Deep Learning with Keras and TensorFlow

Artificial Neural Network

Learning Objectives

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

- Understand the structure and function of neural networks, including perceptrons and multilayer perceptrons.
- Evaluate the performance of different neural networks, including DNNs, CNNs, and RNNs
- Analyze various activation functions like ReLU, Sigmoid, and Softmax and their effects on performance.
- Create and optimize neural network models, addressing issues like vanishing and exploding gradients

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.

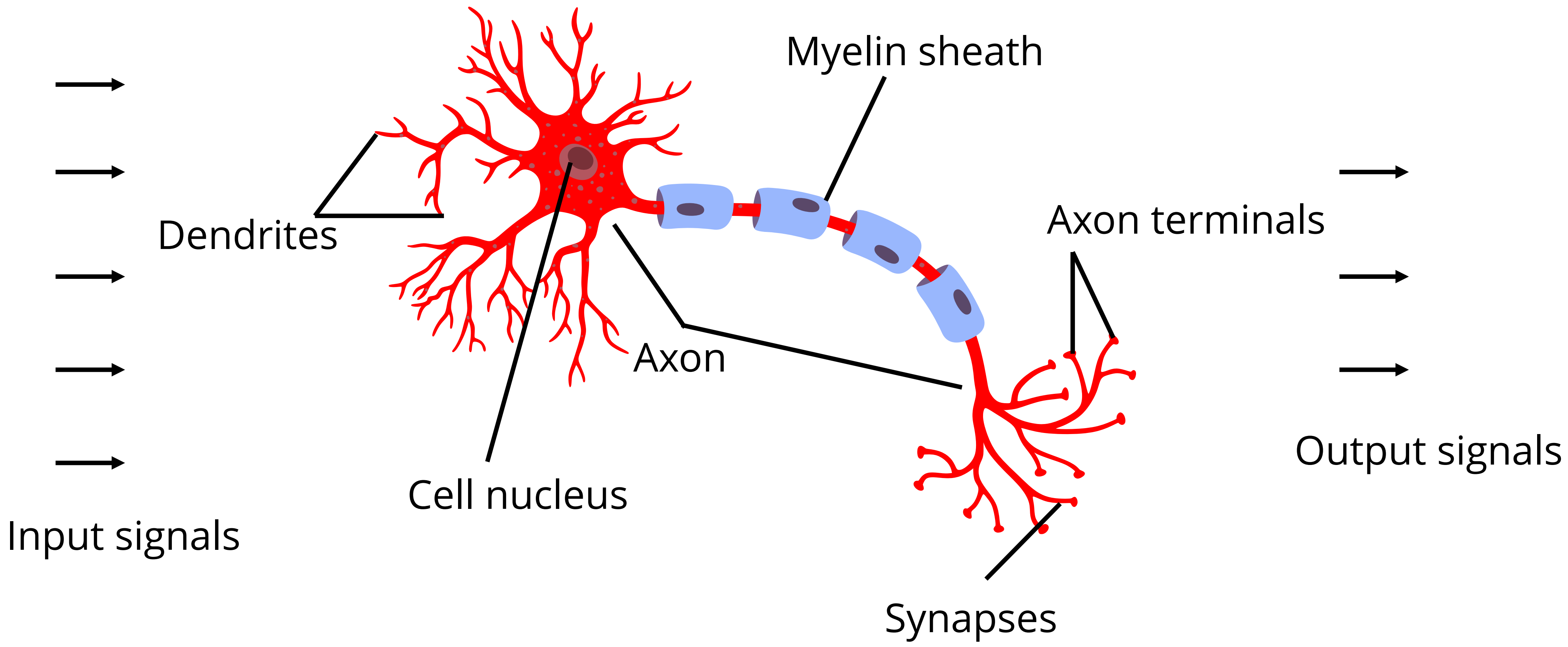
The system takes in large volumes of financial transaction data and uses forward propagation to classify each transaction as either fraudulent or legitimate. If a transaction is classified as fraudulent, the system alerts the relevant authorities for further investigation.

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.

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:

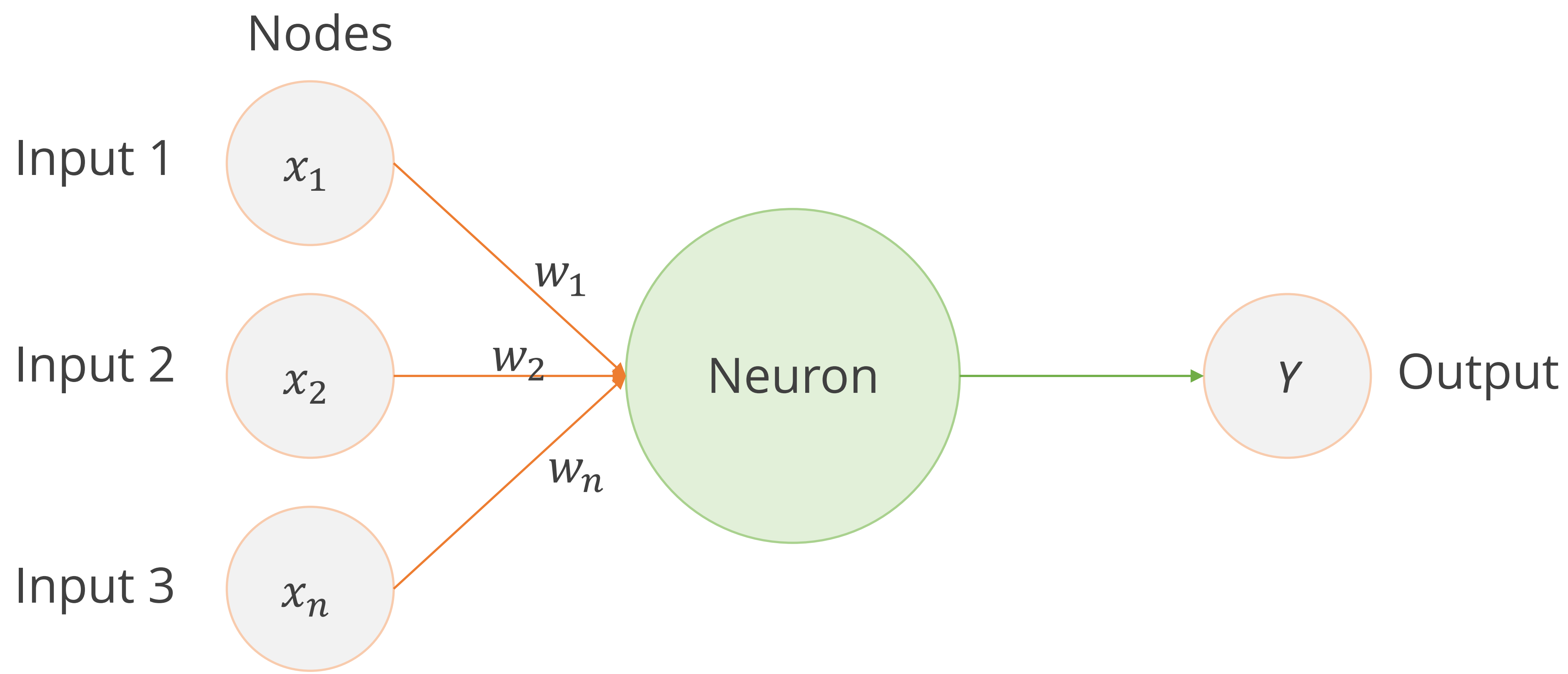
Dendrites receive inputs from other neurons.

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

Axons transmit the biological neuron's output.

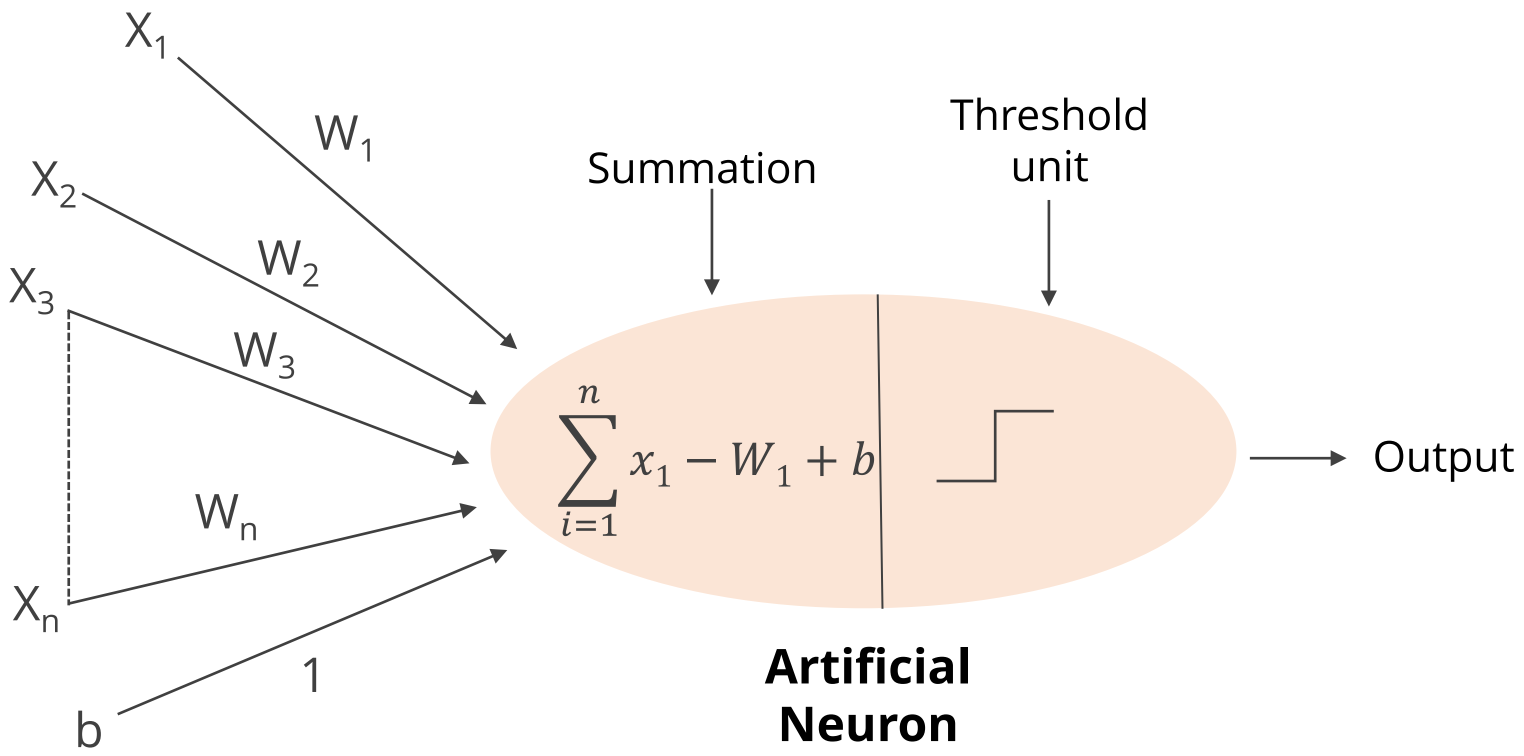
Synapse is the connection between two nerve cells.

Definition of Artificial Neuron



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



Artificial Neuron

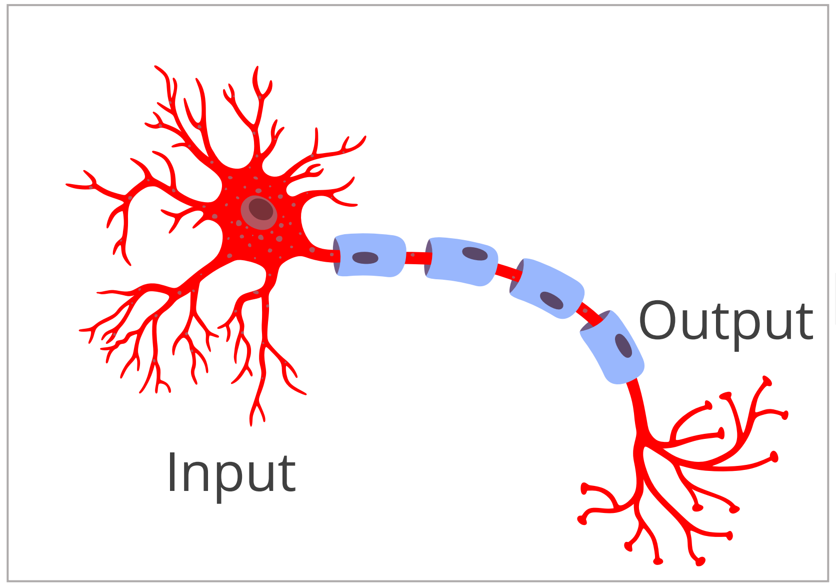
W_1, W_2, ……. W_n - Weights of connection

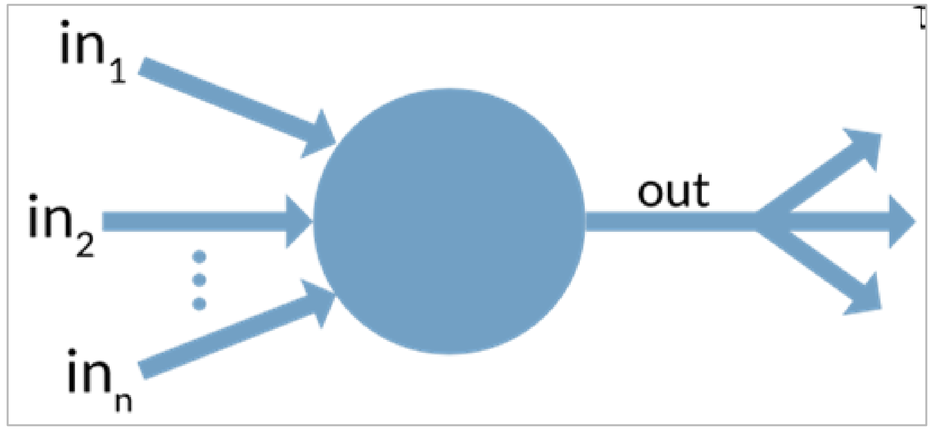
X_1, X_2, …… X_n, - Inputs

b - Bias

- A nerve cell is 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



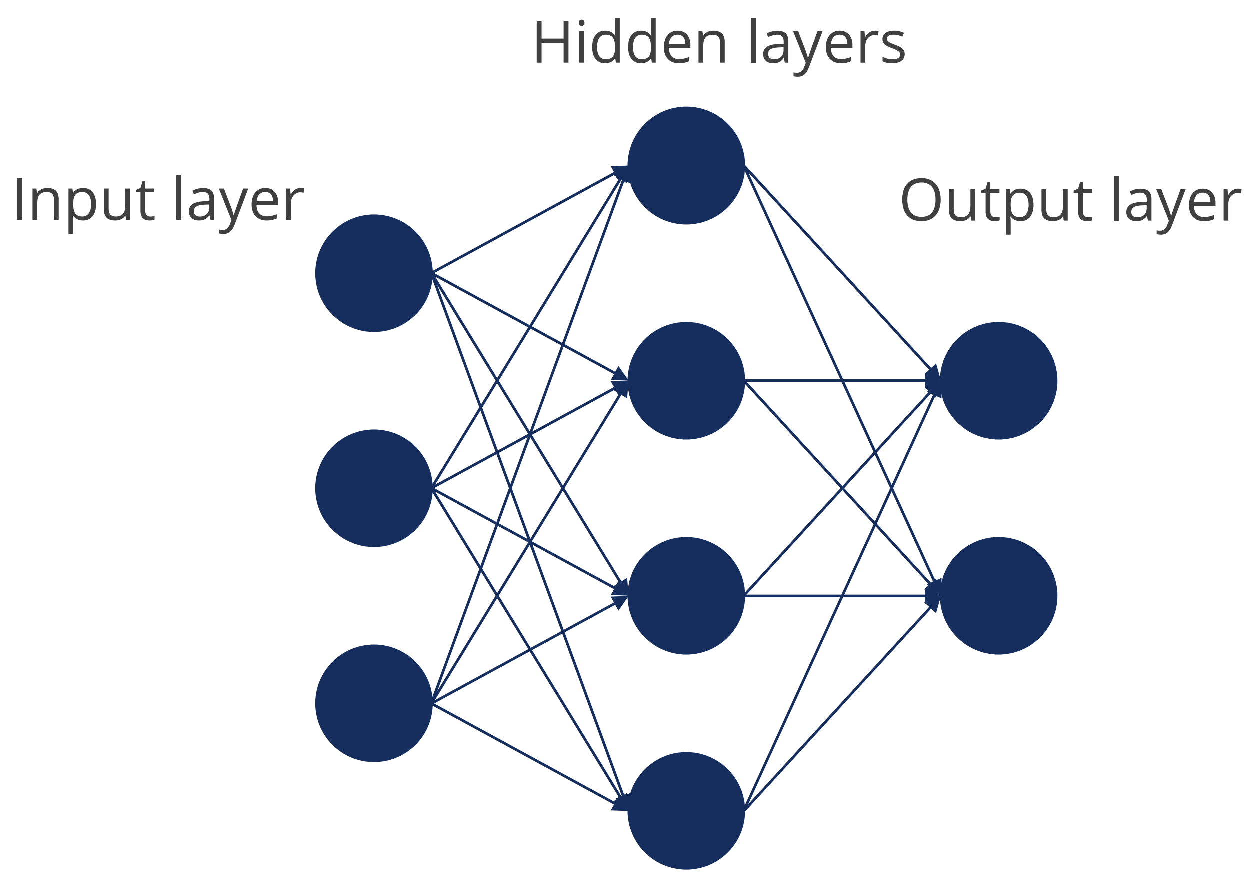


Biological Neurons	Artificial Neurons
Cell nucleus	Node
Dendrites	Input
Synapses	Weights or interconnections
Axon	Output

Neural Networks and Types of Neural Networks

Neural Networks

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



Each neural network consists of multiple node layers such as:

- Input layers
- One or more hidden layers
- Output layers

Neural Networks

ANN processes the information using the following steps:

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.

Types of Neural Networks

Neural networks are used to solve complex problems that require analytical calculations.

Some of the major types of neural networks are:

- Perceptron
- Multilayer perceptron
- Deep neural networks or DNNs
- Convolutional neural networks or CNNs
- Recurrent neural networks or RNNs

Perceptron and Multilayer Perceptron

Perceptron:

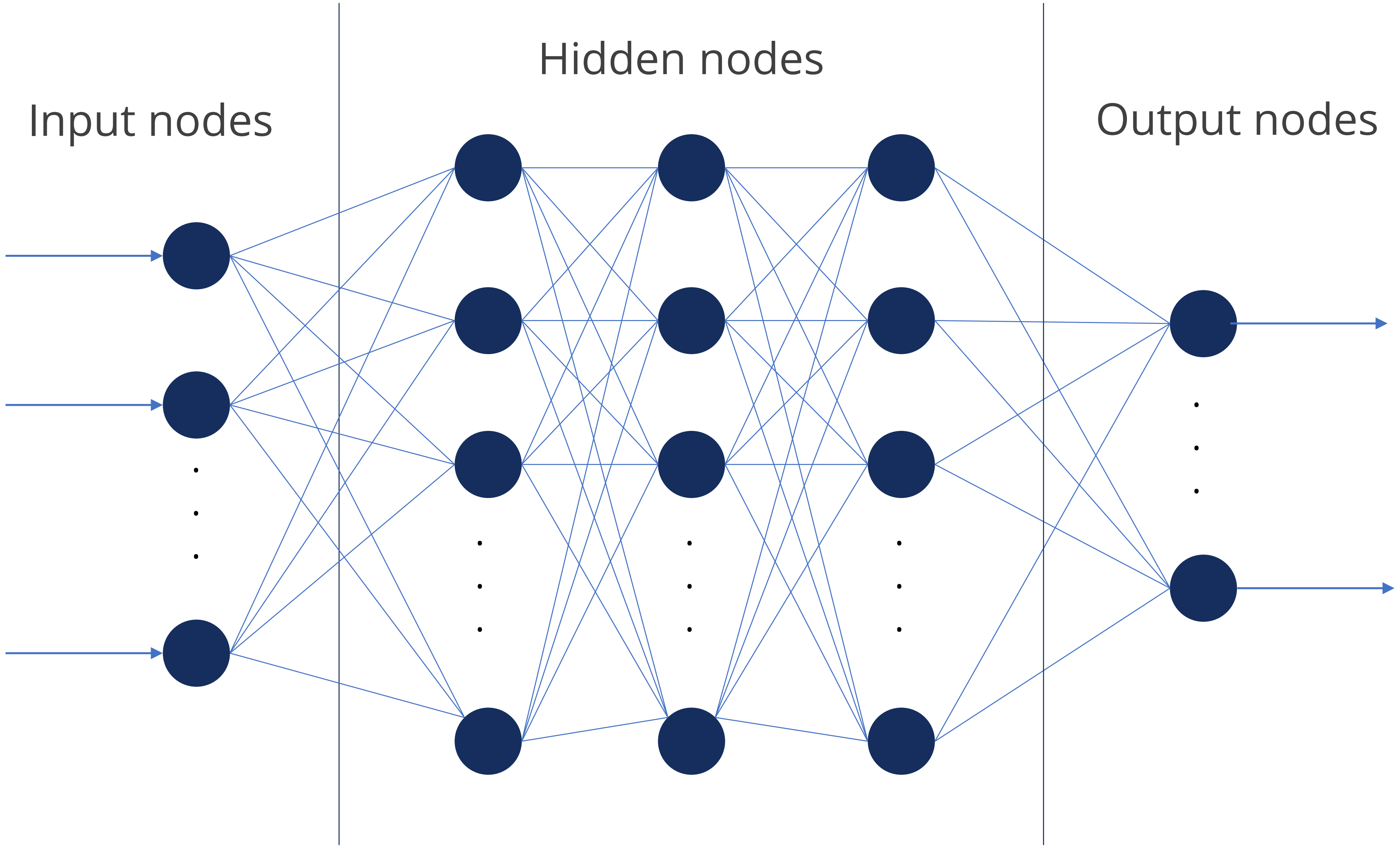
- This is the simplest type of ANN and is mainly used for binary prediction.
- A perceptron can only work if the data is linearly separable.

Multilayer perceptron (MLP):

- It consists of multiple layers of perceptrons.
- The inputs pass through each layer, and the outputs of the last layer are the final outputs of the MLP.
- It can handle more complex problems by learning non-linear relationships.

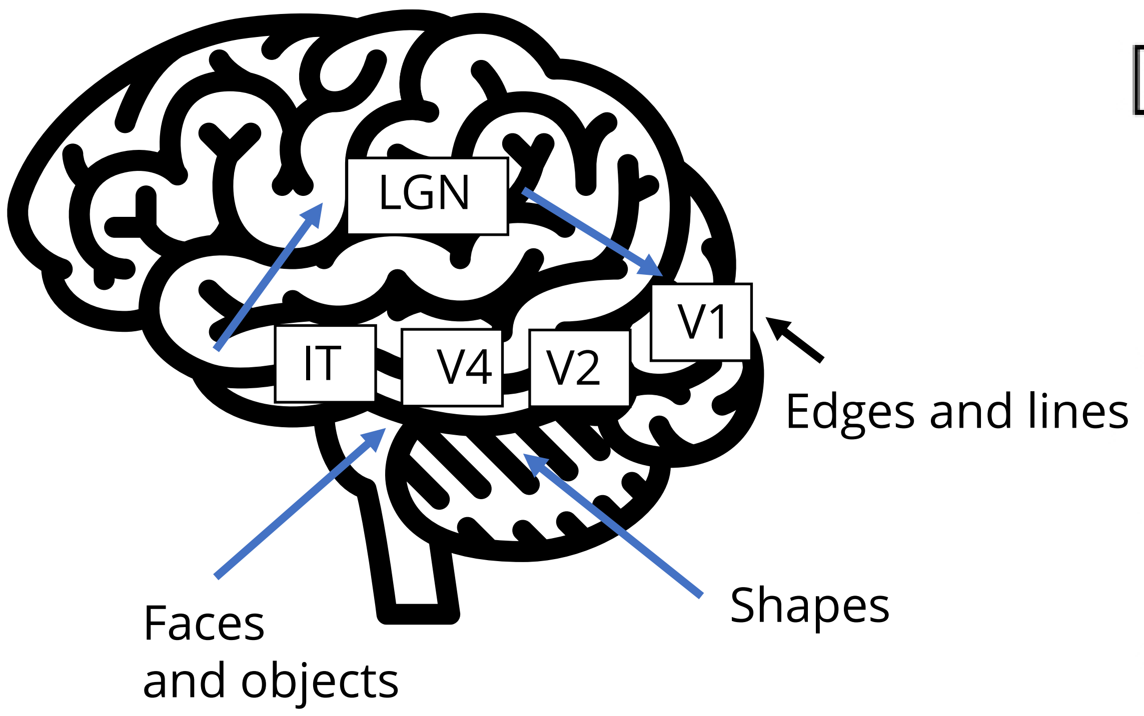
Deep Neural Networks (DNN)

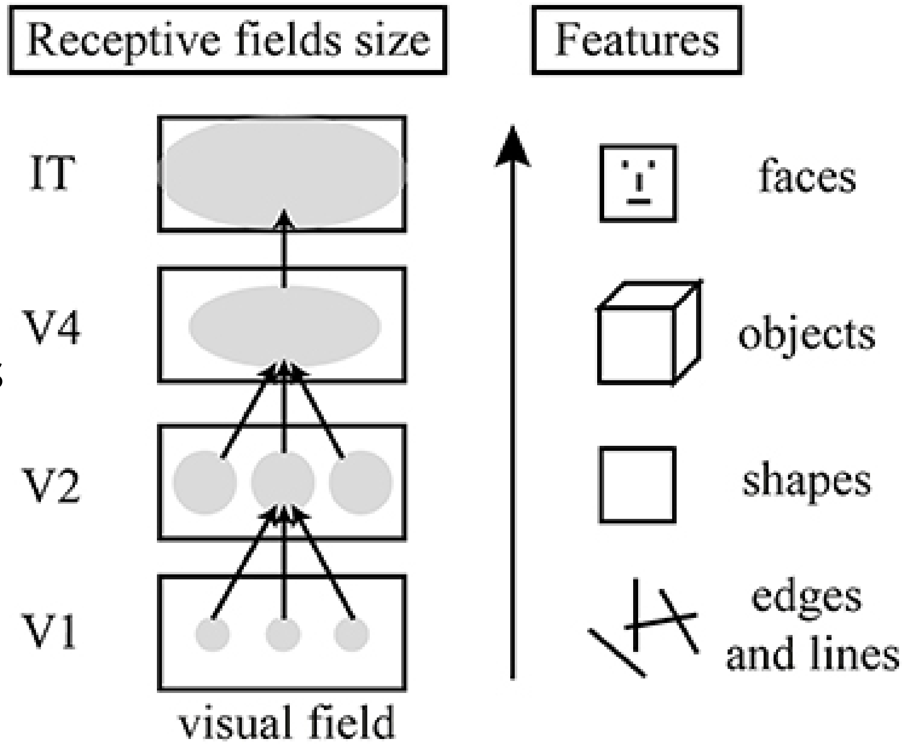
A DNN is a multilayered computational model that processes data in a layered manner, refining information at each layer, analogous to how a human brain functions.



