NEURAL NETWORKS

Module 4

BLACK BOX METHODS

- Some ML algorithms belong to the category of black box methods due to the complex mathematics used in them to function.
- Examples
  - **Neural Networks** (NN)
  - **Support Vector Machines** (SVM)
- Neural networks mimic the structure of animal brains to model arbitrary functions.
- Support vector machines use multidimensional surfaces to define the relationship between features and outcomes.

UNDERSTANDING NEURAL NETWORKS

- An Artificial Neural Network (ANN) models the relationship between a set of input signals and an output signal.
- It is accomplished using a model derived from our understanding of how a biological brain responds to stimuli from sensory inputs.
- Just as a brain uses a network of interconnected cells called neurons, ANN uses a network of artificial neurons or nodes to solve learning problems.

UNDERSTANDING NEURAL NETWORKS

- The human brain is made up of about 85 billion neurons, forms a network capable of representing a tremendous amount of knowledge.
- ANNs have been used for over 50 years to simulate the brain's approach to problem-solving. At first, this involved learning simple functions like the logical AND function or the logical OR function.
- However, as computers have become increasingly powerful in the recent years, the complexity of ANNs has likewise increased so much.
- They are now frequently applied to more practical problems including:

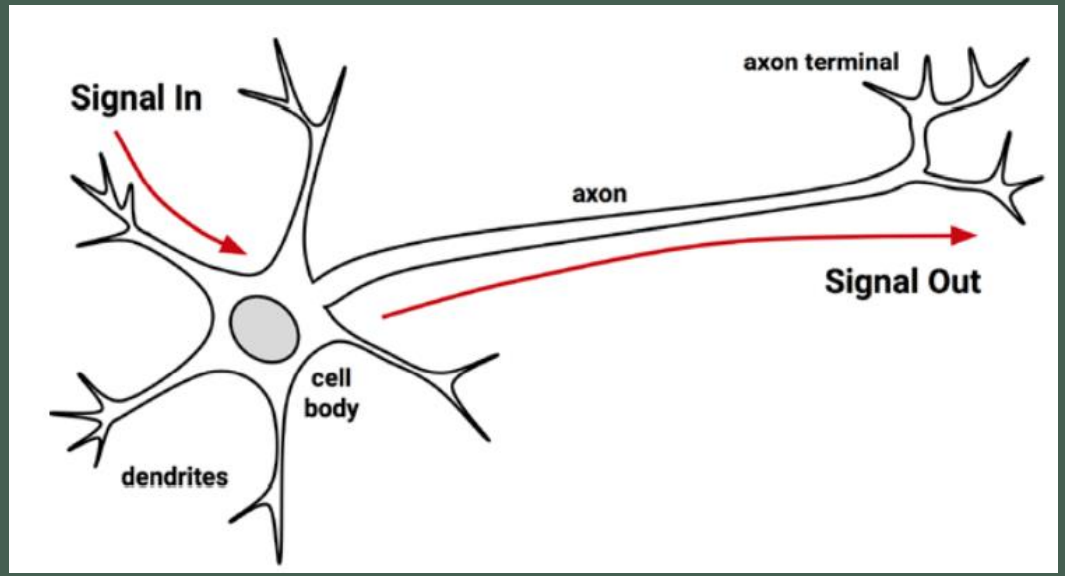
UNDERSTANDING NEURAL NETWORKS

- Speech and handwriting recognition programs.
- The automation of smart devices like an office building's environmental controls or self-driving cars and self-piloting drones.
- Sophisticated models of weather and climate patterns, tensile strength, fluid dynamics, and many other scientific, social, or economic phenomena.

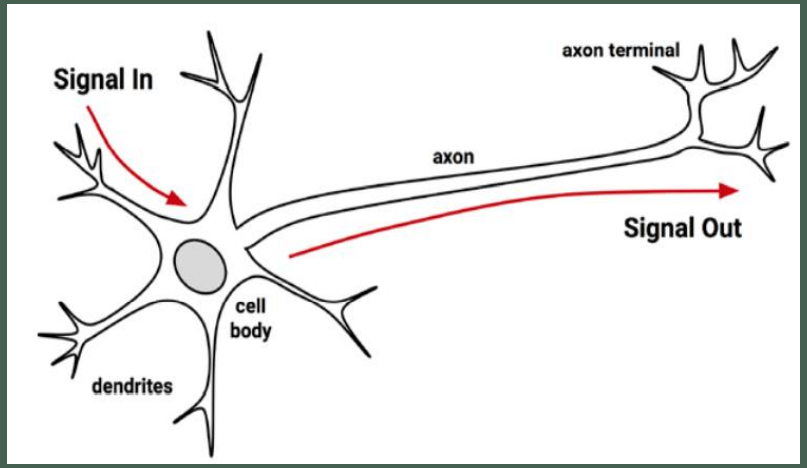
ANNs are versatile learners that can be applied to nearly any learning task: classification, numeric prediction, and for unsupervised pattern recognition.

FROM BIOLOGICAL TO ARTIFICIAL NEURONS

- ANNs were intentionally designed as conceptual models of human brain activity, it is helpful to first understand how biological neurons function.



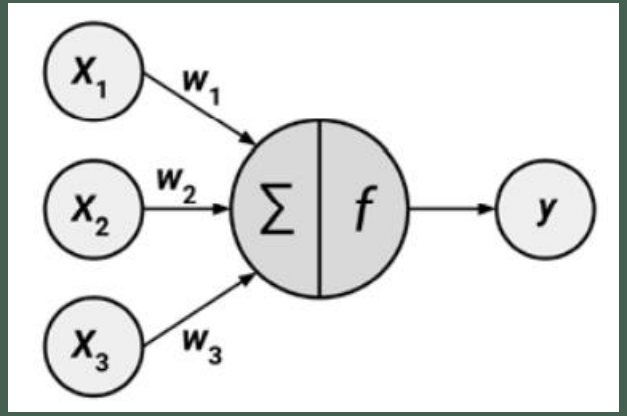
FROM BIOLOGICAL TO ARTIFICIAL NEURONS



- Incoming signals are received by the cell's dendrites through a biochemical process.
- The process allows the impulse to be weighted according to its relative importance or frequency.
- As the cell body begins accumulating the incoming signals, a threshold is reached at which the cell fires and the output signal is transmitted via an electrochemical process down the axon.
- At the axon's terminals, the electric signal is again processed as a chemical signal to be passed to the neighboring neurons across a tiny gap known as a synapse.

FROM BIOLOGICAL TO ARTIFICIAL NEURONS

- The model of a single artificial neuron:



- It is a directed network diagram defines a relationship between the input signals received by the dendrites (x variables), and the output signal (y variable).
- Just as with the biological neuron, each dendrite's signal is weighted (w values) according to its importance.
- The input signals are summed by the cell body and the signal is passed on according to an activation function denoted by f

FROM BIOLOGICAL TO ARTIFICIAL NEURONS

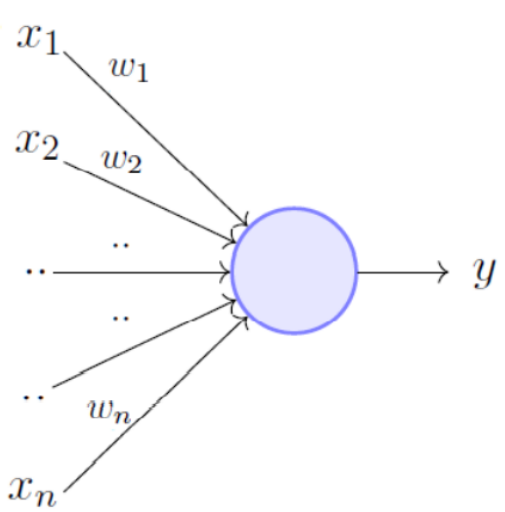
- A typical **artificial neuron with n input dendrites** can be represented by the formula that follows.

$$y(x) = f\left(\sum_{i=1}^{n} w_i x_i\right)$$

- The w weights allow each of the n inputs (x_i) to contribute a greater or lesser amount to the sum of input signals.
- The net total is used by the activation function f(x).
- The resulting signal, y(x), is the output axon.

