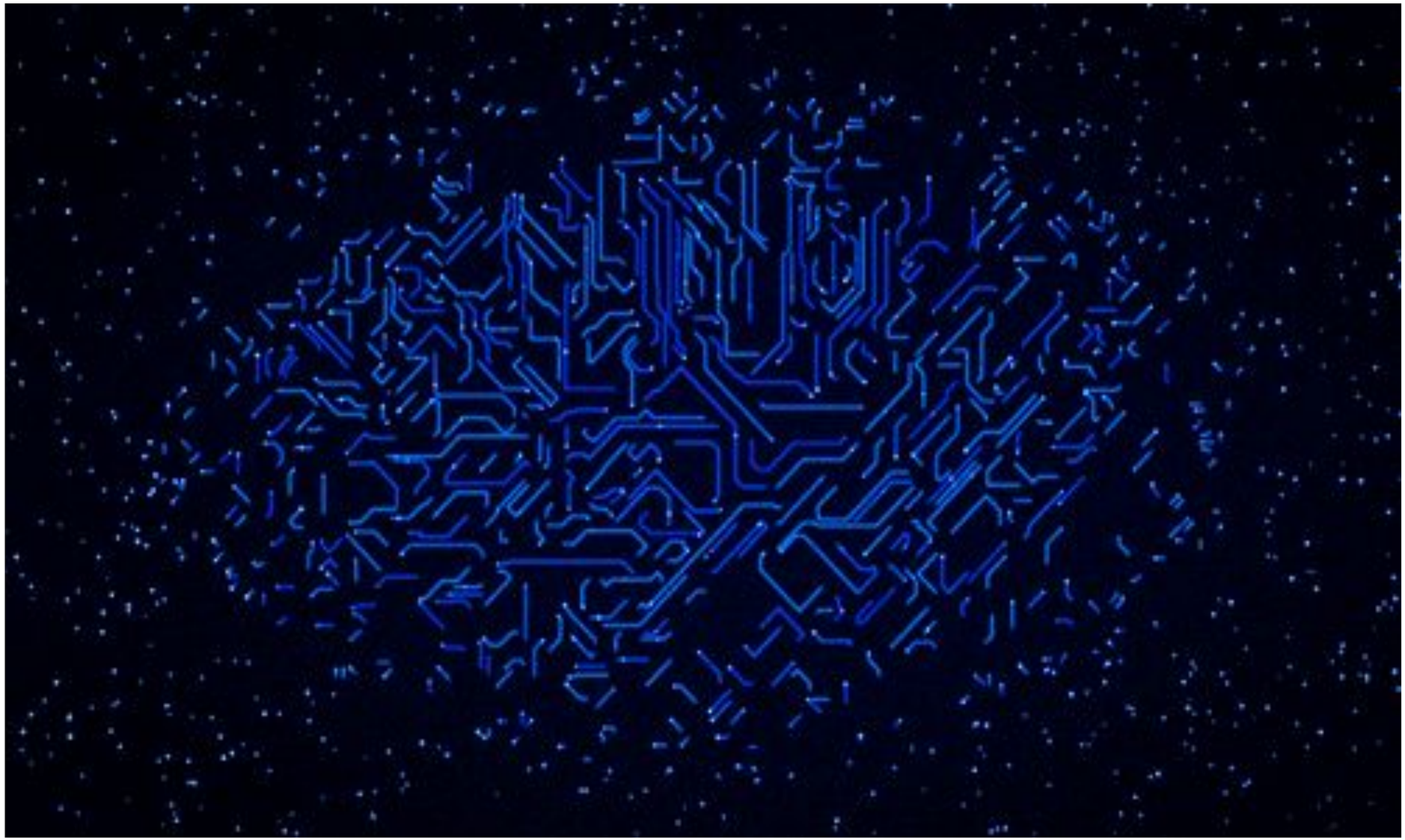


# UNIT 2 - Learning Algorithms

# Contents

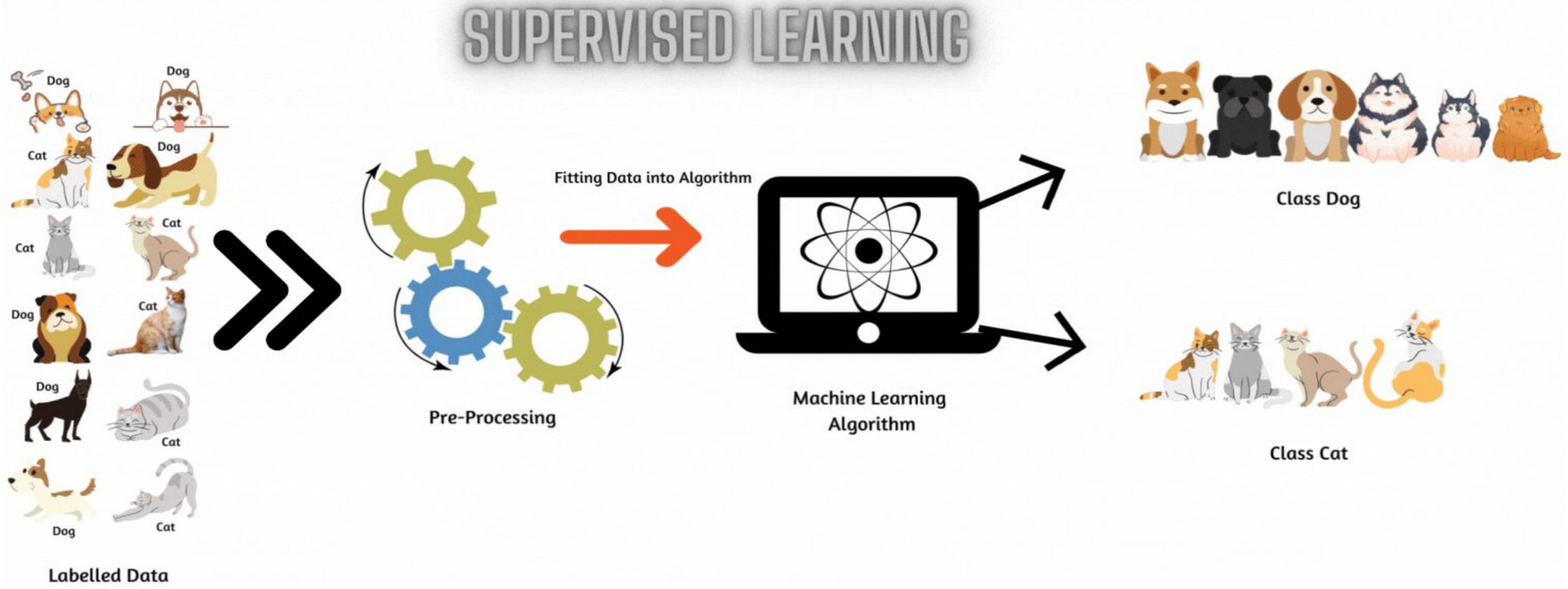
- Learning and Memory, Learning Algorithms,
- Numbers of hidden nodes,
- Error Correction and Gradient Decent Rules, Perceptron Learning Algorithms,
- Supervised Learning Backpropagation,
- Multilayered Network Architectures,
- Back propagation Learning Algorithm,
- Feed forward and feedback neural networks,
- example and applications.
- #Exemplar/Case Studies Medical diagnosis, Automated trading systems

# Learning and Memory

- Neural networks learn patterns and make predictions by training on data. There are different learning paradigms used based on the nature of data and tasks.

- Types of Learning in Neural Networks:
  - Supervised Learning
  - Unsupervised Learning
  - Semi-Supervised Learning
  - Reinforcement Learning
  - Self-Supervised Learning

# Supervised Learning



- **Supervised Learning is a type of learning where a model is trained using labeled data. Each input in the dataset has a corresponding correct output (label), and the model learns to map inputs to outputs based on this labeled information.**

- Key Features:
  - Requires labeled dataset
  - Uses a loss function to measure errors
  - Optimized through back propagation and gradient descent
  - Examples: Image classification, speech recognition, spam detection

- A network is fed with a set of training samples (inputs and corresponding output), and it uses these samples to learn the general relationship between the inputs and the outputs.
- This relationship is represented by the values of the weights of the trained network.

# Supervised Learning - Example: Image Classification

**Example: Classifying images of cats and dogs.**

**1. Dataset Preparation:**

- Input: Thousands of images labeled as 'Cat' or 'Dog'.
- Output: Model learns to classify images correctly.

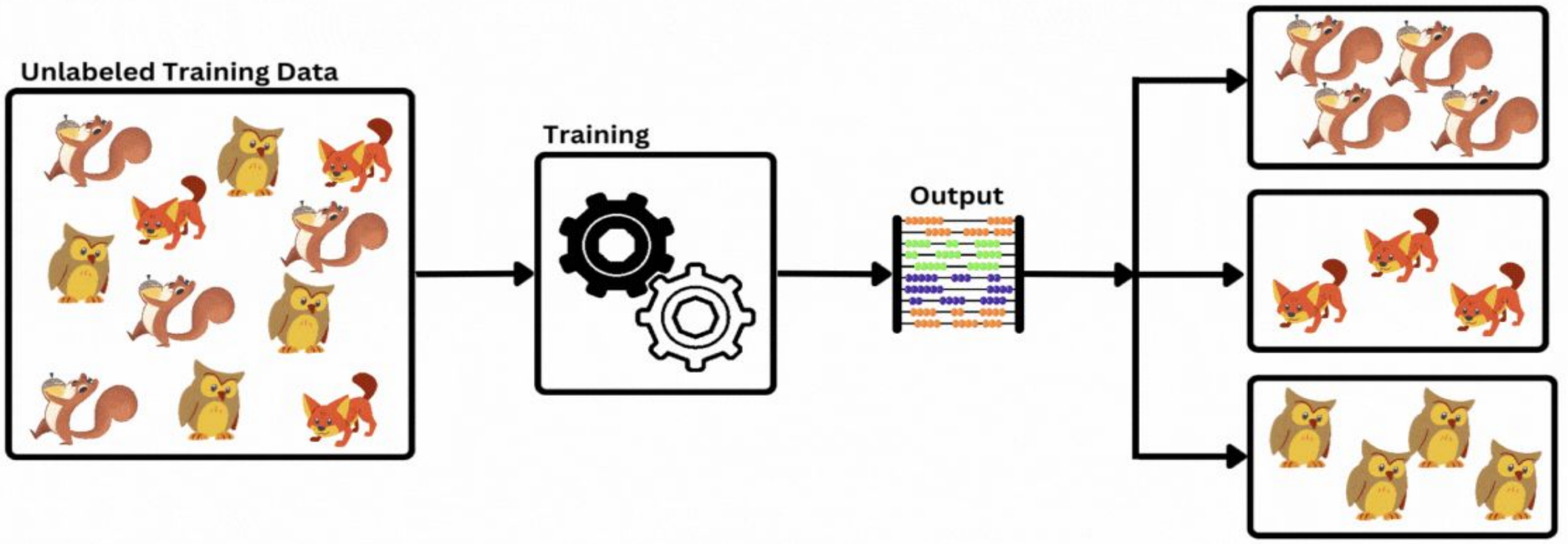
**2. Training Process:**

- The model extracts features (e.g., edges, colors, shapes).
- Adjusts weights to minimize classification errors.
- Uses loss function (e.g., cross-entropy) to improve accuracy.

**3. Prediction:**

- Given a new image, the model predicts whether it's a cat or a dog.

# Unsupervised Learning



# Unsupervised Learning

- **Unsupervised Learning is a type of learning where the model is trained on an unlabeled dataset. The algorithm identifies patterns, structures, and relationships in the data without predefined labels.**

- Key Features:
  - No labeled data required
  - Focuses on clustering and pattern recognition
  - Examples: Customer segmentation, anomaly detection, topic modelling
- No desired output is associated with the training data!
- Faster than supervised learning

# Unsupervised Learning - Example: Customer Segmentation

**Example: Grouping customers based on shopping behavior.**

**1. Dataset Preparation:**

- Input: Purchase history, browsing data, demographics (no labels).

**2. Clustering Process:**

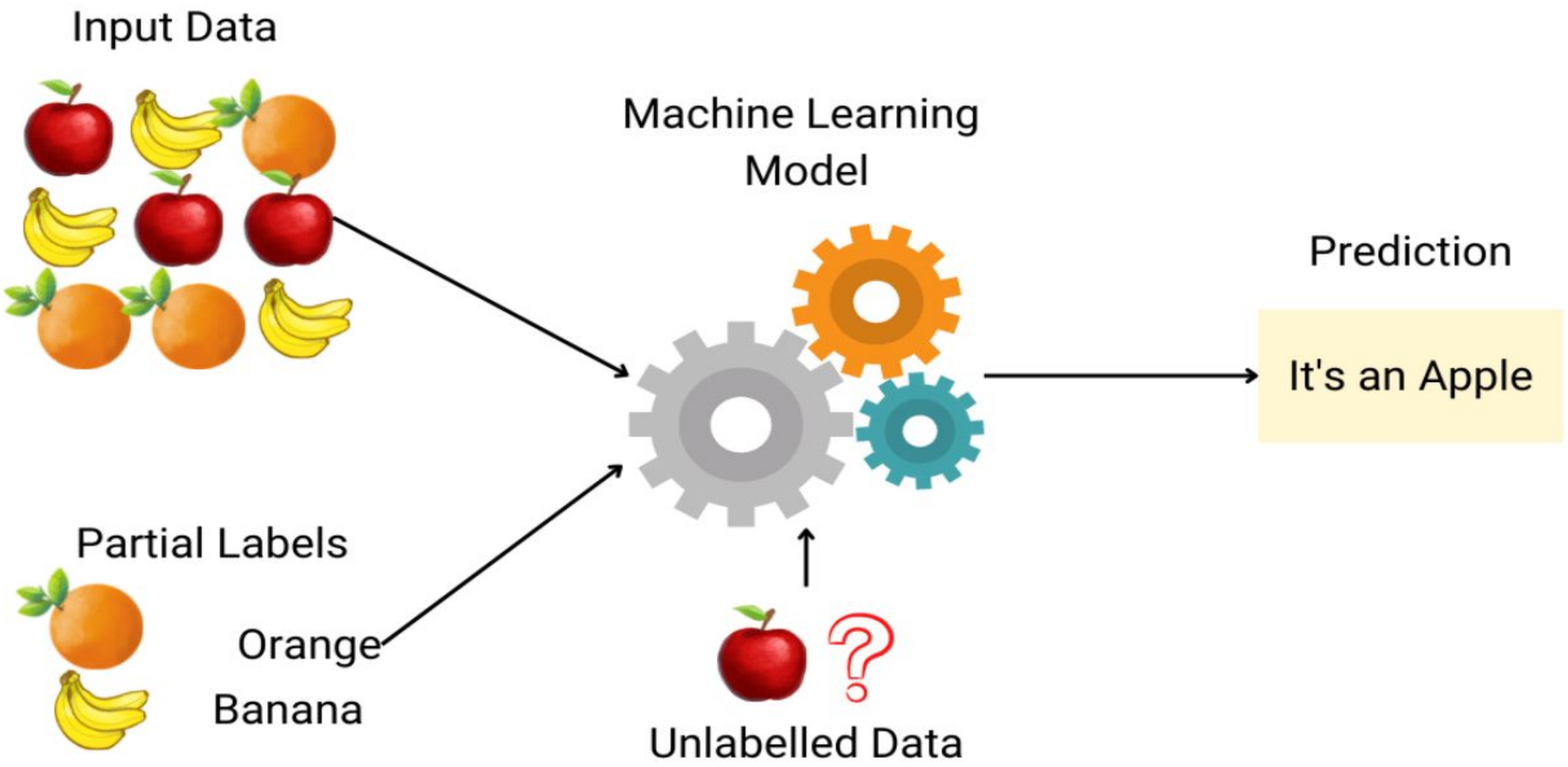
- Model groups similar customers into clusters.
- Uses algorithms like K-Means or DBSCAN.

**3. Business Application:**

- Helps in targeted marketing strategies.
- Personalizes recommendations based on customer clusters.

- Unsupervised learning is crucial for discovering hidden patterns in data.

