# 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) have significantly advanced the field of machine learning by allowing computational models to recognise intricate patterns and relationships that are difficult for traditional algorithms to represent. Among the various neural network architectures, the Multilayer Perceptron (MLP) is one of the foundational designs and has inspired the development of modern systems such as convolutional, recurrent, and transformer-based networks.

This tutorial provides a detailed explanation about 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 (MLP) ?

A Multilayer Perceptron is characterized by its forward-only flow of information, where inputs move sequentially through hidden layers toward the output without any backward connections.

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 computes:

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

$$a = \phi(z)$$

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:

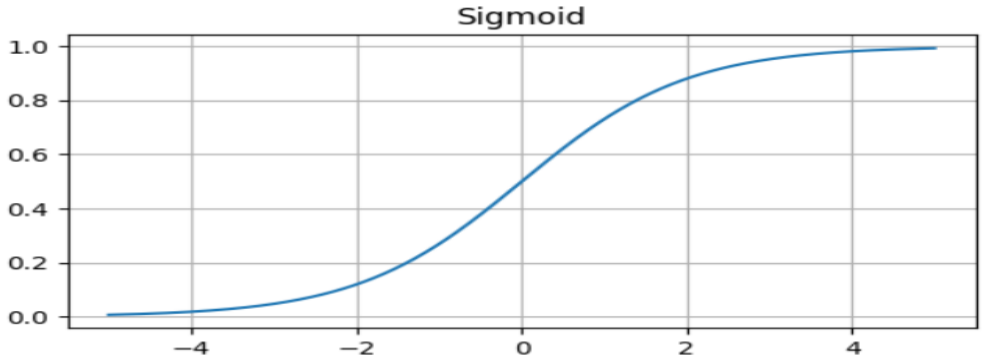
Without activation functions, multiple layers of linear operations would behave like a single linear model, regardless of depth. Activation functions introduce the non-linearity required for learning complex behaviours.

### 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.

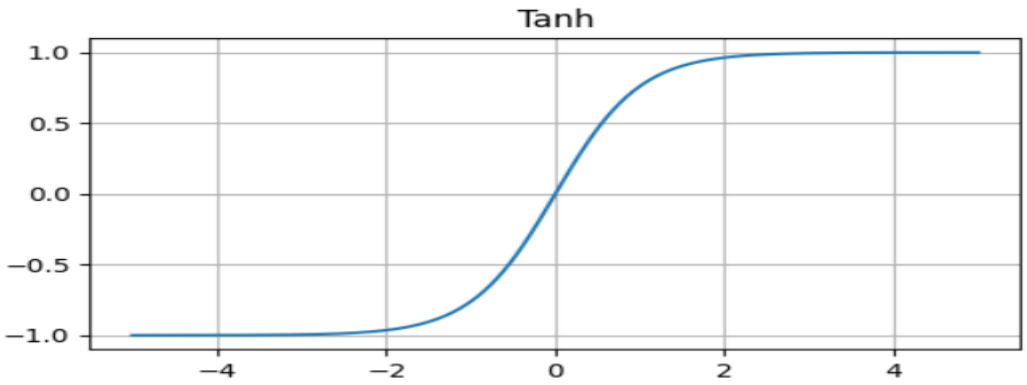


## Tanh

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.

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

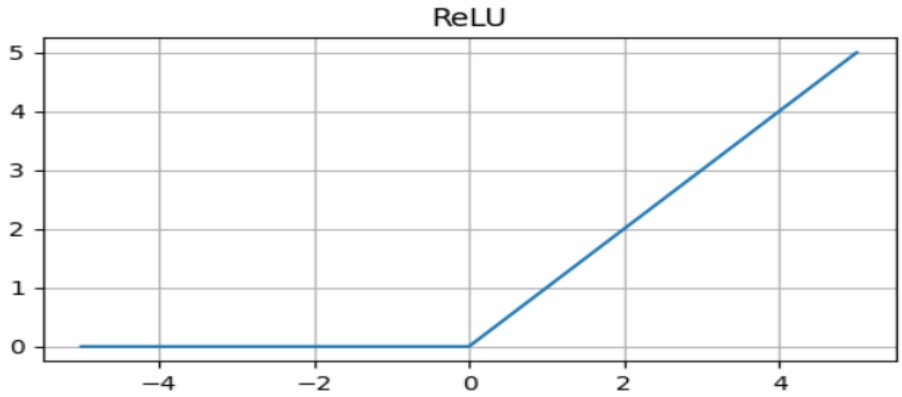
Advantages:

- 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 Explanation

Building an MLP requires:

- Number of hidden layers (depth)
- Number of neurons in each layer (width)

These choices strongly affect how the model learns and generalizes.

## Width

Adding more neurons increases :

- Represent a broader range of underlying patterns.
- 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

The experiment uses 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

The features are standardized, the data is divided into training and testing sets, and three different MLP architectures are evaluated.