PERCEPTRON

- Ex: Perceptron model



$$y = 1 \quad if \sum_{i=1}^{n} w_i * x_i \geq \theta$$

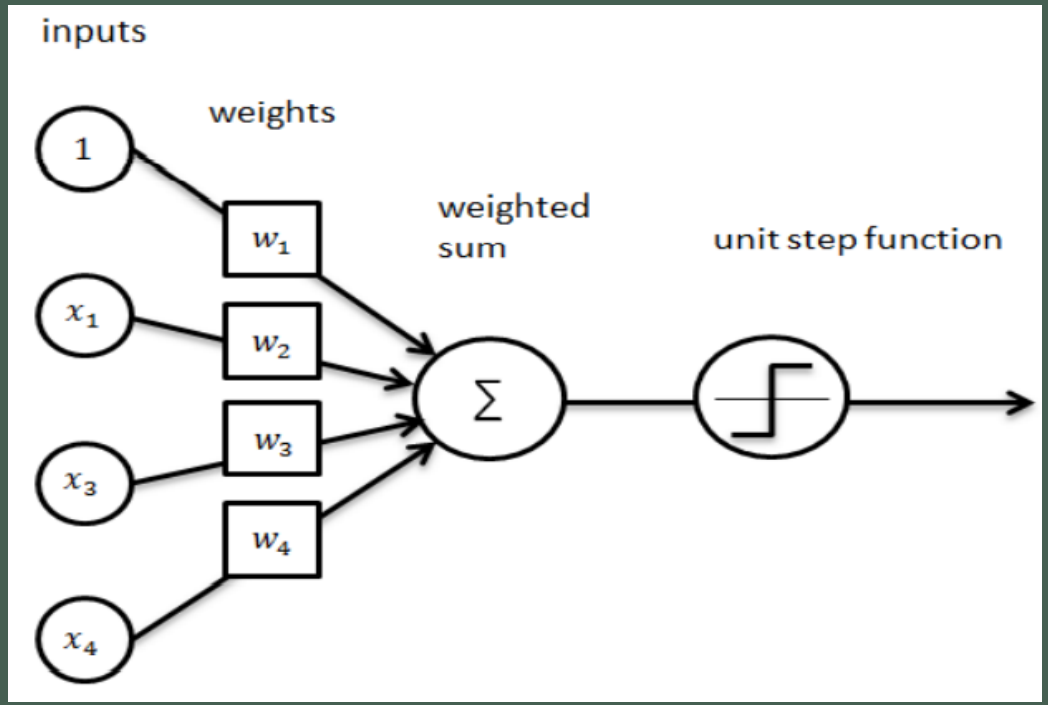
$$= 0 \quad if \sum_{i=1}^{n} w_i * x_i < \theta$$

Rewriting the above,

$$y = 1 \quad if \sum_{i=1}^{n} w_i * x_i - \theta \geq 0$$

$$= 0 \quad if \sum_{i=1}^{n} w_i * x_i - \theta < 0$$

PERCEPTRON MODEL

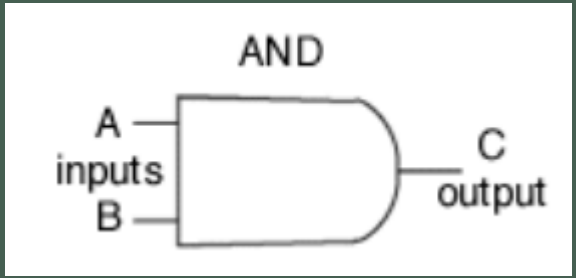


PERCEPTRON ALGORITHM

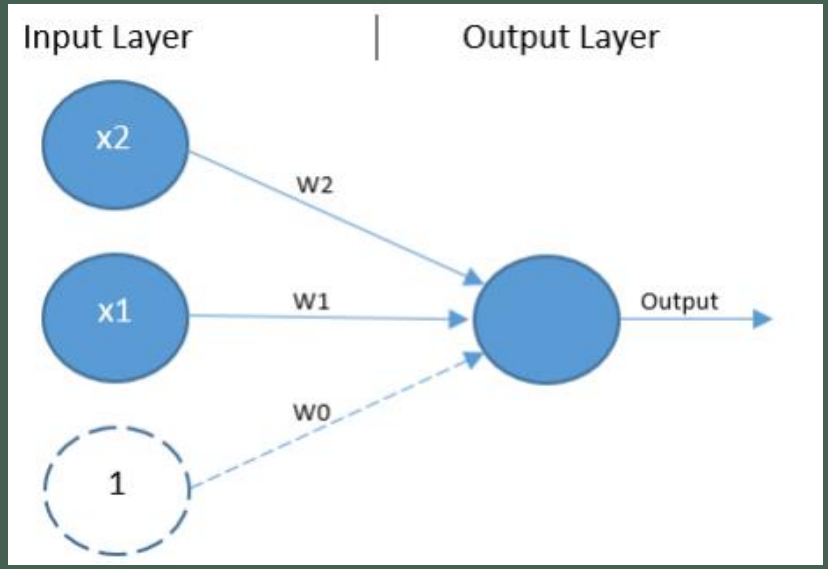
Algorithm: Perceptron Learning Algorithm

$P \leftarrow$ *inputs with label* 1;
$N \leftarrow$ *inputs with label* 0;
Initialize **w** randomly;
while !*convergence* **do**
  Pick random $\mathbf{x} \in P \cup N$;
  if $\mathbf{x} \in P$ *and* $\mathbf{w}.\mathbf{x} < 0$ **then**
    $\mathbf{w} = \mathbf{w} + \mathbf{x}$;
  end
  if $\mathbf{x} \in N$ *and* $\mathbf{w}.\mathbf{x} \geq 0$ **then**
    $\mathbf{w} = \mathbf{w} - \mathbf{x}$;
  end
end
//the algorithm converges when all the inputs are classified correctly

PERCEPTRON- LEARNING *AND* GATE



A	B	C
0	0	0
1	0	0
0	1	0
1	1	1

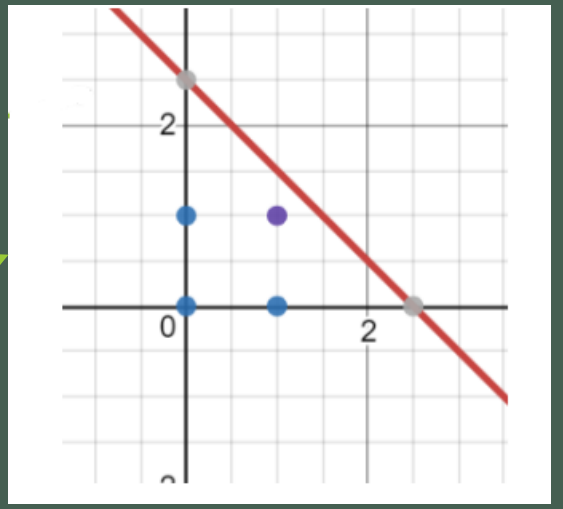


PERCEPTRON- LEARNING *AND* GATE

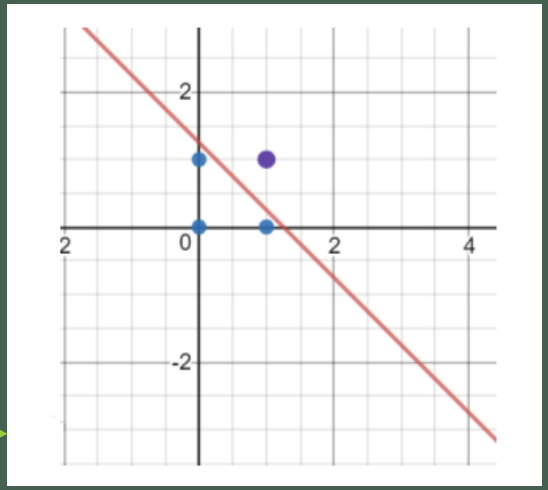
$F = w_1.x_1 + w_2.x_2 - \theta$

$1\ x_1 + 1\ x_2 - 2.5 = 0$

$w_1=1,\ w_2=1,\ \theta= 2.5$



w1	w2	(x1,x2)	F
1	1	(0,1)	-1.5
1	1	(1,1)	-0.5
2	2	(0,0)	-2.5
2	2	(1,0)	-0.5
2	2		

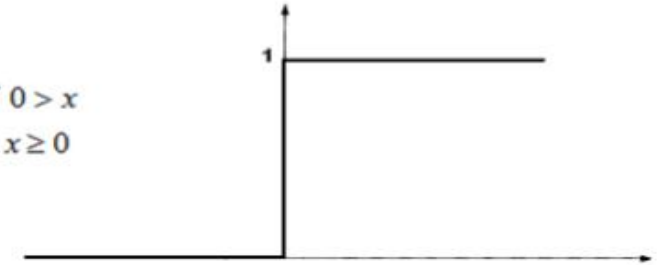


PERCEPTRON- LEARNING *AND* GATE

x1	x2	y = 2 x1+ 2 x2 - 2.5	f(y)
0	0	-2.5	0
0	1	-0.5	0
1	0	-0.5	0
1	1	1.5	1

Unit step (threshold)

$$f(x) = \begin{cases} 0 \text{ if } 0 > x \\ 1 \text{ if } x \geq 0 \end{cases}$$



