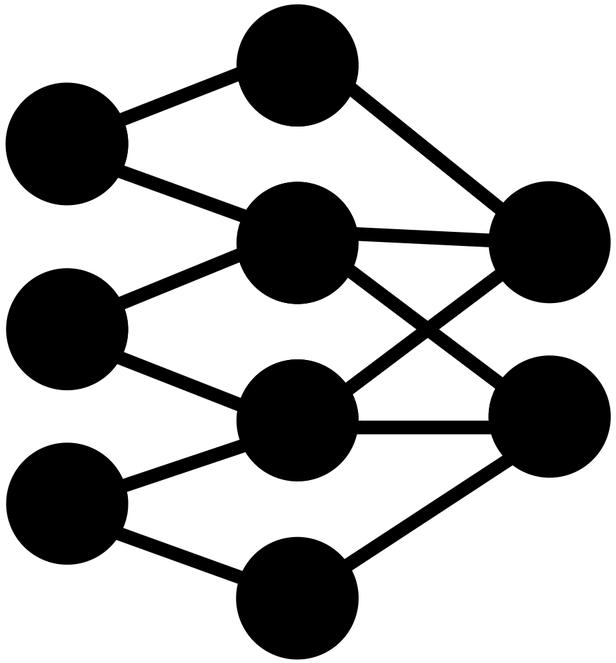
ARTIFICIAL INTELLIGENCE

BACS2003|BACS3074|BMCS2003

CHAPTER 9 ARTIFICIAL NEURAL NETWORK

OUTCOMES

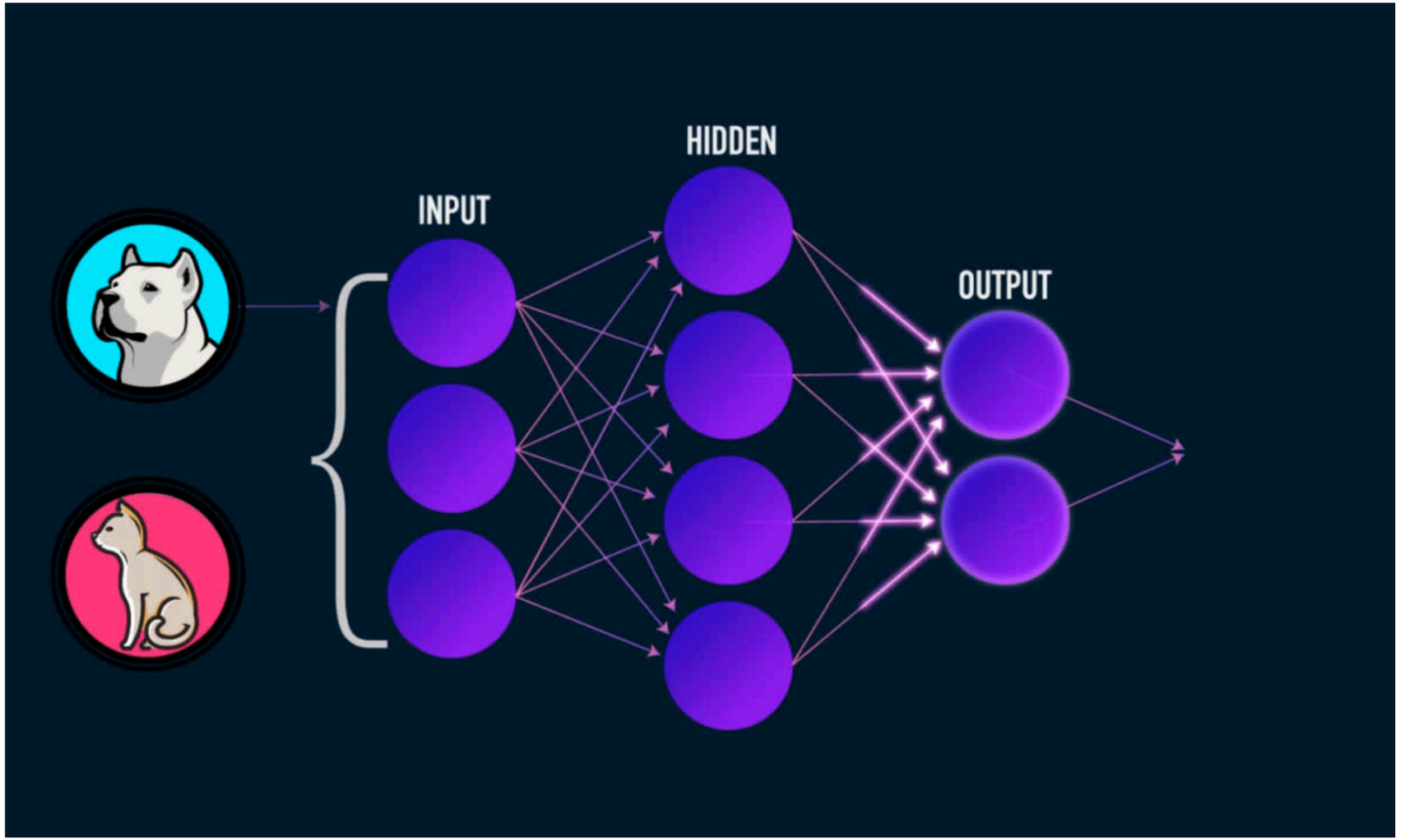
1. **Biological Neural Networks**
2. **Perceptron (Learning & Prediction)**
3. **Architecture of Multilayer neural networks**