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

Convolutional Neural Network (CNN)



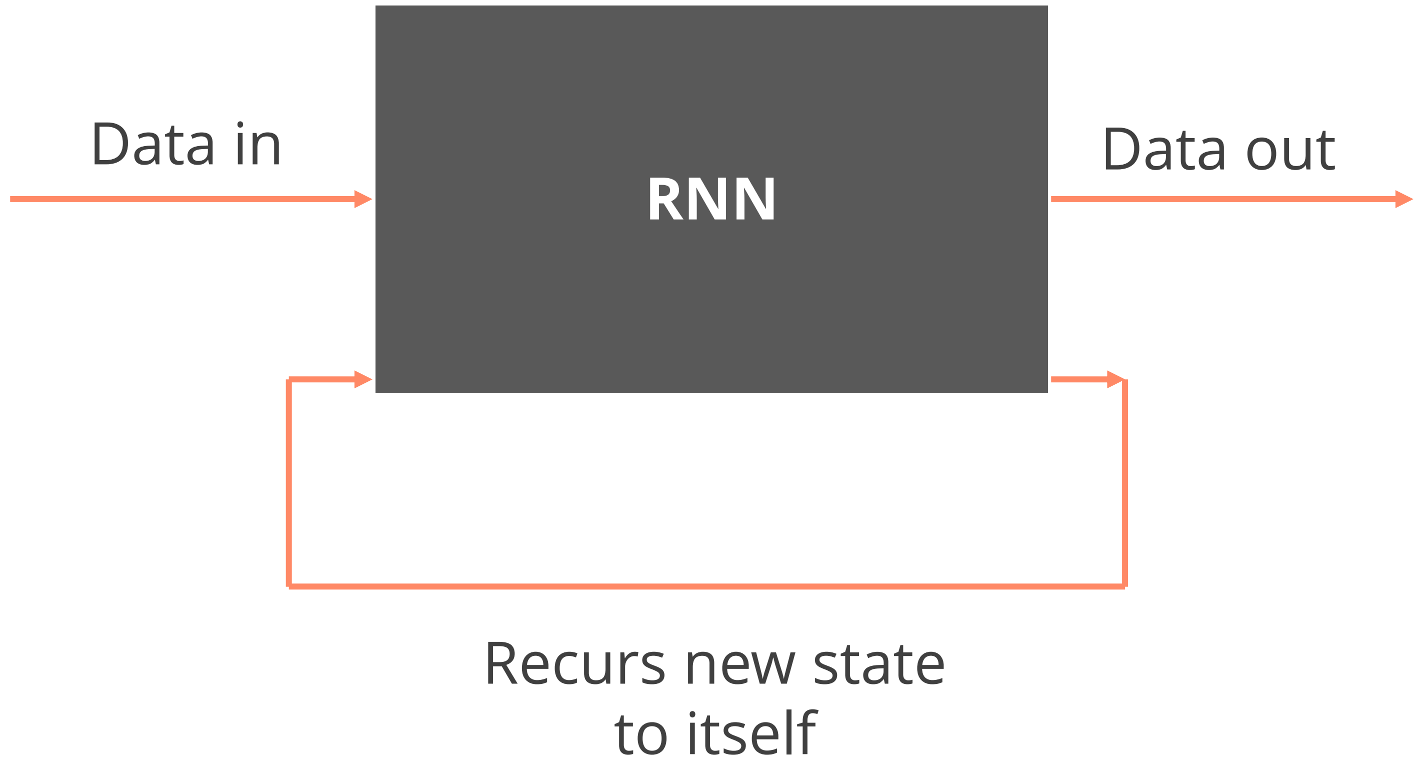


- The idea of CNNs was neurobiologically motivated by the findings of locally sensitive and orientation-selective nerve cells in the visual cortex.
- CNNs analyze visual data and are inspired by the brain's visual cortex. Effective for recognizing patterns in images.
- Example: Image Classification

Recurrent Neural Network (RNN)

RNNs handle sequential data, suitable for time series and language modeling tasks.

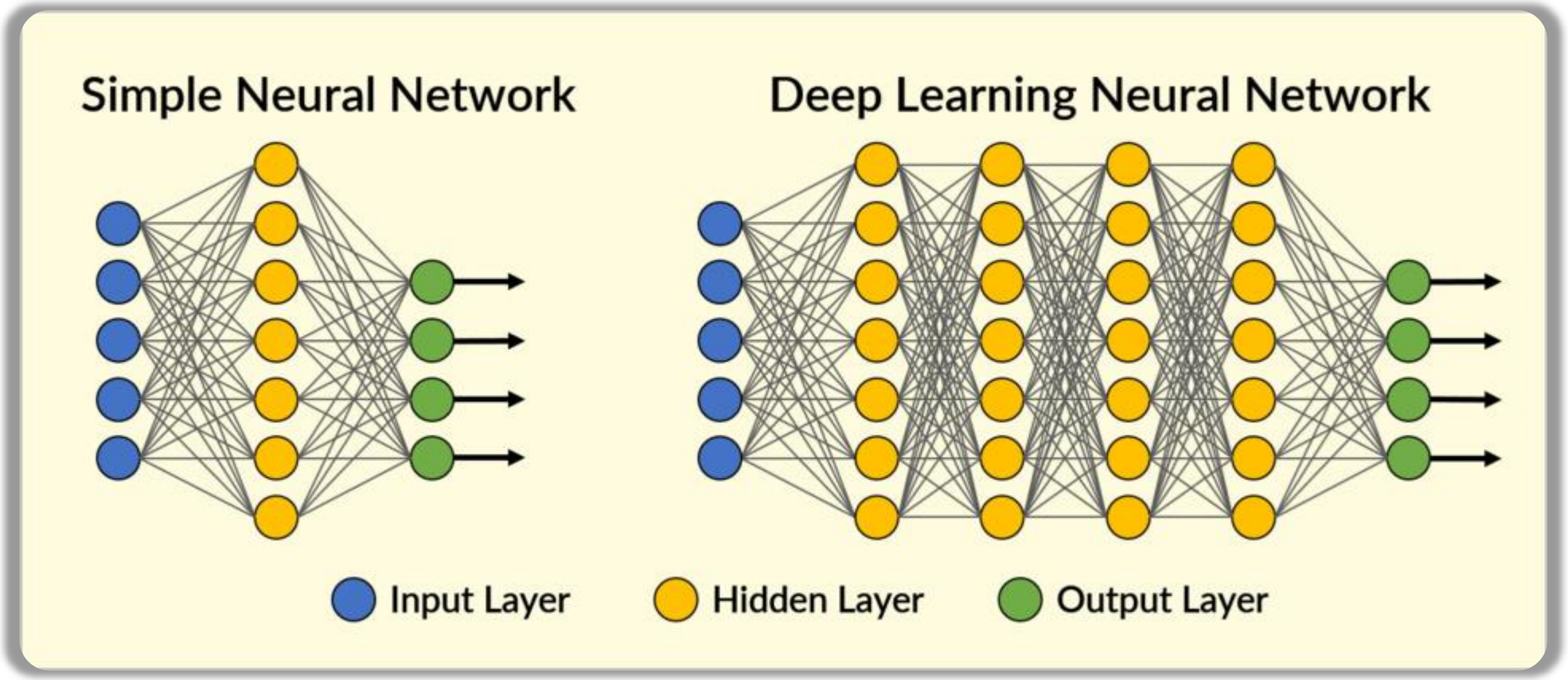
Maintain a state (memory) of previous inputs.



Note: You can think of the **state** as the **memory** of the RNN that recurs into the net with each new input.

Network Architecture

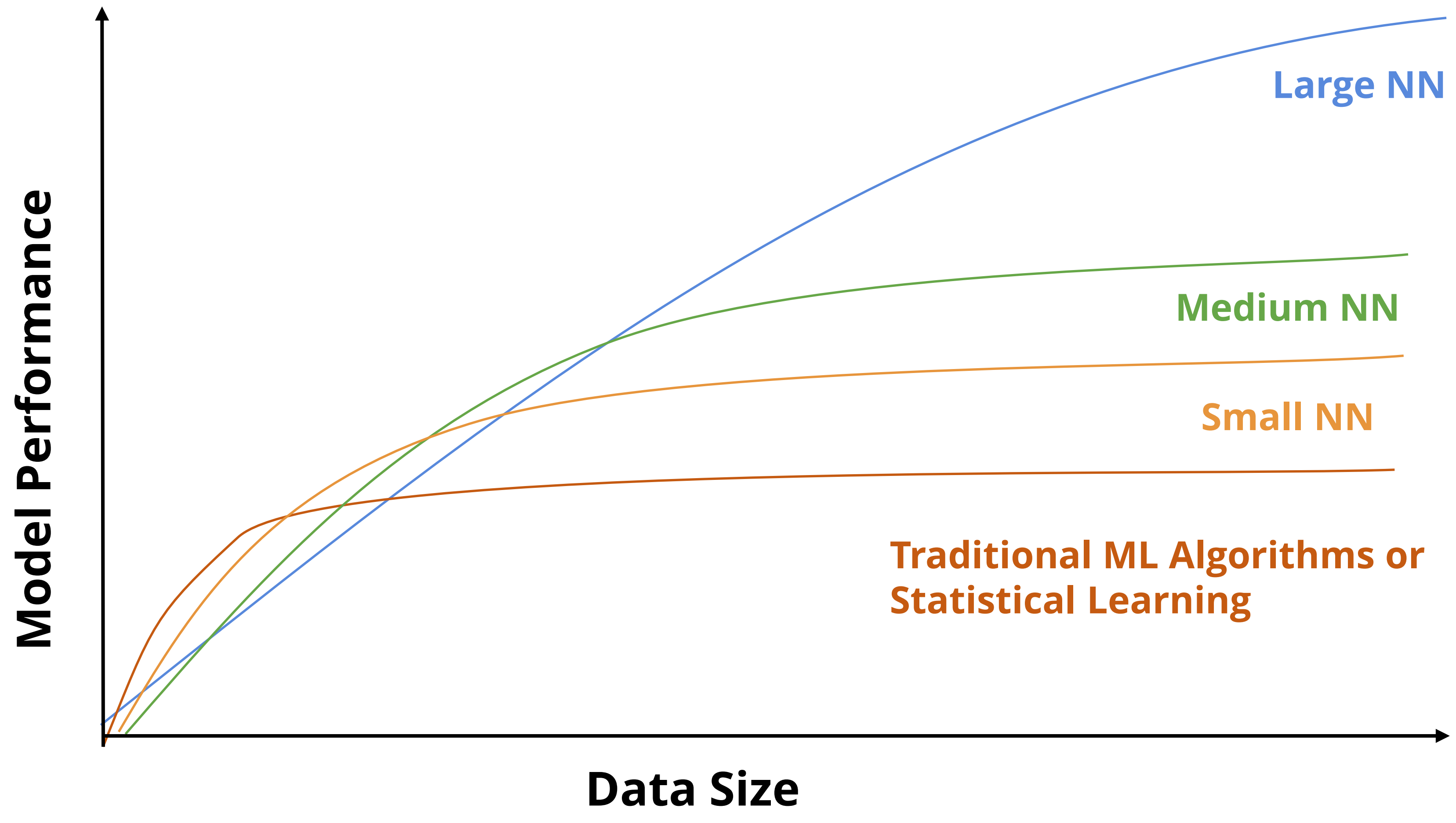
One can programmatically set the number and type of neural layers and the number of neurons comprising each layer.



Deep neural networks consist of multiple hidden layers.

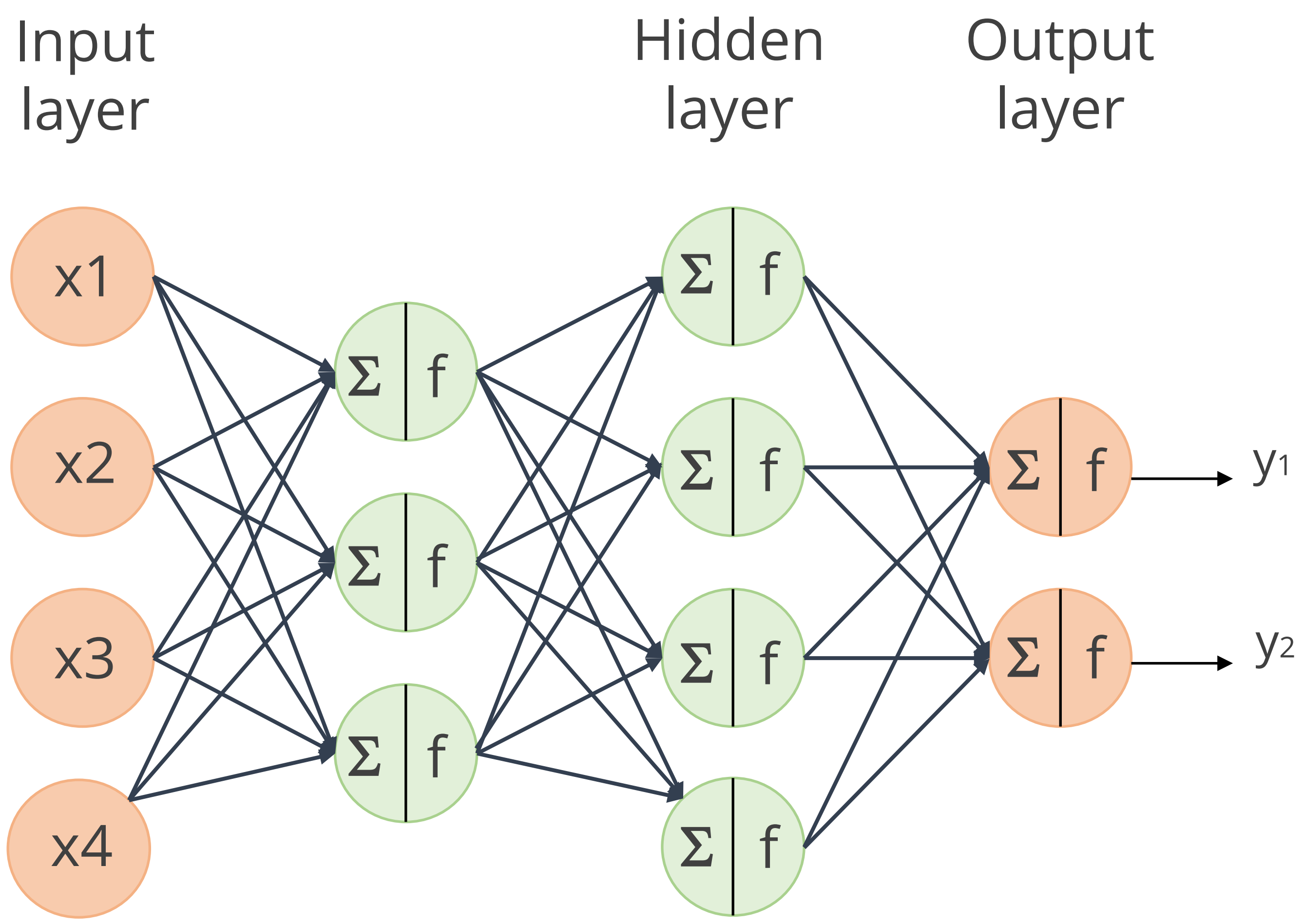
Performance

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



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.

Perceptron

What Is a Perceptron?

The perceptron is a fundamental type of artificial neural network designed for binary classification tasks. It computes a weighted sum of its inputs, which is then passed through a step function to determine the output class.

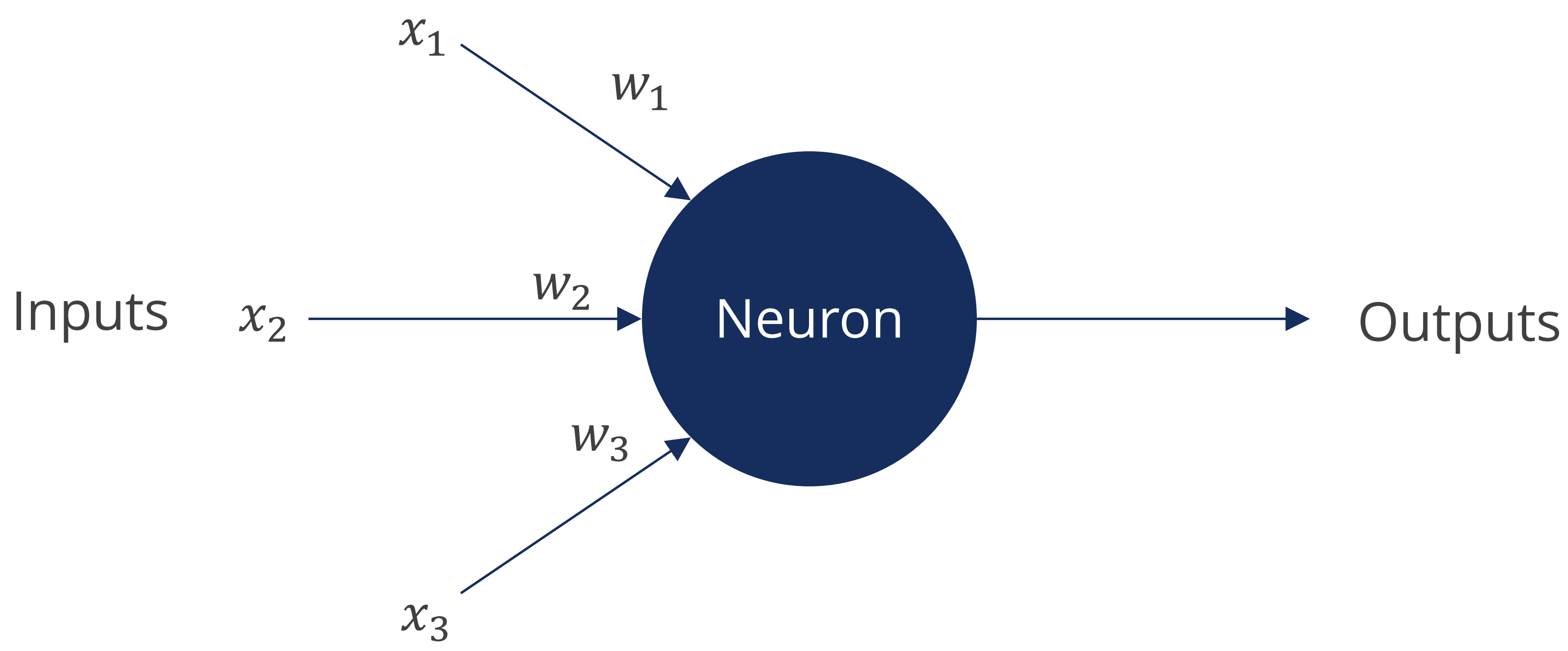
Perceptron equation

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

- 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.

Working of a Perceptron

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

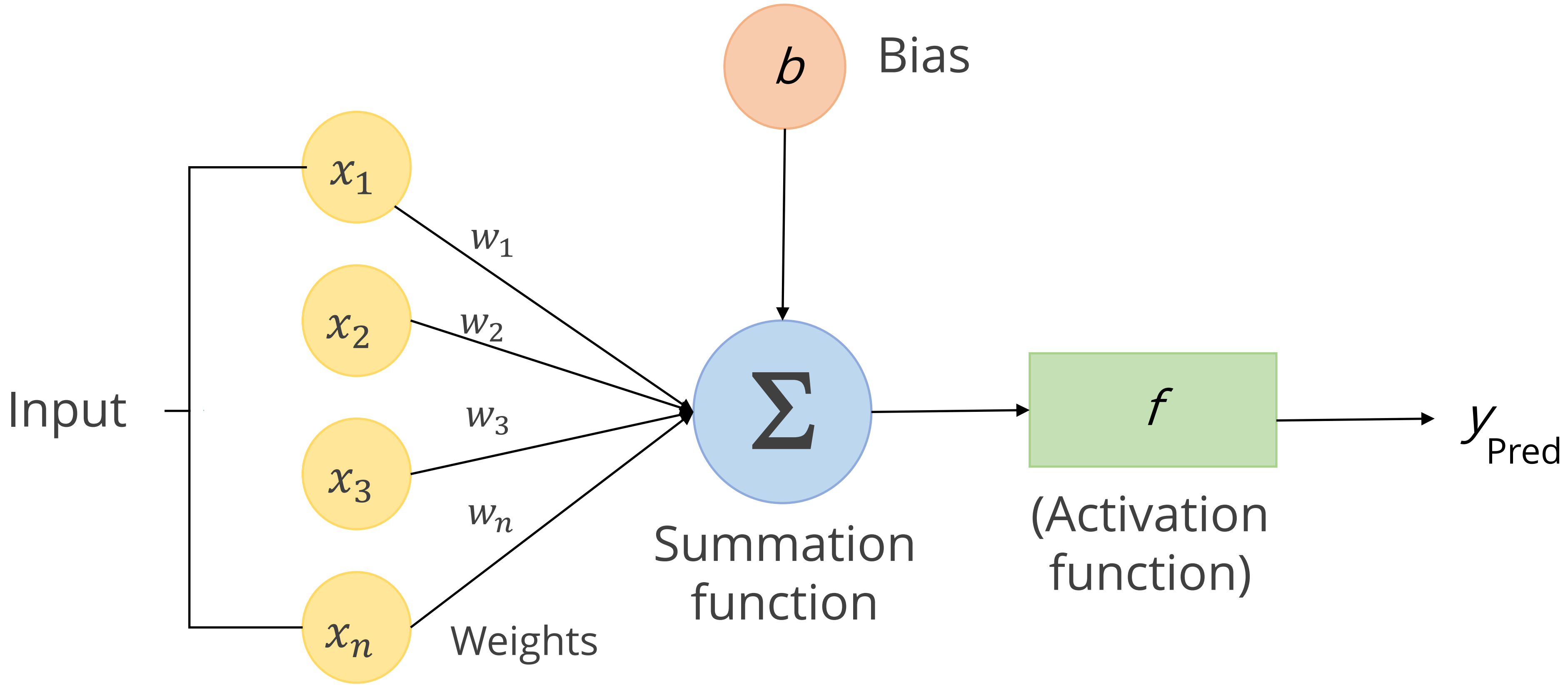
At every passage of inputs, the perceptron produces the output as either 0 or 1, which is compared to the ground truth.

The weights are adjusted accordingly for better predictions.

