layer in `BaseCPNN` and MLP in `ModifiedCPNN`), `nn.CrossEntropyLoss(reduction='mean')` is used. This is a standard and effective loss function for multi-class classification problems, combining `log_softmax` and `NLLLoss` for numerical stability.

- **Neighborhood Function Implementation:** The `update_kohonen` method dynamically applies the chosen neighborhood function ('gaussian', 'triangular', 'rectangular') based on the `neighborhood_function` attribute. This allows for flexible experimentation with different neighborhood influences during the unsupervised learning phase. The calculation of `diff_indices` (the distance in index space between the winning neuron and all other neurons) is central to applying these functions.

- **Neighborhood Size Decay:** The `fit` method incorporates a decaying `neighborhood_size` ($\sigma(t)$) over epochs. This is a crucial aspect of SOM training:

$$\sigma(t) = \max\left(1.0, \sigma_0 \cdot e^{-\lambda t}\right) \tag{3.1}$$

  where $\sigma(t)$ is the initial neighborhood size, t is the current epoch, and $\lambda = \frac{\ln \sigma_0}{\text{epochs}}$. is the time constant for decay. This ensures that the Kohonen layer initially organizes broadly (large neighborhood) and then refines its clusters (small neighborhood) as training progresses. The minimum σ_0 is set to 1.0 to prevent division by zero and ensure that at least one neuron is being fully influenced.

- **Early Stopping:** Both implementations support early stopping based on validation loss. If `early_stopping` is enabled and the validation loss does not improve for `patience` consecutive epochs, training is halted. This prevents overfitting and saves computational resources.

- **Kohonen Snapshots:** The `kohonen_snapshots` list stores a clone of the Kohonen weights at the end of each epoch. This feature is invaluable for visualizing the evolution of the Kohonen map during training, allowing for qualitative analysis of the unsupervised learning process.

3.1.2 Specific Design Choices for BaseCPNN

- **Kohonen Optimizer (`optimizer_kh`):** `optim.SGD` (Stochastic Gradient Descent) with `weight_decay=1e-4`, `momentum=0.95`, and `nesterov=True` is used for updating Kohonen weights. SGD is a robust optimizer, and the added momentum and Nesterov acceleration help in faster convergence and navigating flatter loss landscapes in the unsupervised learning phase. The `weight_decay` acts as a regularization term.

- **Grossberg Optimizer (`optimizer_gr`):** `optim.AdamW` is chosen for the Grossberg layer. AdamW is an adaptive learning rate optimizer that is generally

well-suited for supervised learning tasks due to its efficiency and ability to handle sparse gradients.

- **Manual Gradient for Kohonen and Grossberg:** A key design choice reflecting the CPNN's unique learning rules is the manual calculation and assignment of gradients for `kohonen_weights` and `grossberg_weights`. Instead of relying solely on PyTorch's automatic differentiation (`.backward()`), the gradients are explicitly computed based on the specific update rules of the Kohonen and Grossberg layers, and then assigned to `self.kohonen_weights.grad` and `self.grossberg_weights.grad` before the optimizer step. This is essential because the Kohonen and Grossberg updates are not standard backpropagation steps.

3.1.3 Specific Design Choices for ModifiedCPNN

- **MLP Architecture:** The MLP design is flexible, allowing for any number of hidden layers, with subsequent layers having its number of neurons reduced by a factor.

- **Softmax Temperature:** The `ModifiedCPNN` introduces a `temperature` parameter in the softmax calculation of the Kohonen layer's activations. This parameter controls the "softness" of the winner-take-all behavior. A high temperature (e.g., 1.0 or more) leads to more uniform activations across Kohonen neurons, while a low temperature (e.g., 0.1) makes the activations more concentrated around the winning neuron.

- **Dropout Regularization:** Since the Kohonen layer now applies a softer winner-takes-all behavior, more neurons in the MLP layers will be active, and thus, dropout layers are strategically placed between MLP layers. Dropout is a powerful regularization technique that randomly sets a fraction of input units to zero at each update during training, preventing complex co-adaptations on training data and improving generalization.

- **MLP Optimizer (`optimizer_mlp`):** `optim.AdamW` is used for the MLP, consistent with its strong performance in supervised learning contexts.

- **Standard Backpropagation for MLP:** Unlike the Grossberg layer in `BaseCPNN`, the MLP in `ModifiedCPNN` is trained using standard backpropagation. The `loss.backward()` call automatically computes gradients for all MLP parameters, which are then updated by `optimizer_mlp.step()`. This simplifies the supervised learning phase and leverages PyTorch's automatic differentiation capabilities for the MLP.

3.2 Hyperparameters

The performance of any neural network, including the CPNN, is highly dependent on the careful selection of its hyperparameters. These values are not learned by the

model but are set prior to training and significantly influence the learning process and final performance.

