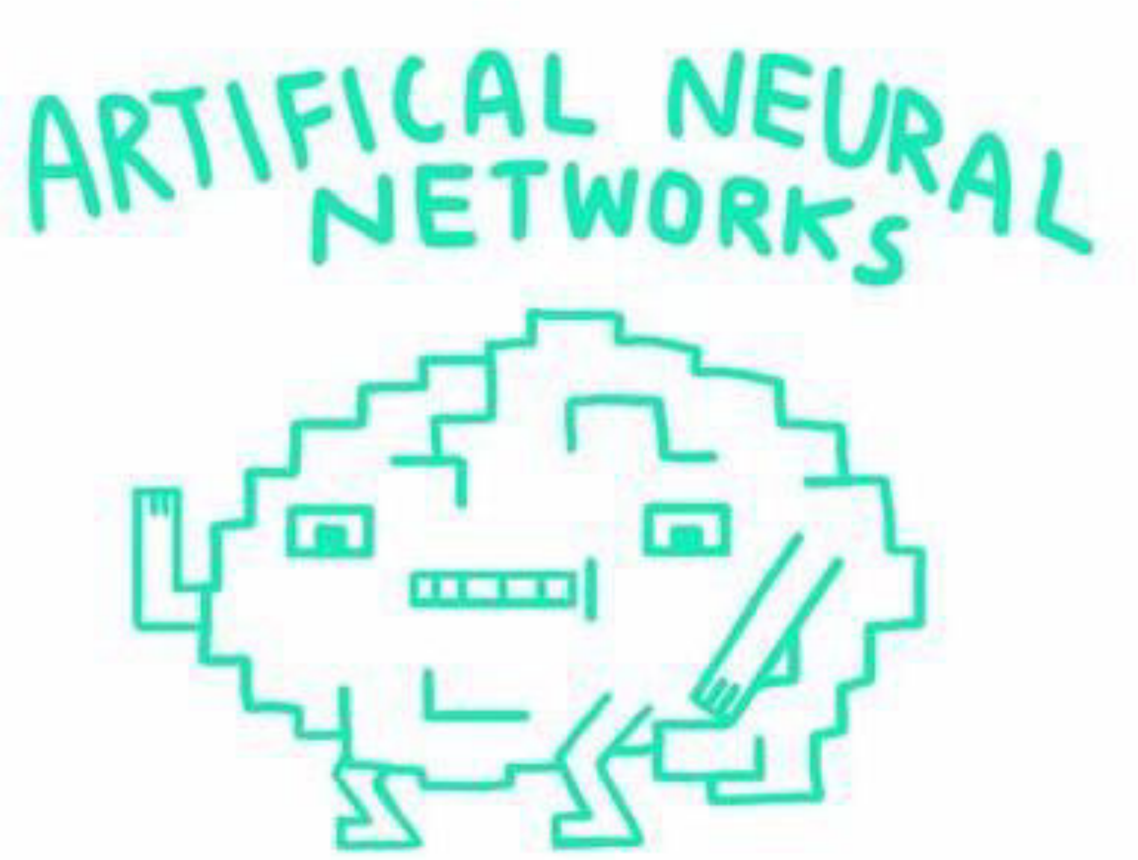
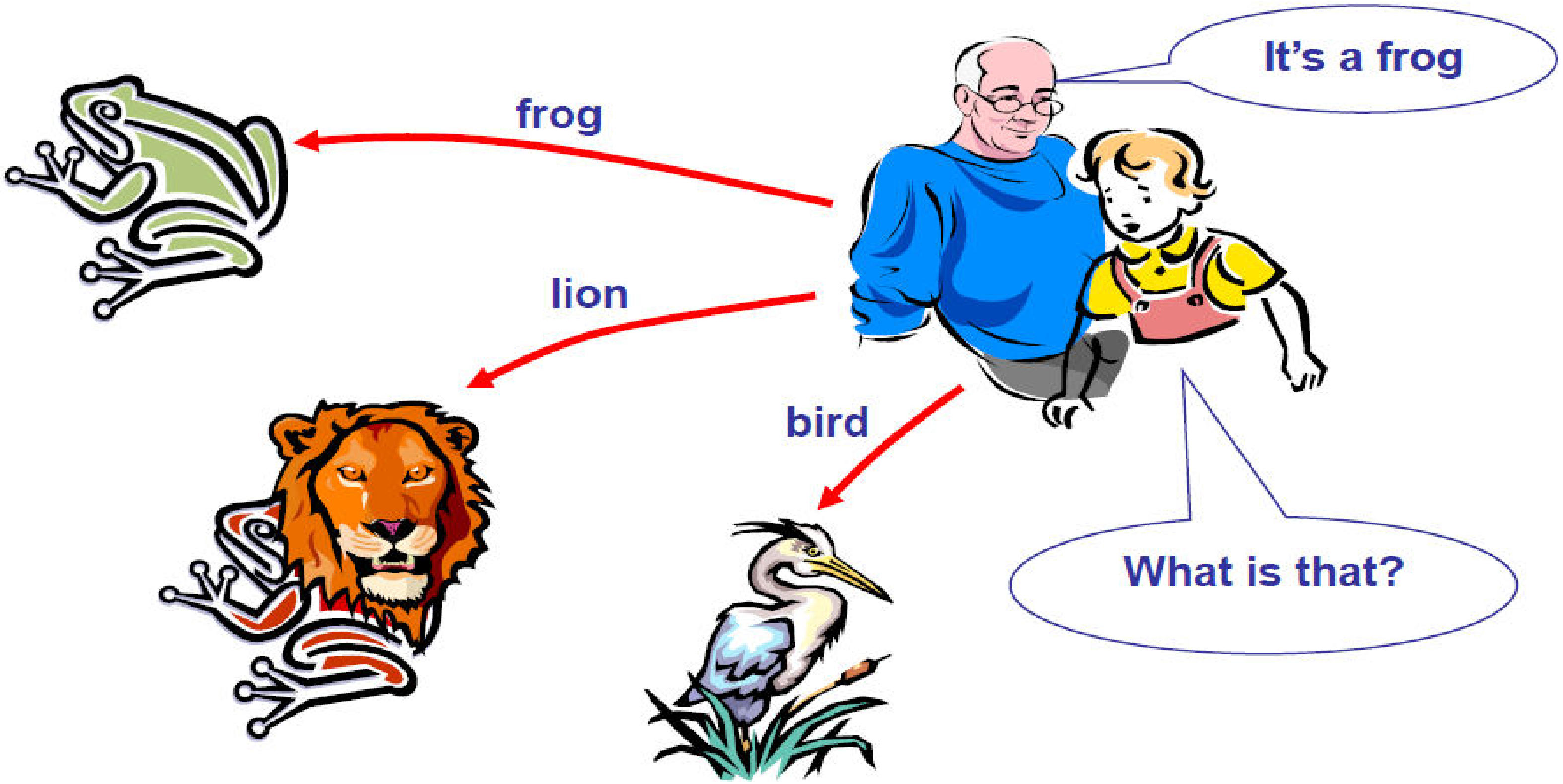
Introduction to Artificial Neural Networks(ANN)