# Semi-Supervised Learning



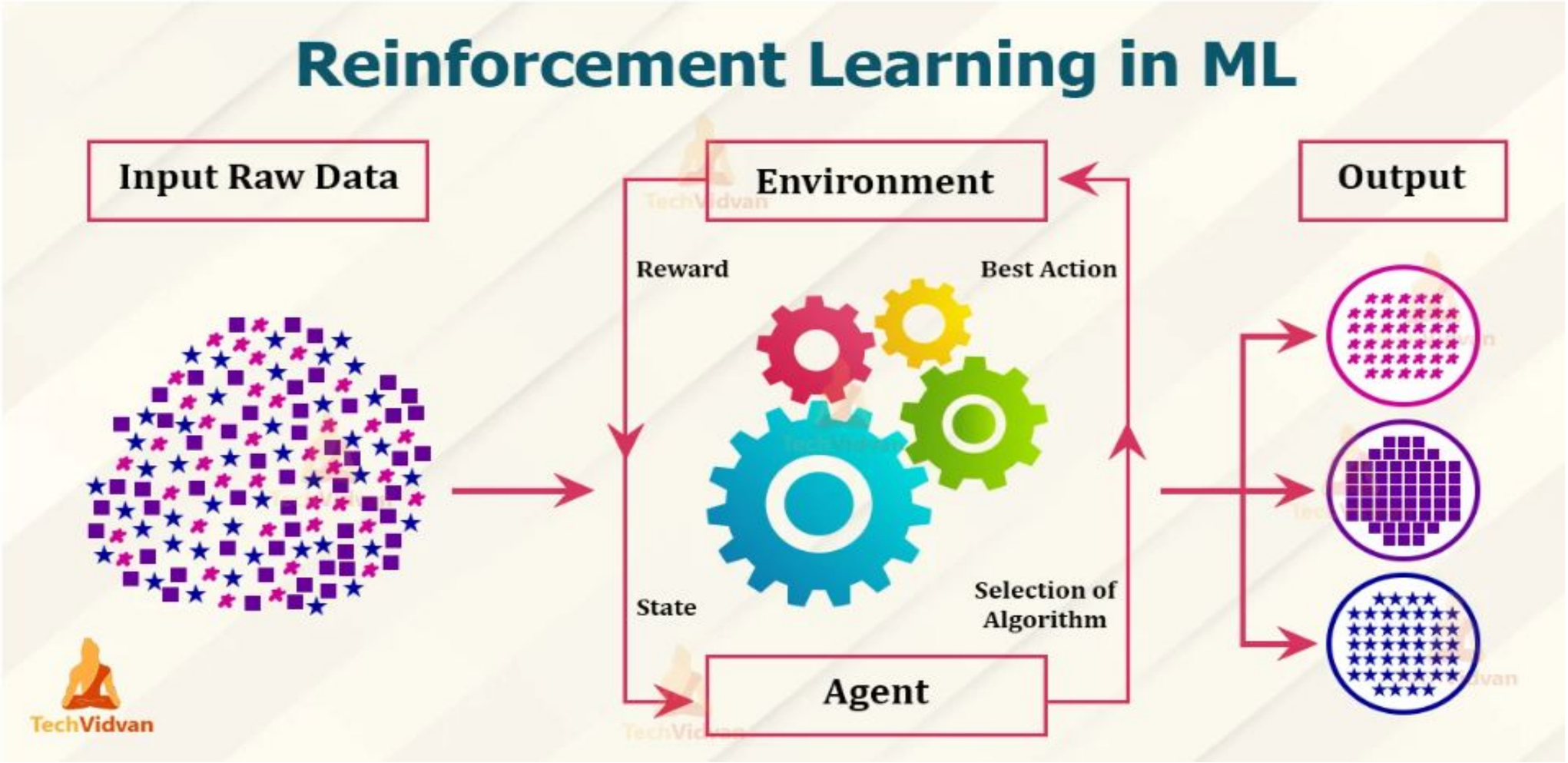
- **Semi-Supervised Learning combines supervised and unsupervised learning, where a small portion of the data is labeled while the majority remains unlabeled.**

- Example: Disease Diagnosis in Medical Imaging
  - Limited labeled X-ray images for training.
  - The model learns from both labeled and unlabeled images.
  - Reduces the cost of manual labeling while improving performance.

- Semi-supervised learning is particularly useful when there is a large amount of unlabeled data available, but it’s too expensive or difficult to label all of it.

Intuitively, one may imagine the three types of learning algorithms as Supervised learning where a student is under the supervision of a teacher at both home and school, Unsupervised learning where a student has to figure out a concept himself and Semi-Supervised learning where a teacher teaches a few concepts in class and gives questions as homework which are based on similar concepts.

# Reinforcement Learning



- **Reinforcement Learning (RL) is a type of learning where an agent interacts with an environment and learns by receiving rewards or penalties based on its actions.**
- **Reinforcement Learning (RL) is a branch of machine learning focused on making decisions to maximize cumulative rewards in a given situation.**
  - RL involves learning through experience. In RL, an agent learns to achieve a goal in an uncertain, potentially complex environment by performing actions and receiving feedback through rewards or penalties.

- **Key Concepts for RL**
  **Agent:** The learner or decision-maker.

  **Environment:** Everything the agent interacts with.

  **State:** A specific situation in which the agent finds itself.

  **Action:** All possible moves the agent can make.

  **Reward:** Feedback from the environment based on the action taken.

- Reinforcement learning is all about making decisions sequentially. In simple words, we can say that the output depends on the state of the current input and the next input depends on the output of the previous input

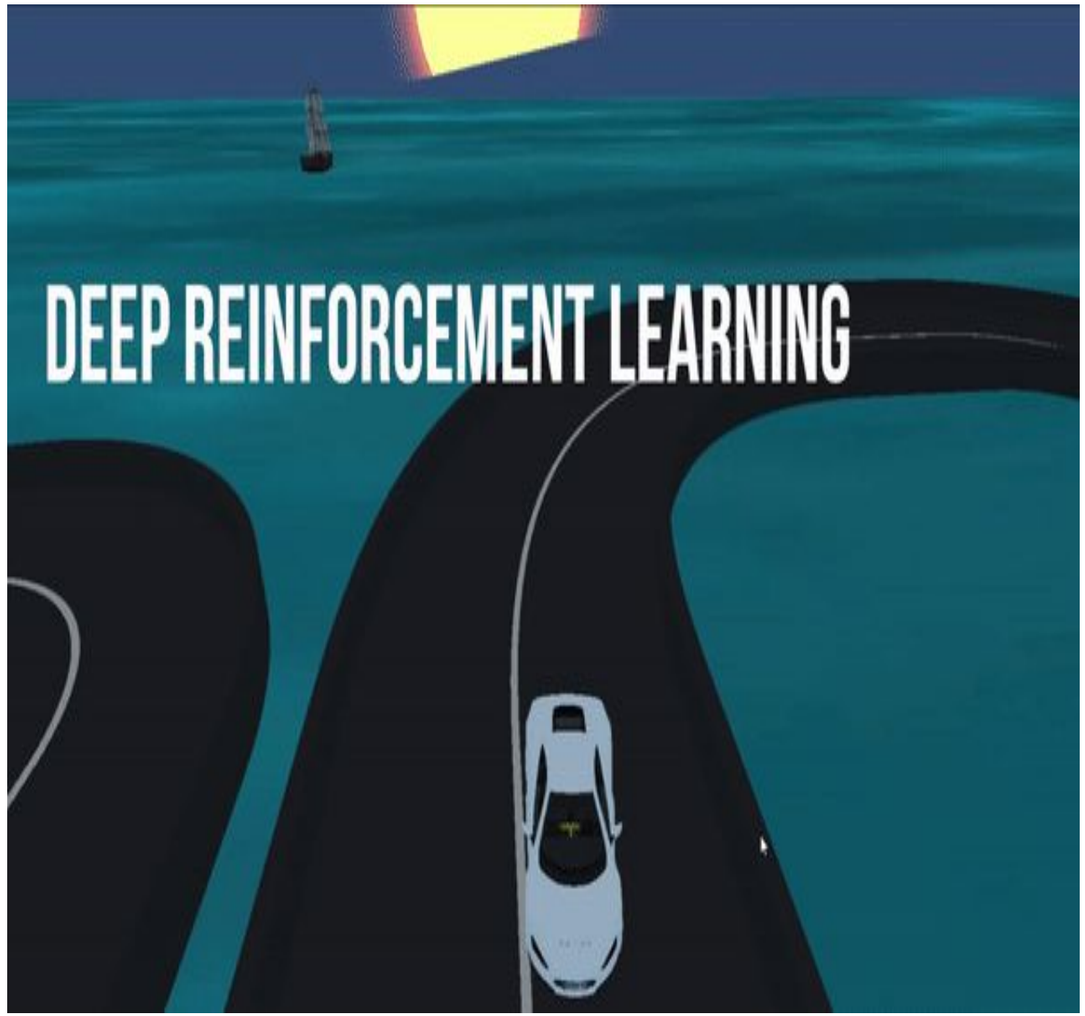
- Key Features:
  - No predefined labels; learning through experience
  - Uses reward functions to guide learning
  - Examples: Robotics, self-driving cars, game AI

# Reinforcement Learning - Example: Self-Driving Cars

**Example: Training a self-driving car.**

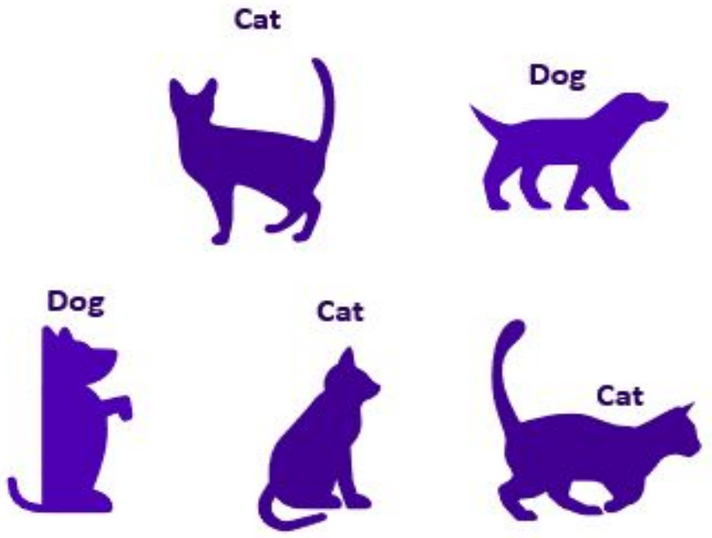
- 1. **Agen**t: The self-driving car.
- 2. **Environment**: The road, traffic lights, pedestrians.
- 3. **Actions**: Accelerate, brake, turn, change lanes.
- 4. **Reward System**:
- - Positive reward for safe driving.
- - Negative reward for collisions.

- The car learns optimal driving strategies through trial and error.



# Self-Supervised Learning

Supervised Learning




*Davide Coccomini | 2021*

- **Self-Supervised Learning is an emerging approach where the model generates its own labels from the data instead of relying on external labeling.**
- Self-supervised learning is a deep learning methodology where a **model is pre-trained using unlabelled data** and the data labels are generated automatically, which are further used in subsequent iterations as ground truths.

- Example: Natural Language Processing (NLP)
  - GPT and BERT models learn by predicting missing words in sentences.
  - No manual labeling required; model learns language representations.
  - Used in text generation, sentiment analysis, and chatbots.

  A self-supervised learning model might be trained to predict the location of an object in an image given the surrounding pixels to classify a video as depicting a particular action.

# Comparison of Machine Learning Paradigms

| Learning Type | Description | Example Applications |
|---|---|---|
| Supervised Learning | Learns from labeled data (input-output pairs). The model generalizes from known patterns. | Image classification, speech recognition. |
| Unsupervised Learning | Learns from unlabeled data by discovering hidden structures or patterns. No explicit supervision. | Market segmentation, anomaly detection. |
| Semi-Supervised Learning | Uses a small amount of labeled data combined with a large amount of unlabeled data to improve learning efficiency. | Fraud detection, medical diagnosis. |
| Reinforcement Learning | Learns via trial and error by interacting with an environment, receiving rewards or penalties. | Robotics, game AI, autonomous driving. |
| Self-Supervised Learning | A form of unsupervised learning where the model generates pseudo-labels from raw data to create its own supervision signals. | NLP (e.g., BERT, GPT), computer vision (e.g., contrastive learning), AI models. |

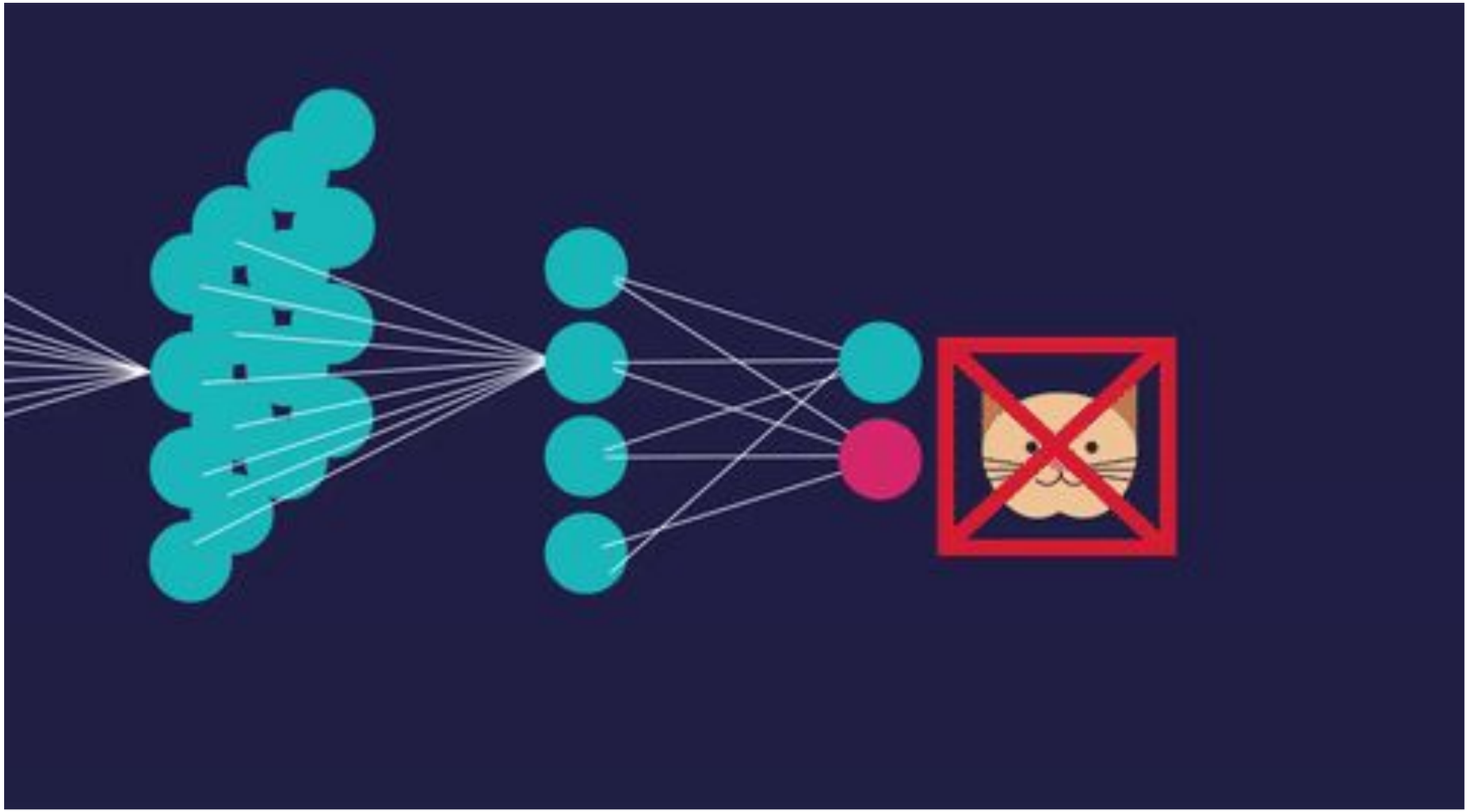
# Summary & Applications

- Comparison of Learning Types:
  - **Supervised Learning**: Used in image classification, speech recognition.
  - **Unsupervised Learning**: Used in market segmentation, anomaly detection.
  - **Semi-Supervised Learning**: Used in fraud detection, medical diagnosis.
  - **Reinforcement Learning**: Used in robotics, game AI, autonomous driving.
  - **Self-Supervised Learning**: Used in NLP, computer vision, AI models.

# Conti…

- Like supervised learning, but:
  - Weights adjusting is not directly related to the error value.
  - The error value is used to randomly, shuffle weights!
  - Relatively slow learning due to ‘randomness’.

# Role of the Hidden Layer in the Neural Network



- The role of a hidden layer is to learn complex patterns in the data and to transform the input data into a new representation that is easier for the network to learn from.
    - This is done by applying a series of mathematical functions to the input data. These functions are called activation functions and they introduce non-linearity into the network. Non-linearity is necessary for the network to learn complex patterns in the data.
- Hidden layers are essential for neural networks to be able to learn complex tasks, such as image recognition and natural language processing.
    - Without hidden layers, neural networks would be limited to learning only linear relationships between the input and output data.

**What is the role of neural networks in deep learning and AI?**

- Deep learning is a type of machine learning that uses artificial neural networks with multiple hidden layers to learn complex patterns in the data.
- AI is a field of computer science that deals with the creation of intelligent agents, which are systems that can reason, learn, and act autonomously.

**Why do neural networks need more than one hidden layer?**

- In short, neural networks don’t require more than one hidden layer. Computer scientists and artificial intelligence engineers have been using multilayer neural networks since the invention of the multilayer perceptron in 1958.
- Adding additional layers allows the neural networks to make more complex calculations or interact with the data more sophisticatedly.
- Stacking large numbers of hidden layers between the input and the output allows your neural network to perform deep-learning tasks regarding the input.
- The number of layers in each neural network depends on the goals you wish to accomplish with the algorithm, but deep learning isn’t possible with a single hidden layer between the input and the output layers.

