Deep Learning

Artificial Neural Network

Learning Objectives

By the end of this lesson, you will be able to:

- Analyze neural networks

- Interpret a variety of activation functions to update the weight

- Implement the backpropagation algorithm in Python

- Explain the concepts in perceptron

Learning Objectives

By the end of this lesson, you will be able to:

- Create forward propagation in perceptron

- Evaluate the role of forward propagation in perceptron

- Investigate backward propagation

- Apply the gradient descent algorithm

Business Scenario

A startup company called AI Detect is developing a new product that uses perceptron-based machine learning to detect fraud in financial transactions. The perceptron-based algorithm is specifically designed to identify patterns of fraudulent behavior in data, allowing the system to detect fraud with high accuracy.

The system takes in large volumes of financial transaction data and uses forward propagation to classify each transaction as either fraudulent or legitimate. If a transaction is classified as fraudulent, the system alerts the relevant authorities for further investigation.

The company has already tested the system on a small scale and achieved promising results, but it is now seeking funding to scale up its operations and expand its customer base. AI Detect plans to market its product to financial institutions and government agencies responsible for investigating financial crimes.

Neural Networks and Types of Neural Networks

Discussion
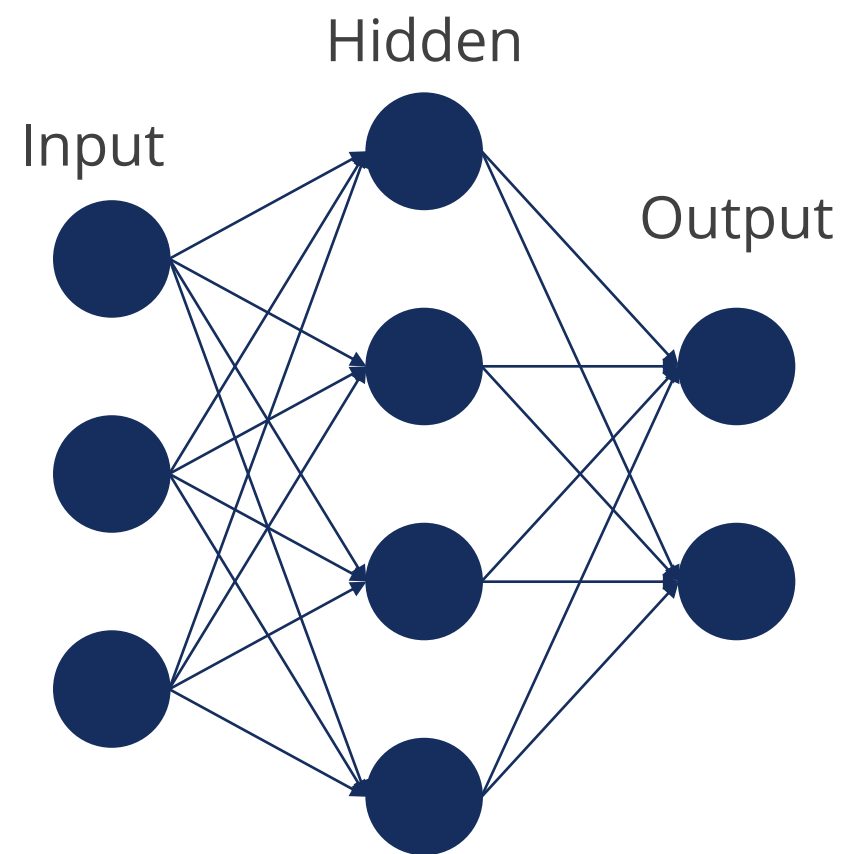
Discussion: Neural Networks

Duration: 10 minutes

- How does a neural network work?
- What are some real-world applications of neural networks?

Neural Networks

Neural networks consist of interconnected computation modules that simulate the behavior of biological neurons.

Hidden

Input

Output

Each neural network consists of multiple node layers such as:

✓ Input layers

✓ One or more hidden layers

✓ Output layers

Neural Networks

They are used to solve complex problems that require analytical calculations.

Some of the major neural networks are:

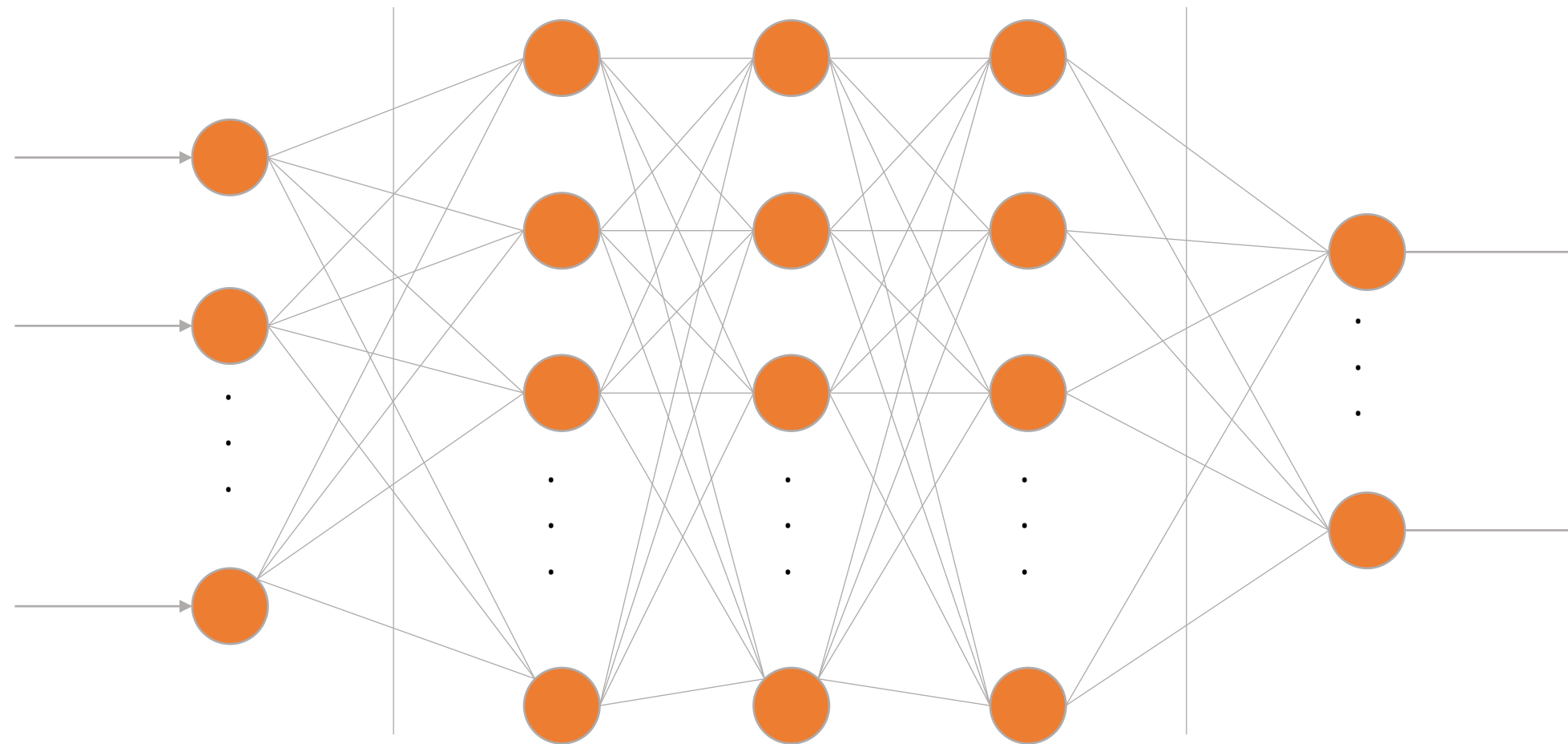| Artificial Neural Networks or ANNs | Deep Neural Networks or DNNs | Convolutional Neural Networks or CNNs | Recurrent Neural Networks or RNNs |

Artificial Neural Network

A neuron in an ANN is like a human neuron in the brain, and the ANN itself emulates the neural network of the brain.

A neuron serves as a computational unit responsible for information processing and collaborates with other neurons to facilitate complex decision-making.

Artificial Neural Network

ANNs were developed in the 1940s by Warren McCulloch and Walter Pitts.

The progress of AI was relatively slow, even during the latter half of the 20th century.

Artificial Neural Network

Perceptron: A basic binary classifier that takes inputs, applies weights, and passes them through an activation function to produce an output.
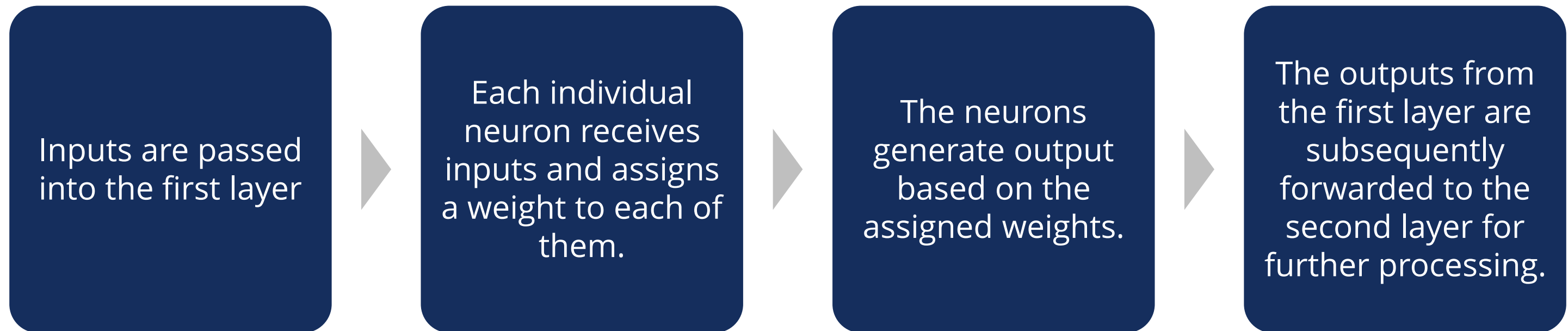

Multilayer Perceptron (MLP): Consists of multiple layers of perceptrons. Inputs pass through each layer, and the outputs of the last layer are the final outputs of the MLP. Can handle more complex problems by learning non-linear relationships.

Deep Learning: Neural networks with multiple hidden layers, enable them to learn complex representations of data. Overcome limitations of MLPs by introducing more layers. Use activation functions and backpropagation to learn optimal weights.

Artificial Neural Network

ANN includes the following steps to process information:

| Inputs are passed into the first layer | → | Each individual neuron receives inputs and assigns a weight to each of them. | → | The neurons generate output based on the assigned weights. | → | The outputs from the first layer are subsequently forwarded to the second layer for further processing. |

The process continues until the final output is produced.

Artificial Neural Network (ANN)

ANNs try to imitate neuronal activity in the brain to learn to recognize data patterns.
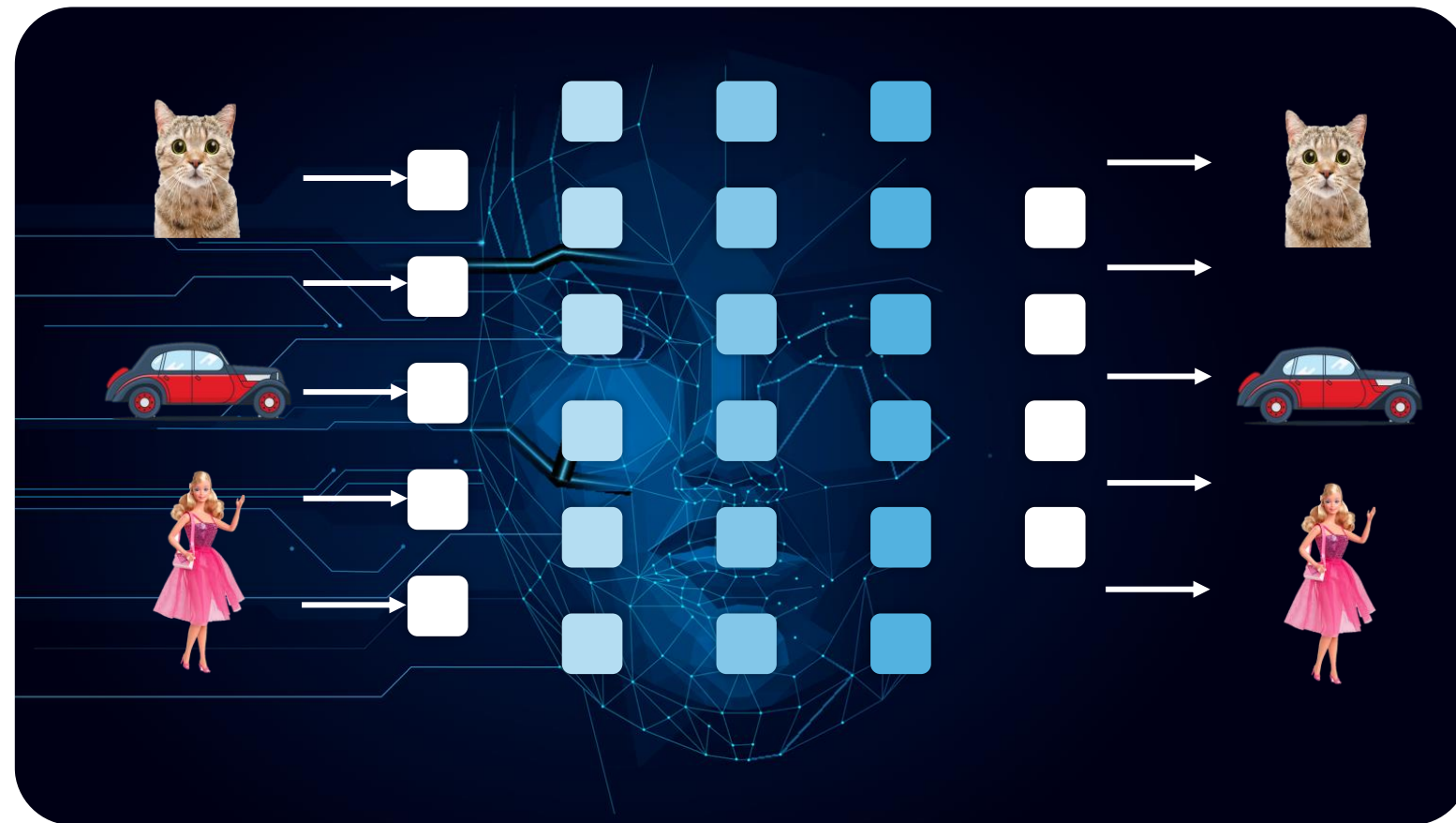
ANNs mimic brain activity to recognize patterns in data, using input nodes representing objects like a "cat," "car," and "Barbie doll" to produce corresponding outputs.

Deep Neural Networks (DNN)

A DNN is a multilayered computational model that processes data in a layered manner, refining information at each layer, analogous to how a human brain functions.

Hidden nodes

Input nodes

Output nodes
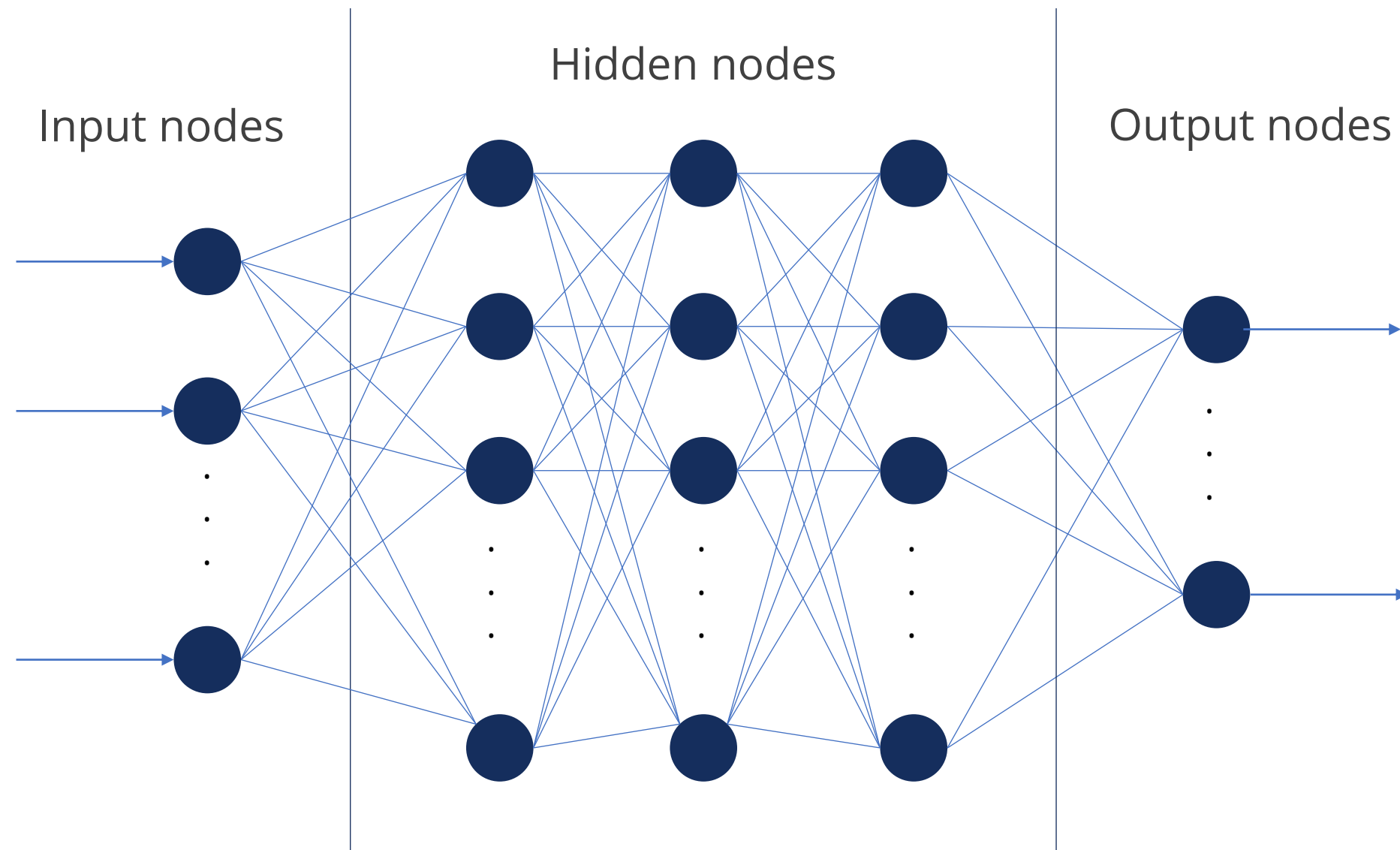
The depth of DNN enables it to effectively address complex problems, such as image processing and speech recognition.

Convolutional Neural Network (CNN)
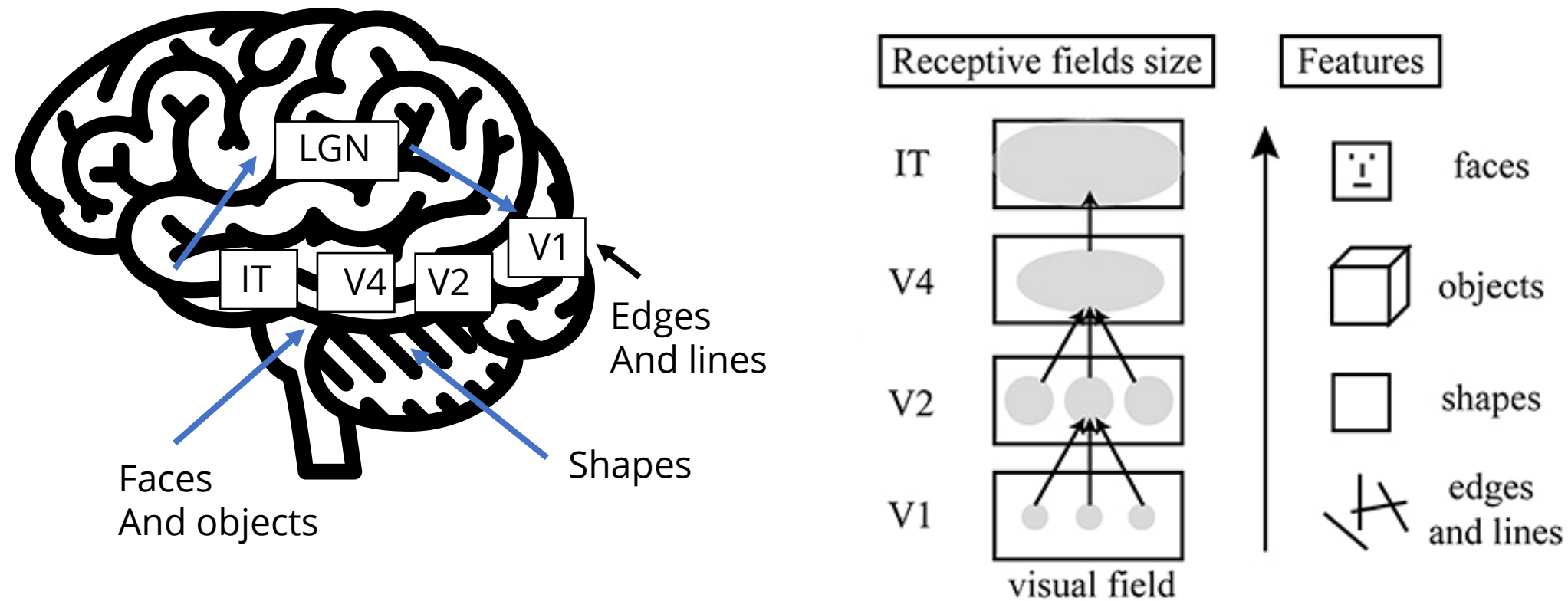
- The idea of CNNs was neurobiologically motivated by the findings of locally-sensitive and orientation-selective nerve cells in the visual cortex.
- The inventors of CNN designed a network structure that implicitly extracts relevant features.
- CNNs are a special kind of multilayer neural network.

Recurrent Neural Network (RNN)

The RNN remembers the analysis done up to a given point by maintaining a **state**.

Data in → **RNN** → Data out

Recurs new state
to itself
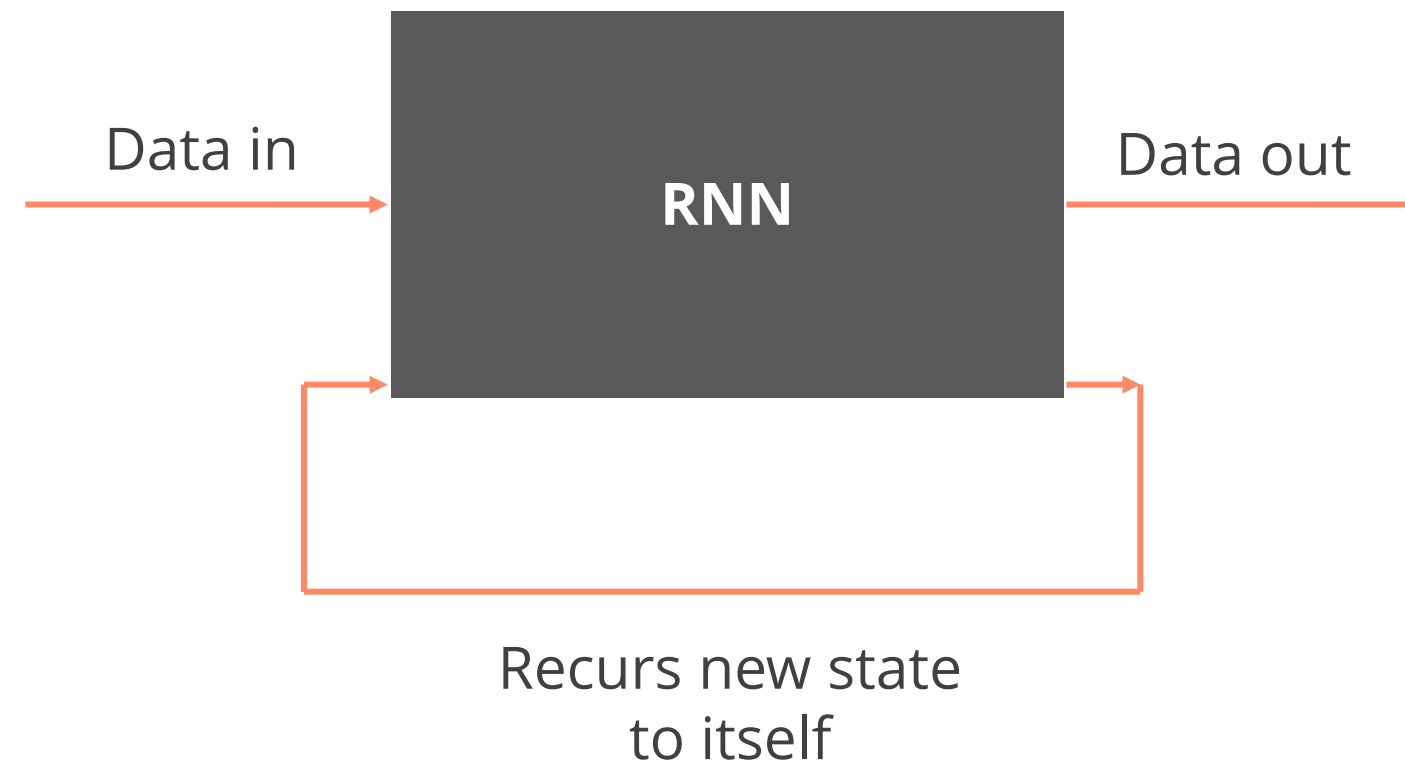
Note: You can think of the **state** as the **memory** of the RNN which recurs into the net with each new input.
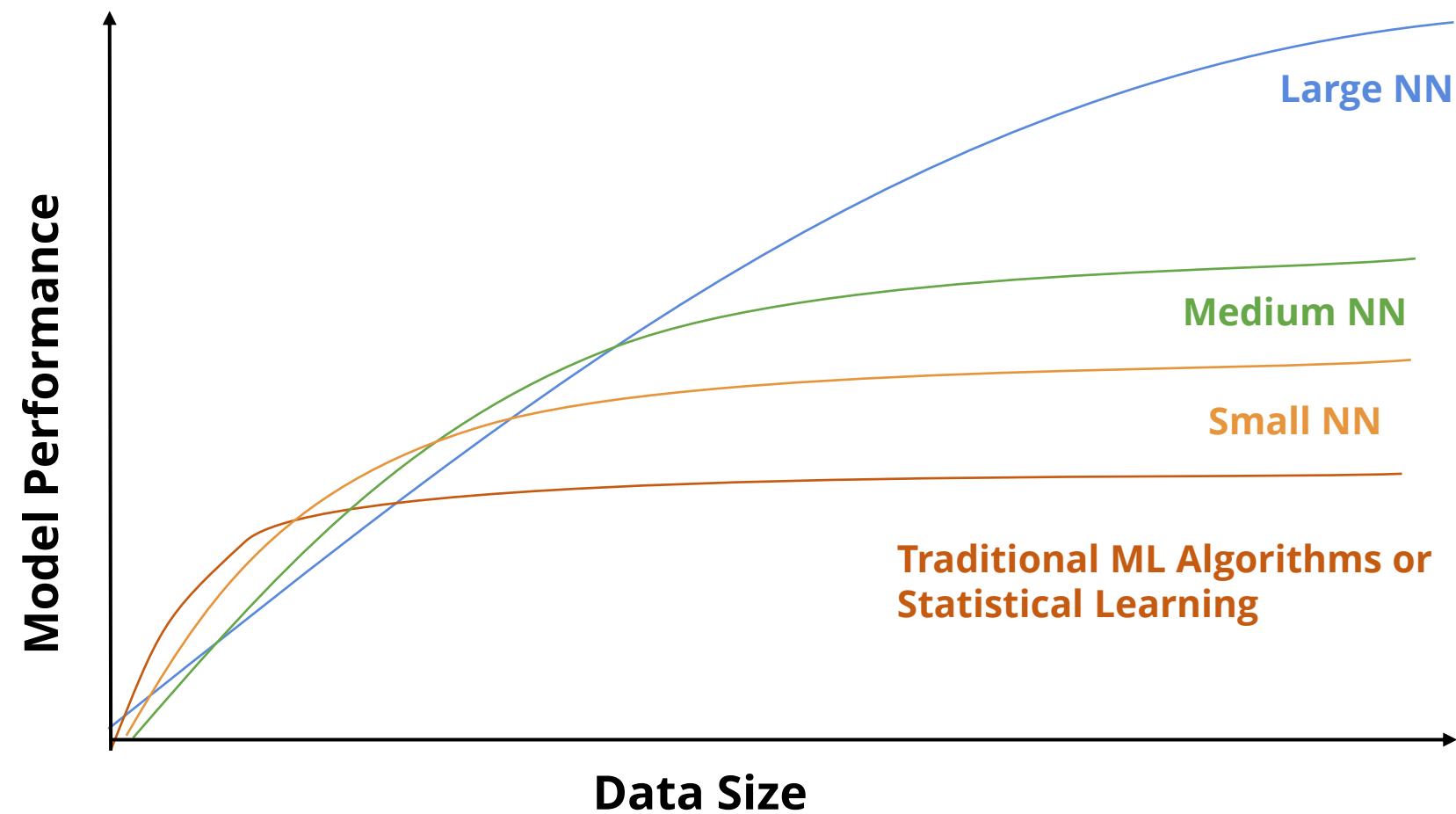
Network Architecture

One can programmatically set the number and type of neural layers and the number of neurons comprising each layer.
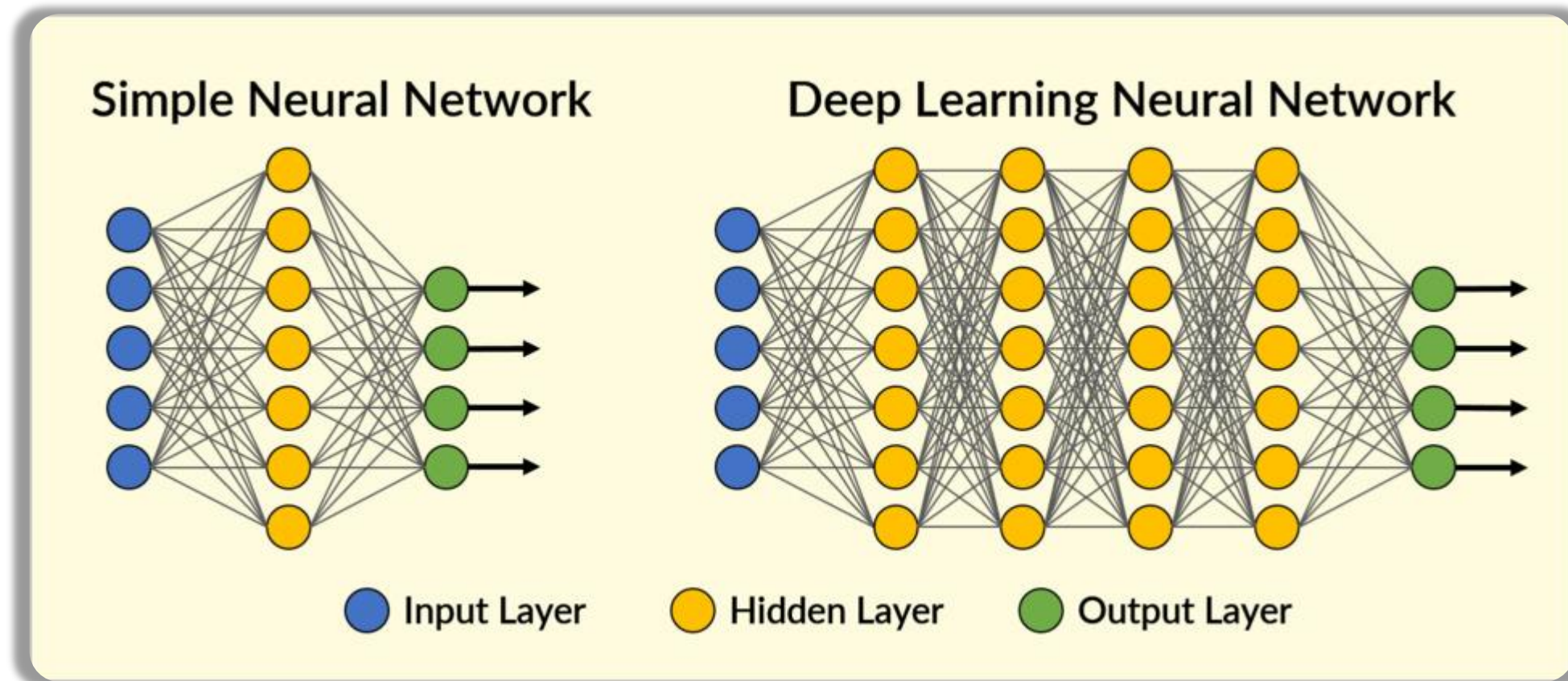
Deep neural networks consist of only one or two hidden layers.