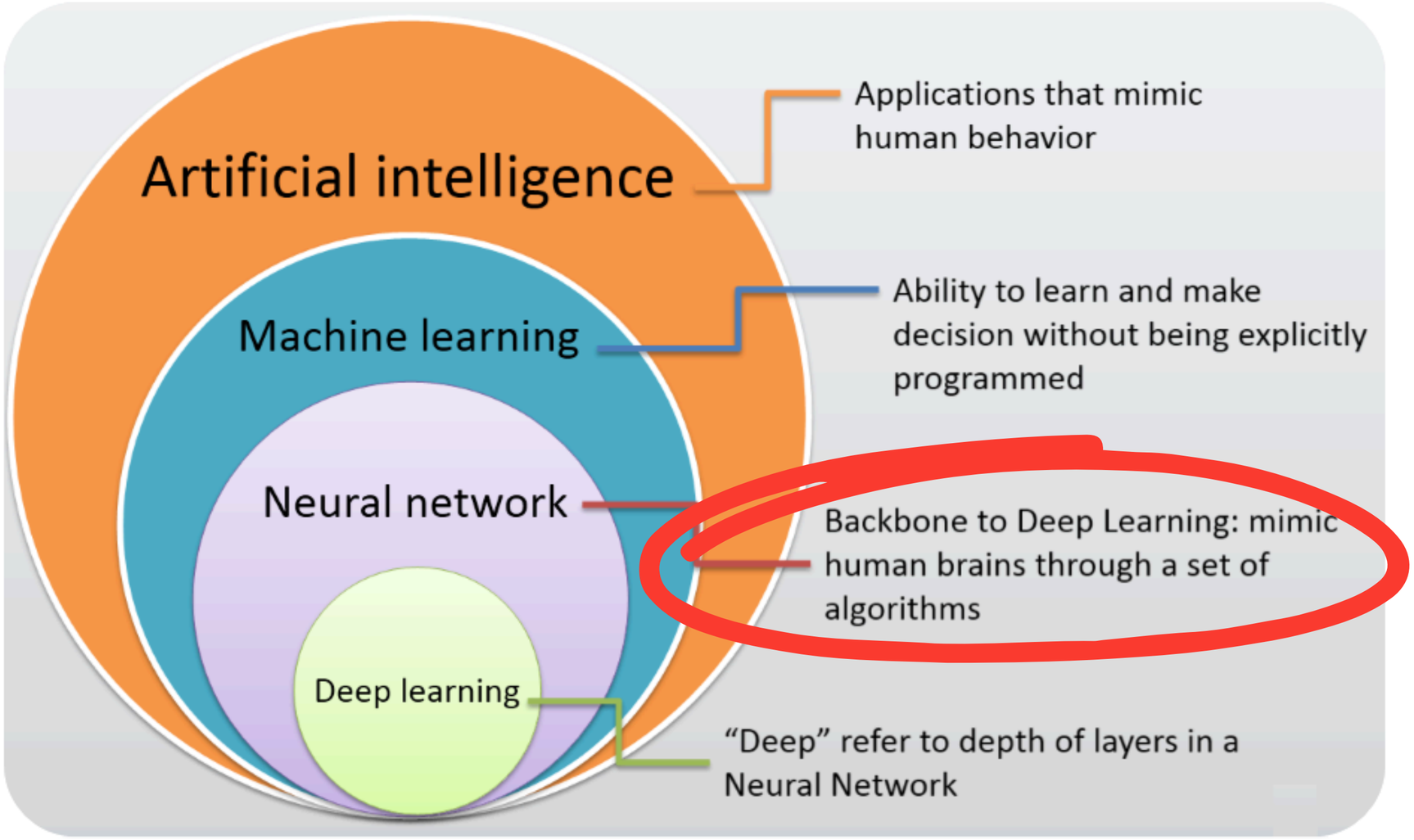
ARTIFICIAL NEURAL NETWORKS

Also known as

- Neural Networks
- Neural Net
- ANN

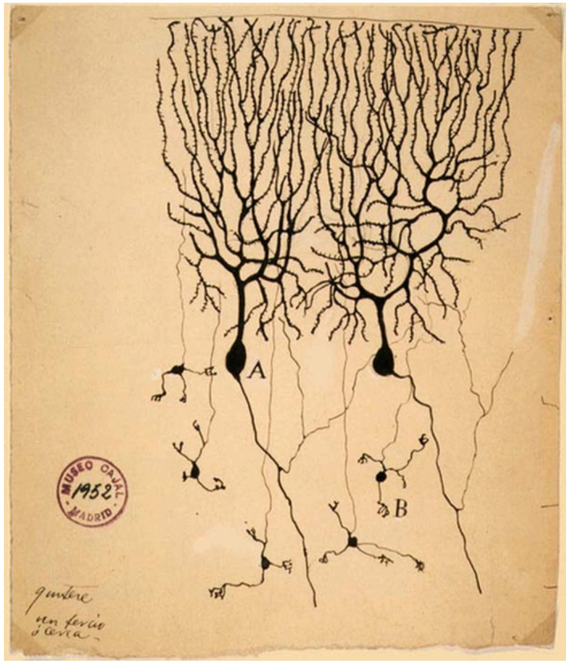


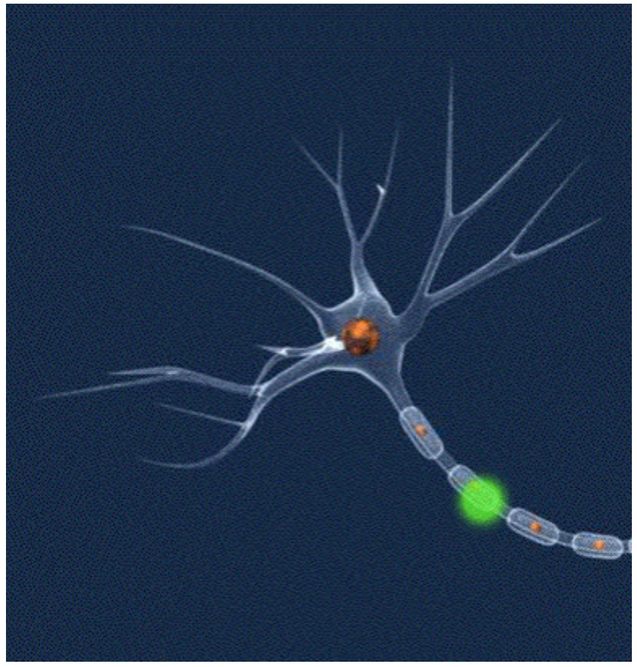
ARTIFICIAL NEURAL NETWORKS

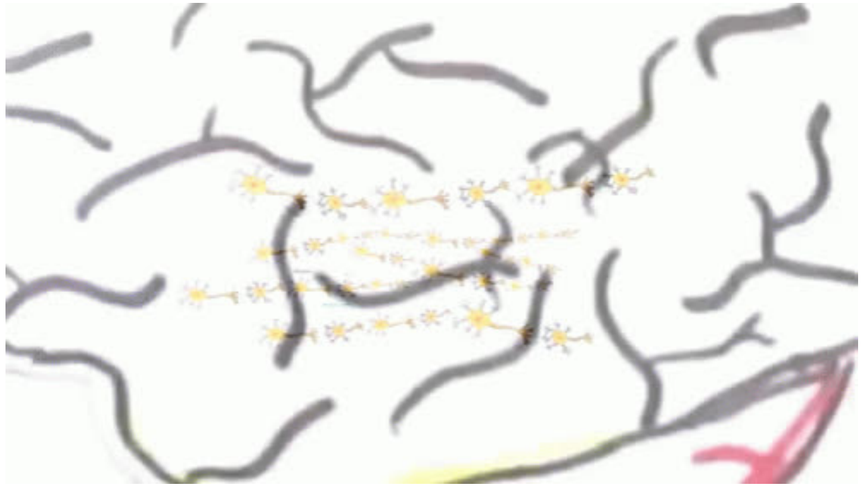


BRAIN

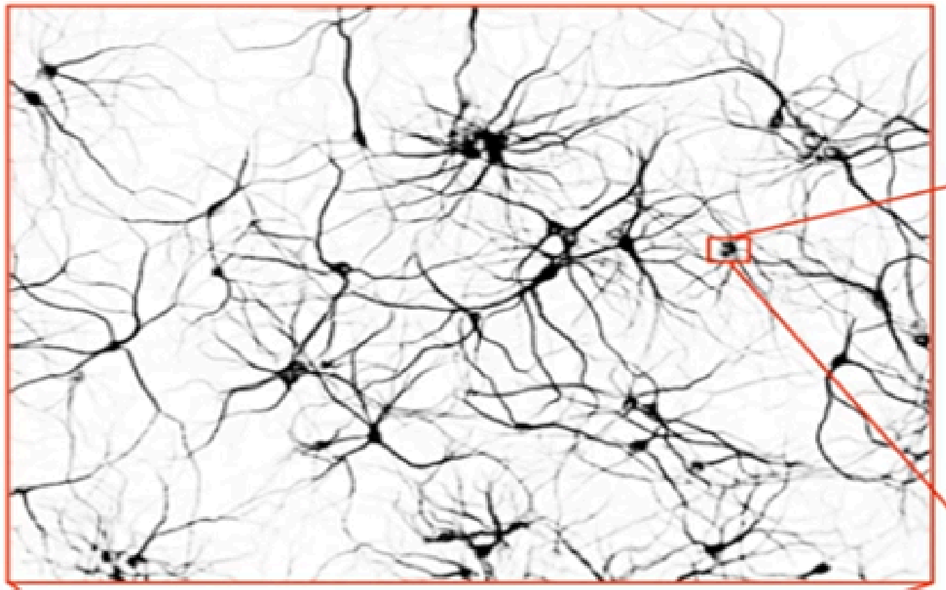
Reasoning based on the human brain. The human brain incorporates nearly 100 billion interconnected set of neurons and 60 trillion connections, synapses, between them.