PERCEPTRON- IMPLEMENTING *AND* GATE

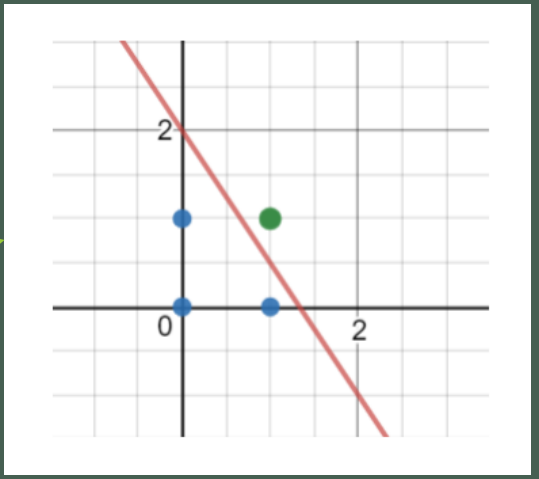
```python
#importing perceptron model from sklearn
from sklearn.linear_model import Perceptron

#training data for AND
X_train = [[0,0],[0,1],[1,0],[1,1]]
y_train= [0,0,0,1]

#model creation
clf = Perceptron(tol=1e-3, random_state=0)
clf.fit(X_train, y_train)

#prediction
y_pred=clf.predict(X_train)
print(y_pred)

print(clf.coef_,clf.intercept_)
```

Tools

- Python
- Sklearn

Output

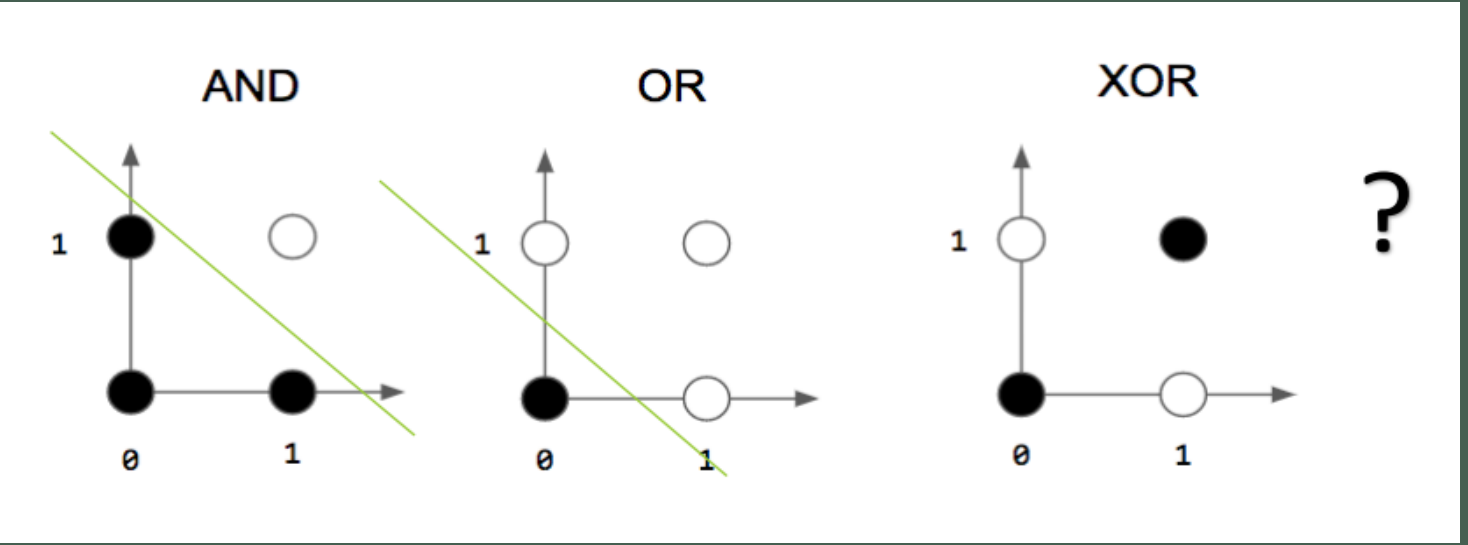
```
[0 0 0 1]
[[3. 2.]] [-4.]
```



Try it for

OR
NAND
NOR
XOR

IMPLEMENTING *OF MORE* GATES



FROM BIOLOGICAL TO ARTIFICIAL NEURONS

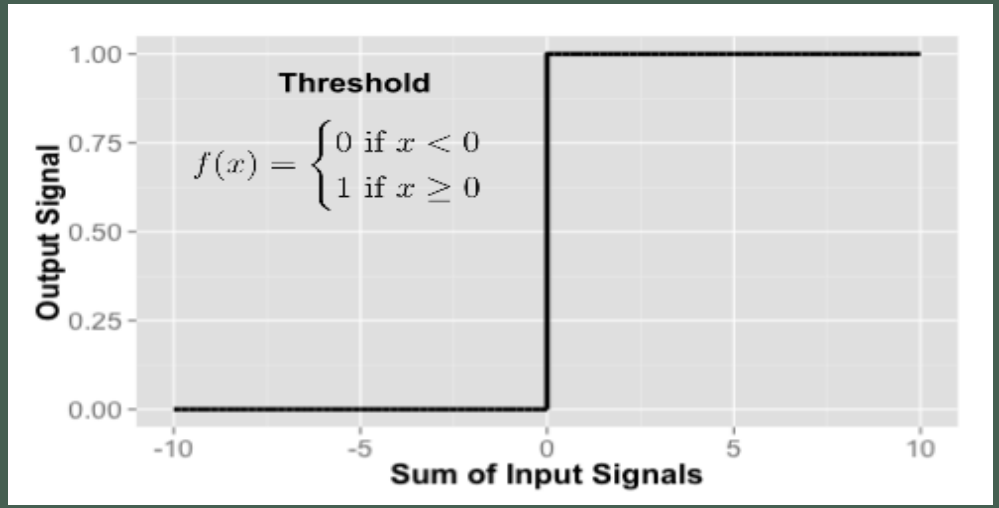
- Neural networks use neurons as building blocks to construct complex models of data.
- Variants of neural networks are there, each can be defined in terms of the following characteristics:
  - An ***activation function***, which transforms a neuron's combined input signals into a single output signal.
  - A ***network topology*** (or architecture), which describes the number of neurons in the model as well as the number of layers and manner in which they are connected.
  - The ***training algorithm*** that specifies how connection weights are set in order to inhibit or excite neurons in proportion to the input signal.

(I) ACTIVATION FUNCTION

- The activation function is the mechanism by which the artificial neuron processes incoming information and passes it throughout the network.
- In the biological case, the activation function could be imagined as a process that involves summing the total input signal and determining whether it meets the firing threshold. If so, the neuron passes on the signal; otherwise, it does nothing.
- In ANN terms, this is known as a threshold activation function, as it results in an output signal only once a specified input threshold has been attained.

(I) ACTIVATION FUNCTION

- The following figure depicts a typical threshold function; in this case, the neuron fires **when the sum of input signals is at least zero.** Because its shape resembles a stair, it is sometimes called a ***unit step activation function***.



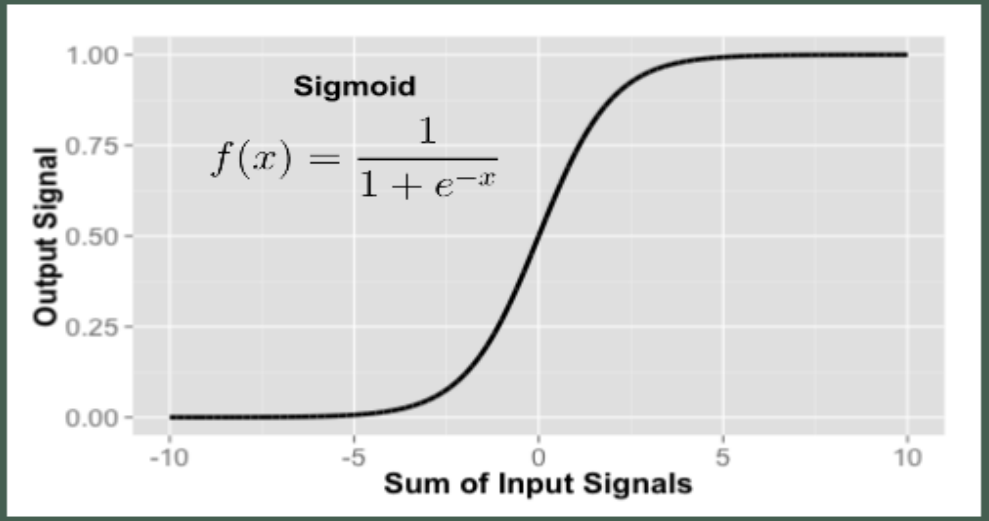
(I) ACTIVATION FUNCTION

- Sigmoid Activation Function

- Perhaps the most commonly used alternative is the **sigmoid activation function** (specifically the logistic sigmoid) shown in the following figure, where e is the base of natural logarithms.
- Although it shares a similar step or S shape with the threshold activation function, the output signal is no longer binary; output values can fall anywhere in the **range from 0 to 1.**
- Additionally, the sigmoid is differentiable, which means that it is possible to **calculate the derivative** across the entire range of inputs.