3.2.1 Common Hyperparameters

- `input_size` **(Integer):** The dimensionality of the input features. This must match the flattened size of the input data (e.g., for a 28×28 grayscale image, it would be 784).
- `hidden_size` **(Integer):** The number of neurons in the Kohonen layer. This effectively determines the resolution of the Kohonen map and the number of clusters the unsupervised layer can form. A larger `hidden_size` allows for more fine-grained clustering but increases computational cost and the risk of dead neurons.
- `output_size` **(Integer):** The number of output classes. This corresponds to the number of distinct categories the network is designed to classify.
- `epochs` **(Integer):** The total number of training iterations over the entire dataset. A sufficient number of epochs is necessary for the network to converge, but too many can lead to overfitting.
- `neighborhood_size` **(Float):** The initial size of the neighborhood for the Kohonen layer. This parameter, $\sigma(0)$, dictates the initial radius of influence for the winning neuron's updates. A larger initial size encourages broader self-organization, while a smaller size leads to more localized updates from the start. It is crucial for the initial organization of the Kohonen map.
- `neighborhood_function` **(String):** Defines the shape of the influence function used during Kohonen weight updates ('gaussian', 'triangular', or 'rectangular'). Gaussian is often preferred for its smooth decay, while triangular and rectangular offer simpler, more abrupt transitions.
- `kohonen_lr` **(Float):** The learning rate for the Kohonen layer's weight updates. This controls the step size at which Kohonen neurons adjust their weights towards input vectors. A carefully chosen `kohonen_lr` is essential for effective self-organization.

3.2.2 BaseCPNN Specific Hyperparameters

- `grossberg_lr` **(Float):** The learning rate for the Grossberg layer's weight updates. This determines how quickly the Grossberg layer adapts its weights to correctly map Kohonen activations to target outputs.

3.2.3 ModifiedCPNN Specific Hyperparameters

- `hidden_layers` **(Integer):** The number of hidden layers in the MLP. This allows for varying the depth and complexity of the supervised mapping com-

ponent. A value of 0 means the MLP will be a single linear layer from the Kohonen activations to the output.

- `div_layer_value` **(Integer):** A factor used to reduce the number of neurons in successive hidden layers of the MLP. For example, if `div_layer_value` is 2, each subsequent hidden layer will have half the neurons of the previous one. This creates a tapering architecture, potentially forcing the MLP to learn more compressed representations.

- `dropout_rate` **(Float):** The probability of a neuron's output being set to zero during training in the MLP's dropout layers. This is a regularization technique to prevent overfitting by reducing co-adaptation between neurons. Typical values range from 0.1 to 0.5.

- `temperature` **(Float):** A parameter applied to the softmax function in the Kohonen layer's forward pass. It controls the "sharpness" of the soft activations, as already explained.

- `mlp_lr` **(Float):** The learning rate specifically for the MLP's parameters. This controls the step size for gradient updates during the supervised training of the MLP.

The selection of these hyperparameters during the project was done using the Optuna library in Python, which applies a Bayesian Search algorithm to the pool of hyperparameters.

Chapter 4

Experimentation

After building all the CPNN versions we wanted to evaluate, we first had to validate their correct functioning.

4.1 Validation

The two most important steps to take are:

- Check the network is generalizing correctly after training - Question: **Is the test error reasonable?**
- Confirm the network is capable of overfitting - Question: **Can the network memorize a training set?**

To answer these, we can use a toy dataset, such as MNIST. This dataset presents a large number of handwritten digits, normalized to fit into a 28x28 pixel wide grayscale image. It contains 60,000 training images and 10,000 testing images.

To evaluate the testing error, the networks were trained on the training set and tested on the testing set. A common benchmark to reach to confirm the network IS correctly training and generalizing is 95% accuracy during testing, which all networks achieved handily.

To check for memorization, the training process was maintained the same but testing was done using the same training set. The benchmark for memorizing is higher, at around 99-100%. Once again, this was reached by all previously tested networks.

4.2 Experimental design

Chapter 5

Result Evaluation

Chapter 6

Conclusions

Chapter 7

Future research

Bibliography

[1] Hecht-Nielsen, R. (1990) Neurocomputing. Reading, Massachussets: Addison-Wesley.

[2] Margaris, Athanasios & Kotsialos, E.. (2004). Parallel counter-propagation networks. Acta Universitatis Apulensis. Mathematics - Informatics. 8.

[3] Parallel counter-propagation networks - Scientific Figure on ResearchGate. https://www.researchgate.net/figure/A-typical-counter-propagation-network_fig7_268162260 [accessed 28 Feb 2025]

[4] "Counter Propagation Networks — Study Glance." Studyglance.in, 2020, https://studyglance.in/nn/display.php?tno=13&topic=Counter-Propagation-Networks. Accessed 21 Feb. 2025.