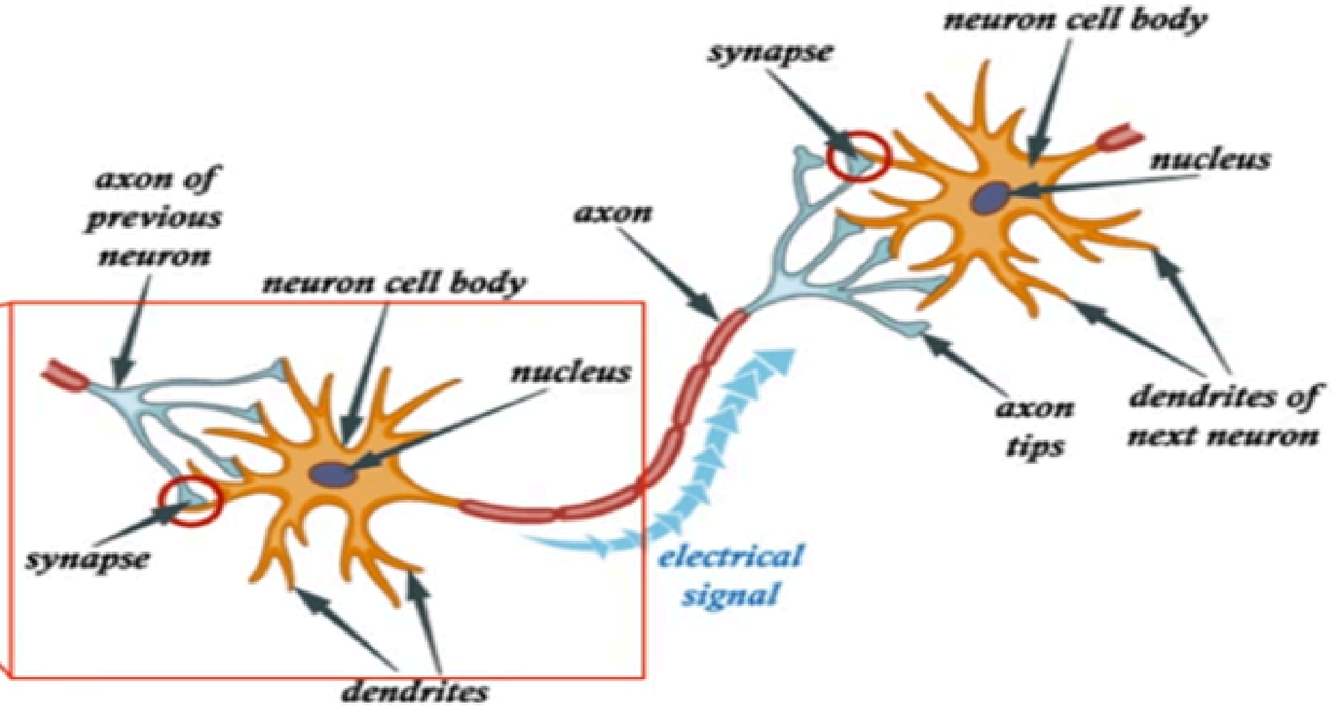






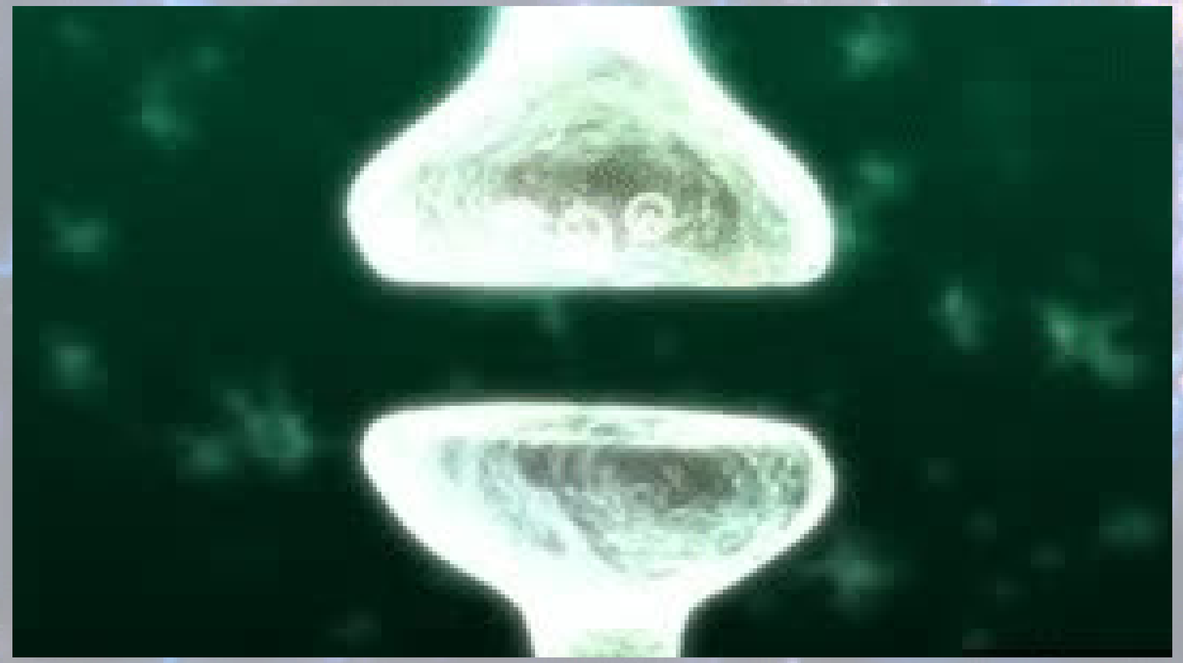
BRAIN

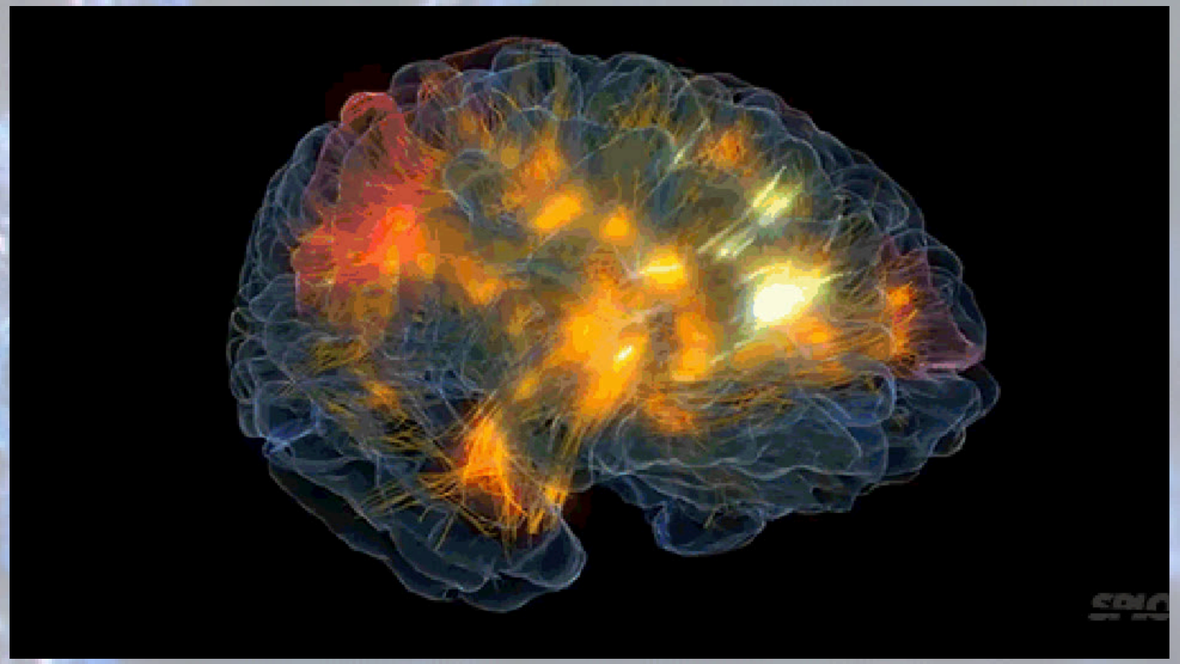




Soma	The basic cell body
Dendrite	Receive signals from axon
Axon	Propagate signals from one neuron to another
Synapse	Release chemical substance to cause the change in the electrical potential of the cell body

SYNAPSE

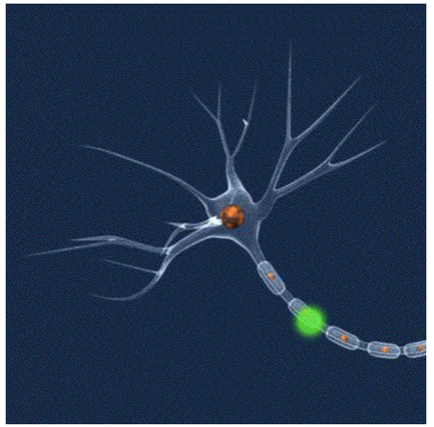


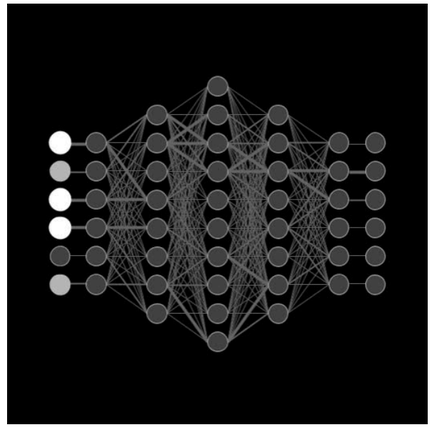


ARTIFICIAL NEURAL NETWORK

- An artificial neural network consists of a number of very simple processors, also called neurons, which are analogous to the biological neurons in the brain.
- The neurons are connected by weighted links passing signals from one neuron to another.

ANALOGY BNN VS ANN

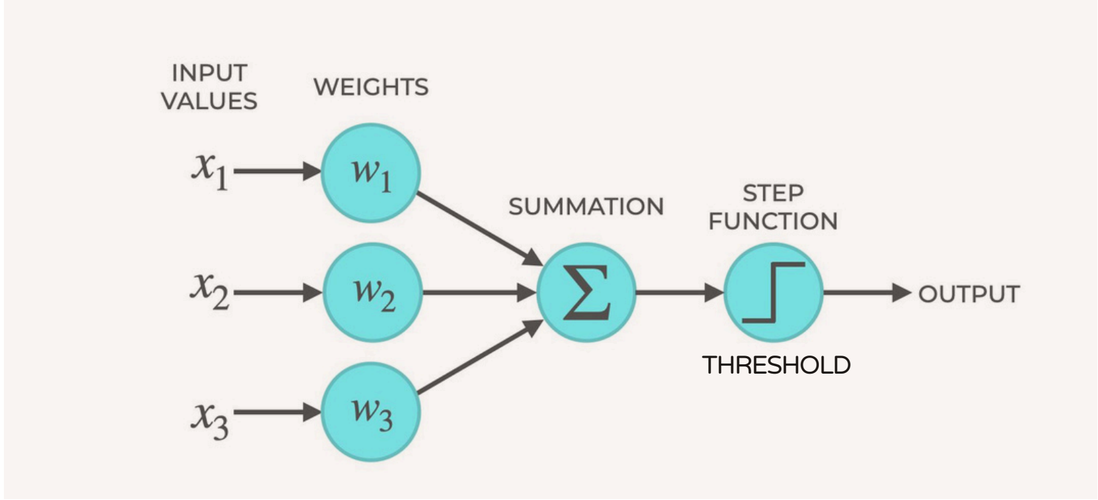




BNN	ANN	Functions
Soma	Neuron	Information processing
Dendrite	Input	Receive data
Axon	Output	Transmit data to another neuron
Synapse	Weight	To express the strength or the importance of neuron input