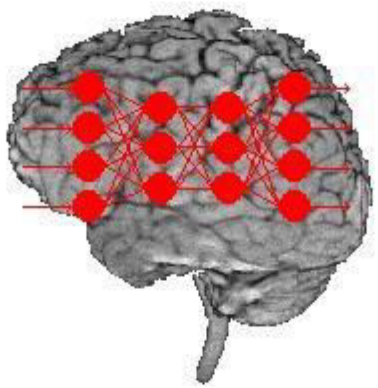


The idea of ANNs..?

- NNs learn the relationship between cause and effect or organize large volumes of data into orderly and informative patterns.



Neural Networks to the Rescue…

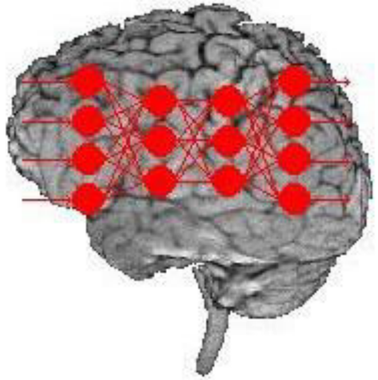


Neural network: *information processing paradigm inspired by biological nervous systems, such as our brain.*

• Structure: a large number of highly interconnected processing elements (*neurons*) working together.

• Like people, they learn *from* experience (by example)

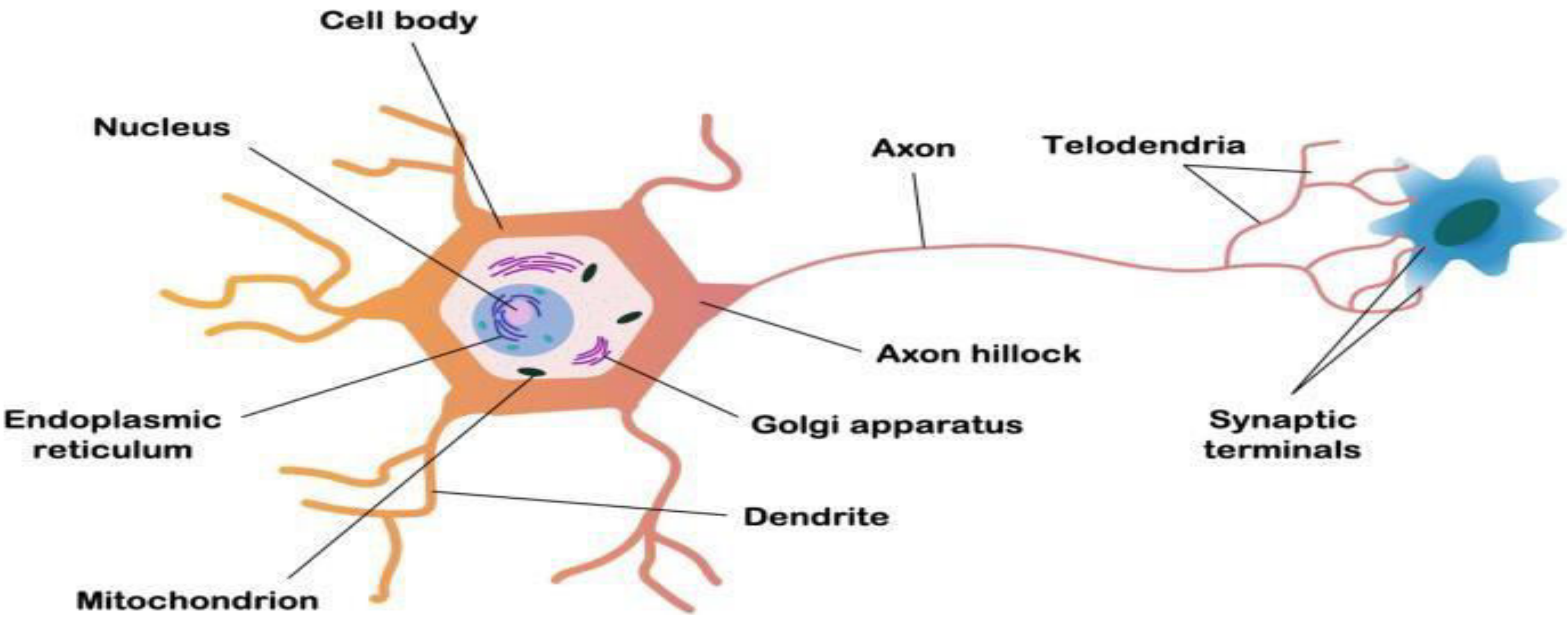
Definition of ANN



- Data processing system consisting of a large number of simple, highly interconnected processing elements (artificial neurons) in an architecture inspired by the structure of the cerebral cortex of the brain.