[5] Find, install and publish python packages with the python package index. PyPI. https://pypi.org/

[6] Ansel, J., Yang, E., He, H., Gimelshein, et al. (2024). PyTorch 2: Faster Machine Learning Through Dynamic Python Bytecode Transformation and Graph Compilation [Conference paper]. 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2 (ASPLOS '24). https://doi.org/10.1145/3620665.3640366

[7] Abadi, M., Agarwal, A., Barham, P., et al. (2015). TensorFlow, Large-scale machine learning on heterogeneous systems [Computer software]. https://doi.org/10.5281/zenodo.4724125

[8] Git. https://git-scm.com/

[9] GitHub · build and ship software on a single, Collaborative Platform. GitHub. ttps://github.com/

[10] . *Glassdoor*. https://www.glassdoor.es/Empleo/index.htm. Accessed on: 11/03/2025.

[11] . *Glassdoor Project Manager Spain*. https://www.glassdoor.es/Sueldos/espa%C3%B1a-project-manager-sueldo-SRCH_IL.0,6_IN219_KO7,22.htm. Accessed on: 11/03/2025.

[12] . *Glassdoor ML Engineer Spain*. https://www.glassdoor.es/Sueldos/espa%C3%B1a-ml-engineer-sueldo-SRCH_IL.0,6_IN219_KO7,18.htm. Accessed on: 11/03/2025.

[13] Asurion. (2023, February 15). How long do laptops last on average? — Asurion. Asurion. https://www.asurion.com/connect/tech-tips/how-long-does-a-laptop-last/. Accessed on: 11/03/2025

[14] Comprar Windows 11 Home - Microsoft Store. (n.d.). Microsoft Store. https://www.microsoft.com/es-es/d/windows-11-home/dg7gmgf0krt0. Accessed on: 11/03/2025

[15] Bravo 15 - Rise through the ranks. (n.d.). msi.com. All Rights Reserved. https://es.msi.com/Laptop/Bravo-15-B5DX/Specification. Accessed on: 11/03/2025

[16] Precio de la tarifa de luz por horas HOY — 12 Marzo 2025. https://tarifaluzhora.es/. Accessed on: 12/03/2025

7.1 BIBLIOGRAFÍA A REDACTAR

7.1.1 Puntos de introduccion

- Leibniz, Gottfried Wilhelm Freiherr von (1920). The Early Mathematical Manuscripts of Leibniz: Translated from the Latin Texts Published by Carl Immanuel Gerhardt with Critical and Historical Notes (Leibniz published the chain rule in a 1676 memoir). Open court publishing Company. ISBN 9780598818461.

7.1.2 Otros

Appendix A

Project Management

A.1 Description of Tasks

This project is estimated to require approximately 530 hours of work between the 27th of January (project acceptance) and the 25th of June (expected delivery). The plan allocates around 3.5 hours of work per day, with additional time reserved for unforeseen challenges.

A.1.1 Task Definition

Below are the tasks to be executed throughout the project, listed in the order in which they will be carried out. Where tasks can be performed concurrently, this will be noted. The tasks are as follows:

- **Project Management** This is the foundation of the work, establishing the project's conceptual basis, defining the tasks, and outlining the expected timeline and budget. It includes:
  - **PM1 - Context and Scope:** Define the overall objectives of the project, justify its selection, and list the development tools to be used. (Estimated 20 hours)
  - **PM2 - Project Planning:** Detail the task schedule and explain the rationale behind the sequencing based on task relevance, dependencies, and deadlines. (Estimated 20 hours)
  - **PM3 - Budget and Sustainability:** Estimate the financial costs and assess the project's economic, environmental, and social impact. (Estimated 20 hours)
  - **PM4 - Meetings:** Schedule and conduct weekly online meetings (with additional in-person sessions if required) with the project tutor to review progress and address challenges. (Estimated 30 hours)
  - **PM5 - Integration into Documentation:** Consolidate all completed tasks and revisions into the final project document, ensuring clarity and consistency. (Estimated 25 hours)
- **Preliminary Work** These tasks involve preparatory research and groundwork before initiating the experimental phase. They are grouped into:
  - **Theoretical Research:** Deepen understanding of neural networks and specifically Counterpropagation Neural Networks (CPNNs) by reviewing relevant literature and prior studies. This phase includes:

    - **PTR1 - Expanding Knowledge on Neural Networks:** Study the fundamentals and advanced concepts of neural networks to build a robust theoretical base. (Estimated 10 hours)
    - **PTR2 - Understanding How CPNNs Work:** Focus on the unique characteristics of CPNNs, including their dual learning approach. (Estimated 25 hours)
    - **PTR3 - Analysis of Previous Works:** Review and analyze existing research and experiments related to CPNNs, noting successful approaches and potential pitfalls. (Estimated 15 hours)
  - **Environment Preparation:** Set up a development environment that allows building the network from scratch, ensuring sufficient flexibility for customization:
    - **PEP1 - Choosing a Framework:** Evaluate and select an appropriate framework (e.g., PyTorch or TensorFlow) for implementing the CPNN. PyTorch has been chosen for its ease of use and flexibility. (Estimated 10 hours)
    - **PEP2 - Getting Familiar with the Framework:** Invest time in learning the framework's functionalities and best practices to enable efficient development. (Estimated 25 hours)