Performance

Deep AI models have greater precision than conventional techniques but require more information to train and attain this precision.

Large NN

Medium NN

Small NN

Traditional ML Algorithms or Statistical Learning

Model Performance

Data Size

Combination of Neural Network Layers

The number of layers in a CNN or RNN depends on the complexity of the task and data, with deeper architectures often used for more complex problems.
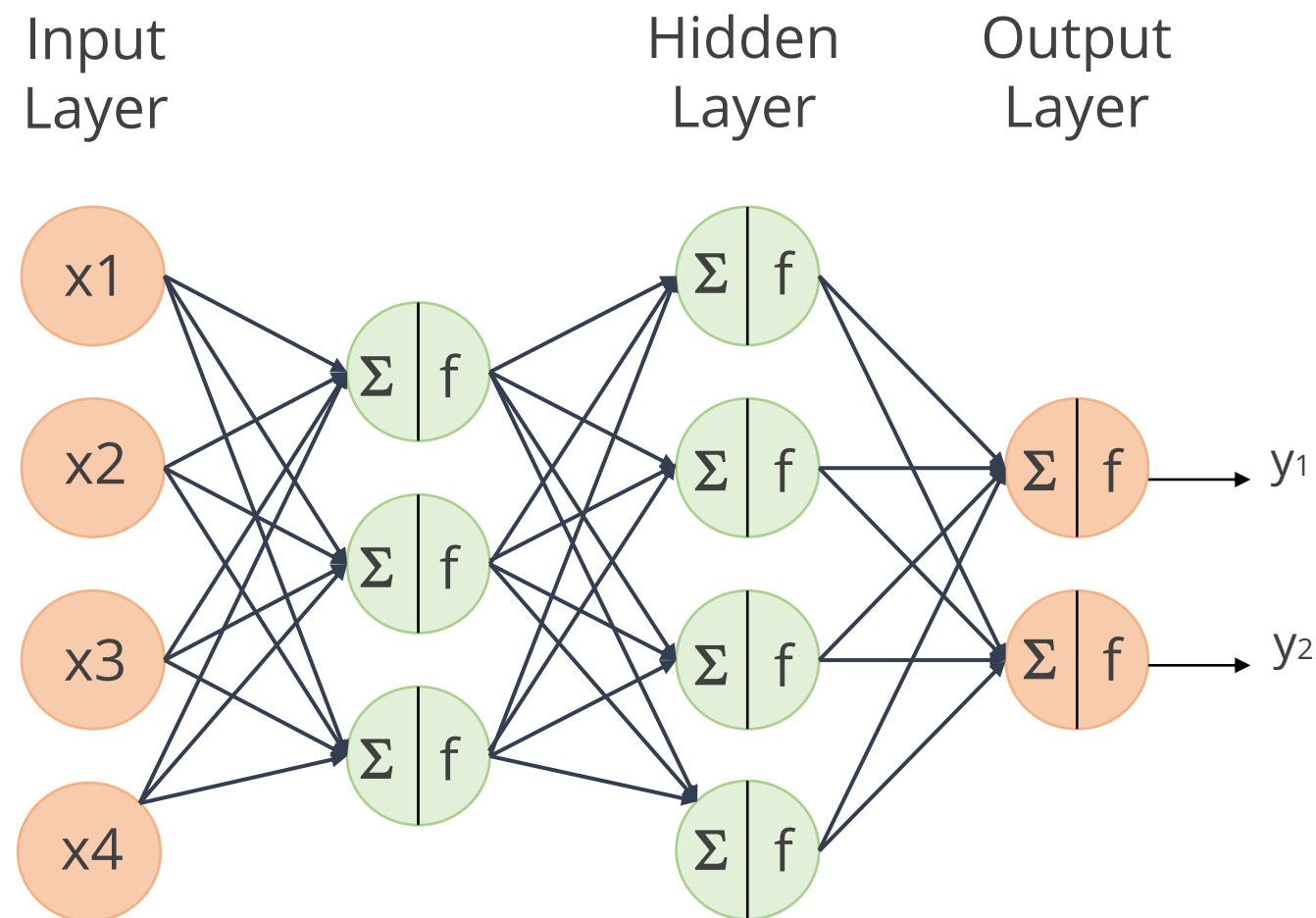
- The input layer depicts the dimensions of the input vector.

- The hidden layer depicts the intermediary nodes that divide the input space into regions with soft boundaries.

- The output layer depicts the output of the neural network.

Discussion: Neural Networks

Duration: 10 minutes

- How does a neural network work?

Answer: A neural network operates by allowing information to flow through layers of interconnected neurons. These neurons receive inputs, apply weights to the inputs, and pass the weighted results through activation functions to generate outputs.

- What are some real-world applications of neural networks?

Answer: Neural networks find applications in various fields, including image recognition, natural language processing, speech recognition, fraud detection, autonomous vehicles, and drug discovery, among others.

Neurons

Biological Neurons

Neurons are interconnected nerve cells that build the nervous system and transmit information throughout the body.
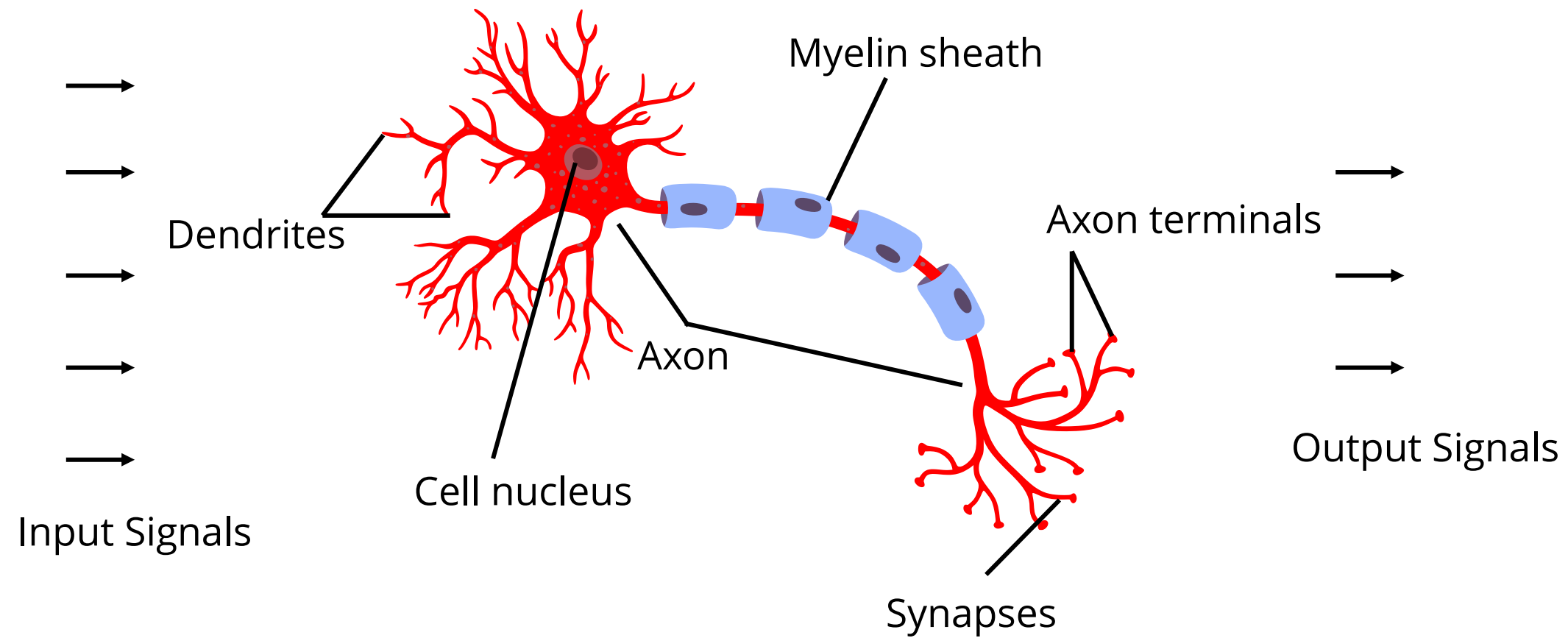
Biological Neurons: A Simplified Illustration

The components of the biological neuron network are as follows:

Dendrites receive inputs from other neurons.

The **cell** nucleus is used for information processing.

Parts of a Neuron

Axons transmit the biological neuron's output.

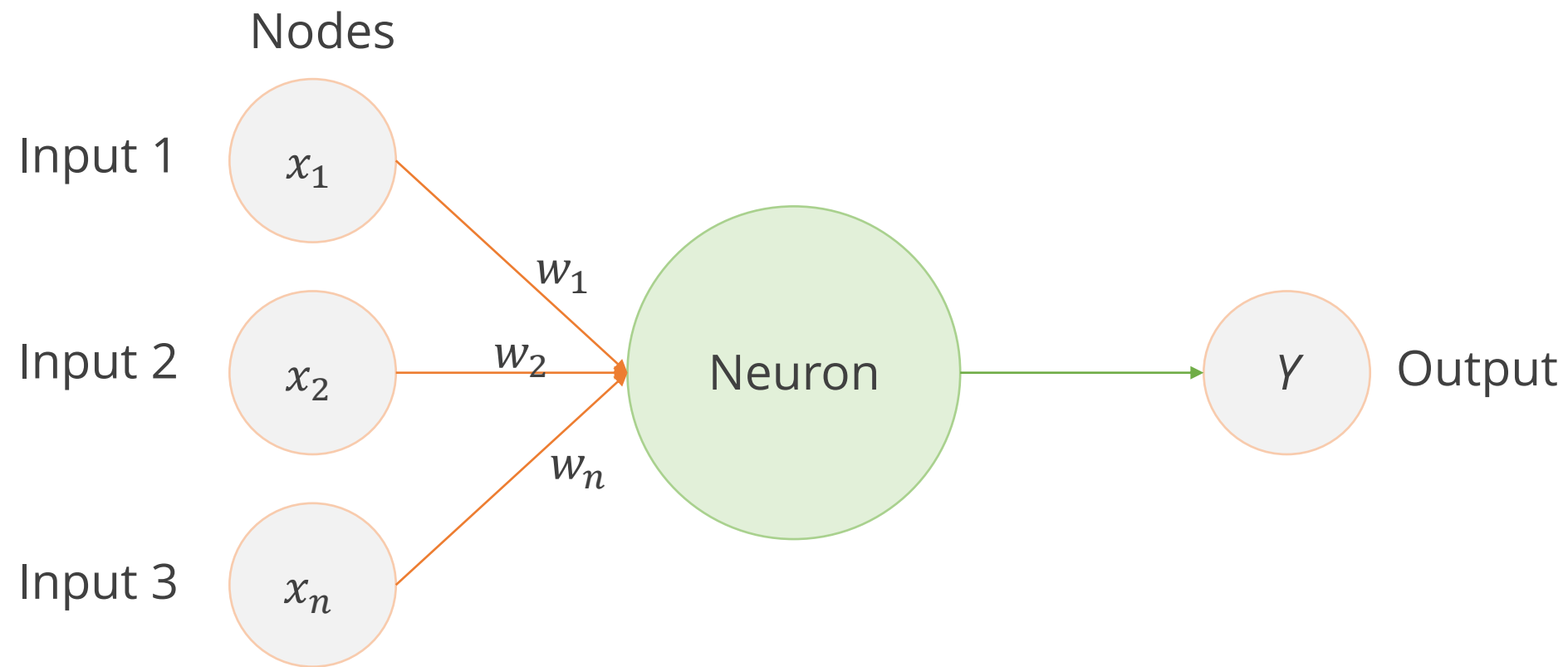**Synapse** is the connection between two nerve cells.

History of Artificial Neuron

In 1943, scientists Warren McCullock and Walter Pitts released the first version of their simplified model of a brain cell.


https://www.historyofinformation.com/image.php?id=5647
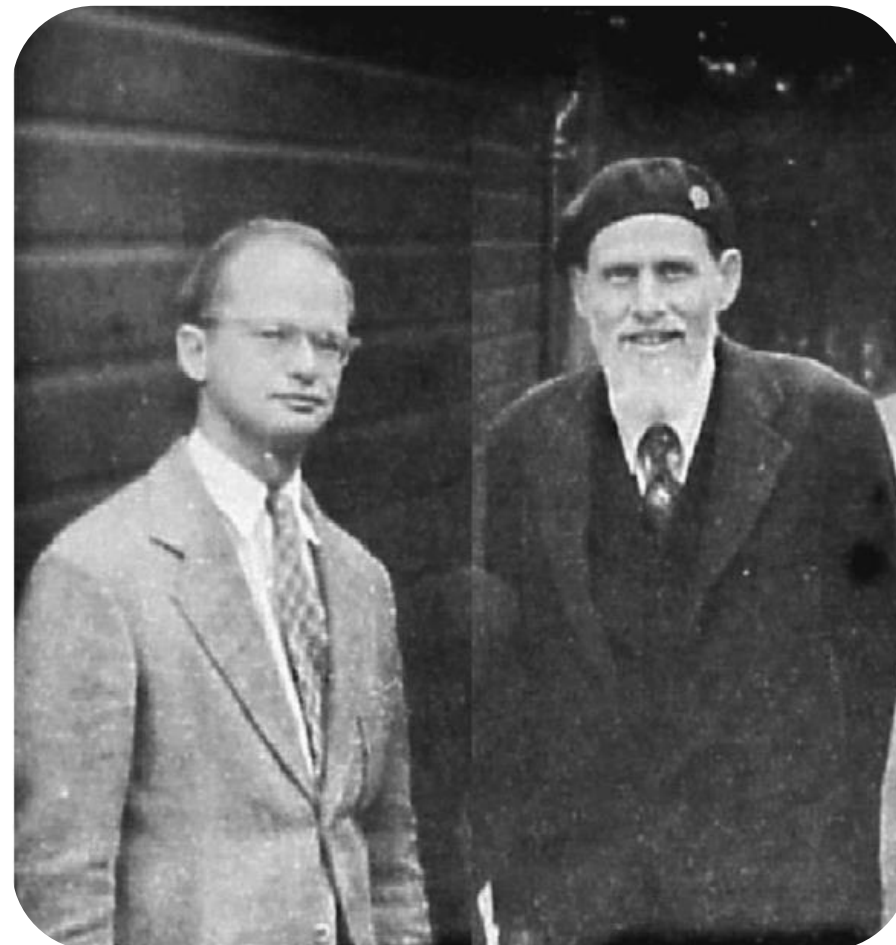
Definition of Artificial Neuron

Nodes

Input 1

x_1

Input 2

x_2

w_1

w_2

Neuron

w_n

Input 3

x_n

Y Output

An artificial neuron is analogous to biological neurons, where each neuron takes inputs, adds weights to them separately, sums them up, and passes this sum through a transfer function to produce a nonlinear output.
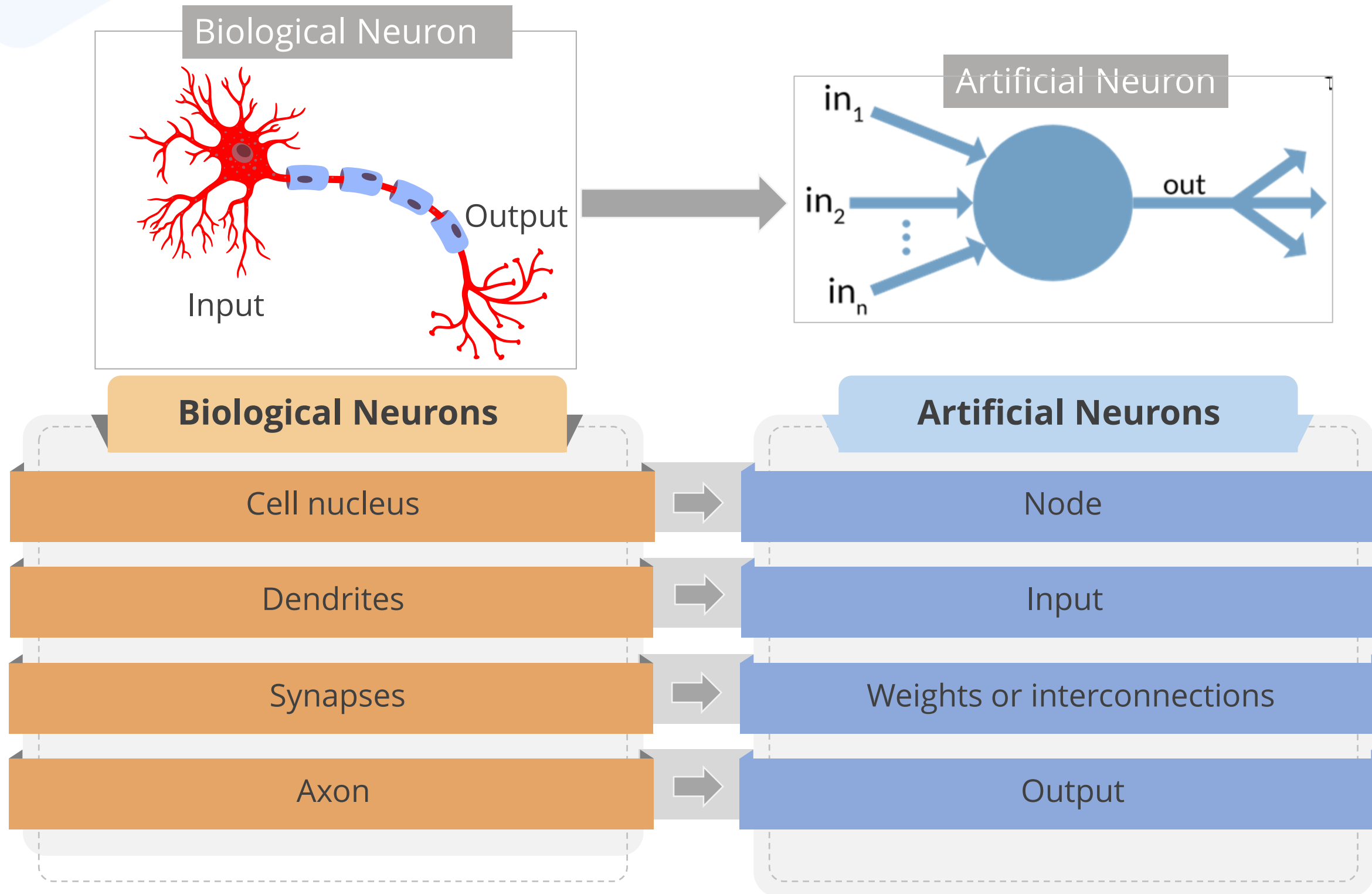
Artificial Neurons

x_1

x_2

x_3

x_n

w_1

w_2

w_3

w_n

1

b

Summer

Threshold Unit

$$\sum_{i=1}^{n} x_1 - w_1 + b$$
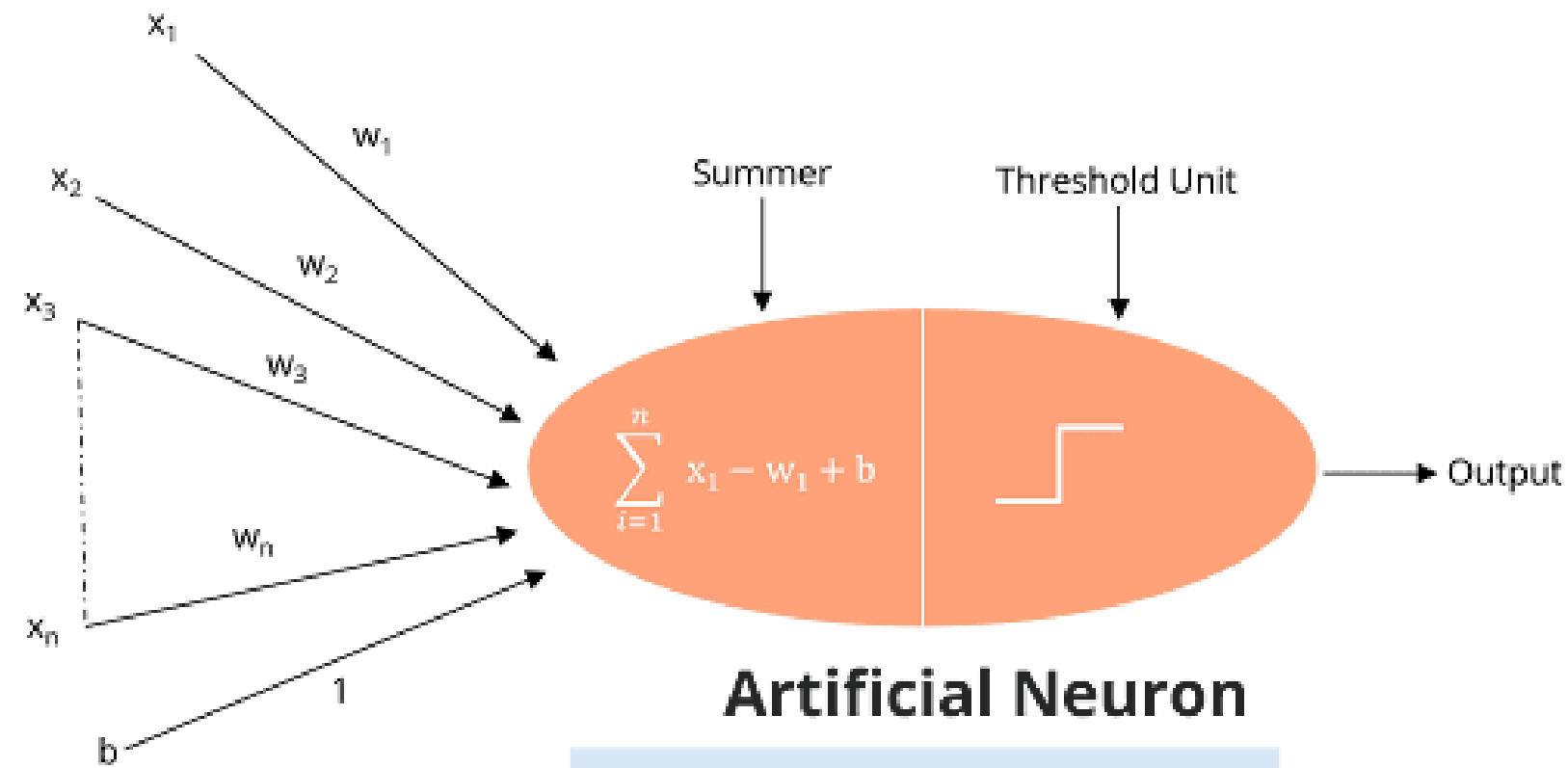
Output

Artificial Neuron

$w_1 w_2 w_3 w_n$ – Weights of Connection
$x_1 x_2 \ x_3 x_n$ – Inputs | b - Bias

- A nerve cell is considered similar to a simple logic gate with binary outputs.
- Dendrites can process the input signal with a certain threshold, such that if the signal exceeds the threshold, the output signal is generated.

Biological Neurons vs. Artificial Neurons

Biological Neuron

Output

Input

Artificial Neuron

in_1

in_2

in_n

out

Biological Neurons		Artificial Neurons
Cell nucleus	→	Node
Dendrites	→	Input
Synapses	→	Weights or interconnections
Axon	→	Output

Perceptron

Discussion

Discussion: Perceptron

Duration: 10 minutes

- What is a perceptron?
- How does training a perceptron work?

What Is a Perceptron?

A perceptron is a simplified model of a neuron used in neural networks.

A human brain has billions of neurons.

Neurons are interconnected nerve cells that process and transmit chemical and electrical signals.

Dendrites are branches that receive information from other neurons.

History of the Perceptron

The concept of the human brain led to the invention of the perceptron.

In 1957, Frank Rosenblatt introduced the perceptron.

It is an artificial neuron that performs certain computations to detect features or business intelligence in the input data.

Equation of the Perceptron

Perceptron is a function that maps its input x, which is multiplied with the learned weight coefficient. An output value $f(x)$ is generated.
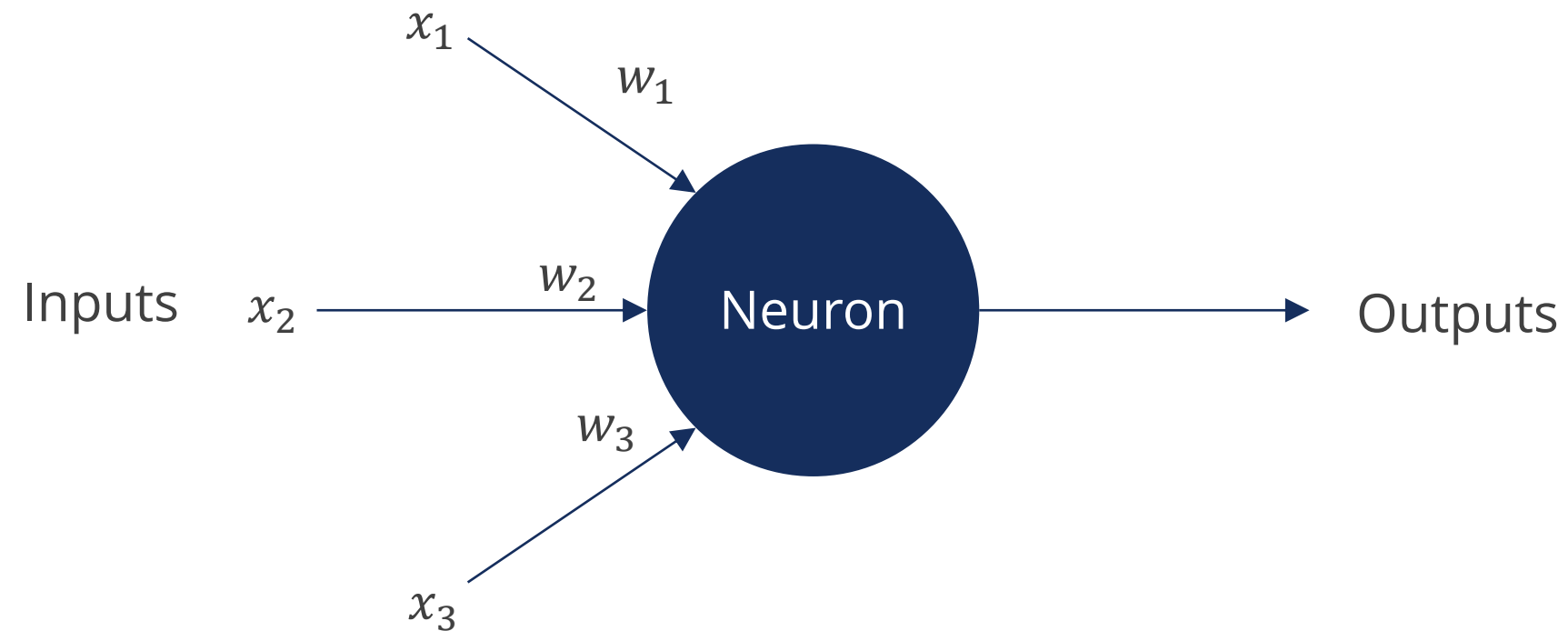
Perceptron Equation

$$f(x) = \begin{cases} 1 \; if \; w.x + b > 0, \\ 0 \; Otherwise \end{cases}$$

- w: It denotes a vector of real-valued weights.