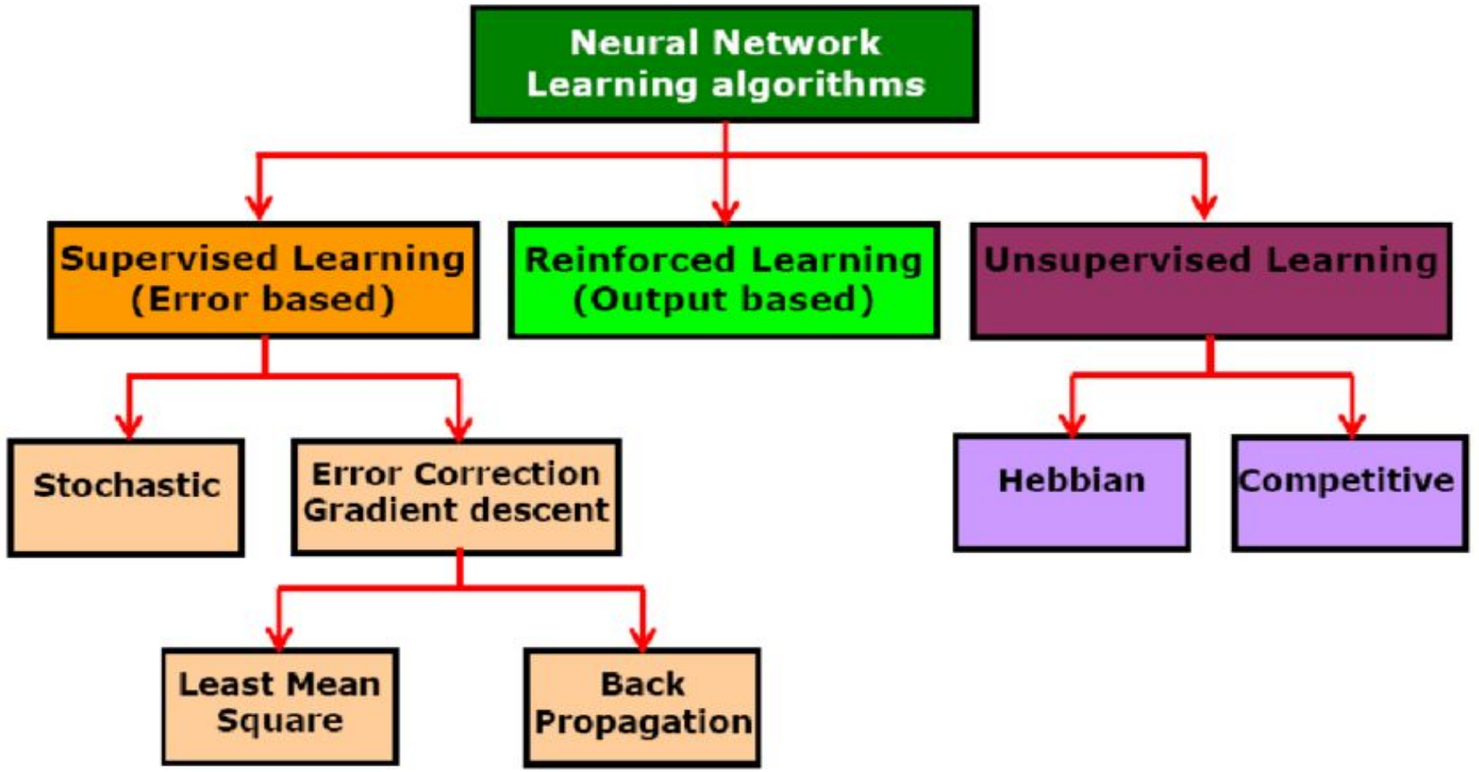
# Artificial Neural Networks Algorithms

Artificial neural networks are inspired by the structure and function of the human brain. They consist of interconnected nodes, called neurons, which process and transmit information.

But how do these networks learn?

# Learning Algorithms

- Learning Rule-



# Error Correction Learning

## The Fundamentals

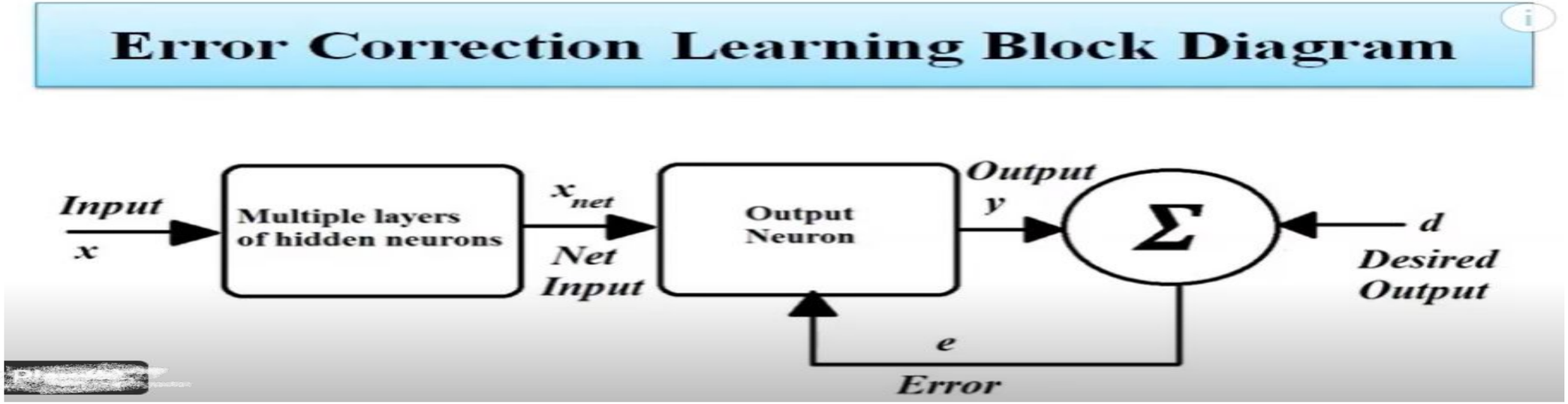
- Error correction learning is a supervised learning technique. It adjusts the weights of connections in a neural network based on the difference between the network's output and the desired output. By minimizing this error, the network learns to produce accurate predictions.

## Real-World Applications

- Error correction learning is widely used in various applications such as image recognition, natural language processing, and speech synthesis. By iteratively adjusting weights based on errors, the network improves its ability to correctly classify or predict outcomes.

# Conti..

- Error Correction Learning, in the context of machine learning, refers to a training method where a model learns by comparing its predicted outputs to the correct target values, then adjusting its internal parameters to minimize the difference between them, essentially "correcting" its errors with each iteration; a common example is using the backpropagation algorithm in neural networks to update weights based on the error signal.



# Error Signal

Depending on the output and desired or target input the error signal is given by,

$$e_k(n) = d_k(n) - y_k(n)$$

## ERROR CORRECTION LEARNING

- error signal = desired response – output signal

  $e_k(n) = d_k(n) - y_k(n)$

  $e_k(n)$ actuates a control mechanism to make the output signal $y_k(n)$ come closer to the desired response $d_k(n)$ in step by step manner

- Also called deltra rule, Widrow-Hoff rule

# ERROR CORRECTION LEARING

The error correction learning procedure is simple enough in conception. The procedure is as follows:

During training an input is put into the network and flows through the network generating a set of values on the output units. Then, the actual output is compared with the desired target, and a match is computed. If the output and target match, no change is made to the net. However, if the output differs from the target a change must be made to some of the connections.

# Example

## REAL LIFE APPLICATIONS

- Automated medical diagnosis :

play an important role in medical decision-making, helping physicians to provide a fast and accurate diagnosis. In many research papers researchers discussed the diagnosis by using back propagation algorithm which is a use of error correction learning.

- With the development of back propagation algorithm it has now become possible to successfully attack problems requiring neural networks with high degrees of nonlinearity and high precision.

# Memory-Based Learning

### Storing Past Experiences

This algorithm relies on storing past experiences as data points. When presented with a new input, it searches for similar examples in its memory and uses them to make predictions or classifications.

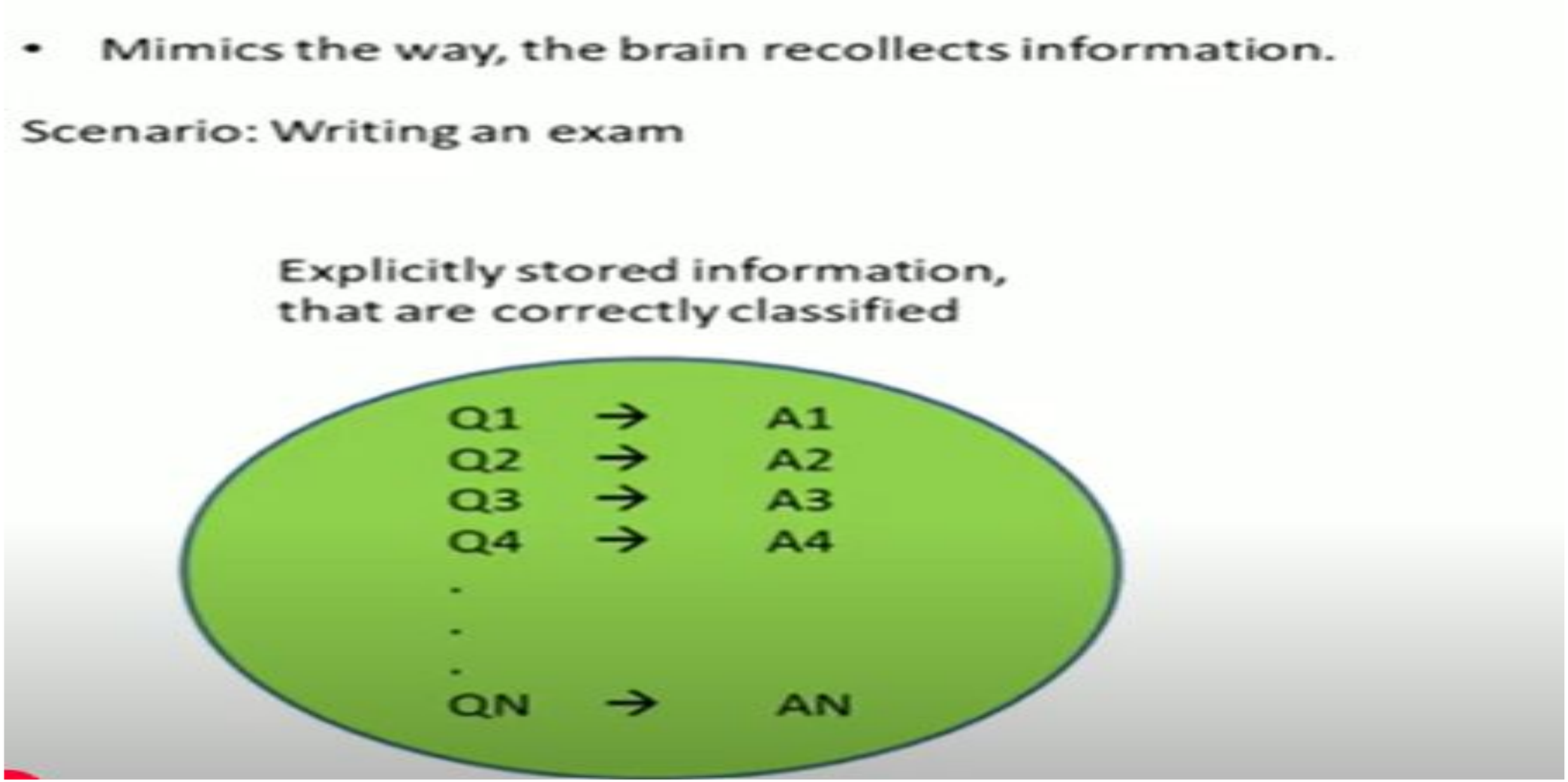
### Applications in Pattern Recognition

Memory-based learning is effective in pattern recognition tasks. Examples include classifying handwritten digits, recognizing faces, and predicting stock market trends. By analyzing past data, the network learns to identify patterns and make informed decisions.

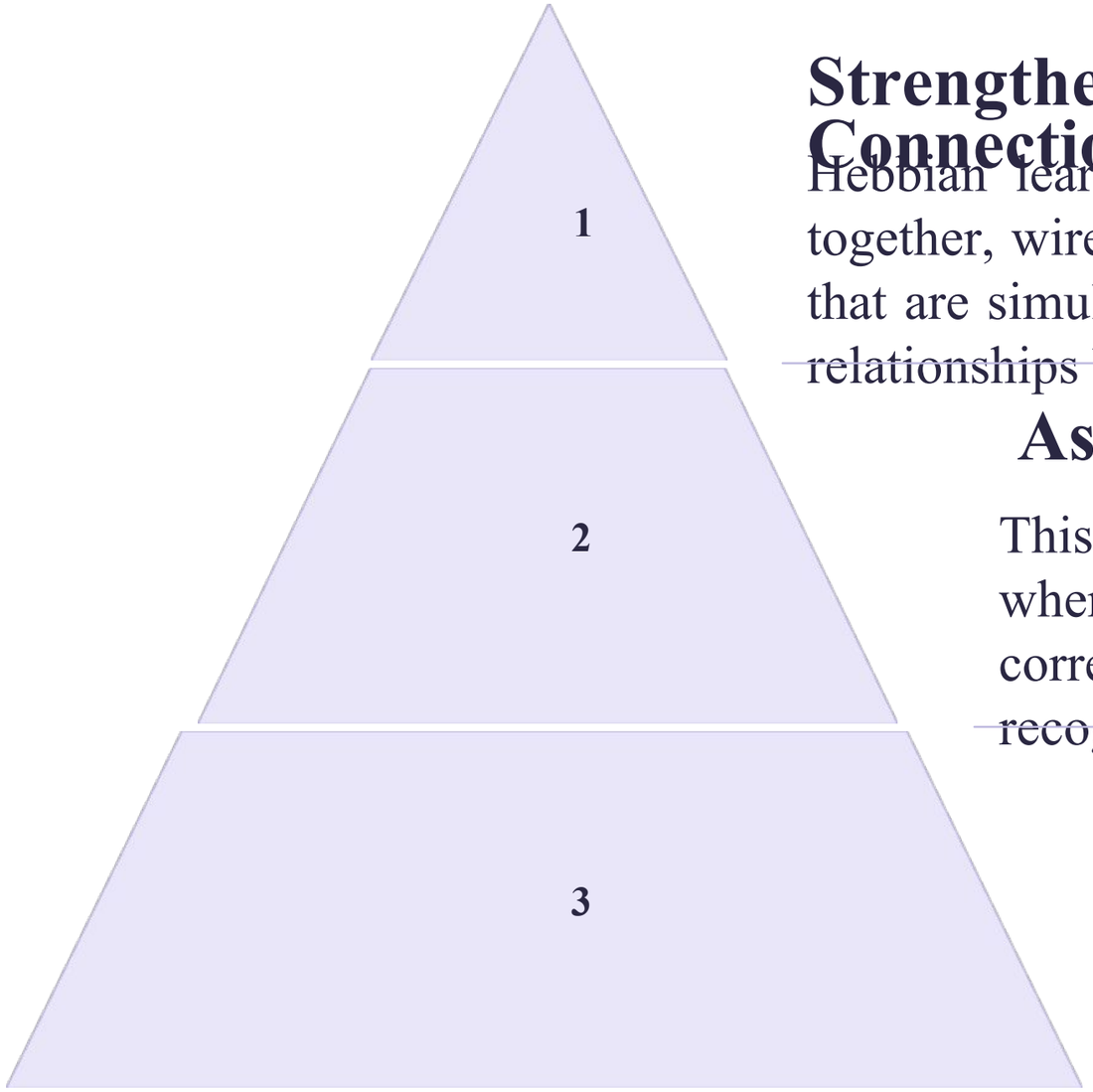
# Example

- Mimics the way, the brain recollects information.

Scenario: Writing an exam



# Hebbian Learning




## Strengthening Connections

Hebbian learning is based on the principle of "neurons that fire together, wire together." It strengthens connections between neurons that are simultaneously active, effectively encoding correlations and relationships between data.

## Associative Learning

This algorithm is particularly useful for associative learning, where the network learns to associate different inputs with corresponding outputs. It's often used in tasks like pattern recognition and language modeling.

## Building Memory

By strengthening connections based on correlated activity, Hebbian learning allows the network to build internal representations of data, effectively forming memories and associations that can be used for future prediction.

# Example-

## Application of Hebbian Learning

- This simple Hebbian learning example can be applied to various fields, such as:

**Pattern Recognition**: The network can learn to recognize and recall simple binary patterns.

**Associative Memory**: Hebbian learning can be used to implement associative memory, where the network can recall outputs associated with given inputs.