- **Implementation** This phase involves the actual coding and development of the CPNN, broken down into:
  - **IMP1 - Architectural Design:** Define the overall architecture of the CPNN, including the structure of both the Kohonen (unsupervised) and Grossberg (supervised) layers. (Estimated 20 hours)
  - **IMP2 - Development of the Unsupervised Component:** Code the Kohonen layer responsible for organizing input data through competitive learning. (Estimated 20 hours)
  - **IMP3 - Development of the Supervised Component:** Implement the Grossberg layer that maps clustered data to the target outputs using error feedback. (Estimated 20 hours)
  - **IMP4 - Integration and System Testing:** Combine both components into a unified model and conduct unit tests to ensure correct functionality. (Estimated 20 hours)

- **Experimentation** In this stage, the developed CPNN will be rigorously tested and refined:
  - **EXP1 - Dataset Preparation:** Identify, acquire, and preprocess datasets suitable for various tasks (e.g., classification, regression), ensuring data quality and relevance. (Estimated 30 hours)
  - **EXP2 - Training and Parameter Tuning:** Execute training sessions, fine-tune hyperparameters, and optimize learning strategies through iterative experiments. (Estimated 30 hours)

- **EXP3 - Robustness Testing:** Evaluate the network under different conditions, including noise and data variability, to assess convergence and generalization. (Estimated 30 hours)
- **EXP4 - Comparative Analysis:** Benchmark the performance of the CPNN against traditional neural network architectures, particularly those using backpropagation. (Estimated 30 hours)

- **Evaluation of Results** The final phase focuses on analyzing and documenting the outcomes:
  - **EVAL1 - Performance Analysis:** Analyze key performance metrics such as accuracy, convergence speed, and robustness based on experimental data. (Estimated 20 hours)
  - **EVAL2 - Comparative Assessment:** Compare the experimental results with theoretical expectations and results from conventional networks. (Estimated 15 hours)
  - **EVAL3 - Identification of Strengths and Limitations:** Discuss the unique advantages and any shortcomings of the CPNN approach. (Estimated 15 hours)

- **Documentation**
  - **DOC1 - Ongoing documentation:** Write all important and relevant data to the project while it is being developed. (Estimated total time: 20 hours).
  - **DOC2 - Final Reporting:** Prepare a comprehensive report detailing the methodology, experiments, results, and future research directions. (Estimated total time: 50 hours)
  - **DOC2 - Presentation:** Preparing and studying the presentation done for the defense of the project. (Estiamated total time: 10 hours)

A.1.2 Resources

To complete the tasks as mentioned, certain resources shall be used. We've decided to separate them in three groups:

Human resources

- **Researcher (R).** The student in charge of development of the project. We shall not mention in him during tasks as it is implicitly or explicitly involved in all of them.
- **Thesis tutor (TT)**. The tutor of the project, in charge of guiding and overlooking the correct development of tasks.
- **Project management tutor (PMT).** Responsible for reviewing and giving feedback during the initial project management tasks.

Moreover, we shall make a distinction in the roles the researcher will take on during the project, as they change according to the task at hand.

- **Project Manager:** In charge of defining, scheduling and recognizing requirements and possible delays in the tasks needed to accomplish the project.
- **ML Engineer:** This role is responsible in the development, programming and training of the algorithms and models used in Machine Learning.

Tasks	Role
PM1,PM2,PM3,PM4,PM5,DOC1,DOC2,DOC3	Project Manager
PTR1-3,PEP1-2,IMP1-4,EXP1-4,EVAL1-3	ML Engineer

Table A.1: Role Breakdown

Material resources

- Laptop (L): It shall be used both for the implementation and experimentation of the networks as well as the project documentation.
- Previous research works (PRW): they shall be read and used as a basis for improving our research and experimentation.

Software resources

- Git / GitHub (Git): both allowing for version control of our code, keeping backups and allowing to create branches of the project (if tasks were to diverge) and merging or deleting them when needed.
- Sublime Text (ST): the researcher's preferred code editor. Chosen due to comfortability and familiarity.
- Overleaf (OL): an online LaTeXeditor which makes writing complex documents faster as well as keeping backups of documents online.
- PyTorch (PT): the aforementioned framework for developing NNs, allowing a high level of liberty while keeping Python-level syntax.