- b: It denotes bias, which is an object that adjusts the bounding line without any dependency on the input values.

- x: It denotes the vector of input x values.

Working of a Perceptron

The perceptron below has three inputs, x_1, x_2, and x_3, and one output.

The importance of the inputs is determined by the corresponding weights w_1, w_2, and w_3.

Working of a Perceptron

The output is a sum of inputs multiplied by their corresponding weights.

$$x_1 * w_1 + x_2 * w_2 + x_3 * w_3$$

The output is calculated for any number of inputs.

It is also a linear classification algorithm.

It can learn a decision boundary that is a straight line or a hyperplane.

Perceptron works better with data that has linearly separable classes.

Working of a Perceptron

The weights of the perceptron are trained with different sets of inputs.

At every passage of inputs, the perceptron produces the output as either 0 or 1, which is compared to the ground truth.

The weights are adjusted accordingly for better predictions.

A perceptron works well when the classes in the data are linearly separable.

Weights and Biases in a Perceptron

The following are the components of a basic artificial neuron:
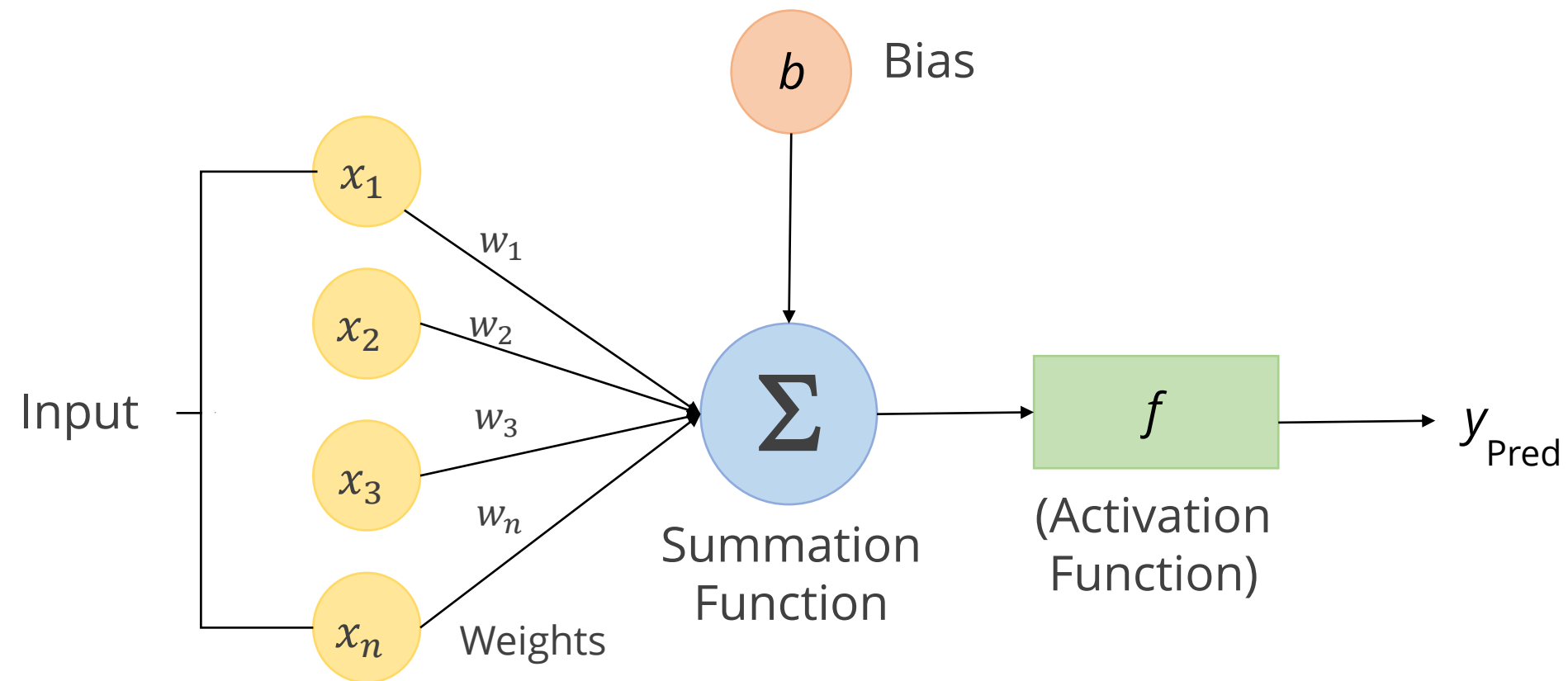
Inputs

Weights

Bias

Summation function

Activation function

x_1

x_2

x_3

x_n

Input

w_1

w_2

w_3

w_n

Weights

b Bias

Σ

Summation Function

f

(Activation Function)

y_{Pred}

Equation:
$$Output = f(w * x + b)$$

Weights and Biases in a Perceptron

The following are the components of a basic artificial neuron:

Inputs	They are a set of values for which one needs to predict the output value.
Weights	They are a set of real values that are associated with each feature and state the importance of that feature in predicting the final value.
Bias	It is used for shifting the activation function left or right and can be referred to as a y-intercept in the line equation.
Summation function	It binds the weights and inputs together and finds their sum.
Activation function	It is used to depict nonlinearity in the model.

Perceptron: Feedforward Neural Network

It is a fundamental building block of artificial neural networks and a basic unit of computation in deep learning.

Input layer   Hidden layer   Output layer
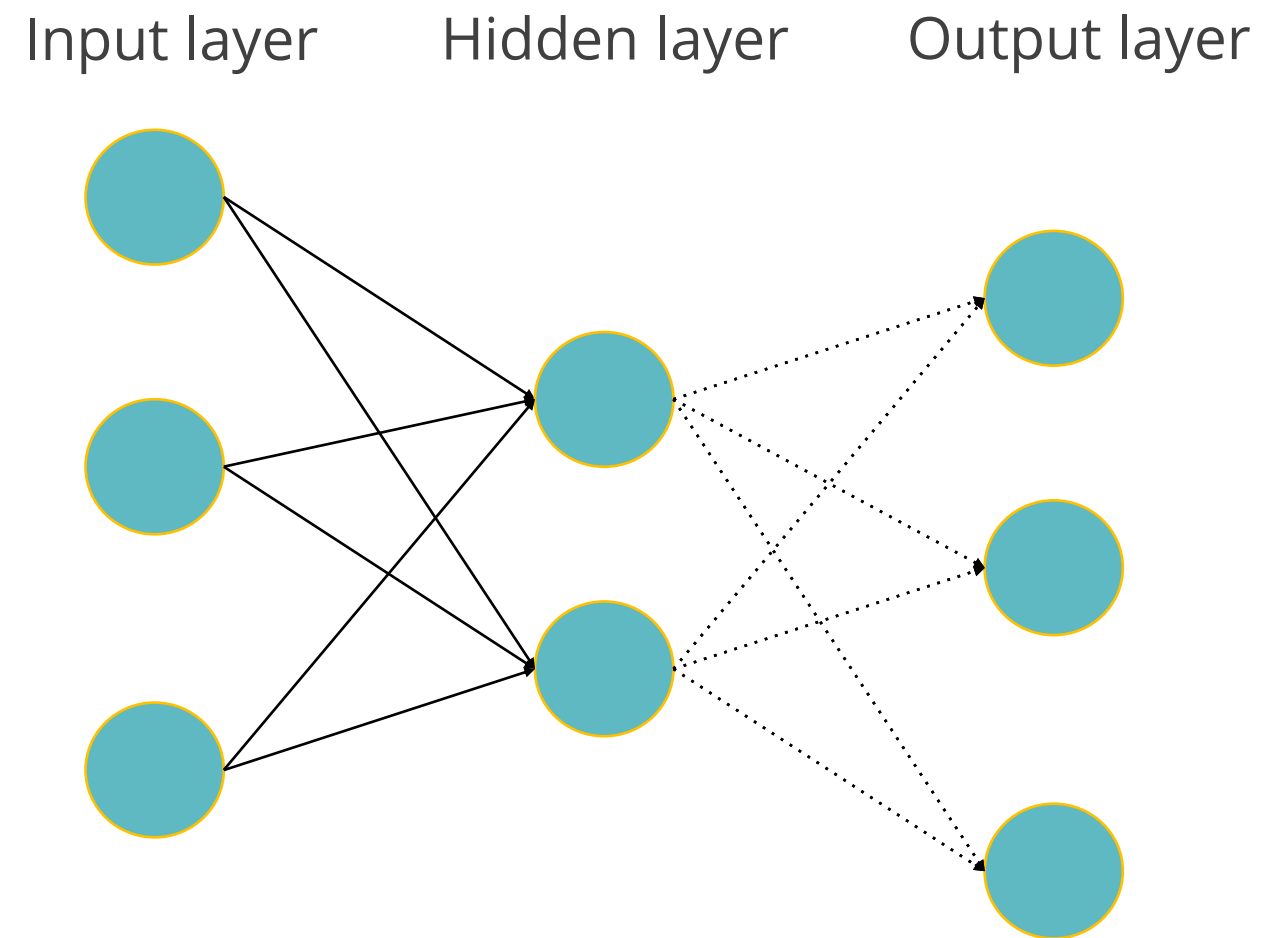
- Information flow is unidirectional: Process input to output without any backward or recurrent connections

- Information is distributed: Weight each input independently to contribute to decision-making

- Information processing is parallel: Perform calculations on all inputs simultaneously

Feedforward Neural Networks

In feedforward neural networks (FNN), the information only moves from the input layer to the output layer through the hidden layers.

It cannot remember anything that happened in the recent past, except its training.

The information moves straight through the network and never touches a node twice.

Since it only considers the current input, it has no notion of order in time.

It has no memory of the input it receives.
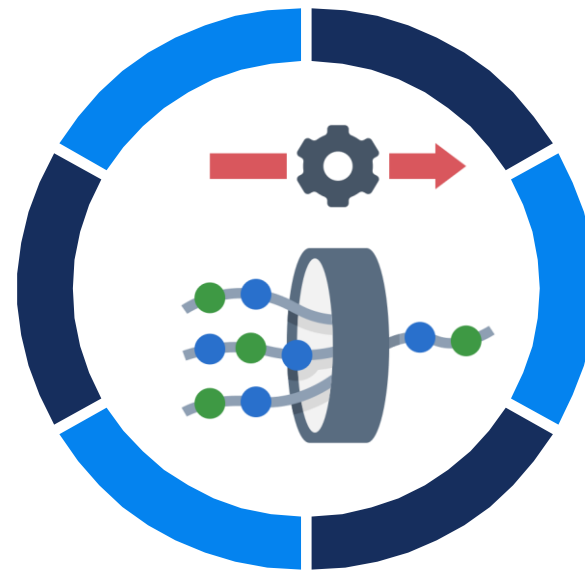
Multilayer Perceptrons

They are feed-forward artificial neural network that create a collection of outputs from a set of inputs.

Input layer

Hidden layers

Output layer

Linear Algebraic Equation for a DNN or MLP:

$$y = f(w * x + b)$$

where y is the output,

x is the input,

w is the weight matrix,

b is the bias vector,

and f is the activation function.
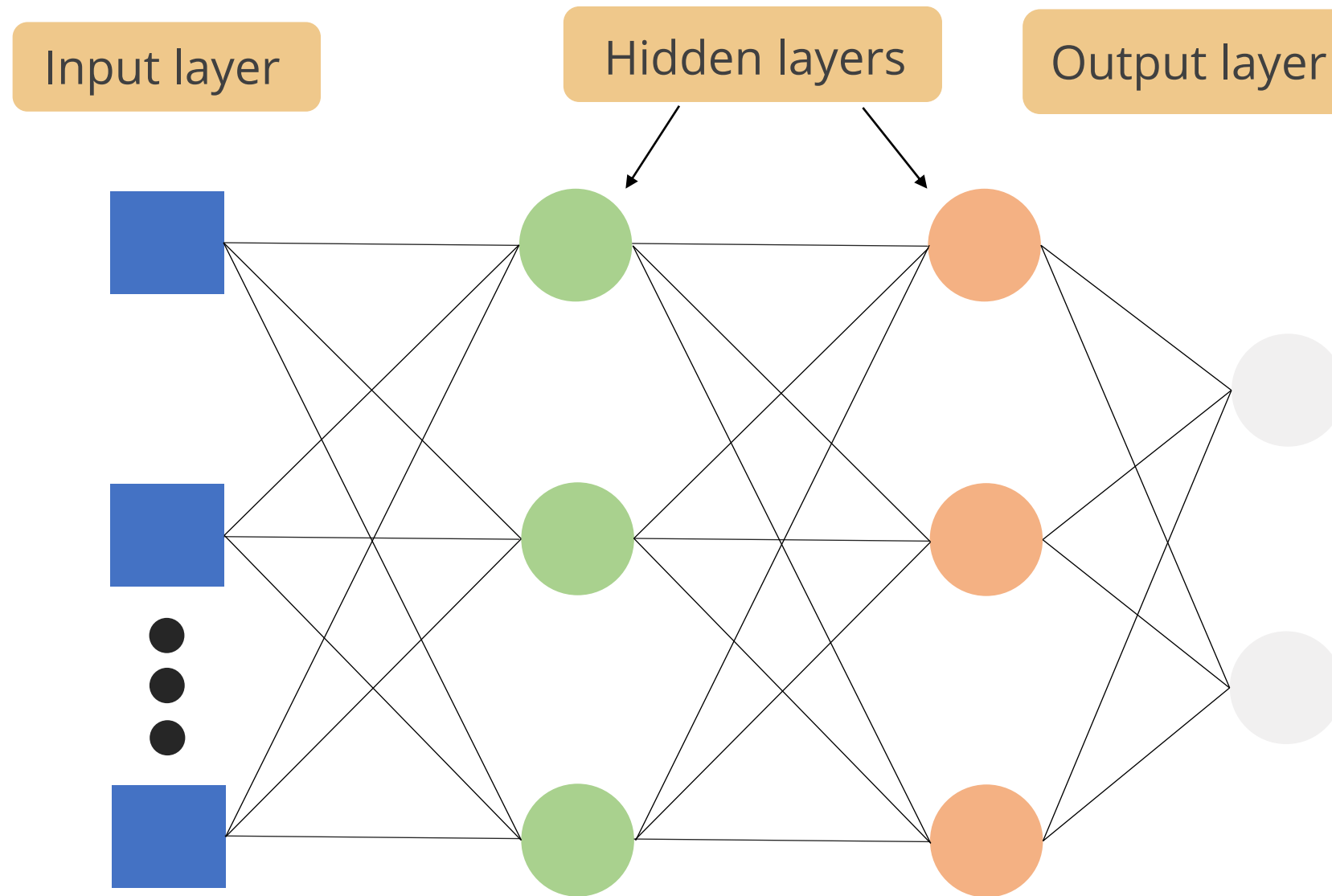
They are formed from multiple layers of the perceptron.

The Exclusive OR (XOR) Problem

A perceptron can learn anything that it can represent, that is, anything separable from a hyperplane. However, it cannot represent XOR since it is not linearly separable.

X1	X2	X1 XOR X2
-1	-1	-1
-1	1	1
1	-1	1
1	1	-1
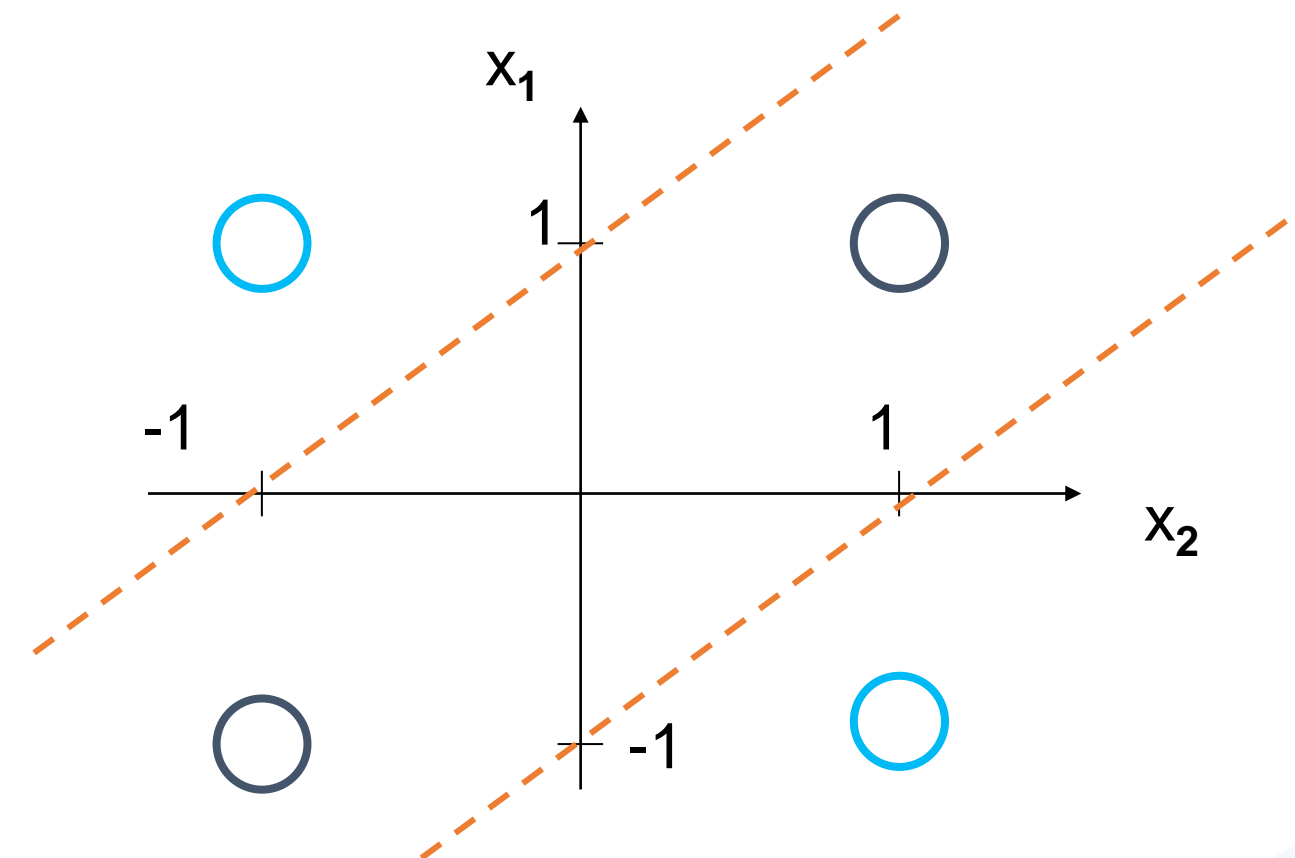
Discussion: Perceptron

Duration: 10 minutes

- What is a perceptron?

Answer: A perceptron serves as a binary classifier in artificial neural networks.

- How does training a perceptron work?

Answer: Training a perceptron involves adjusting the weights connected to each input by comparing the predicted output to the desired output and minimizing the error.

Let's understand the concept of perceptron using Jupyter Notebooks.

- 3.04_Perceptron-Based Classification Model

Note: Please refer to the Reference Material section to download the notebook files corresponding to each mentioned topic

Activation Function

Activation Function

Before producing an output from a perceptron, it is important to decide whether to activate the neuron or not.
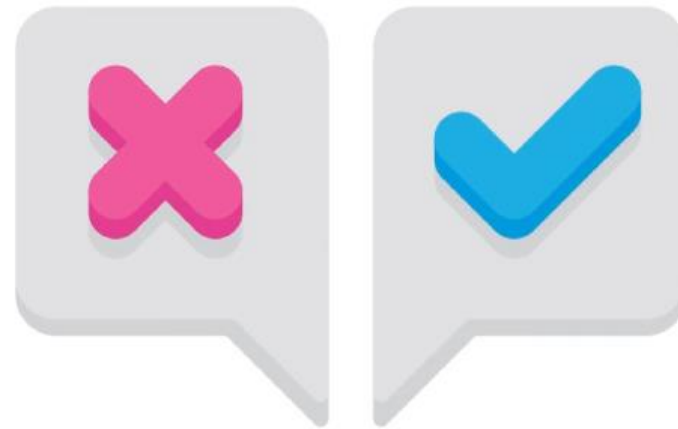
The activation function helps achieves this.

Activation Function

Now, consider the following:

- ✓ What are its functions?

- ✓ Why is the activation needed?

- ✓ What are the types of activation functions?

Functions of Activation Function

The sum of products of inputs and weights is passed to an activation function.
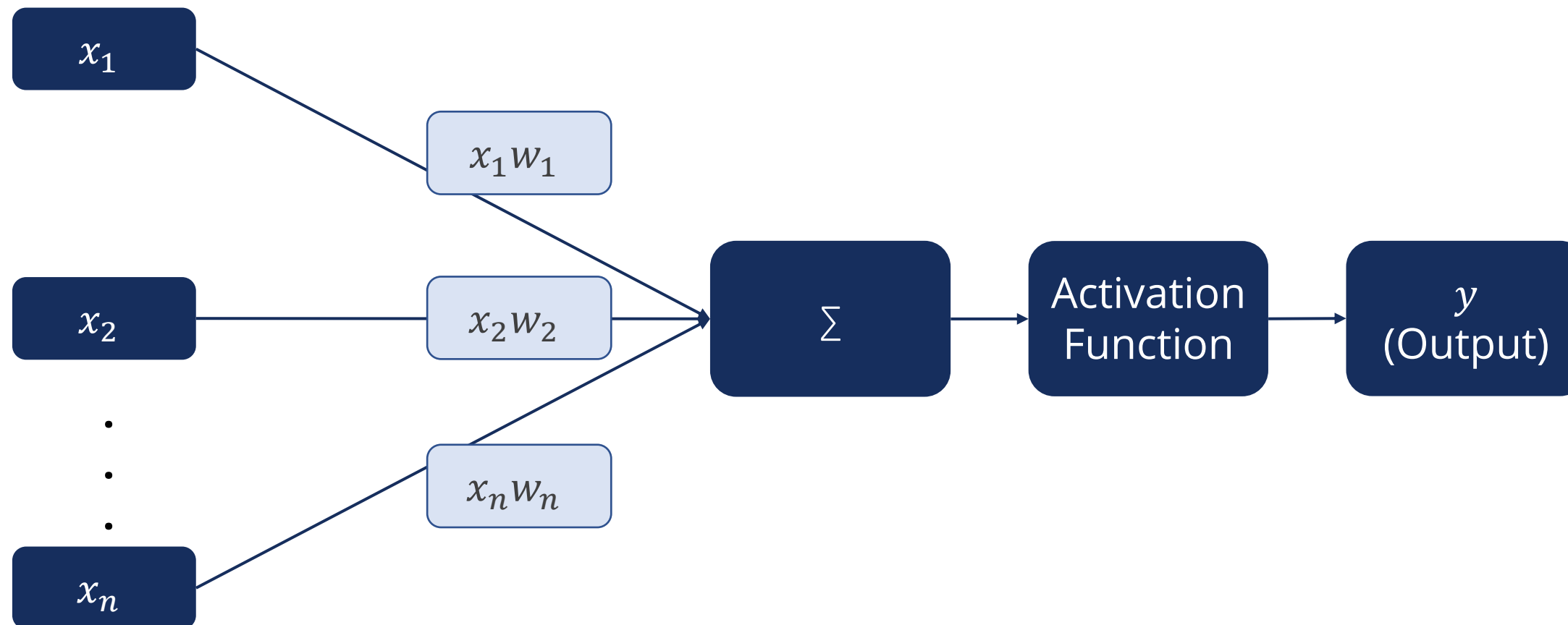
It narrows the value between 0 and 1 or -1 and +1 based on the type of activation function used.

Functions of Activation Function

Output y = Σ(weights * input + bias), ranging from -∞ and +∞

Bound the output to get the desired prediction or generalised results.

After adding activation,
y = Activation function (Σ(weights * input + bias))

Need For Activation Function

Activation functions are mathematical equations that determine the output of a neural network model.
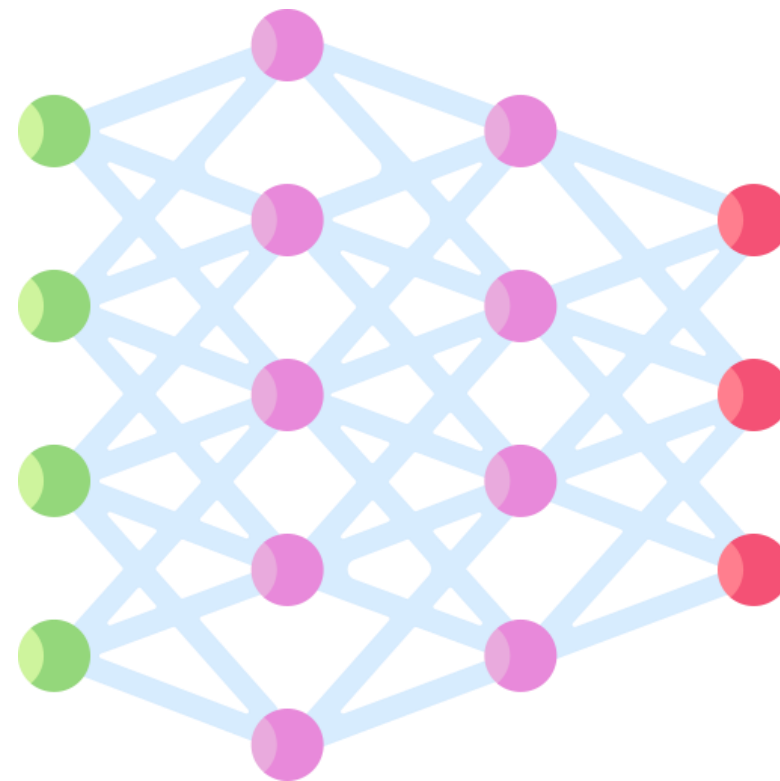
It plays a crucial role in a neural network's ability to converge and affect the convergence speed.

Need For Activation Function

The activation function is needed in neural networks to introduce non-linearity and enable the model to learn complex relationships and make more expressive predictions.

Activation Function

Input

Output

Neuron
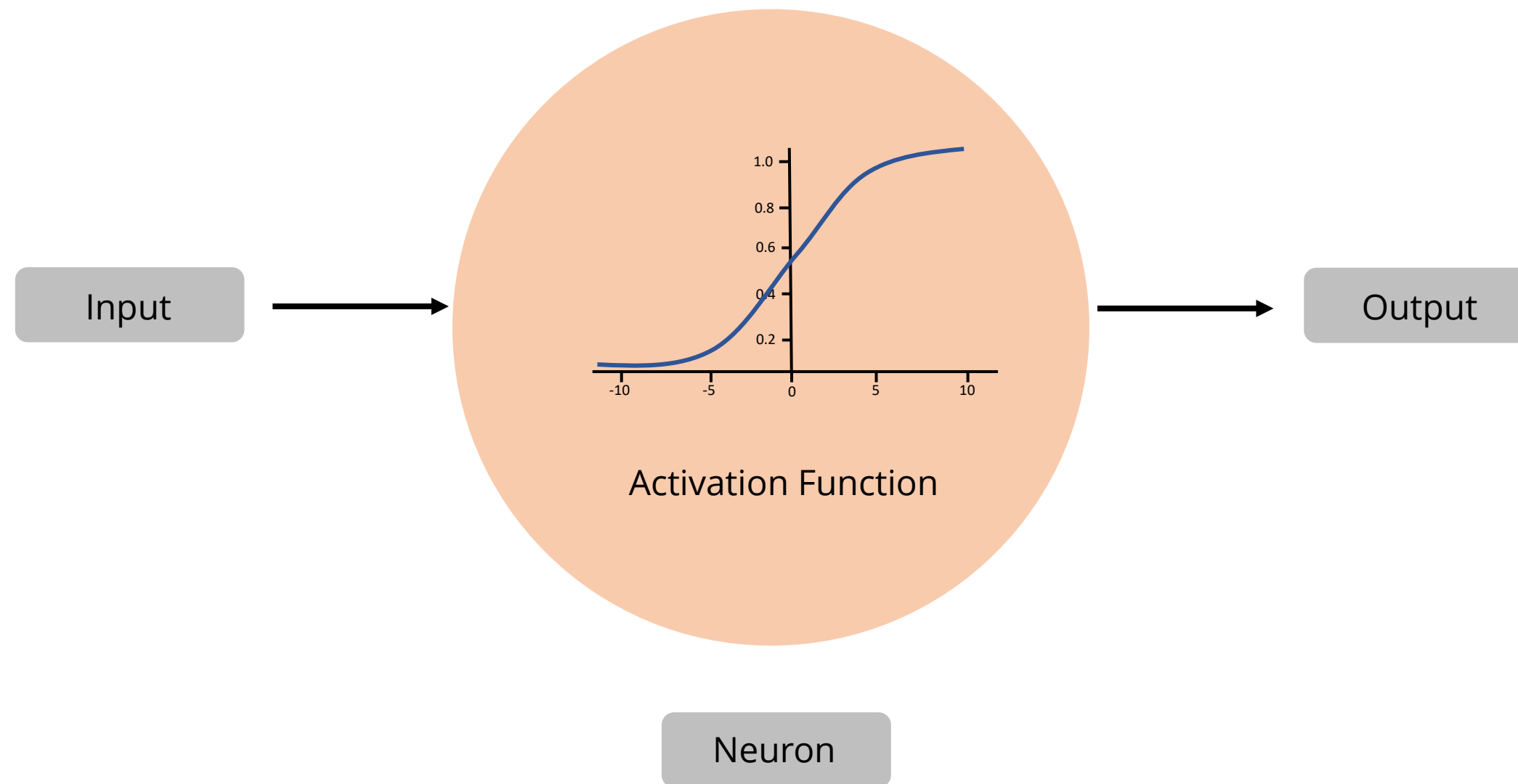
Activation functions limit output by preventing large positive or negative values in neural networks.