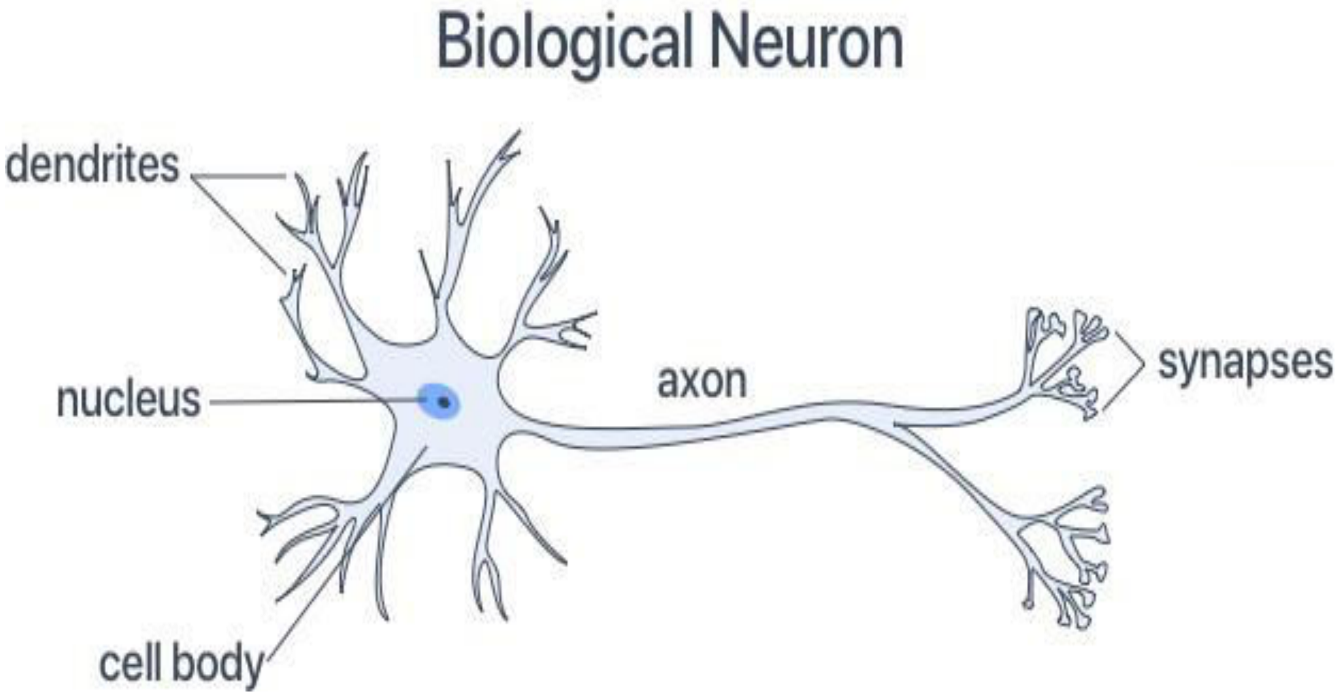
L. H. Tsoukalas and R. E. Uhrig,1997.