PERCEPTRON

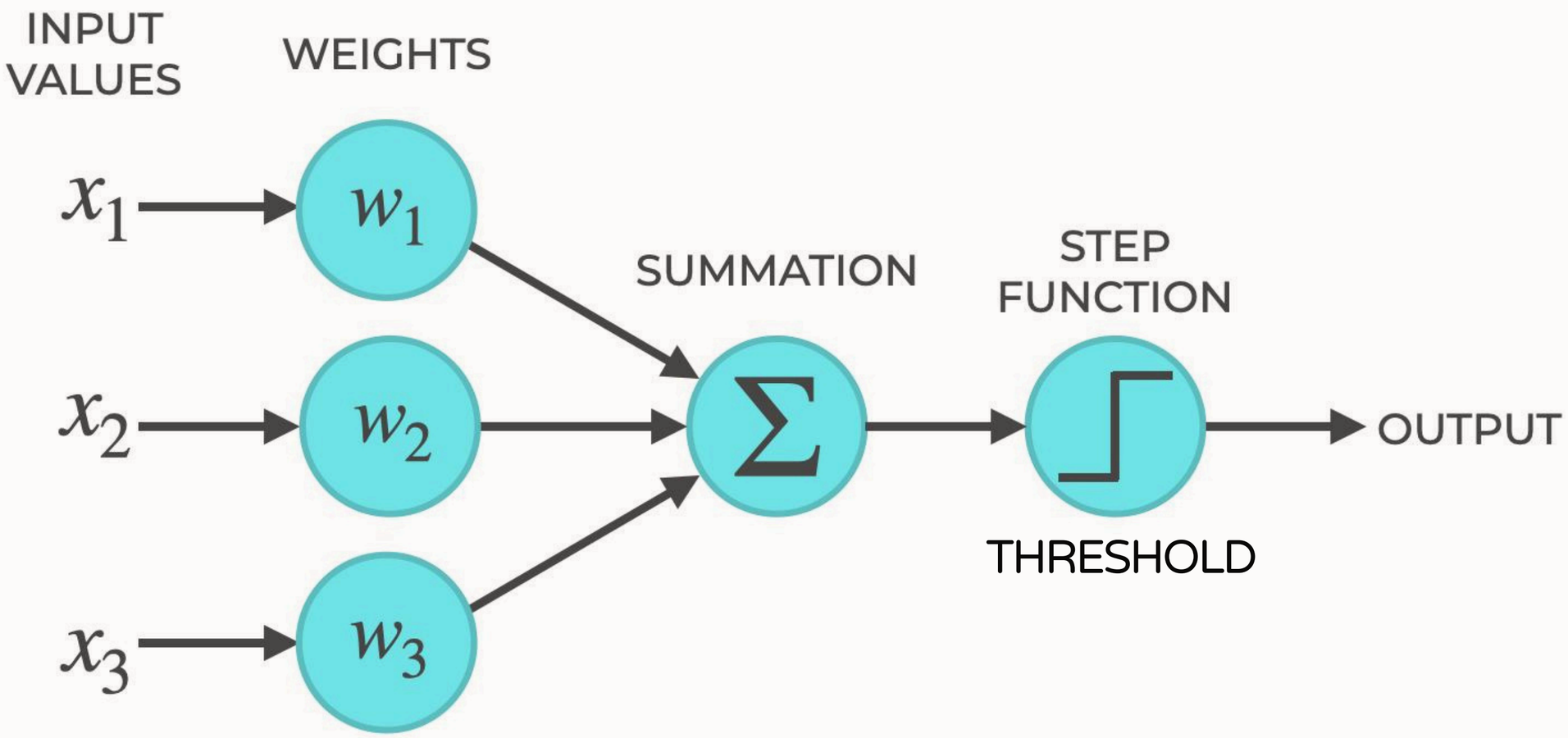
Single-layer perceptron/ single neuron model



1. Frank Rosenblatt, 1958
2. based on the McCulloch and Pitts neuron model
3. It consists of
   - a single neuron
   - adjustable synaptic weights
   - a hard limiter (step function)
   - bias

PERCEPTRON

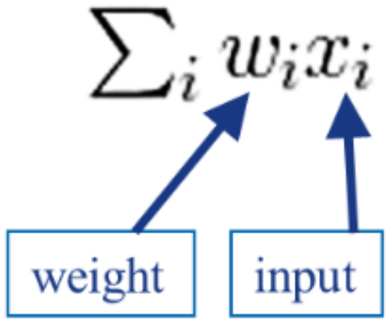
Single-layer perceptron/ single neuron model



SUMMATION

The neuron computes the weighted sum of the input signals

weighted sum:



A weight represents the strength of the connection between units.

ACTIVATION FUNCTION

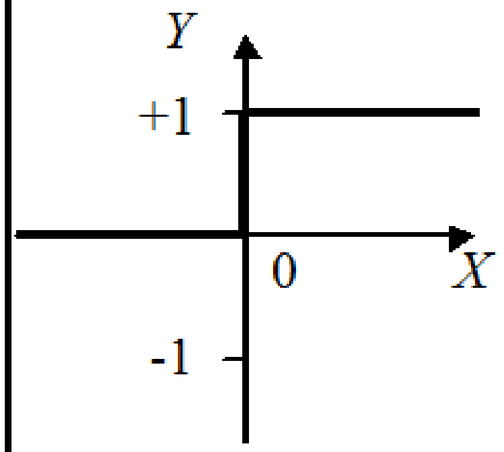
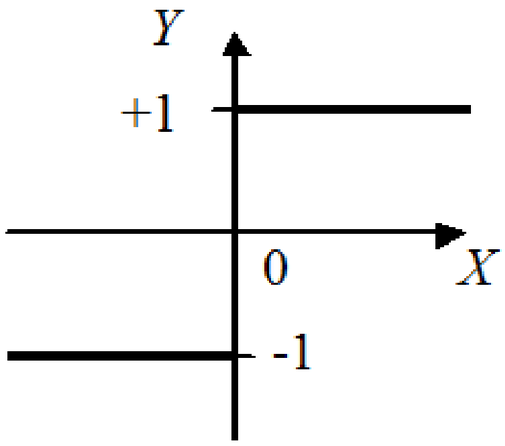
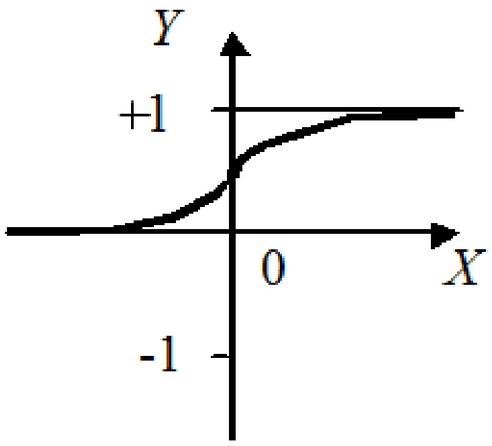
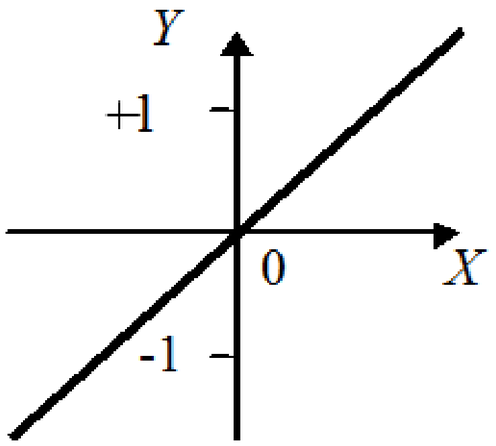
1. Used to determine the output of the perceptron.

2. This type of activation function is called a **step function**.

$$output = \begin{cases} 0 \text{ if } \sum_i w_i x_i < threshold \\ 1 \text{ if } \sum_i w_i x_i \geqslant threshold \end{cases}$$

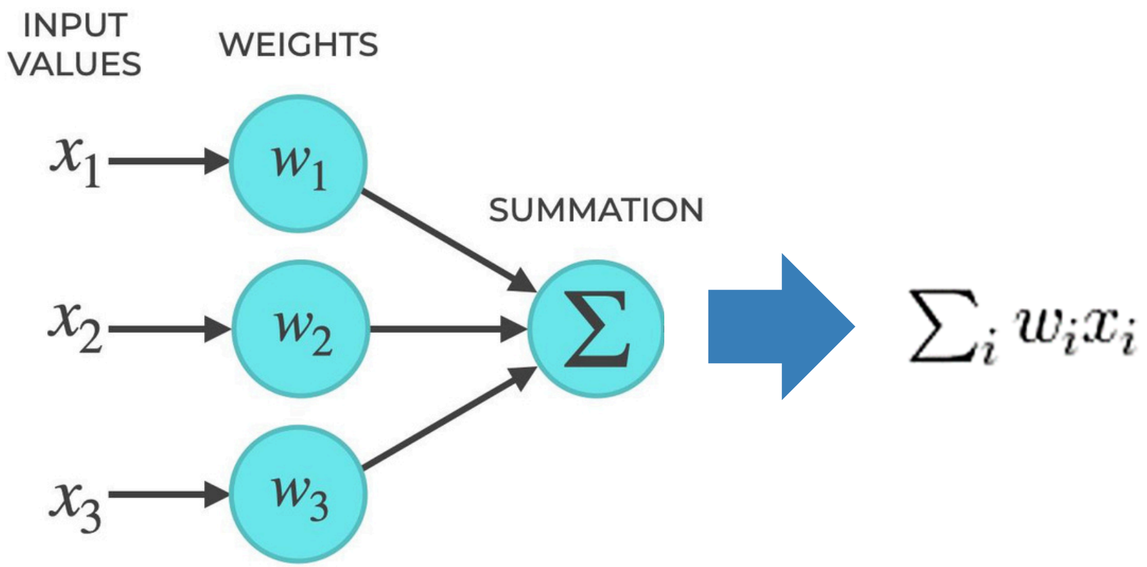
$$output = \begin{cases} 0 \text{ if } \sum_i w_i x_i + bias < 0 \\ 1 \text{ if } \sum_i w_i x_i + bias \geqslant 0 \end{cases}$$