Types of Activation Functions

Activation functions squash inputs to outputs by using a threshold value.

(a) Step function

(b) Sign function

(c) Sigmoid function
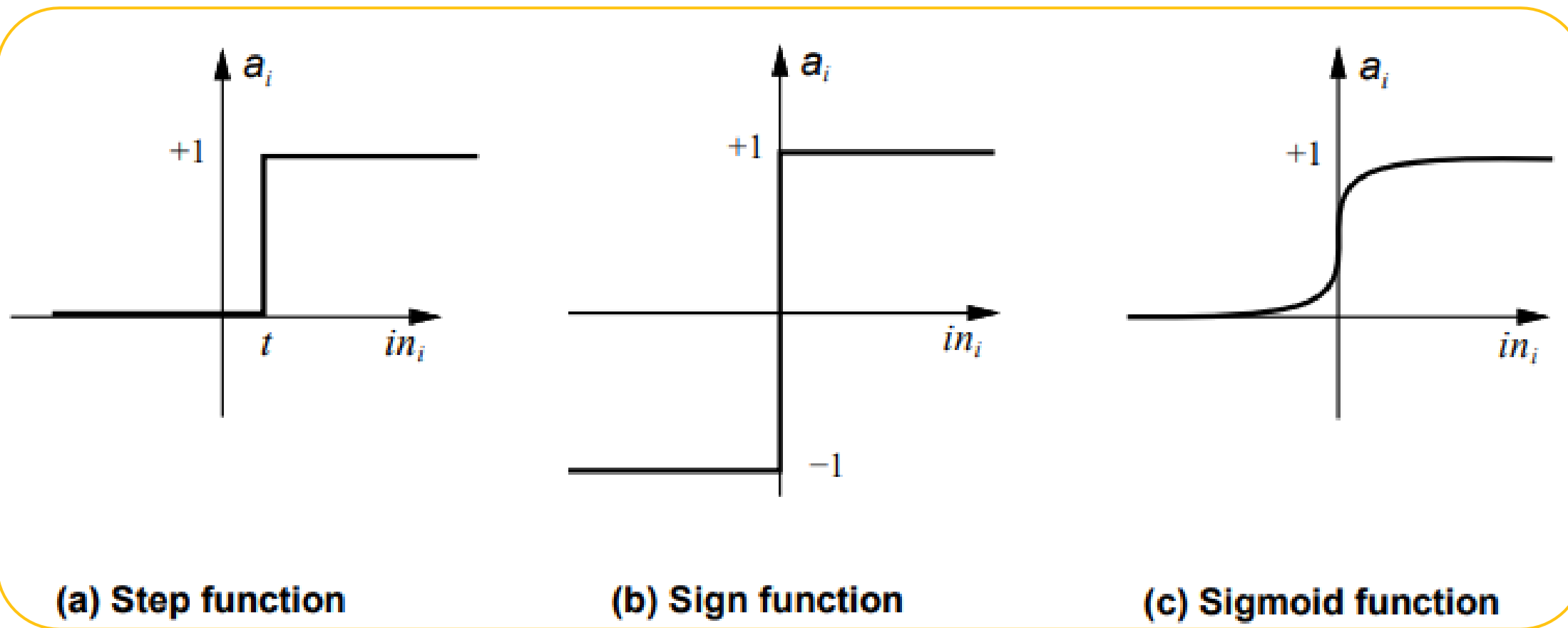
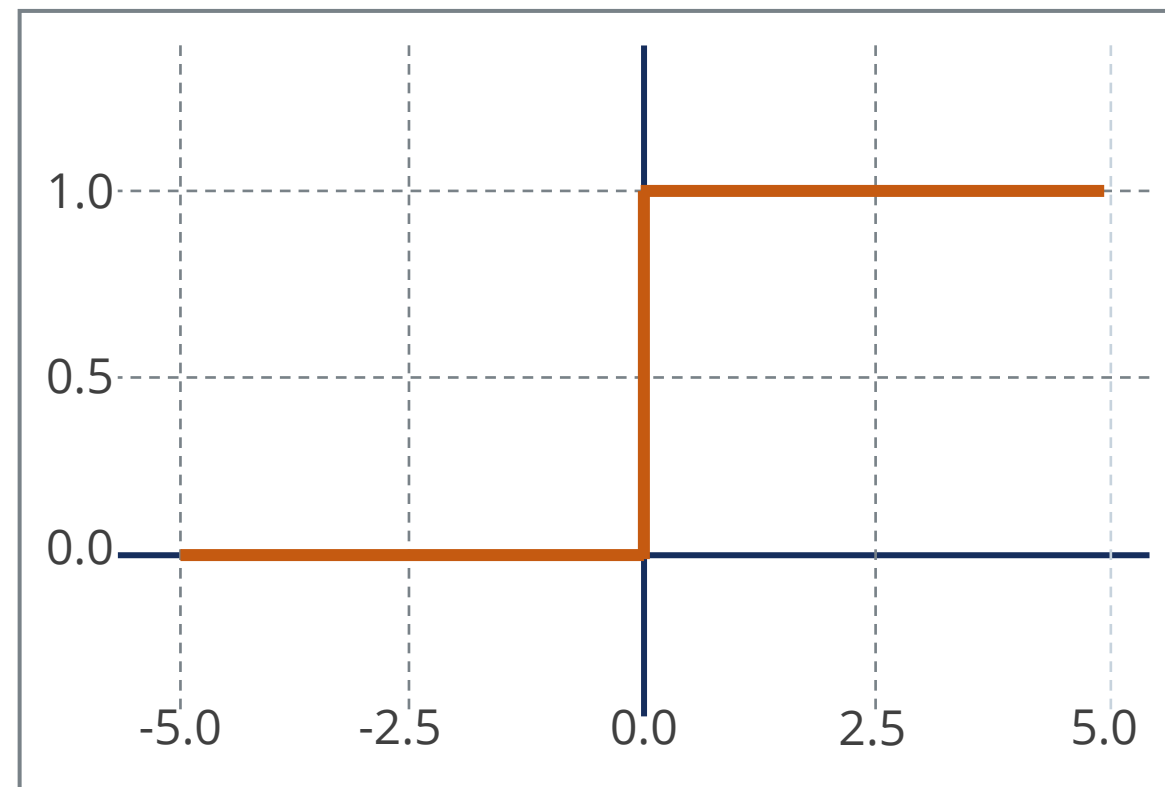# Activation Functions: Step Function

Consider a perceptron:
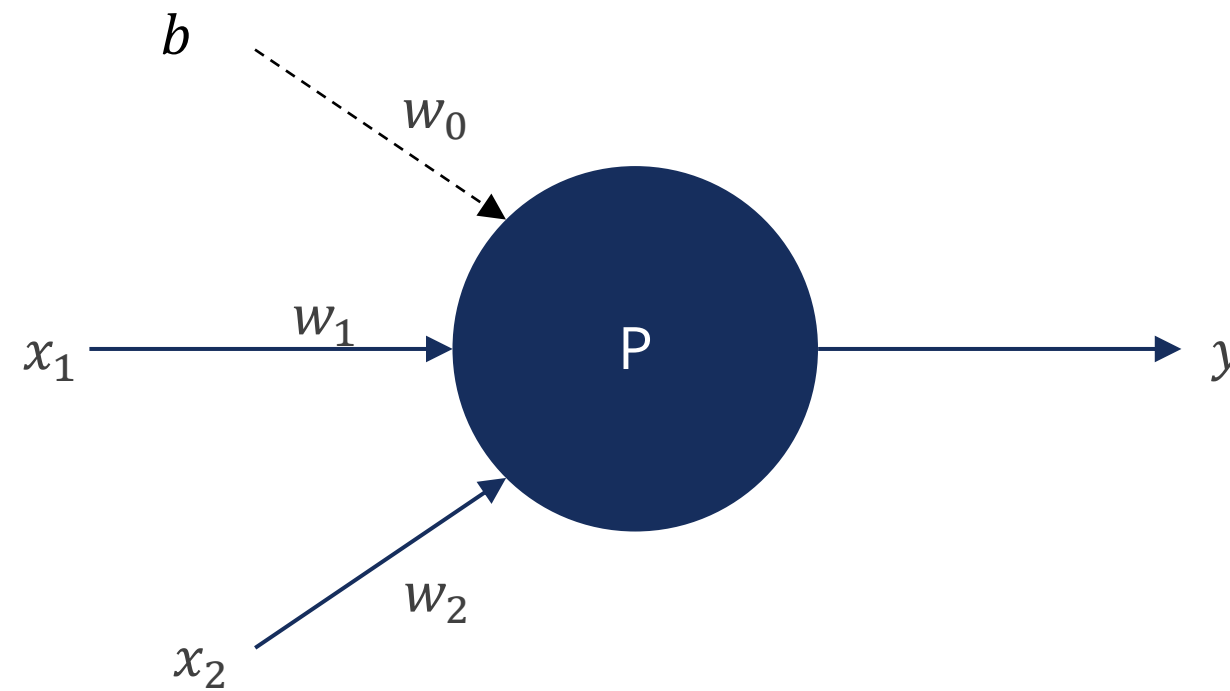
y = 1, when the sum of weighted inputs ($x_1w_1 + x_2w_2 + bw_0$) was greater than 0, else y = 0.
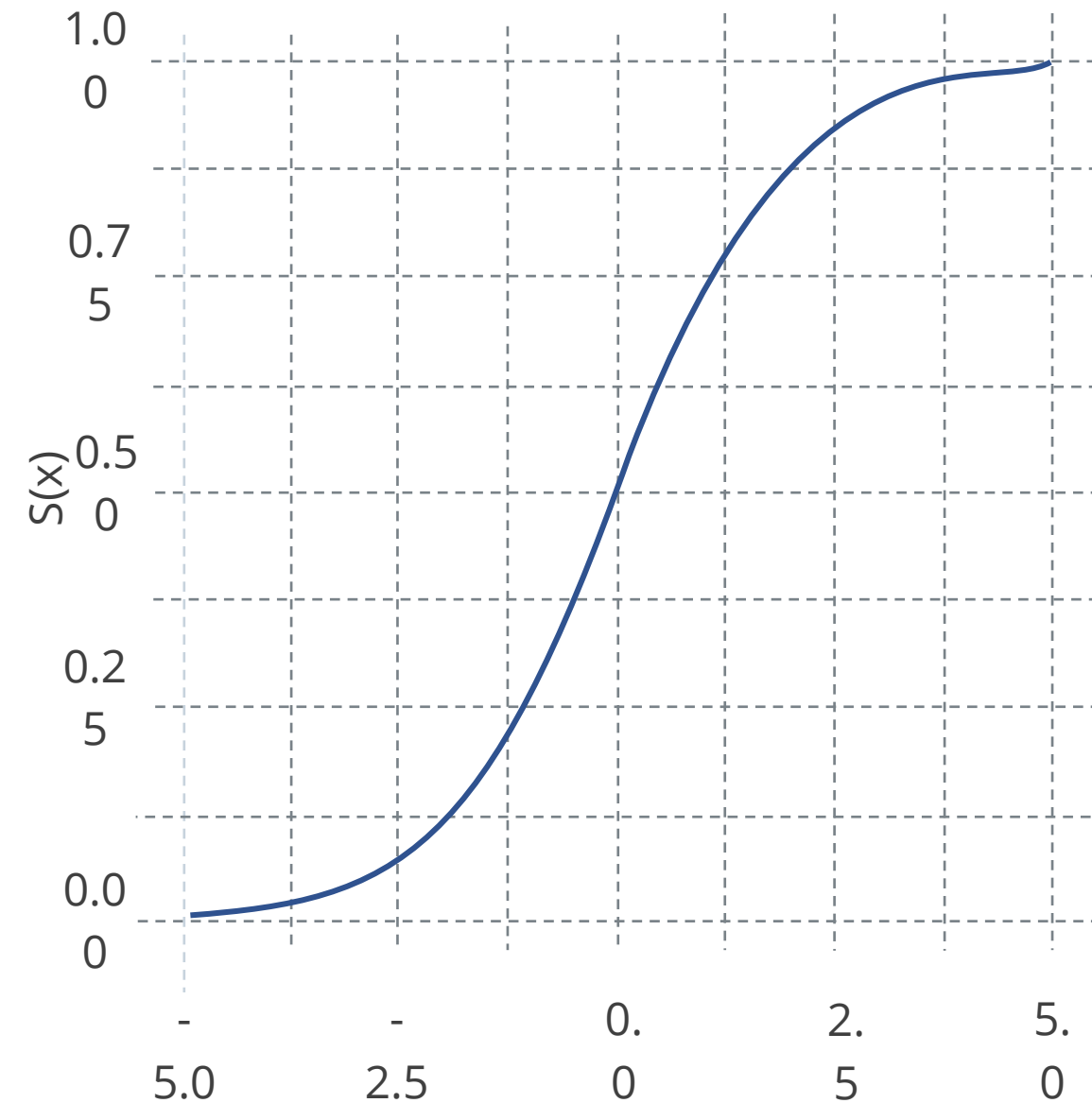
Activation Functions: Step Function

A perceptron gets activated whenever the sum of weighted inputs is non-zero and positive.

This is because the perceptron uses an activation called step function.

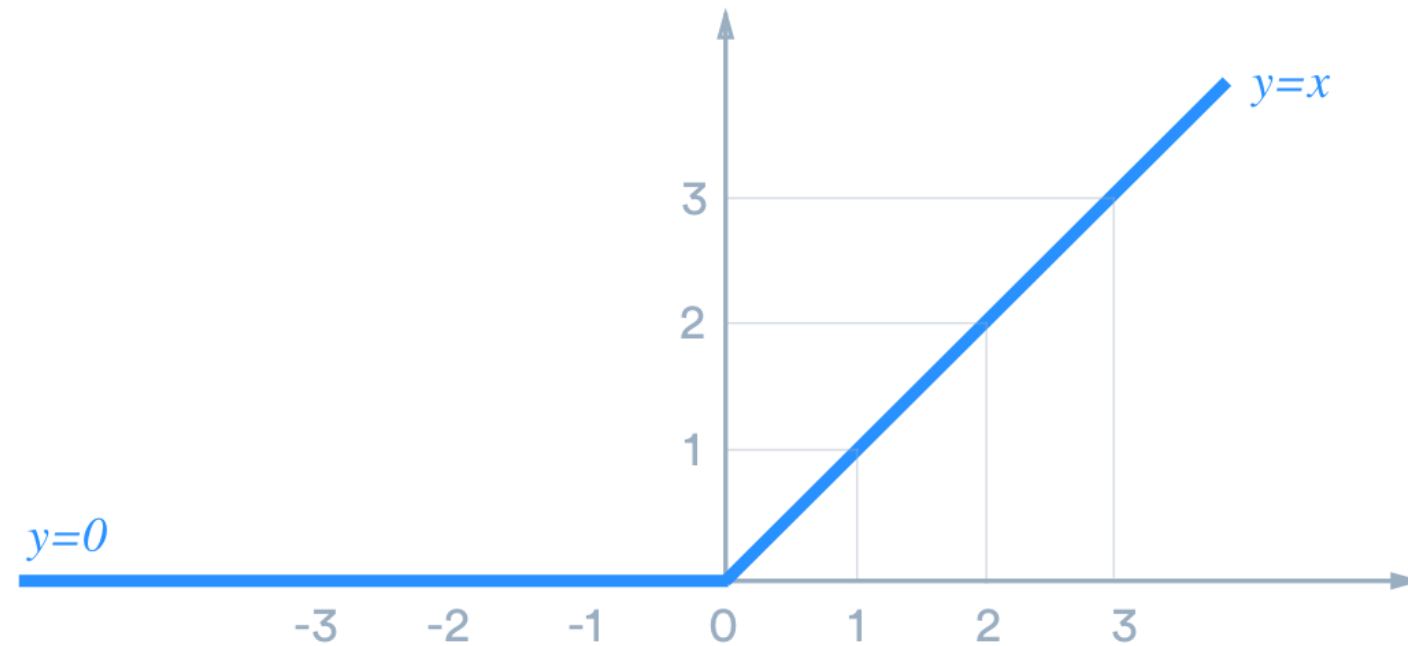# Activation Functions: Step Function

It maps the entire number line into a small range, such as between 0 and 1.

Activation Functions: Rectified Linear Unit (ReLU)

It is the most widely used activation function and is defined as:

$$f(x) = \max(0, x)$$

If the value of input to a neuron is zero or less, ReLU assigns 0 as the output value.

If not, the output value is equal to the input.

ReLU vs. Sigmoid Function

ReLU has an upper hand over the sigmoid function.

Sigmoid activation function	ReLU
When used in deep learning, it suffers from a vanishing gradient.	The vanishing gradient problem is prominent on the negative side of the function.
The weights of the neural network do not get updated.	It is a widely used activation function in neural networks.
It is mathematically complex.	It has a simpler calculation.

Assisted Practices

Let's understand the concept of neural networks and activation function using Jupyter Notebooks.

- 3.06_Configure_Neural_Network_and_Activation_Function

Note: Please refer to the Reference Material section to download the notebook files corresponding to each mentioned topic

Forward Propagation in Perceptron
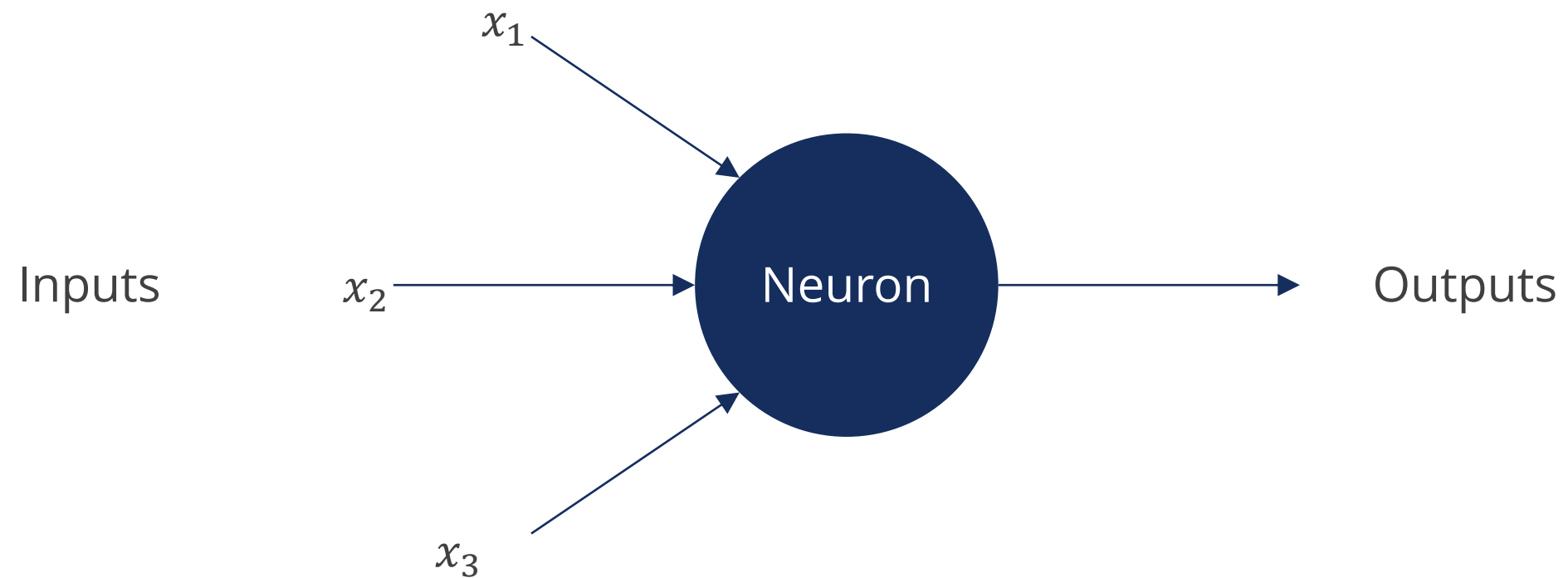
Phases of Perceptron Model

Training a perceptron model involves two phases:

Forward propagation
(Forward pass)

Backward propagation
(Backprop)

Concept of Forward Propagation

A perceptron is a type of feed-forward network.

Inputs
$$x_1$$
$$x_2$$
$$x_3$$

Neuron

Outputs

Here, the process of generating an output flows in one direction, from the input layer to the output layer.

Forward Propagation

In forward propagation, the output of the perceptron model will be:
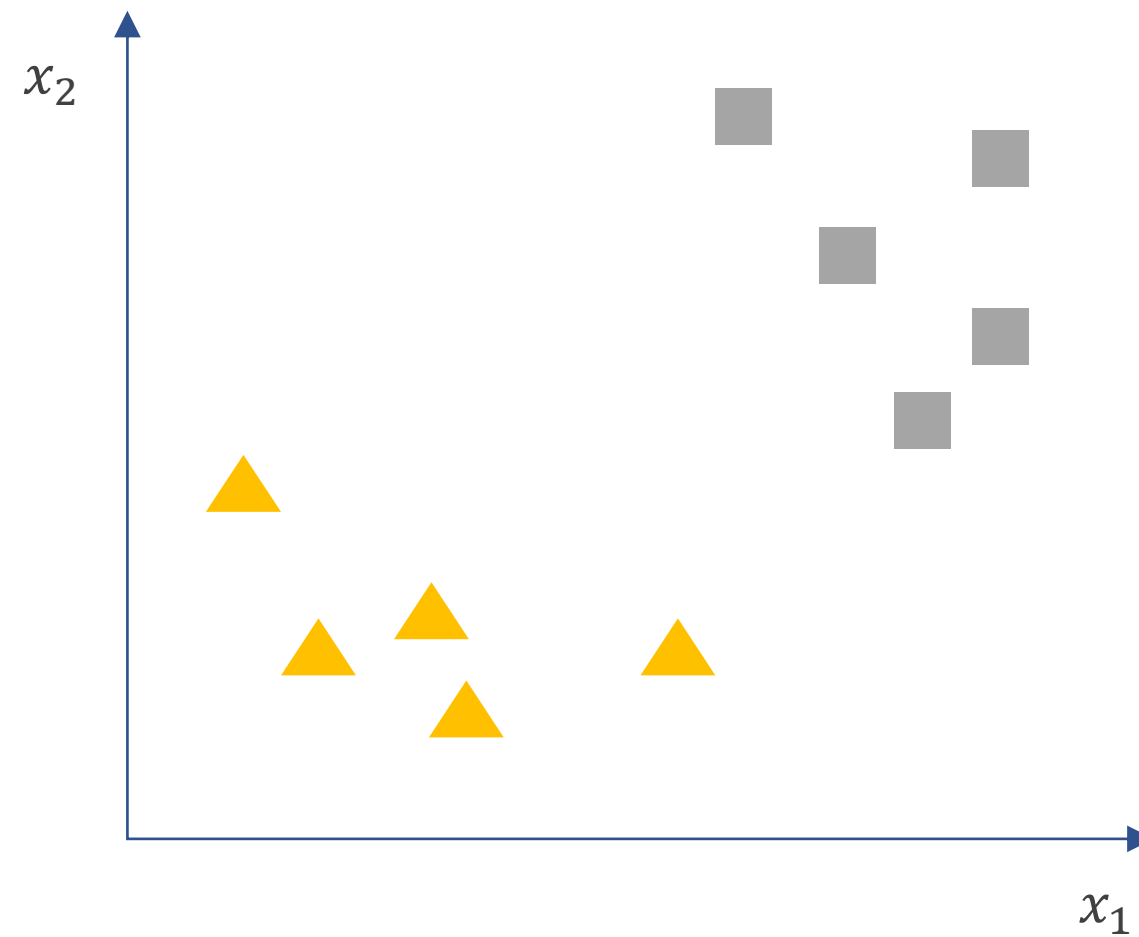
✓ If $x_1w_1 + x_2w_2 + bw_0 > 0$, then y = 1

✓ If $x_1w_1 + x_2w_2 + bw_0 <= 0$, then y = 0

The values of the weights w_1, w_2, and w_3 are randomly initialized, and the objective is to find the right set of weights for the best solution.
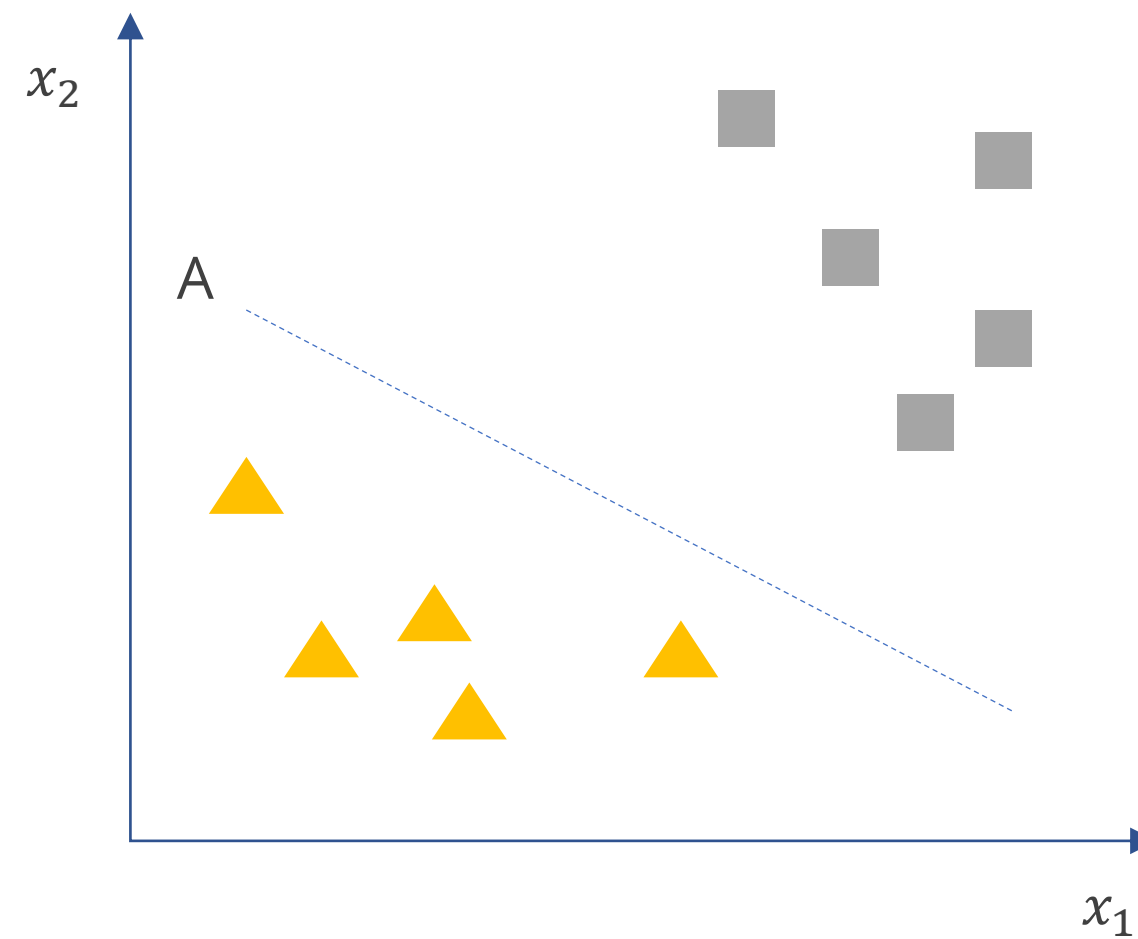
Consider building a binary classification model that should be able to differentiate between triangles and squares.
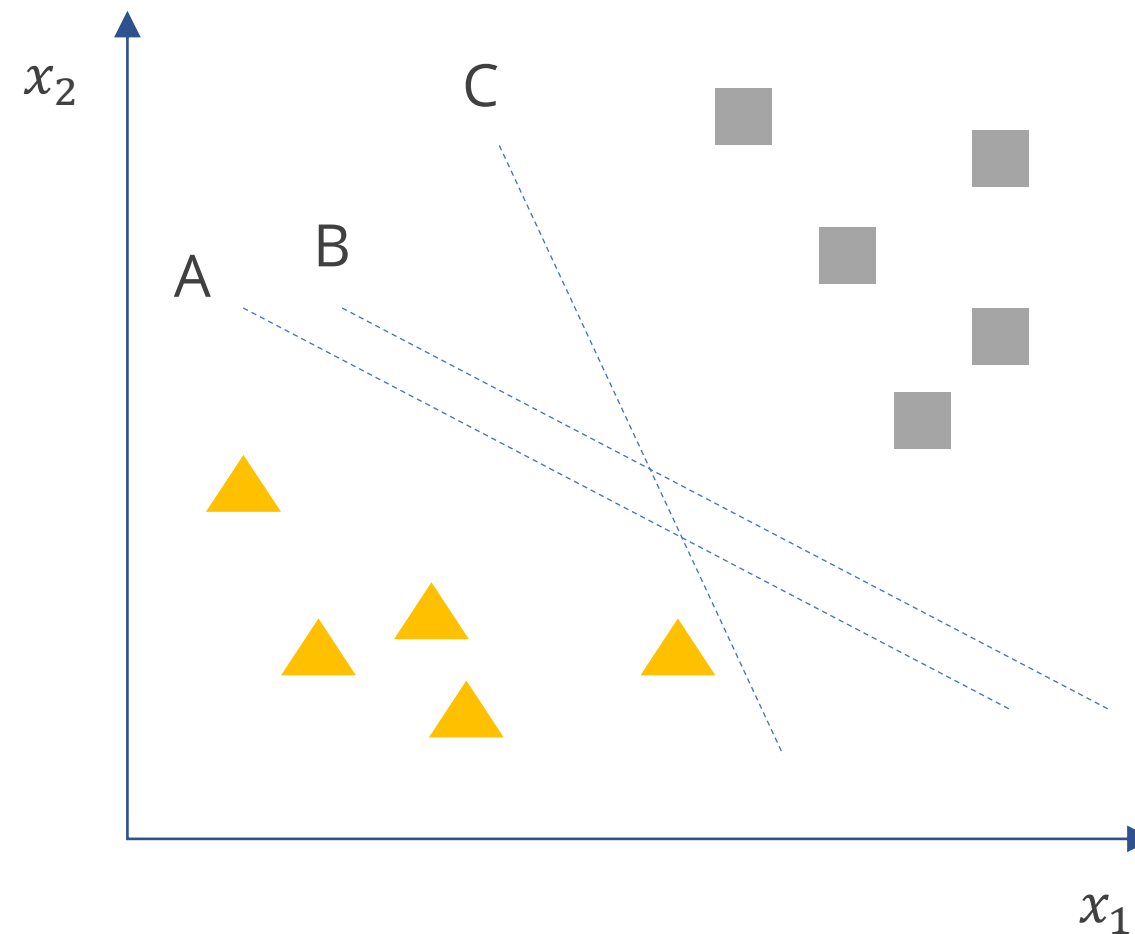
The objective is to find a linear boundary that separates the triangles and squares.