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

Components of a Perceptron

The following are the components of a basic artificial neuron:



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

- Inputs
- Weights
- Bias
- Summation function
- Activation function

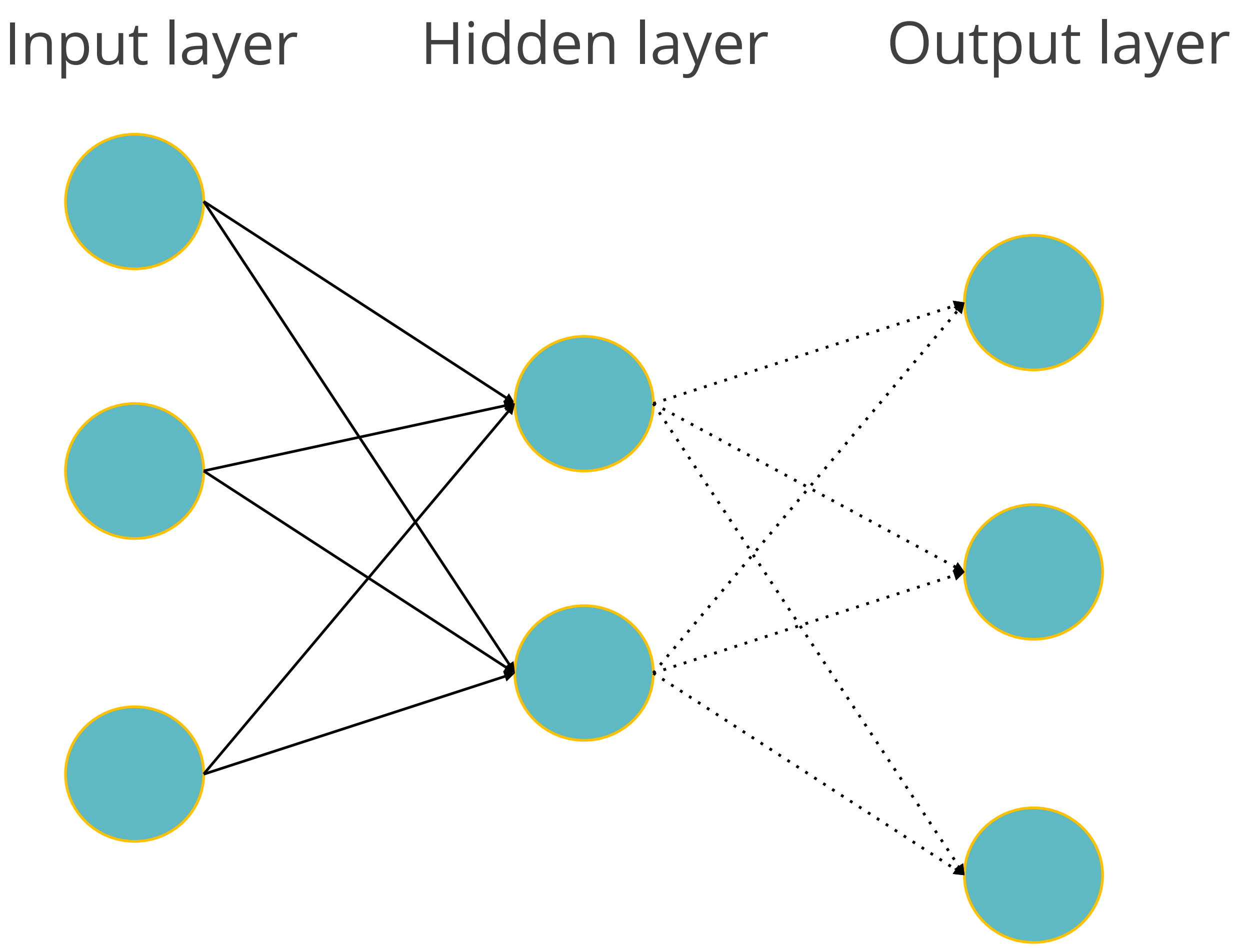
Components of a Perceptron

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

Feedforward neural networks, also known as feedforward nets, are a type of artificial neural network (ANN) where information flows only in one direction, from the input layer to the output layer, without any feedback connections.



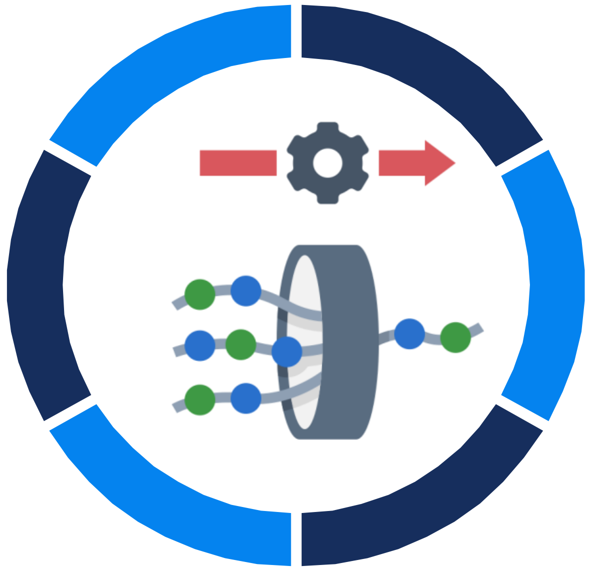
- Information flow is unidirectional.
- Information is distributed.
- Information processing is parallel.

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.

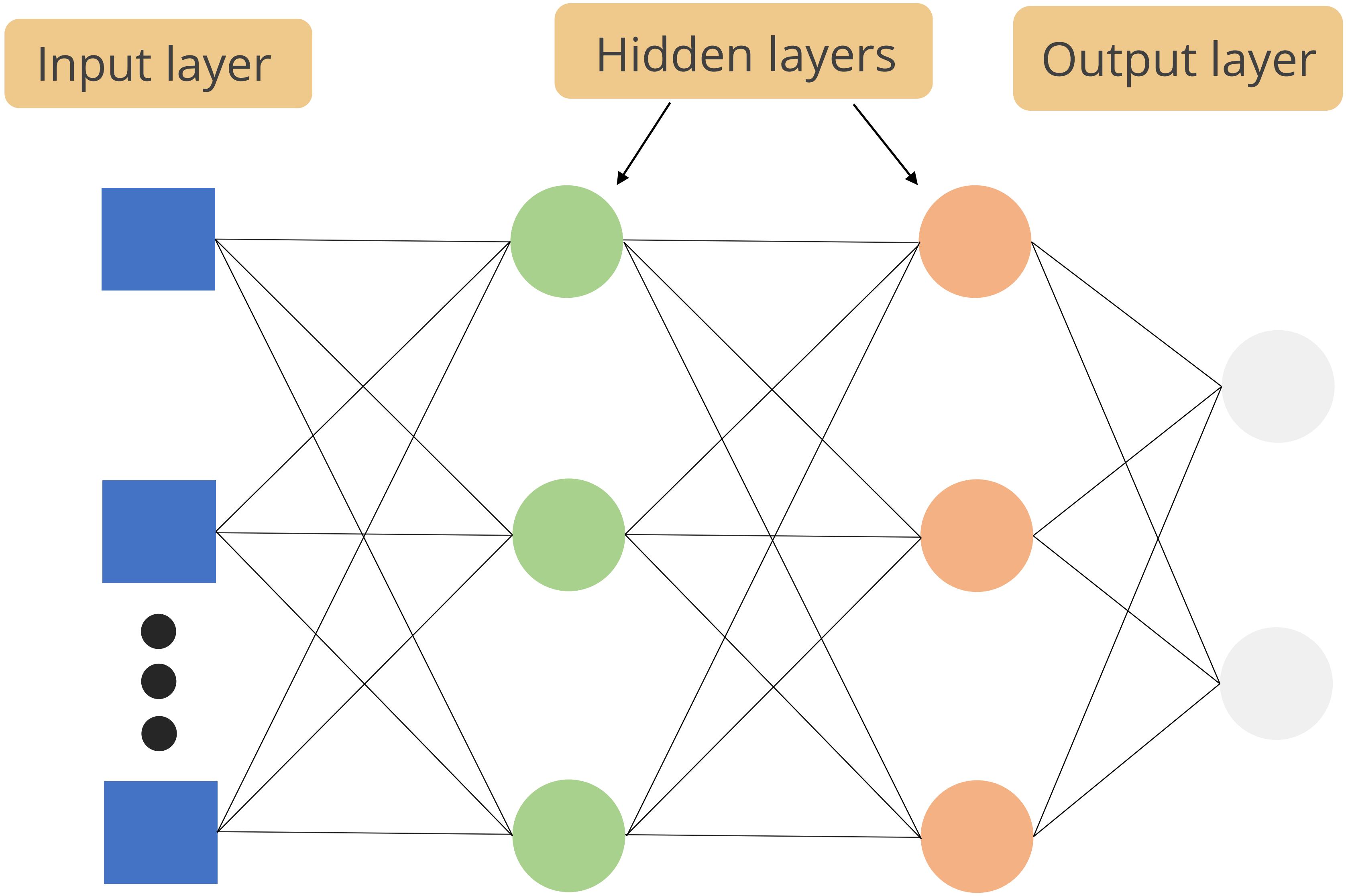


Since it only considers the current input, it has no notion of order in time.

It has no memory of the input it receives.

Multilayer Perceptrons

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



They are formed from multiple layers of the perceptron.

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,

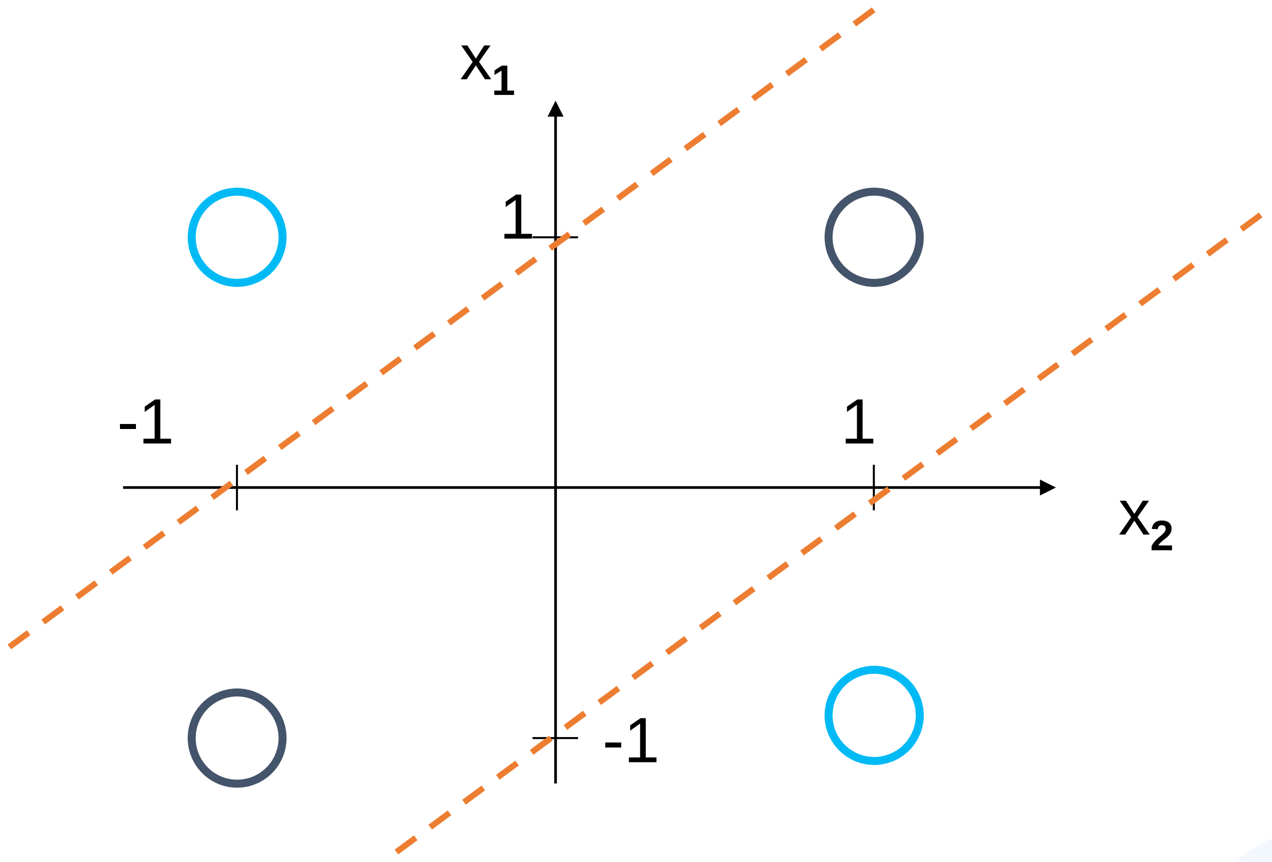
b is the bias vector,

and f is the activation function.

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



Assisted Practice

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

- 3.04_Perceptron-Based Classification Model

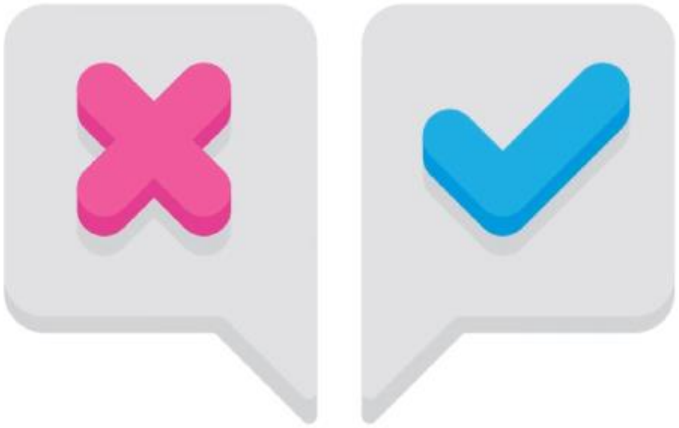
Note: Please refer to the Reference Material section to download the notebook files corresponding to each mentioned topic

ASSISTED PRACTICE

Activation Function

Activation Function

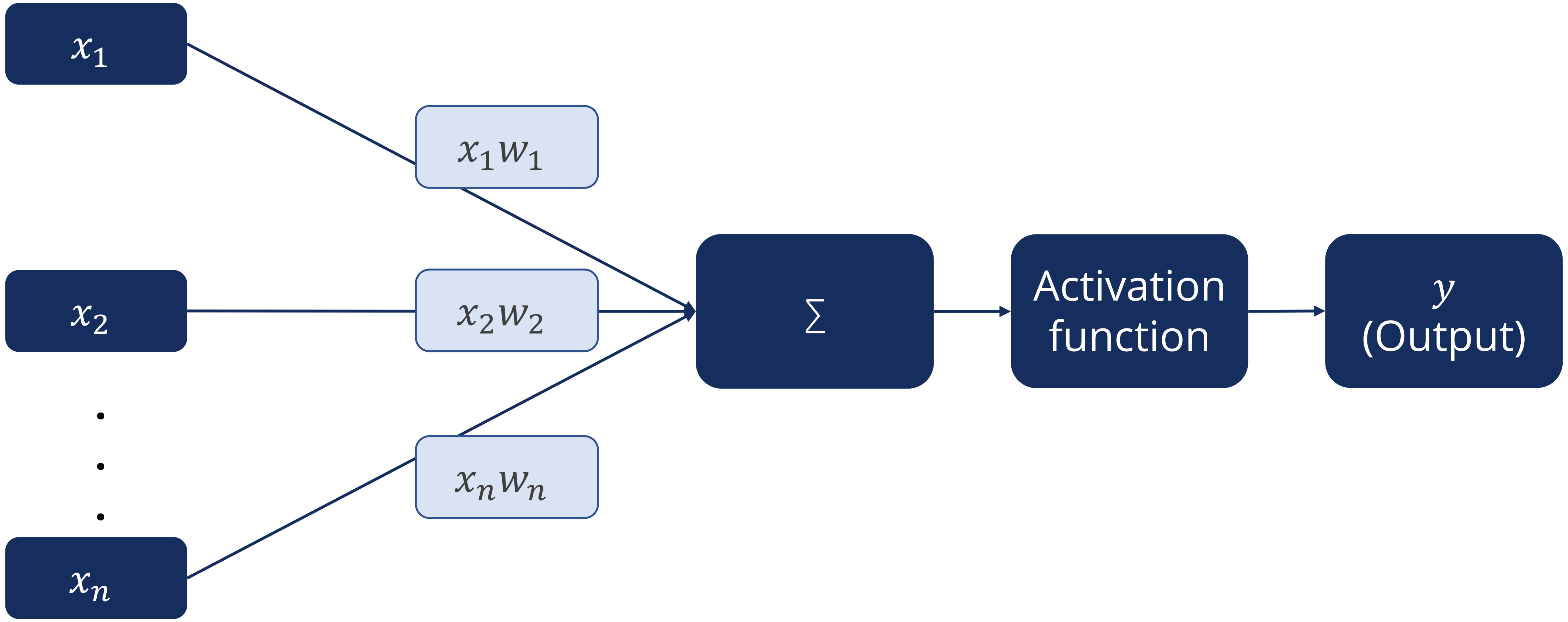
Before producing an output from a perceptron, it is important to decide whether to activate the neuron or not.



The activation function helps achieves this.

Understanding 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.

Understanding Activation Function

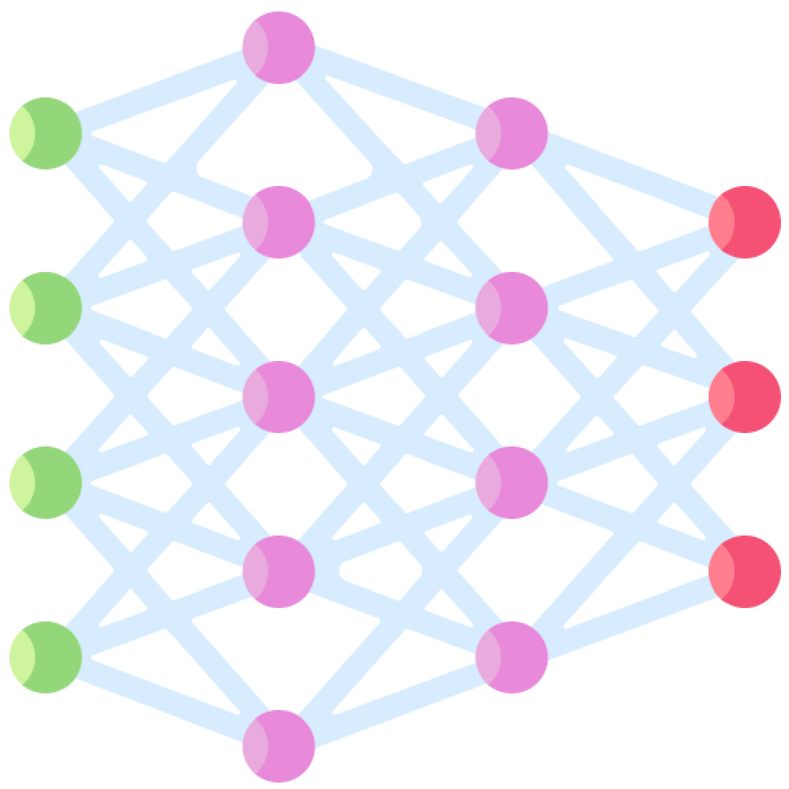
Output y = Σ(weights * input) + bias, ranging from -∞ and +∞

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

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

Need for Activation Function

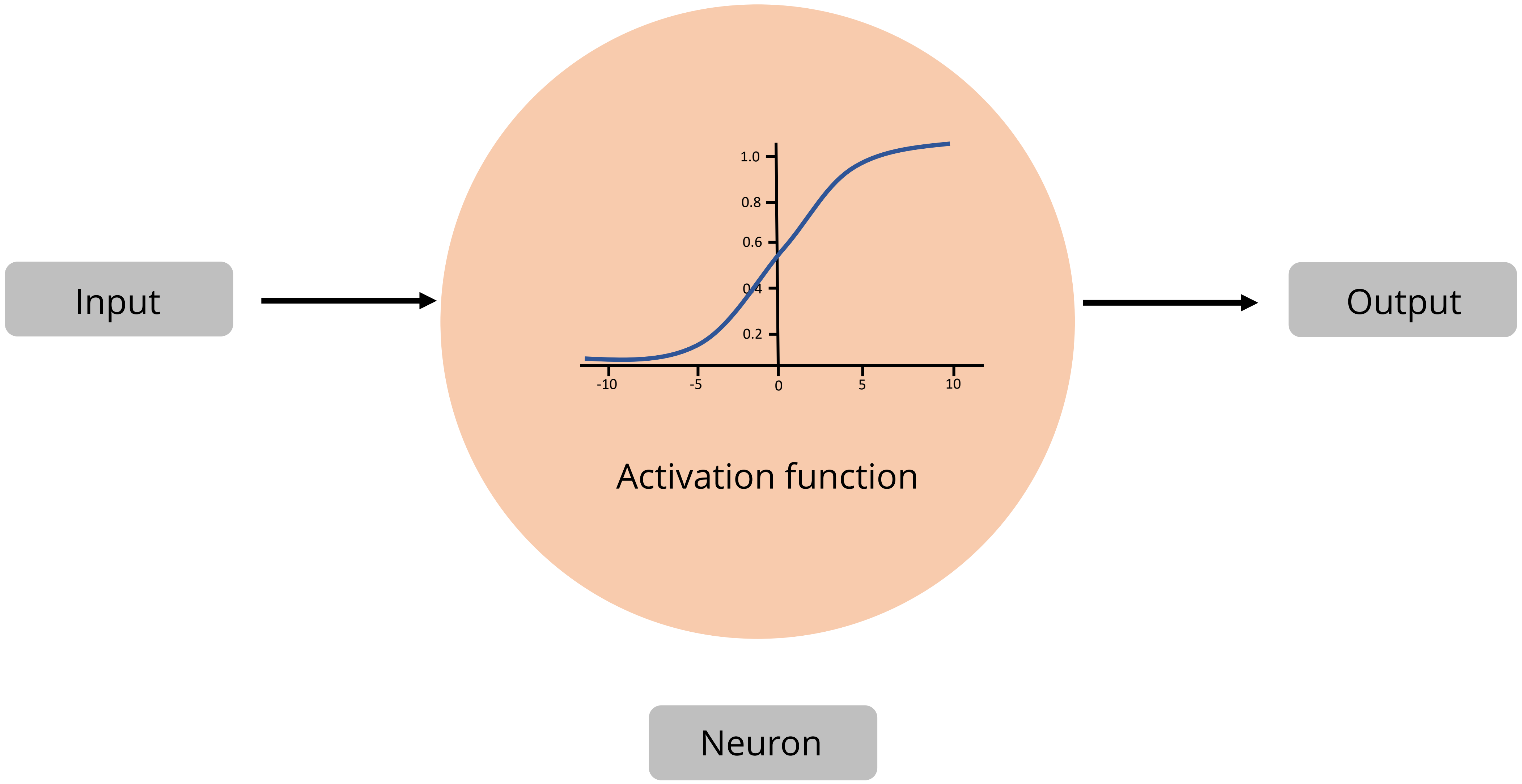
Activation functions are mathematical equations that determine the output of a neural network model.



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

Types of Activation Functions

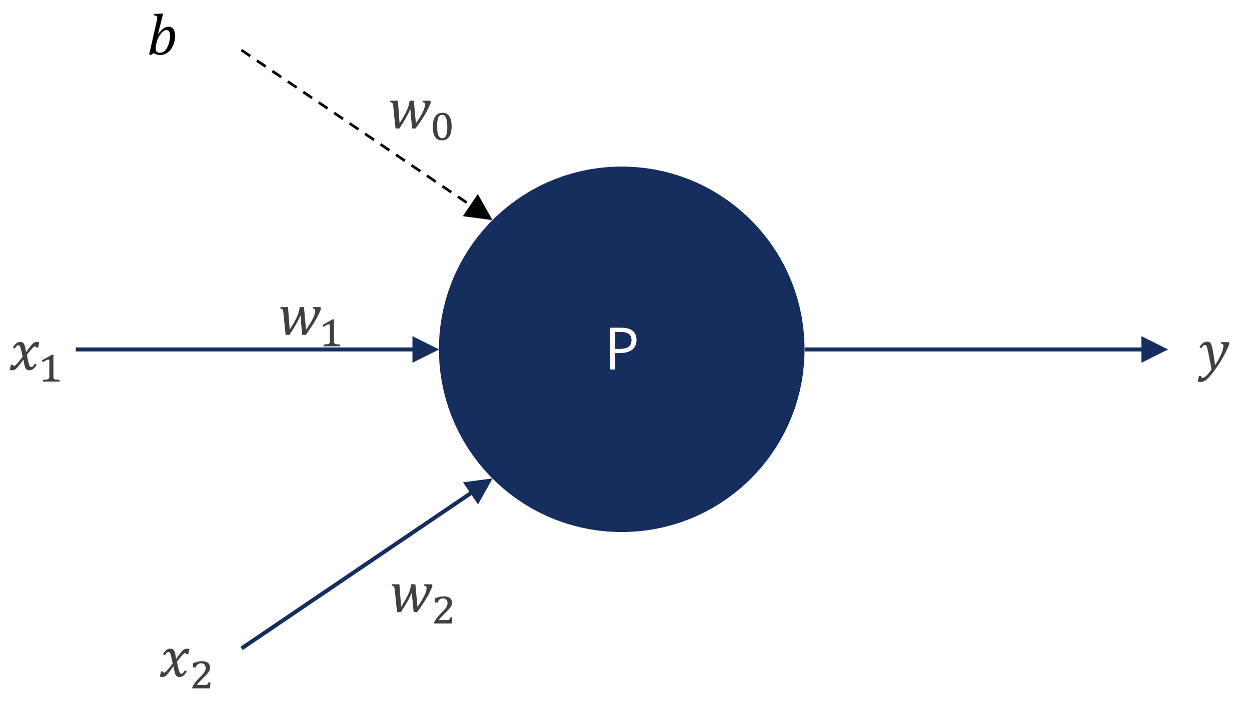
Activation functions transform inputs into outputs using a threshold value.

