Multi-Layer Perceptron's (MLPs): Understanding Depth, Width, and Performance Using Kaggle Dataset

Author: Renusree Kakarla

Student ID: 24087145

Course: Machine Learning and Neural Networks

GIT Repo Link: https://github.com/kakarlarenusree/MLP_Tutorial

Introduction

Artificial neural networks (ANNs) are one of the most important developments in machine learning, enabling algorithms to learn highly non-linear relationships, hierarchical patterns, and abstract concepts. Among these architectures, the **Multilayer Perceptron (MLP)** is the foundational deep-learning model upon which modern architectures—including CNNs, RNNs, transformers, and attention-based networks are built.

This tutorial provides a detailed explanation of how MLPs work, why **depth** (number of hidden layers) and **width** (neurons per layer) affect their performance, and how activation functions allow these networks to learn complex functions. To support these concepts, we apply MLPs to the well-known **Iris dataset** and analyse their performance using diagrams and decision-boundary visualisations.

The aim of this tutorial is not just to demonstrate how MLPs operate but to teach you how to design, tune, and evaluate them systematically so that you can apply MLPs in your own projects.

What is a Multilayer Perceptron ?

An MLP is a type of feedforward neural network, meaning information flows in one direction from input to output through a sequence of layers:

1. **Input Layer**

   Receives numerical features of the dataset.

2. **Hidden Layers**

   Each neuron computes a weighted sum of its inputs, applies an activation function, and passes the result to neurons in the next layer.

3. **Output Layer**

   Produces final predictions, often using SoftMax for classification.

Mathematical Structure

Each neuron performs:

$$z = w^T x + b$$

$$a = \phi(z)$$

Where:

- w = weight vector
- b = bias
- x = inputs
- ϕ = activation function
- a = output (activation)

Layers are stacked so that the output of one layer becomes the input to the next:

$$a(l) = \phi(W(l)a(l-1) + b(l))$$

This compositional structure allows deep networks to learn hierarchical representations, layers detect simple features, while later layers combine them into complex patterns.

The Role of Activation Functions:

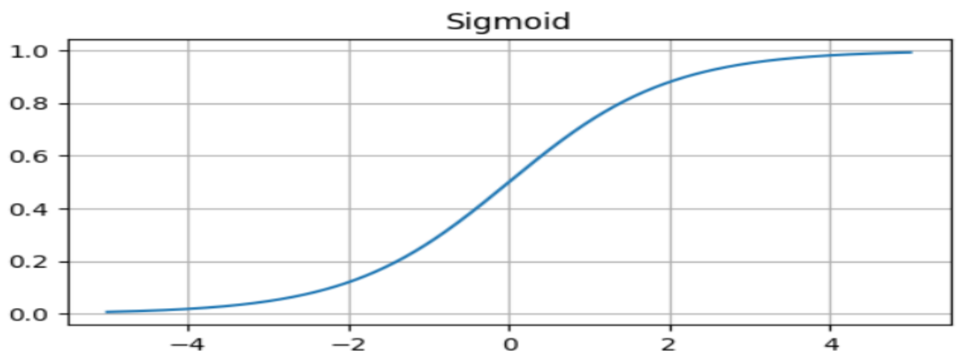
If every layer applied only linear transformations, then no matter how many layers existed, the entire network would collapse into a single linear function. That’s why activation functions are essential.

Sigmoid

Smooth S-shaped curve.
Output range: (0,1).
Used historically in early neural networks.

Limitations:

- Saturates (vanishing gradients).
- Training becomes slow for deep networks.

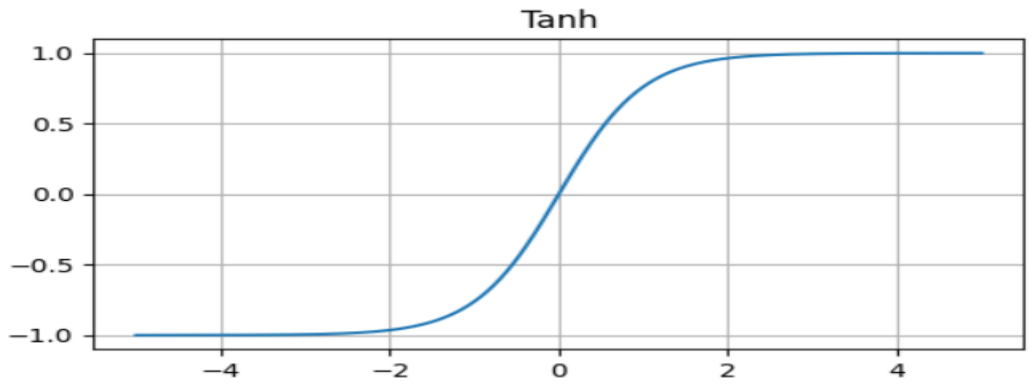


Tanh

Range: (–1,1).
Zero-centred, often preferred over sigmoid.

Still suffers from:

- Vanishing gradients.
- Slow training on deep networks.



