# Multilayer Perceptron (MLP)

`from sklearn.` `neural_network` `import` `MLPClassifier`

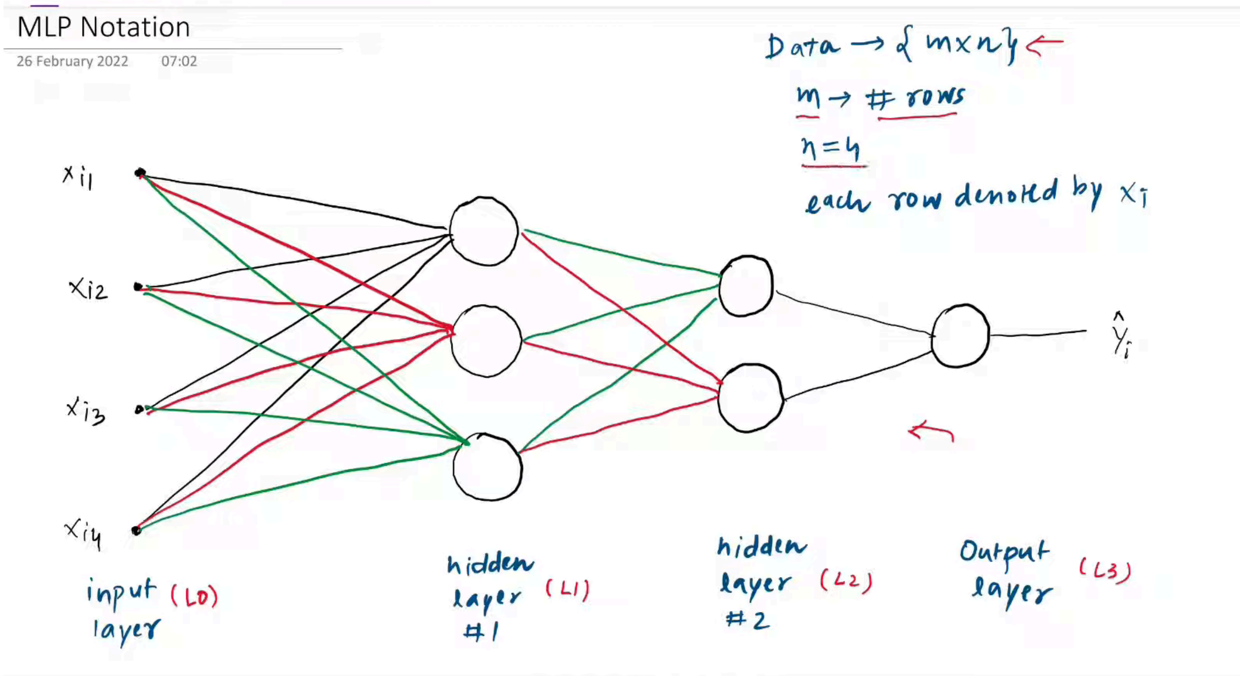
- An **MLP** is a **feedforward neural network(**type of **Artificial Neural Network (ANN))** with **one or more hidden layers** between input and output.
- Unlike a single-layer perceptron, it can solve **non-linear problems** (like XOR) using **backpropagation** and **gradient descent.**

## Structure of MLP

An MLP consists of three types of layers:

- **Input Layer**: Receives raw data (features).
- **Hidden Layer(s)**: Performs computations and extracts patterns.
- **Output Layer**: Produces the final prediction

```
📥 Input Layer → 🟢 Hidden Layer(s) → 📤 Output Layer
```