**Neural Computation**: This example demonstrates fundamental neural computation principles inspired by biological brains.

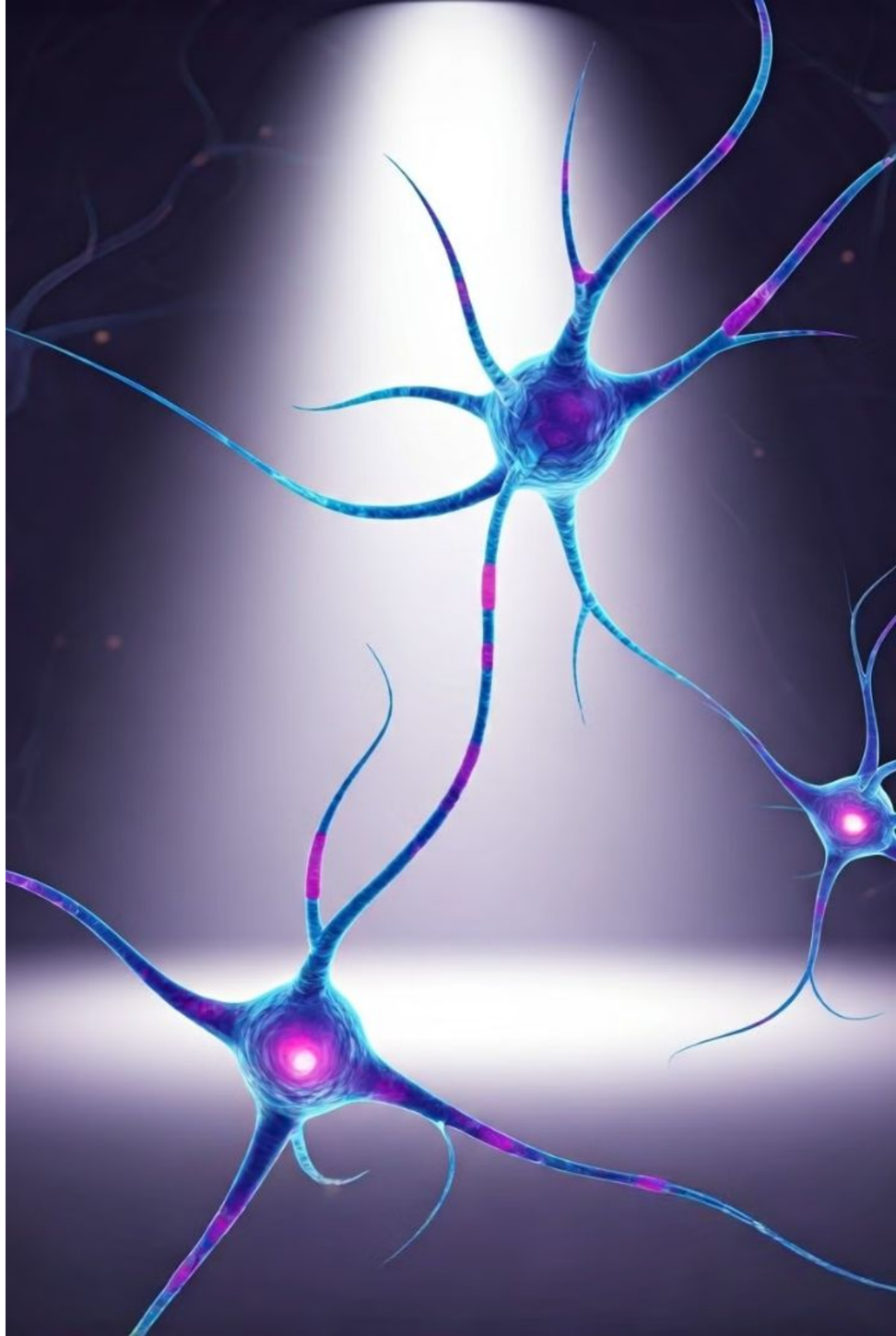
# Competitive Learning

### Competition for Activation

Competitive learning involves multiple neurons competing for activation. The neuron with the highest activation level wins and updates its weights, while others remain unchanged. This process leads to specialization of neurons, with each neuron becoming responsible for a specific

### Clustering and Classification

Competitive learning is often used for clustering tasks, where the network learns to group similar data points together. It can also be applied to classification problems, where the network learns to categorize inputs into distinct classes.



# Example

**Problem:**

Imagine an online store wants to categorize its customers without any pre-defined labels, just based on their purchase history.

**Approach:**

- **Set up:** Create a neural network with several neurons, each representing a potential customer segment (e.g., "high spender", "discount seeker", "frequent buyer").
- **Input:** For each customer, feed their purchase data (total spent, items bought, frequency) into the network.
- **Competition:** Each neuron calculates how well it matches the input customer data.
- **Winner update:** The neuron with the highest match (the "winner") updates its weights to become more similar to the current customer's data.
- **Result:** After processing all customers, each neuron will represent a distinct customer segment based on their buying patterns.

# Boltzmann Learning

1

### Energy Landscape

Boltzmann learning uses an energy function to represent the state of the network. The goal is to minimize this energy function, which corresponds to finding the optimal configuration of connections and neuron activations.

2

### Probabilistic Learning

Boltzmann learning is a probabilistic approach. It learns to generate samples that resemble the training data distribution. This approach is particularly useful for learning complex patterns and relationships in data.

3

### Applications in Machine Learning

Boltzmann learning has applications in areas like image recognition, natural language processing, and robotics. Its ability to learn probabilistic models and find optimal configurations makes it a powerful tool for complex tasks.

# Comparing Learning Algorithms

## 1

### Supervised vs. Unsupervised

Error correction and memory-based learning are supervised, while Hebbian, competitive, and Boltzmann learning are unsupervised or semi-supervised. Supervised learning uses labeled data, while unsupervised learning explores patterns without explicit labels.

## 2

### Network Architecture

Different algorithms may be better suited for different network architectures. For example, Boltzmann learning is often used with restricted Boltzmann machines, while error correction learning is commonly used with feedforward networks.

## 3

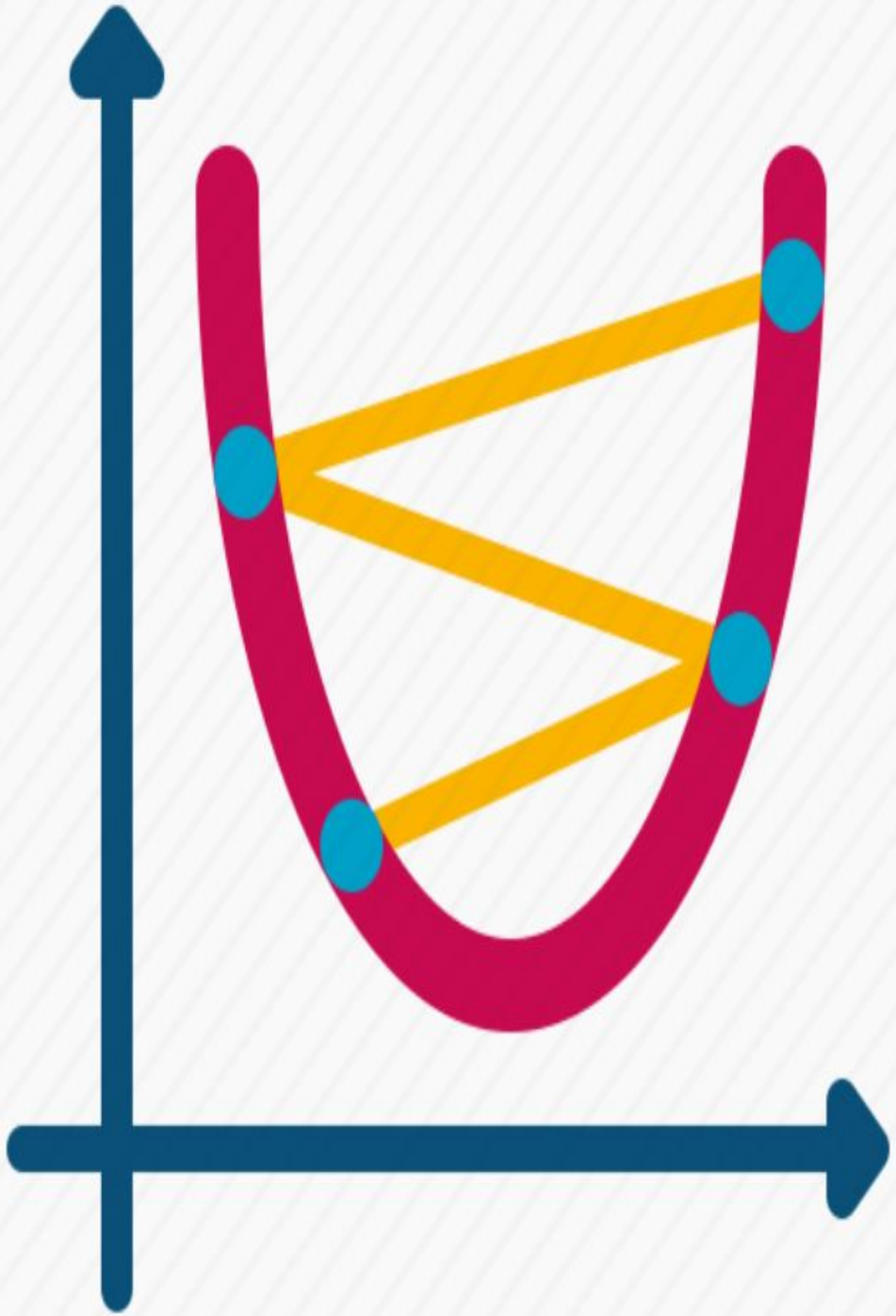
### Computational Complexity

The computational complexity of each algorithm varies. Some algorithms, like Boltzmann learning, can be computationally intensive, while others, like Hebbian learning, are relatively simple and efficient.

## 4

### Application Domain

The choice of learning algorithm depends on the specific application. For instance, competitive learning is often used for clustering and classification, while Hebbian learning is useful for associative learning and memory formation.



# Gradient Descent: A Powerful Optimization Algorithm

# Introduction to Gradient Descent

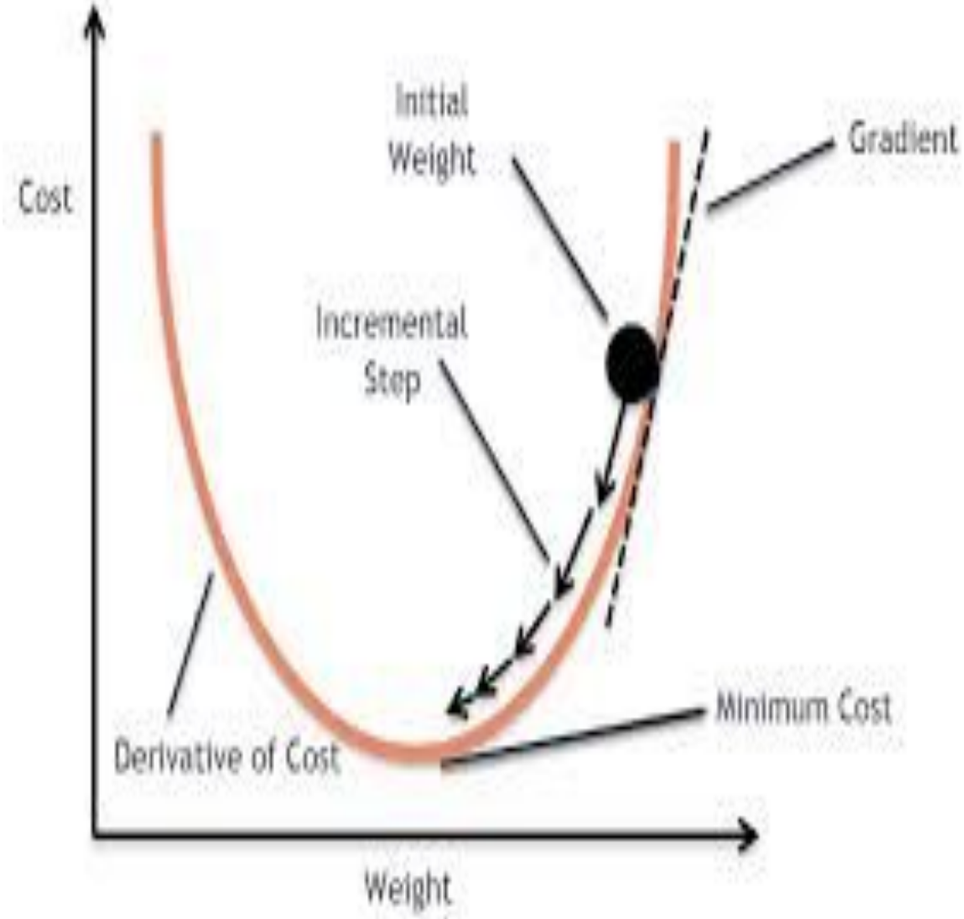
## Core Concept

Gradient descent is an iterative optimization algorithm that finds the minimum value of a function. It starts with an initial guess and repeatedly updates the guess in the direction of the negative gradient until it reaches a local minimum.

## Applications

Widely used in machine learning, data science, statistics, and other fields for tasks like parameter tuning in neural networks, finding optimal solutions in logistics, and minimizing cost functions.

Gradient Descent is an iterative optimization algorithm that finds the minimum value of a function by repeatedly taking steps in the direction of the negative gradient (the steepest downhill slope), effectively "rolling down" a function's landscape until reaching a minimum point; it's commonly used in machine learning to train models by adjusting parameters to minimize a cost function.



# Key Steps in Gradient Descent:

## Initialize Parameters:

Start with random initial values for the parameters (weights and biases) of your model that you want to optimize.

## Calculate the Gradient:

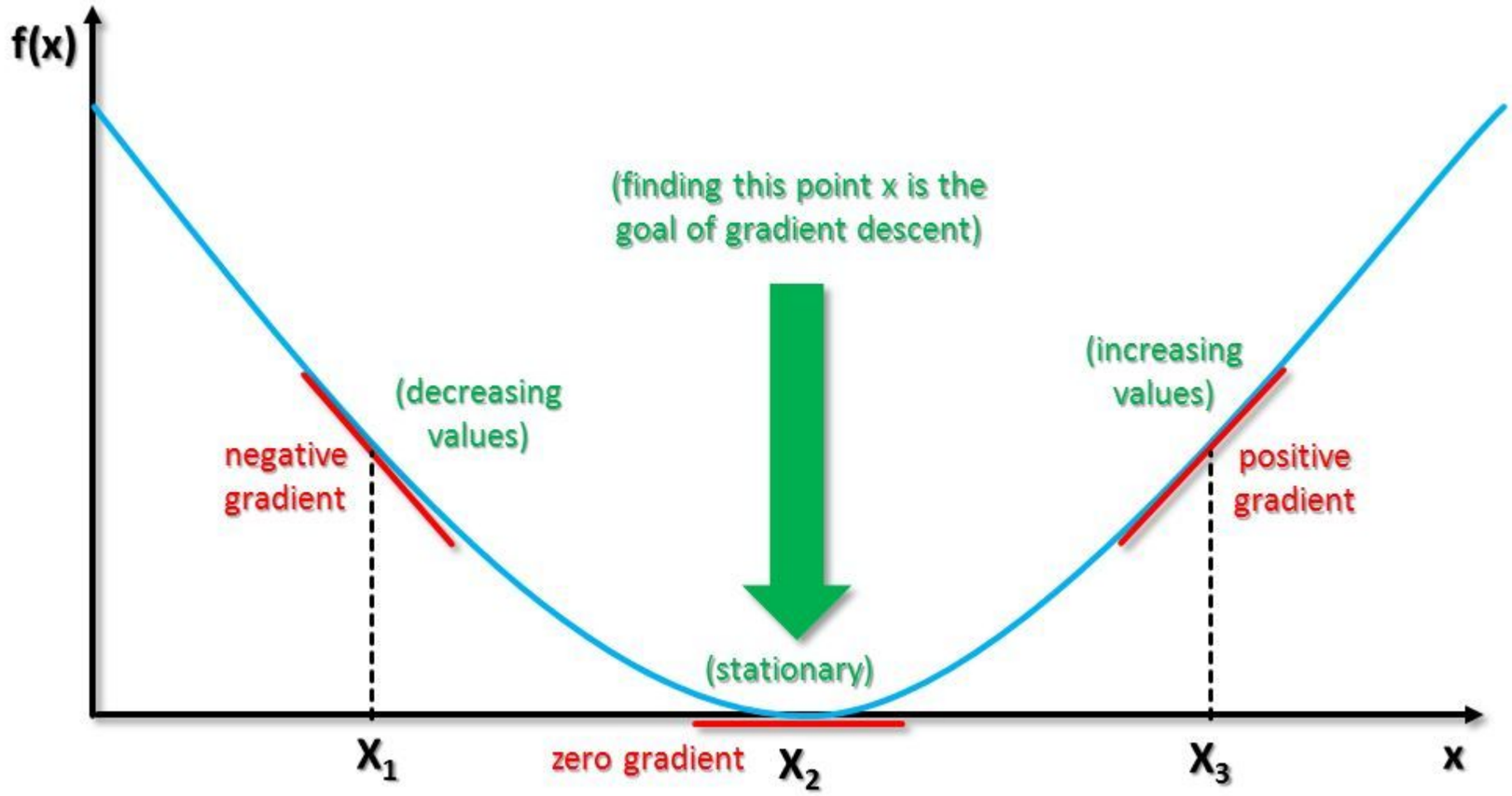
For each iteration, compute the gradient of the cost function with respect to the current parameters. This essentially tells you the direction of the steepest ascent (positive gradient) or descent (negative gradient) at the current point.