ReLU (Rectified Linear Unit)

Most commonly used in modern networks.

$$ReLU(x) = max(0, x)$$

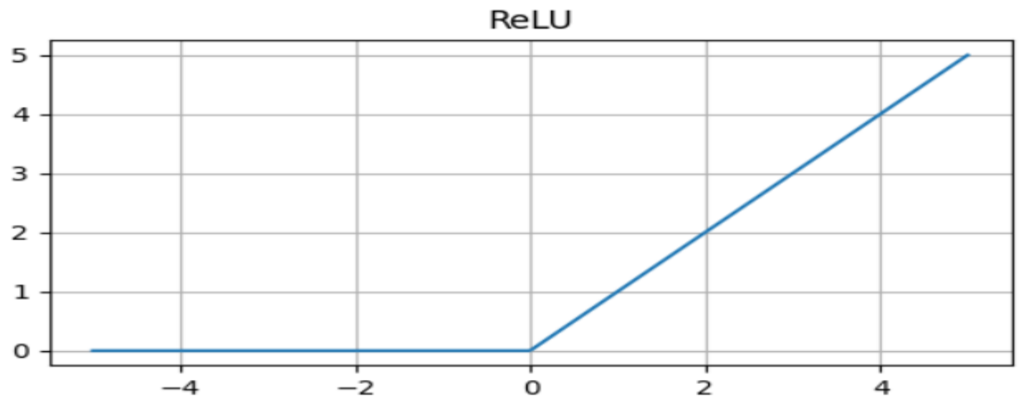
Advantages:

- Does not saturate for positive values (faster training).
- Efficient and simple.

Potential issues:

- “Dead ReLU” problem (neurons stuck outputting 0).

Variants such as Leaky ReLU and ELU aim to fix this by allowing small negative outputs.



Depth vs Width: How Architecture Affects Learning

Designing an MLP architecture involves choosing:

- How many layers (depth)
- How many neurons per layer (width)

Both dramatically influence model behaviour.

Width

Increasing the number of neurons allows the network to:

- Capture more variation in the data
- Fit more complex boundaries

But too much width causes:

- Overfitting
- More parameters (slower training, higher memory use)

Depth

Adding layers allows networks to learn hierarchical structures.

For example:

- Layer 1: simple features (edges)
- Layer 2: patterns from combinations of edges
- Layer 3: object-level concepts

This mirrors the example from your lecture were stacking perceptron's builds more abstract representations.

However:

- Deep networks are harder to train
- More prone to vanishing/exploding gradients
- Require good initialization and normalization

Experimental Setup Using the Iris Dataset

We use the sklearn version of the Iris dataset, consisting of 150 samples from 3 plant species. For visualisation, we select only two features:

- Petal length

- Petal width

These form a clear 2D space ideal for decision boundary plots.

Data Preprocessing

We perform:

- Train-test split (70/30)
- Feature normalization (StandardScaler)
- Three different MLP configurations

MLP Architectures Testing

Model	**Architecture**	**Notes**
MLP-5	1 hidden layer with 5 neurons	Likely underfits
MLP-50	1 hidden layer with 50 neurons	Good capacity
MLP-50x50	2 hidden layers with 50 neurons each	Risk of overfitting

Decision Boundary Visualisations

Decision boundaries help us understand exactly how the network is separating classes.

MLP-5 (Underfitting)

- Boundary is too simple
- Fails to separate overlapping regions
- Lower accuracy

MLP-50 (Balanced)

- Captures curved boundary
- Best generalization performance

MLP-50x50 (Overfitting Risk)

- Very complex, wiggly boundary
- Fits training noise
- May not generalize well

These observations illustrate the practical effects of depth and width.

Interpreting the Results

Underfitting

Occurs when:

- Model too simple
- Insufficient neurons/layers
- High bias

Symptoms:

- Low training & test accuracy
- Smooth, overly simplistic decision boundaries

Overfitting

Occurs when:

- Model too complex
- Too many parameters vs data

Symptoms:

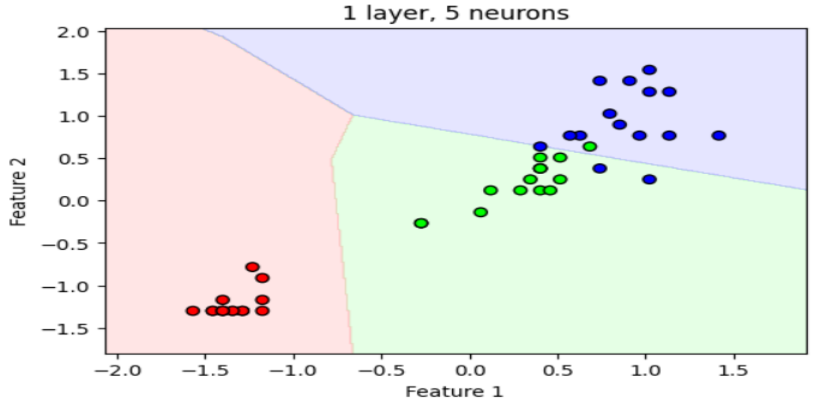
- High training accuracy but lower test accuracy
- Very jagged decision boundaries