TYPES OF ACTIVATION FUNCTION

Step function	Sign function	Sigmoid function	Linear function
			
$Y^{step} = \begin{cases} 1, & \text{if } X \geq 0 \\ 0, & \text{if } X < 0 \end{cases}$	$Y^{sign} = \begin{cases} +1, & \text{if } X \geq 0 \\ -1, & \text{if } X < 0 \end{cases}$	$Y^{sigmoid} = \dfrac{1}{1+e^{-X}}$	$Y^{linear} = X$

COMBINE ALL: PREDICTION

1. The neuron computes the weighted sum of the unput signals.



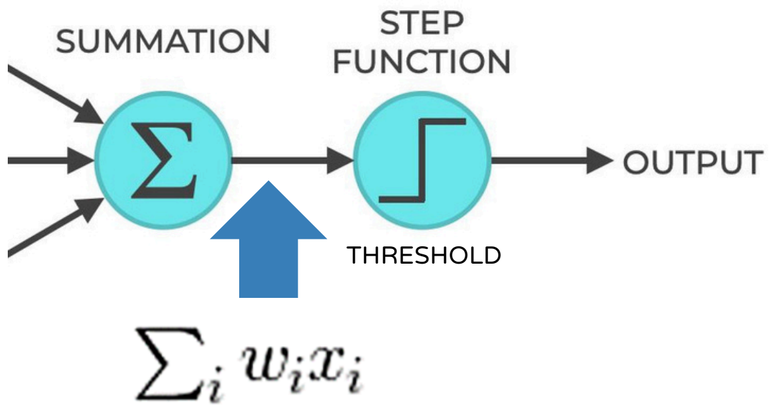
COMBINE ALL: PREDICTION

2. The weighted sum will then send to activation function to generate the output by comparing to a threshold value.

$$output = \begin{cases} 0 \text{ if } \sum_i w_i x_i < threshold \\ 1 \text{ if } \sum_i w_i x_i \geqslant threshold \end{cases}$$

OR

$$output = \begin{cases} 0 \text{ if } \sum_i w_i x_i + bias < 0 \\ 1 \text{ if } \sum_i w_i x_i + bias \geqslant 0 \end{cases}$$



The output is binary, 1 or 0 representing yes or no/ true or false/ female or male and etc.

PERCEPTRON'S TRAINING ALGORITHM

Step 1: Initialization

Set initial weights *w1, w2,…, wn* and threshold θ to random numbers in the range [-0.5, 0.5].

Step 2: Activation

Activate the perceptron by applying inputs $x_1(p)$, $x_2(p)$,…, $x_n(p)$ and desired output Y(p). Calculate the predicted output at iteration p = 1

$$Y(p) = step\left[\sum_{i=1}^{n} x_i(p)\, w_i(p) - \theta\right]$$

OR

$$output = \begin{cases} 0 \text{ if } \sum_i w_i x_i + bias < 0 \\ 1 \text{ if } \sum_i w_i x_i + bias \geqslant 0 \end{cases}$$

where n is the number of the perceptron inputs, and step is a step activation function

PERCEPTRON'S TRAINING ALGORITHM

Step 3: Weight training

Update the weights of the perceptron

$$w_i(p+1) = w_i(p) + \Delta w_i(p)$$

where $\Delta w_i(p)$ is the weight correction at iteration p.

- The weight correction is computed by the delta rule:

$$\Delta w_i(p) = \alpha \cdot x_i(p) \cdot e(p)$$

- learning rate, $\alpha = 0.1$
- Error, $e(p) = Yd(p) - Y(p)$

Step 4: Iteration

Increase iteration p by one, go back to Step 2 and repeat the process until convergence

PERCEPTRON'S TRAINING ALGORITHM

Step 1: Initialization

Set initial weights *w1, w2,…, wn* and threshold θ to random numbers in the range [-0.5, 0.5].

Step 2: Activation

Activate the perceptron by applying inputs $x_1(p)$, $x_2(p)$,…, $x_n(p)$ and desired output Y(p). Calculate the predicted output at iteration p = 1

$$Y(p) = step\left[\sum_{i=1}^{n} x_i(p)\, w_i(p) - \theta\right]$$

where n is the number of the perceptron inputs, and step is a step activation function

Step 3: Weight training

Update the weights of the perceptron

$$w_i(p+1) = w_i(p) + \Delta w_i(p)$$

where $\Delta w_i(p)$ is the weight correction at iteration p.

- The weight correction is computed by the delta rule:

$$\Delta w_i(p) = \alpha \cdot x_i(p) \cdot e(p)$$

- learning rate, $\alpha = 0.1$
- Error, $e(p) = Yd(p) - Y(p)$

Step 4: Iteration

Increase iteration p by one, go back to Step 2 and repeat the process until convergence

$W_1 = 0.3$, $W_2 = -0.1$, $\theta = 0.2$, $\alpha = 0.1$

Epoch	Inputs		Desired output	Initial weights		Predicted Output	Error	Final weights	
	x_1	x_2	Y_d	w_1	w_2	Y	e	w_1	w_2
1	0	0	0	0.3	−0.1	0			
First round	0	1	0						
of training	1	0	0						
(Epoch 1)	1	1	1						

PERCEPTRON'S TRAINING ALGORITHM

Step 1: Initialization

Set initial weights *w1, w2,…, wn* and threshold θ to random numbers in the range [-0.5, 0.5].

Step 2: Activation

Activate the perceptron by applying inputs $x_1(p)$, $x_2(p)$,…, $x_n(p)$ and desired output Y(p). Calculate the predicted output at iteration p = 1

$$Y(p) = step\left[\sum_{i=1}^{n} x_i(p)\, w_i(p) - \theta\right]$$

where n is the number of the perceptron inputs, and step is a step activation function

Step 3: Weight training

Update the weights of the perceptron

$$w_i(p+1) = w_i(p) + \Delta w_i(p)$$

where $\Delta w_i(p)$ is the weight correction at iteration p.

- The weight correction is computed by the delta rule:

$$\Delta w_i(p) = \alpha \cdot x_i(p) \cdot e(p)$$

- learning rate, $\alpha = 0.1$
- Error, $e(p) = Yd(p) - Y(p)$

Step 4: Iteration

Increase iteration p by one, go back to Step 2 and repeat the process until convergence

$W_1 = 0.3$, $W_2 = -0.1$, $\theta = 0.2$, $\alpha = 0.1$

Epoch	Inputs		Desired output	Initial weights		Predicted Output	Error	Final weights	
	x_1	x_2	Y_d	w_1	w_2	Y	e	w_1	w_2
1	0	0	0	0.3	−0.1	0			
First round of training (Epoch 1)	0	1	0						
	1	0	0						
	1	1	1						

= 0 X 0.3 + 0 X (- 0.1) + (- 0.2)

= - 0.2

SINCE -0.2 < 0 , PREDICTED OUTPUT = 0

PERCEPTRON'S TRAINING ALGORITHM

Step 1: Initialization

Set initial weights *w1, w2,…, wn* and threshold θ to random numbers in the range [-0.5, 0.5].

Step 2: Activation

Activate the perceptron by applying inputs $x_1(p)$, $x_2(p)$,…, $x_n(p)$ and desired output Y(p). Calculate the predicted output at iteration p = 1

$$Y(p) = step\left[\sum_{i=1}^{n} x_i(p)\, w_i(p) - \theta\right]$$

where n is the number of the perceptron inputs, and step is a step activation function

Step 3: Weight training

Update the weights of the perceptron

$$w_i(p+1) = w_i(p) + \Delta w_i(p)$$

where $\Delta w_i(p)$ is the weight correction at iteration p.

- The weight correction is computed by the delta rule:

$$\Delta w_i(p) = \alpha \cdot x_i(p) \cdot e(p)$$

- learning rate, $\alpha = 0.1$
- Error, $e(p) = Yd(p) - Y(p)$

Step 4: Iteration

Increase iteration p by one, go back to Step 2 and repeat the process until convergence

$W_1 = 0.3,\ W_2 = -0.1,\ \theta = 0.2,\ \alpha = 0.1$

Epoch	Inputs		Desired output	Initial weights		Predicted Output	Error	Final weights	
	x_1	x_2	Y_d	w_1	w_2	Y	e	w_1	w_2
1	0	0	0	0.3	−0.1	0	0		
First round of training (Epoch 1)	0	1	0						
	1	0	0						
	1	1	1						

ERROR: DESIRED OUTPUT - PREDICTED OUTPUT

= 0-0

= 0

PERCEPTRON'S TRAINING ALGORITHM

Step 1: Initialization

Set initial weights *w1, w2,…, wn* and threshold θ to random numbers in the range [-0.5, 0.5].

Step 2: Activation

Activate the perceptron by applying inputs $x_1(p)$, $x_2(p)$,…, $x_n(p)$ and desired output Y(p). Calculate the predicted output at iteration p = 1

$$Y(p) = step\left[\sum_{i=1}^{n} x_i(p)\, w_i(p) - \theta\right]$$

where n is the number of the perceptron inputs, and step is a step activation function

Step 3: Weight training

Update the weights of the perceptron

$$w_i(p+1) = w_i(p) + \Delta w_i(p)$$

where $\Delta w_i(p)$ is the weight correction at iteration p.