## Update Parameters:

Adjust the parameters by taking a small step in the opposite direction of the calculated gradient, using a "learning rate" to control the step size.

## Repeat:

Continue iterating through steps 2 and 3 until the cost function converges to a minimum value or a predefined stopping criterion is met.




**What is the positive and negative gradient of a line?**

**When a line is sloping uphill from left to right the gradient is given a positive value. When a line is sloping downhill from left to right the gradient is given a negative value.**

(minimization: substract gradient term because we move towards local minima)

(the derivative of f with respect to a)

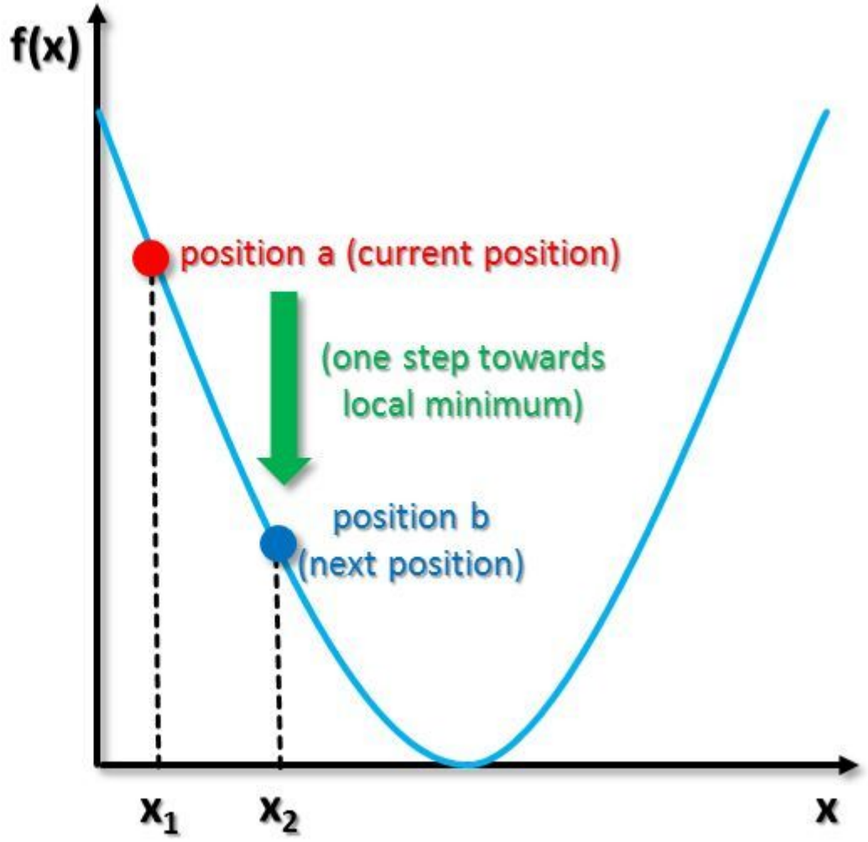
$$b = a - \gamma \nabla f(a)$$

(new position after the step)

(old position before the step)

(weighting factor known as step-size, can change at every iteration, also called learning rate)

(gradient term is steepest ascent)



Gradient descent is an optimization algorithm which is commonly-used to train machine learning models and neural networks. It trains machine learning models by minimizing errors between predicted and actual results.

Training data helps these models learn over time, and the cost function within gradient descent specifically acts as a barometer, gauging its accuracy with each iteration of parameter updates.

Until the function is close to or equal to zero, the model will continue to adjust its parameters to yield the smallest possible error.

Once machine learning models are optimized for accuracy, they can be powerful tools for artificial intelligence (AI) and computer science applications.

# Principles of Gradient Descent

## 1 Gradient

The gradient of a function measures its rate of change at a particular point. It indicates the direction of the steepest ascent.

## 2 Negative Gradient

To minimize a function, gradient descent steps in the direction opposite to the gradient, hence the term "negative gradient."

## 3 Learning Rate

The learning rate determines the size of the step taken in each iteration. It controls how quickly the algorithm converges.

# Working of Gradient Descent Algorithm

1. **Initialization**

   Start with an initial guess for the minimum of the function.

2. **Gradient Calculation**

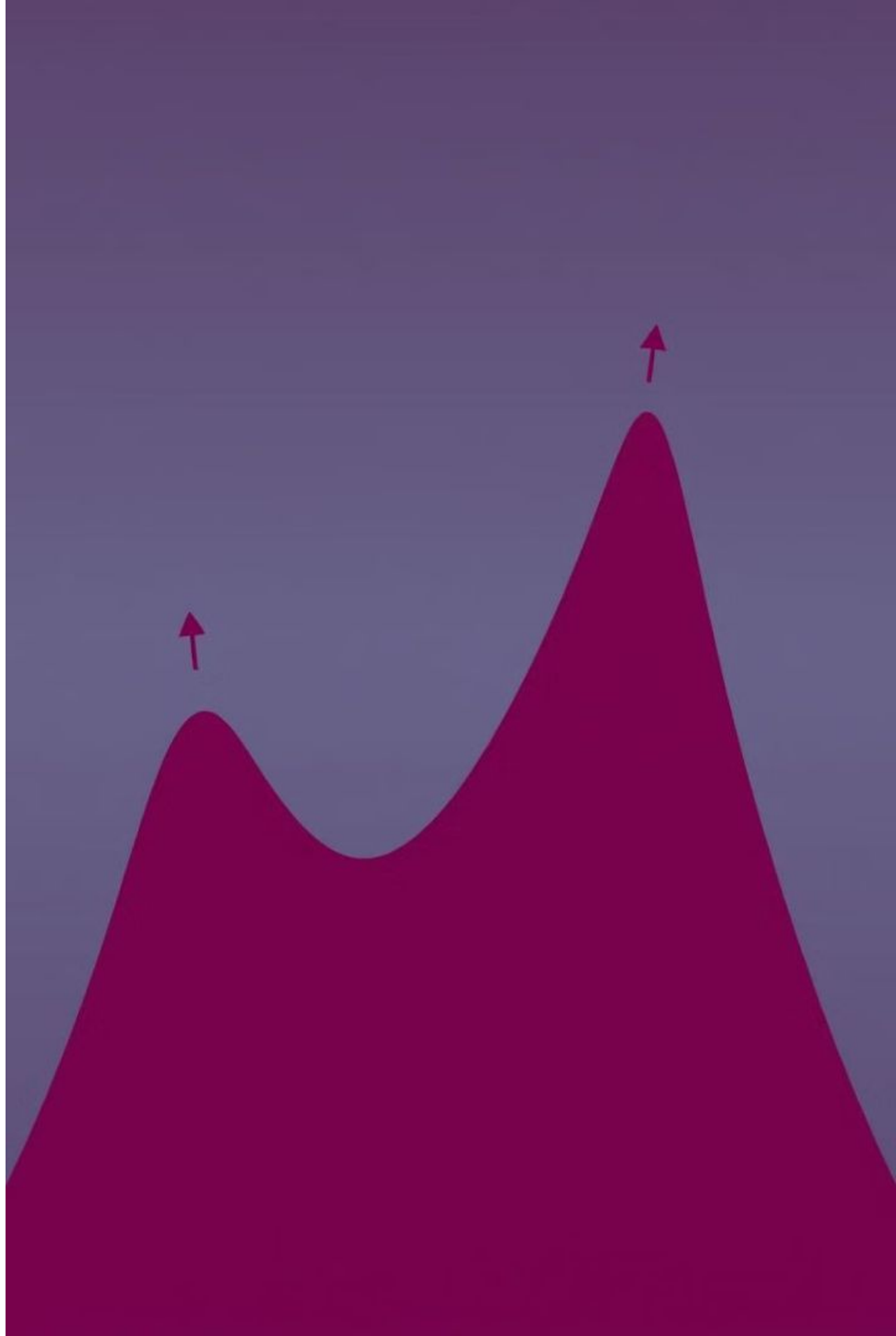
   Calculate the gradient of the function at the current guess.

3. **Update Guess**

   Update the guess by moving in the direction opposite to the gradient, multiplied by the learning rate.

4. **Repeat**

   Repeat steps 2 and 3 until convergence, which means the change in the guess becomes negligible.



# Advantages and Limitations of Gradient Descent

## Advantages

- Can be used for a wide range of optimization problems.
- Relatively simple to implement.
- Can handle high-dimensional data.

## Limitations

- Can get stuck in local minima.
- Sensitive to the learning rate.
- May converge slowly for complex functions.

# Variations of Gradient Descent

## Batch Gradient Descent

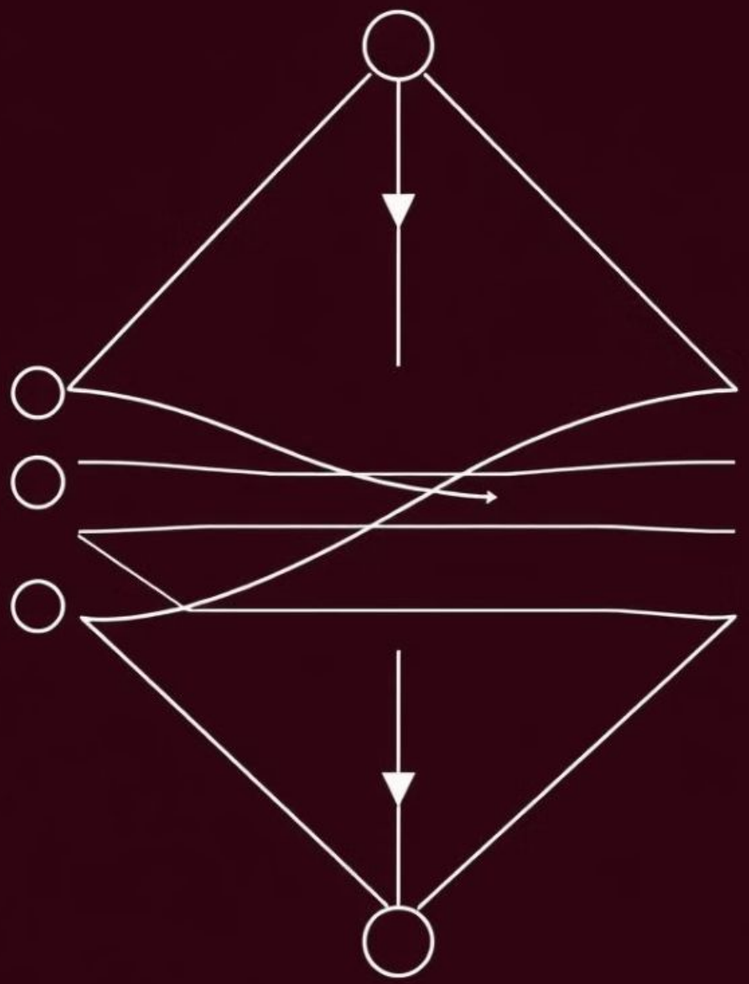
Uses the entire training dataset to compute the gradient in each iteration.

## Stochastic Gradient Descent (SGD)

Uses a single data point at a time to compute the gradient, making it faster but noisier.

## Mini-Batch Gradient Descent

Uses a small batch of data points to compute the gradient, balancing speed and stability.



# Applying Gradient Descent to Machine Learning Problems

1

## Training Neural Networks

Gradient descent is used to adjust the weights and biases of neural networks to minimize the error between predicted and actual outputs.

2

## Linear Regression

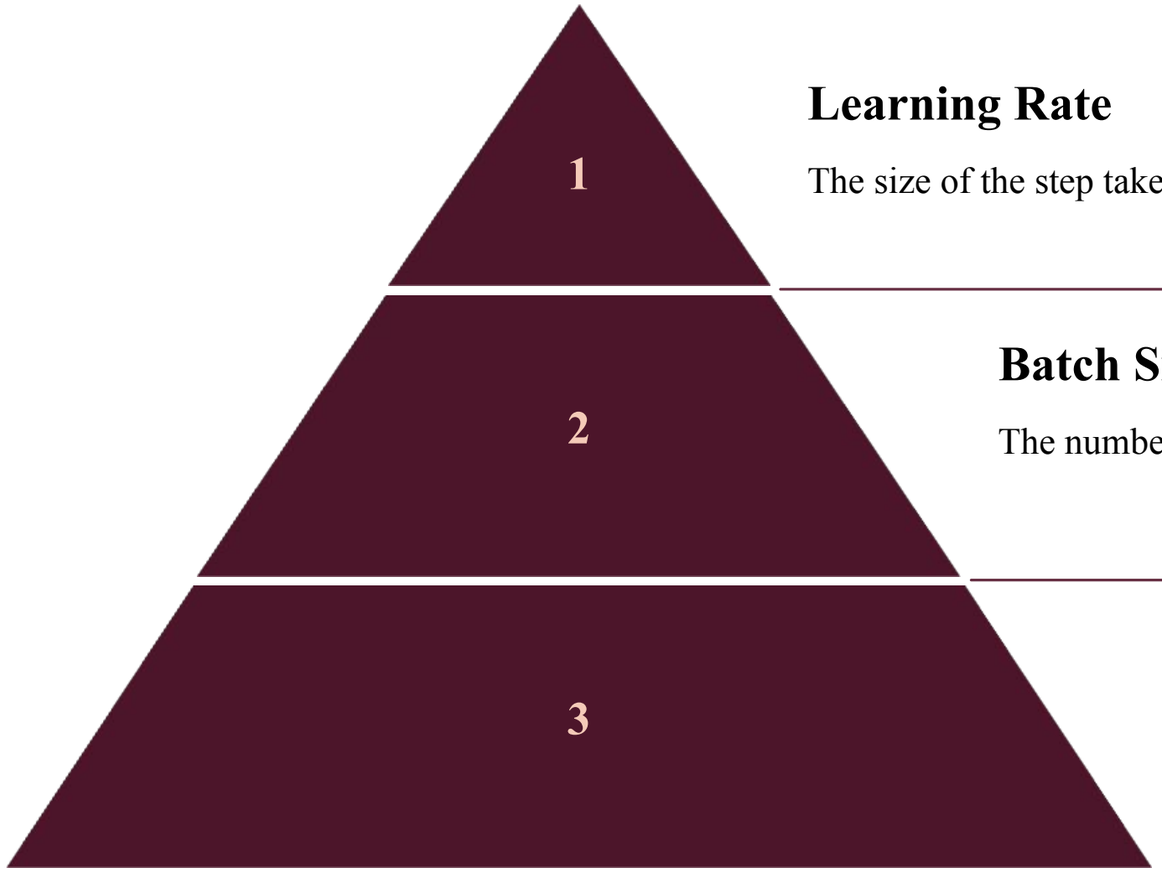
Used to find the best-fit line for a set of data points by minimizing the sum of squared errors.

3

## Logistic Regression

Used to predict categorical outcomes by minimizing the cross-entropy loss function.

# Hyperparameter Tuning in Gradient Descent




## Learning Rate

The size of the step taken in each iteration.

## Batch Size

The number of data points used to compute the gradient in each iteration.

## Momentum

A factor that helps gradient descent avoid getting stuck in local minima.