Different types of Activation Functions include:

- Step function
- Sigmoid function
- ReLU
- Softmax function

Activation Functions: Step Function

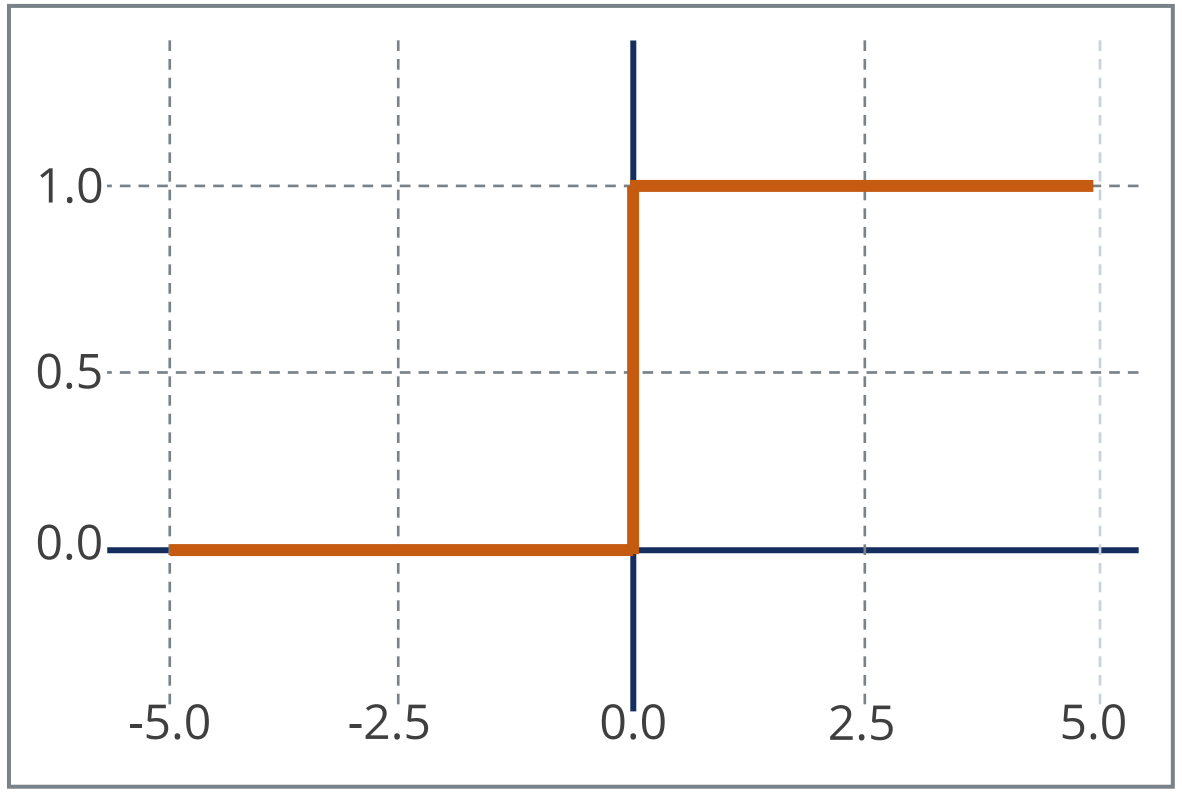
Consider a perceptron:



y = 1, when the sum of weighted inputs (x_1w_1+ x_2w_2 + bw_0) is greater than 0, else y = 0.

Activation Functions: Step Function

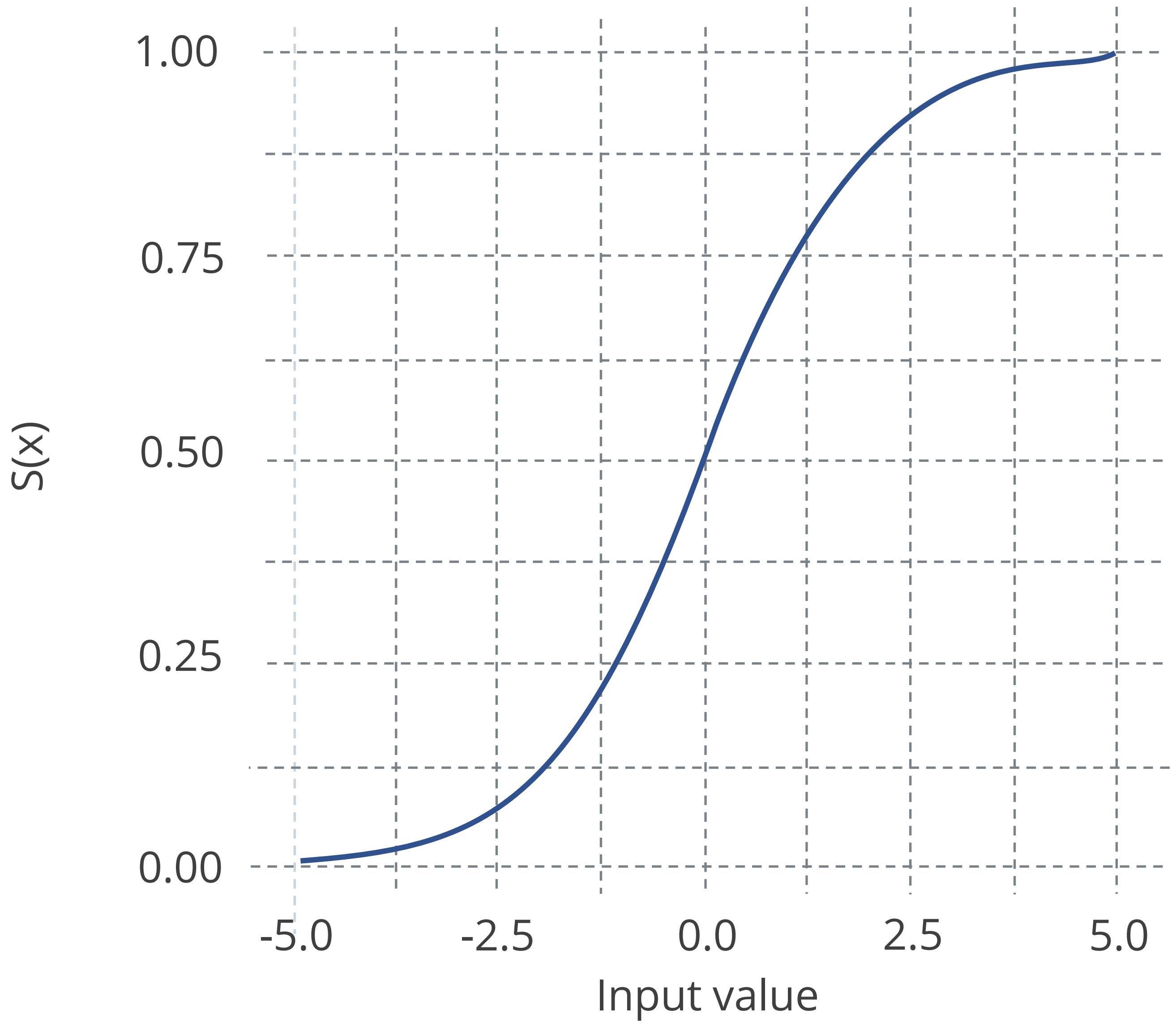
A perceptron gets activated whenever the sum of weighted inputs is non-zero and positive.



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

Activation Functions: Sigmoid Function

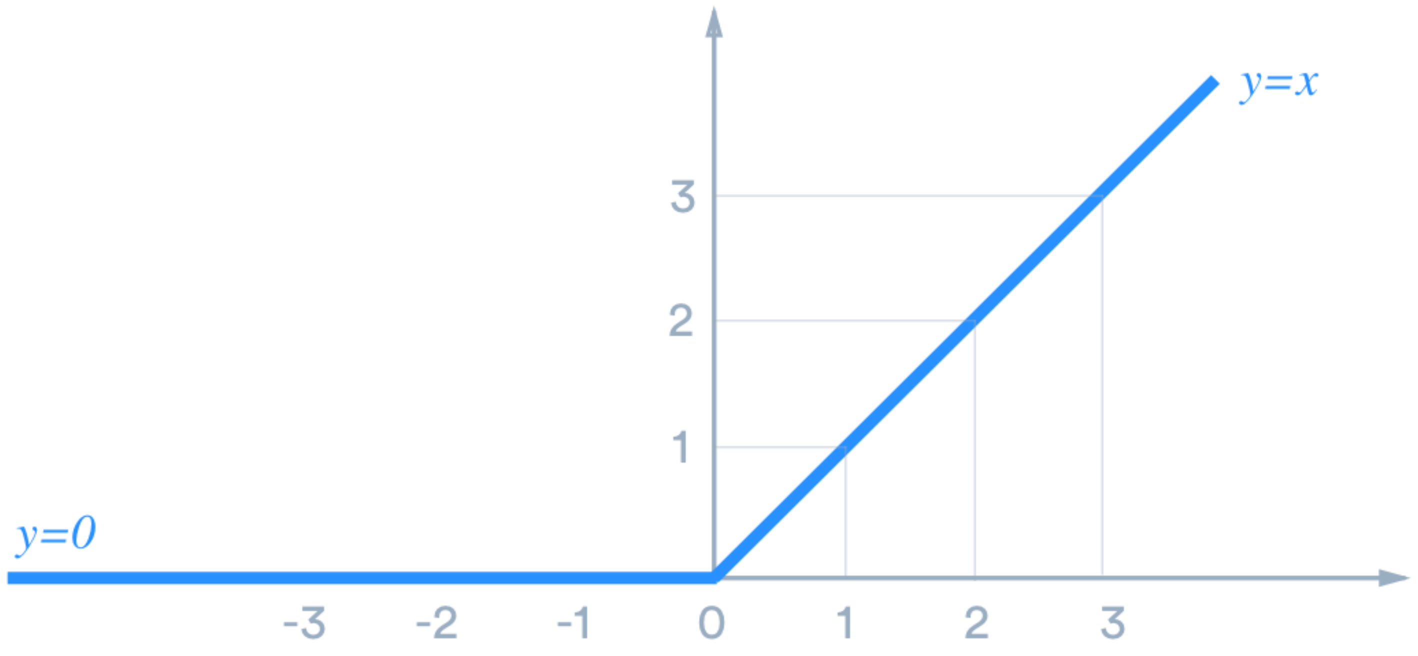
The Sigmoid activation function produces an output in the range of 0 to 1, making it useful for binary classification tasks



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.

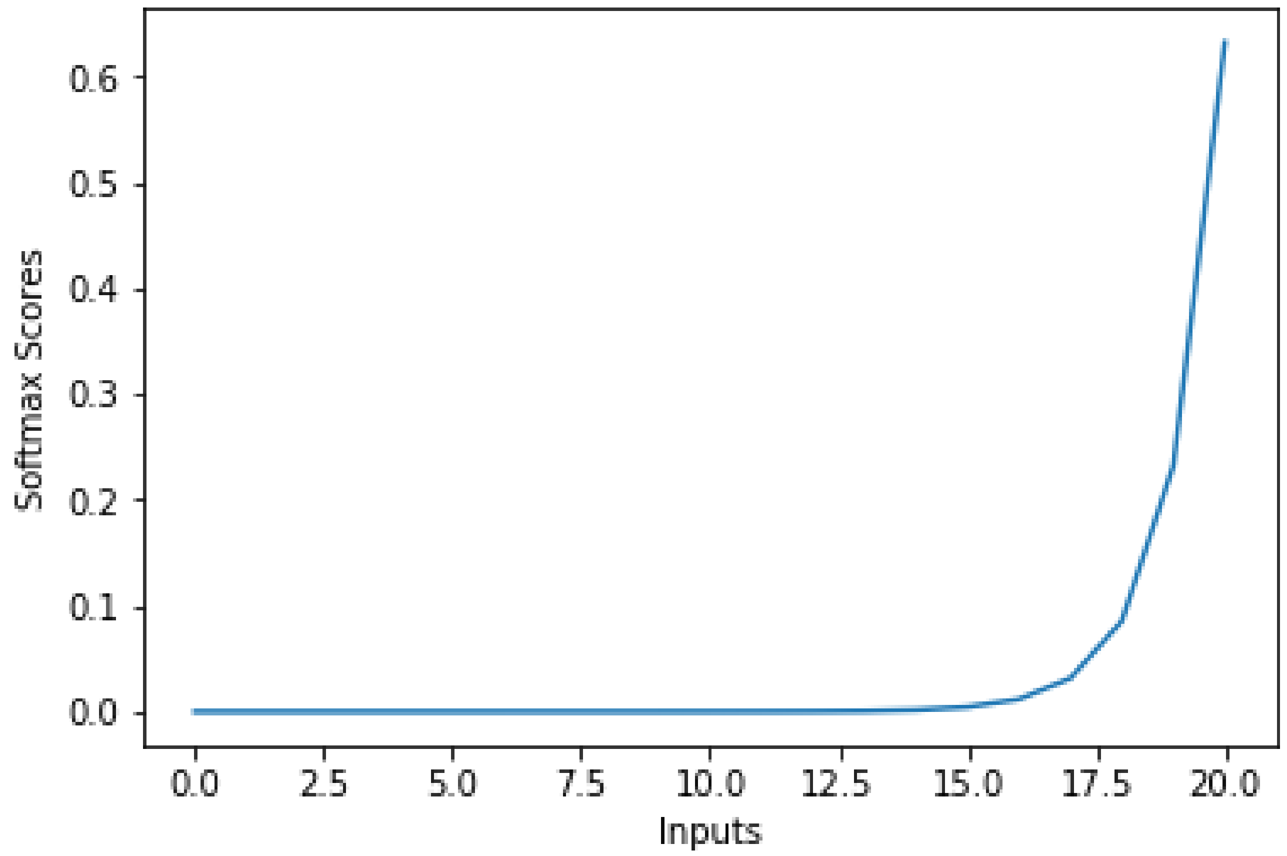
ReLU vs. Sigmoid Function

ReLU has an upper hand over the sigmoid function.

Sigmoid activation function	ReLU
When used in deep learning, it suffers from a vanishing gradient.	ReLU can avoid the vanishing gradient problem to a great extent.
It is mathematically complex.	It has a simpler calculation.

Softmax function

The softmax function, a variant of the sigmoid function, is particularly useful for handling multiclass classification problems



- The softmax function is used to handle multiple classes.
- It is commonly found in the output layer of the image classification problems.
- It normalizes outputs for each class to fall between 0 and 1.
- It achieves this by dividing each output by the sum of all outputs

Assisted Practice

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

ASSISTED PRACTICE

Forward Propagation in Perceptron

Phases of Perceptron Model

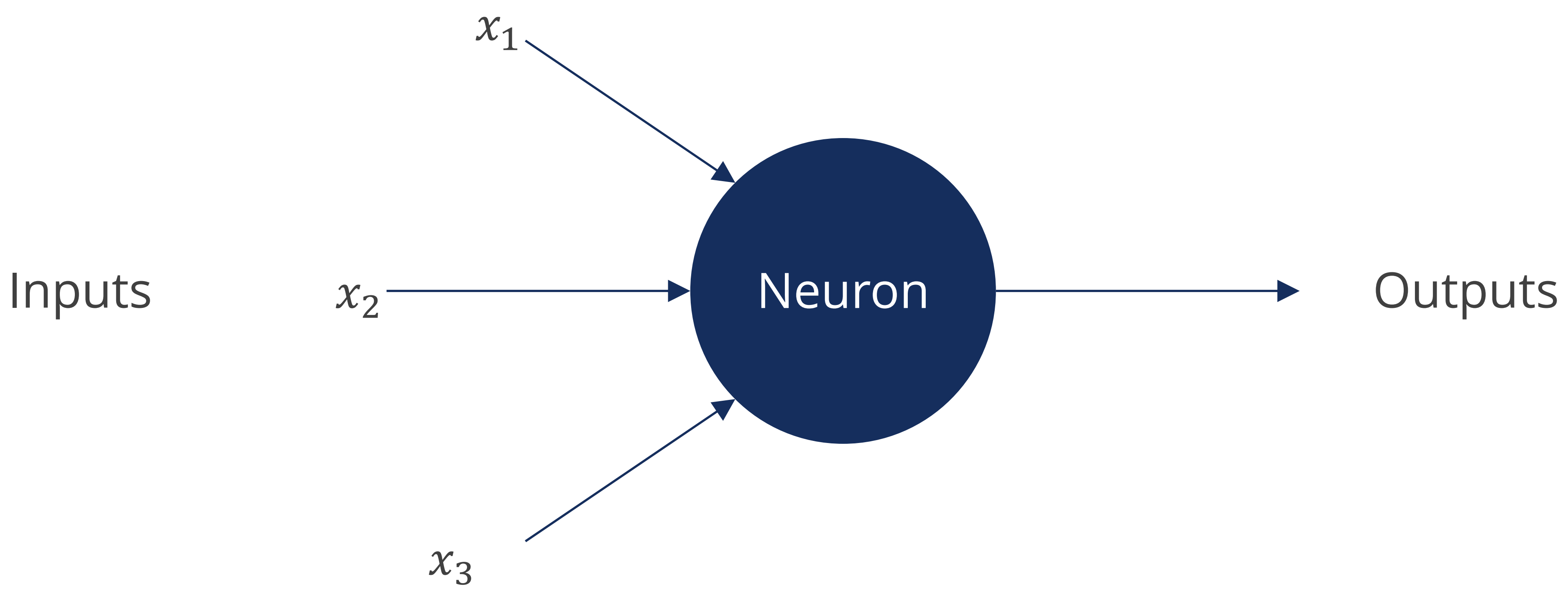
Training a perceptron model involves two phases:

Forward propagation
(Forward pass)

Backward propagation
(Backprop)

Concept of Forward Propagation

A perceptron is a type of feed-forward network.



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:

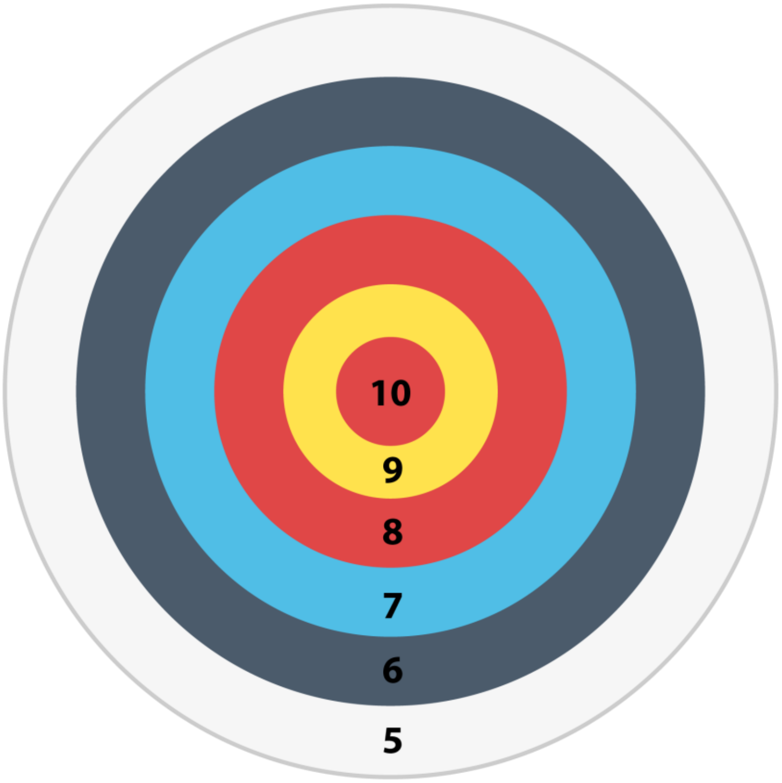
- If $x_1w_1 + x_2w_2 + b > 0$, then y = 1
- If $x_1w_1 + x_2w_2 + b <= 0$, then y = 0

The values of the weights w_1 and w_2 are randomly initialized, and the objective is to find the right set of weights for the best solution.

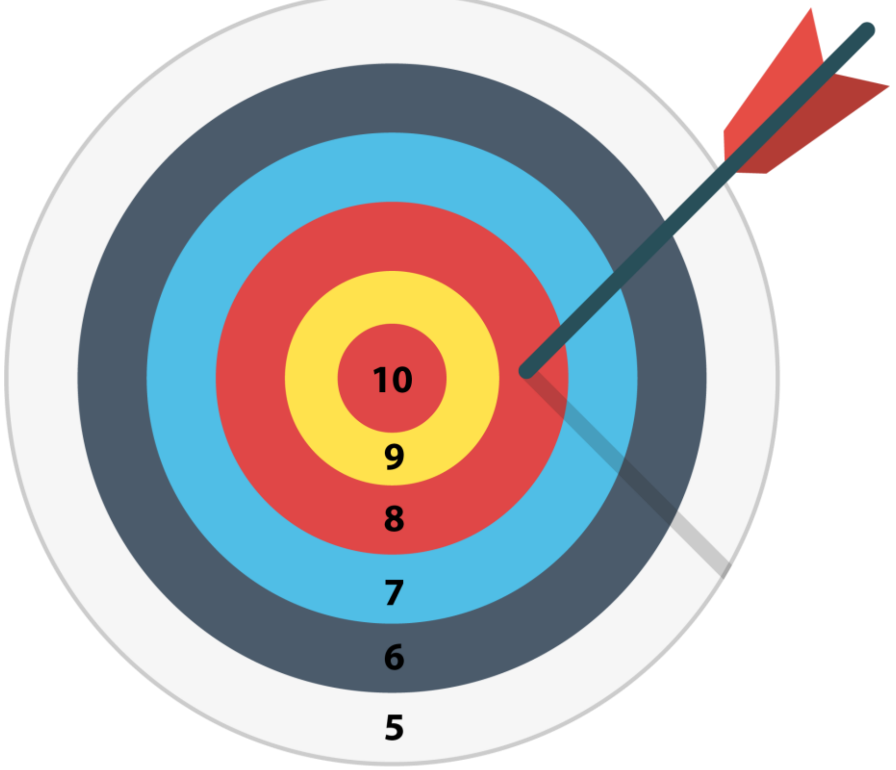
Loss Function and Cost Function

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, 8 minus 10 = -2.