A.1.3 Relationship between tasks and resources

ID	Task	Time (h)	Depends on	Resources
PM1	Context and Scope	20	–	TT, PMT, L, OL
PM2	Project Planning	20	PM1	TT, PMT, L, OL
PM3	Budget and Sustainability	20	PM1, PM2	TT, PMT, L, OL
PM4	Meetings	30	–	TT, PMT, L, OL
PM5	Integration into Documentation	25	PM1, PM2, PM3, PM4	TT, PMT, L, OL
PTR1	Expanding Knowledge on NNs	10	PM2	TT, PRW, L
PTR2	Understanding How CPNNs Work	25	PTR1	TT, PRW, L
PTR3	Analysis of Previous Works	15	PTR1, PTR2	TT, PRW, L
PEP1	Choosing a Framework	10	PM2	TT, L, ST
PEP2	Getting Familiar with the Framework	25	PEP1	TT, L, ST
IMP1	Architectural Design	20	PTR1, PTR2, PTR3, PEP2	L, Git, ST, PT
IMP2	Development of Unsupervised Component	20	IMP1	L, Git, ST, PT
IMP3	Development of Supervised Component	20	IMP1, IMP2	L, Git, ST, PT
IMP4	Integration and System Testing	20	IMP1, IMP2, IMP3	L, Git, ST, PT
EXP1	Dataset Preparation	30	IMP4	L, Git, ST, PT
EXP2	Training and Parameter Tuning	30	EXP1	L, Git, ST, PT
EXP3	Robustness Testing	30	EXP2	L, Git, ST, PT
EXP4	Comparative Analysis	30	EXP1, EXP2, EXP3	L, Git, ST, PT
EVAL1	Performance Analysis	20	IMP4, EXP4	TT, L, Git, ST, PT
EVAL2	Comparative Assessment	15	EVAL1	TT, L, Git, ST, PT
EVAL3	Identification of Strengths and Limitations	15	EVAL1, EVAL2	TT, L, Git, OL, PT
DOC1	Ongoing Documentation	20	All tasks	TT, L, Git, OL
DOC2	Final Reporting	50	All tasks	TT, L, Git, OL
DOC3	Presentation	10	All tasks	TT, L, OL
	Total	**530**		

Table A.2: Task breakdown, estimated time, dependencies, and resource usage

A.1.4 Risk management

Below are the possible risks that may be encountered during development, their assessed risk levels, and the mitigation plans designed to address them:

1. **Algorithm Complexity [High Risk]:**
    - **Affected Tasks:** IMP2 (Development of Unsupervised Component), IMP3 (Development of Supervised Component), IMP4 (Integration and System Testing), EXP2 (Training and Parameter Tuning).
    - **Alternative Plan:** Should parameter tuning issues or convergence problems occur, an additional 10–15% of the implementation timeline will be allocated for focused debugging and adjustment. Detailed unit tests and scheduled consultation sessions with domain experts will be initiated to resolve specific issues.
    - **Additional Resources:** Extra technical reviews and, if needed, access to expert consultation.

2. **Limited Contemporary Research [Medium Risk]:**

   - **Affected Tasks:** PTR1, PTR2, PTR3 (Theoretical Research tasks).
   - **Alternative Plan:** In case the literature is insufficient, a broader search including related fields will be conducted. An extra 5–10 hours will be reserved for accessing additional academic databases and expert opinions to validate the approach.
   - **Additional Resources:** Extended literature review access and potential academic collaborations.

3. **Resource Limitations [Medium Risk]:**

   - **Affected Tasks:** EXP1, EXP2, EXP3 (Experimentation tasks).
   - **Alternative Plan:** If high-performance computing resources are insufficient, code optimizations will be prioritized and experiments may be scheduled during off-peak hours. This may introduce a 5–10% delay in experimentation.
   - **Additional Resources:** Consideration of cloud computing resources or temporary hardware upgrades.

4. **Time Constraints [High Risk]:**

   - **Affected Tasks:** All phases—with critical emphasis on Implementation and Experimentation.
   - **Alternative Plan:** If delays exceed 10% during any sprint, task priorities will be re-assessed. Contingency measures include reallocating resources, extending work hours temporarily, or re-sequencing non-critical tasks.
   - **Additional Resources:** Potential additional support or negotiated timeline extensions with the project supervisor.

5. **Data Accessibility and Relevance [Low Risk]:**

   - **Affected Tasks:** EXP1 and EXP2 (Dataset Preparation and Training).
   - **Alternative Plan:** If existing datasets prove inadequate, additional datasets will be sourced and data augmentation techniques will be applied. An extra 5% of the experimentation phase may be needed for data validation and preprocessing.
   - **Additional Resources:** Access to multiple data repositories and data cleaning tools.

6. **Platform/Library Choice [Medium Risk]:**

   - **Affected Tasks:** PEP1, PEP2 (Environment Preparation) and IMP2, IMP3 (Implementation tasks).