- The weight correction is computed by the delta rule:

$$\Delta w_i(p) = \alpha \cdot x_i(p) \cdot e(p)$$

- learning rate, $\alpha = 0.1$
- Error, $e(p) = Yd(p) - Y(p)$

Step 4: Iteration

Increase iteration p by one, go back to Step 2 and repeat the process until convergence

$W_1 = 0.3,\ W_2 = -0.1,\ \theta = 0.2,\ \alpha = 0.1$

Epoch	Inputs		Desired output	Initial weights		Predicted Output	Error	Final weights	
	x_1	x_2	Y_d	w_1	w_2	Y	e	w_1	w_2
1	0	0	0	0.3	−0.1	0	0	0.3	−0.1
First round of training (Epoch 1)	0	1	0						
	1	0	0						
	1	1	1						

FINAL WEIGHTS, W_1

= 0.3 + 0.1 X 0 X 0

= 0.3

FINAL WEIGHTS, W_2

= -0.1 + 0.1 X 0 X 0

= -0.1

PERCEPTRON'S TRAINING ALGORITHM

Step 1: Initialization

Set initial weights *w1, w2,…, wn* and threshold θ to random numbers in the range [-0.5, 0.5].

Step 2: Activation

Activate the perceptron by applying inputs $x_1(p)$, $x_2(p)$,…, $x_n(p)$ and desired output Y(p). Calculate the predicted output at iteration p = 1

$$Y(p) = step\left[\sum_{i=1}^{n} x_i(p)\, w_i(p) - \theta\right]$$

where n is the number of the perceptron inputs, and step is a step activation function

Step 3: Weight training

Update the weights of the perceptron

$$w_i(p+1) = w_i(p) + \Delta w_i(p)$$

where $\Delta w_i(p)$ is the weight correction at iteration p.

- The weight correction is computed by the delta rule:

$$\Delta w_i(p) = \alpha \cdot x_i(p) \cdot e(p)$$

- learning rate, $\alpha = 0.1$
- Error, $e(p) = Yd(p) - Y(p)$

Step 4: Iteration

Increase iteration p by one, go back to Step 2 and repeat the process until convergence

$W_1 = 0.3$, $W_2 = -0.1$, $\theta = 0.2$, $\alpha = 0.1$

Epoch	Inputs		Desired output	Initial weights		Predicted Output	Error	Final weights	
	x_1	x_2	Y_d	w_1	w_2	Y	e	w_1	w_2
1	0	0	0	0.3	−0.1	0	0	0.3	−0.1
First round of training (Epoch 1)	0	1	0	0.3	−0.1				
	1	0	0						
	1	1	1						

CONVERGENCE

Epoch	Inputs		Desired output	Initial weights		Actual output	Error	Final weights	
	x_1	x_2	Y_d	w_1	w_2	Y	e	w_1	w_2
1	0	0	0	0.3	−0.1	0	0	0.3	−0.1
	0	1	0	0.3	−0.1	0	0	0.3	−0.1
	1	0	0	0.3	−0.1	1	−1	0.2	−0.1
	1	1	1	0.2	−0.1	0	1	0.3	0.0

CONVERGENCE

Epoch	Inputs		Desired output	Initial weights		Actual output	Error	Final weights	
	x_1	x_2	Y_d	w_1	w_2	Y	e	w_1	w_2
1	0	0	0	0.3	−0.1	0	0	0.3	−0.1
	0	1	0	0.3	−0.1	0	0	0.3	−0.1
	1	0	0	0.3	−0.1	1	−1	0.2	−0.1
	1	1	1	0.2	−0.1	0	1	0.3	0.0
2	0	0	0	0.3	0.0	0	0	0.3	0.0
	0	1	0	0.3	0.0	0	0	0.3	0.0
	1	0	0	0.3	0.0	1	−1	0.2	0.0
	1	1	1	0.2	0.0	1	0	0.2	0.0

CONVERGENCE

Epoch	Inputs		Desired output	Initial weights		Actual output	Error	Final weights	
	x_1	x_2	Y_d	w_1	w_2	Y	e	w_1	w_2
1	0	0	0	0.3	−0.1	0	0	0.3	−0.1
	0	1	0	0.3	−0.1	0	0	0.3	−0.1
	1	0	0	0.3	−0.1	1	−1	0.2	−0.1
	1	1	1	0.2	−0.1	0	1	0.3	0.0
2	0	0	0	0.3	0.0	0	0	0.3	0.0
	0	1	0	0.3	0.0	0	0	0.3	0.0
	1	0	0	0.3	0.0	1	−1	0.2	0.0
	1	1	1	0.2	0.0	1	0	0.2	0.0
3	0	0	0	0.2	0.0	0	0	0.2	0.0
	0	1	0	0.2	0.0	0	0	0.2	0.0
	1	0	0	0.2	0.0	1	−1	0.1	0.0
	1	1	1	0.1	0.0	0	1	0.2	0.1

CONVERGENCE

Epoch	Inputs		Desired output	Initial weights		Actual output	Error	Final weights	
	x_1	x_2	Y_d	w_1	w_2	Y	e	w_1	w_2
1	0	0	0	0.3	−0.1	0	0	0.3	−0.1
	0	1	0	0.3	−0.1	0	0	0.3	−0.1
	1	0	0	0.3	−0.1	1	−1	0.2	−0.1
	1	1	1	0.2	−0.1	0	1	0.3	0.0
2	0	0	0	0.3	0.0	0	0	0.3	0.0
	0	1	0	0.3	0.0	0	0	0.3	0.0
	1	0	0	0.3	0.0	1	−1	0.2	0.0
	1	1	1	0.2	0.0	1	0	0.2	0.0
3	0	0	0	0.2	0.0	0	0	0.2	0.0
	0	1	0	0.2	0.0	0	0	0.2	0.0
	1	0	0	0.2	0.0	1	−1	0.1	0.0
	1	1	1	0.1	0.0	0	1	0.2	0.1
4	0	0	0	0.2	0.1	0	0	0.2	0.1
	0	1	0	0.2	0.1	0	0	0.2	0.1
	1	0	0	0.2	0.1	1	−1	0.1	0.1
	1	1	1	0.1	0.1	1	0	0.1	0.1

CONVERGENCE

Epoch	Inputs		Desired output	Initial weights		Actual output	Error	Final weights	
	x_1	x_2	Y_d	w_1	w_2	Y	e	w_1	w_2
1	0	0	0	0.3	−0.1	0	0	0.3	−0.1
	0	1	0	0.3	−0.1	0	0	0.3	−0.1
	1	0	0	0.3	−0.1	1	−1	0.2	−0.1
	1	1	1	0.2	−0.1	0	1	0.3	0.0
2	0	0	0	0.3	0.0	0	0	0.3	0.0
	0	1	0	0.3	0.0	0	0	0.3	0.0
	1	0	0	0.3	0.0	1	−1	0.2	0.0
	1	1	1	0.2	0.0	1	0	0.2	0.0
3	0	0	0	0.2	0.0	0	0	0.2	0.0
	0	1	0	0.2	0.0	0	0	0.2	0.0
	1	0	0	0.2	0.0	1	−1	0.1	0.0
	1	1	1	0.1	0.0	0	1	0.2	0.1
4	0	0	0	0.2	0.1	0	0	0.2	0.1
	0	1	0	0.2	0.1	0	0	0.2	0.1
	1	0	0	0.2	0.1	1	−1	0.1	0.1
	1	1	1	0.1	0.1	1	0	0.1	0.1
5	0	0	0	0.1	0.1	0	0	0.1	0.1
	0	1	0	0.1	0.1	0	0	0.1	0.1
	1	0	0	0.1	0.1	0	0	0.1	0.1
	1	1	1	0.1	0.1	1	0	0.1	0.1

The weights are the same compared to previous round. Stop at Epoch 5!