Mitigation:

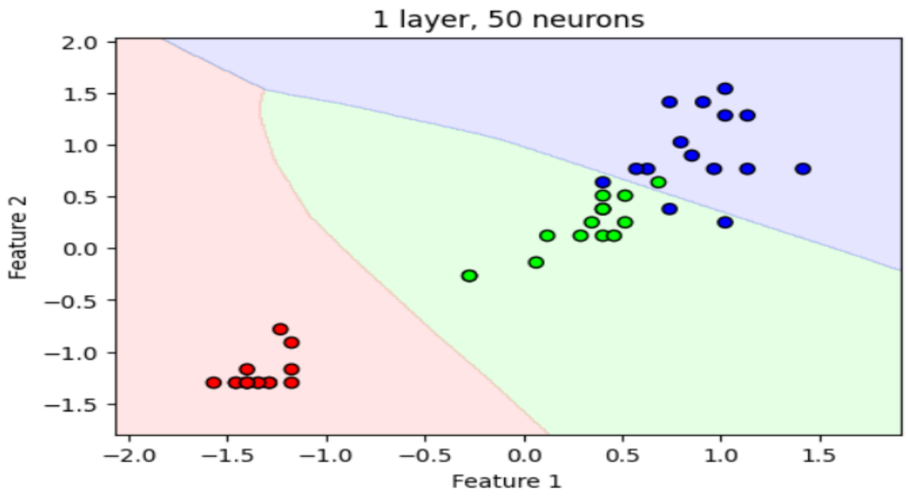
- Regularization
- Dropout
- Reducing width or depth
- Increasing dataset size

Decision boundary plots for three models

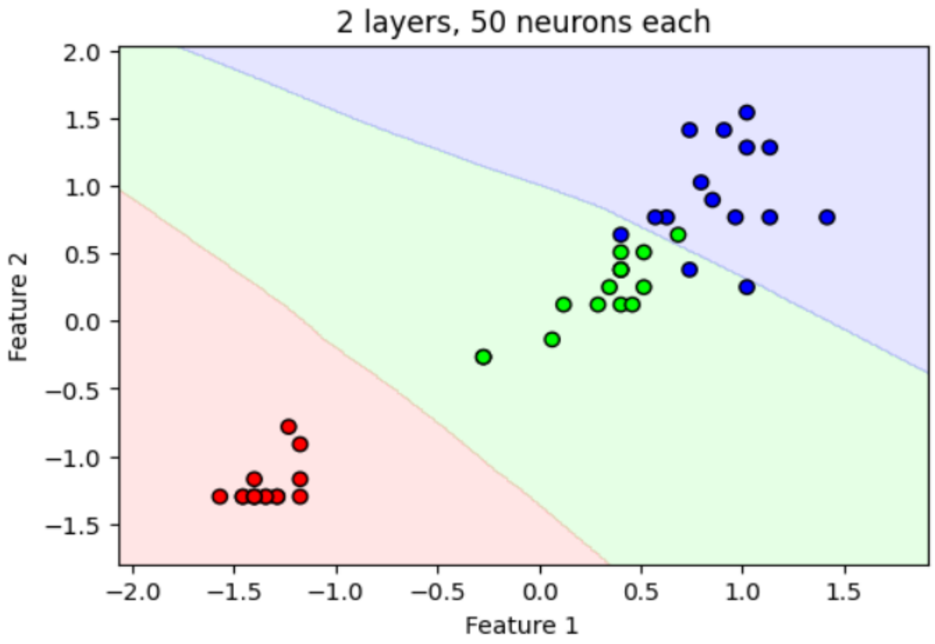
```
1 layer, 5 neurons — Accuracy: 0.91
```



```
1 layer, 50 neurons — Accuracy: 0.91
```



```
2 layers, 50 neurons each — Accuracy: 0.91
```



Balanced Model

The best-performing architecture:

- Learns meaningful curvature
- Avoids modelling noise
- Provides stable accuracy

Best Practices for Designing MLPs

1. Start small — increase complexity only if needed.
2. Use ReLU or ELU activations for hidden layers.
3. Normalize inputs using StandardScaler.
4. Use a validation set to tune hyperparameters.
5. Apply L2 regularization or dropout in deeper/wider networks.
6. Visualize results (loss curves, decision boundaries).

Summary

This tutorial demonstrated:

- How MLPs function mathematically
- Why activation functions are essential
- Differences between depth and width
- How model complexity affects generalization
- Practical visual analysis using real data

MLPs remain fundamental building blocks in modern machine learning, and understanding their behaviour prepares you for more advanced deep-learning architectures.

References

1. Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep Learning. MIT Press.
2. Pedregosa et al. (2011). Scikit-learn: Machine Learning in Python.
3. Iris Dataset (Fisher, 1936).