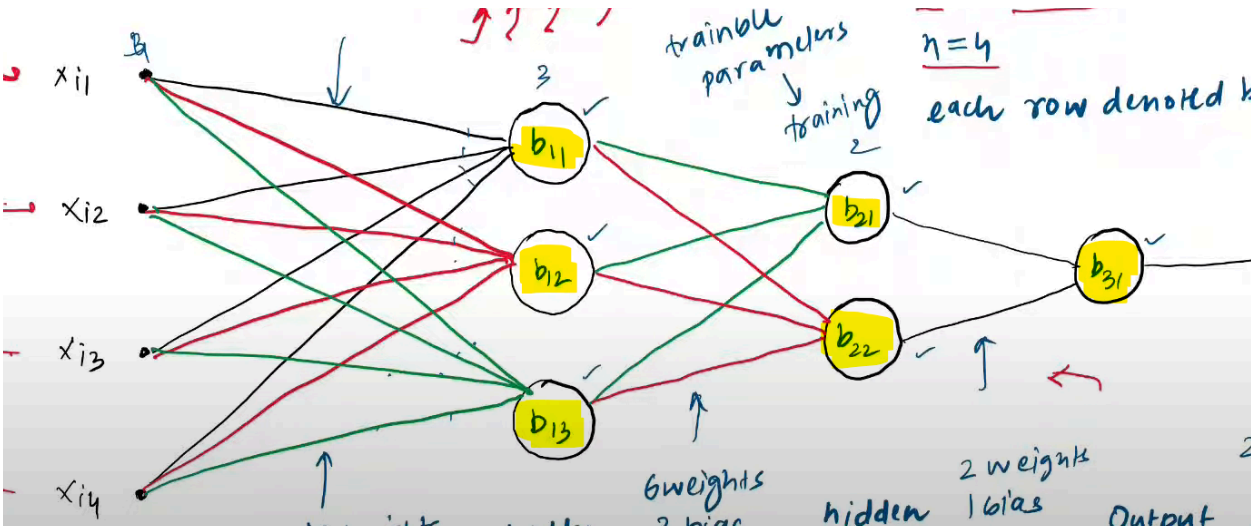

- Find out how much trainable parameters you have
  - When you'll train this algorithm, how many **biases and weights** will be calculated
- In above image:
  - **Weights:**
    - 3∗4 =12
    - 3∗2=6
    - 2∗1=1
    - 1
  - **Biases:**
    - 3
    - 2
    - 1
- Add all the 👆 above weights & biases:

- 12+6+1+3+2+1=26

- **There are 26 trainable parameters.**

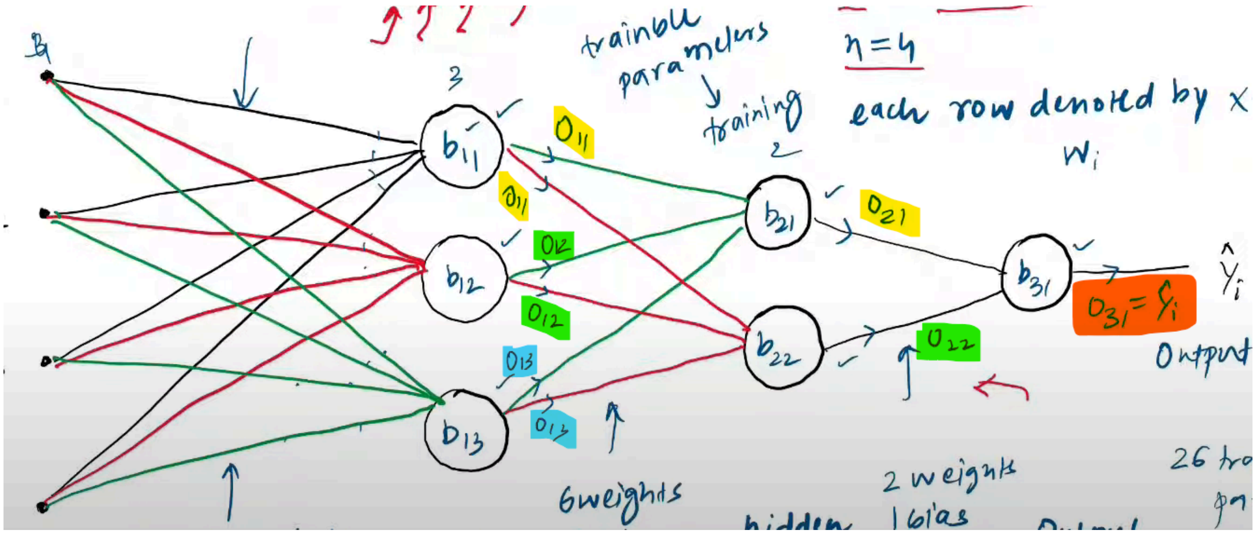
# MLP Notation

- **Bias** → $b_{ij}$
  - $i$ = layer number
  - $j$ = node in the layer $i$




- 👆 $b_{11}$ → First layer, first node

- **Output** → $o_{ij}$




- **Weights**

picture taken from CampusX tutorial of MLP Notation. Thank You :)

Notation: $W_{ij}^h$

where,

i = From which node weight is passing to the next layer's node.

j= To which node weight is arriving.

h = Layer in which weight is arriving.