Inspiration from Neurobiology



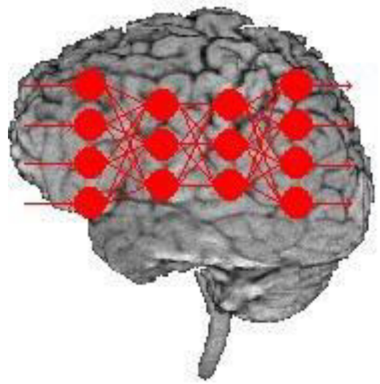
Human Biological Neuron

Human Biological Neuron



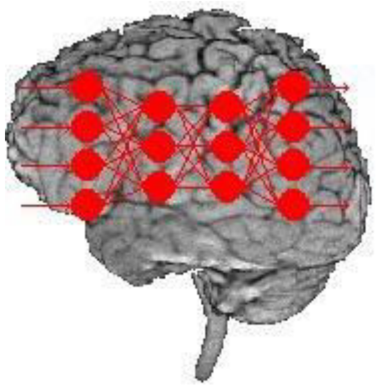
- A biological neuron has three types of main components; dendrites, soma(or cell body), and axon.
- Dendrites receive signals from other neurons.
- The soma, sums the incoming signals. When sufficient input is received, the cell fires; that is it transmits a signal over its axon to other cells.

Facts of Human Neurobiology



- Number of neurons ~ 10^{11}
- Connection per neuron ~ 10^{4-5}
- Neuron switching time ~ 0.001 seconds or 10^{-3}
- Scene recognition time ~ 0.1 second
- 100 inference steps don't seem like enough
- Highly parallel computation based on distributed representation