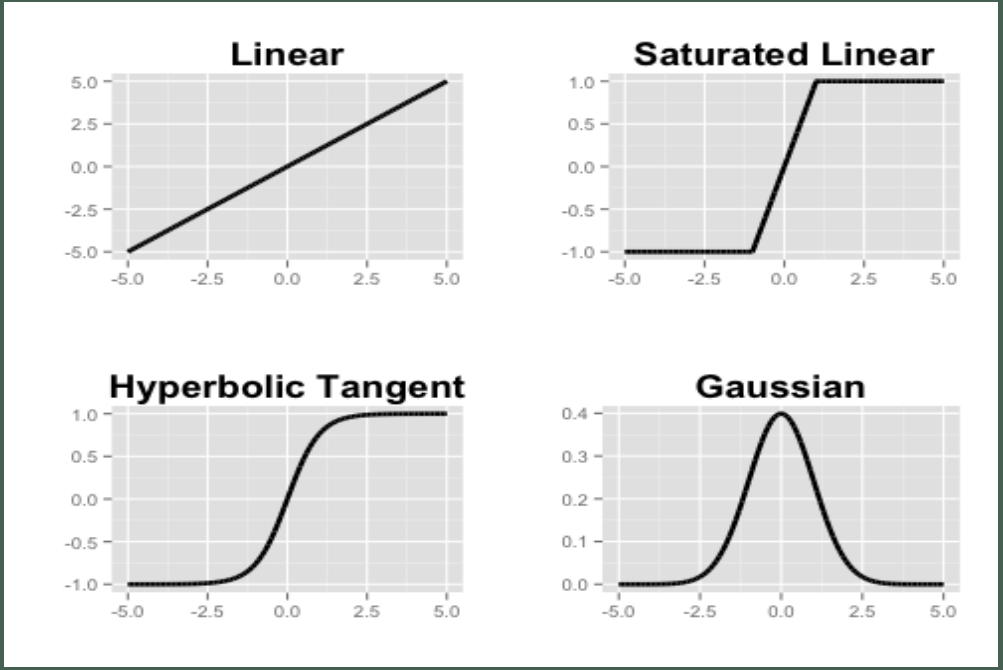
(I) ACTIVATION FUNCTION

- Sigmoid Activation Function



(I) ACTIVATION FUNCTION

- Types of Activation functions: Some neural network algorithms allow a choice of alternatives. A selection of such activation functions is shown in the following figure-



(I) ACTIVATION FUNCTION

- The primary detail that differentiates these activation functions is the output signal range.
- Typically, this is one of (0, 1), (-1, +1), or (-inf, +inf).
- The choice of activation function biases the neural network such that it may fit certain types of data more appropriately, allowing the construction of specialized neural networks.
- For instance, a linear activation function results in a neural network very similar to a **linear regression model**, while a Gaussian activation function results in a model called a Radial Basis Function (RBF) network.

ACTIVATION FUNCTION COMPARISON

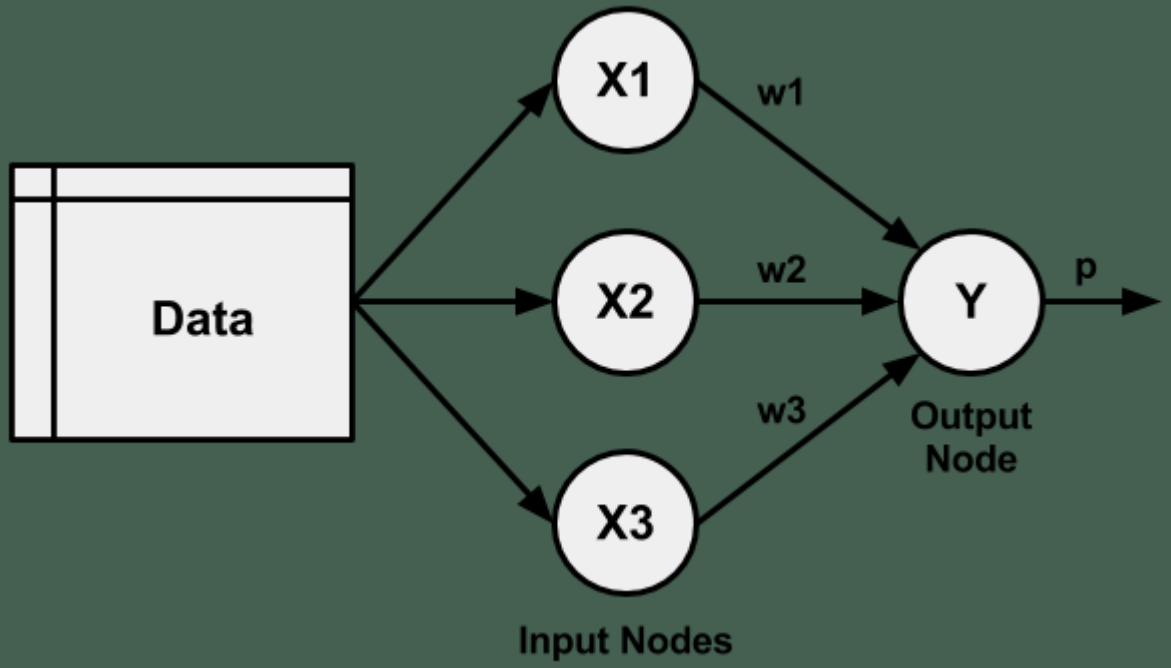
ACTIVATION FUNCTION	EQUATION	RANGE
Linear Function	$f(x) = x$	$(-\infty, \infty)$
Step Function	$f(x) = \begin{cases} 0 \; for \; x < 0 \\ 1 \; for \; x \geq 0 \end{cases}$	$\{0, 1\}$
Sigmoid Function	$f(x) = \sigma(x) = \frac{1}{1 + e^{-x}}$	$(0, 1)$
Hyperbolic Tanjant Function	$f(x) = \tanh(x) = \frac{(e^{x} - e^{-x})}{(e^{x} + e^{-x})}$	$(-1, 1)$
ReLU	$f(x) = \begin{cases} 0 \; for \; x < 0 \\ x \; for \; x \geq 0 \end{cases}$	$[0, \infty)$
Leaky ReLU	$f(x) = \begin{cases} 0.01 \; for \; x < 0 \\ x \; for \; x \geq 0 \end{cases}$	$(-\infty, \infty)$
Swish Function	$f(x) = 2x\sigma(\beta x) = \begin{cases} \beta = 0 \; for \; f(x) = x \\ \beta \to \infty \; for \; f(x) = 2\max(0, x) \end{cases}$	$(-\infty, \infty)$

(II) NETWORK TOPOLOGY

- The capacity of a neural network to learn is rooted in its **topology,** or the patterns and structures of interconnected neurons. Although there are countless forms of network architecture, they can be differentiated by three key characteristics:
  - The number of layers
  - Whether information in the network is allowed to travel backward
  - The number of nodes within each layer of the network

The number of layers

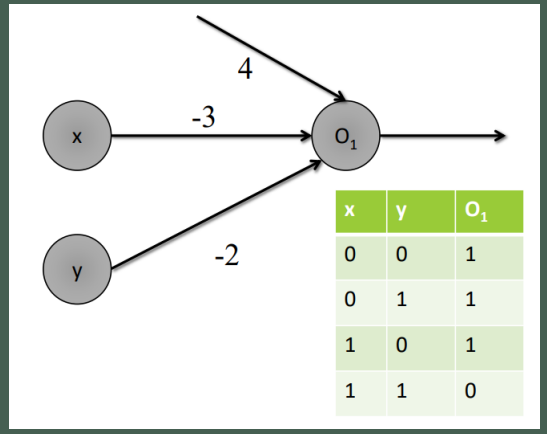
- To define topology, we need a terminology that distinguishes artificial neurons based on their position in the network. The figure that follows illustrates the topology of a very simple network.
- A set of neurons called Input Nodes receive unprocessed signals directly from the input data. Each input node is responsible for processing a single feature in the dataset; the feature's value will be transformed by the node's activation function.
- The signals resulting from the input nodes are received by the Output Node, which uses its own activation function to generate a final prediction


Data
X1
X2
X3
w1
w2
w3
Y
p
Output Node
Input Nodes

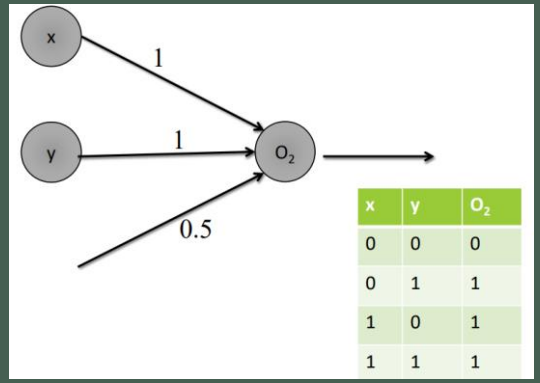
Single Layer Networks

- **The input and output nodes are arranged in groups known as layers.** Because the input nodes process the incoming data exactly as received, the network has only one set of connection weights (labeled here as w1, w2, and w3).
- It is therefore termed a single-layer network. Single-layer networks can be used for **basic pattern classification, particularly for patterns that are linearly separable**