Loss Function: Definition and Formula

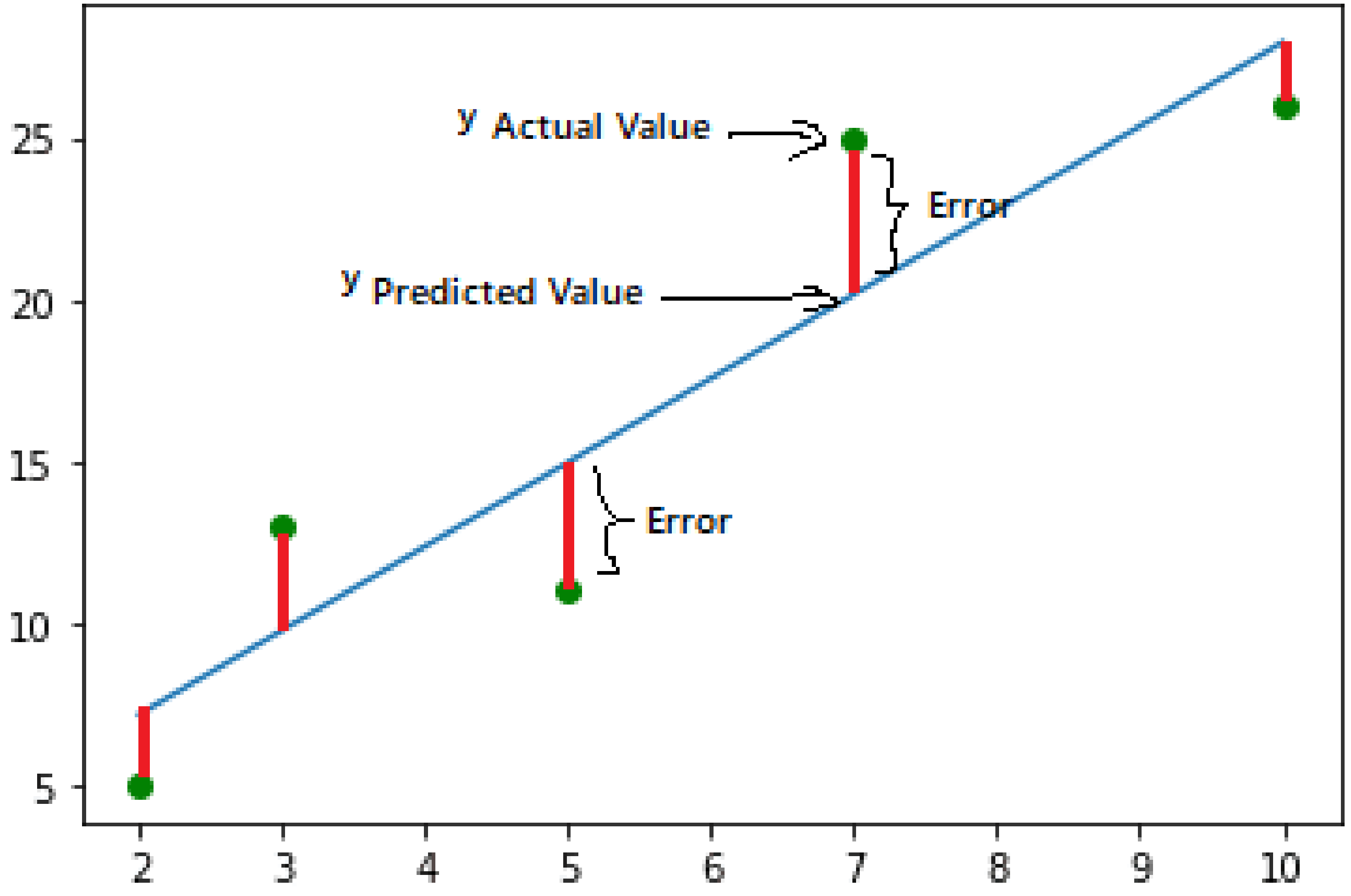
The loss function, also known as the cost function or objective function, measures the discrepancy between the predicted outputs of a machine learning model and the true values of the training data.

$$Loss\ function\ (MSE) = \frac{1}{n}\sum_{i=0}^{n}(y_{true,i} - ypr_{ed,i})^2$$

- n is the number of data points in the training set.
- $y_{true,i}$: represents the true values of the target variable.
- $y_{pred,i}$:represents the predicted values of the target variable by the model.

What Is Cost Function?

The cost function aggregates the difference for the entire training dataset.



To measure the model's accuracy, compare the predicted results with the actual values. The greater the discrepancy between these two, the higher the error metric will be.

Need of Cost Function

The cost function is necessary for implementing gradient descent, and it should be minimized as much as possible by changing the values of the parameters.

$$Cost\ function\ (MSE) = \frac{1}{n}\sum_{i=0}^{n}(y_{true,\ i} - ypr_{ed,\ i})^2$$

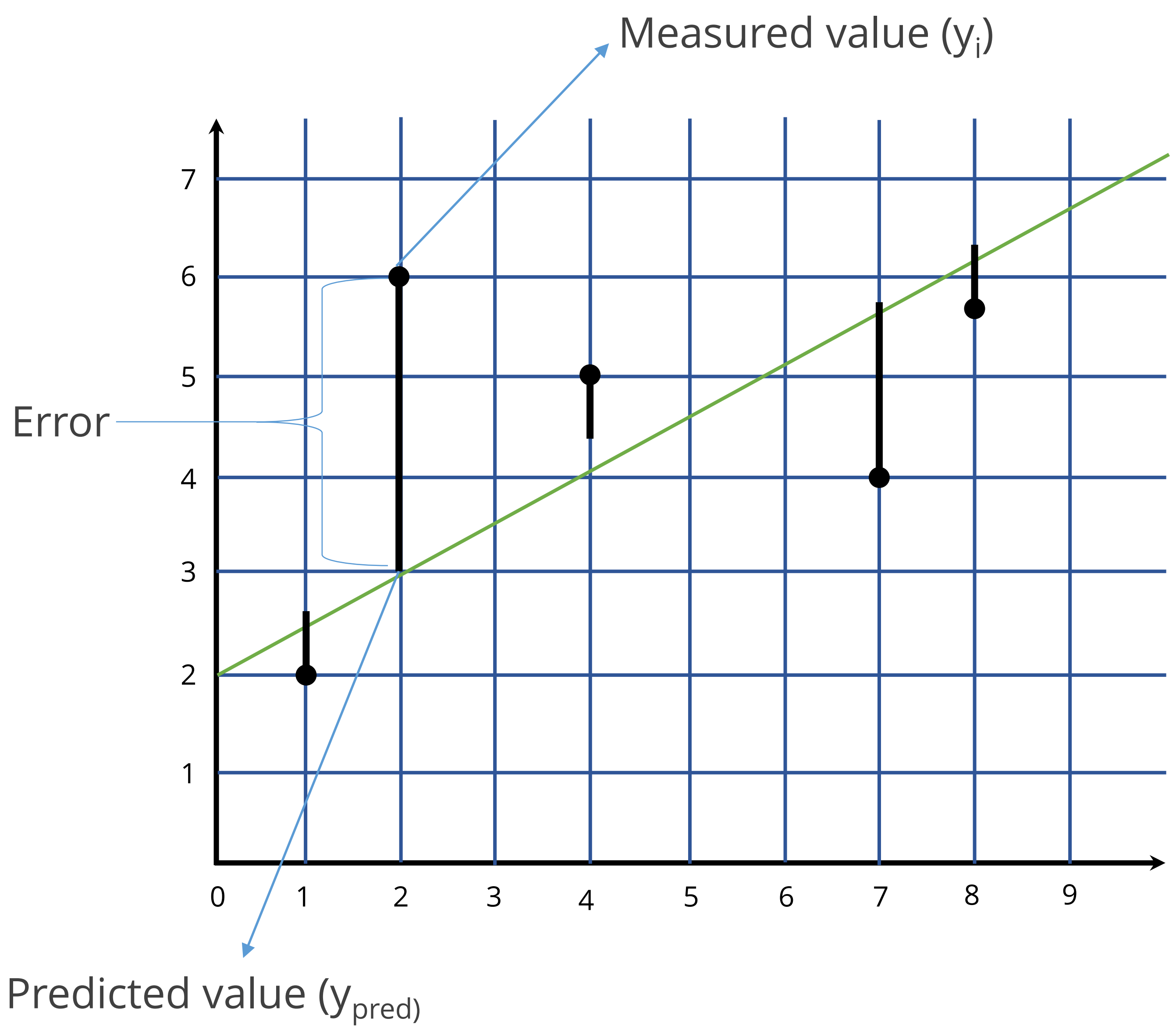
$$Replace\ y_{pred,\ i}\ with\ the\ linear\ regression:$$
$$mxi + c$$

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

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.

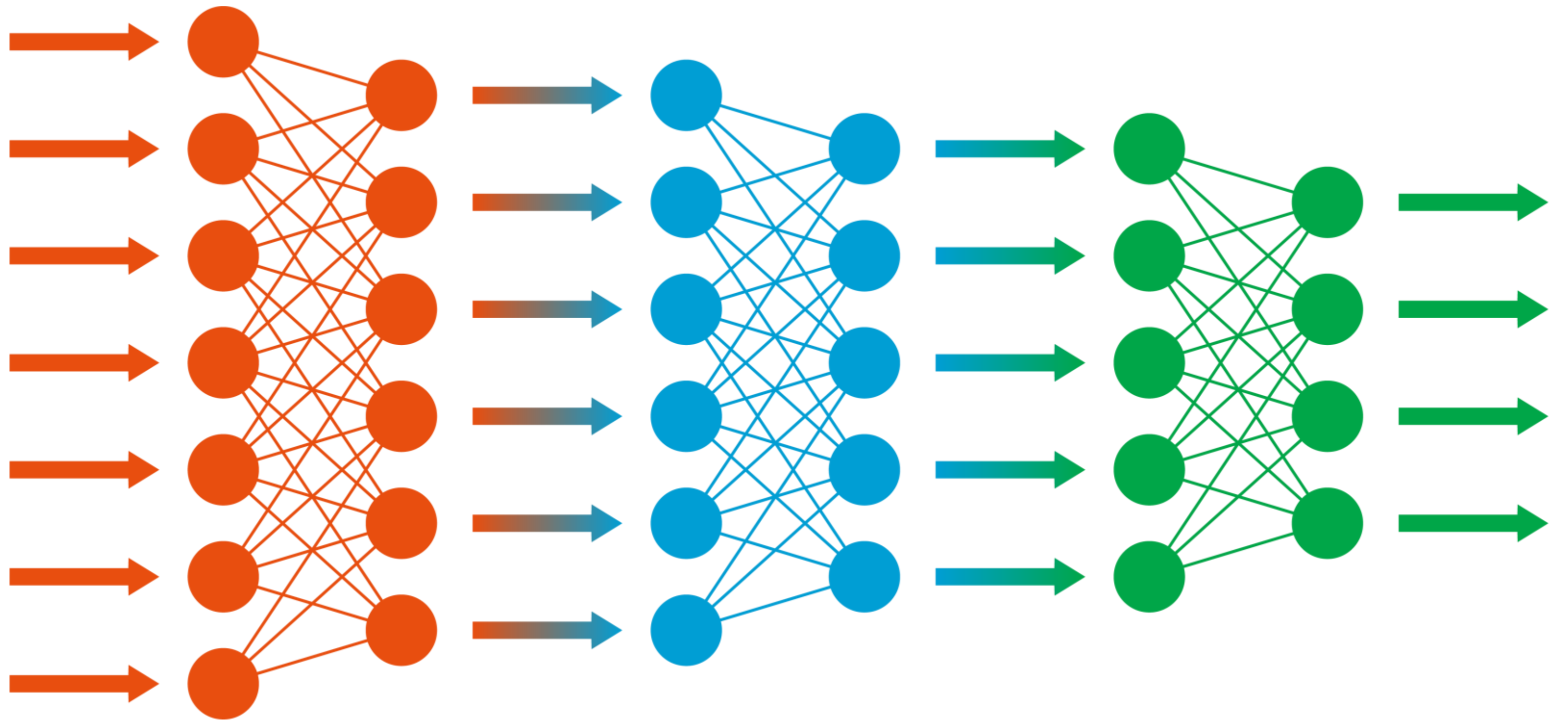


Gradient descent optimizes linear regression by iteratively adjusting parameters (slope and y-intercept) to find the best-fit line when an exhaustive search is impractical.

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.

1. **Initialize the weights and the threshold**

   Let w_i be the initial weight

2. **Provide the input and calculate the output**

   Let x be the input and y be the output

3. **Update the weights**

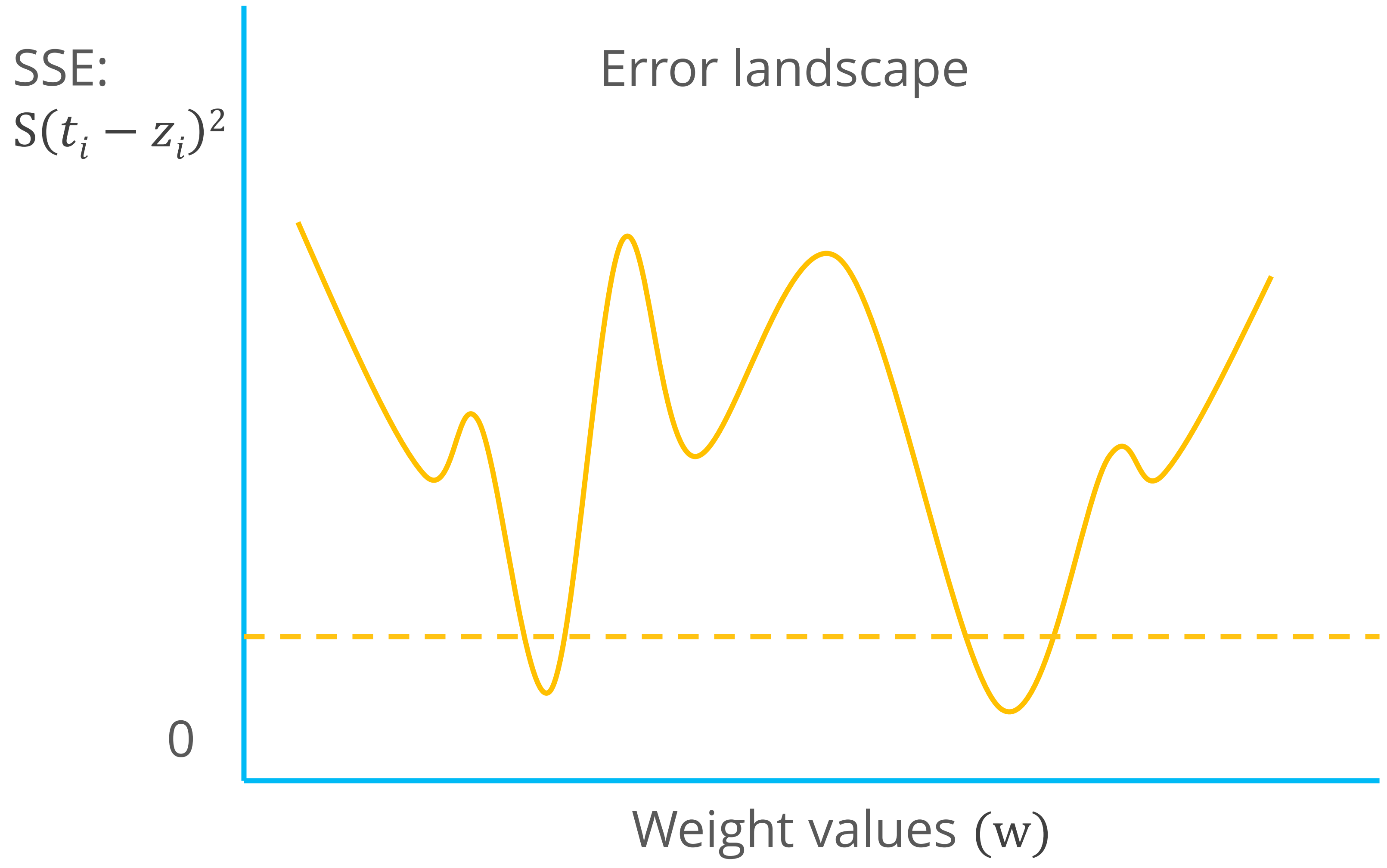
   $w_i(t+1) = (w_i)t + n(d-y)x$, where d is the desired output, n is the learning rate, and y is the actual output.

4. **Repeat the initial steps**

   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 of squared error (SSE).



$S(t_i - z_i)^2$ 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_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 any further.

This is the concept of backpropagation.

Backpropagation

Weights are updated in the following manner in each iteration:

$$w_1 = w_1 + \eta \text{ . error. } x_1$$

$$w_2 = w_2 + \eta \text{ . error. } x_2$$

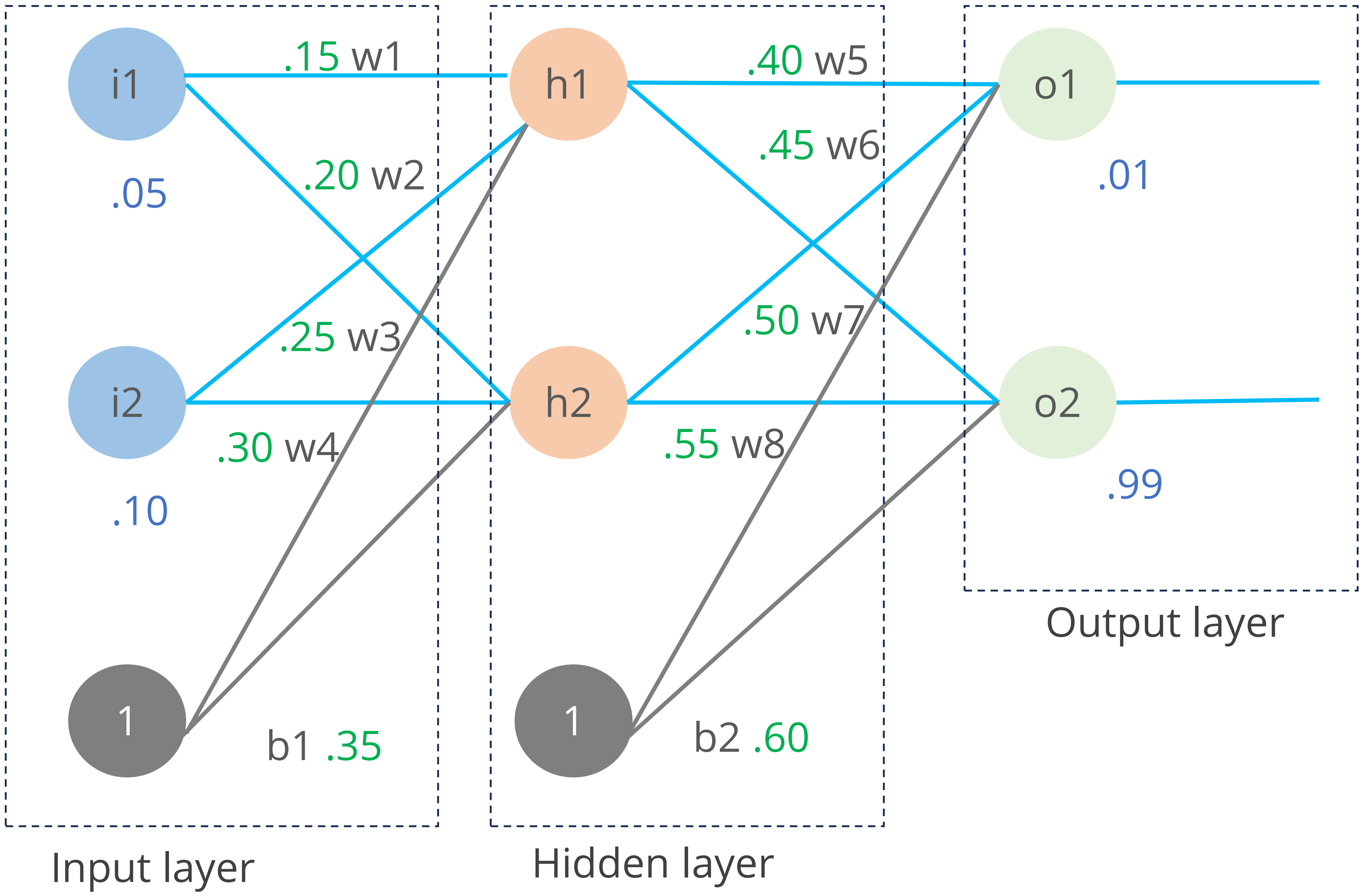
- η = Learning rate, ranging from 0 to 1
- Error = Difference between the desired output and actual output
- w_1,and w_2 = Weight parameters
- x_1 and x_2 = Input variables

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.

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:

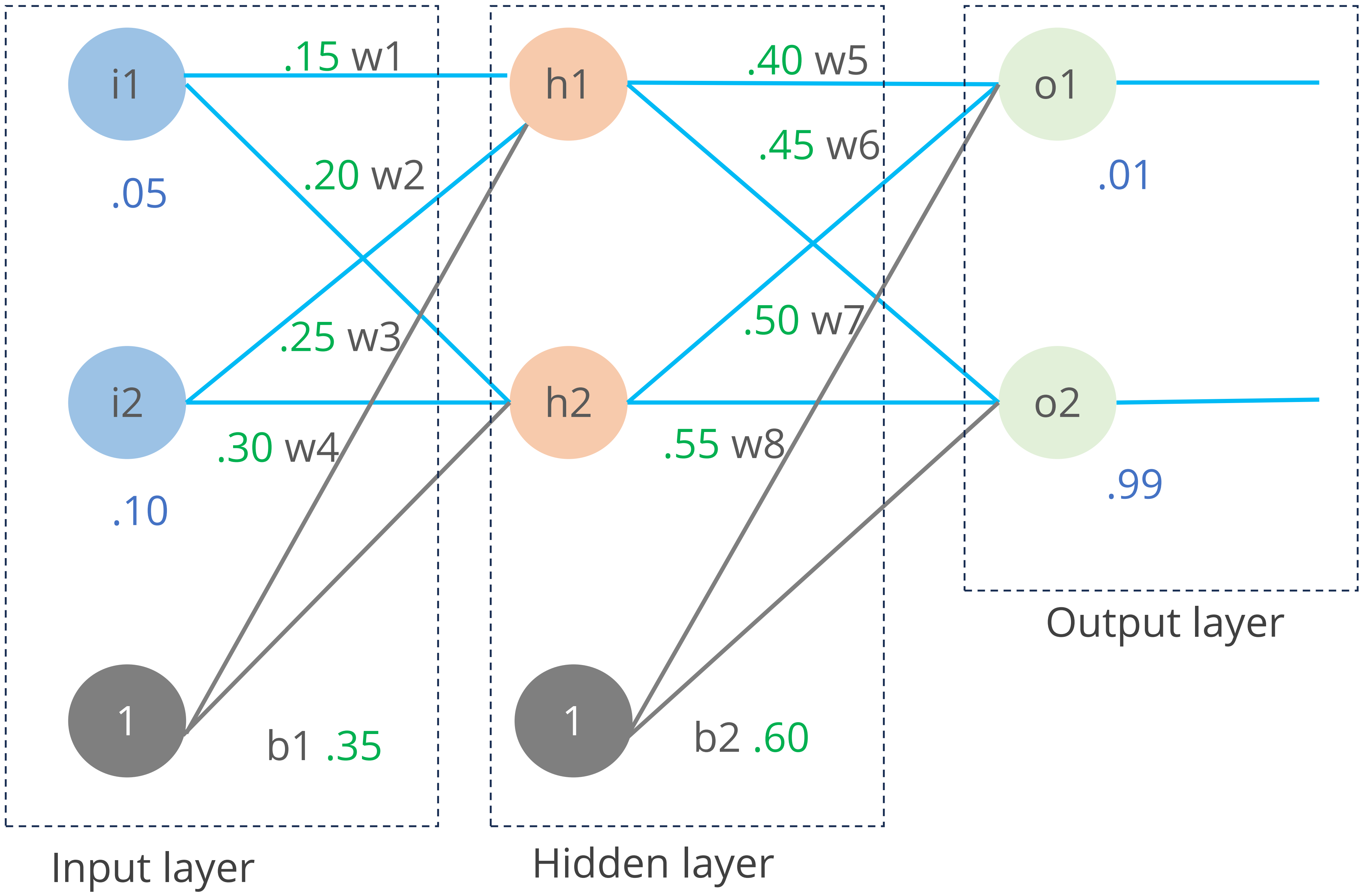


Input Layer

1. Two input neurons: i1 and i2.
2. Values for inputs: i1=0.05, i2=0.10.

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:

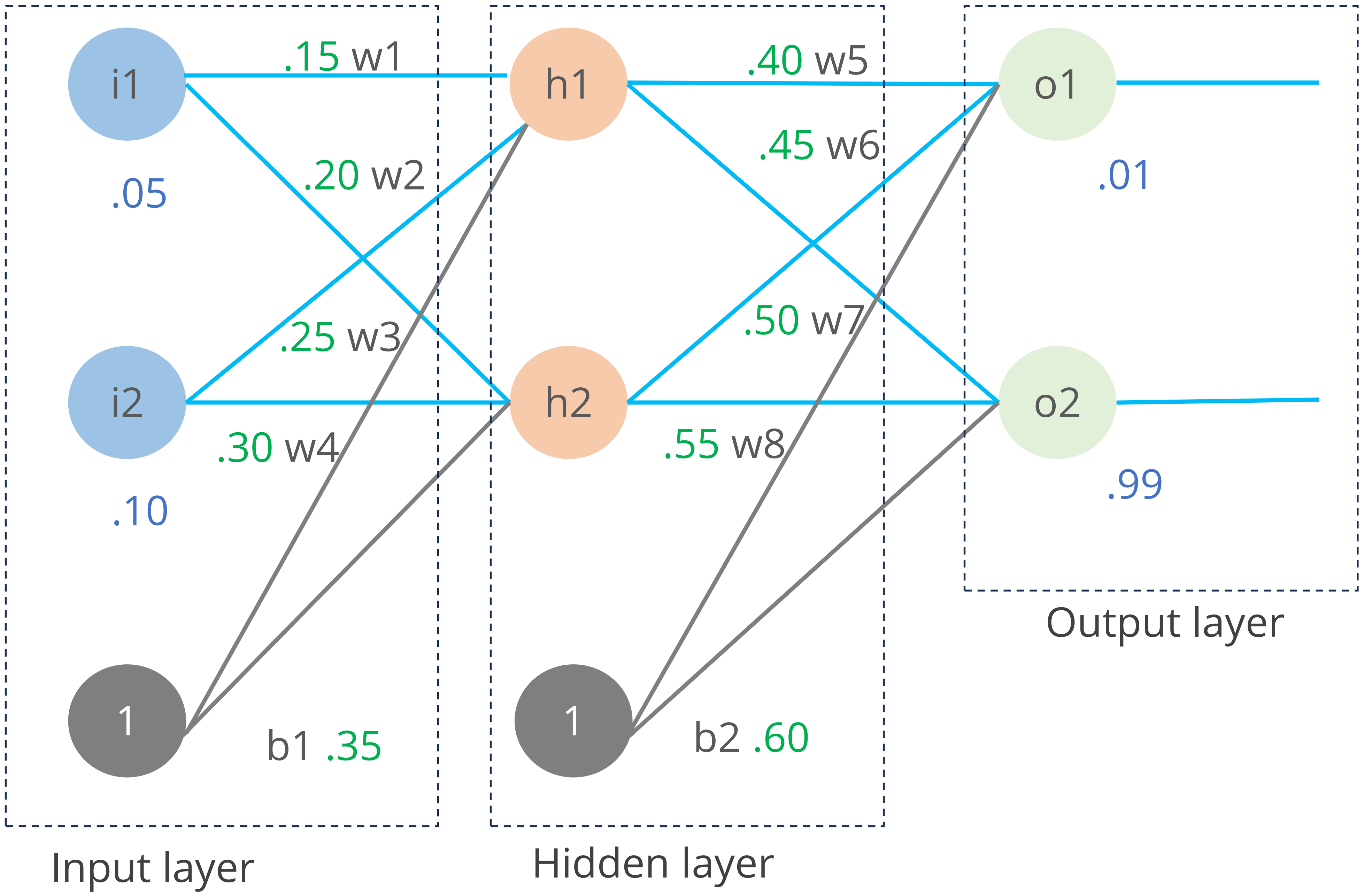


Hidden Layer

1. Two hidden neurons: h1 and h2.
2. Bias for hidden neurons: b1=0.35.
3. Weights from input to hidden layer:
   1. w1=0.15, w2=0.20 (connecting i1 to h1 and h2).
   2. w3=0.25, w4=0.30 (connecting i2 to h1 and h2).