Working of a Forward Propagation

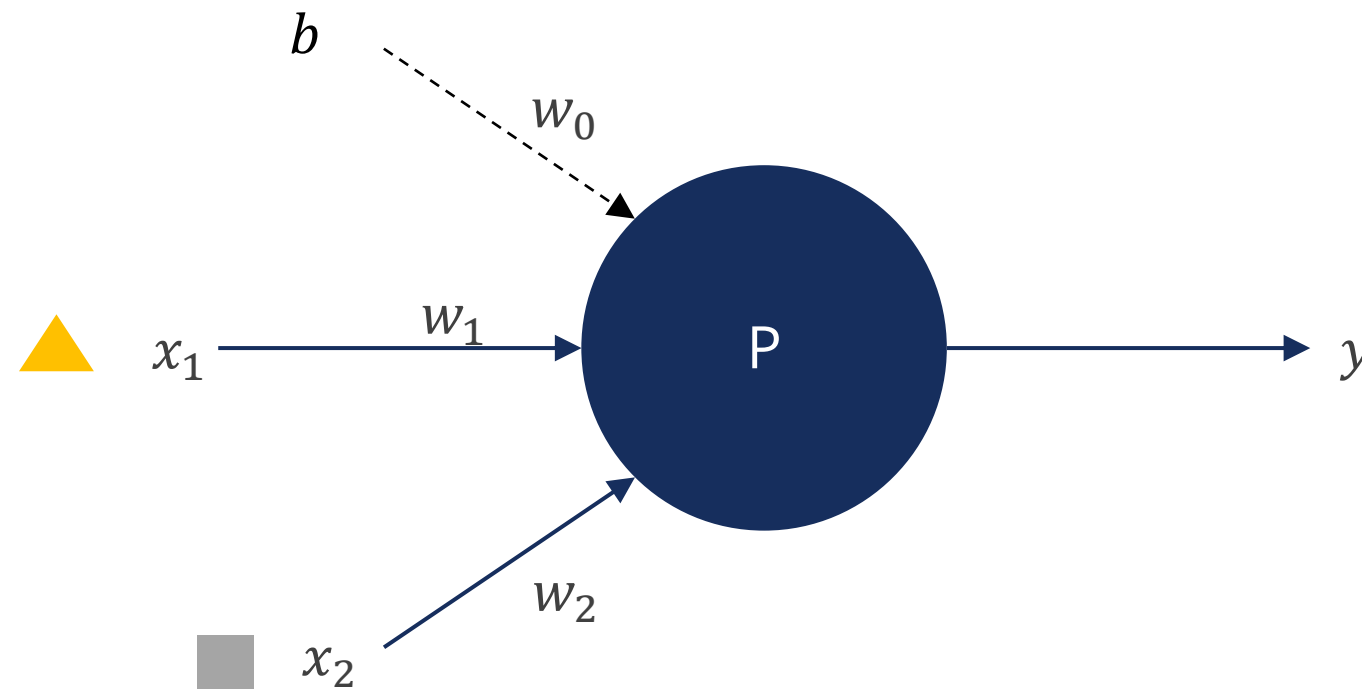The plot has multiple linear decision boundaries like A, B, and C.

Any of these, A, B, C, or any other decision boundary, can be the final solution of the perceptron.

The solution depends on the size of the data used to train the model.
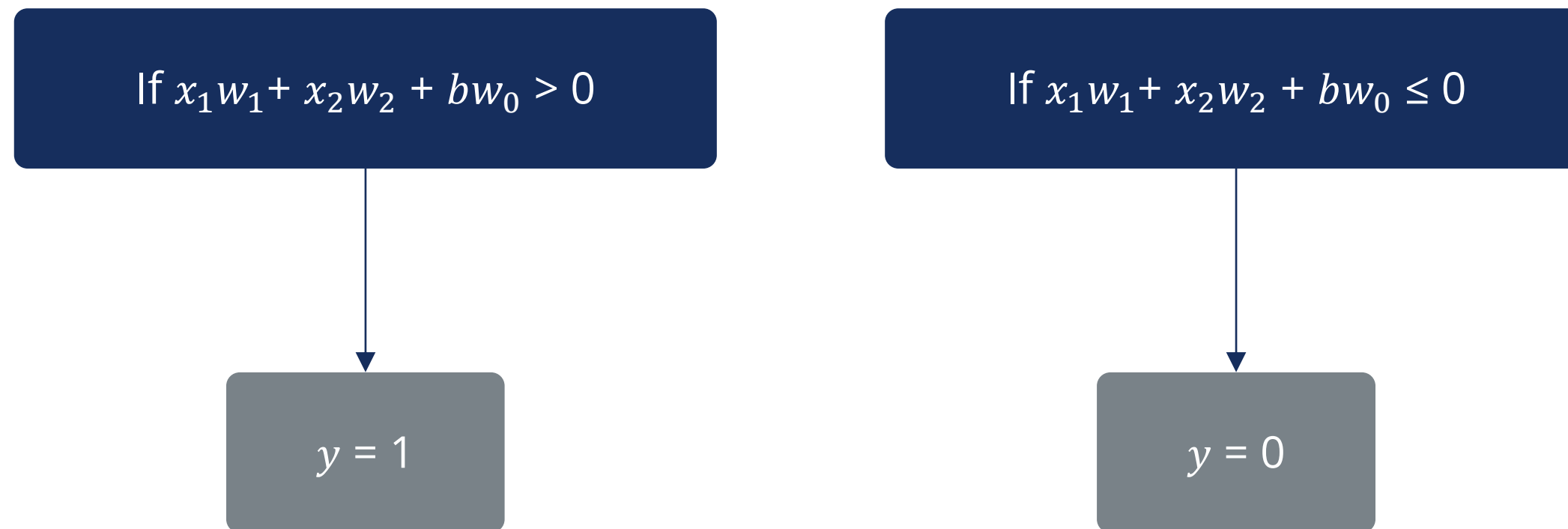
Working of a Forward Propagation

Consider the following two features or inputs:

We get $x_1w_1 + x_2w_2 + bw_0$, where w_1, w_2, and w_0 are randomly initialized weights with values between -1 and 1.

Updated Weight

The weights are updated automatically as the model training progresses.

If $x_1 w_1 + x_2 w_2 + b w_0 > 0$

If $x_1 w_1 + x_2 w_2 + b w_0 \leq 0$

$y = 1$

$y = 0$

The behavior of y being either 1 or 0 is attributed to the activation function used by the perceptron.

Updated Weight

Assume that squares are represented by 1s and triangles by 0s.

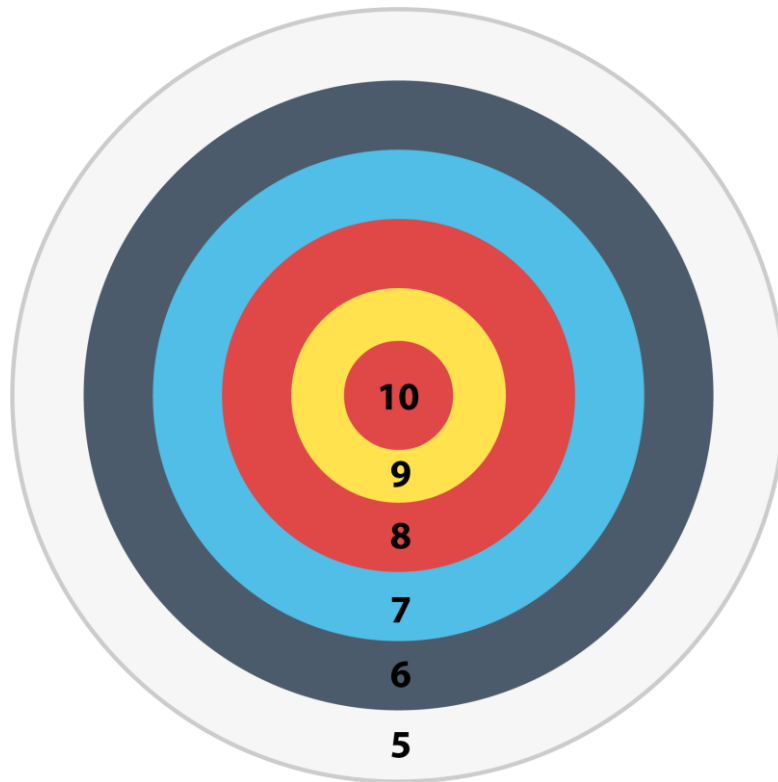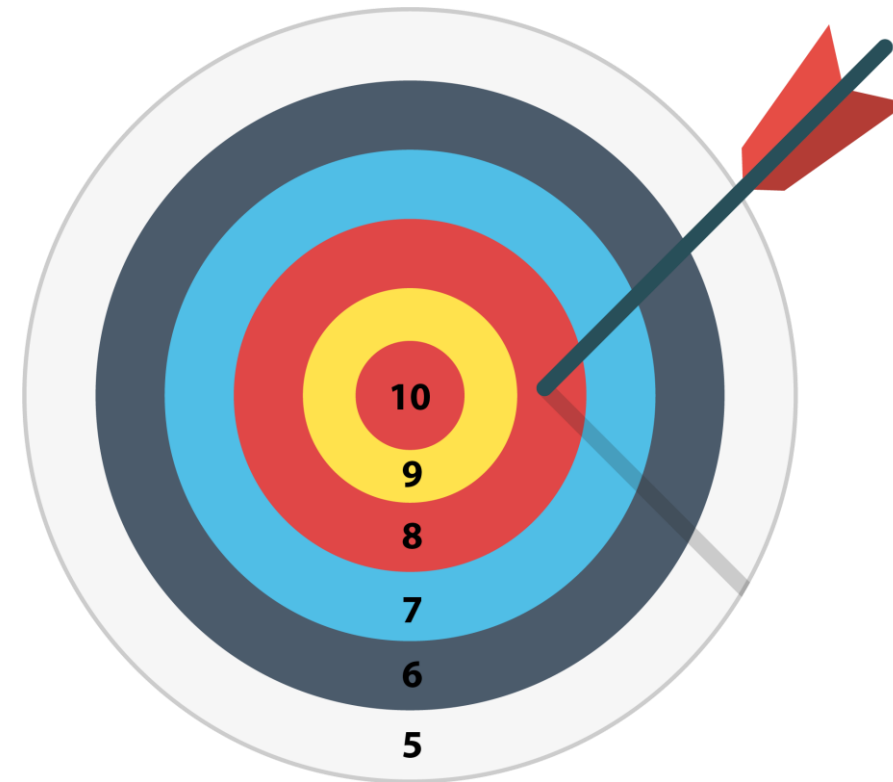| If $y = 1$ | The model predicted the data point as a square. |
| If $y = 0$ | The model predicted the data point as a triangle. |

Loss Function and Cost Function

What Is Loss Function?

In a deep learning model, while predicting, the output deviates from the actual value; the quantitative measure of this difference is called loss. For example;

Here, the predicted value is 10.

The arrow hit the circle at point 8.

Here, the loss will be the actual value minus the predicted value, that is, 10 minus 8 = 2.

Loss Function: Definition and Formula

The loss function, also known as the cost function or objective function, measures the discrepancy between the predicted outputs of a machine learning model and the true values of the training data.

$$Loss\ Function\ (MSE) = \frac{1}{n}\sum_{i=0}^{n}(y_{true} - yipred)^2$$

- n is the number of data points in the training set.

- y_{true} represents the true values of the target variable.

- y_{ipred} represents the predicted values of the target variable by the model.

What Is Cost Function?

It aggregates the difference for the entire training dataset.
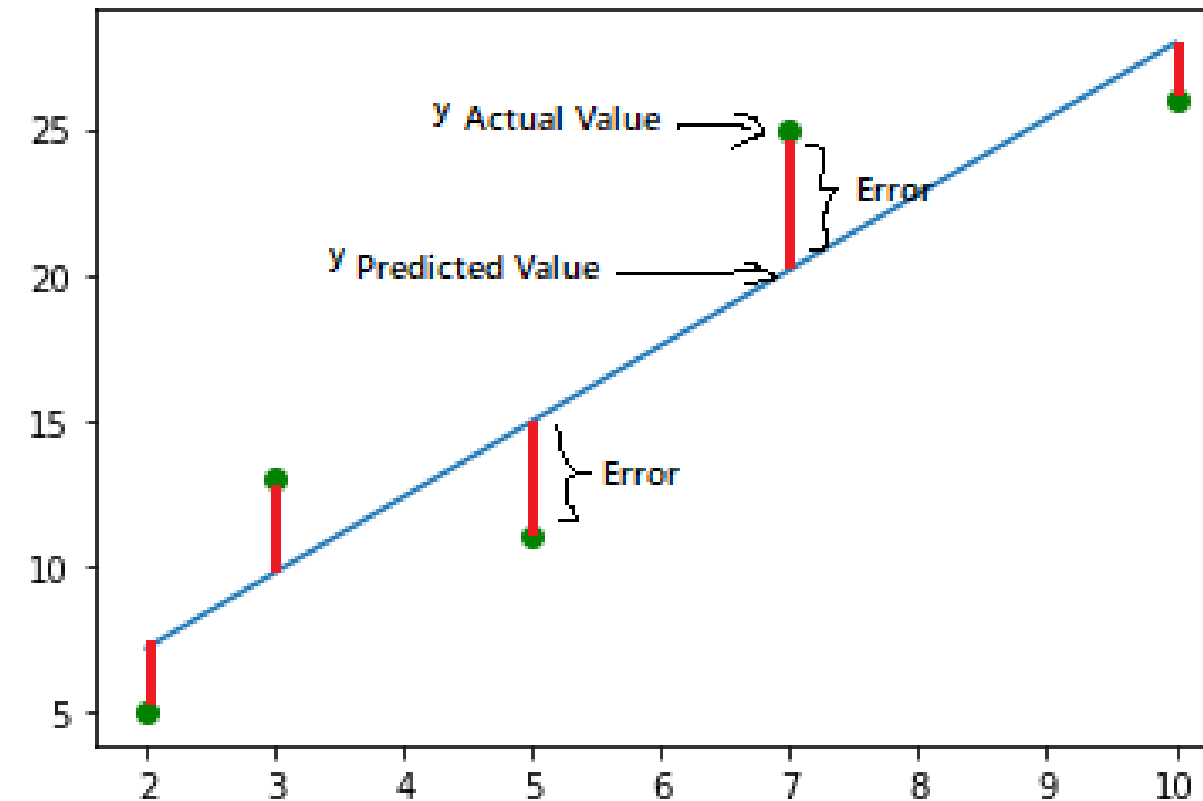
Calculating how much the model mispredicted the results uses both the predicted and actual results. It outputs a higher number if our predictions differ a lot from the actual values.

Need of Cost Function

The cost function is needed to utilize gradient descent, and it should be minimized as much as possible by changing the values of the parameters.

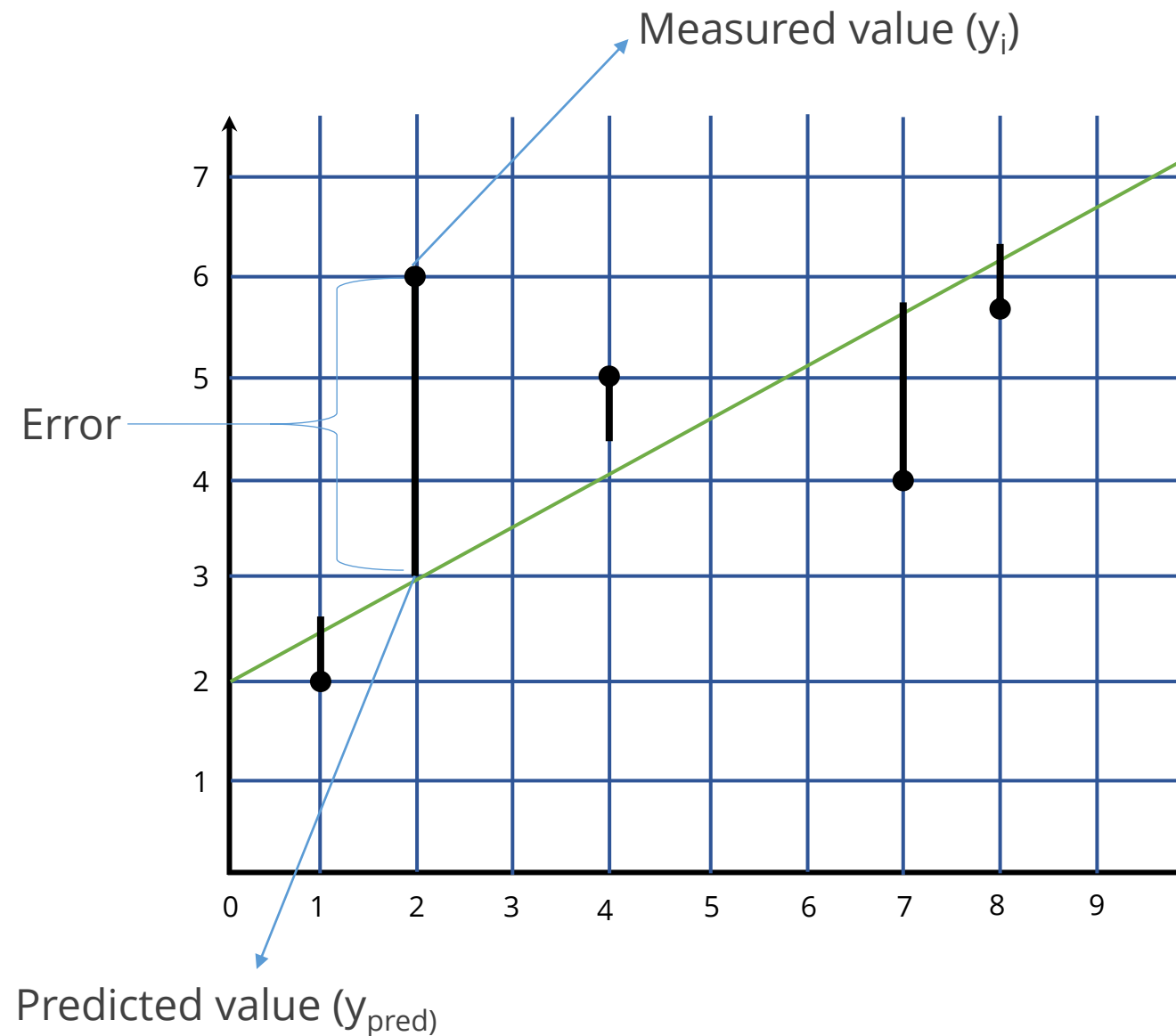$$Cost\ Function\ (MSE) = \frac{1}{n}\sum_{i=0}^{n}(y_i - yipred)^2$$

Replace yipred with the linear regression:
$$mxi + c$$

$$Cost\ Function\ (MSE) = \frac{1}{n}\sum_{i=0}^{n}\left(y_i - (mx_i + c)\right)^2$$

In linear regression, MSE is commonly used as the cost function or loss function.

Need of Cost Function

Here, y_i represents the ground truth values, and y_{pred} represents the predicted or estimated values.


Gradient descent optimizes linear regression by iteratively adjusting parameters (slope and y-intercept) to find the best-fit line when an exhaustive search is impractical.

Backpropagation in Perceptron

Learning Networks

- The network learns from input data or training examples to generalize and acquire knowledge.

- By adjusting weights and biases, it aims to find a line, plane, or hyperplane that can accurately separate different classes.

- The network configures itself through training to effectively solve the problem at hand.
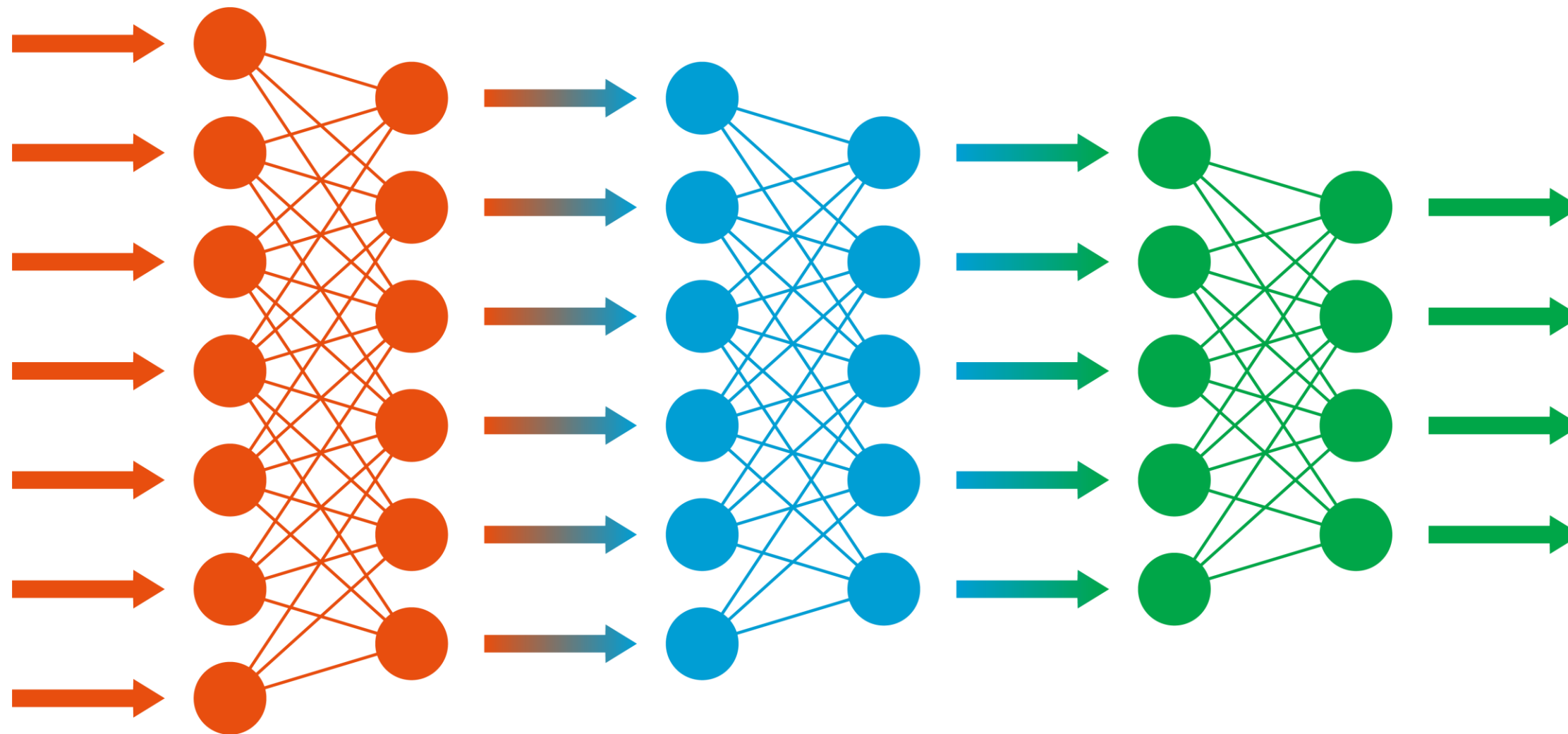
The Backpropagation Algorithm

The following steps outline the backpropagation process within the context of training a neural network.

| Initialize the weights and the threshold	Provide the input and calculate the output	Update the weights	Repeat the initial steps

Let W_i be the initial weight

Let x be the input and y be the output

$W_i (t + 1) = (W_i)t + n(d - y)x$,where d is the desired output, n is the learning rate, and y is the actual output

The former steps are iterated continuously by changing the values of n till a considerable output is obtained

The Error Landscape

The objective is to minimize the loss or sum squared error (SSE).

SSE:
$$S(t_i - z_i)^2$$

Error Landscape

0
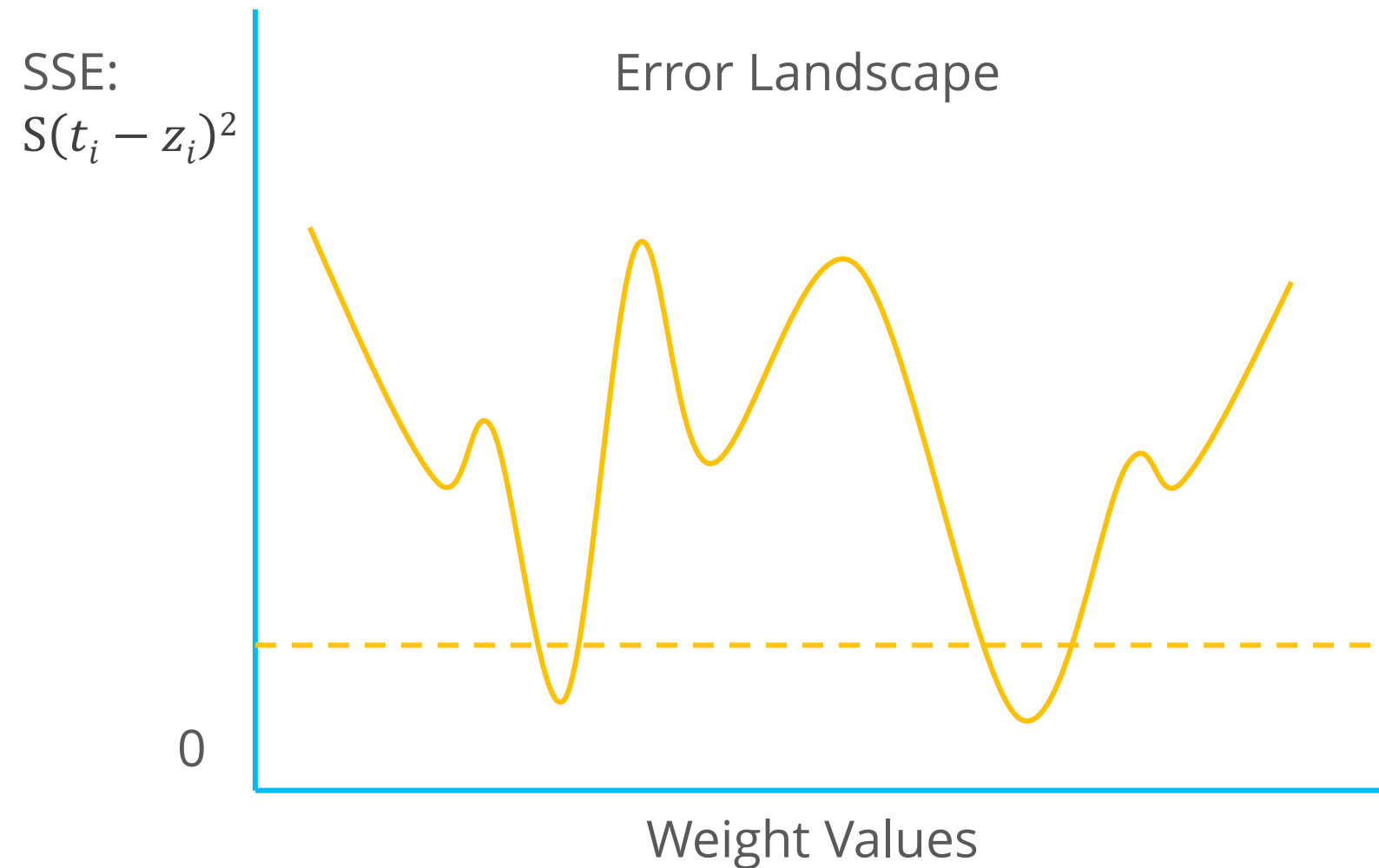
Weight Values

$S(t_i - z_i)^2$ represents the sum of squared differences between the target values (t) and the predicted values (z) in a dataset.