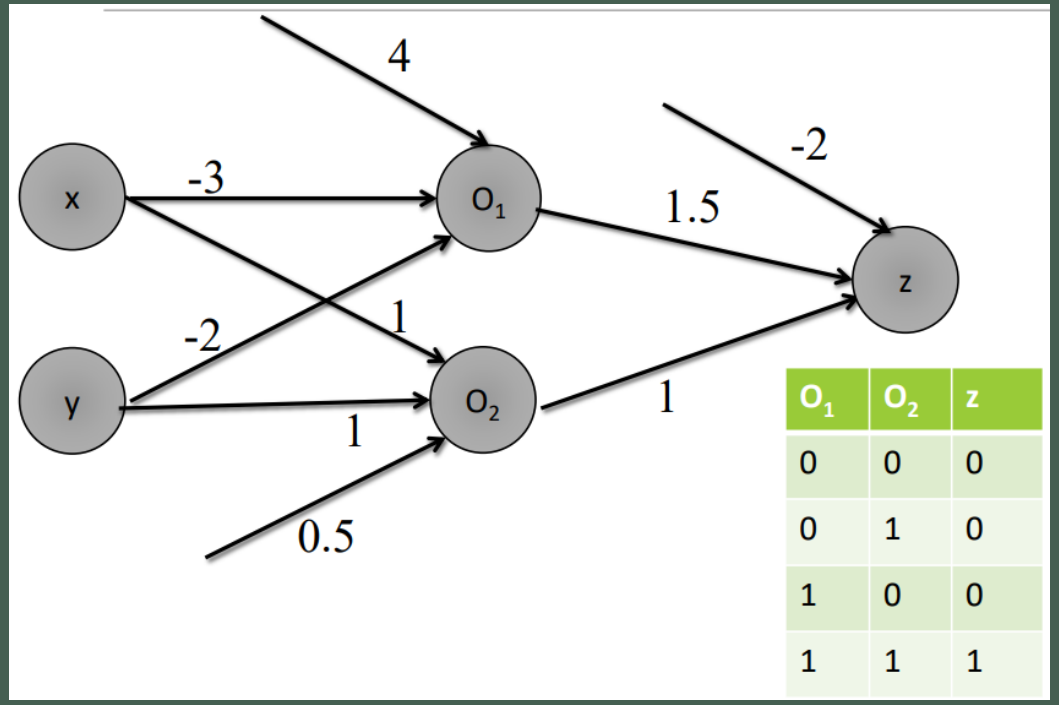
Solution of XOR Problem- needs MLP (MultiLayer Perceptron)



x	y	O_1
0	0	1
0	1	1
1	0	1
1	1	0



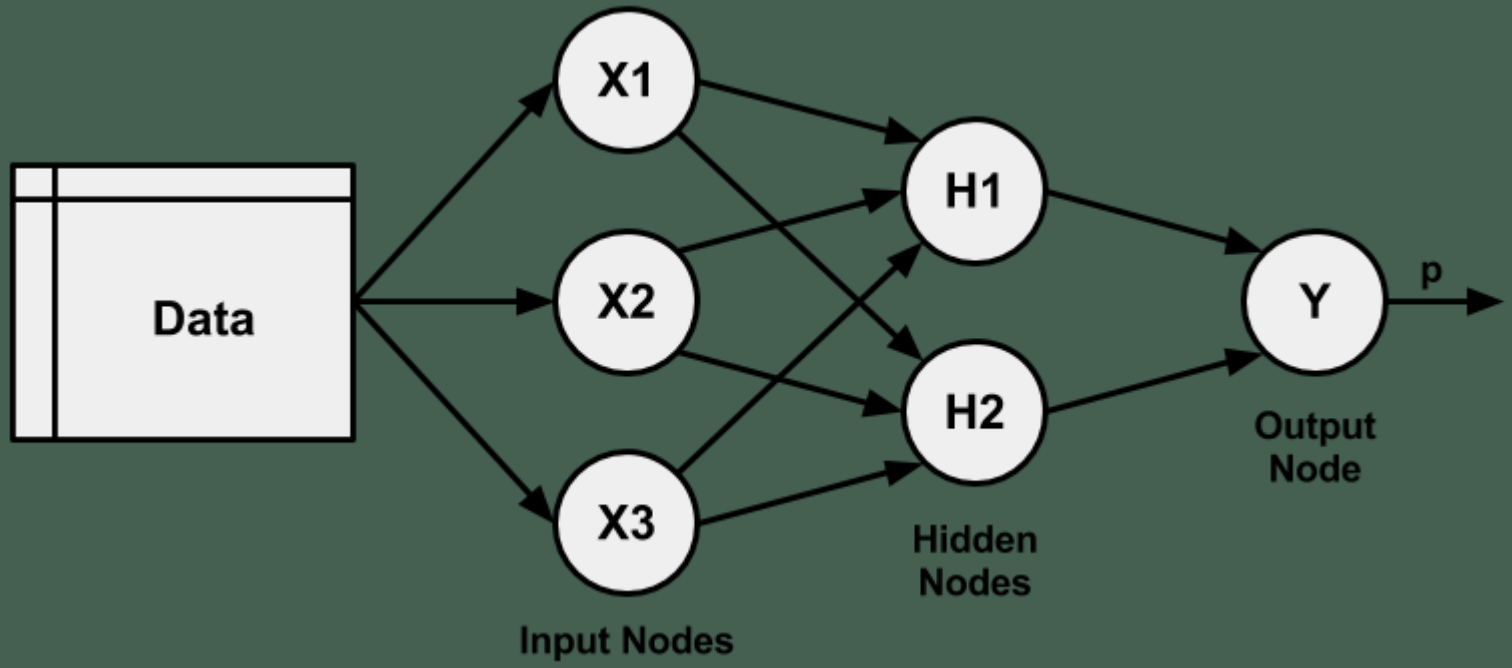
x	y	O_2
0	0	0
0	1	1
1	0	1
1	1	1



O_1	O_2	z
0	0	0
0	1	0
1	0	0
1	1	1

multilayer network

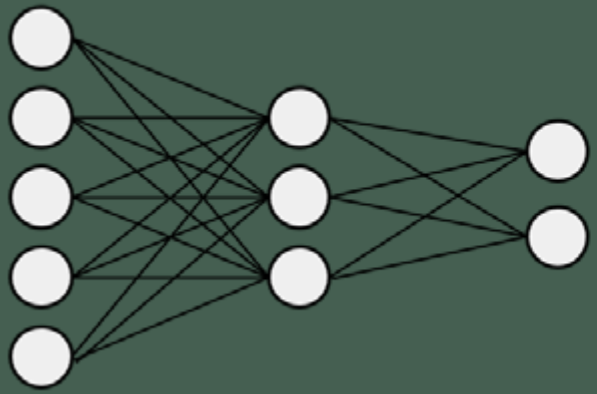
- A multilayer network adds one or more hidden layers that process the signals from the input nodes prior to reaching the output node.
- Most multilayer networks are fully connected, which means that every node in one layer is connected to every node in the next layer


Data
X1
X2
X3
Input Nodes
H1
H2
Hidden Nodes
Y
Output Node
p

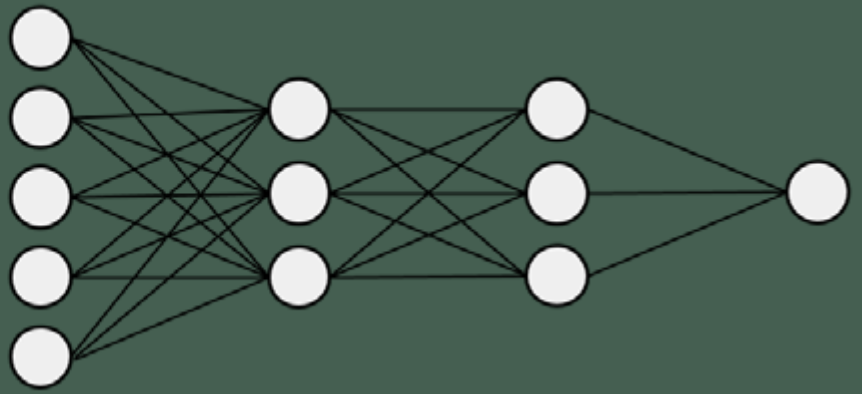
The direction of information travel

- Networks in which the input signal is fed continuously in one direction from connection-to-connection until reaching the output layer are called feedforward networks.
- The number of levels and nodes at each level can be varied, multiple outcomes can be modeled simultaneously, or multiple hidden layers can be applied (a practice that is sometimes referred to as deep learning).