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:



Output Layer

1. Two output neurons: o1 and o2.
2. Bias for output neurons: b2=0.60.
3. Weights from hidden to output layer:
   1. w5=0.40, w6=0.45 (connecting h1 to o1 and o2).
   2. w7=0.50, w8=0.55 (connecting h2 to o1 and o2)

Forward Pass

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

Calculate the total net input for hidden layer $h1$:

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

Using the given weights and inputs:

$$net_{h1} = 0.15 * 0.05 + 0.25 * 0.1 + 0.35 * 1 = 0.3775$$

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

Forward Pass

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

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

$$out_{h2} = 0.59869$$

Carrying out the same process for h_2, you get: 0.59869

Forward Pass

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

$$net_{o1} = w_5 * out_{h1} + w_7 * out_{h2} + b_2 * 1$$

$$net_{o1} = 0.4 * 0.593269992 + 0.45 * 0.596884378 + 0.6 * 1 = 1.105905967$$

$$out_{o1} = 1/1 + e^{-net_{o1}} = 0.7571$$

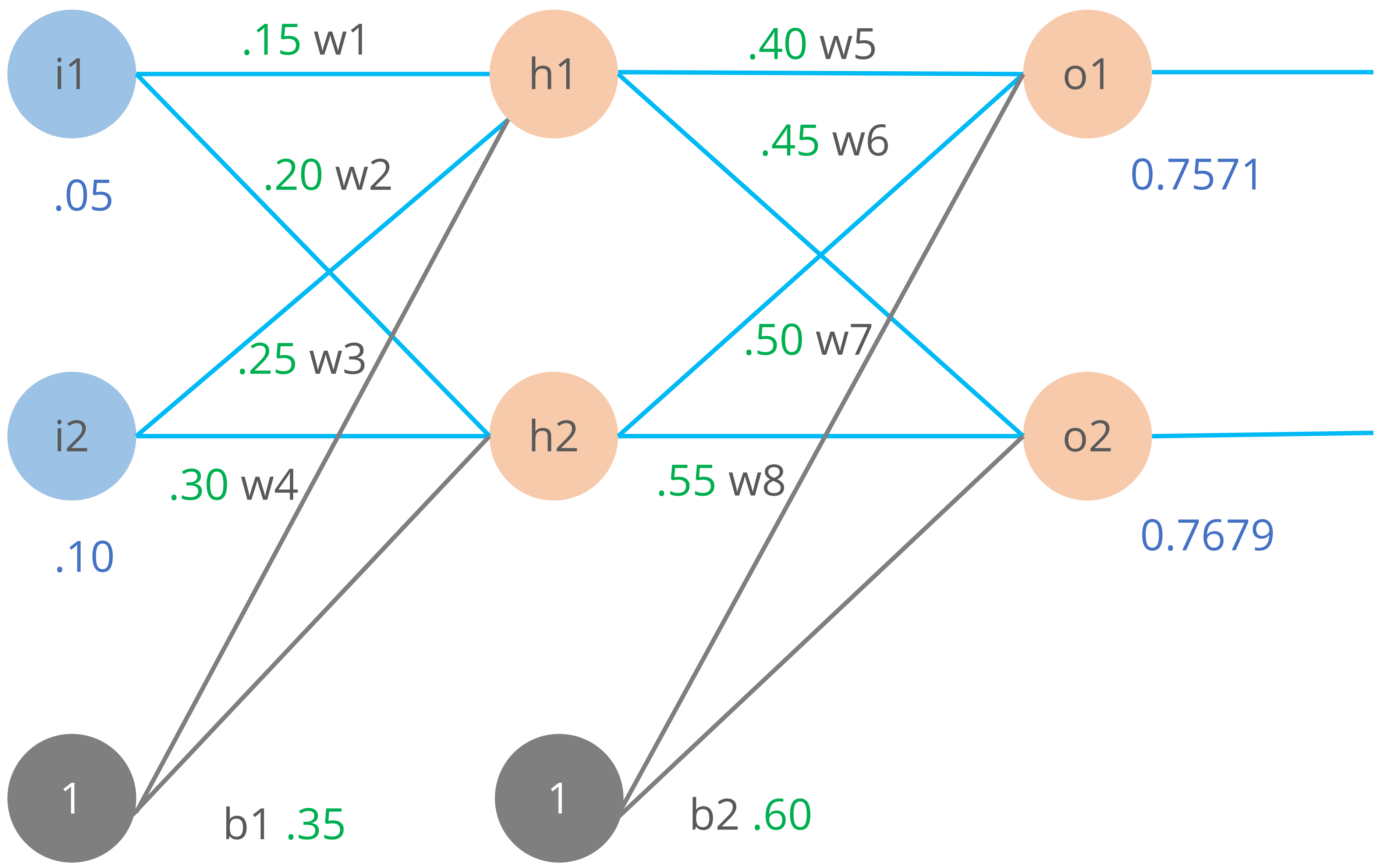
Output for o1

Carrying out the same process for o2, you get:

$$out_{o2} = 0.7679$$

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:

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

$For\ example, the\ target\ output\ for\ o_1\ is\ 0.01\ but\ the\ neural\ network\ output\ is\ 0.75136507, therefore\ its\ error\ is$:

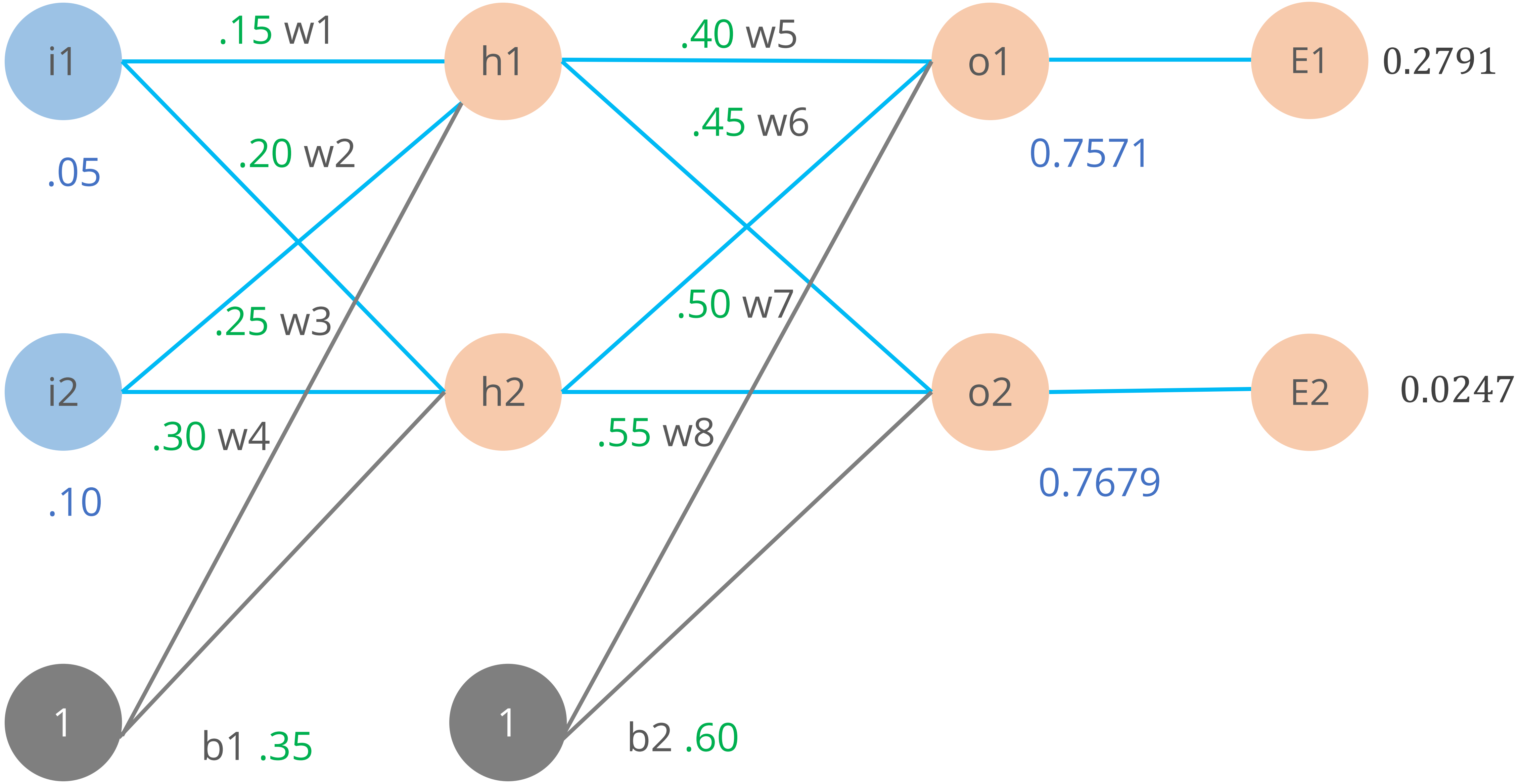
$$E_{o1} = 1/2 \ (target_{o1} \_ \ out_{o1})^2 = 1/2(0.01 - 0.7571)^2 = 0.2791$$

$$E_{o2} = 0.0247$$

Calculating Total Error

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

$$E_{Total} = E_{o1} + E_{o2} = 0.2791 + 0.0247 = 0.3038$$



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 (E_{Total}) *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:

Error calculation:

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

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

Activation function:

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

Net input calculation:

$net_{o1} {}_{=} w_5 * out_{h1} {}_{+} w_7 * out_{h2} {}_{+} b_2 * 1$

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 $\partial E_{Total}/\partial out_{01}$

$$\frac{\partial E_{Total}}{\partial out_{01}} = 2 * 1/2(target_{o1}\,\_\,out_{01}) * -1$$

$$\frac{\partial E_{Total}}{\partial out_{01}} = -(target_{01} - out_{01}) = -(0.01 - 0.7571) = 0.7471$$

-(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?

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

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

Calculate $\partial out_{01}/\partial net_{01}$:

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

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

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

$$\frac{\partial out_{01}}{\partial net_{01}} = 0.7571(1 - 0.7571) = 0.1839$$

Backward Pass

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

$net_{01} = w_5 * out_{h1} + w_7 * out_{h2} + b_2 * 1$

Calculate $\partial net_{01} / \partial w_5$:

$$\frac{\partial net_{01}}{\partial w_5} = out_{h1} = 0.5933$$

Putting it all together:

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

$$\frac{\partial E_{Total}}{\partial w_5} = 0.7471 * 0.1839 * 0.5933 = 0.0815$$

Updated Weight

To decrease the error, subtract this value from the current weight.

$$w_5^+ = w_5 - \eta * \frac{\partial E_{Total}}{\partial w_5} = 0.4 - 0.5 * 0.0815 = 0.35925$$

Similarly, $w6$, $w7$, and $w8$ are provided with their respective values:

$$w_6^+ = 0.4617$$

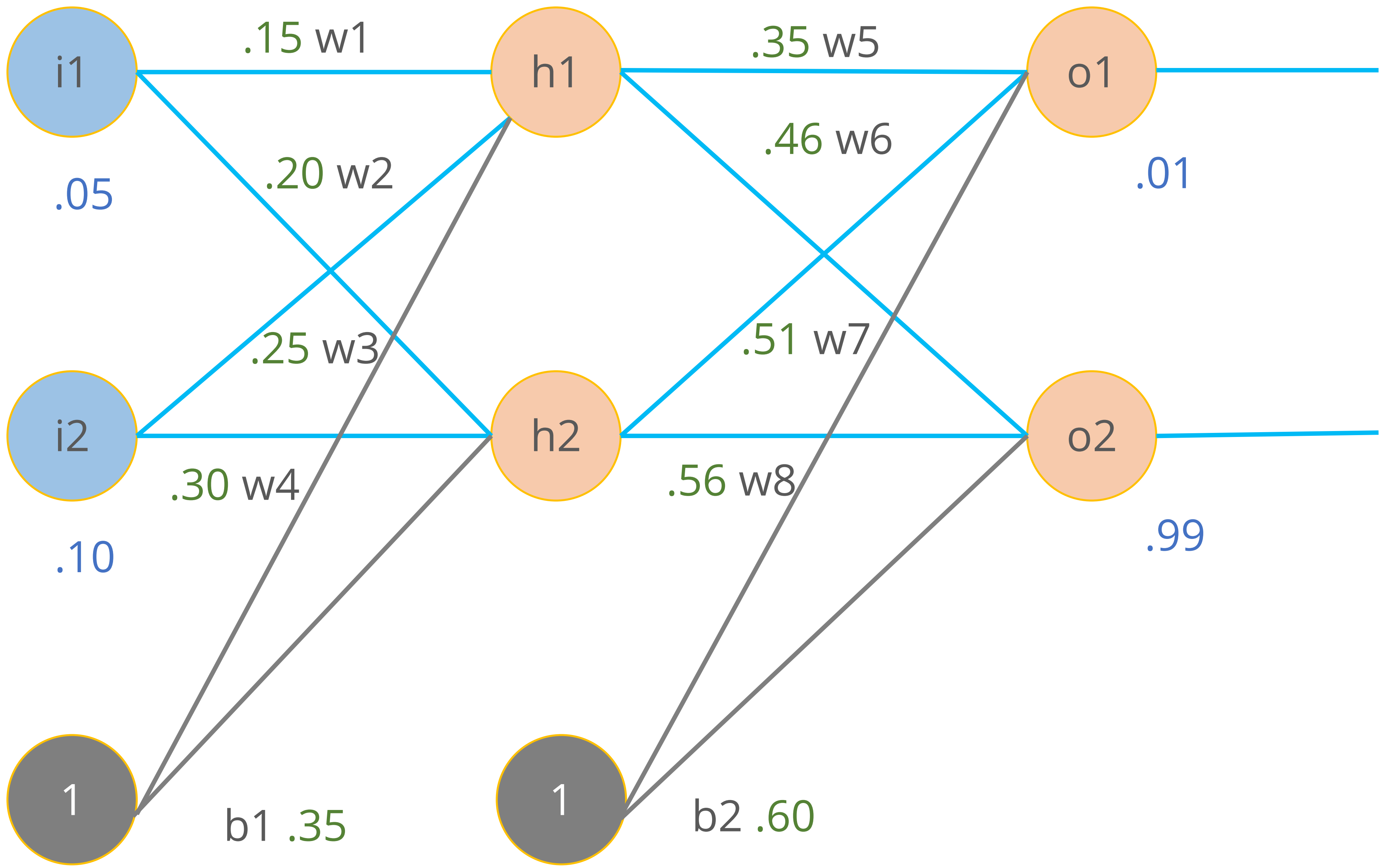
$$w_7^+ = 0.51183$$

$$w_8^+ = 0.56183$$

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

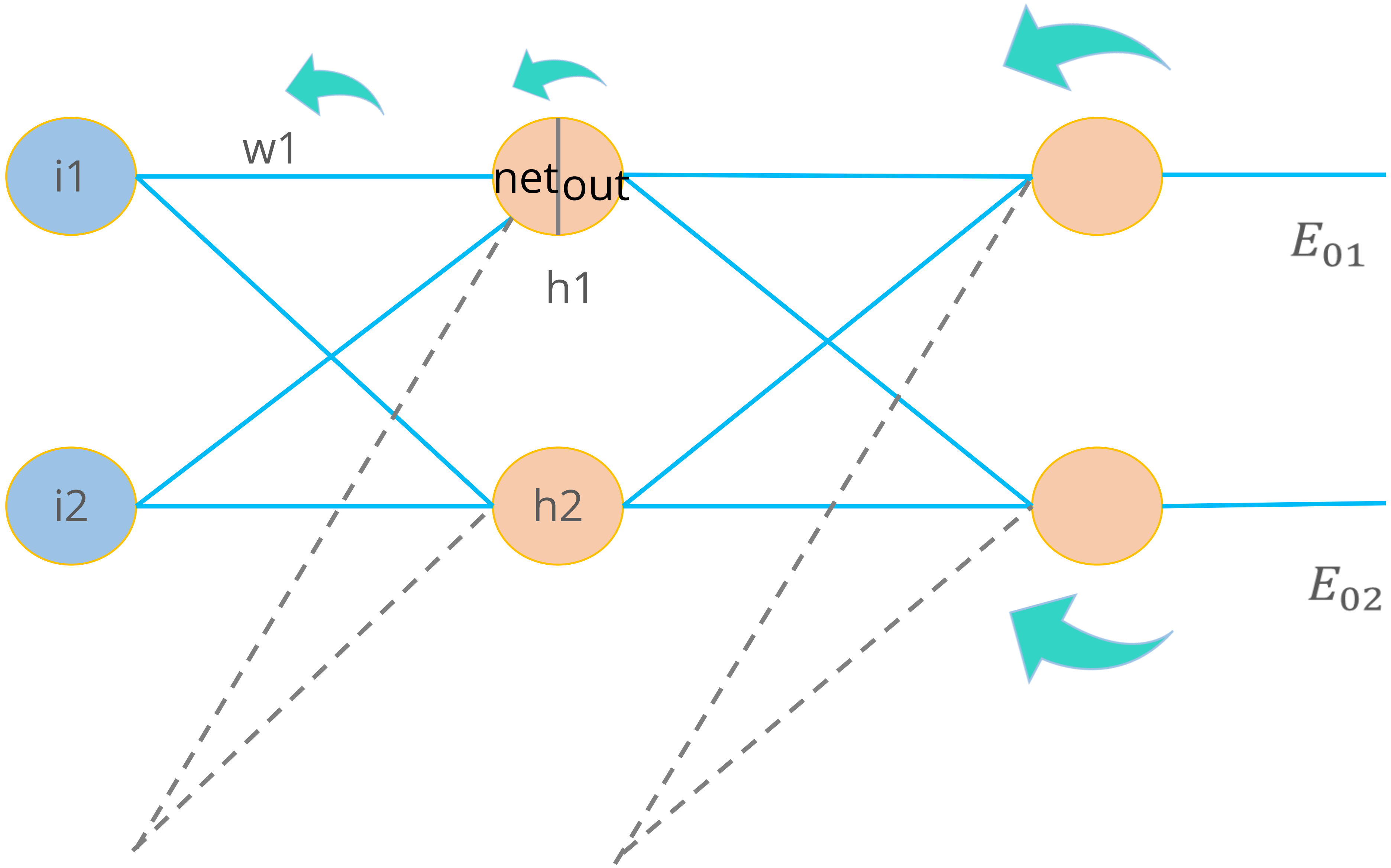
Updated Weight

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

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:

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

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

$$net_{01} = w_5 * out_{h1} + w_7 * out_{h2} + b_2 * 1$$

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 $\frac{\partial E_{02}}{\partial out_{h1}}$, you 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:

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

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

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

$=0.15-0.5*0.000438568=0.14978071$

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

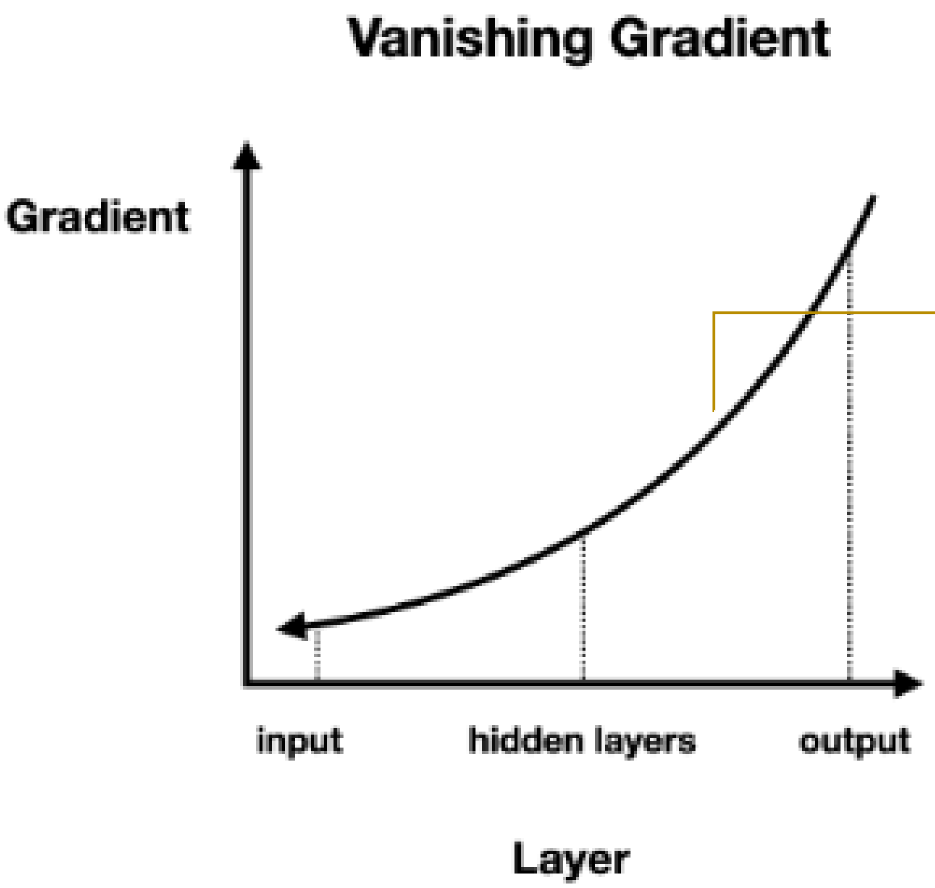
$w_2^{+} = 0.19956143$

$w_3^{+} = 0.24975114$

$w_4^{+} = 0.29950229$

Vanishing Gradient

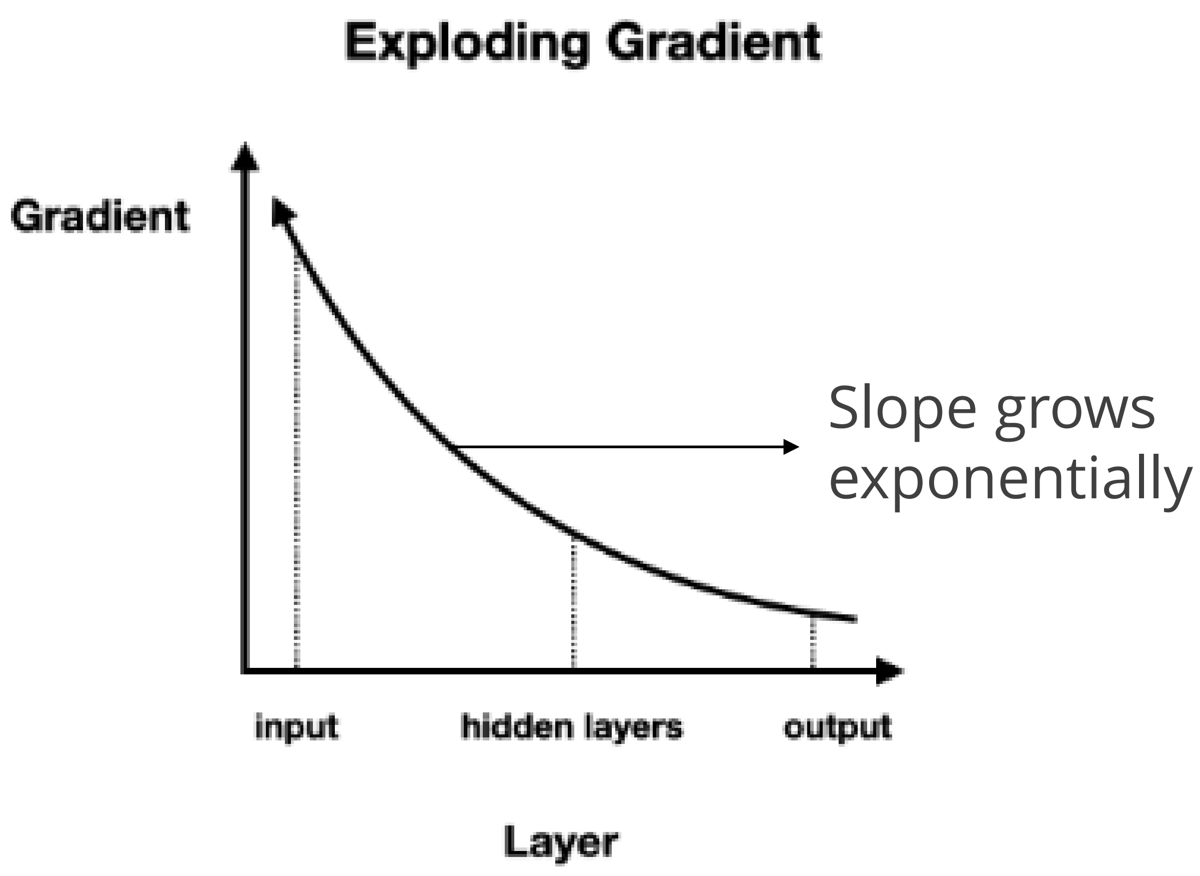
The vanishing gradient problem occurs during the training of deep neural networks, particularly those using gradient-based learning methods and backpropagation.