Backpropagation

Once the model receives the sum of the weighted inputs, it is passed through the activation function, which gives 0 or 1 as the final output of the perceptron model for the given inputs.

Prediction	Actual	Error
1	1	0
1	0	1

The output value is compared with the ground truth value, and an error is computed.

Backpropagation

To minimize the error, the neural network traverses back to change the weights of the input neurons.

Now, w_0, w_1, and w_2 have different weights.

The new predictions with different weights are compared with the ground truth to check the error.

This process continues until the error cannot be reduced further.

This is the concept of backpropagation.

Backpropagation

If there are 1000 sets of inputs in the training set, there will be 1000 iterations of:

Forward propagation

Backpropagation (Backprop)

Weights are updated in each iteration.

Backpropagation

Weights are updated in the following manner in each iteration:

$$w_0 = w_0 + α \cdot error$$

$$w_1 = w_1 + α \cdot error \cdot x_1$$

$$w_2 = w_2 + α \cdot error \cdot x_2$$

- α = Learning rate, ranging from 0 to 1
- Error = Difference between the desired output and actual output
- W_0, w_1, and w_2 = Weight parameters
- x_1 and x_2 = Input variables

Backpropagation takes more time than simpler algorithms, but it is more efficient in terms of its ability to effectively minimize error and improve the accuracy of the network's output.

Vanishing Gradient

In a feedforward network (FFN), the backpropagated error signal typically decreases or increases exponentially as a function of the distance from the final layer.
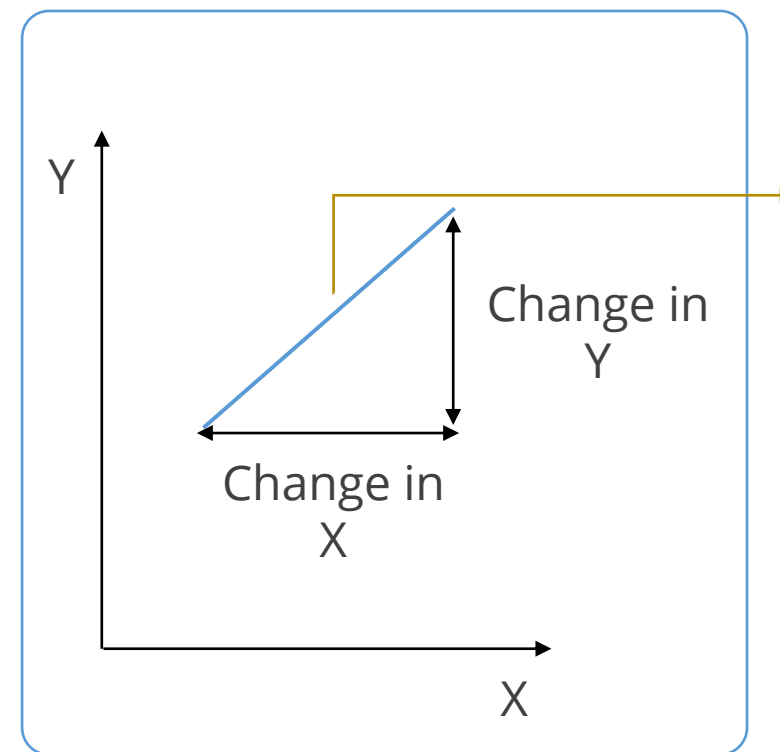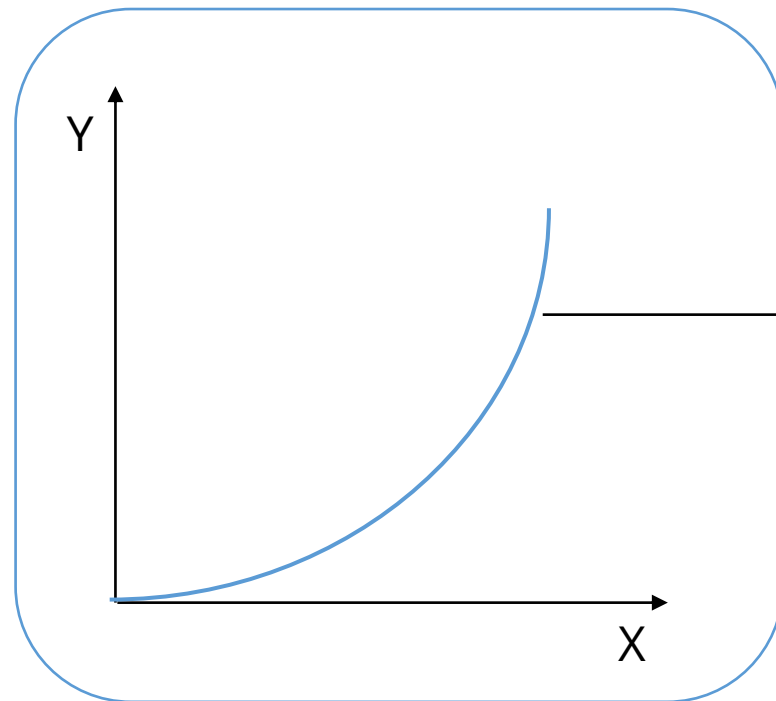

Slope decreases gradually to a very small value and makes training difficult

A small gradient or a gradient that approaches zero can pose challenges during training of machine learning models.
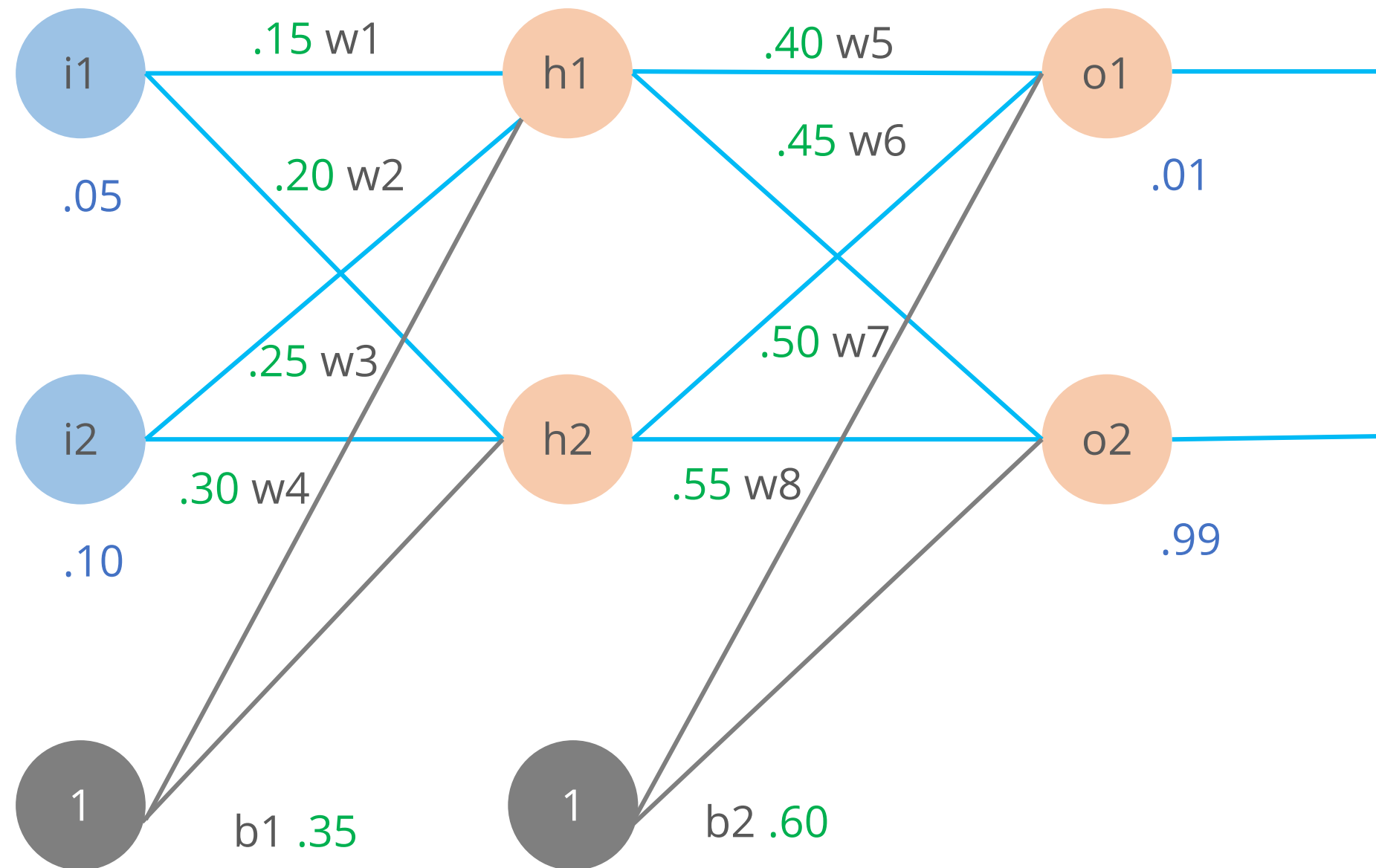
Exploding Gradient

Slope grows
exponentially

It is a problem that occurs when large error gradients accumulate and result in very large updates to neural network model weights during training.

Backpropagation: Example

A Feed Forward Network

In order to have some numbers to work with, here are the initial weights, the biases, and the training inputs/outputs:

Forward Pass

The forward pass computes the output of each neuron by applying weights to the input data.

Calculate the total net input for $h1$:

$$net_{h1} = w_1 * i_1 + w_2 * i_2 + b_1 * 1$$

Using the given weights and inputs:

$$net_{h1} = 0.15 * 0.05 + 0.2 * 0.1 + 0.35 * 1 = 0.3775$$

Therefore, the total net input for $h1$ is 0.3775

Forward Pass

Squash it using the logistic function to get the output for $h1$:

$$out_{h1} = 1/1 + e^{-net_{h1}} = 0.593269992$$

$$out_{h2} = 0.596884378$$

Carrying out the same process for h_2, we get: 0.596884378

Forward Pass

Repeat the process for the output layer neurons using the output from the hidden layer neurons as inputs.

$$net_{o1} = w_5 * out_{h1} {}_+ w_6 * out_{h2} + b_2 * 1$$
$$net_{o1} = 0.4 * 0.593269992 + 0.45 * 0.596884378 + 0.6 * 1 = 1.105905967$$
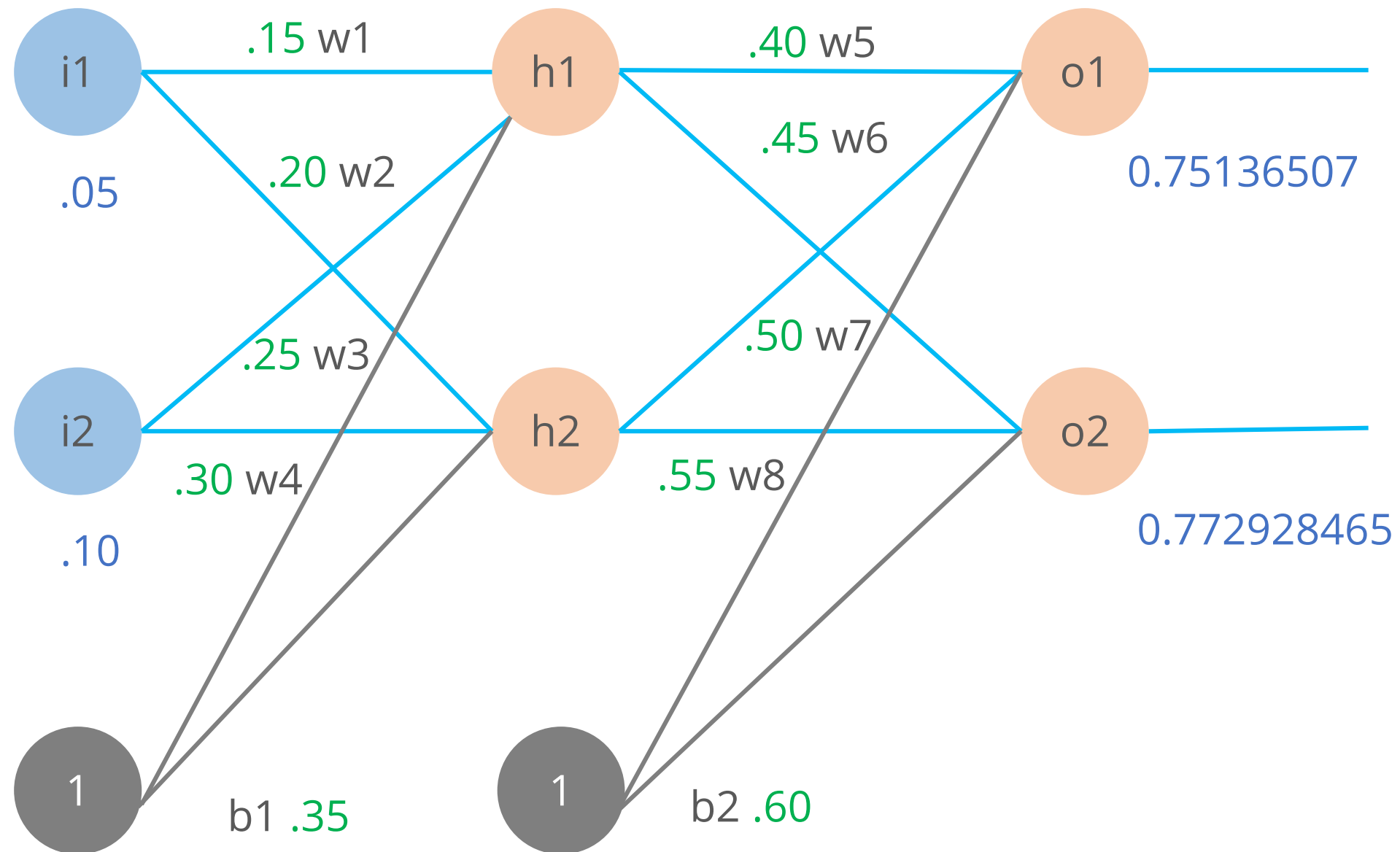
$$out_{o1} {}_= 1/1 + e^{-net}{}_{o1} = 0.75136507$$

Output for o1

Carrying out the same process for o2, we get:

$$out_{o2} = 0.772928465$$

Forward Pass

The values of output 1 and output 2 have been added to the diagram.

Calculating Total Error

Calculate the error for each output neuron using the squared error function and sum them to get the total error:

$$E_{Total} = \sum 1/2 \ (target - output)^2$$

For example, the target output for o_1 is 0.01 *but the neural network output is* 0.75136507, *therefore its error is*:
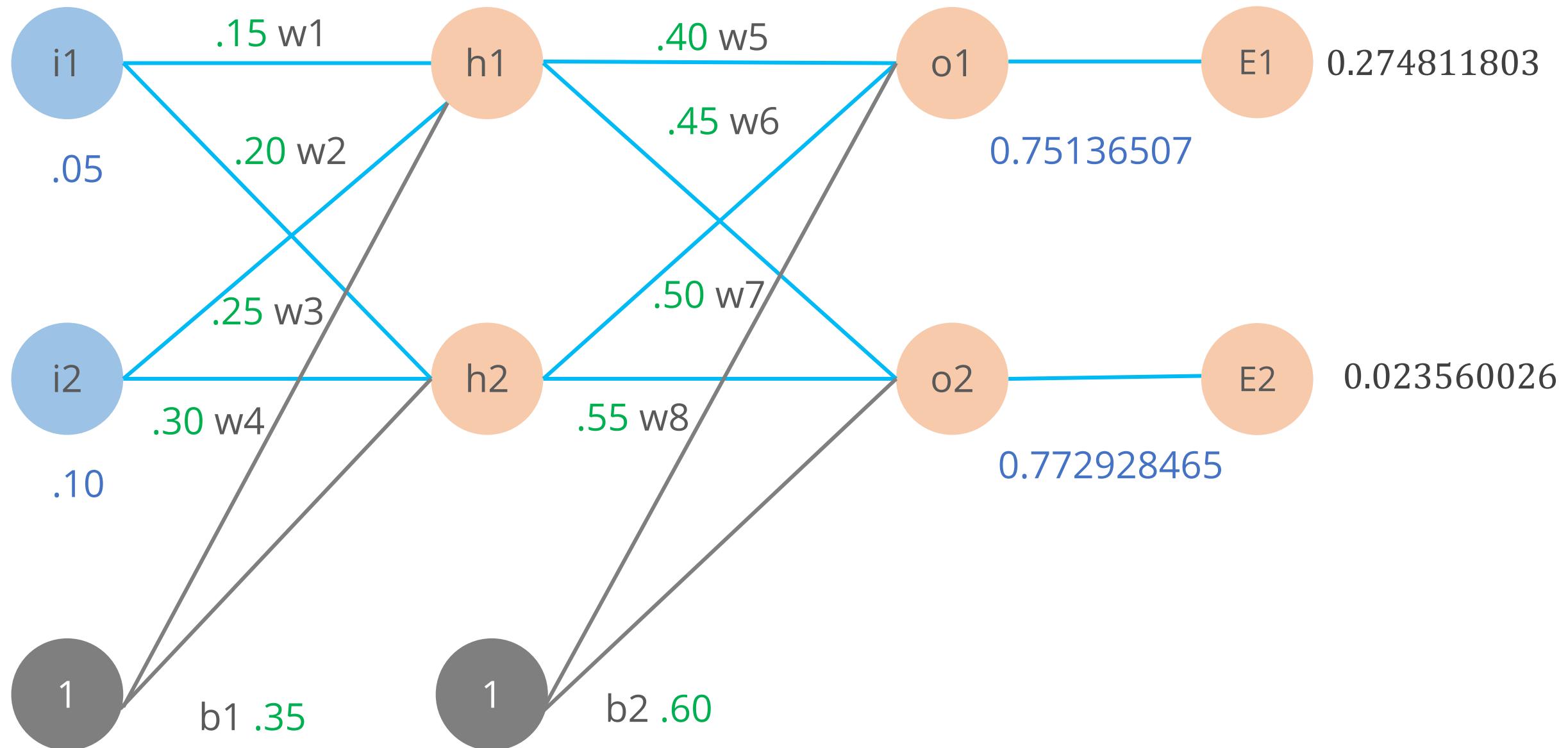
$$E_{o1} = 1/2 \ (target_{o1} \_ out_{o1})^2 = 1/2(0.01 - 0.753136507)^2 = 0.274811803$$

$$E_{o2} = 0.023560026$$

Calculating Total Error

The total error for the neural network is the sum of these errors:

$$E_{Total} = E_{o1} + E_{o2} = 0.274811803 + 0.023560026 = 0.298371109$$

Backward Pass

It computes the gradients of the parameters to optimize the neural network.

Let us consider W_5 and see how much it affects the total error.

Calculate the derivative of the total error $(ETotal)$ with respect to the weight parameter $w5$.

$$\frac{\partial E_{Total}}{\partial w_5} = \frac{\partial E_{Total}}{\partial out_{01}} * \frac{\partial out_{01}}{\partial net_{01}} * \frac{\partial net_{01}}{\partial w_5}$$

$$\delta 01 = \frac{\partial E_{Total}}{\partial out_{01}} * \frac{\partial out_{01}}{\partial net_{01}} = \frac{\partial E_{Total}}{\partial net_{01}}$$

The goal of backpropagation is to update the weights to minimize the error between the actual and target outputs, improving the network's overall performance.

Backward Pass

By applying the chain rule, we know that:

Error calculation:

$$E_{Total} = E_{o1} + E_{o2}$$

$$E_{o1} = \frac{1}{2}(target_{o1} - outo_1)^2$$

Activation function:

$$out_{o1} = 1/1 + e^{-net_{o1}}$$

Net input calculation:

$$net_{o1} = w_5 * out_{h1} + w_6 * out_{h2} + b_2 * 1$$

Backward Pass

Figure out each piece in this equation. First, how much does the total error change with respect to the output?

$$\frac{\partial E_{Total}}{\partial w_5} = \frac{\partial E_{Total}}{\partial out_{01}} * \frac{\partial out_{01}}{\partial net_{01}} * \frac{\partial net_{01}}{\partial w_5}$$

$$E_{Total} = 1/2\ (target_{o1}\_out_{01})^2 + 1/2\ (target_{o2}\_out_{02})^2$$

Now, let's calculate the partial derivative $\partial E_{Total}/\partial out_{01}$

$$\frac{\partial E_{Total}}{\partial out_{01}} = 2 * 1/2(target_{o1}\_out_{01})^2 - 1 * -1 + 0$$

$$\frac{\partial E_{Total}}{\partial out_{01}} = -(target_{01} - out_{01}) = -(0.01 - 0.75136507) = 0.74136507$$

-(target – out) is sometimes expressed as **out - target**

Backward Pass

Next, how much does the output of o_1 change with respect to its total net input?

$$\frac{\partial E_{Total}}{\partial w_5} = \frac{\partial E_{Total}}{\partial out_{01}} * \frac{\partial out_{01}}{\partial net_{01}} * \frac{\partial net_{01}}{\partial w_5}$$

$$out_{01} = 1/1 + e^{-net_{01}}$$

we can calculate $\partial out_{01}/\partial net_{01}$:

$$\frac{\partial out_{01}}{\partial net_{01}} = out_{01}(1 - out_{01})$$

$$f(x) = 1/1 + e^{-x} = e^{x/1} + ex$$

$$\frac{df(x)}{dx} = f(x)(1 - (f(x)))$$

$$\frac{\partial out_{01}}{\partial net_{01}} = 0.75136507(1 - 0.75136507) = 0.186815602$$

Backward Pass

Finally, how much does the total net input of o_1 change with respect to w_5?

$$net_{01} = w_5 * outh_1 + w_6 * outh_2 + b_2 * 1$$

we can calculate $\partial net_{01} / \partial w_5$:

$$\frac{\partial net_{01}}{\partial w_5} = 1 * outh_1 * w_5^{(1-1)} + 0 + 0 = outh_1 = 0.593269992$$

Putting it all together:

$$\frac{\partial E_{Total}}{\partial w_5} = \frac{\partial E_{Total}}{\partial out_{01}} * \frac{\partial out_{01}}{\partial net_{01}} * \frac{\partial net_{01}}{\partial w_5}$$

$$\frac{\partial E_{Total}}{\partial w_5} = 0.74136507 * 0.186815602 * 0.593269992 = 0.082167041$$

Updated Weight

$$w_5^+ = w_5 - \eta * \frac{\partial E_{Total}}{\partial w_5} = 0.4 - 0.5 * 0.082167041 = 0.35891648$$

Similarly, $w6$, $w7$, and $w8$ are provided with their respective values:
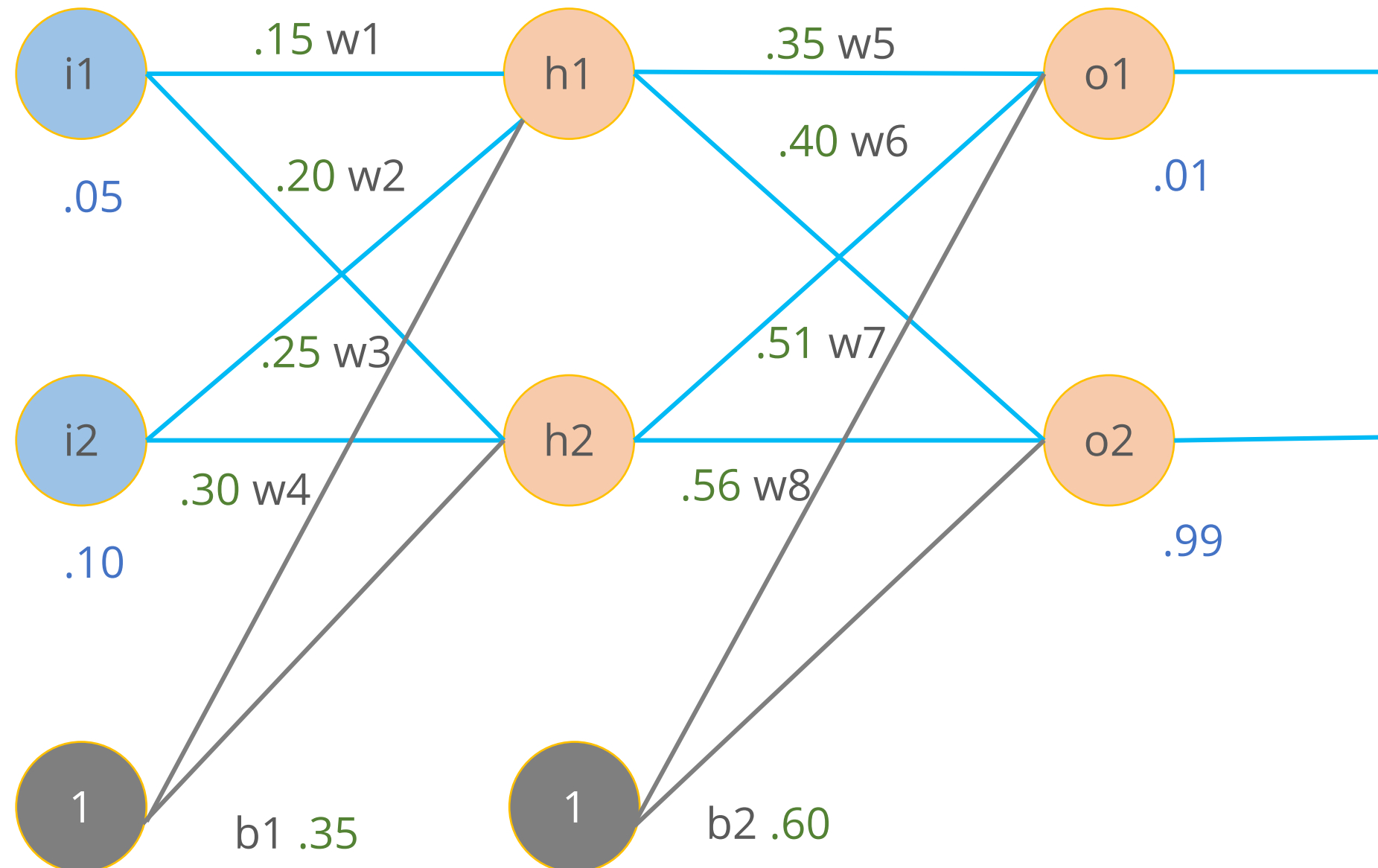
$$w_6^+ = 0.408666186$$

$$w_7^+ = 0.511301270$$

$$w_8^+ = 0.561370121$$

Here, $w5$ is being updated using the learning rate (η) multiplied by the derivative of the total energy with respect to $w5$.
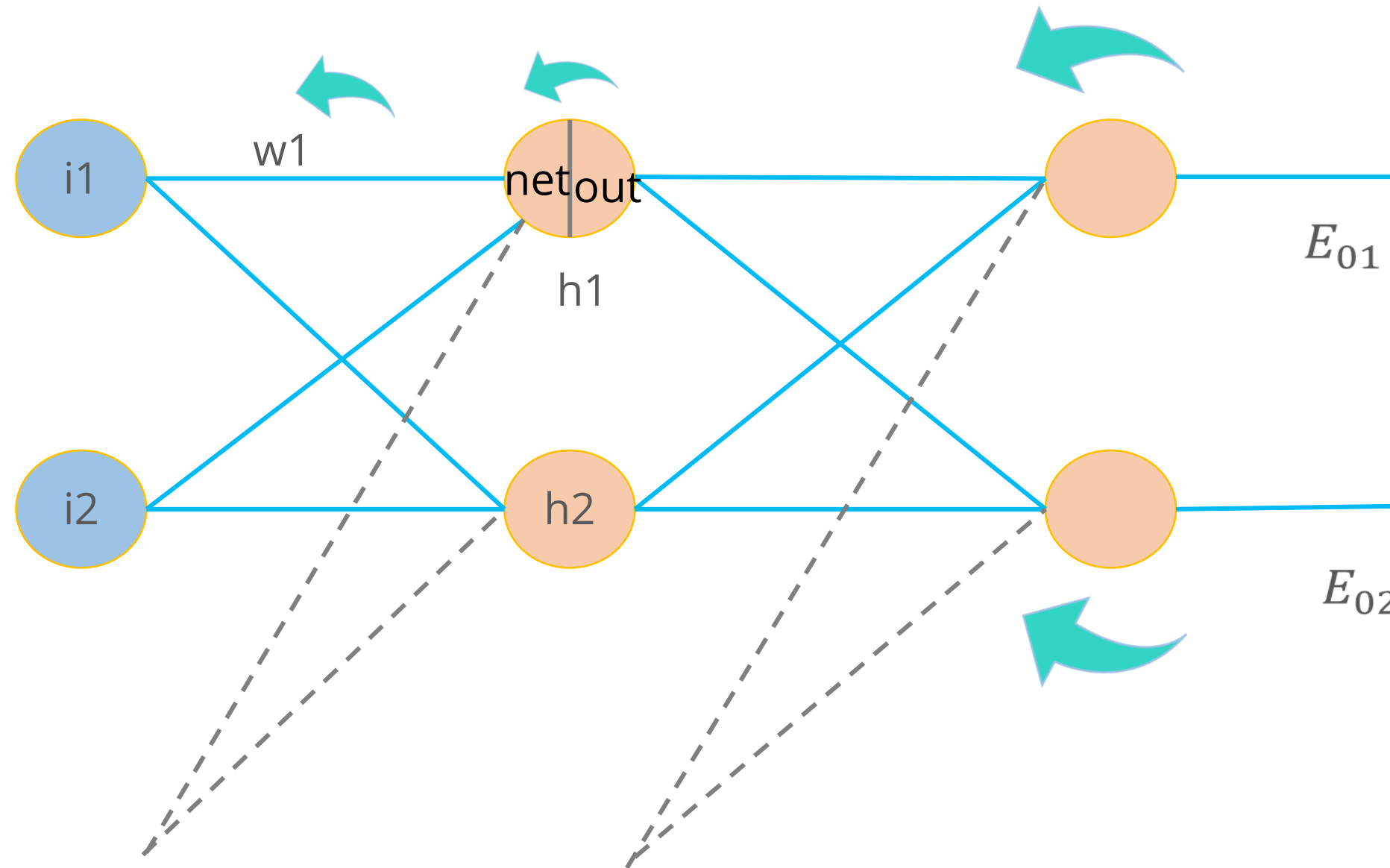
Updated Weight

The values of output 1 and output 2 have been added to the diagram.