Multiple Output Nodes

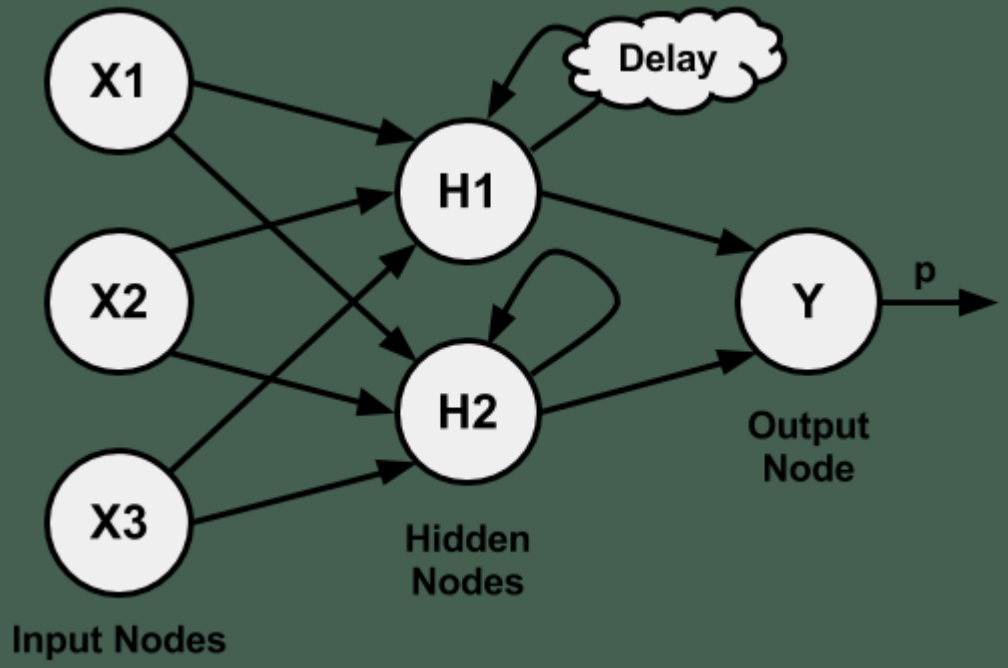


Multiple Hidden Layers



Feed back/Recurrent Network

- In contrast, a recurrent network (or feedback network) allows signals to travel in both directions using loops. This property, which more closely mirrors how a biological neural network works, allows extremely complex patterns to be learned.
- The addition of a short term memory (labeled Delay in the following figure) increases the power of recurrent networks immensely. Notably, this includes the capability to understand sequences of events over a period of time. This could be used for stock market prediction, speech comprehension, or weather forecasting.



- The multilayer feed forward network also called Multilayer Perceptron (MLP).

The number of nodes in each layer

- The **number of input nodes is predetermined by the number of features** in the input data.
- Similarly, the **number of output nodes is predetermined by the number of outcomes** to be modeled or the number of class levels in the outcome.
- However, the number of hidden nodes is left to the user to decide prior to training the model.

- There is no reliable rule to determine the number of neurons in the hidden layer.
- The appropriate number depends on the **number of input nodes,the amount of training data, the amount of noisy data, and the complexity of the learning task** among many other factors.

- A best practice is to use the fewest nodes that result in adequate performance on validation dataset.
- In most cases, even with only a small number of hidden nodes—often as few as a handful—the neural network can offer a tremendous amount of learning ability.

Training neural networks with back propagation

- Training a neural network by adjusting connection weights is very computationally intensive.
- The algorithm, which used a strategy of back-propagating errors, is now known simply as back propagation.
- In its most general form, the back propagation algorithm iterates through many cycles of two processes. Each iteration of the algorithm is known as an **epoch.**

- Because the network contains no a priori (existing) knowledge, typically the weights are set randomly prior to beginning. Then, the algorithm cycles through the processes until a stopping criterion is reached. The cycles include:

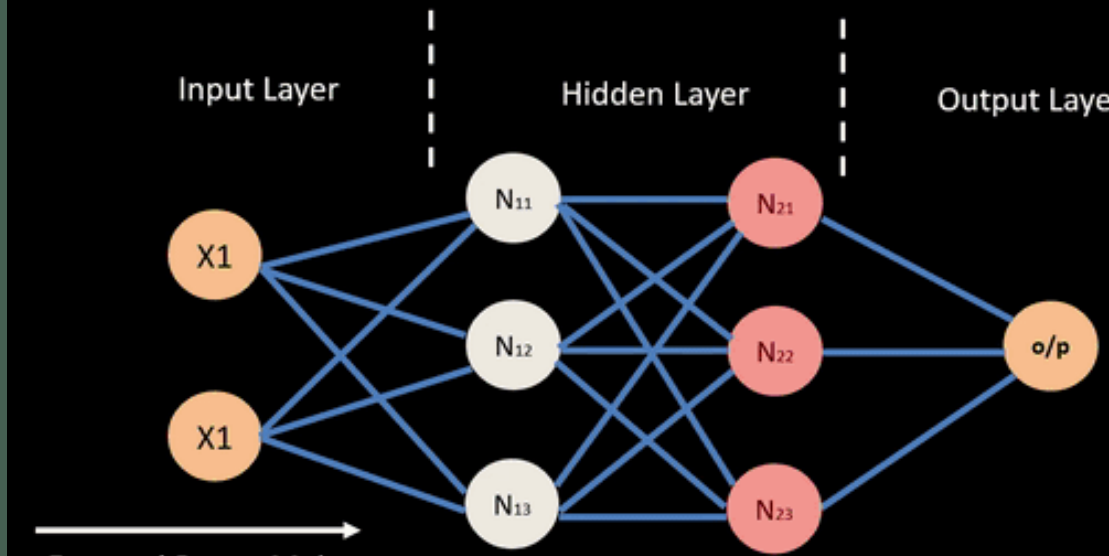
- BP has two phases:

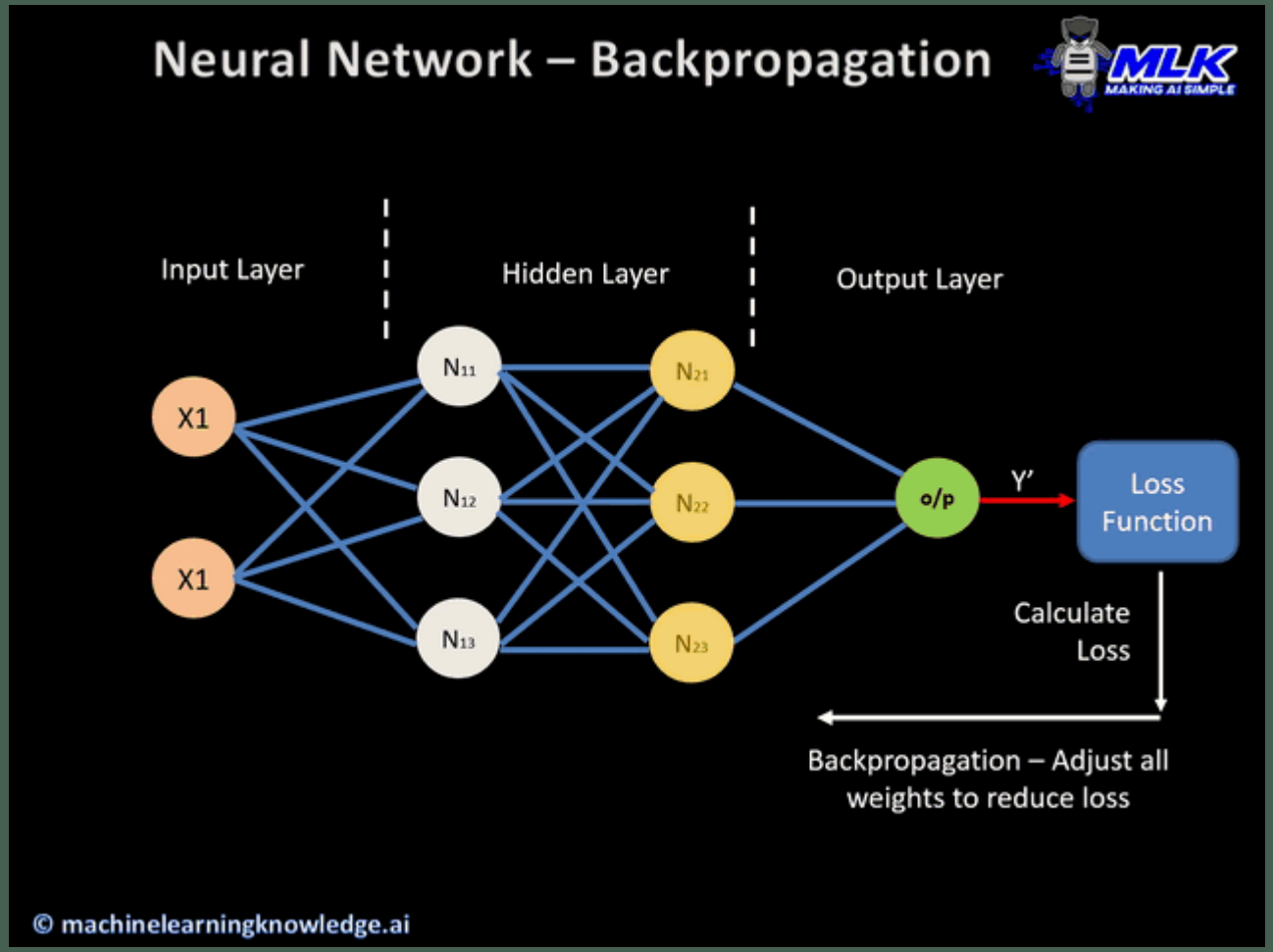
  Forward pass phase: computes 'functional signal', feed forward propagation of input pattern signals through network