Example:

$W_{11}^1$ = Weight passing into 1st node of 1st layer of 1st node of the previous layer.

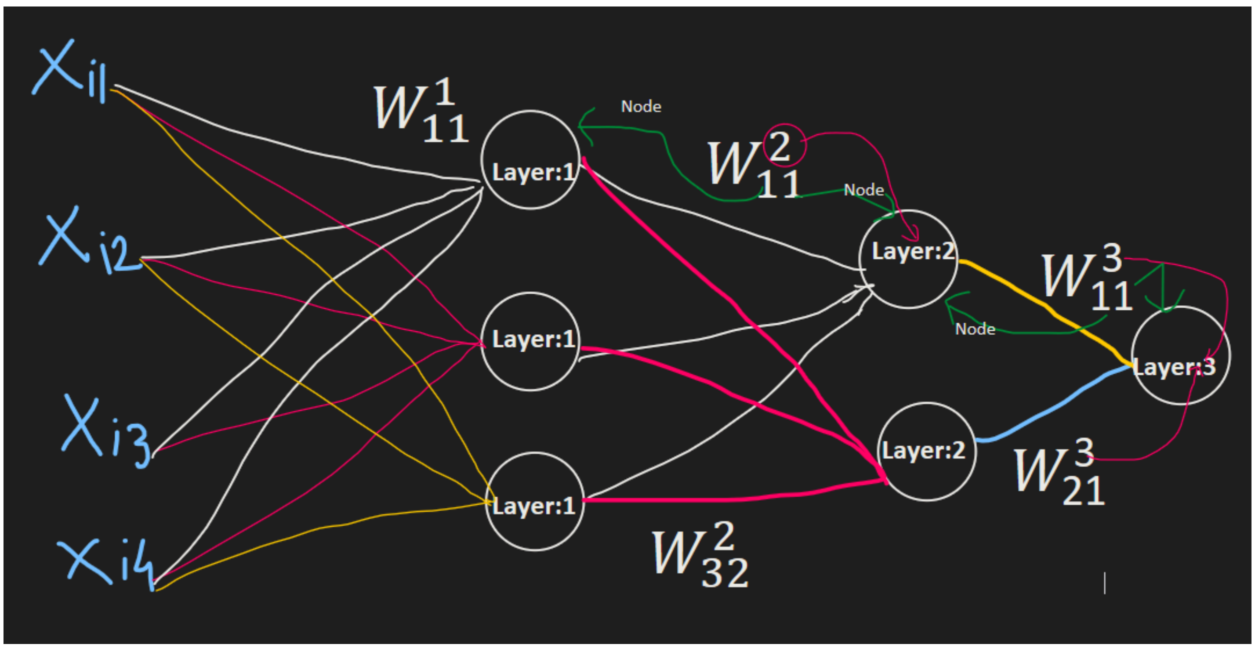
$W_{23}^1$ = Weight passing into the 3rd node of the 1st hidden layer from the 2nd node of the previous layer.

$W_{45}^1$ = Weight passing into the 5th node of the 2nd hidden layer from the 4th node of the previous layer.

3. Notations for Outputs:

$$W_{ij}^{k}$$

- **k**: Layer in which weight is arriving
- **i**: Current layer's **node**
- **j**: **Node** to which it's arriving




## 📊 MLP vs. Single-Layer Perceptron

| Feature | Single-Layer Perceptron | MLP |
|---|---|---|
| **Layers** | Input → Output | Input → Hidden → Output |
| **Non-Linear?** | Fails (XOR problem) | Solves non-linear problems |
| **Training** | Perceptron Trick | Backpropagation + Gradient Descent |

| | | |
|---|---|---|
| **Use Case** | Linear classification | Complex tasks (images, NLP) |