ANN ALGORITHM

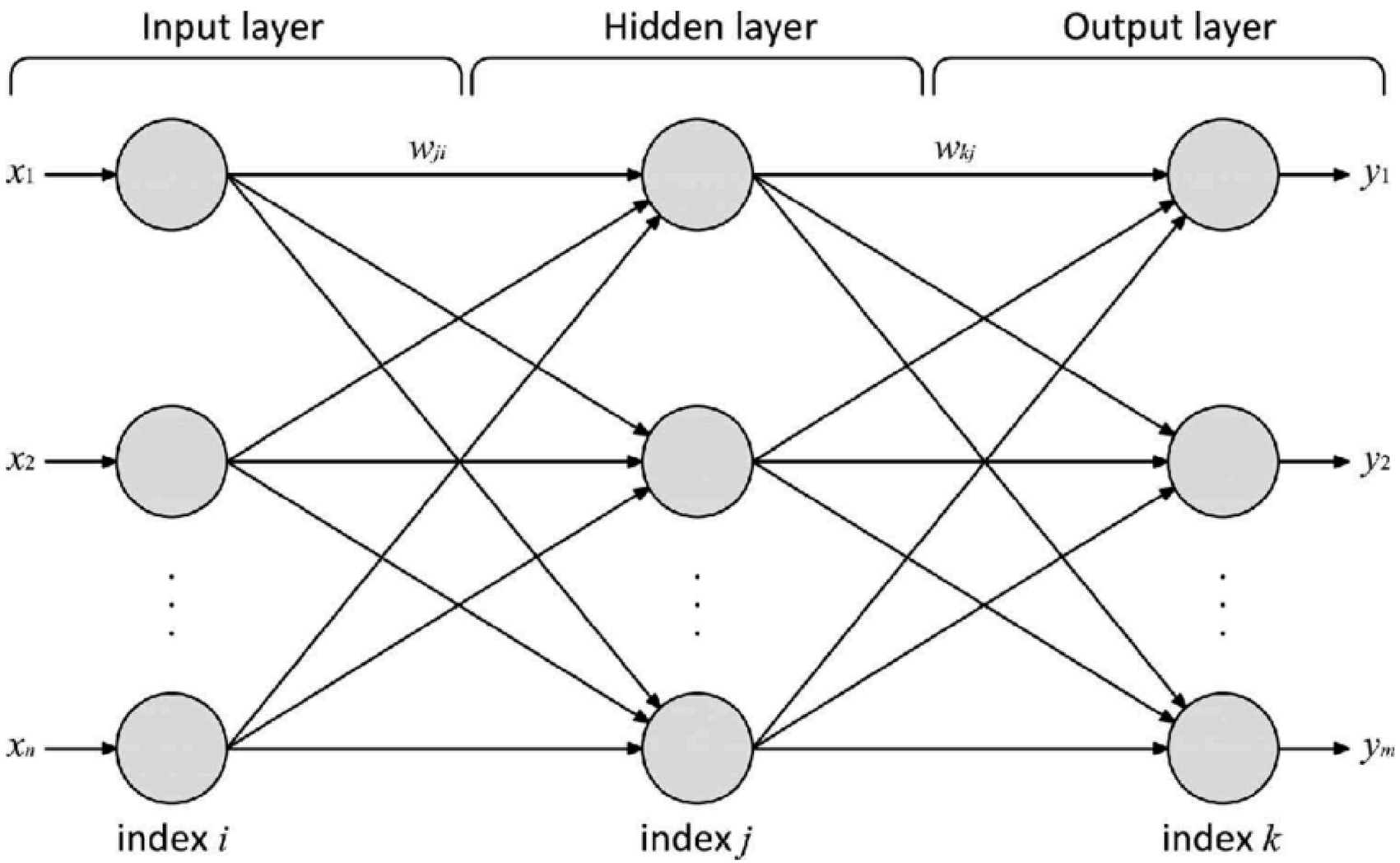
1. Supervised learning
   - Single layer NN
     - Perceptron
   - Multilayer NN
     - Back-propagation
     - Accelerated learning
     - The Hopfield network
     - Bidirectional Associative memory
2. Unsupervised learning
   - Hebbian Learning
   - Competitive Learning

MULTILAYER PERCEPTRON

1. A multilayer perceptron is a feedforward neural network with one or more hidden (middle) layers.
2. The network consists of an input layer of source neurons, at least one middle or hidden layer of computational neurons, and an output layer of computational neurons.
3. The input signals are propagated in a forward direction on a layer-by-layer basis.

