🌀 Deep Learning Notes – Detailed, Visual & Explained

1. 🧠 Perceptron (1958 – Rosenblatt)

🧠 Definition

A **Perceptron** is the **simplest neural network**, used for **binary linear classification**. It computes a weighted sum of inputs, applies an activation function, and produces a binary output.

⚙️ Working Steps

Inputs: ($x_1, x_2, \ldots, x_n$)
Weights: ($w_1, w_2, \ldots, w_n$)
Bias: (b)
Weighted sum:

```
z = ∑ wi xi + b
```

Activation:

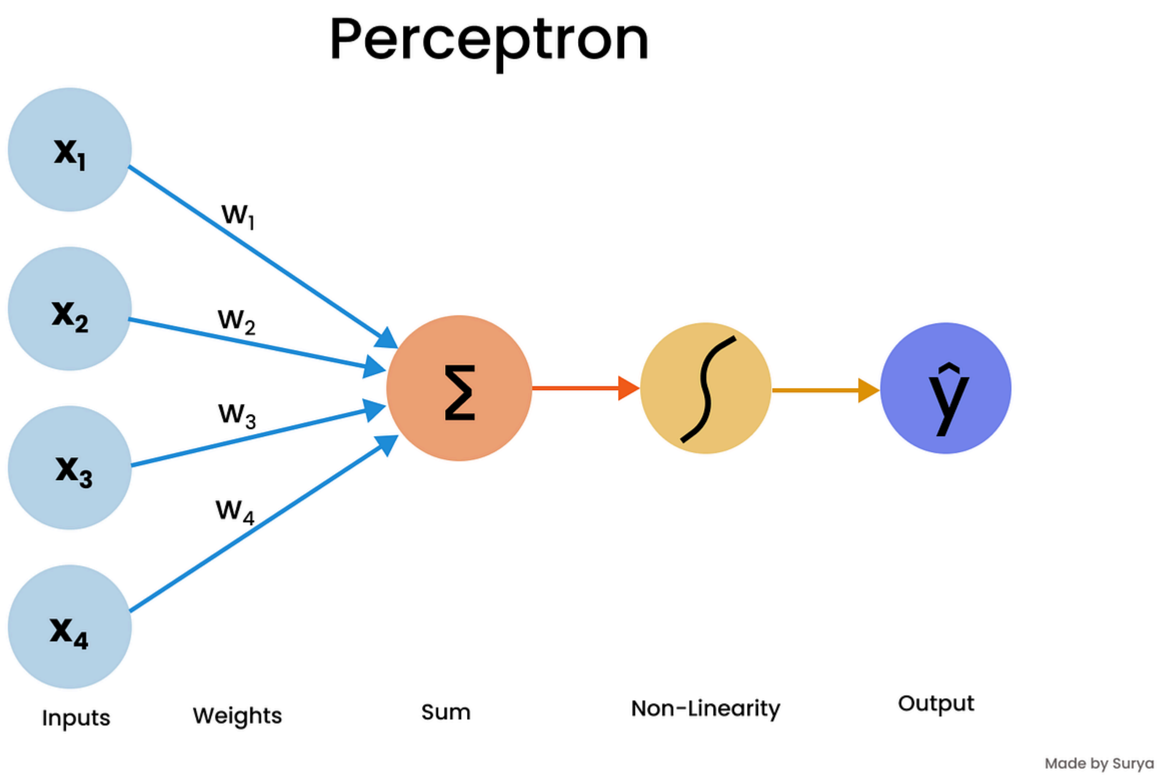
```
ŷ = f(z)
```

🔍 Key Notes

Only works for **linearly separable** problems
Can solve **AND/OR**, but not **XOR**

📊 Mental Diagram

```
Inputs → Weighted Sum → Bias → Activation → Output
```



2. 🧠 Artificial Neural Network (ANN)