- **Alternative Plan:** Although PyTorch is currently chosen, if significant issues arise, a contingency plan involves evaluating TensorFlow or an alternative framework. This switch could result in an estimated delay of 10–15% in development time.
- **Additional Resources:** Extra research and training in the alternative platform.

A.1.5 Complete Gantt chart

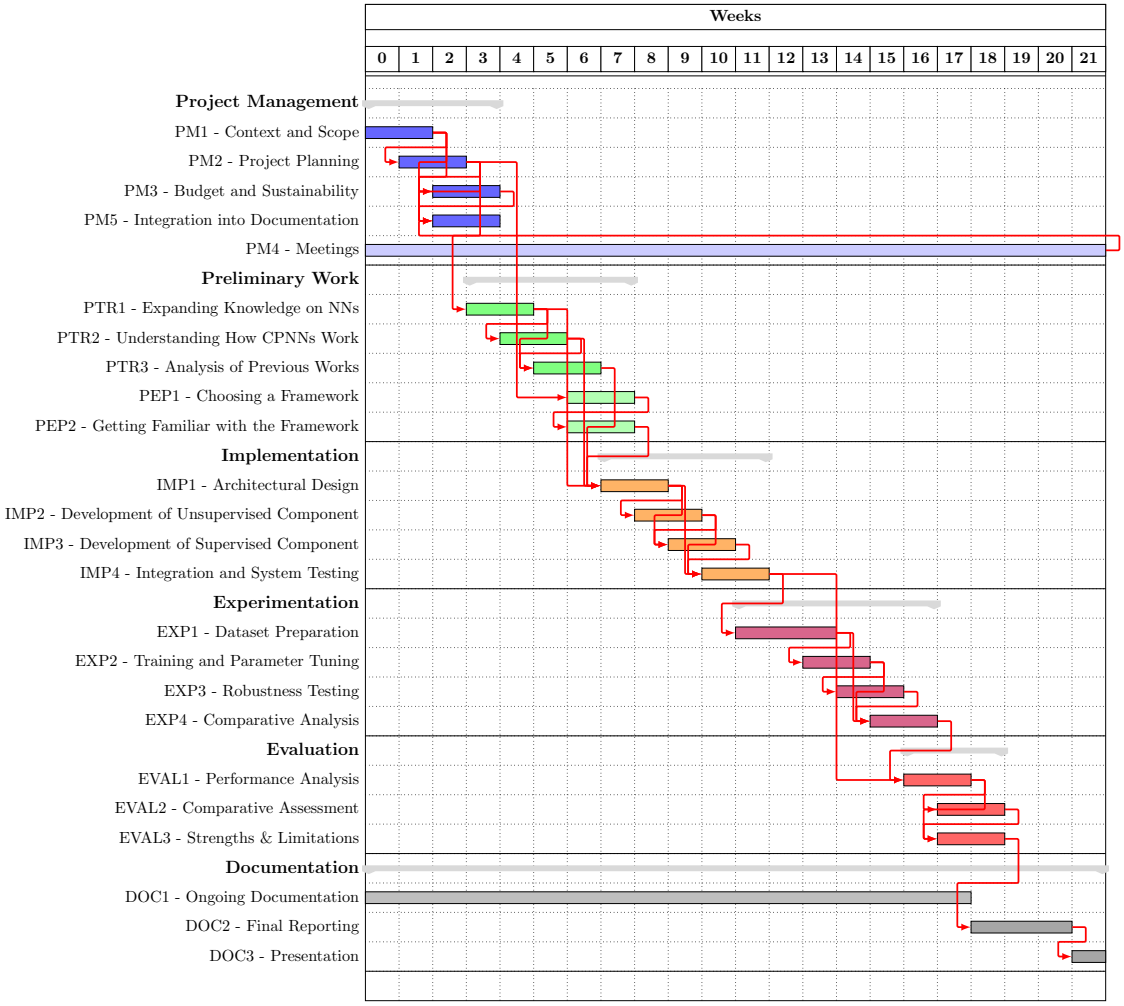

Figure A.1: Gantt chart for the project timeline (done with the 'pgfgantt' package).

A.2 Budget

We will now described the elements to take into account regarding the budget for this project. This budget is calculated over the costs of human and material resources, as well as generic and other other such as unexpected costs, which will also be tackled.

It's worth noting that this section is very much an estimation of the costs, and assumptions and averages shall be used. Keep this in mind during reading as calculating exact amount is a nigh-impossible task.