Properties of Neural Networks

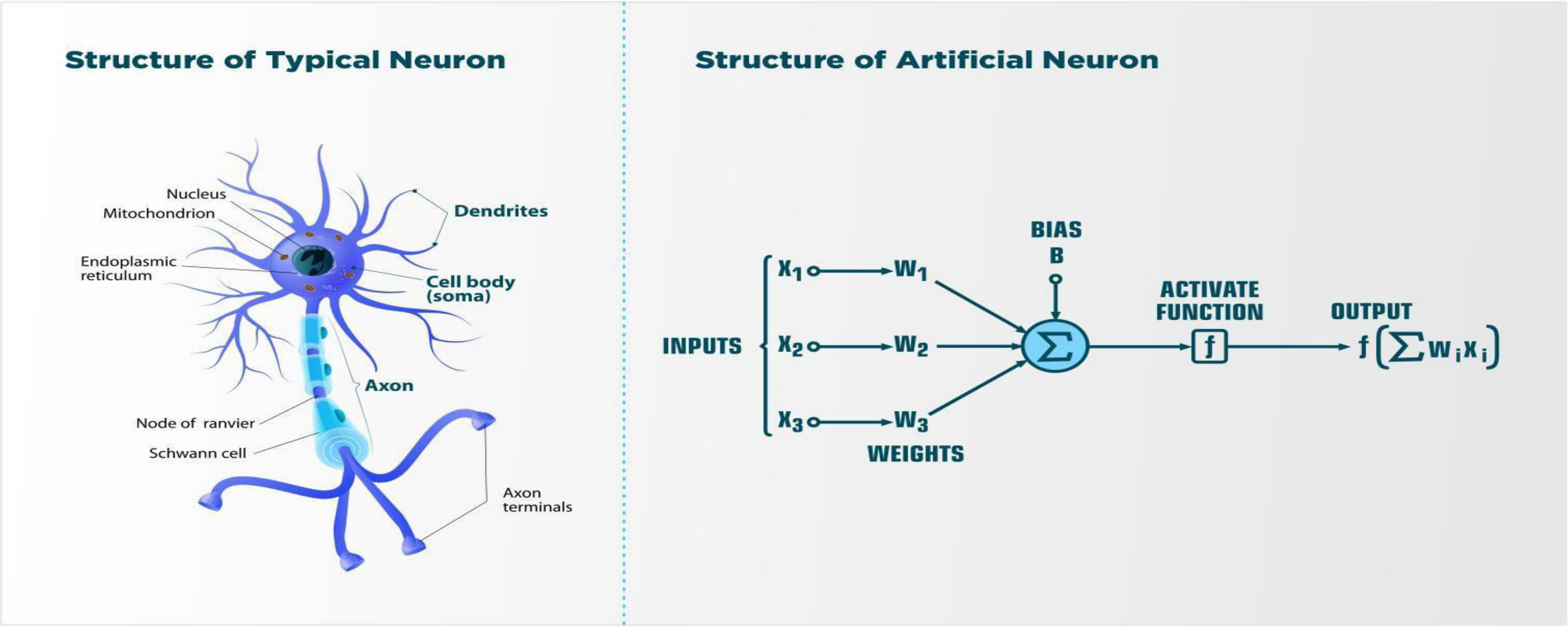


- Many neuron-like threshold-switching units
- Many weighted interconnections among units
- Highly parallel, distributed process
- Emphasis on tuning weights automatically
- Input is a high-dimensional discrete or real-valued (e.g, sensor input)

Artificial Neurons

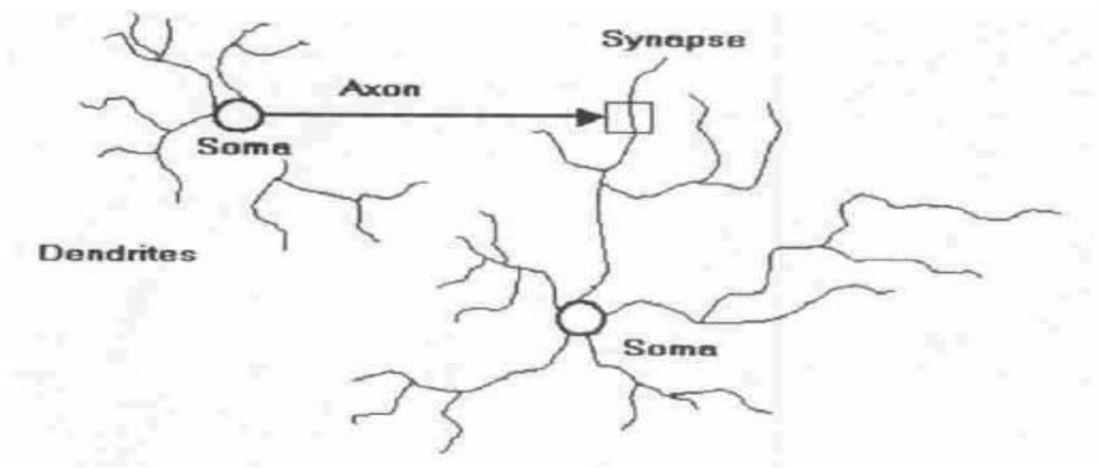
- Neurons ANN is an information processing system that has certain performance characteristics in common with biological nets.
- Several key features of the processing elements of ANN are suggested by the properties of biological neurons:
  - The processing element receives many signals.
  - 2. Signals may be modified by a weight at the receiving synapse.
  - 3. The processing element sums the weighted inputs.
  - 4. Under appropriate circumstances (sufficient input), the neuron transmits a single output.
  - 5. The output from a particular neuron may go to many other neurons.