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

In such networks, gradients of the network's output with respect to the parameters in the earlier layers are calculated during training to update the parameters. However, if the gradients are very small (close to zero), they effectively prevent weights from changing their values, which means that the network stops learning or learns very slowly.

Exploding Gradient



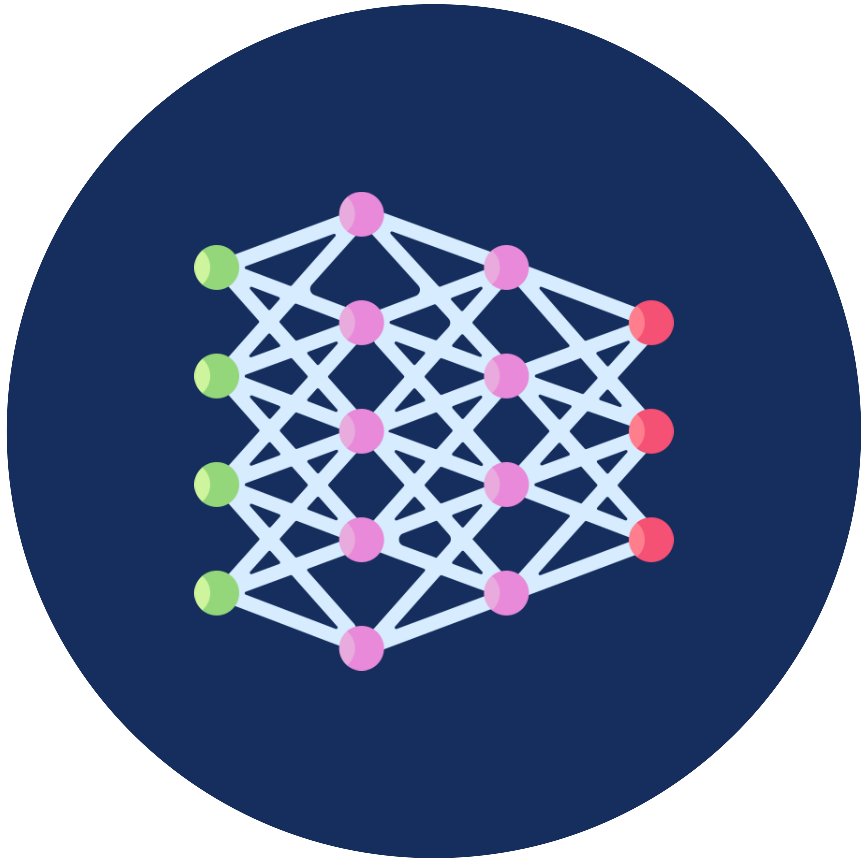
Conversely, the exploding gradient problem occurs when the gradients of the network's parameters become too large; this can lead to large changes in weights, resulting in an unstable network

During training, this often leads to a scenario where the model fails to converge, meaning the weights can diverge and the cost function (usually representing some form of error or loss) can become infinitely large.

Gradient Descent

Gradient Descent

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



It works by repetitively adjusting the input variables in the direction that reduces loss 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

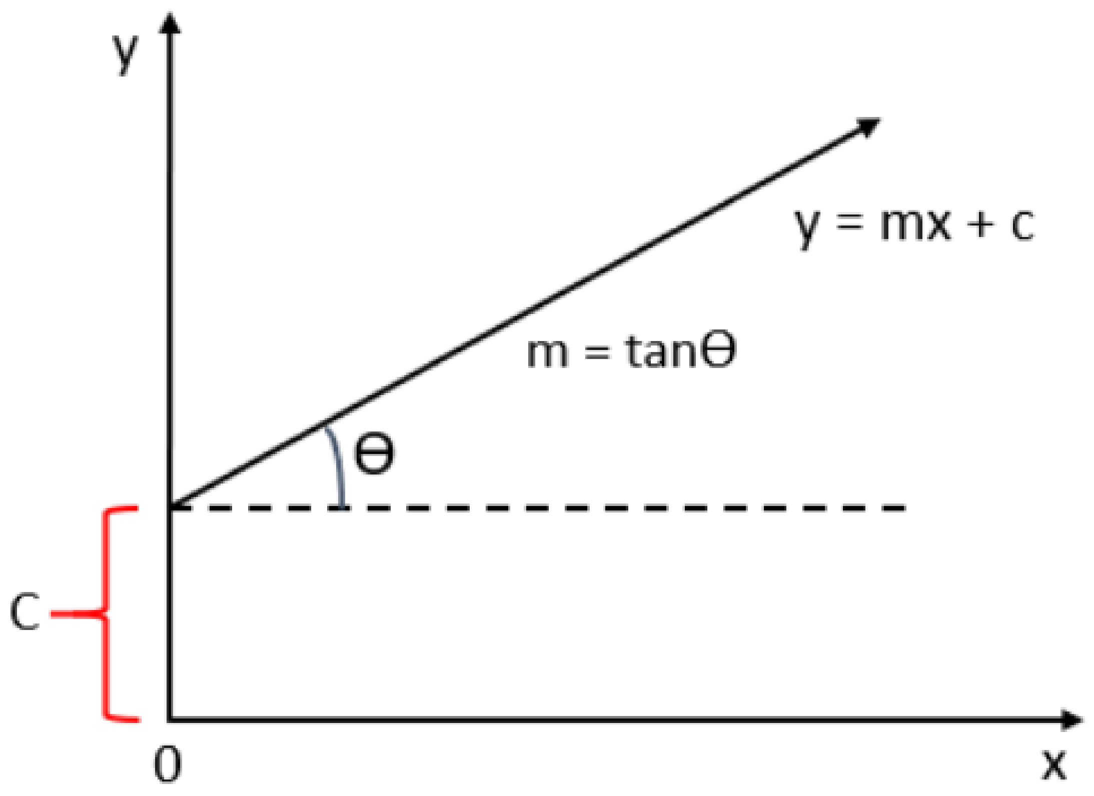
c = intercept

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.

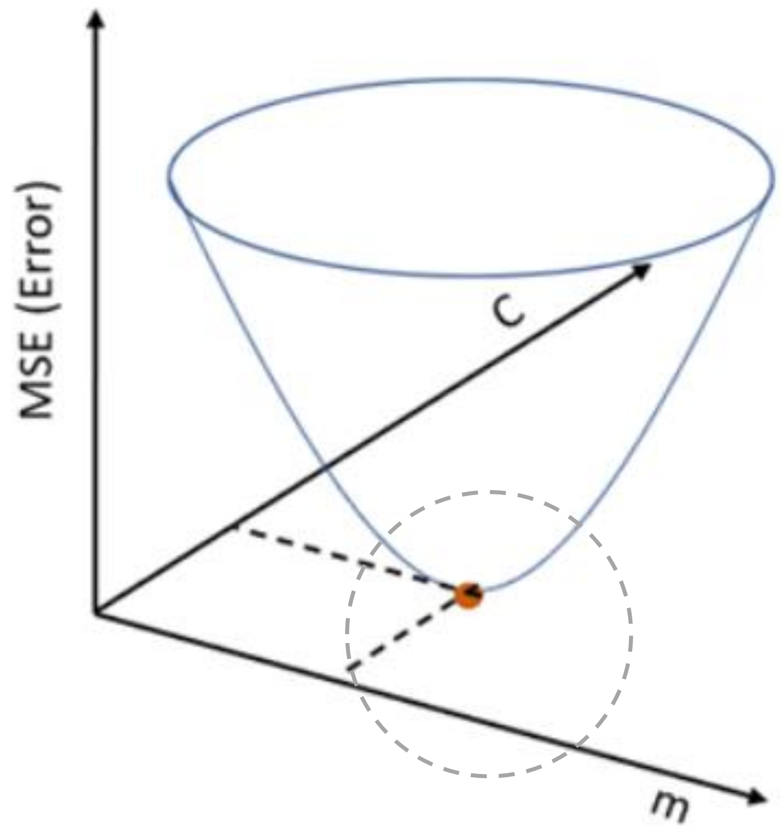
Working on Gradient Descent

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.

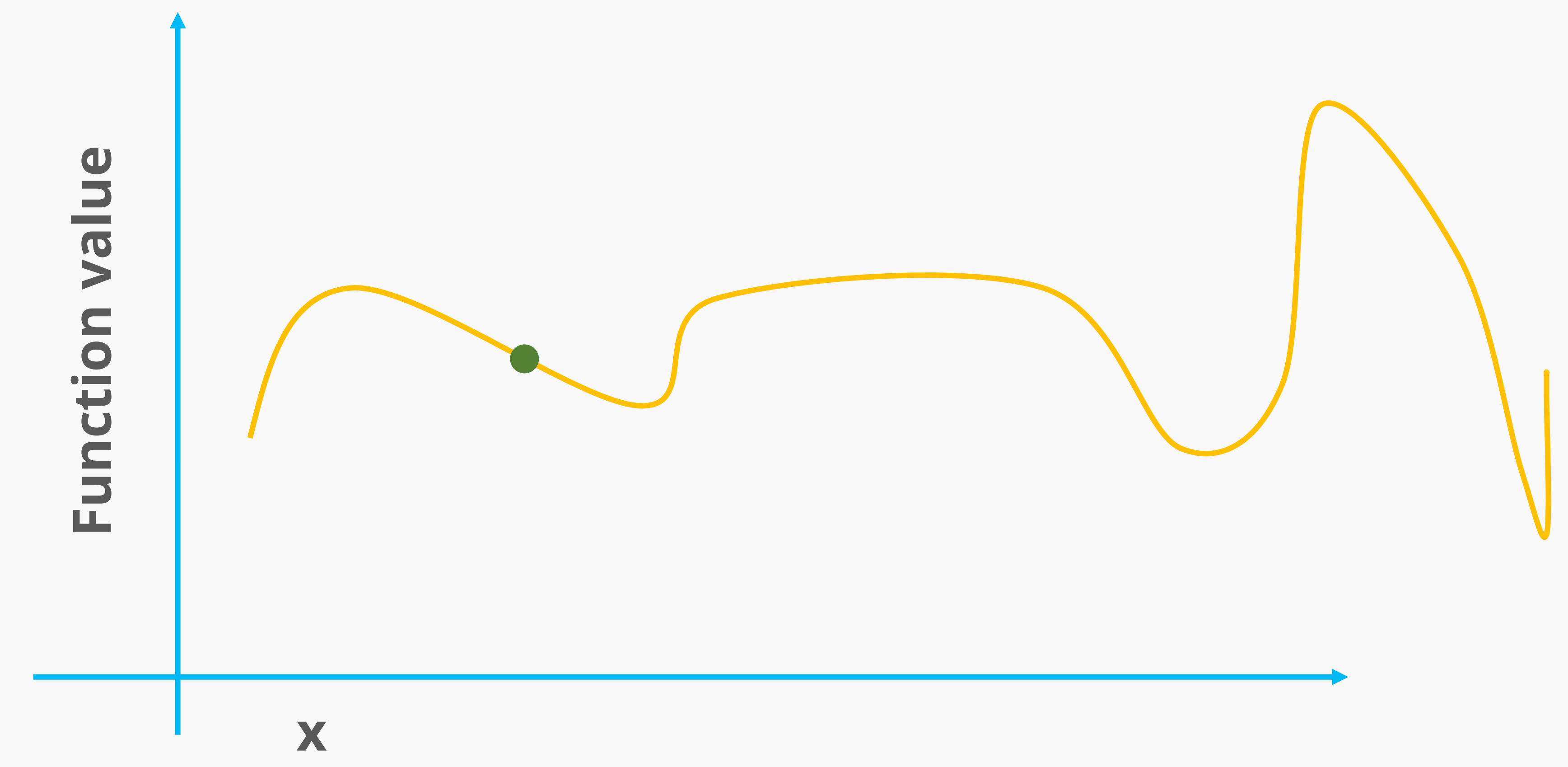
At this point, the values of c and m are their optimal values.

Deriving a Gradient Descent or Ascent Algorithm

- The algorithm iteratively updates weights to minimize the overall error or 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 toward the optimum.

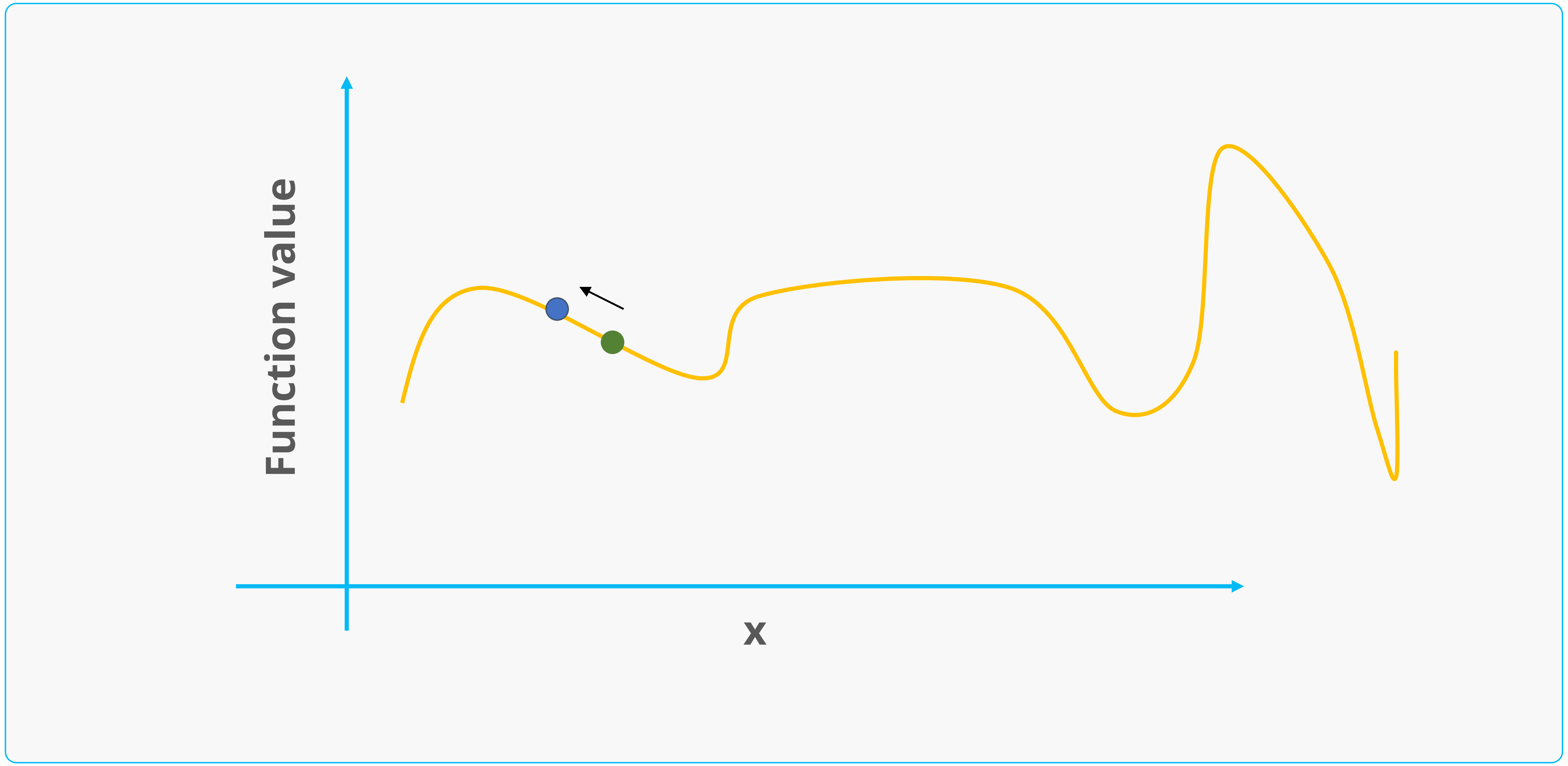
Gradient Ascent: Step 1

Initialize the model's parameters by selecting random values as a starting point for the optimization process



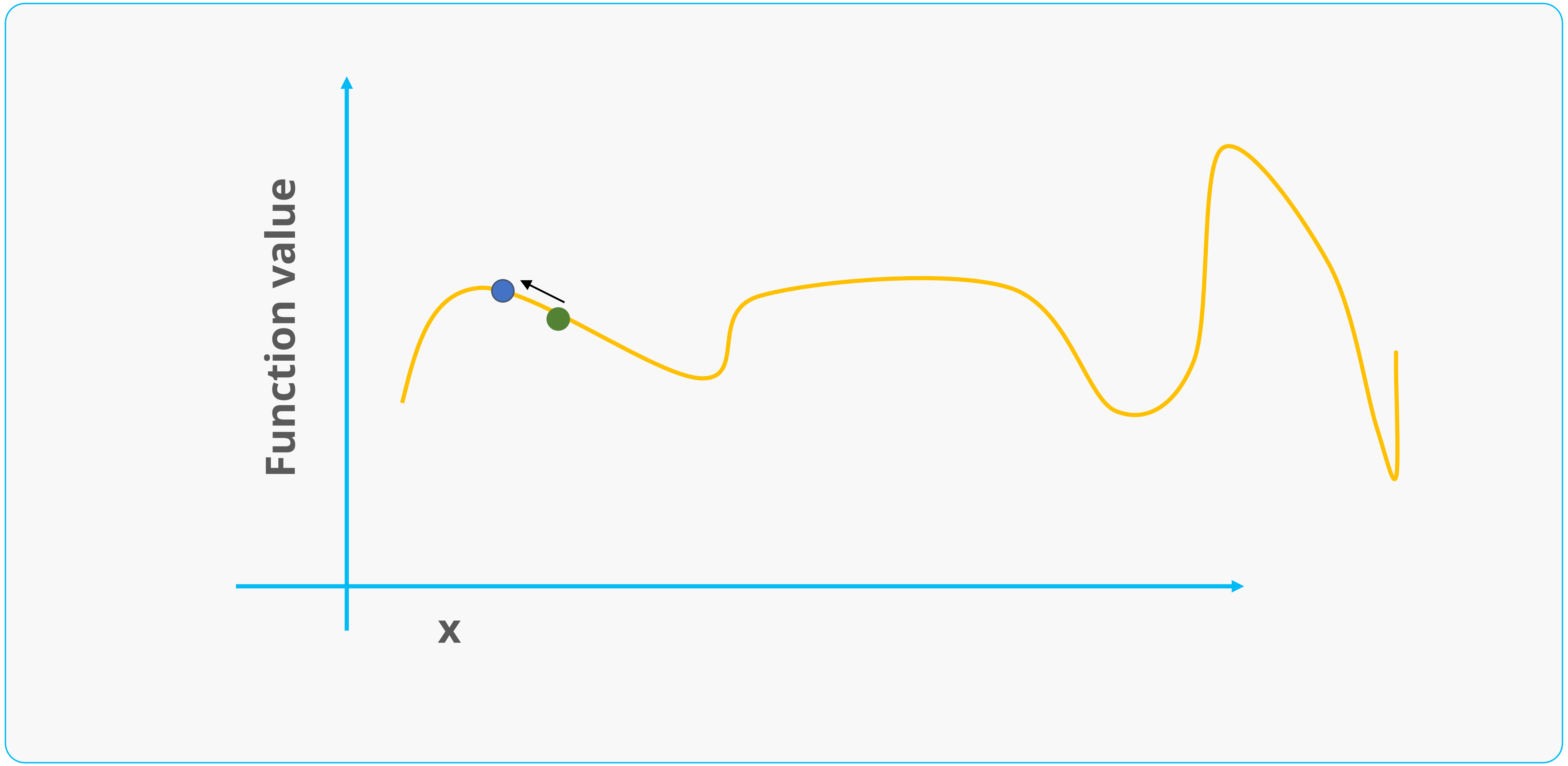
Gradient Ascent: Step 2

Take steps in the direction of the steepest ascent, maximizing the objective function, to iteratively approach the optimal solution