Hidden Layer Weight Assignment

Next, continue the backwards pass by calculating new values for w1, w2, w3, and w4.

$$\frac{\partial E_{Total}}{\partial w_1} = \frac{\partial E_{Total}}{\partial out_{h1}} * \frac{\partial out_{h1}}{\partial net_{h1}} * \frac{\partial net_{h1}}{\partial w_1}$$

Hidden Layer Weight Assignment

We know that out_{h1} affects both out_{o1} and out_{o2}. Therefore, $\frac{\partial E_{Total}}{\partial out_{h1}}$ needs to take into consideration its effect on both output neurons:

$$\frac{\partial E_{Total}}{\partial out_{h1}} = \frac{\partial E_{01}}{\partial out_{h1}} * \frac{\partial E_{02}}{\partial out_{h1}}$$

The total energy E_{01} is given by:

$$E_{01} = 1/2\ (target_{o1} \_ out_{01})^2$$

$$out_{01} = 1/1 + e^{-\ net_{01}}$$

$$net_{01} = w_5 * outh_1 + w_6 * outh_2 + b_2 * 1$$

we can calculate $\partial E_{Total} / \partial out_{h1}$:

$$\frac{\partial E_{Total}}{\partial out_{h1}} = \frac{\partial E_{01}}{\partial net_{01}} * \frac{\partial net_{01}}{\partial out_{h1}}$$

$$\frac{\partial E_{01}}{\partial net_{01}} = \frac{\partial E_{01}}{\partial out_{01}} * \frac{\partial out_{01}}{\partial net_{01}} = \frac{\partial net_{01}}{\partial out_{h1}} = w_5$$

Therefore, $\partial E_{01} / \partial net_{01}$ is equal to w_5.

Hidden Layer Weight Assignment

Plugging them in and following same process for $\frac{\partial E_{02}}{\partial out_{h1}}$, we get:

$$\frac{\partial E_{01}}{\partial out_{h1}} = \frac{\partial E_{01}}{\partial net_{01}} * \frac{\partial net_{01}}{\partial out_{h1}} = 0.138498562 * 0.40 = 0.055399425$$

$$\frac{\partial E_{02}}{\partial out_{h1}} = -0.019049119$$

$$\frac{\partial E_{Total}}{\partial out_{h1}} = \frac{\partial E_{01}}{\partial out_{h1}} + \frac{\partial E_{02}}{\partial out_{h1}} = 0.055399425 + -0.019049119 = 0.036350306$$

Hidden Layer Weight Assignment

As $\frac{\partial Etotal}{\partial out_{h1}}$ is present, figure out $\frac{\partial out_{h1}}{\partial net_{h1}}$ and then $\frac{\partial net_{h1}}{\partial w}$ for each weight:

$$\frac{\partial E_{Total}}{\partial w_1} = \frac{\partial E_{Total}}{\partial out_{h1}} * \frac{\partial out_{h1}}{\partial net_{h1}} * \frac{\partial net_{h1}}{\partial w_1}$$

$$out_{h1} = 1/1 + e^{-neth_1}$$

$$\frac{\partial out_{h1}}{\partial net_{h1}} = out_{h1}(1-out_{h1})$$

$$= 0.59326999(1-0.59326999) = 0.241300709$$

$$net_{h1} = w_1 * i_1 + w_3 * i_2 + b_1 * 1$$

$$\frac{\partial net_{h1}}{\partial w_1} = i_1 = 0.05$$

$$\frac{\partial E_{Total}}{\partial w_1} = 0.036350306 * 0.241300709 * 0.05 = 0.000438568$$

Hidden Layer Weight Assignment

W_1 can be updated now:

$(1-out_{h1})$ = 0.59326999(1−0.59326999)=0.241300709

$net_{h1} = w_1 * i_1 + w_3 * i_2 + b_1 * 1$

$w_1^+ = w_1 - \eta * \dfrac{\partial E_{Total}}{\partial w_1}$

$=0.15-0.5*0.000438568=0.14978071$

Similarly, w_2, w_3, and w_4 are provided with their respective values:

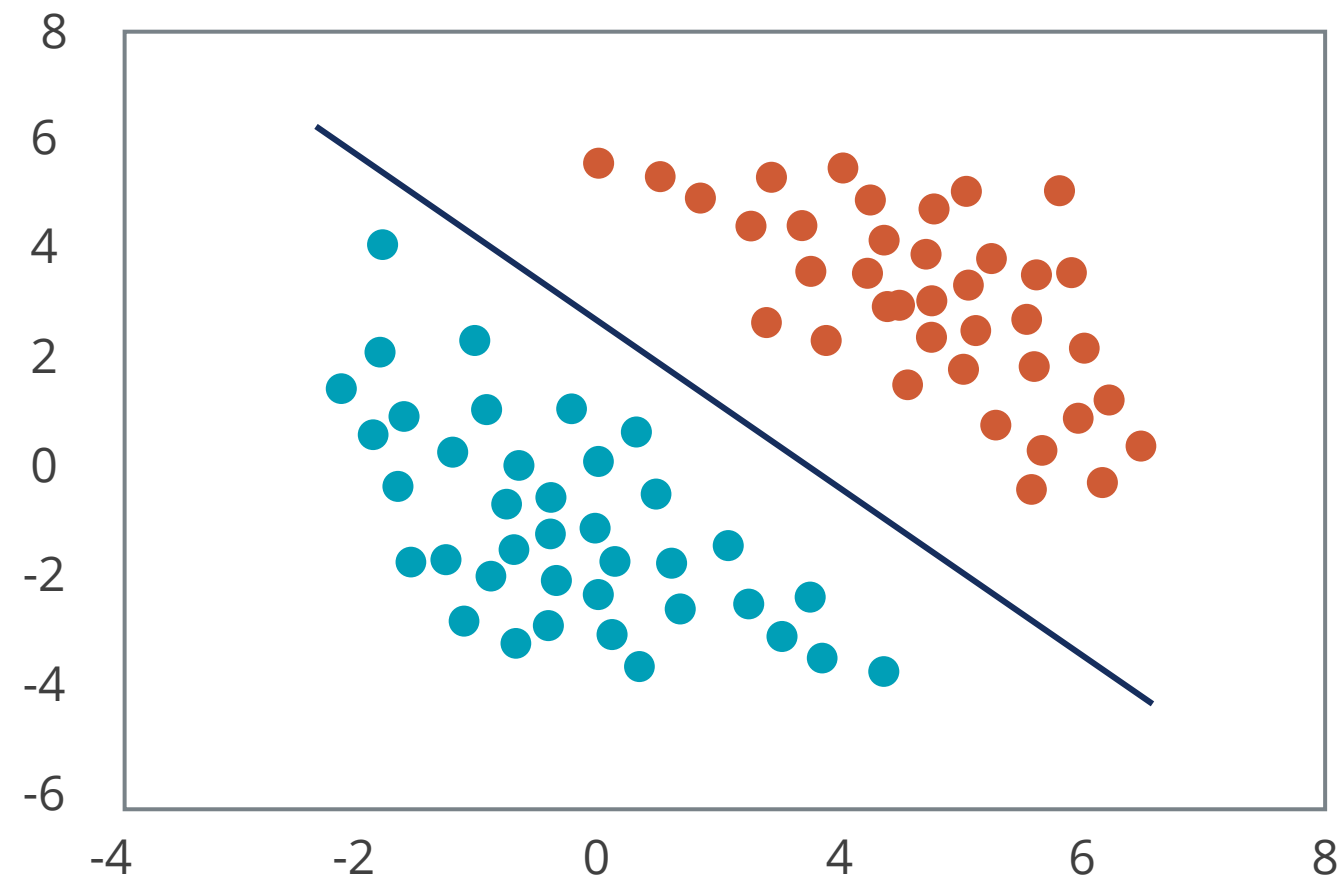$w_2^+ = 0.19956143$

$w_3^+ = 0.24975114$

$w_4^+ = 0.29950229$

Limitations of a Perceptron

Though a perceptron is a simple and efficient supervised learning algorithm, it has certain limitations.

1 The output of a perceptron takes only one of two values 0 or 1.

Limitations: Linearly Separable
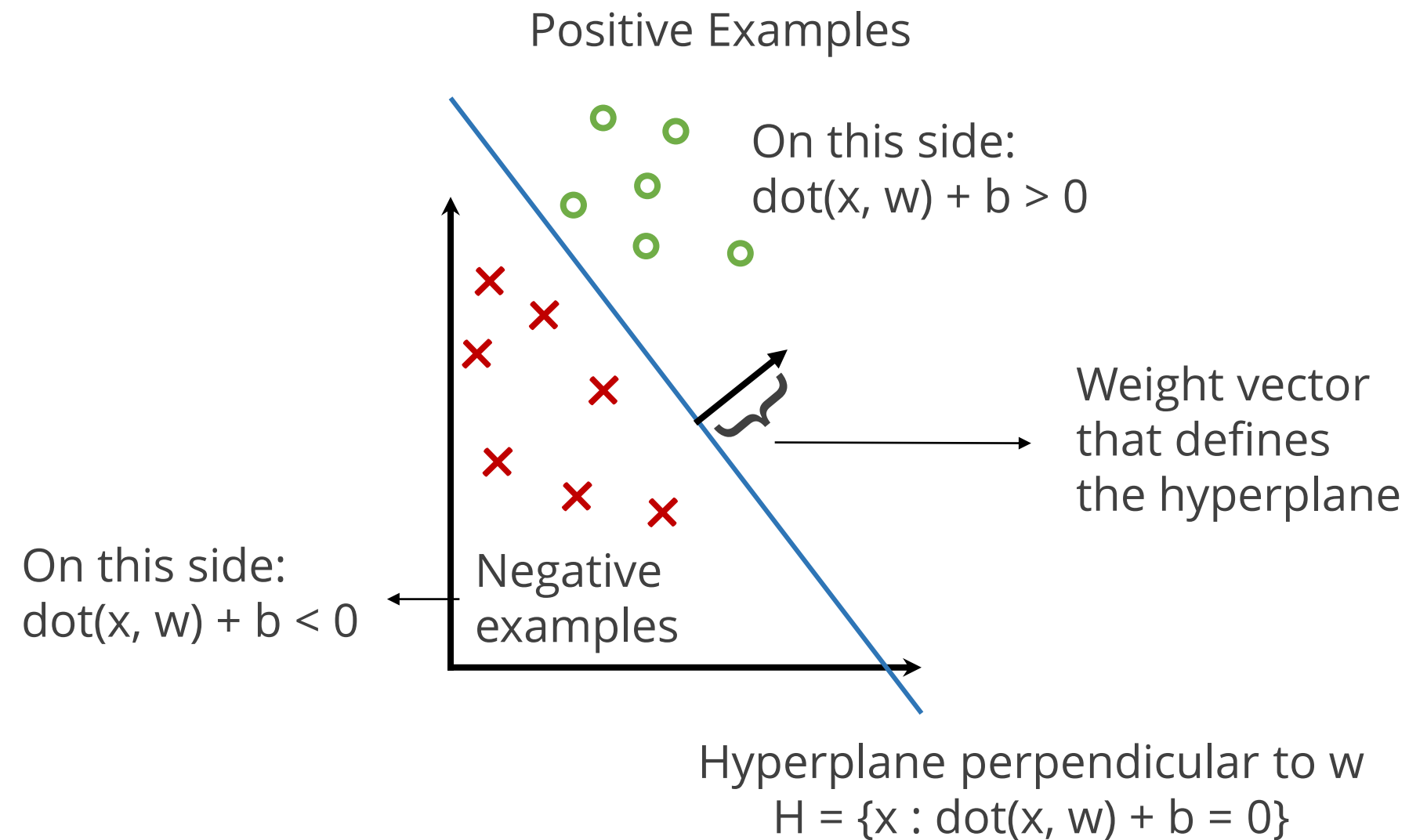
2 It works only with linearly separable data.
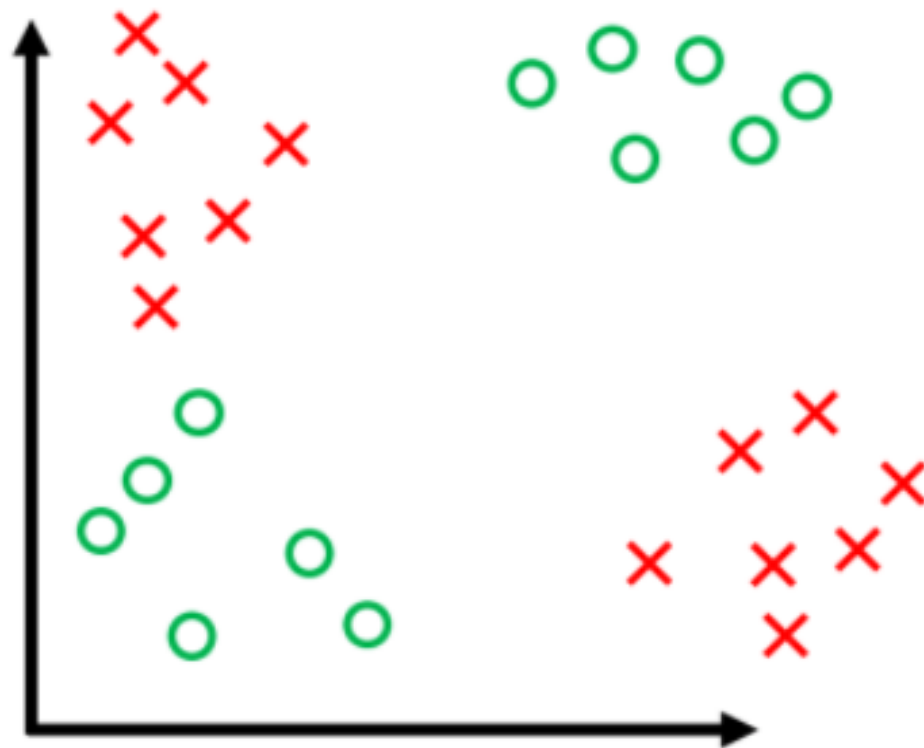
Example

The two-dimensional data shown here is not linearly separable.

No single straight line can completely separate the circles and crosses.
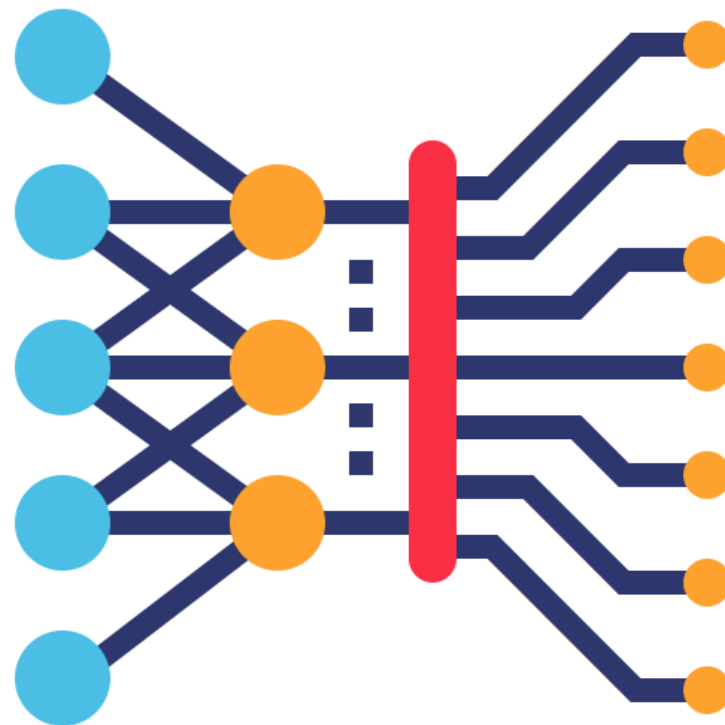
Limitations: Linearly Separable

An example of linearly separable data and a straight line that perfectly separates the two groups:

Positive Examples

On this side:
dot(x, w) + b > 0

Weight vector
that defines
the hyperplane

On this side:
dot(x, w) + b < 0

Negative
examples

Hyperplane perpendicular to w
H = {x : dot(x, w) + b = 0}

Limitations: Convergence

The perceptron algorithm struggles with data that is not linearly separable, as it fails to converge to a solution, leading to an endless cycle of weight updates.
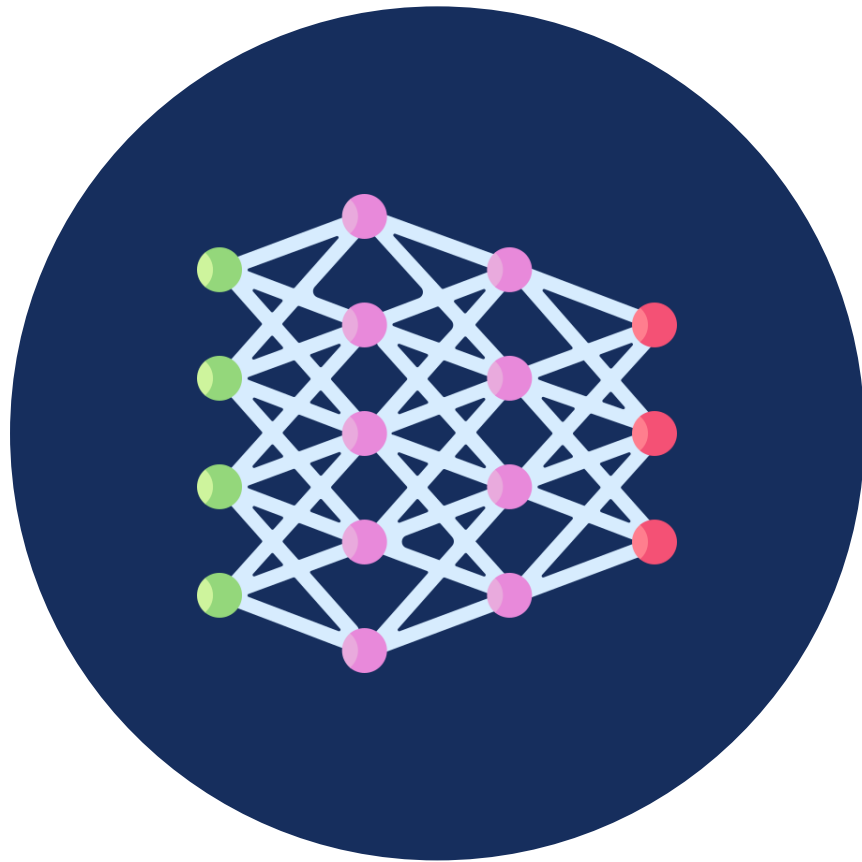

Setting a limit on iterations can prevent the endless cycle, but it does not address the core problem of the perceptron's inability to accurately classify data that is not linearly separable.

Gradient Descent

Gradient Descent

It is an optimization algorithm used to minimize a function iteratively.

It works by repetitively adjusting the input variables in the direction that reduces the function's value the most.

It can end up at different minimum points, depending on the function and the initial starting point.

What Is Gradient Descent?

A linear regression model finds the equation of a straight line that is used to estimate the output.

The straight line is represented as:

$$y = mx + c$$

y = target or dependent variable
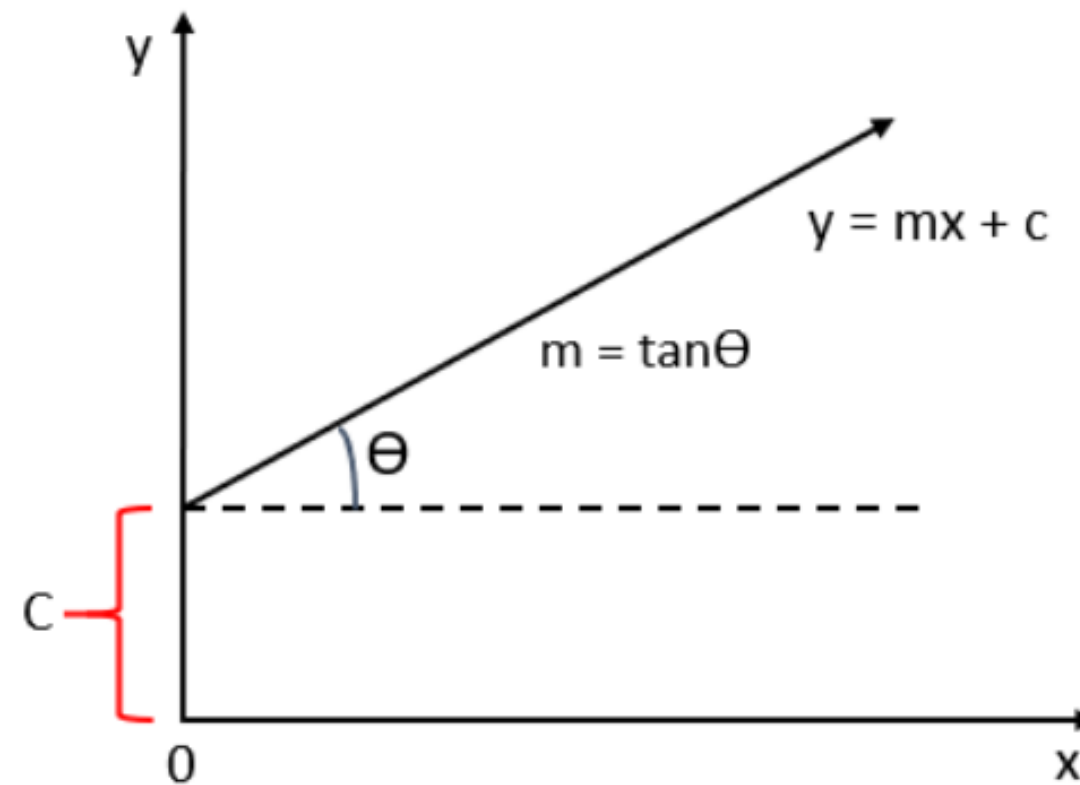
x = input variable

c = intercept

m = slope of the straight line

Linear regression focuses on finding the best-fit line for regression tasks, while simple perceptron aims to classify data into different classes.
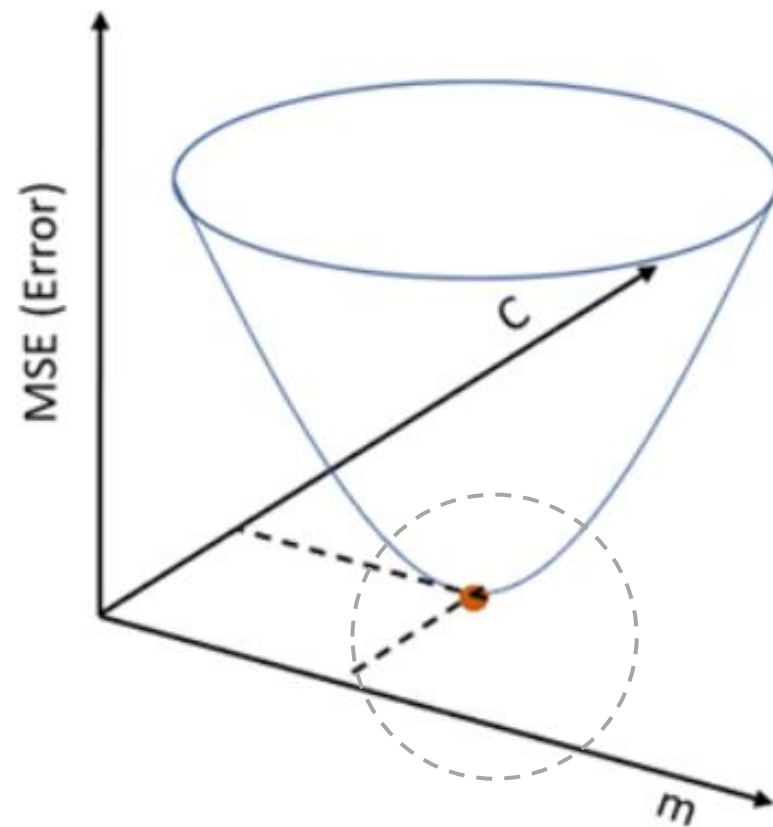
Working on Gradient Descent

There are two parameters, m, and c, that should be optimized to find the best possible solution.

Working on Gradient Descent

If m and c are plotted against MSE, it forms a bowl shape.

The bottom of the bowl-shaped curve is the minimum value of the cost function.

At this point, the values of c and m are their optimal values.

Deriving a Gradient Descent or Ascent Algorithm

The algorithm iteratively updates weights to minimize the overall error/objective function.
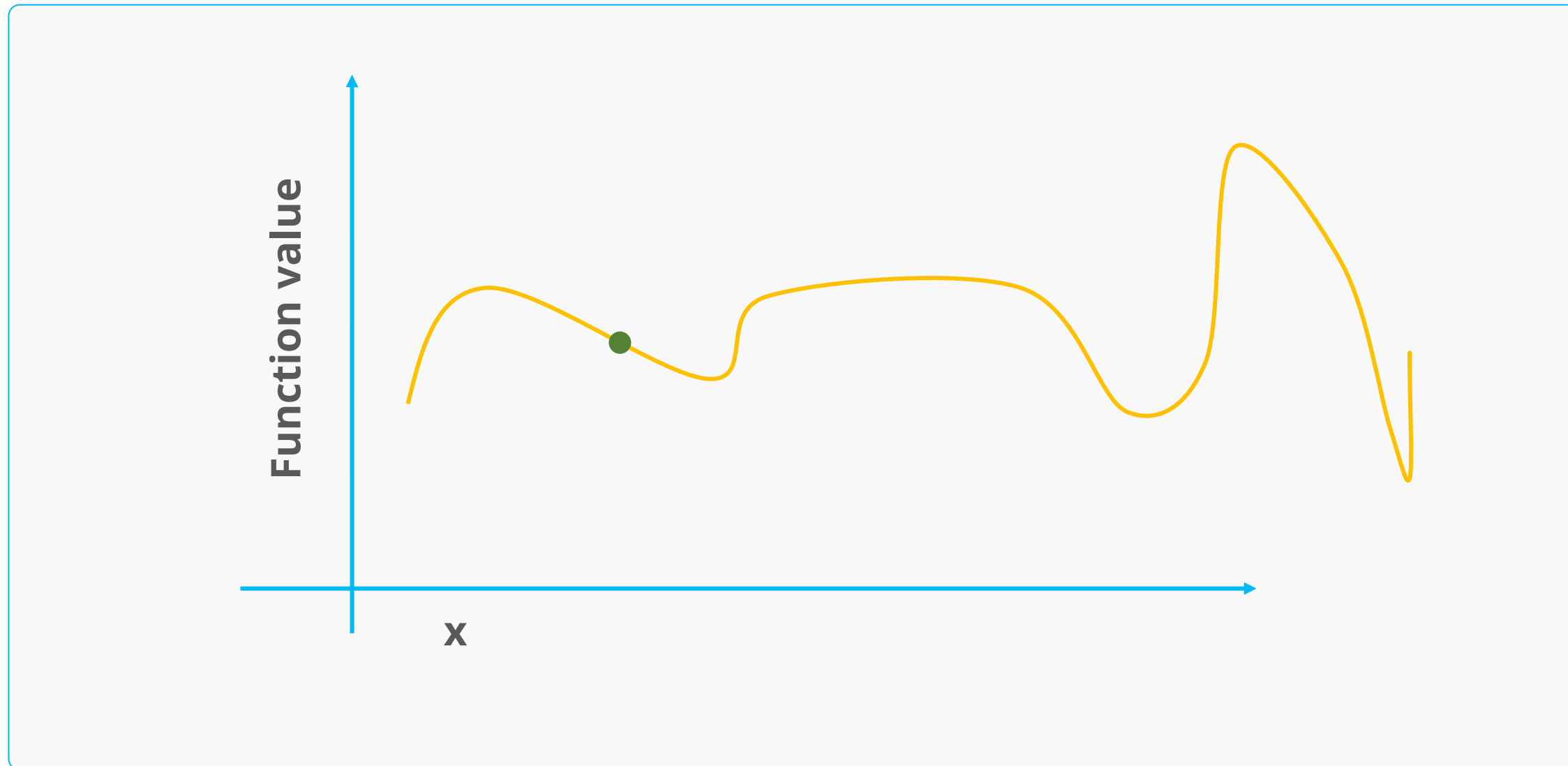
It considers the local gradient, indicating the direction of the largest change.

Weights are updated by taking a step proportional to the gradient.