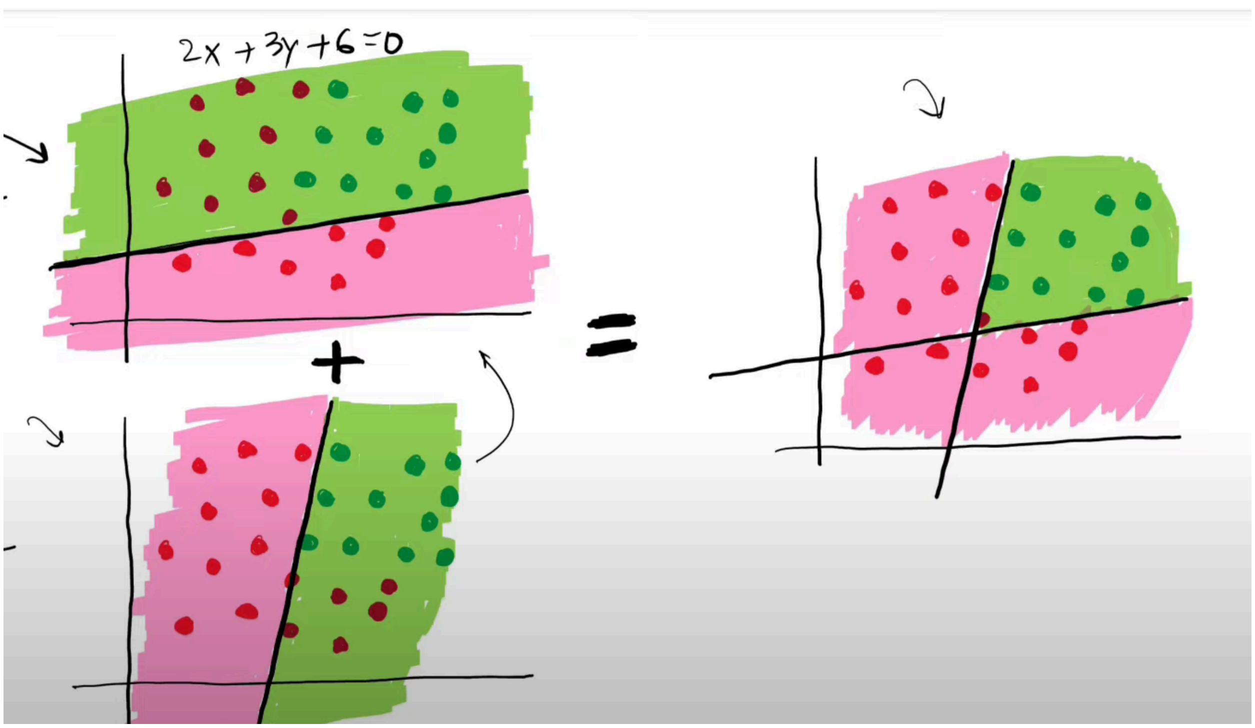
## Loss Functions used:

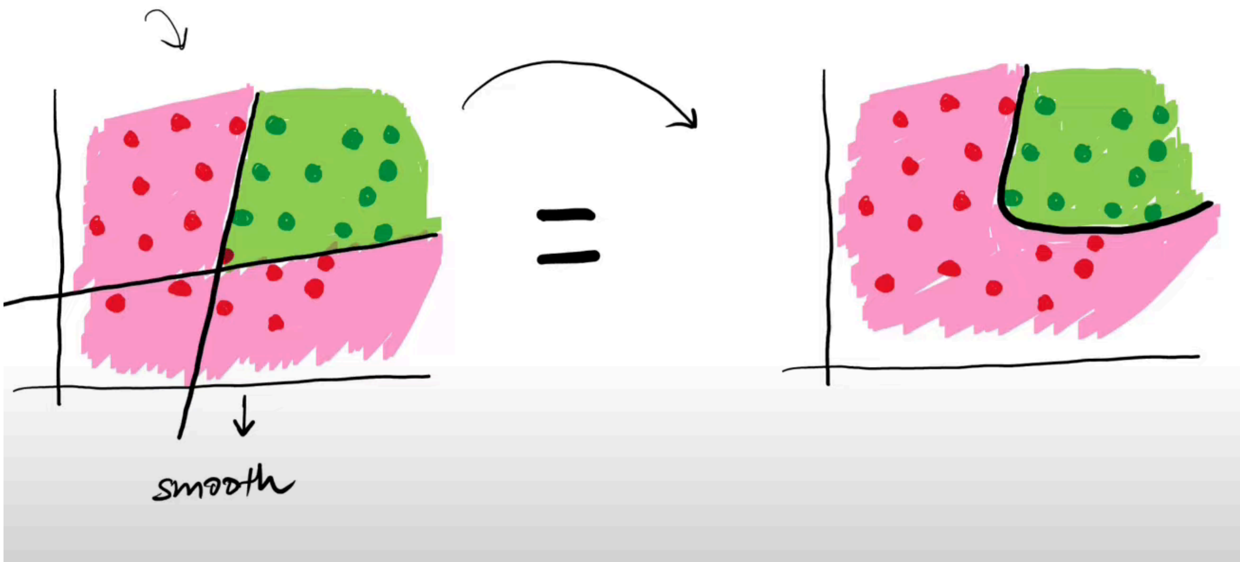
**Sigmoid:** $\sigma(z) = \frac{1}{1+e^{-z}}$ (for probabilities)