**Example with Explanation:**

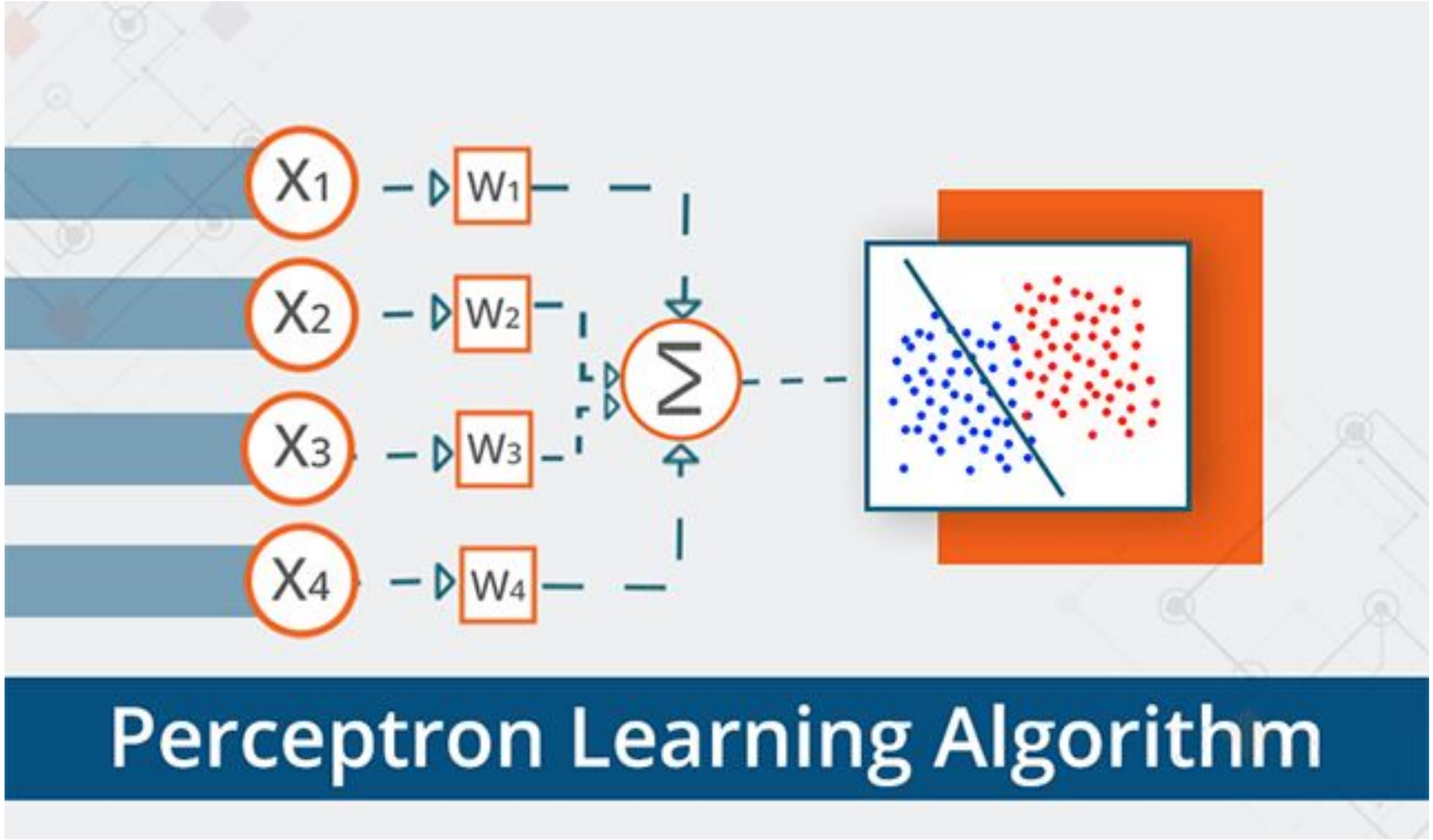
**Problem: Minimize the function f(x) = x^2, where the goal is to find the value of x that gives the lowest value of the function.**

## Steps:

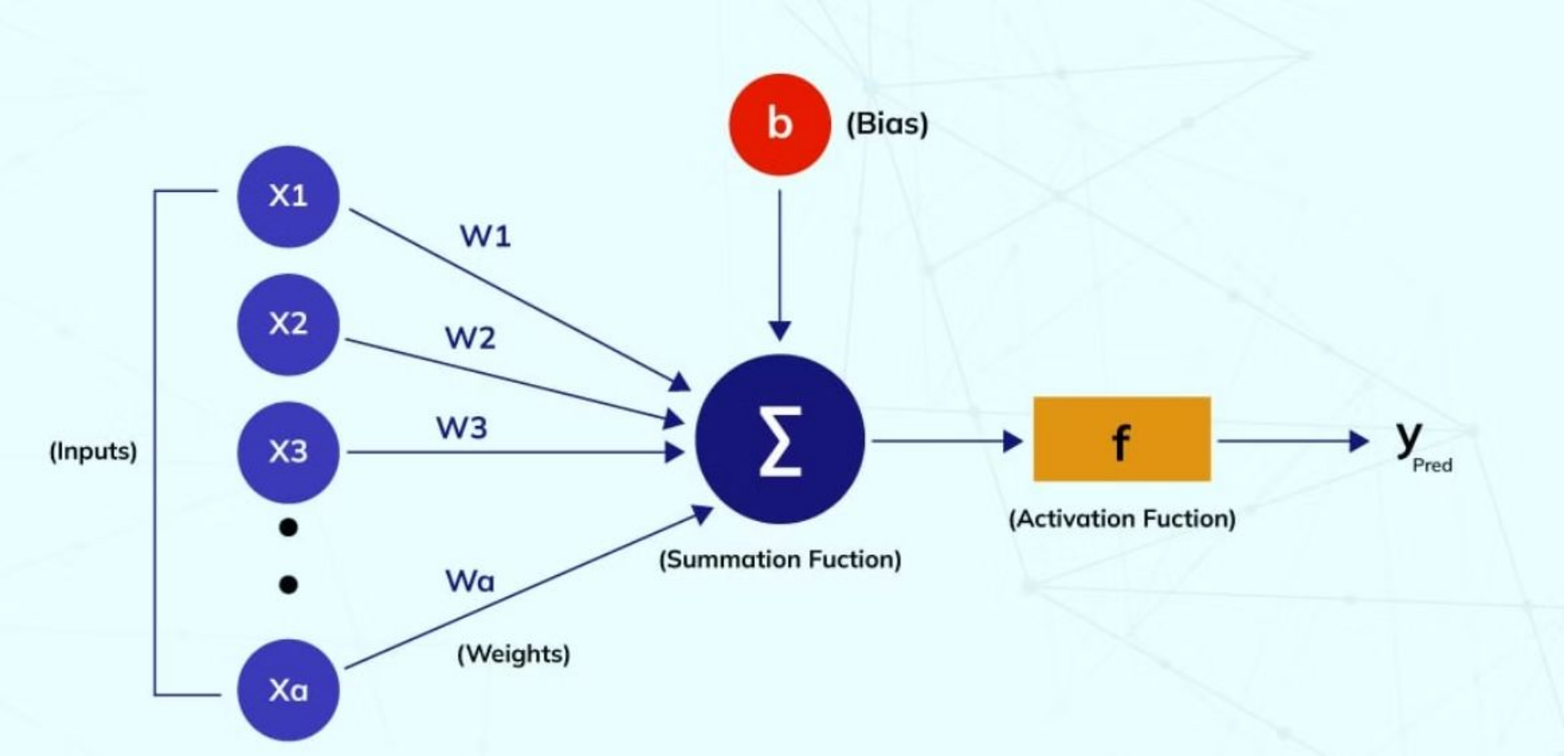
- **Initialize:** Choose an initial value for x, let's say x = 5.
- **Calculate Gradient:** The gradient of f(x) is 2x. So, at x = 5, the gradient is 2 * 5 = 10.
- **Update Parameter:**
  - Choose a learning rate (alpha) - let's say alpha = 0.1.
  - Update x: x_new = x - alpha * gradient = 5 - 0.1 * 10 = 4.
- **Repeat:**
  - Calculate the new gradient at x = 4: 2 * 4 = 8.
  - Update x again: x_new = 4 - 0.1 * 8 = 3.2

Continue this process until the gradient becomes very small, which in this case would mean x is approaching the minimum value of 0.

# Perceptron and Backpropagation



# Perceptron Learning Algorithm




## Single-Layer Perceptron Model

A basic unit that makes a binary classification decision based on weighted inputs.

## Multi-Layer Perceptron (MLP) Model

Networks with multiple layers of interconnected perceptrons, capable of learning complex patterns.

# Can perceptron learning generate backpropagation?

- **Key Reasons Why Perceptron Learning Cannot Generate Backpropagation**

1. **Single-layer vs. Multi-layer**
    1. Perceptron learning applies to a **single-layer perceptron**, meaning it cannot handle multi-layer networks.
    2. Backpropagation is designed for **multi-layer perceptrons (MLPs)** and deep networks.
2. **Activation Function**
    1. The perceptron uses a **step function** , which is **not differentiable**.
    2. Backpropagation requires differentiable activation functions (e.g., **sigmoid, ReLU**) to compute gradients.
3. **Error Propagation**
    1. Perceptron learning updates weights **only at the output layer** based on a simple rule.
    2. Backpropagation distributes the error **backward** through hidden layers using **partial derivatives**.
4. **Learning Method**
    1. The perceptron updates weights using a simple **error-based rule**: $W=W+\eta(Y-Y^)X$
    2. Backpropagation uses **gradient descent** and **chain rule differentiation** to adjust weights across multiple layers.

# Answer

Perceptron learning **cannot** generate backpropagation because:

- It lacks **hidden layers**.
- It does not use **gradient descent**.
- It does not support **non-linear problems**.
- However, **the perceptron was a stepping stone** that led to the development of multi-layer networks and the backpropagation algorithm.

# Components of Perceptron

1

## Input

Represents the data's features, like the color and size of a fruit.

2

## Weights

Each input feature has a weight, determining its importance in making a decision.

3

## Bias

An extra parameter added to adjust the output, ensuring flexibility in decisions.

4

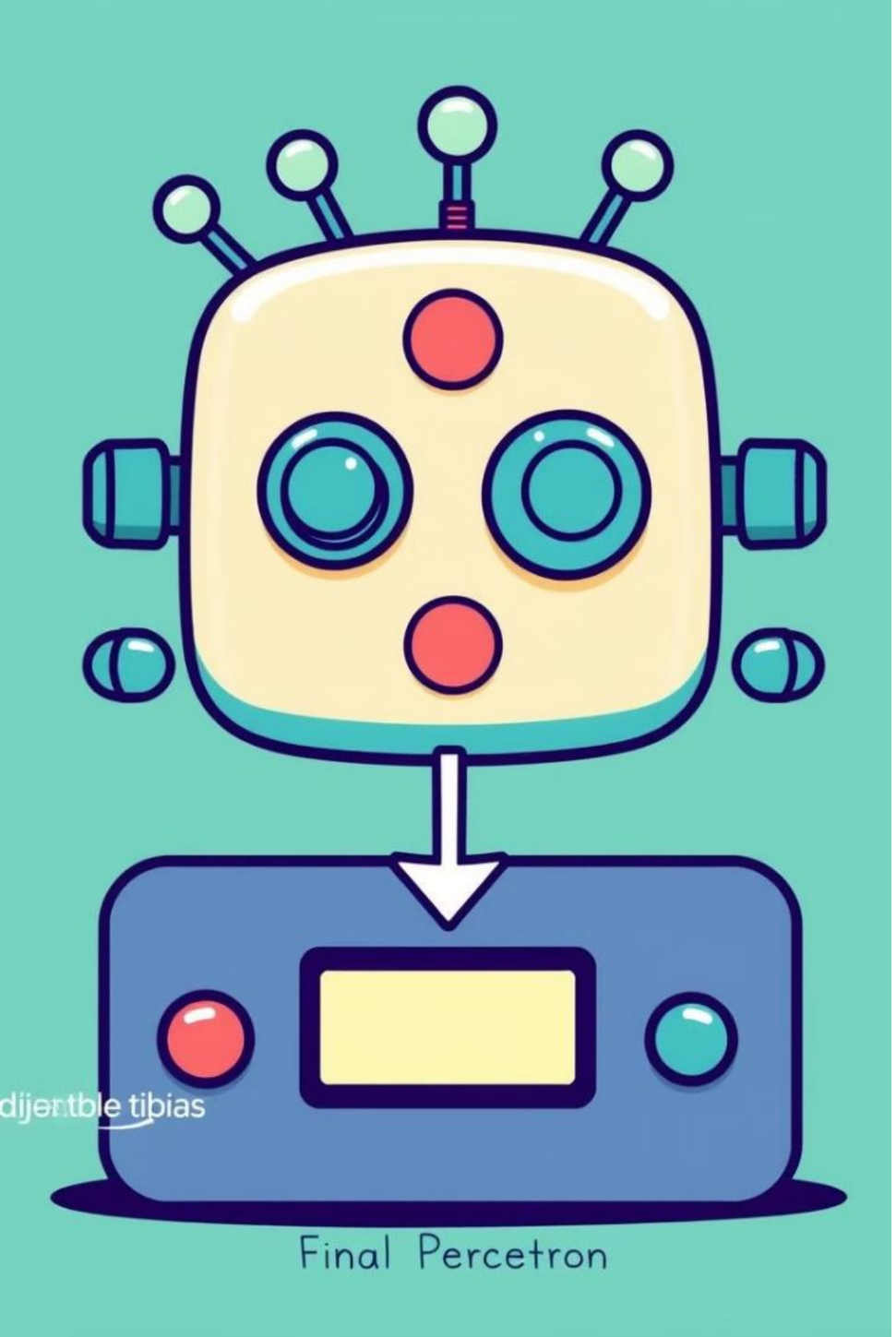
## Activation Function

Determines whether the neuron "fires" based on the weighted sum of inputs.

5

## Output

The final prediction made by the perceptron, such as classifying an input as an apple or an orange.




# Why do we Need Weight and Bias?

Weights and bias are learnable parameters that the network adjusts during training to achieve the desired output.

### Weights

Measure the importance of each feature in predicting output. Features close to zero have less significance.

### Bias

A constant used to adjust the output and help the model provide the best fit for the given data.

# How Perceptron Works

The perceptron algorithm iteratively adjusts its weights and bias to minimize errors and improve accuracy.

1

### Weighted Sum Calculation

Multiply each input value by its corresponding weight.

2

### Add Bias

Add the bias to the weighted sum.

3

### Activation Function Application

Apply an activation function (e.g., step function) to produce output.

# Perceptron Limitations

Perceptrons have limitations when dealing with complex data patterns and non-linear decision boundaries.

## Linear Classification

Limited to problems with linearly separable data. Cannot classify data requiring non-linear decision boundaries.

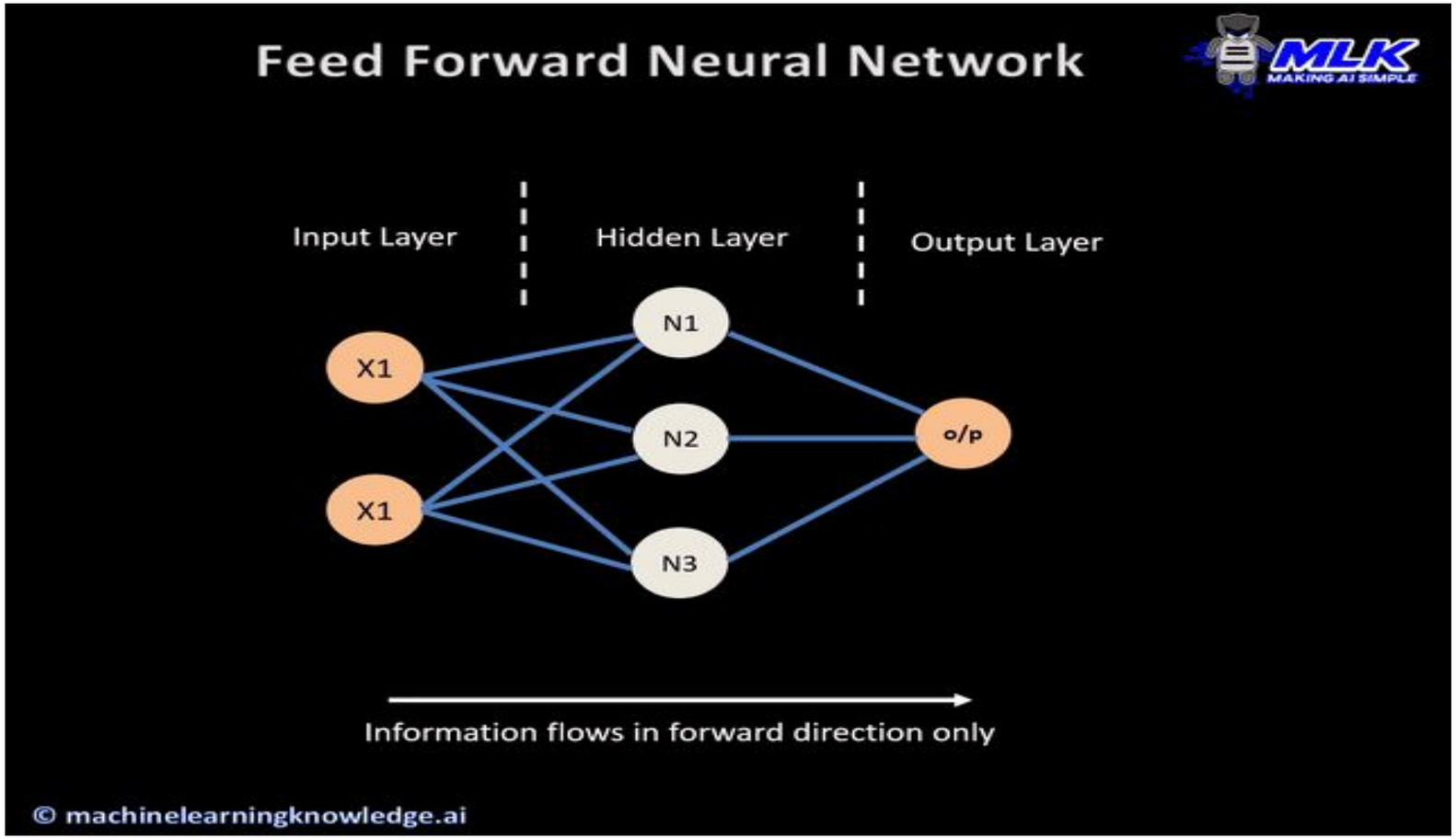
## Complex Patterns

Lack the capacity to process complex relationships in data, needed for tasks like image recognition.