Artificial Neurons

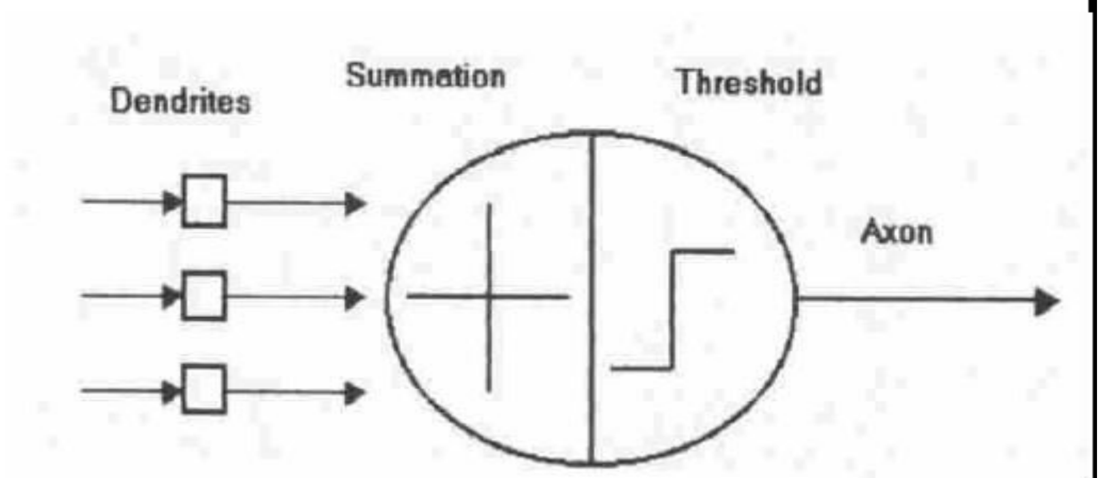


Learning the solution to a problem = changing the connection **weights**

Artificial Neurons



Four basic components of a human biological neuron



The components of a basic artificial neuron

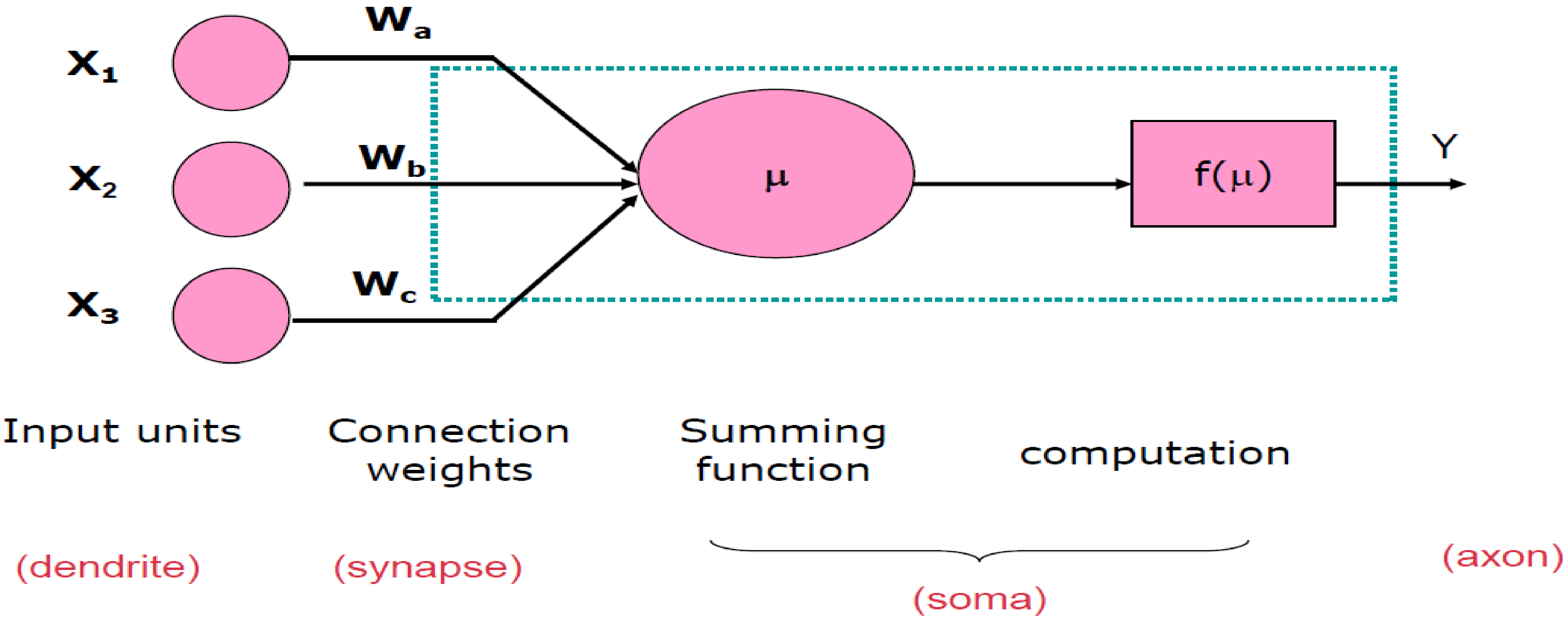
1. Information processing occurs at many simple elements called neurons.