A.2.1 Human resources

The largest source of costs for the project. Due to the fact that the costs of the tutor of the thesis and the GEP tutor are not computed (as their work hours are expected to be part of their job as teachers), this section is entirely reliant on the costs of the roles played by the researcher, table A.3 shows the total costs, computed as the sum of the hourly salaries multiplied by the number of hours each role shall be expected to be played. Of course, these hourly salaries are just averages extracted from Glassdoor [10] over a one year range here in Spain. The formula to calculate the hourly salary is fairly simple, but it should be mentioned anyways as it assumes 8 hour days over 20 days of work per month (4 weeks working monday to friday).

$$\text{Hourly Rate} = \frac{\text{Annual Salary}}{12months \times 20days \times 8hours} = \frac{\text{Annual Salary}}{1920}$$

Role	Salary	Expected Hours	Total	Total + S. Sec.	Social Security
Project Manager	20.83 €/hour (40k €/year) [11]	195 hours	4,061.85	5483.5	1,421.65
ML Engineer	21.61 €/hour (41.5k €/year) [12]	335 hours	7,239.35	9,773.12	2,533.77
Total	–	530 hours	11,301.2	15,256.62	3955.42

Table A.3: Cost of researcher's roles with a 35% social security factor (i.e. total = base cost × 1.35), self elaborated

A.2.2 Material resources

Laptop

Most tasks shall be done using the researcher's personal laptop, these include all implementation, experimentation and documentation tasks, as well as part of the preliminary research. The laptop's model is an **MSI Bravo 15 B5DD-005XES**, which had a cost of 700€ when the researcher bought it (30% of discount over the original 1000€). If we assume an average useful life of around 5 years [13], and 24.5 hours of work with it per week, we can calculate the amortization costs:

$$\text{Total Work Hours} = 24.5\,\frac{\text{hours}}{\text{week}} \times 52\,\frac{\text{weeks}}{\text{year}} \times 5\,\text{years} = 6370\,\text{hours},$$

$$\text{Total Amortized Cost} = \frac{700\,€}{6370\,\text{hours}} \times 530\,\text{hours} = 58,24€.$$

Previous research works

Most of the research works that will be used to improve and guide the project can be found online for free, and the only one that is being read as a physical book, Hecht-Nielsen's *Neurocomputing* [1], are borrowed from the college's library for free. Therefore, none of these resources add any cost to our project.

A.2.3 Software resources

Adding onto the software resources already mentioned in previous sections, we shall mention our operating system, as it is true that some require a fee to use, such as Microsoft's Windows [14]. In our case, since we're using Manjaro, a free Linux distribution, we don't have such costs. The rest of the software resources are also free of use (either due to being Open-Source or just plain free), but we shall show them in a table for the sake of clarity in terms of amortization.

Resource	Cost	Amortization
Manjaro (OS)	0€	0€
Git / GitHub	0€	0€
Sublime Text	0€	0€
Overleaf	0€	0€
PyTorch (OS)	0€	0€

Table A.4: Software costs, self elaborated

A.2.4 Indirect costs

The final type of costs to discuss are the indirect ones, these are electricity costs, connection, etc.

It's important to mention that, out of the 3.5 estimated daily hours of work, 1 of these is done outside the researcher's house, during his free time at the office he works at. This ammounts to 5 out of the 24.5 hours of weekly work (working monday to friday), or around 20.4% of the total ammount. Therefore, costs like electricity will be multiplied by a factor of **1 - 0.204 = 0.796**, to account for this.

Electricity

The aforementioned laptop has 5 hours of battery life, with a capacity of around 46,35 Wh (by design, 52 Wh [15], however battery degradation from use has left it at an 89% of the original capacity), and a full recharge time of 2.5 hours. Assuming an average Joule effect of 10%, we can estimate the power needed for a full charge:

$$\text{Required energy} = \frac{46.35\,\text{Wh}}{0.9\,\text{hours}} = 51.5\text{Wh}$$

$$\text{Average power} = \frac{51.5\,\text{Wh}}{2.5\,\text{hours}} = 20.6\text{W}$$

Now, we shall calculate the ammount of hours of charge total needed to complete the project, as well as the ammount of electricity needed in that time, to then calculate the total cost.

$$\text{Recharges} = 530\,\text{hours} \times \frac{1\ \text{recharge}}{5\,\text{hours}} \times 0.796 = 84.38 \approx 84\ \text{recharges}$$

$$\text{Recharge hours} = 84 \times \frac{2.5\ \text{hours}}{1\,\text{recharge}} \times 0.796 = 167.16\ \text{hours} \approx 167\ \text{hours}$$

$$\text{Kilowatts} = 84 \times \frac{20.6\ \text{hours}}{1\,\text{recharge}} \times \frac{1\ \text{kW}}{1000\,\text{W}} = 1.73\ \text{kW}$$

Knowing the mean electricity price here in Spain as of the day of writing, March 12, 2025 is 0.1162 €/kWh [16]:

$$\text{Electricity cost} = 167\ \text{h} \times 1.73\text{kW} \times \frac{0.1162\ €}{1\,\text{kWh}} = 33.57\ €$$

Connection

Most of the tasks performed during the project will require an internet connection, whether to search information, communicate with the tutor, etc.