🧠 Definition

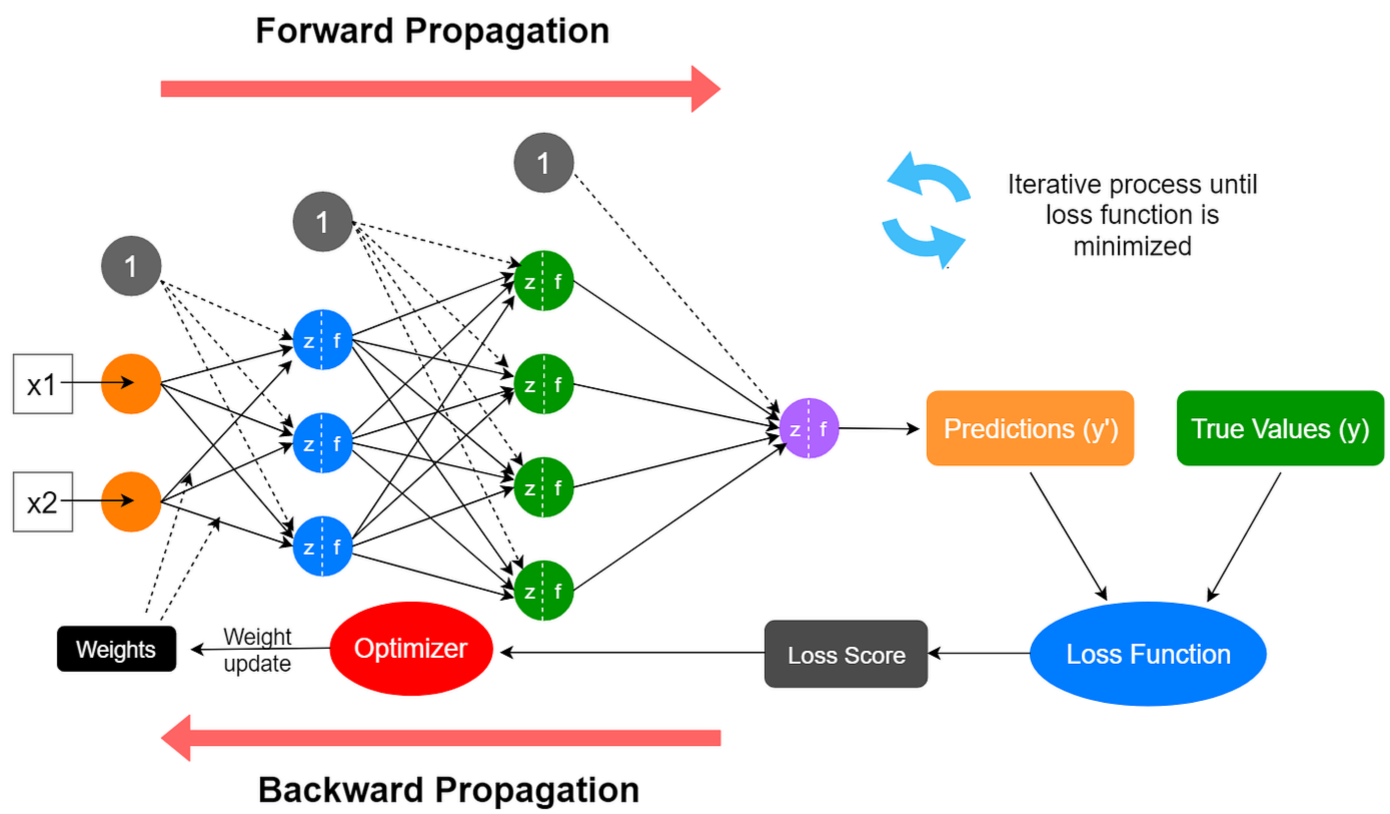
An **ANN** is a **multi-layer neural network** that learns patterns by adjusting weights through **forward and backward propagation**. It can learn **non-linear relationships** between input and output.

🏗️ Structure

Input Layer: Receives features (pixels, text, etc.)
Hidden Layers: Learn intermediate patterns
Output Layer: Generates prediction or label

🔁 Learning Process

Forward Propagation
Loss Calculation
Backward Propagation
Weight Update (via Gradient Descent)



🚀 Why Use ANN?

Learns **non-linear** and **hierarchical** patterns
Can **approximate any function** (Universal Approximation Theorem)
Works for image, text, audio, etc.

📈 Visual Structure

```
Input → Hidden Layer(s) → Output
```

▶️ Forward Propagation

Pass input through each layer
Compute activations:

```
z = w · x + b,  a = f(z)
```

◀️ Backward Propagation

Compute **error (loss)**
Propagate error backward using **chain rule**

Compute gradients and update weights

3. ❌ Vanishing Gradient Problem

❌ Problem

In deep networks, gradients become very small during backprop, especially with **sigmoid/tanh**.
This causes early layers to **stop learning**.