## Single-Layer Limitation

A single perceptron cannot handle tasks involving complex interactions between multiple features.

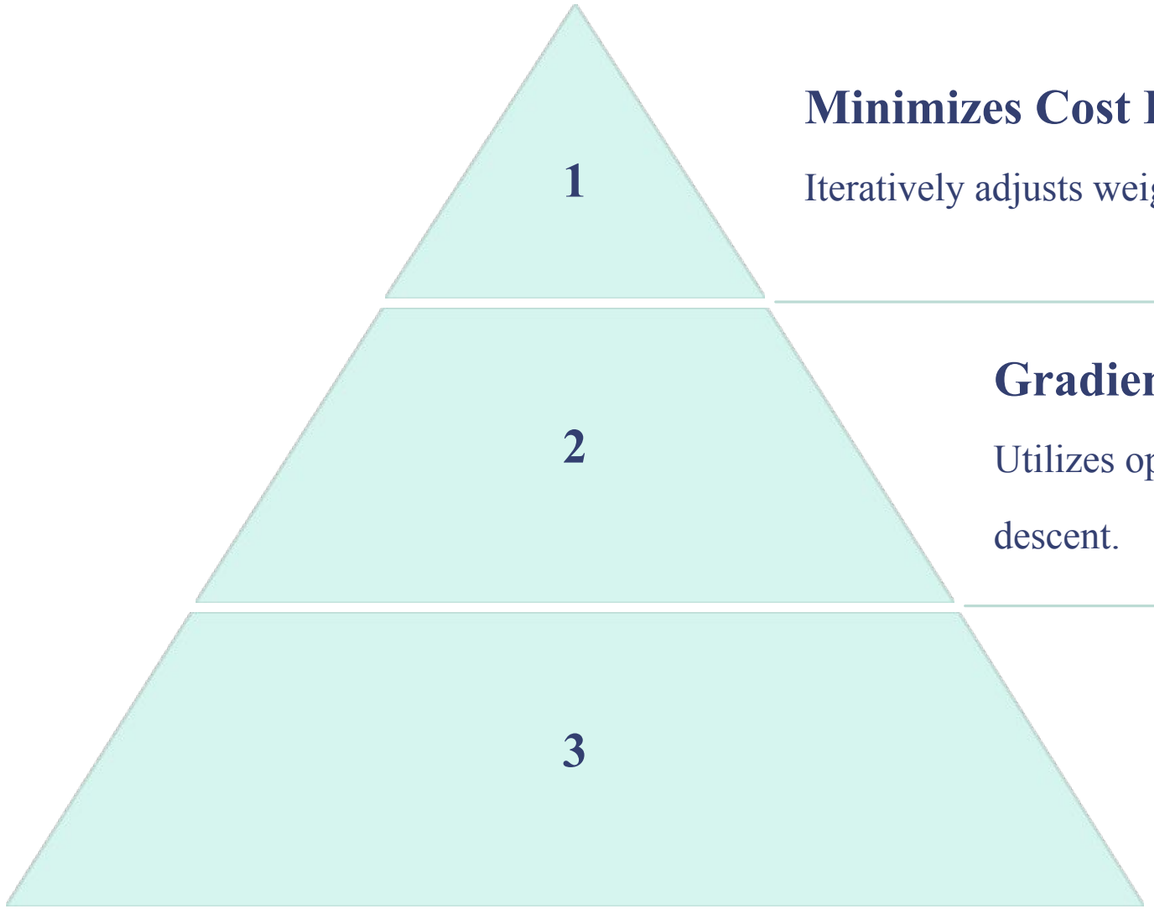
# Beyond Perceptrons: Multi-Layer Perceptrons (MLPs)




For non-linear classification problems, use Multi-Layer Perceptrons (MLPs), which are more complex architectures with multiple layers.

# Backpropagation Algorithm

Backpropagation is a powerful algorithm used to train artificial neural networks, particularly feed-forward networks.




### Minimizes Cost Function

Iteratively adjusts weights and biases by following the error gradient.

### Gradient Descent

Utilizes optimization algorithms like gradient descent or stochastic gradient descent.

### Chain Rule

Computes the gradient using the chain rule from calculus, allowing it to navigate complex layers.

# Importance of Backpropagation

Backpropagation is crucial for efficient weight update, scalability, and automated learning in deep learning.

1

## Efficient Weight Update

Computes the gradient of the loss function with respect to each weight, enabling efficient weight update.

2

## Scalability

Scales well to networks with multiple layers and complex architectures, making deep learning feasible.

3

## Automated Learning

The learning process becomes automated, allowing the model to adjust itself for optimal performance.

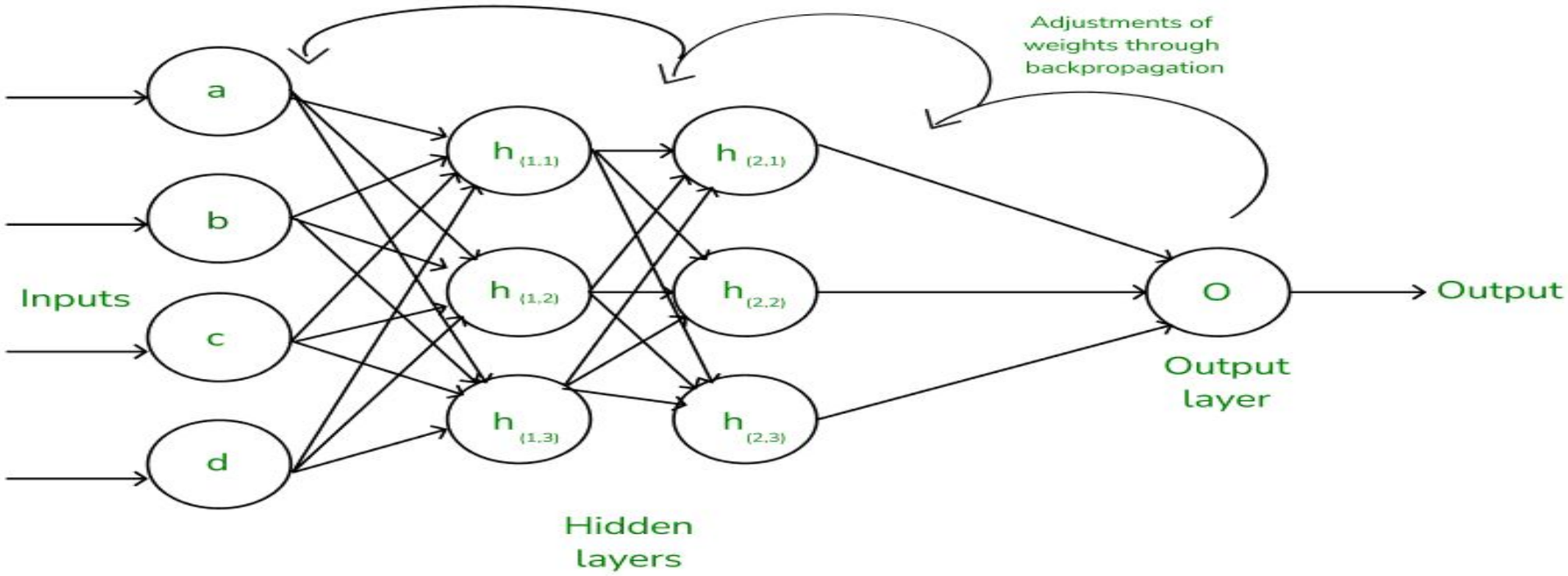
# Why is Backpropagation Important?

- Backpropagation plays a critical role in how neural networks improve over time. Here's why:
- **Efficient Weight Update**: It computes the gradient of the loss function with respect to each weight using the chain rule, making it possible to update weights efficiently.
- **Scalability**: The backpropagation algorithm scales well to networks with multiple layers and complex architectures, making deep learning feasible.
- **Automated Learning**: With backpropagation, the learning process becomes automated, and the model can adjust itself to optimize its performance.

- **What is Backpropagation?**
- **Backpropagation** is a powerful algorithm in deep learning, primarily used to train artificial neural networks, particularly feed-forward networks. It works iteratively, minimizing the cost function by adjusting weights and biases.
- In each epoch, the model adapts these parameters, reducing loss by following the error gradient. Backpropagation often utilizes optimization algorithms like gradient descent or stochastic gradient descent. The algorithm computes the gradient using the chain rule from calculus, allowing it to effectively navigate complex layers in the neural network to minimize the cost function.

# Back propagation Learning Algorithms

***Backpropagation*** **(**short for "***Backward Propagation of Errors***") is a method used to train artificial neural networks. Its goal is to reduce the difference between the model's predicted output and the actual output by adjusting the weights and biases in the network.



# Conti…

- A **neural network** is a structured system composed of computing units called neurons, which enable it to compute functions. These neurons are interconnected through edges and assigned an **activation function**, along with adjustable parameters.
- These parameters allow the neural network to compute specific functions. Regarding activation functions, higher activation values indicate greater neuron activation in the network.

# Working of Algorithms

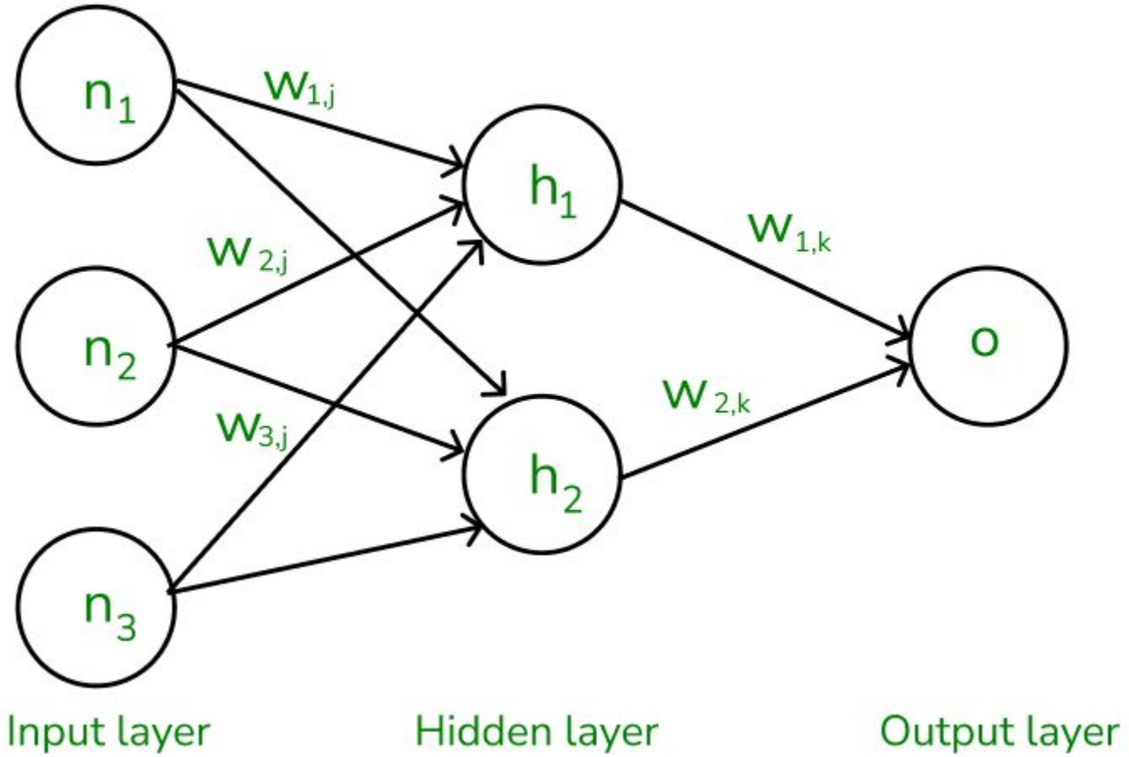
- The Backpropagation algorithm involves two main steps:

1. **Forward Pass**
2. **Backward Pass**.

## How Does the Forward Pass Work?

- In the **forward pass**, the input data is fed into the input layer. These inputs, combined with their respective weights, are passed to hidden layers.
- For example, in a network with two hidden layers (h1 and h2 as shown in Fig. (a)), the output from h1 serves as the input to h2. Before applying an activation function, a bias is added to the weighted inputs.
- Each hidden layer applies an activation function like **ReLU (Rectified Linear Unit)**, which returns the input if it's positive and zero otherwise. This adds non-linearity, allowing the model to learn complex relationships in the data. Finally, the outputs from the last hidden layer are passed to the output layer, where an activation function, such as **softmax**, converts the weighted outputs into probabilities for classification.

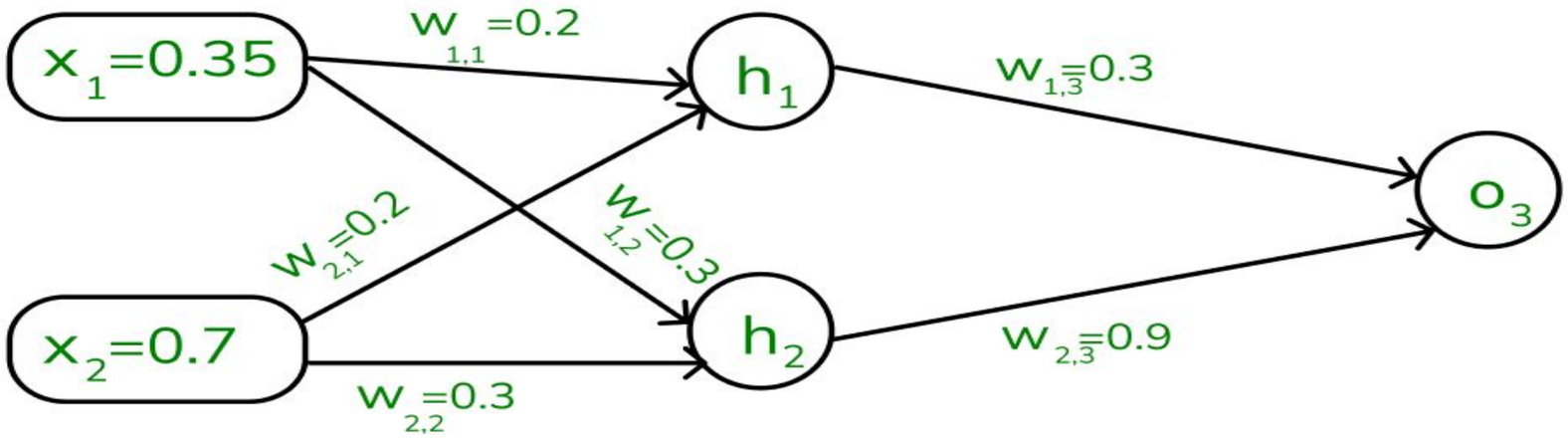
# Conti..


$n_1$
$n_2$
$n_3$
$w_{1,j}$
$w_{2,j}$
$w_{3,j}$
$h_1$
$h_2$
$w_{1,k}$
$w_{2,k}$
o
Input layer
Hidden layer
Output layer

- **How Does the Backward Pass Work?**
- In the backward pass, the error (the difference between the predicted and actual output) is propagated back through the network to adjust the weights and biases.
- **Error=(Predicted Label−True Label)**
- Once the error is calculated, the network adjusts weights using **gradients**, which are computed with the chain rule.
- These gradients indicate how much each weight and bias should be adjusted to minimize the error in the next iteration.
- The backward pass continues layer by layer, ensuring that the network learns and improves its performance. The activation function, through its derivative, plays a crucial role in computing these gradients during backpropagation.