**ReLU:** $f(z) = \max(0, z)$ (for deep networks)

**Softmax:** Used in multi-class classification.

- In MLP, we **combine multiple perceptrons** & **smoothen the boundary**.






- We can manipulate importance of 2 perceptrons by multiplying them by weights

$$0.7 \times 10 + 0.8 + 5 = \underset{1}{z}$$

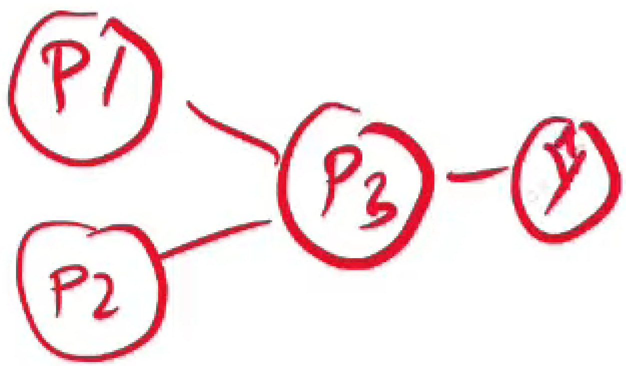
- Here, 10 & 5 are weights of 1st & 2nd perceptron
- After this, we send this to sigmoid function.

> You can also add a bias.
>
> $$\boxed{z} = 0.7 \times 10 + 0.8 \times 5 + 3$$

- 👆 **3** is bias.