📉 Example

If gradient = 0.5 per layer for 10 layers:

```
(0.5)10 = 0.000976
```

✅ Solutions

Use **ReLU** (or variants)
Apply **Batch Normalization**
Use advanced optimizers like **Adam**, **RMSProp**

4. 🧠 Activation Functions

Activation functions introduce **non-linearity**, enabling neural networks to model complex relationships.

🔷 Linear

Definition: Outputs input as-is
Formula:

```
f(x) = x
```

Range: (-∞, ∞)
Use: Regression
Cons: 🔻 No non-linearity → not useful in deep networks

🔷 Sigmoid

Definition: Squashes values between 0 and 1

Formula:

```
f(x) = 1 / (1 + e-x)
```

Range: (0, 1)
Use: Binary classification
Cons: 🔻 Vanishing gradients, not zero-centered

🔷 Tanh

Definition: Like sigmoid, but output is centered at 0
Formula:

```
f(x) = (ex - e-x) / (ex + e-x)
```

Range: (-1, 1)
Use: Hidden layers
Cons: 🔻 Still vanishes at extreme values

🔷 ReLU (Rectified Linear Unit)

Definition: Activates only for positive inputs
Formula:

```
f(x) = max(0, x)
```

Range: [0, ∞)
Use: Default for hidden layers
Cons: ❌ Dead neurons when (x < 0)

🔷 Leaky ReLU

Definition: Small negative slope for (x < 0)
Formula:

```
f(x) =
  x,    x > 0
  0.01x,    x ≤ 0
```

Use: Avoids dead ReLU neurons

🔷 PReLU (Parametric ReLU)

Definition: Learns the slope for (x < 0)

Formula:

```
f(x) =
  x,    x > 0
  a • x,    x ≤ 0
```

Use: Advanced deep networks

🔷 Swish

Definition: Smooth, non-monotonic function
Formula:

```
f(x) = x * sigmoid(x)
```

Use: Deep learning tasks in NLP & vision

📊 Activation Function Summary

Function	Range	Use Case	Pros	Cons
Linear	(-∞, ∞)	Regression	Simple	🔻 No non-linearity
Sigmoid	(0, 1)	Binary classification	Probabilistic, smooth	🔻 Vanishing gradient
Tanh	(-1, 1)	Hidden layers	Zero-centered	🔻 Still vanishes at extremes
ReLU	[0, ∞)	Most common	Fast, avoids vanishing	❌ Dead neurons
Leaky ReLU	(-∞, ∞)	Hidden layers	Solves dead ReLU	🔻 Minor performance cost
PReLU	(-∞, ∞)	Large networks	Learns best slope	🔻 Adds parameters
Swish	(-∞, ∞)	Deep networks	Smooth, powerful	🔻 Computationally heavier

5. 🧠 Loss Functions

Loss functions tell us **how wrong the model is**.

🔢 Regression Losses

MSE (Mean Squared Error)

```
∑ (y - ŷ)2 / n
```

Penalizes large errors

MAE (Mean Absolute Error)

```
∑ |y - ŷ| / n
```

Robust to outliers

Huber Loss

Combines MSE and MAE for stability

🔤 Classification Losses

Binary Cross-Entropy:

For binary output (sigmoid)

Categorical Cross-Entropy:

For multi-class output (softmax)

6. 🔁 Gradient Descent Variants

Type	Description	Pros / Cons
Batch GD	Uses full dataset	✅ Accurate, 🔻 Slow
Stochastic GD	Updates per sample	✅ Fast, 🔻 Noisy
Mini-Batch GD	Small batch updates	✅ Balanced → ⭐ Most used

7. 🔧 Optimizers

Optimizer	Description	Best For
SGD	Basic gradient descent	Small models
Momentum	Adds inertia to SGD	Faster convergence
Adagrad	Adapts learning rate per weight	Sparse features
RMSProp	Exponentially decaying avg of grads	RNNs
Adam	RMSProp + Momentum (default choice)	Deep networks

8. 📊 Evaluation Metrics

Metric	Meaning
`loss`	Training error
`val_loss`	Validation error (unseen data)
`accuracy`	Correct predictions / total samples
`val_accuracy`	Accuracy on validation set

🖼️ Optional Visual Notes (Mental PNG style)

```
Activation Functions:

Linear      : straight line
Sigmoid     : S-curve (0 to 1)
Tanh        : S-curve (-1 to 1)
ReLU        : flat 0 for x<0, linear x>0
Leaky ReLU  : small slope for x<0, linear x>0
PReLU       : slope learned for x<0
Swish       : smooth curve, rises slowly negative → positive
```