The cost of the router used for home WiFi is 25€ / month. Since the router is paid whether we are using the connection or not throughout the month, and we can expect the researcher to use it during the months from january to june, the operation is quite simple:

$$\text{WiFi cost} = 6\ \text{months} \times \frac{25\ €}{1\,\text{month}} = 150\ €$$

Location

The location of the researcher is not considered as a cost for this project since his apartment has already been paid in full, and therefore no monthly costs have to be taken into consideration.

Travel

The public transport shall be used at times to go to the library if interesting material is found there or to meet up with the thesis tutor in person. The researcher will need a T-jove to travel from the months of april to june, as his expires in the month of april. This adds 44€ to the budget.

Consumables

The only consumables used in the project are a notebook and the included pen, which are used to take notes and mark important dates during the project. However, these were given for free during an event at the college campus (FIB Visiona's Feria de empresas 2024) and as so, added no extra cost to the project.

The complete table of indirect costs is shown next: A.5

Resource	**Cost**
WiFi	150€
Travel	44€
Electricity	33.57€
Total	227.57€

Table A.5: Indirect costs, self elaborated

A.2.5 Contingency

To cover uncertain circumstances while on course, we have to increase our expected budget by around 10% per resource type.

Source	**Cost**	**10%**
Human Resources	15,256.62€	1,525.7 €
Material Resources	58.24€	5.82 €
Indirect costs	227.57€	22.76 €
Total	-	1,554.28€

Table A.6: Contingency costs, self elaborated

A.2.6 Incidental costs

The last group of costs to consider are those coming from unexpected and/or unfortunate circumstances. We can separate these between both roles the researcher shall play, and adding in how probable we believe problems may arise in that position.

Role	**Extra hours**	**Cost + S. Sec.**	**Probability**
Project Manager	15 hours	421.81 €	10%
ML Engineer	20 hours	583.47 €	20%
Total	–	1005.28 €	–

Table A.7: Incidental costs, self elaborated

A.2.7 Total budget

The final table of this section summarizes and adds all of the costs mentioned up until now, which shall give us the total cost of the project A.8.

Source	Cost
Human resources	15,256.62€
Contingency costs	1,554.28€
Incidental costs	1,005.28€
Indirect costs	227.57€
Material resources	58.24€
Software resources	0€
Total	18,101.99€

Table A.8: Total budget, self elaborated

A.2.8 Control Management

With our budget, we now have two variables two keep track of, these being the cost and the time. To manage deviations from our expectations, we add two indicators to our project:

$$\text{Cost Deviation (CD)} = Cost_{expected} - Cost_{real}$$

$$\text{Efficiency Deviation (ED)} = Time_{expected} - Time_{real}$$

Per task done, we shall check the level of deviation from our expectations both for cost (CD) and efficiency (CD). Our objective is to remain between neutral and preferably positive values. If any of the two variables grow too negatively, a reschedule of tasks must be made to alleviate said problems.

A.3 Sustainability Report

After taking EDINSOFT's anonymous survey, the sustainability challenges of this project were analyzed in terms of economic, environmental, and social impacts. The following sections discuss each of these dimensions as they relate to the development and potential influence of the project.

A.3.1 Economic Dimension

The CPNN project has been designed with economic sustainability in mind. The budget estimates—based on careful accounting of human and material resources—demonstrate that the project is cost-effective. By leveraging open-source software (e.g., PyTorch) and utilizing existing hardware (such as the researcher's personal laptop), the project minimizes the need for expensive proprietary tools and additional equipment. Moreover, by revisiting an underexplored neural network architecture, the project seeks to provide an alternative solution that may reduce computational and licensing costs compared to more mainstream, resource-intensive models. These factors collectively ensure that the project remains economically viable and potentially attractive for applications in both academic and industrial settings.

A.3.2 Environmental Dimension

The environmental impact of the CPNN project is inherently low due to its computational focus. Most experiments and implementations are conducted on existing hardware and shared computational resources, which minimizes the need for new, energy-intensive equipment. Additionally, if the CPNN approach proves to be more efficient—by reducing training times or computational load—its wider adoption could lead to a decrease in overall energy consumption compared to traditional neural network methods. In this way, the project not only keeps its own environmental footprint small but also paves the way for greener AI solutions that can contribute to a reduction in the carbon footprint of large-scale machine learning operations.

A.3.3 Social Dimension

Social sustainability is a key consideration for the CPNN project. The work fosters significant personal and professional growth for the researcher by deepening expertise in both classical and modern AI methodologies. Beyond individual benefits, the project contributes to the academic community by revitalizing a historically significant neural network model and making it accessible through an open-source implementation. This democratization of technology encourages collaboration and knowledge sharing, potentially leveling the playing field for institutions and professionals who might not have access to expensive commercial solutions. In turn, these efforts support a more inclusive technological landscape and promote the widespread adoption of innovative, cost-effective AI solutions.