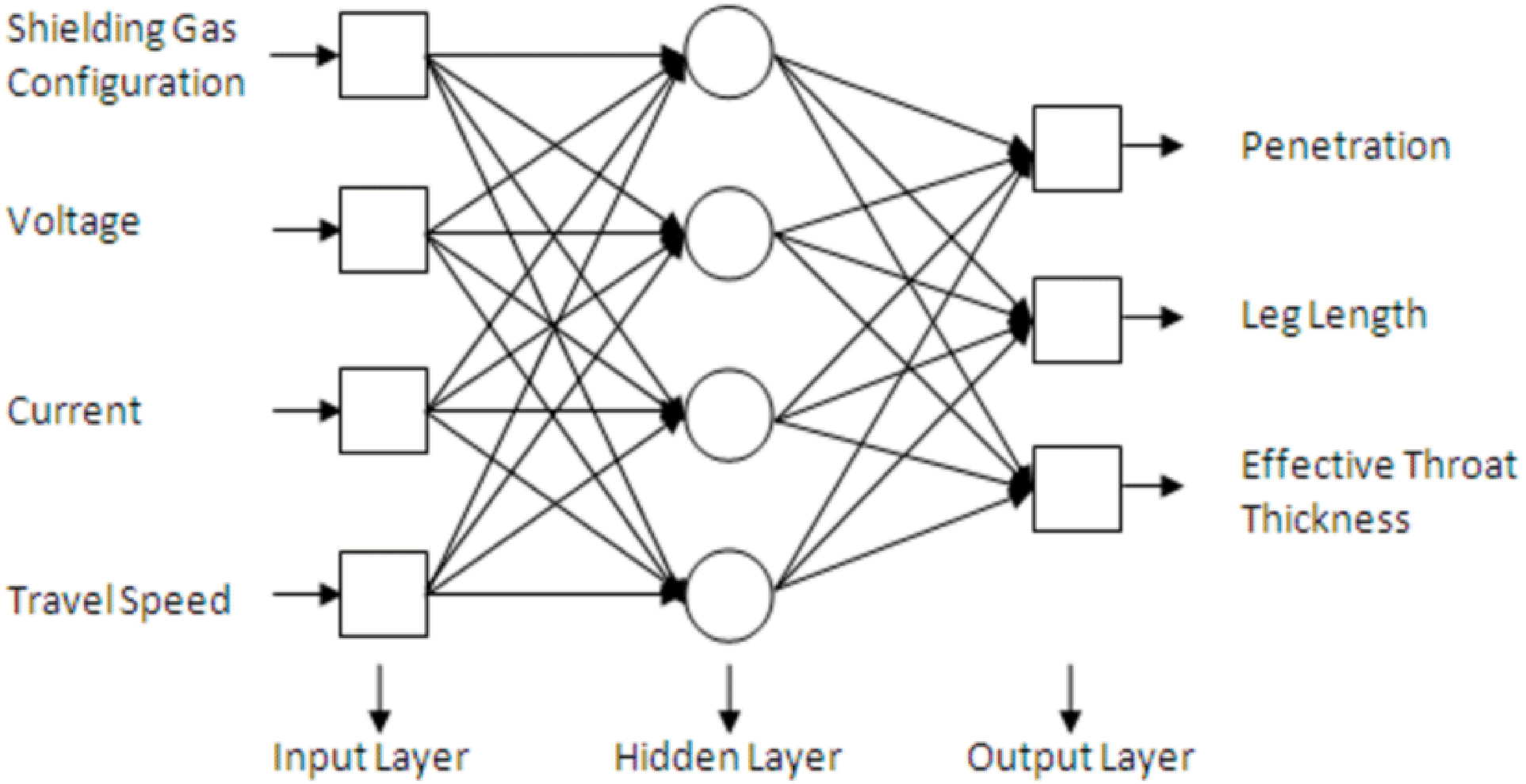
ARCHITECTURE

MULTILAYER PERCEPTRON WITH ONE HIDDEN LAYERS



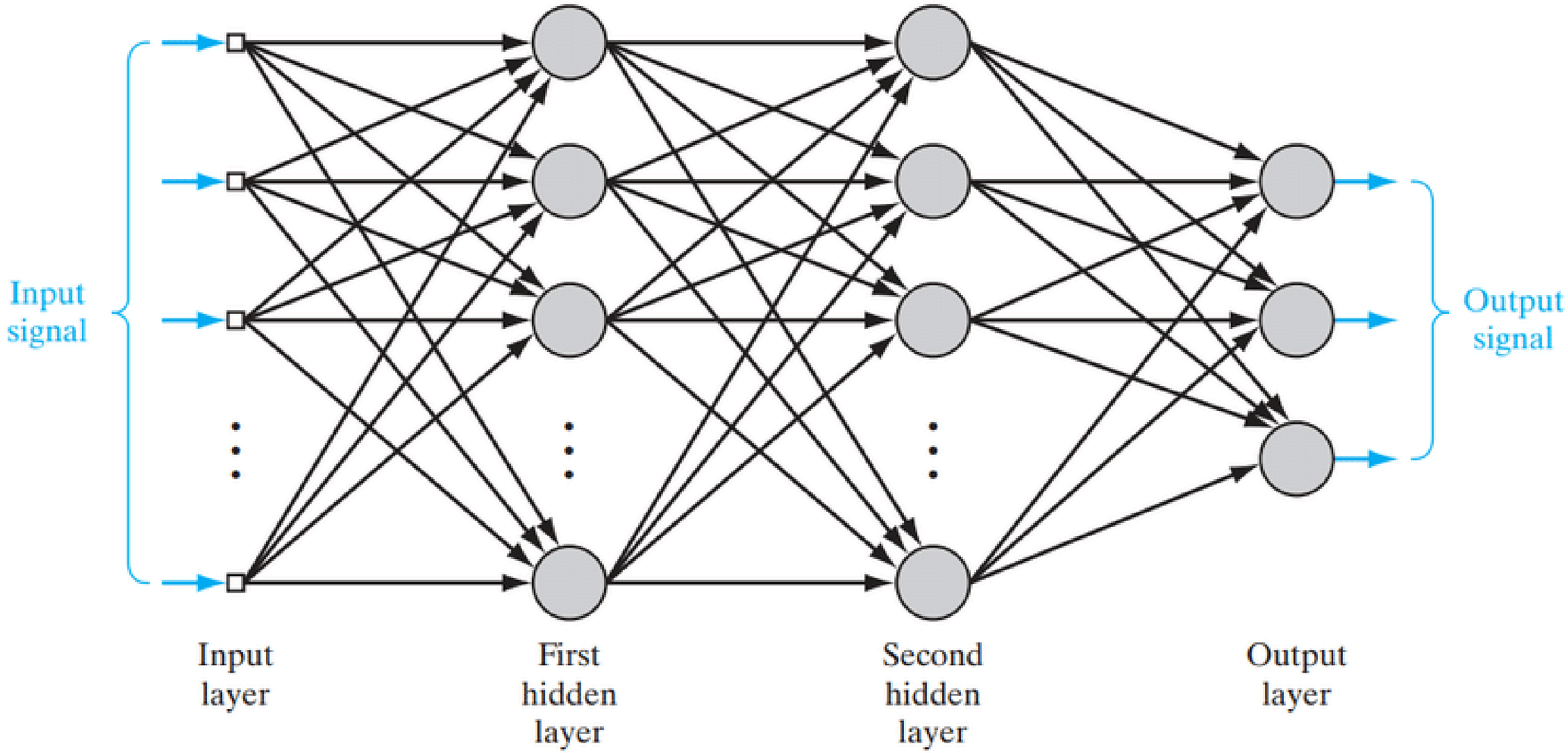
EXAMPLE

MULTILAYER PERCEPTRON WITH ONE HIDDEN LAYERS

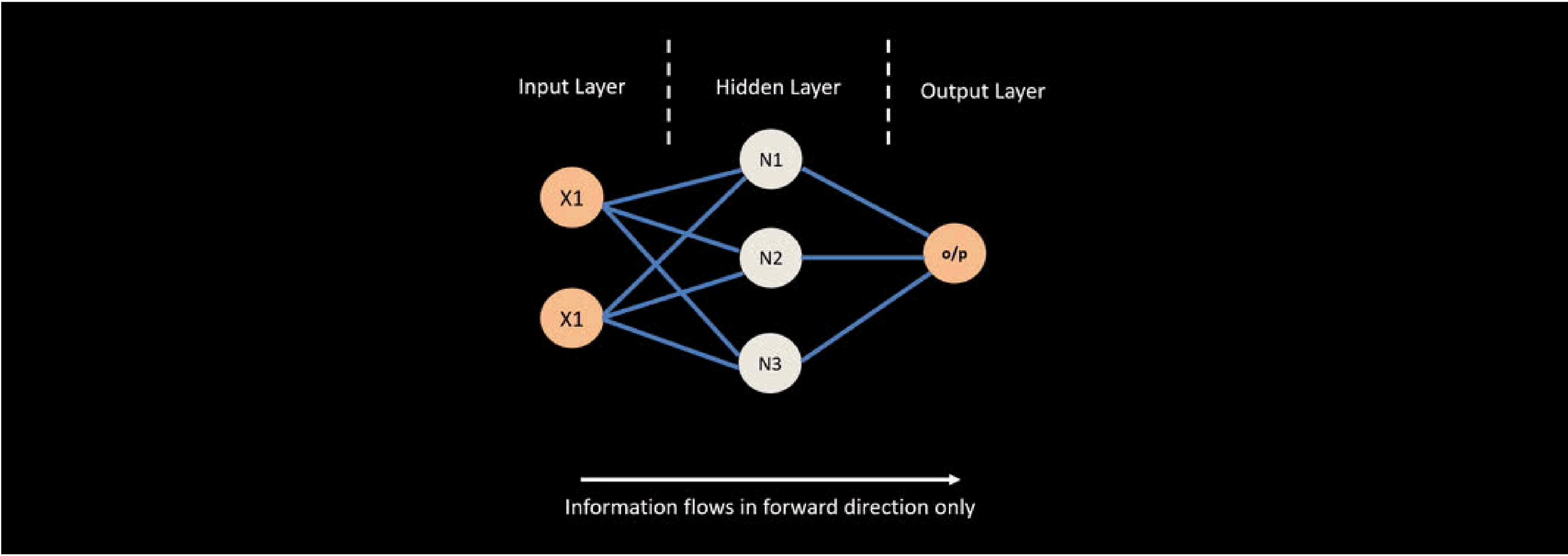


ARCHITECTURE

MULTILAYER PERCEPTRON WITH TWO HIDDEN LAYERS



FEED FORWARD NN

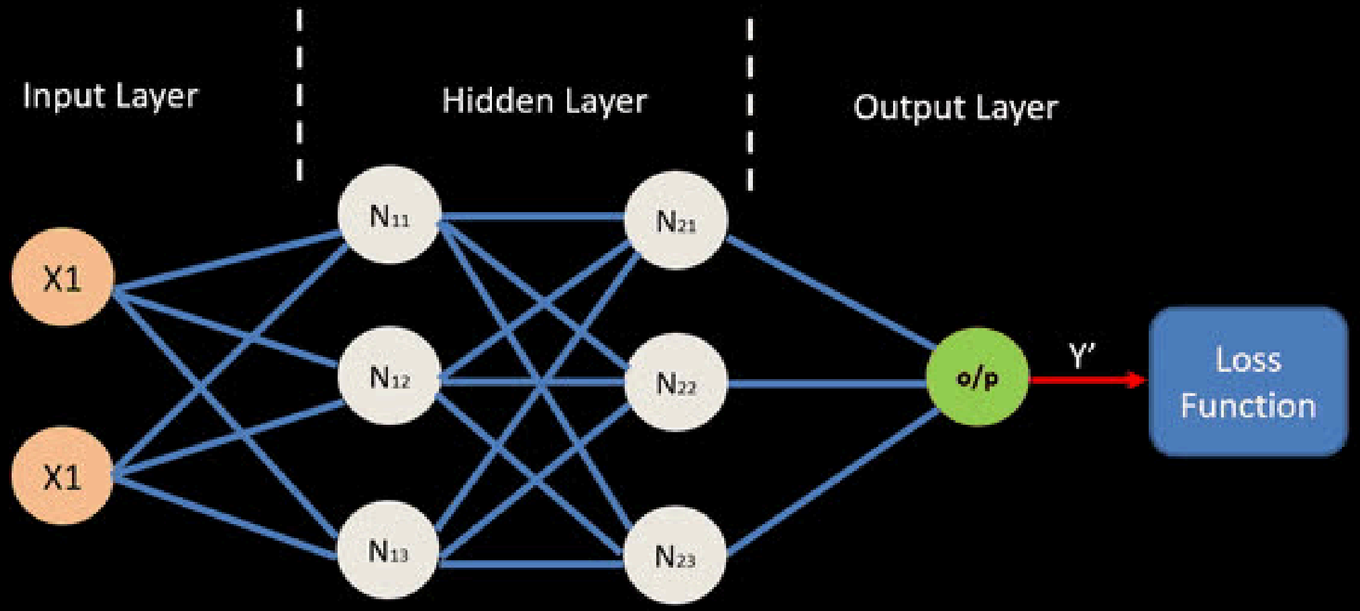


WHAT DOES THE MIDDLE LAYER HIDE?

1. A hidden layer “hides” its desired output.
2. Neurons in the hidden layer cannot be observed through the input/output behavior of the network. There is no obvious way to know what the desired output of the hidden layer should be.
3. Commercial ANNs incorporate three and sometimes four layers, including one or two hidden layers.
4. Each layer can contain from 10 to 1000 neurons. Experimental neural networks may have five or even six layers, including three or four hidden layers, and utilize millions of neurons.

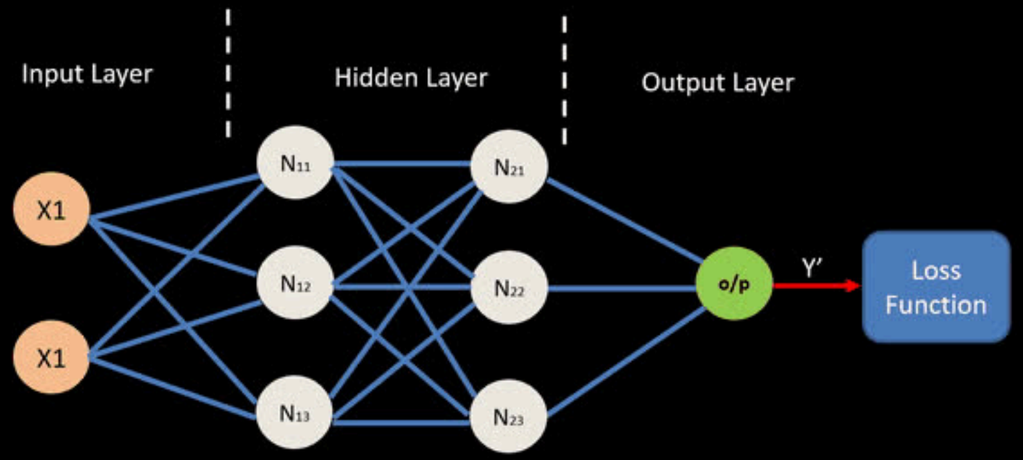
BACK-PROPAGATION NEURAL NETWORK

1. Learning in a multilayer network proceeds the same way as for a perceptron.
2. A training set of input patterns is presented to the network.
3. The network computes its output pattern, and if there is an error - or in other words a difference between actual and desired output patterns - the weights are adjusted to reduce this error.



BACK-PROPAGATION NEURAL NETWORK

1. In a back-propagation neural network, the learning algorithm has two phases.
2. First, a training input pattern is presented to the network input layer. The network propagates the input pattern from layer to layer until the output pattern is generated by the output layer.
3. If this pattern is different from the desired output, an error is calculated and then propagated backwards through the network from the output layer to the input layer. The weights are modified as the error is propagated.



THE END

NEXT LECTURE

Image Processing and Computer Vision