  Backward pass phase: computes 'error signal', propagates the error backwards through network starting at output units (where the error is the difference between actual and desired output values)

- **A forward phase** in which the neurons are activated in sequence **from the input layer to the output layer, applying each neuron's weights and activation function along the way.** Upon reaching the final layer, an output signal is produced.
- **A backward phase** in which the network's **output signal resulting from the forward phase is compared to the true target value in the training data. The difference between the network's output signal and the true value results in an error that is propagated backwards in the network to modify the connection weights between neurons and reduce future errors.**

Neural Network – Backpropagation




Neural Network – Backpropagation
MLK
MAKING AI SIMPLE
Input Layer
Hidden Layer
Output Layer
X1
X1
N11
N12
N13
N21
N22
N23
o/p
Y'
Loss Function
Calculate Loss
Backpropagation – Adjust all weights to reduce loss
© machinelearningknowledge.ai

Loss Function- Example

Problem Type	Output Type	Final Activation Function	Loss Function
Regression	Numerical value	Linear	Mean Squared Error (MSE)
Classification	Binary outcome	Sigmoid	Binary Cross Entropy
Classification	Single label, multiple classes	Softmax	Cross Entropy
Classification	Multiple labels, multiple classes	Sigmoid	Binary Cross Entropy

Mean squared error	$\mathrm{MSE} = \frac{1}{n}\sum_{t=1}^{n} e_t^2$
Root mean squared error	$\mathrm{RMSE} = \sqrt{\frac{1}{n}\sum_{t=1}^{n} e_t^2}$
Mean absolute error	$\mathrm{MAE} = \frac{1}{n}\sum_{t=1}^{n} \lvert e_t \rvert$
Mean absolute percentage error	$\mathrm{MAPE} = \frac{100\%}{n}\sum_{t=1}^{n} \left\lvert \frac{e_t}{y_t} \right\rvert$

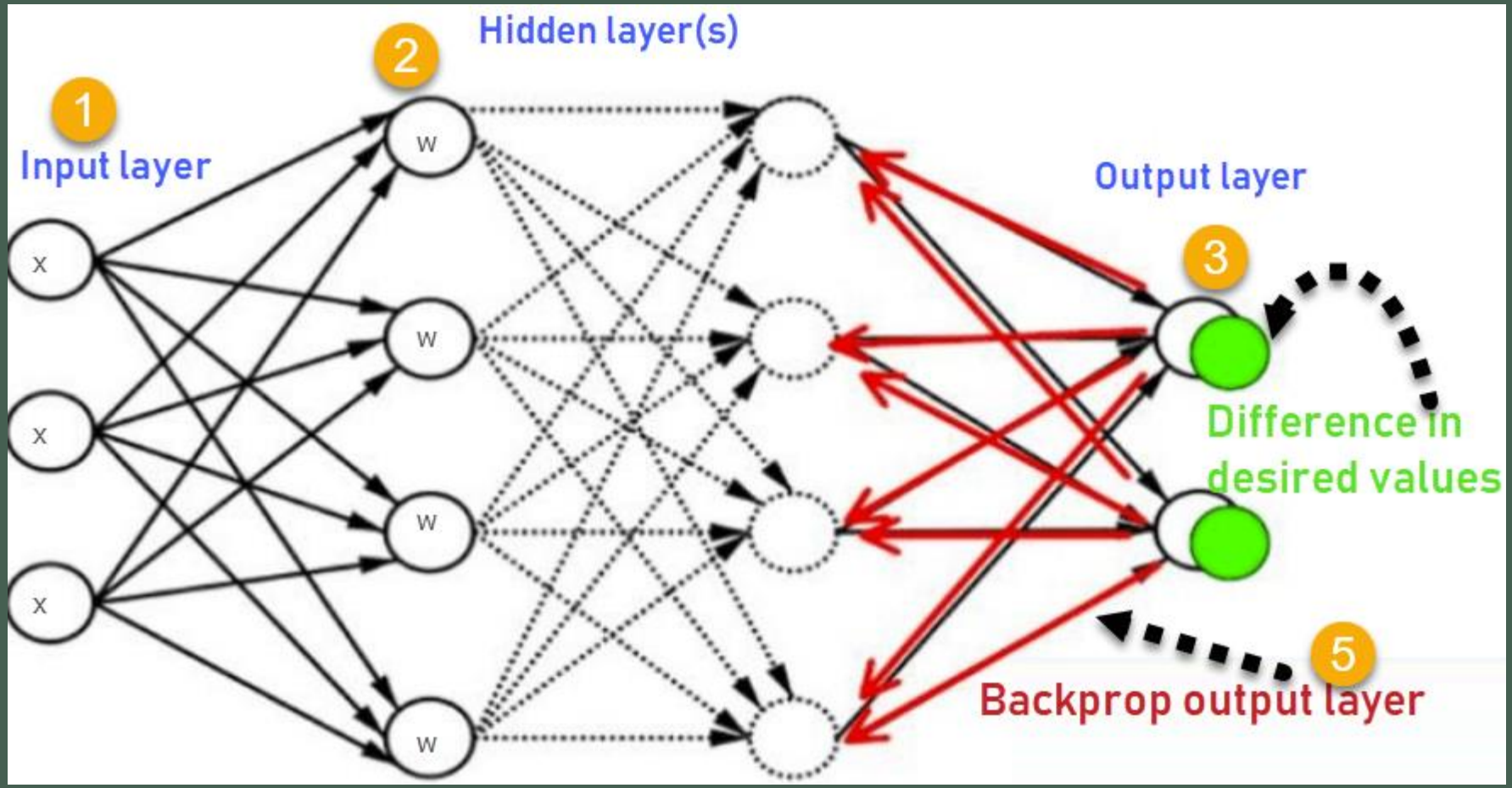
Average of Cross Entropy Loss = function $-\frac{1}{N}\sum_{n=1}^{N}\left[y_n \log \hat{y}_n + (1-y_n)\log(1-\hat{y}_n)\right]$

In Binary Classification problem

If y =0 Loss = $-\frac{1}{N}\sum_{n=1}^{N}\left[\log(1-\hat{y}_n)\right]$

If y=1 Loss = $-\frac{1}{N}\sum_{n=1}^{N}\left[y_n \log \hat{y}\right]$

$\hat{y} = \frac{1}{1+e^{-x}}$ Sigmoid function used in binary classification


Hidden layer(s)
2
1
Input layer
Output layer
3
x
x
x
w
w
w
w
Difference in
desired values
5
Backprop output layer

Back propagation algorithm steps

- 1) Inputs X, arrive through the preconnected path
- 2) Input is modeled using real weights W. The weights are usually randomly selected.
- 3) Calculate the output for every neuron from the input layer, to the hidden layers, to the output layer.
- 4) Calculate the error in the outputs
  - **ErrorB= Actual Output – Predicted Output**

5) Travel back from the output layer to the hidden layer to adjust the weights such that the error is decreased.