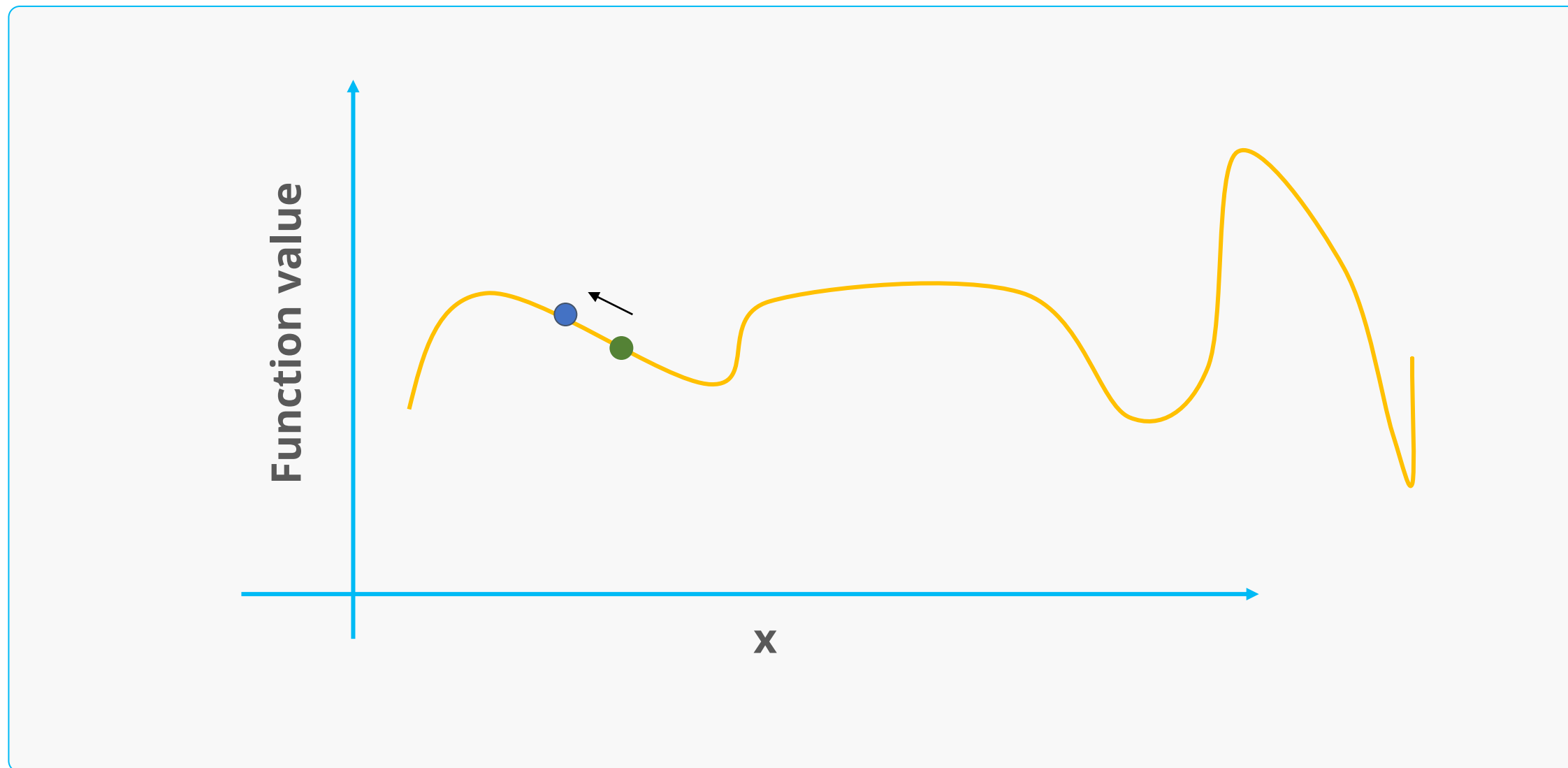Gradient descent accelerates convergence towards the optimum.

Gradient Ascent: Step 01

Initialize the model's parameters by selecting random values as a starting point for the optimization process
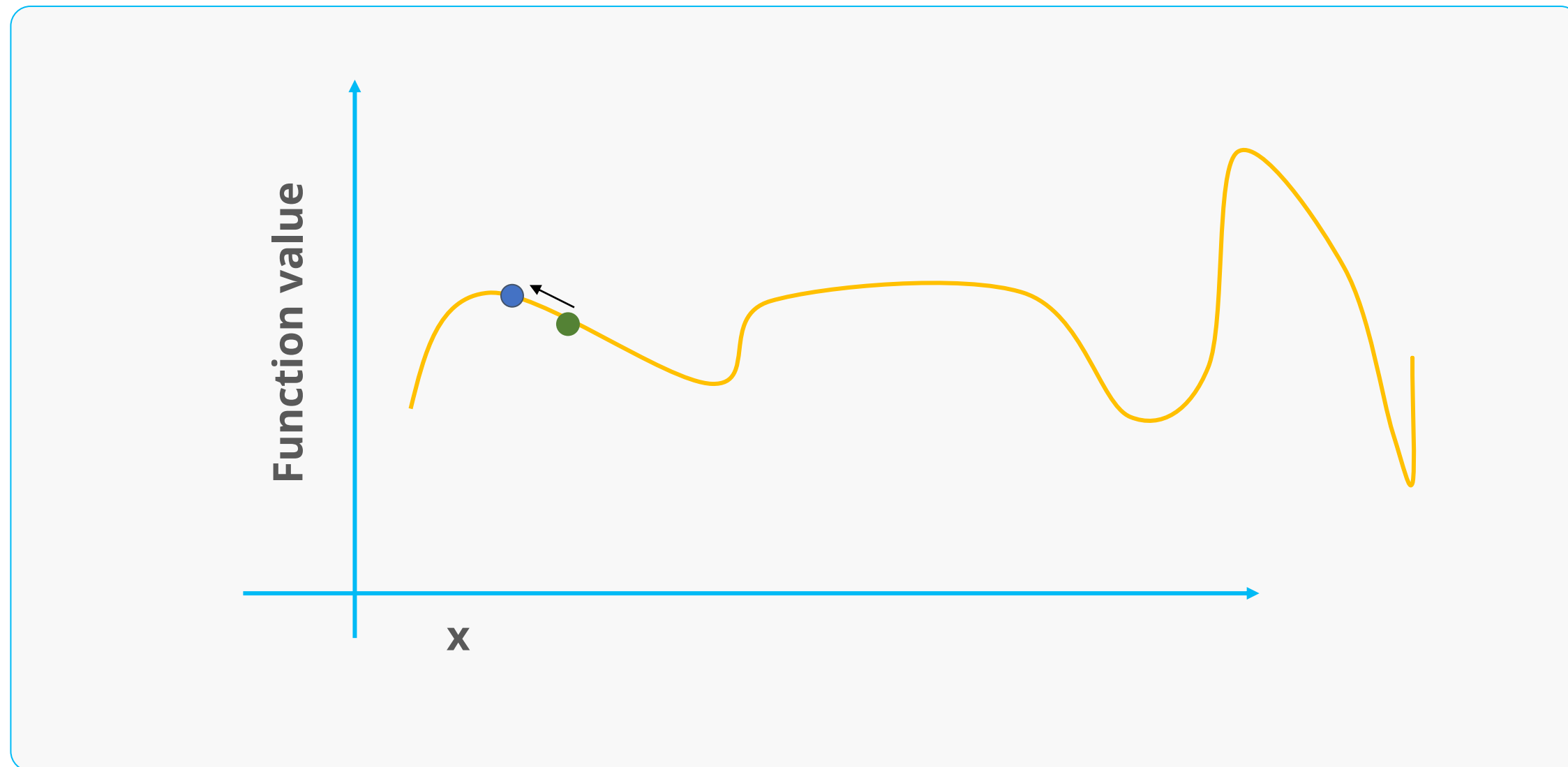
Gradient Ascent: Step 02

Take steps in the direction of the steepest ascent, maximizing the objective function, to iteratively approach the optimal solution
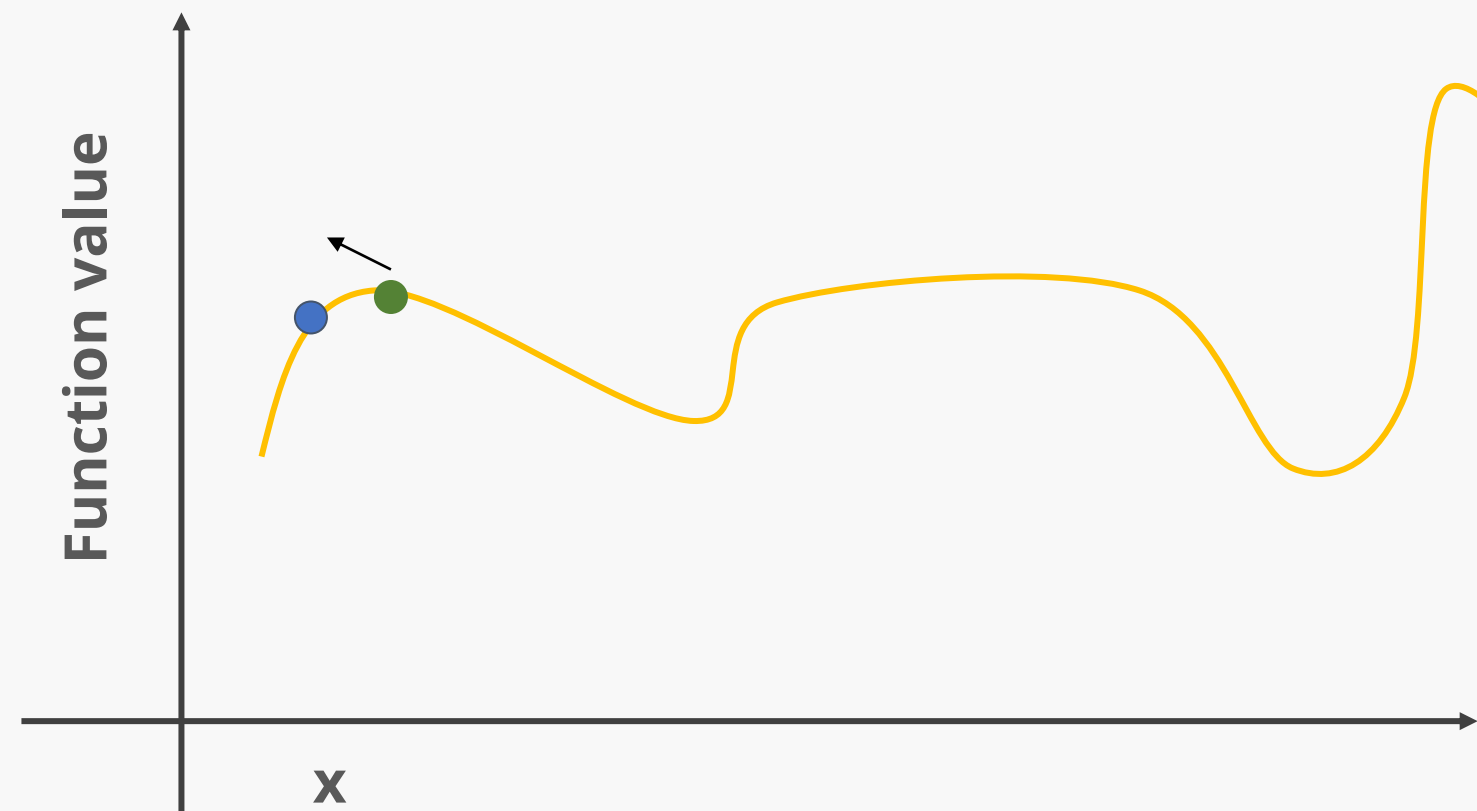
Gradient Ascent: Step 03

Repeat steps 1 and 2 until a stopping criterion is met, enabling continuous parameter updates and refinement toward the optimal solution
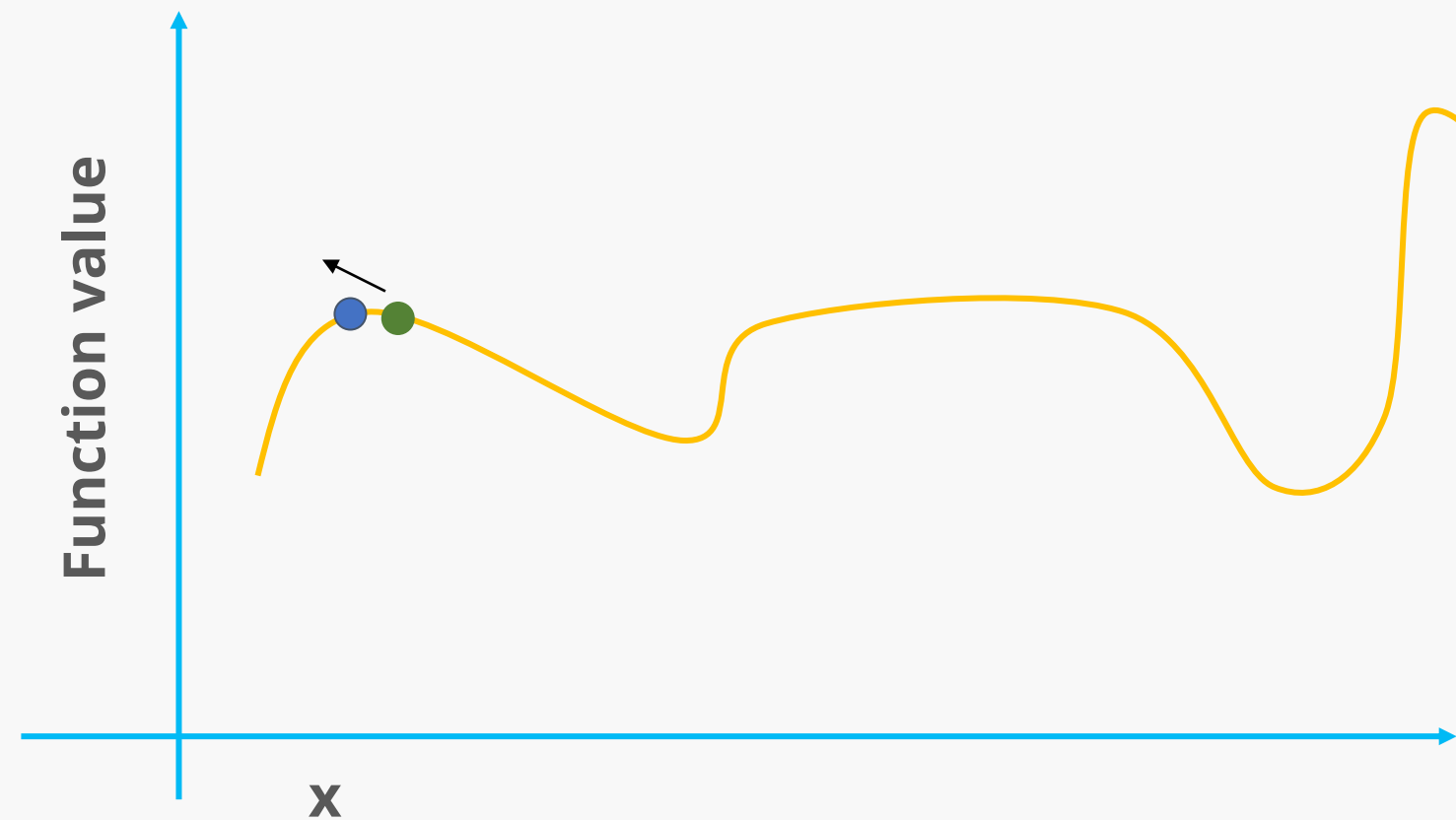
Gradient Ascent: Observations

It stops iterating if the next step reduces the objective function or meets a termination condition, indicating the algorithm has reached a satisfactory or optimal solution.

Gradient Ascent: Observations

The optimization process reduces the step size to take smaller, more precise steps towards the optimal solution, improving convergence and accuracy.

Gradient Ascent: Observations

The optimization process aims to converge to a (local) maximum by iteratively updating parameters and adjusting step size, yielding a potentially optimal solution.
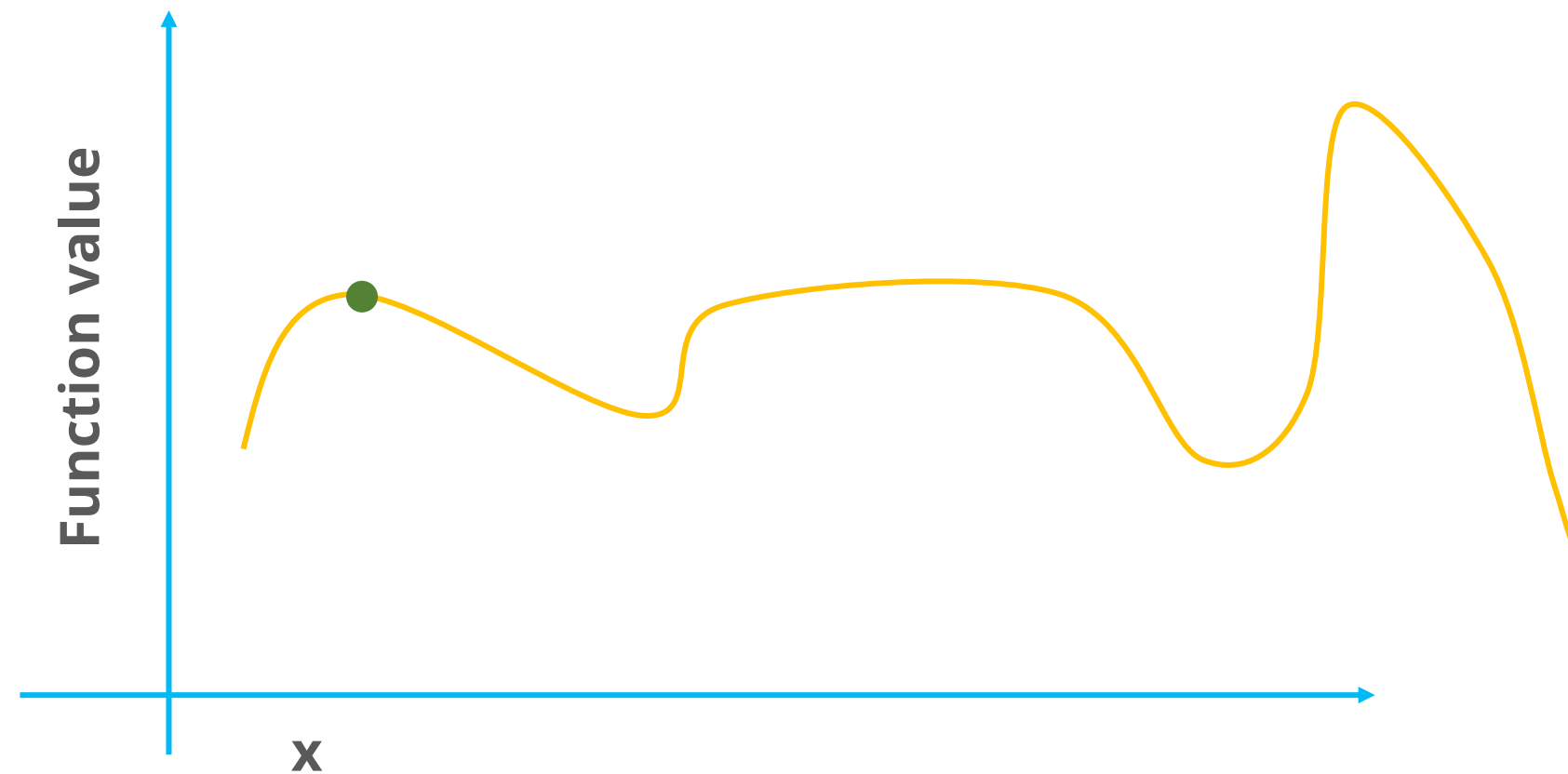
Gradient Ascent Impacts on Neural Network

- In gradient ascent, the Y axis is the function value. The gradient is used to determine whether the weights should change.

- If the gradient is positive, then the weights are decreased.

- If the gradient is negative, then the weights are increased.

- The object of gradient ascent isn't to minimize but to maximize a function.

The Learning Rate

- The learning rate is used to control the changes made to the weights and biases in order to reduce the error.
- It is used to analyze how an error will change when the values of weights and biases are changed by a unit.
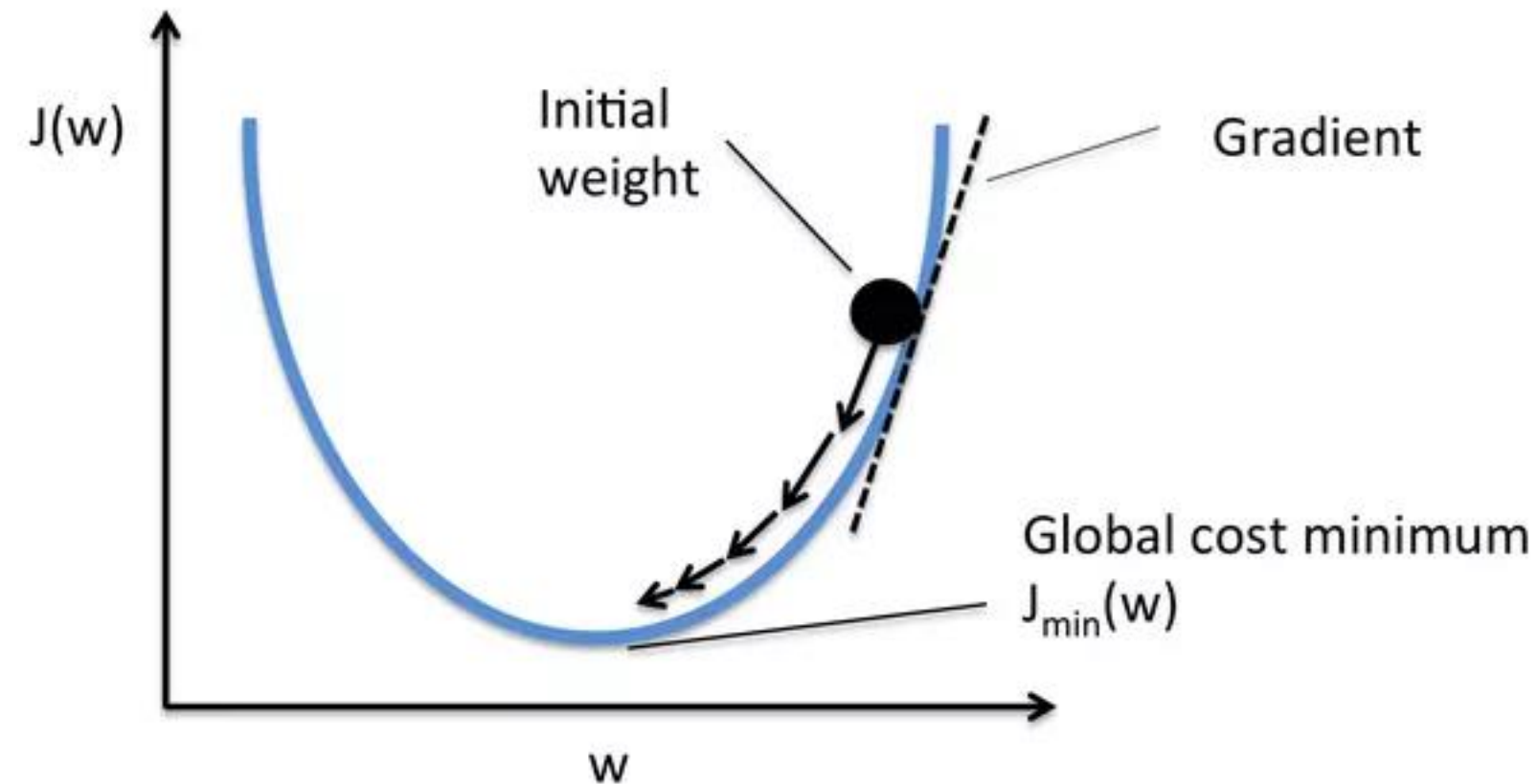

Note: Generally, a learning rate of 0.01 is a safe bet.

Key Takeaways

- Perceptron is a type of artificial neuron used for binary classification in machine learning.

- Forward propagation generates an output in a perceptron model and passes inputs through weights and an activation function.

- Backpropagation involves adjusting the weights of the inputs in order to minimize the error in the perceptron model.

- Gradient descent is a technique used to minimize the cost function in a neural network.

- Perceptrons have limitations as they are able to handle linearly separable data.
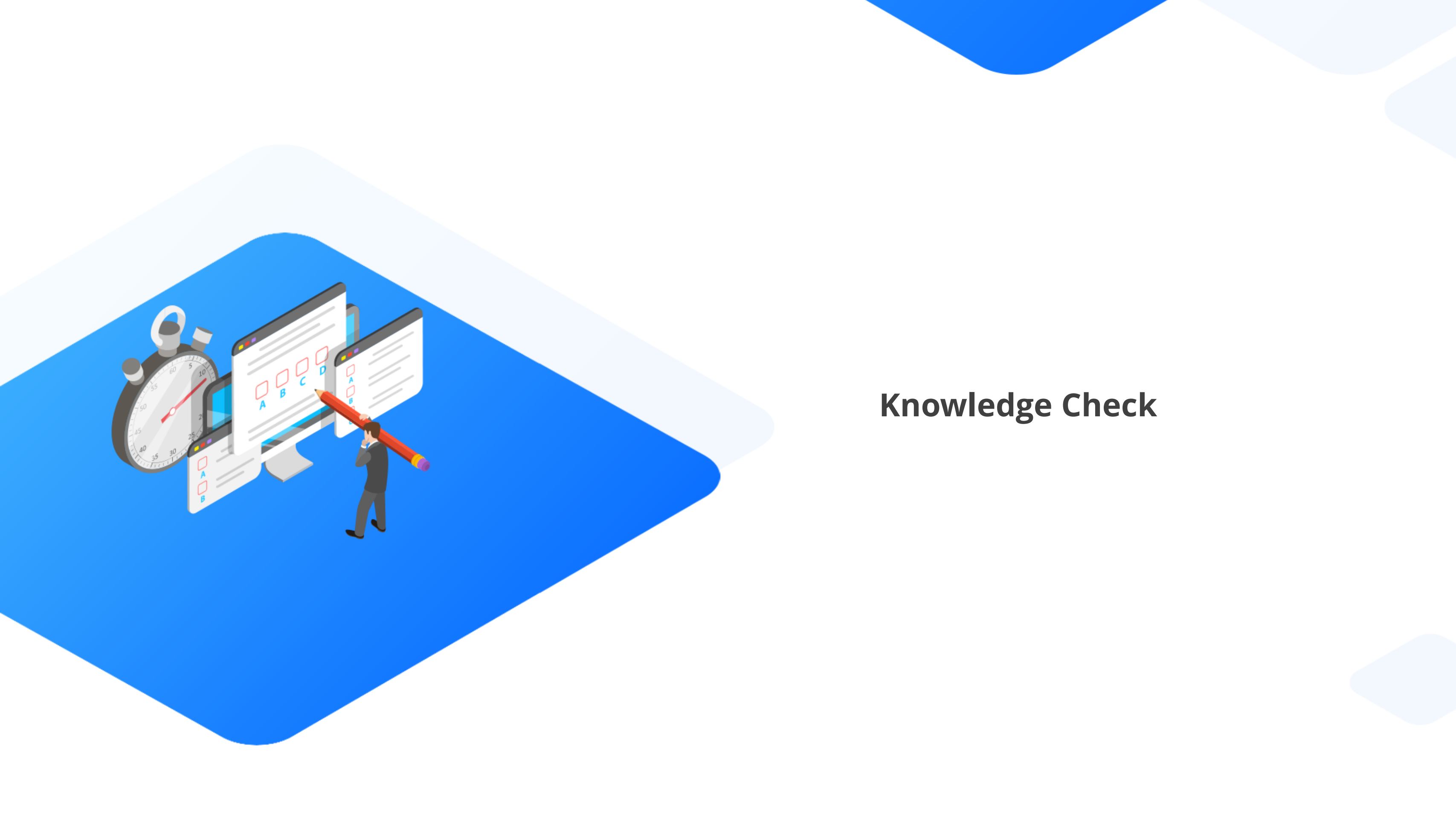
Knowledge Check

What is the importance of weights in a perceptron?

A.   To receive information from other neurons

B.   To determine the input layer

C.   To determine the activation function

D.   To determine the importance of the inputs

What is the importance of weights in a perceptron?

A.   To receive information from other neurons

B.   To determine the input layer

C.   To determine the activation function

D.   To determine the importance of the inputs

The correct answer is **D**

Weights help determine the importance of the inputs.

What is an activation function in a perceptron?

A.   An equation that determines the output of a neural network model

B.   A type of neuron in a neural network

C.   A process of updating the weights in a neural network

D.   A type of supervised learning algorithm

What is an activation function in a perceptron?

A.   An equation that determines the output of a neural network model

B.   A type of neuron in a neural network

C.   A process of updating the weights in a neural network

D.   A type of supervised learning algorithm

The correct answer is **A**

It is an equation that determines the output of a neural network model.

What is the cost function in a neural network?

A.   It generates an output from the input layer to the output layer

B.   It measures the error in the network

C.   It minimizes the error by changing the weights of the input neurons

D.   It finds the optimal values of the parameters

What is the cost function in a neural network?

A. It generates an output from the input layer to the output layer

B. It measures the error in the network

C. It minimizes the error by changing the weights of the input neurons

D. It finds the optimal values of the parameters

The correct answer is **B**

The cost function measures the error in the network by comparing the predicted output with the ground truth.

Thank You