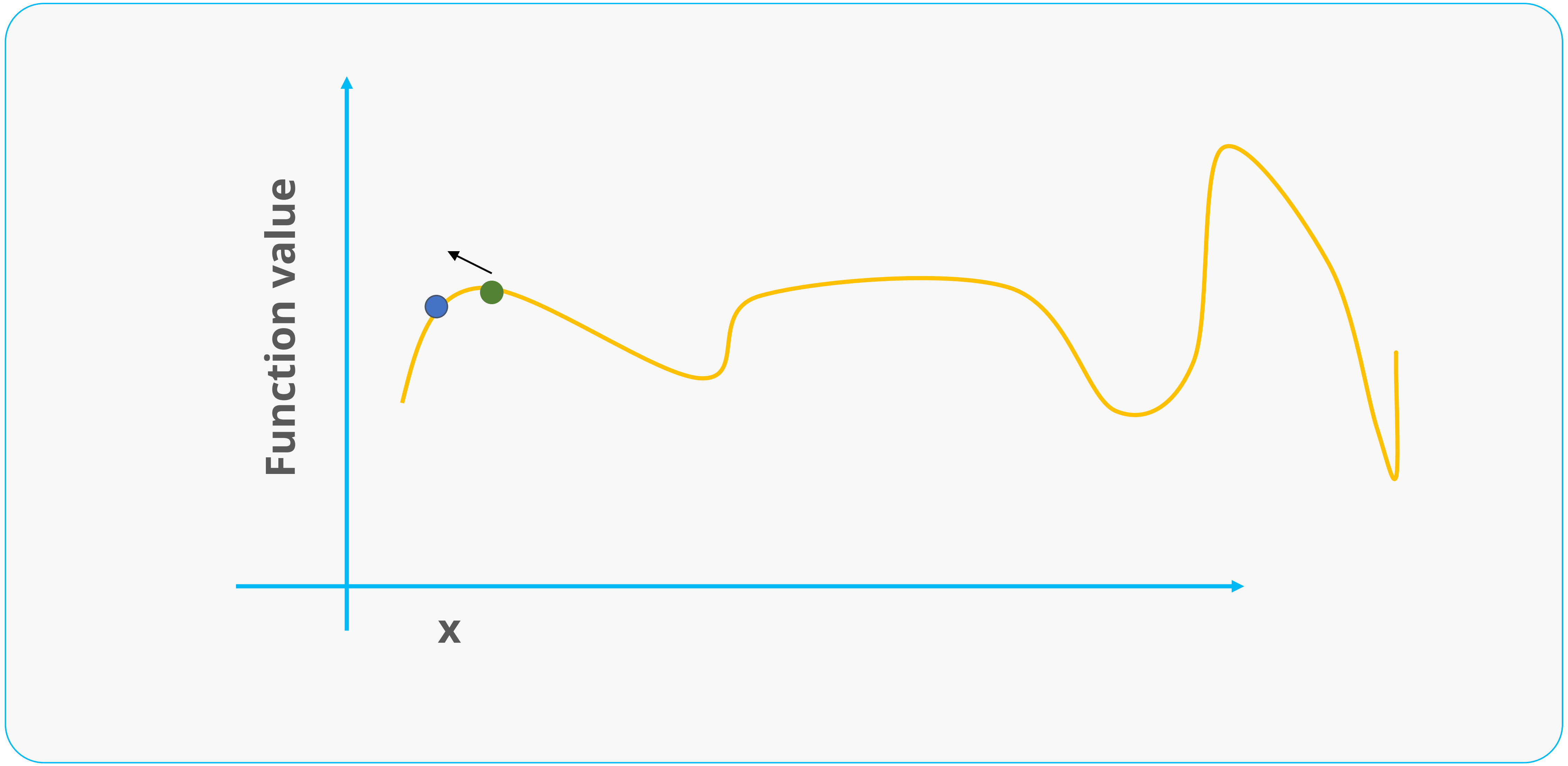
Gradient Ascent: Step 3

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



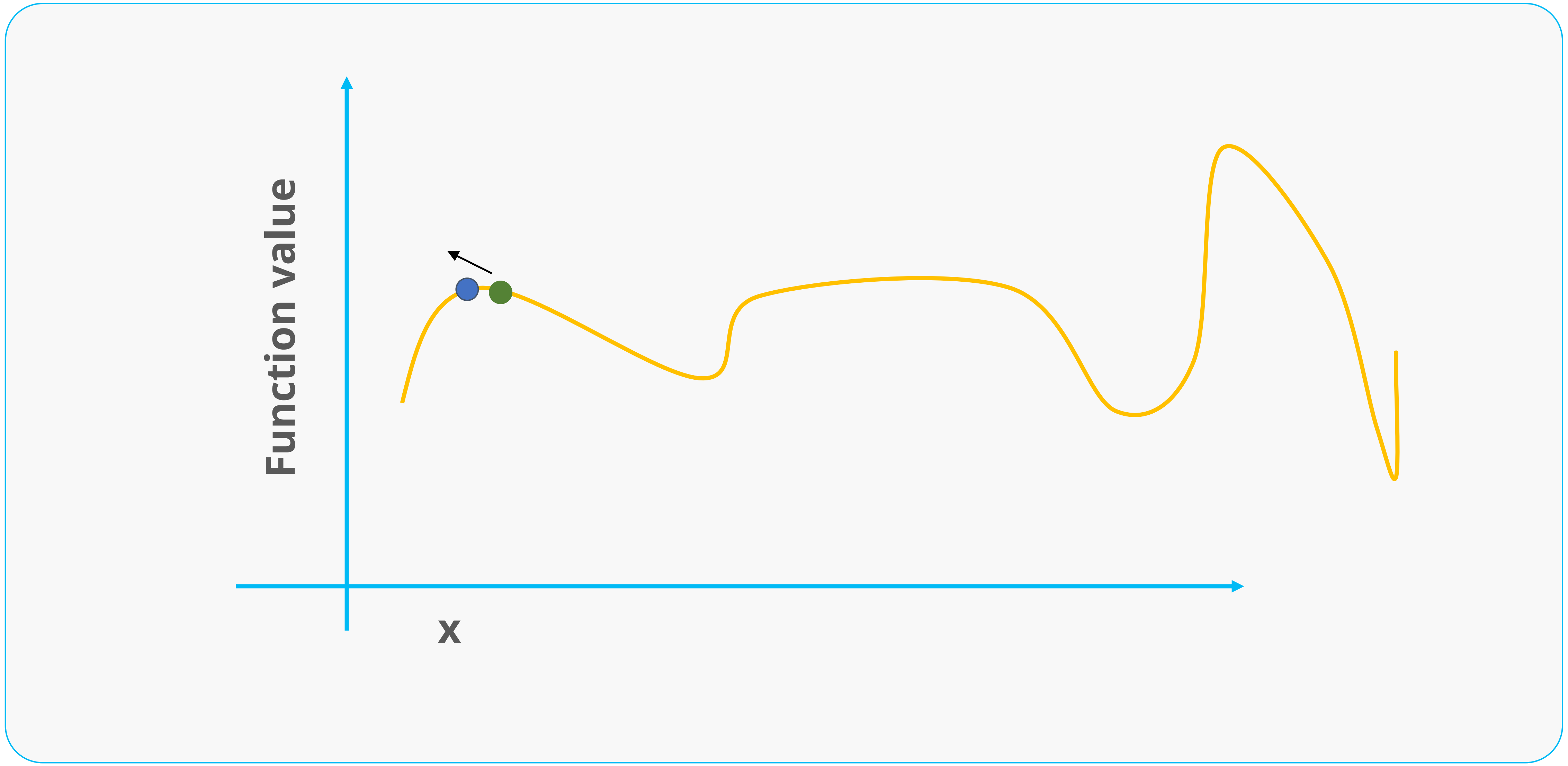
Gradient Ascent: Observations

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.



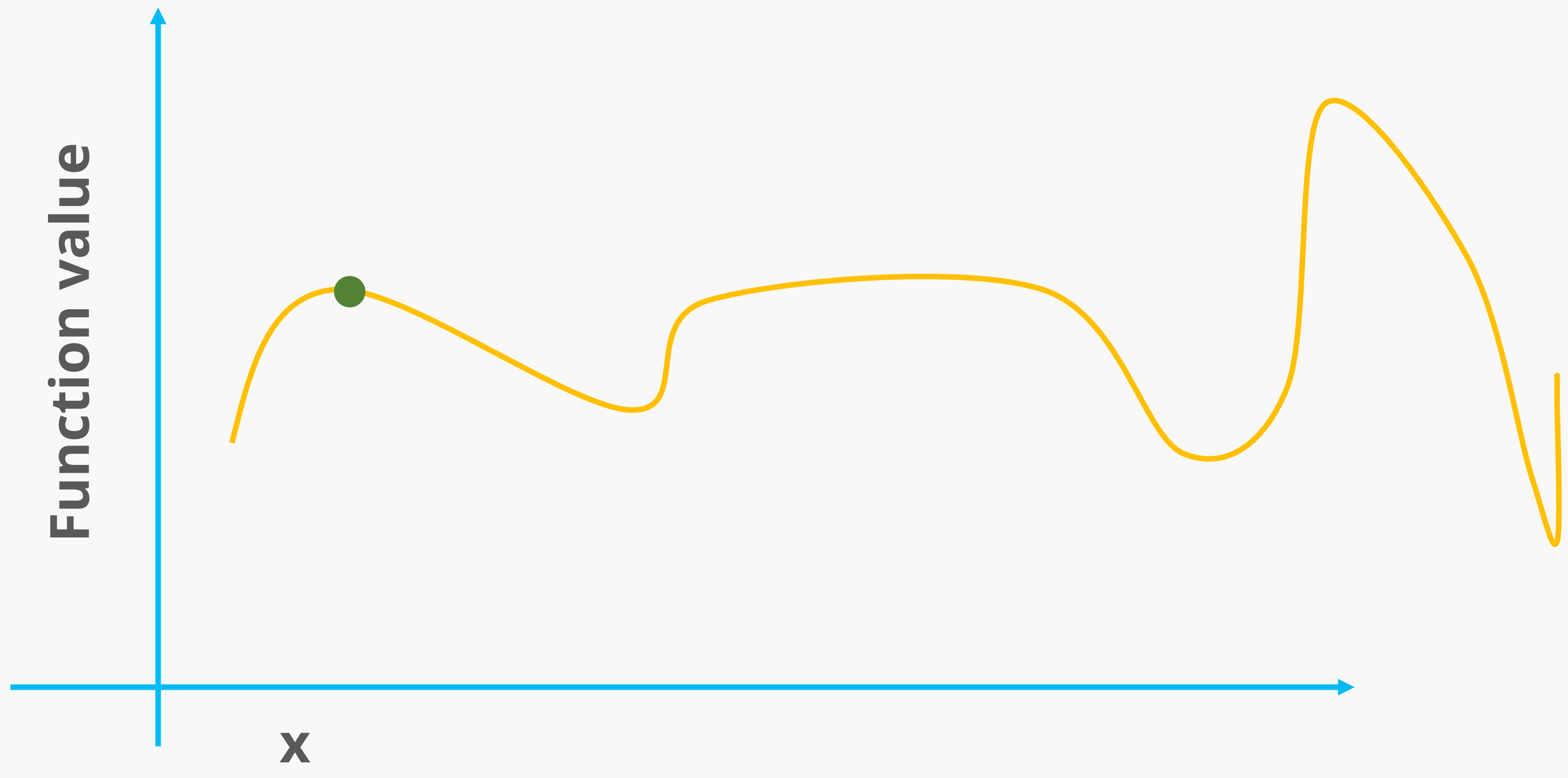
Gradient Ascent: Observations

The optimization process reduces the step size to take smaller, more precise steps toward 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.

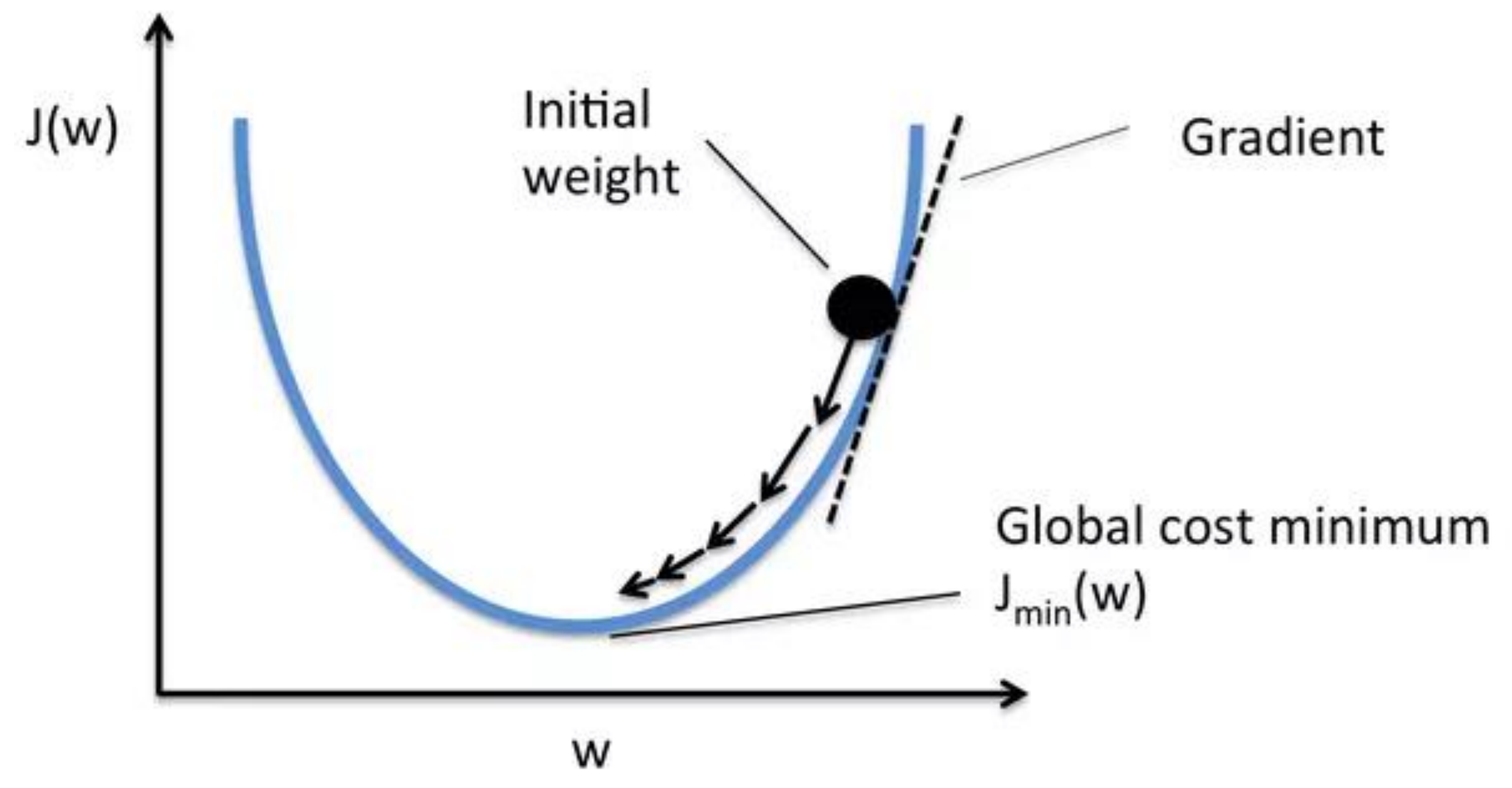


Gradient Ascent: Impacts on Neural Network

- 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 to minimize errors.
- It is used to analyze how an error will change when the values of weights and biases are changed by a unit.



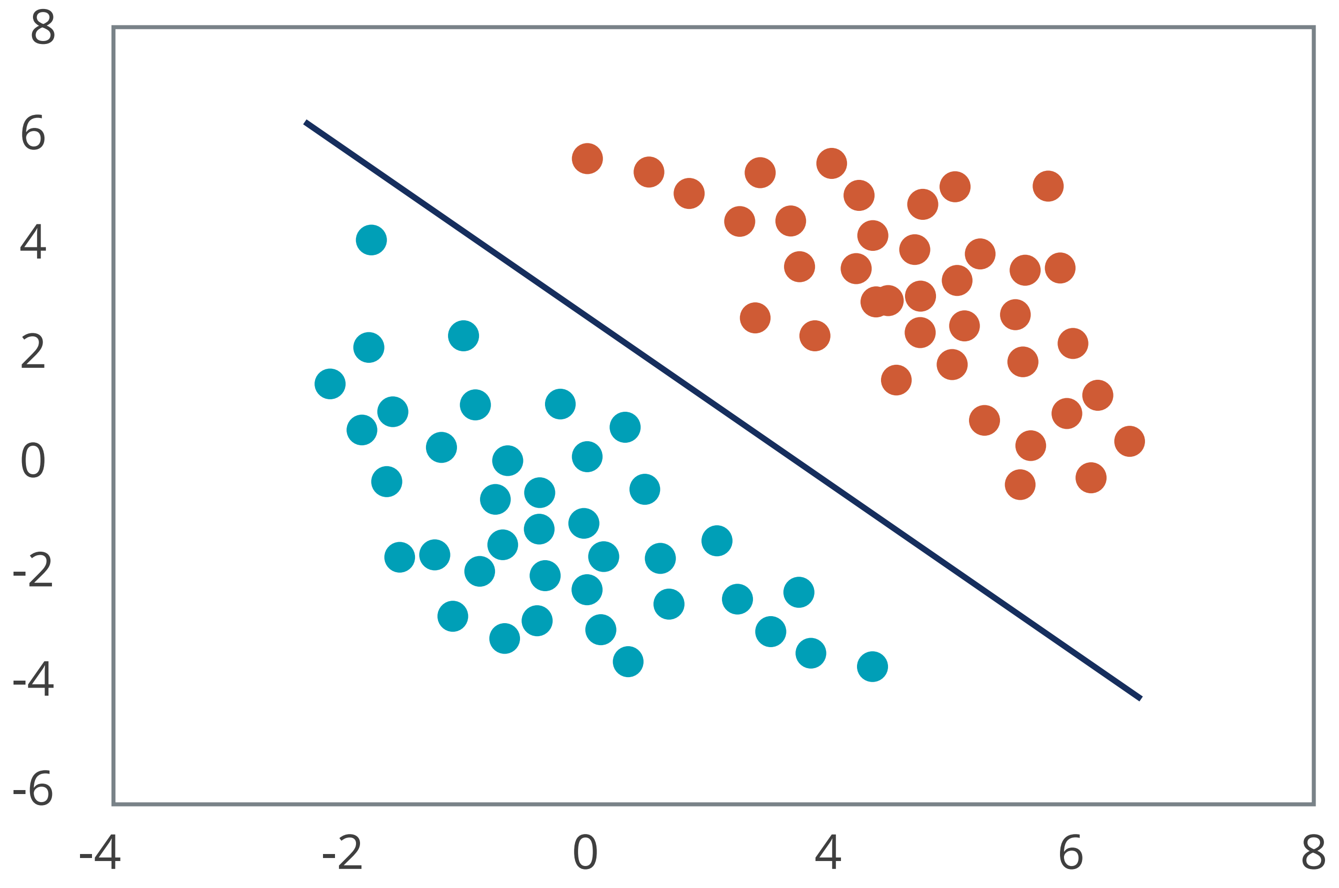
Note: Generally, a learning rate of 0.01 is a safe bet.

Limitations of a Perceptron

Limitations: Binary Number

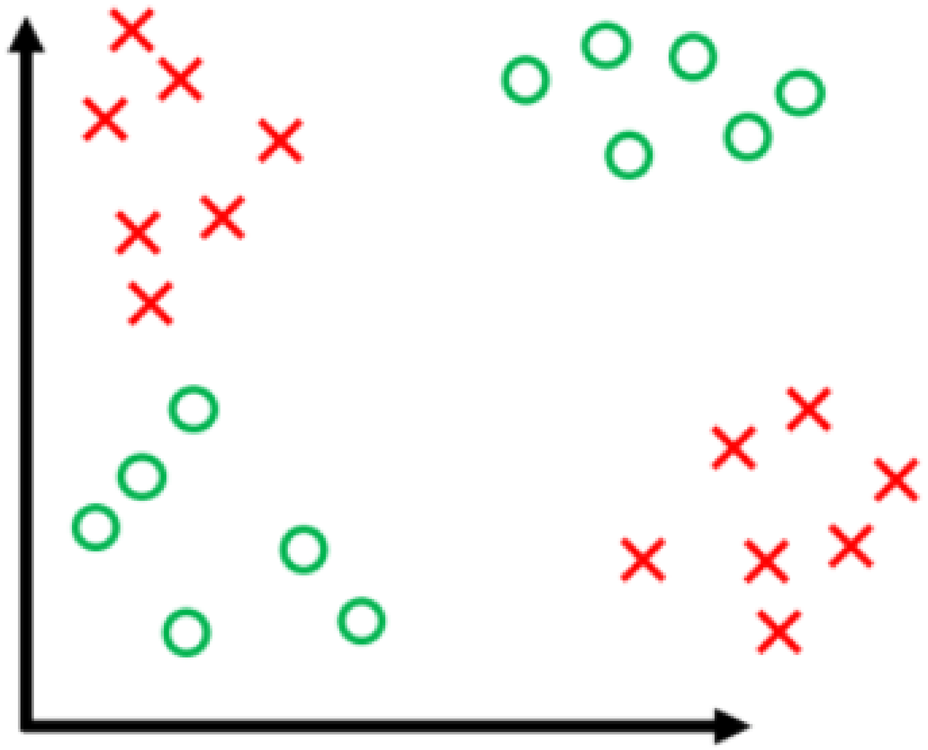
Though a perceptron is a simple and efficient supervised learning algorithm, it has certain limitations.

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.



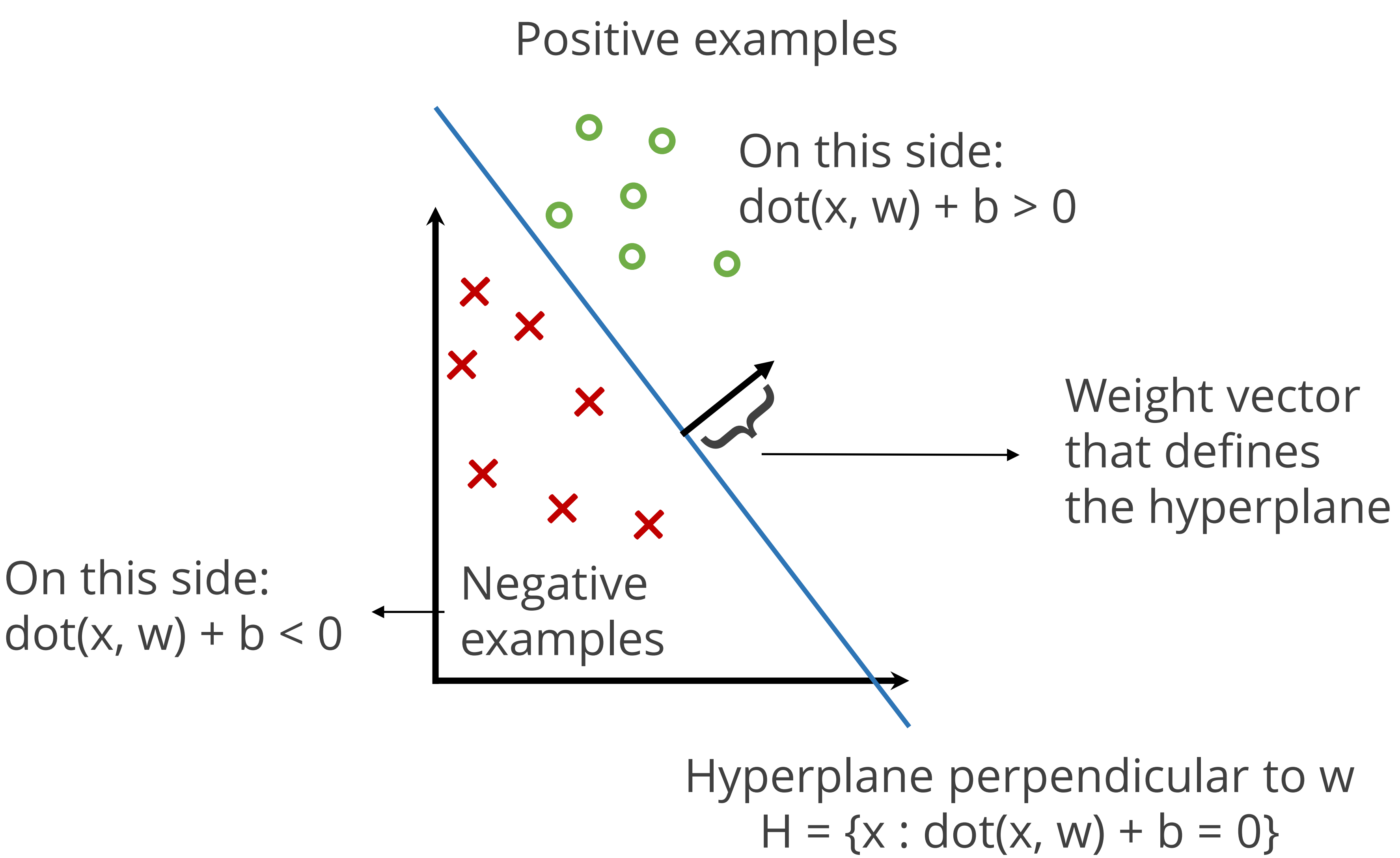
Example

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

No single straight line can separate the circles and crosses.

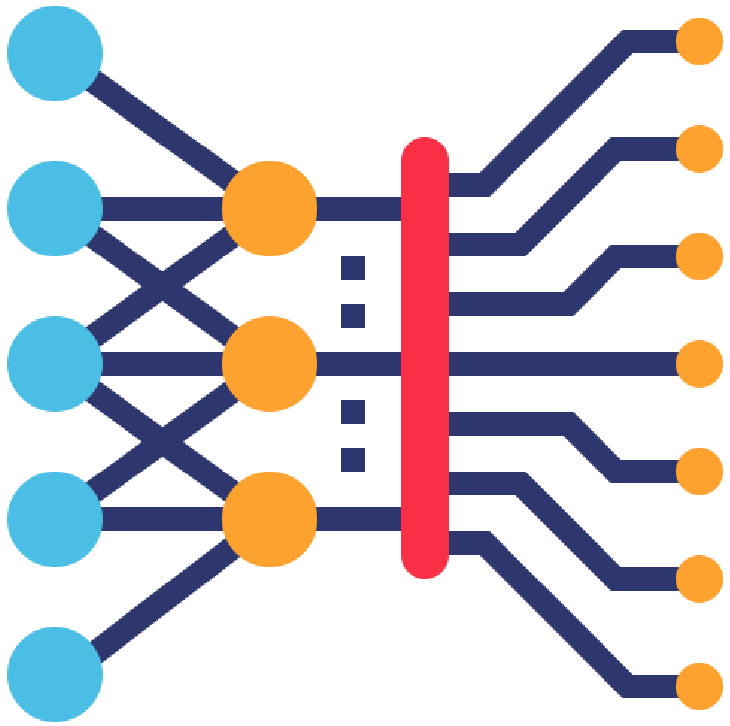
Limitations: Linearly Separable

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



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.

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 can only handle linearly separable data.

Knowledge Check

Knowledge Check 1

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

Knowledge Check

1

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.

Knowledge Check

2

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

Knowledge Check

2

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.

Knowledge Check 3

What is the use of 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.

Knowledge Check

3

What is the use of 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