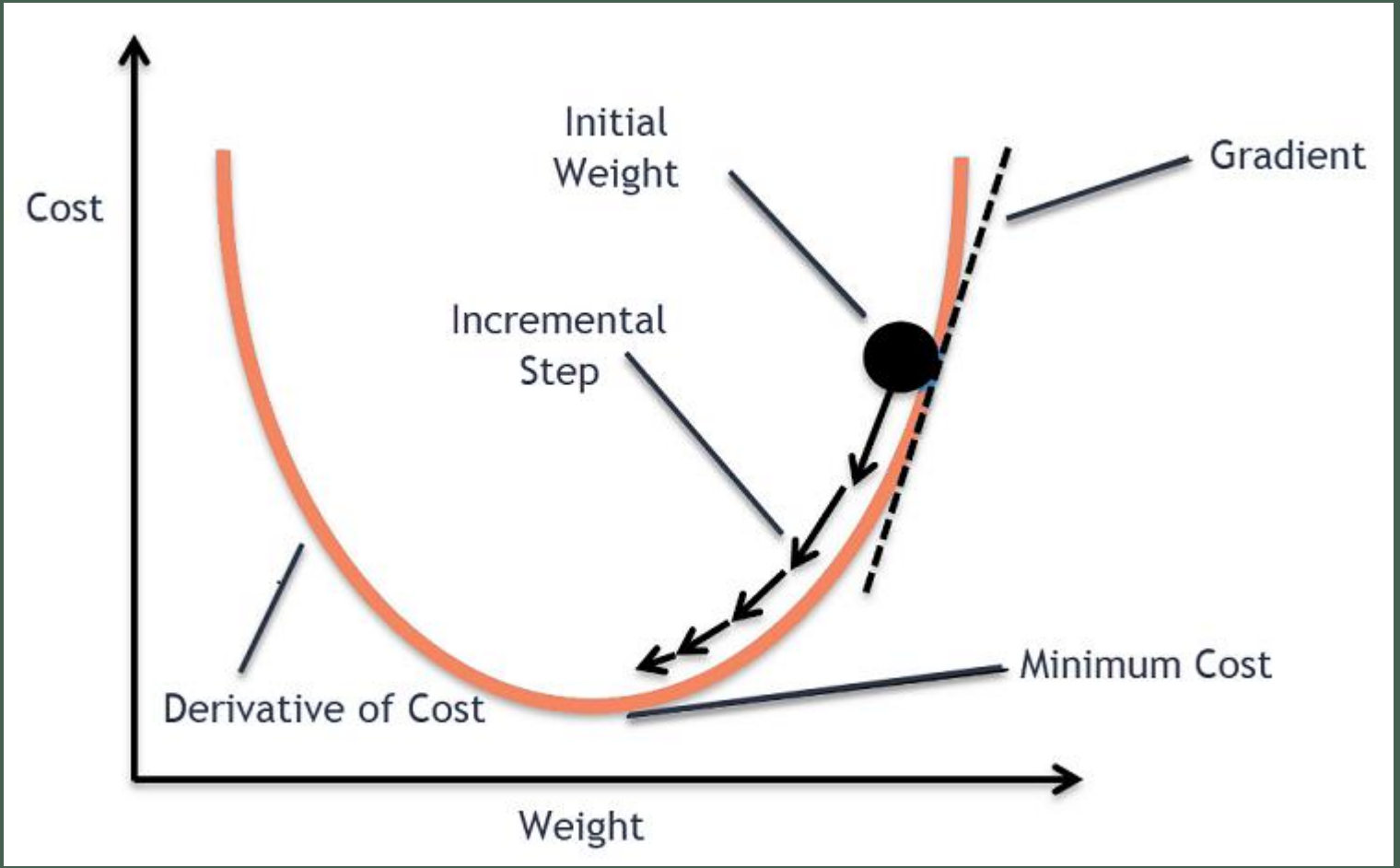
6) Keep repeating the process until the desired output is achieved

Gradient Descent

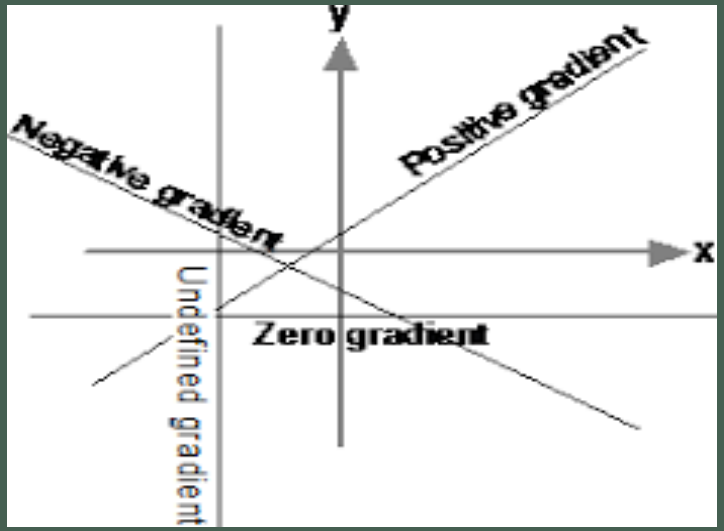
- Over time, the network uses the information sent backward to reduce the total error of the network.
- An Important question here is
- **How does the algorithm determine how much (or whether) a weight should be changed?**
- The answer to this question involves a technique called **gradient descent.**

- The back propagation algorithm uses the derivative of each neuron's activation function to identify the gradient in the direction of each of the incoming weights.
- The gradient suggests how steeply the error will be reduced or increased for a change in the weight. The algorithm will attempt to change the weights that result in the greatest reduction in error by an amount known as the **learning rate.**
- The greater the learning rate, the faster the algorithm will attempt to descend down the gradients, which could reduce training time.

Gradient descent

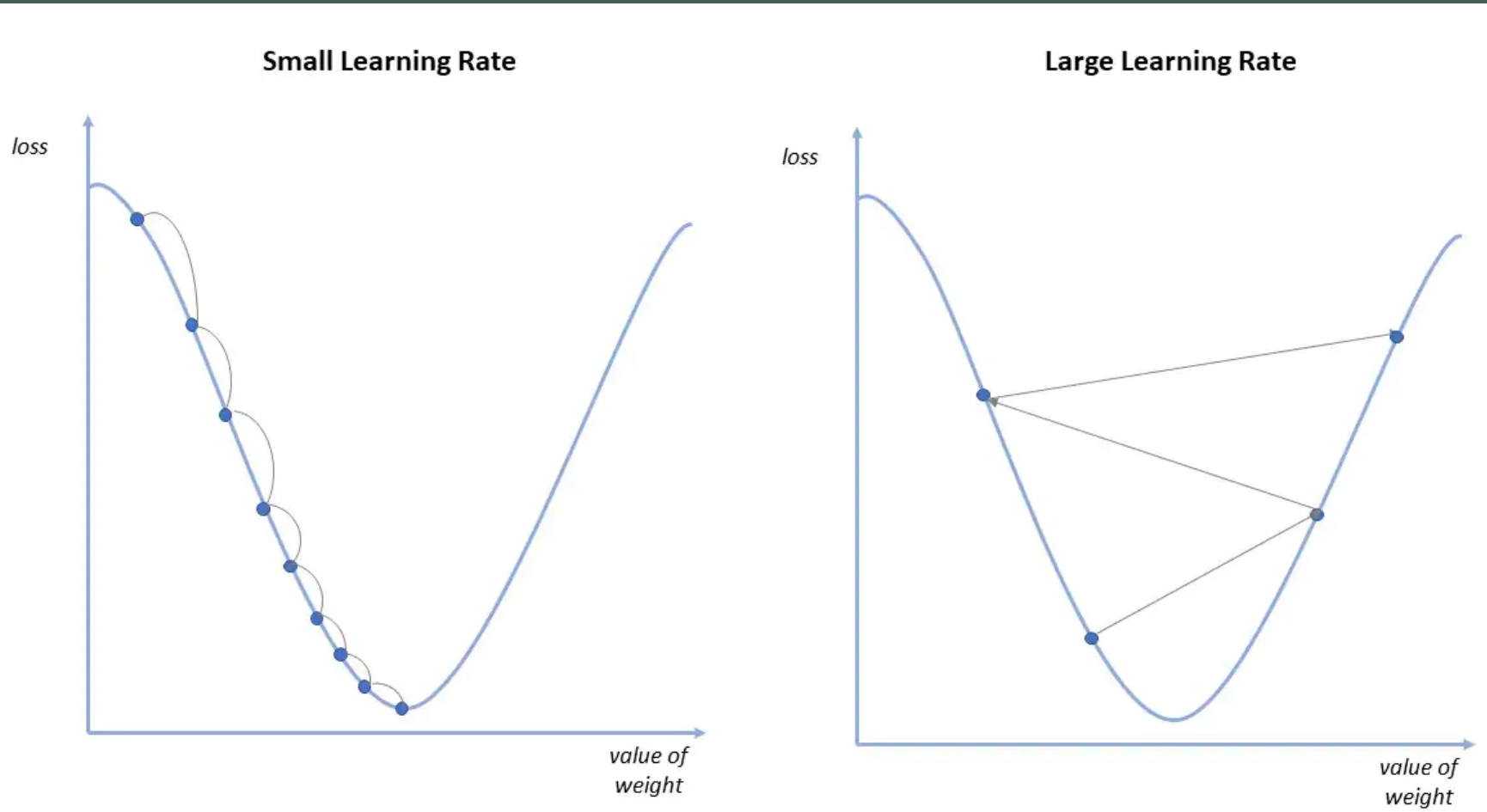


- The gradient descent algorithm takes a step in the direction of the negative gradient in order to reduce loss as quickly as possible.



Learning rate

- The learning rate controls how quickly the model is adapted to the problem.
- Smaller learning rates require more training epochs given the smaller changes made to the weights each update, whereas larger learning rates result in rapid changes and require fewer training epochs


Small Learning Rate
loss
value of weight
Large Learning Rate
loss
value of weight