*Project 2 Briefing:*

The Basics of Neural Networks

Author: Iván Blanco (Noax Trading). Date 09/05/2025

# Project objective:

To introduce students to the structure, logic, and training of basic neural networks. The project begins with the perceptron—the foundational unit of modern networks—and builds toward a functional multi-layer feedforward network trained with backpropagation. Students will implement core neural network logic from scratch before transitioning to PyTorch. The Heart Disease dataset will serve as a unifying reference across the four notebooks.

# Development Constraints:

* You must not use any external data sources or external APIs in any part of the notebooks.
* All examples, exercises, and model training must be based exclusively on the provided CSV file: <https://archive.ics.uci.edu/dataset/45/heart+disease>
* The use of internet connections, dynamic data fetching, or third-party API-based datasets is strictly prohibited.
* Book (only for reference and NOTATION): https://udlbook.github.io/udlbook/

# Project Structure

This project consists of four progressively structured Jupyter notebooks:

|  |  |  |
| --- | --- | --- |
| Notebook | Title | Core Topic |
| 1 | The Perceptron and Binary Classification | Single neuron + binary classification task |
| 2 | Feedforward Neural Networks (FNN) | Forward pass through a network (manual) |
| 3 | Backpropagation from Scratch | Backward pass and weight updates via code |
| 4 | Training FNN with PyTorch | Model, loss and optimizer using PyTorch framework |

# Notebook 1 – The Perceptron and Binary Classification

**Learning Objectives:**

- Understand the structure and computation of a single perceptron.  
- Learn how to manually implement binary classification using the perceptron rule.  
- Visualize decision boundaries in a 2D space.  
- Understand the limitations of linearly separable models.

**Developer Instructions:**

- Introduce the perceptron model mathematically and conceptually (include equation and diagram).  
- Use a simplified binary classification version of the Heart Disease dataset (e.g., two features and a binary label).  
- Visualize the data distribution.  
- Implement the perceptron update rule manually in NumPy and PyTorch.  
- Show how decision boundaries shift over iterations.  
- Conclude by discussing why some data may not be linearly separable and motivate deeper models.

# Notebook 2 – Feedforward Neural Networks (FNN)

**Learning Objectives:**

- Understand how layers of neurons are connected in a feedforward neural network.  
- Learn how to implement the forward pass manually with multiple neurons.  
- Explore how non-linear activation functions expand model capacity.  
- Visualize output space transformation by hidden layers.

**Developer Instructions:**

- Define a network manually with one hidden layer and one output.  
- Use NumPy (first) and PyTorch without autograd to implement linear transformations + activation.  
- Visualize the predictions and decision surfaces for a 2D subset of the dataset.  
- Add activation functions like ReLU or Sigmoid and show how they change the outcome.  
- Compare predictions with and without non-linearities.

# Notebook 3 – Backpropagation from Scratch

**Learning Objectives:**

- Understand how errors are propagated backward through a network.  
- Implement the backward pass manually using the chain rule.  
- Explore the effect of learning rate on convergence.  
- Visualize gradients and their impact on weight updates.

**Developer Instructions:**

- Provide symbolic explanation of the backpropagation steps using LaTeX.  
- Extend the manual network from Notebook 2 with backward computation.  
- Use simple architecture and input dimensions for clarity.  
- Plot loss curves over iterations.  
- Highlight how gradients relate to the changes in weights.

# Notebook 4 – Training FNN with PyTorch

**Learning Objectives:**

- Learn how to define and train a neural network using PyTorch tools.  
- Use loss functions and optimizers in `torch.nn` and `torch.optim`.  
- Evaluate model performance on training and test sets.  
- Understand overfitting, underfitting, and tuning model complexity.

**Developer Instructions:**

- Rebuild the model from Notebook 3 using `nn.Module` or `nn.Sequential`.  
- Load and prepare the dataset properly (train/test split).  
- Use `BCELoss` or `BCEWithLogitsLoss` for binary classification.  
- Train the model using a standard optimizer like `Adam` or `SGD` (just use them, details in the next project)  
- Track loss over time and visualize performance (confusion matrix, accuracy).  
- Conclude with Markdown discussion on model capacity and generalization.

# Dataset Requirements

* Use the Heart Disease dataset from the UCI Machine Learning Repository.  
  Filename: heart.csv  
  URL: <https://archive.ics.uci.edu/dataset/45/heart+disease>
* The dataset should be used consistently across all four notebooks.

# Technical and Style Guidelines

- Alternate consistently between Markdown (for theory) and code (for implementation).  
- Markdown cells must include LaTeX-style equations where needed.  
- Each notebook should follow a clear pedagogical progression: introduction, concept, code, reflection.  
- Code must be clean, commented, and executable top-to-bottom.  
- Use visualizations where helpful (e.g., scatter plots, training curves, confusion matrices).

# Deliverables

- Four complete .ipynb notebooks (or Google Colab), one for each part of the project.  
- Clear and structured use of Markdown and code.  
- LaTeX used in Markdown to explain mathematical ideas when needed.  
- Visualizations included where helpful for understanding.  
- Notebooks should be clean, readable, and runnable from top to bottom without error.

**What’s Coming in the Next Project**

**Project 3: Training Neural Networks**

* Cover practical **optimization techniques**, including basic gradient descent, stochastic gradient descent, and mini-batch variants.
* Compare different **loss functions**, such as Mean Squared Error and Cross-Entropy.
* Introduce **regularization techniques** to combat overfitting, including **Dropout** and **L2 regularization**.

**Project 4: Advanced Deep Neural Networks**

* Learn how to build and train **deep networks** with more layers and more complex architecture.
* Introduce classic deep architectures such as **LeNet**, **AlexNet**, and **VGG**.
* Address challenges like the **vanishing and exploding gradient problems**, and introduce basic mitigation strategies (e.g., better initialization, batch normalization).

**Project 5: Convolutional Neural Networks (CNNs) Overview**

* Present the fundamental concepts of **Convolutional Neural Networks (CNNs)**: convolutional layers, pooling layers, and fully connected layers.
* Discuss **real-world applications** of CNNs, particularly in image classification.
* Revisit foundational CNN architectures such as **LeNet**, **AlexNet**, and **VGG** from a structural point of view.

**Project 6: Advanced CNN Architectures**

* Introduce **Transfer Learning** and how pre-trained models can be adapted to new tasks.
* Explain **Data Augmentation** as a technique to improve model generalization.
* Explore advanced architectures such as **ResNet** and **Inception Networks**, focusing on their structural innovations and training efficiency.