- Train-test split
- Feature normalization (StandardScaler)

## 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 data
- High bias

Symptoms:

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

## Overfitting

Occurs when:

- Model too complex
- Too many parameters vs data

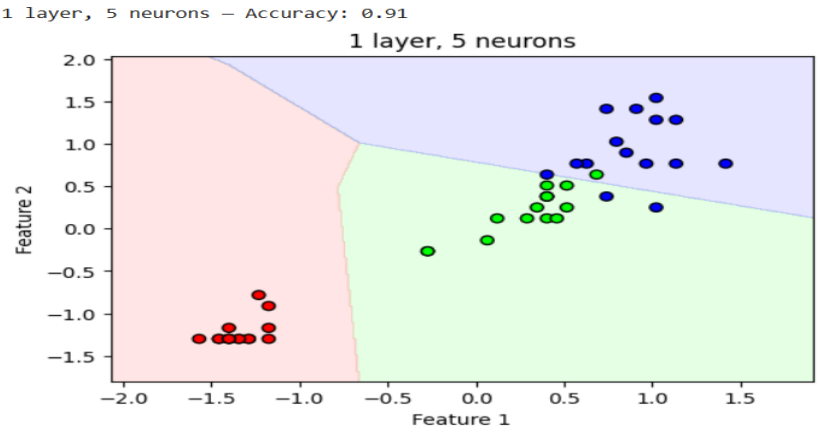
Symptoms:

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

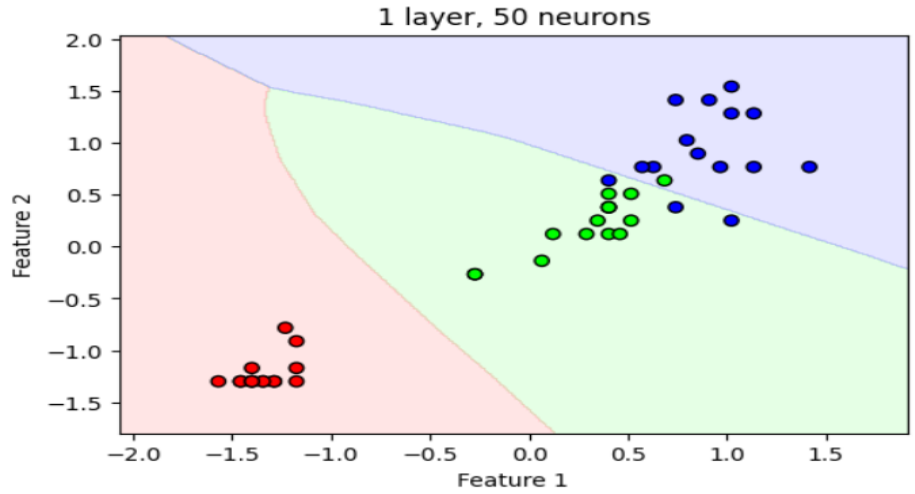
Mitigation:

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

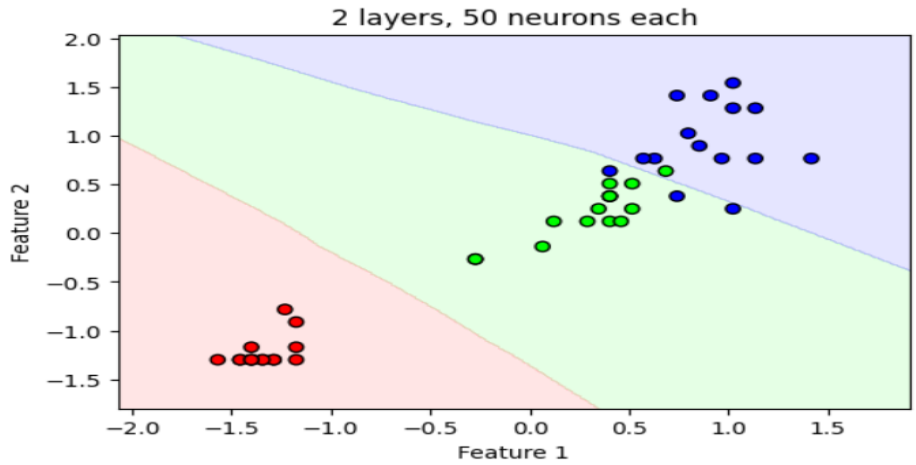
# Decision boundary plots for three models



```
1 layer, 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. Begin with a simple architecture and only expand its size when the baseline model cannot learn the necessary structure.
2. Use ReLU or ELU activations for hidden layers.
3. Normalize inputs using StandardScaler.
4. A separate validation subset should be used to experiment with different hyperparameter.
5. Apply L2 regularization or dropout in deeper/wider networks.
6. Visualize results (loss curves, decision boundaries).

## Summary

This tutorial covered:

- How MLPs work mathematically
- Why activation functions are essential
- How the depth of a network and the number of units within each layer influence the model's learning dynamics and predictive ability.
- 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).