# Example of Backpropagation in Machine Learning

- Let’s walk through an example of backpropagation in machine learning. Assume the neurons use the sigmoid activation function for the forward and backward pass. The target output is 0.5, and the learning rate is 1.
-



Here's how backpropagation is implemented:

## Forward Propagation

### 1. Initial Calculation

The weighted sum at each node is calculated using:

$$a_j = \sum(w_i, j * x_i)$$

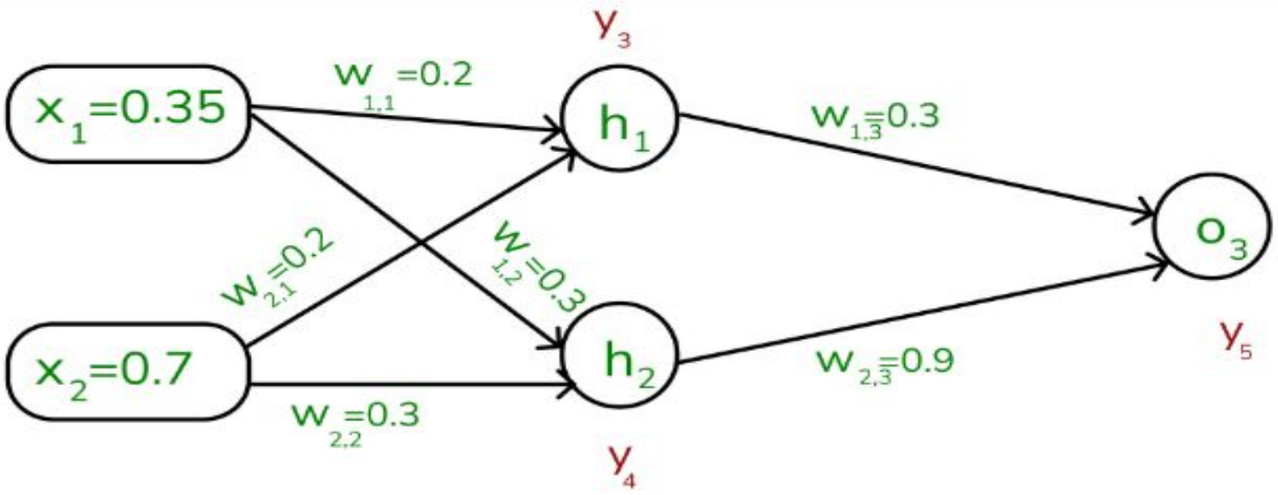
Where,

- $a_j$ is the weighted sum of all the inputs and weights at each node,
- $w_{i,j}$ represents the weights associated with the $j^{th}$ input to the $i^{th}$ neuron,
- $x_i$ represents the value of the $j^{th}$ input,

### 2. Sigmoid Function

The sigmoid function returns a value between 0 and 1, introducing non-linearity into the model.

$$y_j = \frac{1}{1+e^{-a_j}}$$




*To find the outputs of y3, y4 and y5*

**3. Computing Outputs**

At h1 node,

$$\begin{aligned} a_1 &= (w_{1,1}x_1) + (w_{2,1}x_2) \\ &= (0.2 * 0.35) + (0.2 * 0.7) \\ &= 0.21 \end{aligned}$$

Once, we calculated the $a_1$ value, we can now proceed to find the $y_3$ value:

$$y_j = F(a_j) = \frac{1}{1+e^{-a_1}}$$

$$y_3 = F(0.21) = \frac{1}{1+e^{-0.21}}$$

$$y_3 = 0.56$$

Similarly find the values of $y_4$ at **$h_2$** and $y_5$ at $O_3$ .

### 4. Error Calculation

*Note that, our actual output is 0.5 but we obtained 0.67.*

To calculate the error, we can use the below formula:

$$Error_j = y_{target} - y_5$$

$$Error = 0.5 - 0.67 = -0.17$$

Using this error value, we will be backpropagating.

## Backpropagation

### 1. Calculating Gradients

The change in each weight is calculated as:

$$\Delta w_{ij} = \eta \times \delta_j \times O_j$$

Where:

- $\delta_j$ is the error term for each unit,
- $\eta$ is the learning rate.

### 2. Output Unit Error

For O3:

$$\delta_5 = y_5(1 - y_5)(y_{target} - y_5)$$

$$= 0.67(1 - 0.67)(-0.17) = -0.0376$$

### 3. Hidden Unit Error

For h1:

$$\delta_3 = y_3(1 - y_3)(w_{1,3} \times \delta_5)$$

$$= 0.56(1 - 0.56)(0.3 \times -0.0376) = -0.0027$$

For h2:

$$\delta_4 = y_4(1 - y_4)(w_{2,3} \times \delta_5)$$

$$= 0.59(1 - 0.59)(0.9 \times -0.0376) = -0.0819$$

### 4. Weight Updates

For the weights from hidden to output layer:

$$\Delta w_{2,3} = 1 \times (-0.0376) \times 0.59 = -0.022184$$

New weight:

$$w_{2,3}(\text{new}) = -0.22184 + 0.9 = 0.67816$$

For weights from input to hidden layer:

$$\Delta w_{1,1} = 1 \times (-0.0027) \times 0.35 = 0.000945$$

New weight:

$$w_{1,1}(\text{new}) = 0.000945 + 0.2 = 0.200945$$

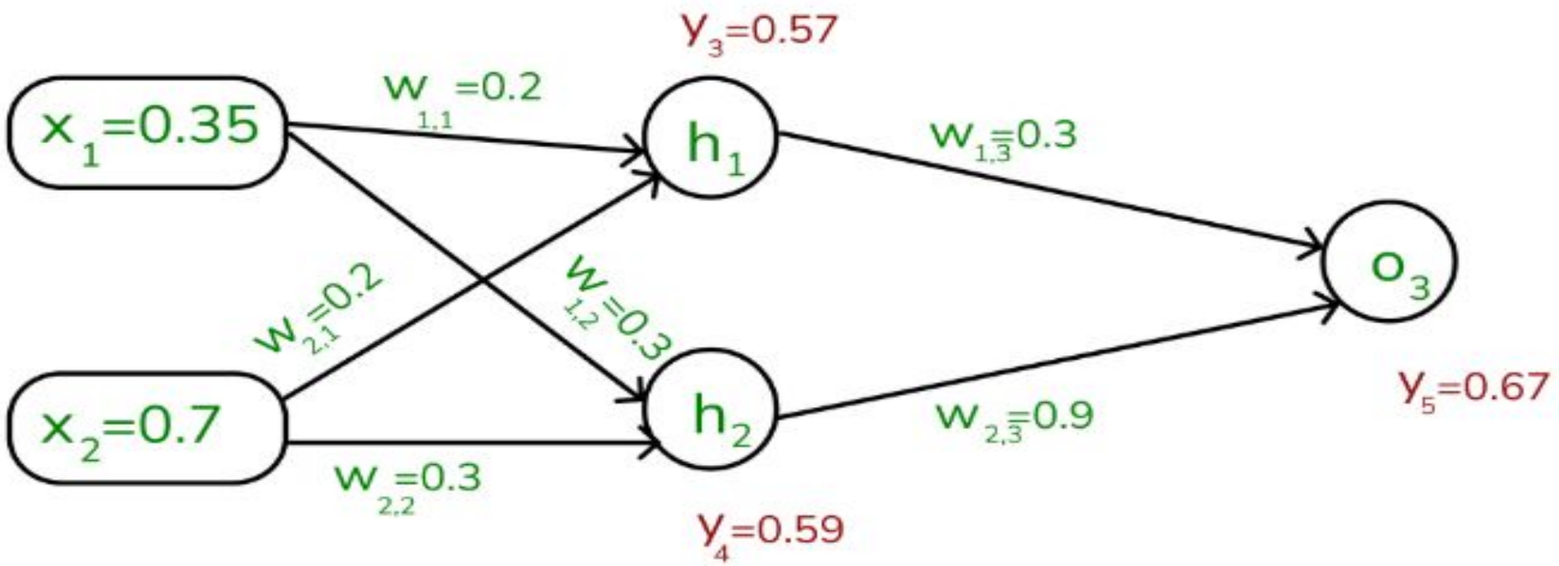
Similarly, other weights are updated:

$$a_2 = (w_{1,2} * x_1) + (w_{2,2} * x_2) = (0.3 * 0.35) + (0.3 * 0.7) = 0.315$$

$$y_4 = F(0.315) = \frac{1}{1+e^{-0.315}}$$

$$a3 = (w_{1,3} * y_3) + (w_{2,3} * y_4) = (0.3 * 0.57) + (0.9 * 0.59) = 0.702$$

$$y_5 = F(0.702) = \frac{1}{1+e^{-0.702}} = 0.67$$

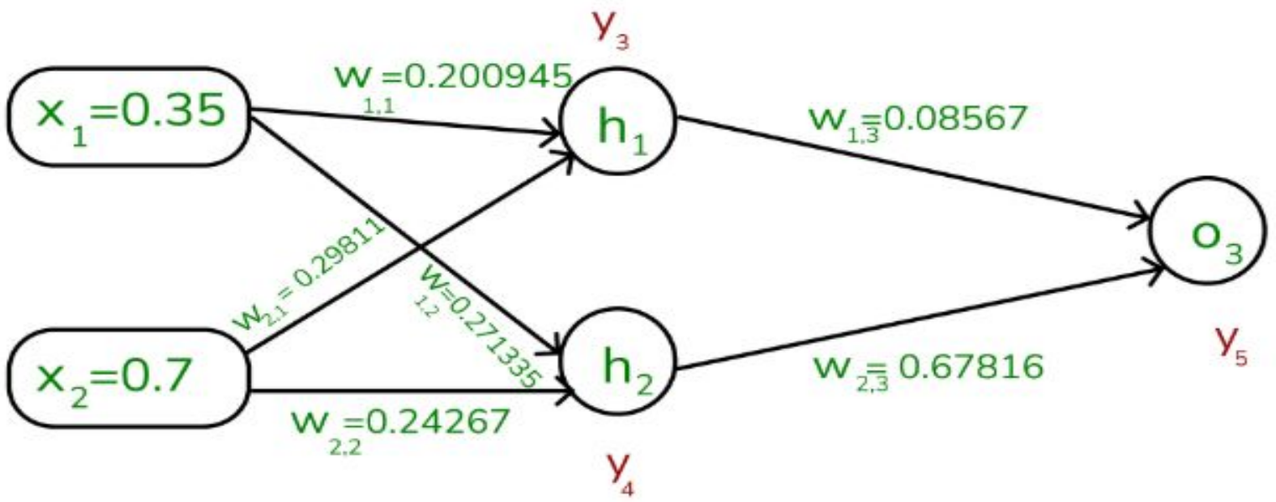



*Values of y3, y4 and y5*

Similarly, other weights are updated:

- $w_{1,2}(\text{new}) = 0.271335$
- $w_{1,3}(\text{new}) = 0.08567$
- $w_{2,1}(\text{new}) = 0.29811$
- $w_{2,2}(\text{new}) = 0.24267$

The updated weights are illustrated below,




*Through backward pass the weights are updated*

Final Forward Pass:

After updating the weights, the forward pass is repeated, yielding:

- $y_3 = 0.57$
- $y_4 = 0.56$
- $y_5 = 0.61$

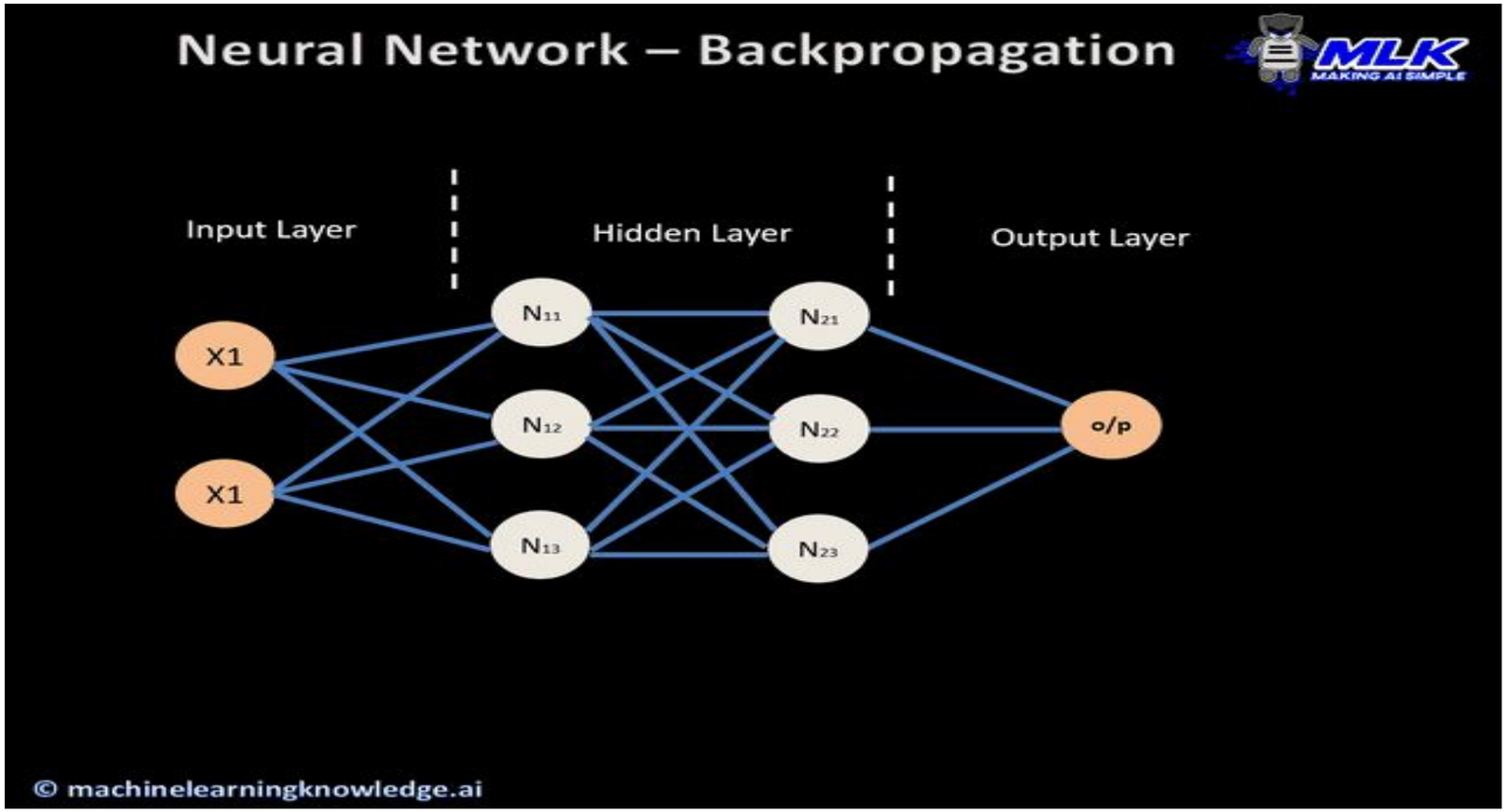
Since $y_5 = 0.61$ is still not the target output, the process of calculating the error and backpropagating continues until the desired output is reached.

This process demonstrates how backpropagation iteratively updates weights by minimizing errors until the network accurately predicts the output.

$$Error = y_{target} - y_5$$

$$= 0.5 - 0.61 = -0.11$$

This process is said to be continued until the actual output is gained by the neural network.




Backpropagation is a powerful algorithm that has revolutionized deep learning, enabling the training of complex neural networks with multiple layers. It's a fundamental concept that helps drive the advancement of artificial intelligence.

# Advantages

## Advantages of Backpropagation for Neural Network Training

The key benefits of using the backpropagation algorithm are:

- **Ease of Implementation:** Backpropagation is beginner-friendly, requiring no prior neural network knowledge, and simplifies programming by adjusting weights via error derivatives.
- **Simplicity and Flexibility:** Its straightforward design suits a range of tasks, from basic feedforward to complex convolutional or recurrent networks.
- **Efficiency:** Backpropagation accelerates learning by directly updating weights based on error, especially in deep networks.
- **Generalization:** It helps models generalize well to new data, improving prediction accuracy on unseen examples.
- **Scalability:** The algorithm scales efficiently with larger datasets and more complex networks, making it ideal for large-scale tasks.

# Challenges with Backpropagation

**Backpropagation** is powerful, it does face some challenges:

- **Vanishing Gradient Problem**: In deep networks, the gradients can become very small during backpropagation, making it difficult for the network to learn. This is common when using activation functions like sigmoid or tanh.
- **Exploding Gradients**: The gradients can also become excessively large, causing the network to diverge during training.
- **Overfitting**: If the network is too complex, it might memorize the training data instead of learning general patterns.

# Application

- Air traffic control could be automated with the help of location, elevation, direction, and speed of each radar blip which could be taken as input the network. As a result of this, the output would be the air traffic controller's instruction which would be a response to each blip.
- Evaluation and valuation of property, buildings, automobiles, machinery, etc. should be an easy task if done with the help of a neural network.
- Wagering on horse races, stock markets, sporting events, etc. could be analyzed with the help of neural network predictions.

# Conti..

- Criminal sentencing could be anticipated using a large sample of crime details as input and the resulting sentences as output.
- Direct mail advertisers could use neural network analysis of their databases to decide which customers ought to be focused on, and avoid wasting money on improbable targets.
- Schedule optimization of buses, airplanes, and elevators could be done efficiently by predicting demand.
- Staff scheduling optimization for restaurants, retail stores, police stations, banks, etc., could be done based on the customer flow, day of the week, paydays, holidays, weather, season, etc.

# Conti..

- Weather prediction may be possible using a neural network. Inputs would include weather reports from surrounding areas. Output(s) could be the future weather in specific areas based on the input information.
- Examination of medical issues is an ideal application for neural networks.
- Research in medical fields relies heavily on classical insights to dissect research data. Therefore, a neural network should be included in the researcher’s tool kit.