$$\text{Sigmoid: } \sigma(z) = \frac{1}{1+e^{-z}} \text{ (for probabilities)}$$

- If σ ≥ threshold (0.5) → 1
- If σ < threshold → 0



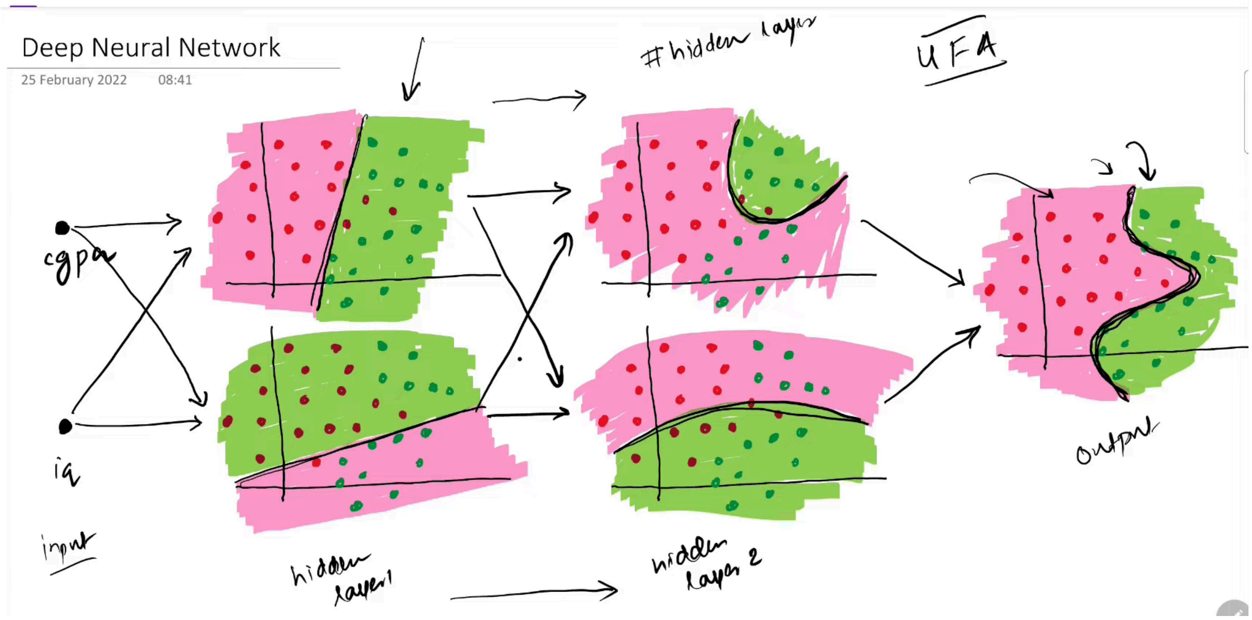

💡 You can add multiple nodes.

- If you add more nodes, more it'll help in defining the non-linear boundary

> Add Multiple output nodes in output layer in case of **multi-class classification** problem.

> You can add No. of hidden layers.

- You can make more complex decision boundary by adding more layers.




## Backpropagation (Learning Process)

- Computes the error (difference between predicted and actual output).
- Uses **Gradient Descent** to update weights and minimize error.
- Adjusts weights using **chain rule of differentiation**.

1. **Forward pass**: Input data is passed through the network's layers to generate an output
2. **Error calculation**: The difference between the input and output is calculated
3. **Backward pass**: The loss is calculated backwards, layer by layer
4. **Weights update**: The weights are updated based on the gradient calculated in the backward pass

# Python code for MLP:

```
from sklearn.neural_network import MLPClassifier
from sklearn.datasets import make_moons
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# Generate dataset
X, y = make_moons(n_samples=1000, noise=0.2, random_state=42)

# Split into train and test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Define MLP model
mlp = MLPClassifier(hidden_layer_sizes=(10, 5), max_iter=1000)

# Train model
mlp.fit(X_train, y_train)

# Predict
y_pred = mlp.predict(X_test)

# Accuracy
print(f"Accuracy: {accuracy_score(y_test, y_pred):.2f}")
```

```
Accuracy: 0.89
```

## `hidden_layer_sizes` (Defining the Network Architecture)

### What it does?

- Specifies the **number of hidden layers** and **neurons per layer** in the MLP.
- Default: `(100,)` → **1 hidden layer with 100 neurons**

| hidden_layer_sizes | Neural Network Architecture |
|---|---|
| (100,) | 1 hidden layer with 100 neurons |
| (50, 30) | 2 hidden layers (50 neurons, 30 neurons) |
| (10, 20, 30) | 3 hidden layers (10, 20, 30 neurons) |
| (150, 100, 50, 25) | 4 hidden layers (150 → 100 → 50 → 25 neurons) |

🔷 **More neurons = More capacity** (but may overfit).

🔷 **More layers = More complexity** (but harder to train).

### `solver` (Choosing the Optimization Algorithm)

### What it does?

- Determines how the model updates weights during training (optimizer).
- Default: `'adam'`

### Available Options & When to Use Them?

| Solver | Algorithm Type | Best For |
|---|---|---|
| `'adam'` (Default) | Adaptive Moment Estimation (fast, adaptive learning rate) | Large datasets, complex problems |
| `'sgd'` | Stochastic Gradient Descent (simple, but slow) | Large-scale problems, online learning |
| `'lbfgs'` | Limited-memory BFGS (second-order optimization) | Small datasets, faster convergence |

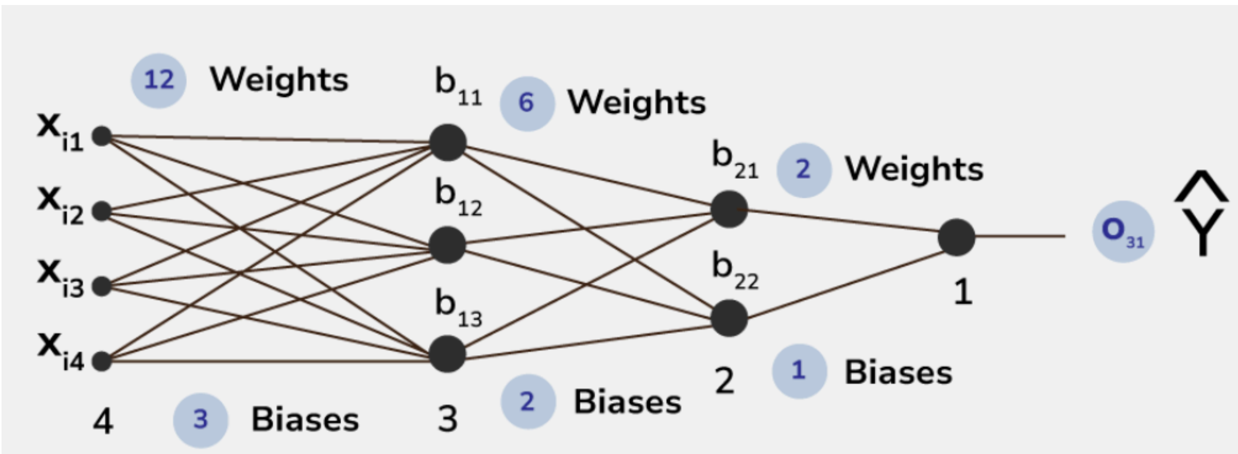
- **Use `'adam'` (default)** if unsure – works well in most cases.
- **Use `'sgd'`** when fine-tuning the learning rate manually.
- **Use `'lbfgs'`** for smaller datasets with fast convergence.

# Forward Propagation (Forward Pass)

- Forward propagation is how an **MLP makes predictions** by passing input data through its layers.

- It involves passing the data through the layers of the network, performing computations at each layer, and generating the predicted result.

💡 **Goal: To compute the predicted output $\hat{y}$ given input $X$ and weights $W$.**




## 📜 Forward Propagation Steps

## 1️⃣ Input Layer → First Hidden Layer

**Input:** $X$

- These are typically the raw data, like pixel values of an image or measurements of some object.

### Weighted Sum (Linear Transformation):

- For each neuron in the hidden layer (and subsequent layers), the input is multiplied by a set of weights. Each connection between neurons has an associated weight

The weighted sum for each neuron is calculated as:

$$z = \sum (x_i \cdot w_i) + b$$

- $x_i$ represents the input values (features),
- $w_i$ represents the weights,
- $b$ is the bias term.

### Activation (e.g., ReLU, Sigmoid, or TanhU):

$$a^1 = \mathrm{ReLU}(z^1)$$

Sigmoid activation:

$$a = \sigma(z) = \frac{1}{1 + e^{-z}}$$

where $\sigma$ is the sigmoid function, and $z$ is the weighted sum.

## 2️⃣ Hidden Layer → Output Layer

- The same process repeats for each hidden layer.
- The final layer computes the output using an activation function (e.g., **Sigmoid for classification, Softmax for multi-class**).

### 3. Example of Forward Propagation (Simple 2-Layer Network)

Assume a neural network with:

- **Input Layer:** 2 neurons
- **Hidden Layer:** 2 neurons
- **Output Layer:** 1 neuron

**Given Inputs:**

$X = [x_1, x_2]$

**Weight Matrices:**

$$W_1 = \begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \end{bmatrix}, W_2 = \begin{bmatrix} w_{31} & w_{32} \end{bmatrix}$$

**Bias Vectors:**

$$b_1 = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}, b_2 = b_3$$

**Step 1: Compute Hidden Layer Activations**

$$Z_1 = W_1 \cdot X + b_1$$

$$A_1 = f(Z_1) \quad \text{(ReLU, Sigmoid, etc.)}$$

**Step 2: Compute Output Layer Activation**

$$Z_2 = W_2 \cdot A_1 + b_2$$

$$\hat{y} = f(Z_2) \quad \text{(Sigmoid, Softmax, etc.)}$$