2. Signals are passed between neurons over connection links.

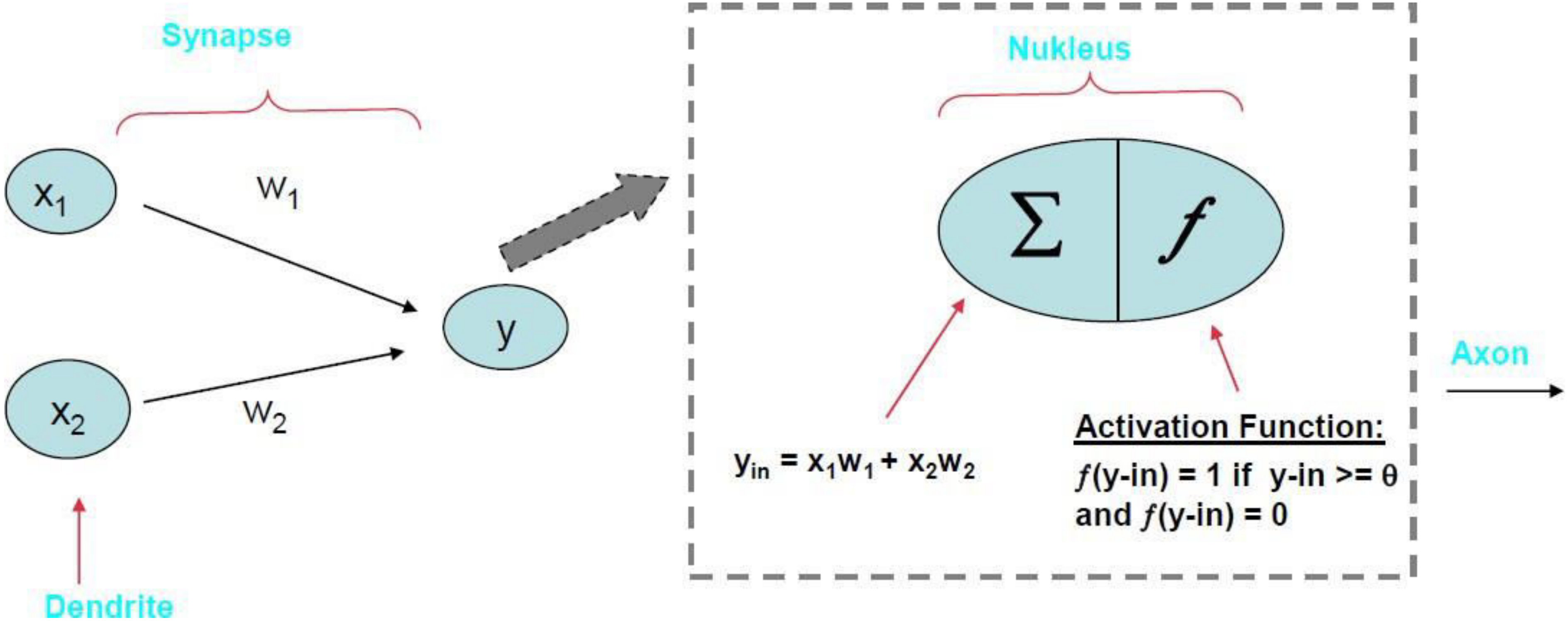
3. Each connection link has an associated weight, which, in a typical neural net, multiplies the signal transmitted.

4. Each neuron applies an activation function to its net input to determine its output signal.

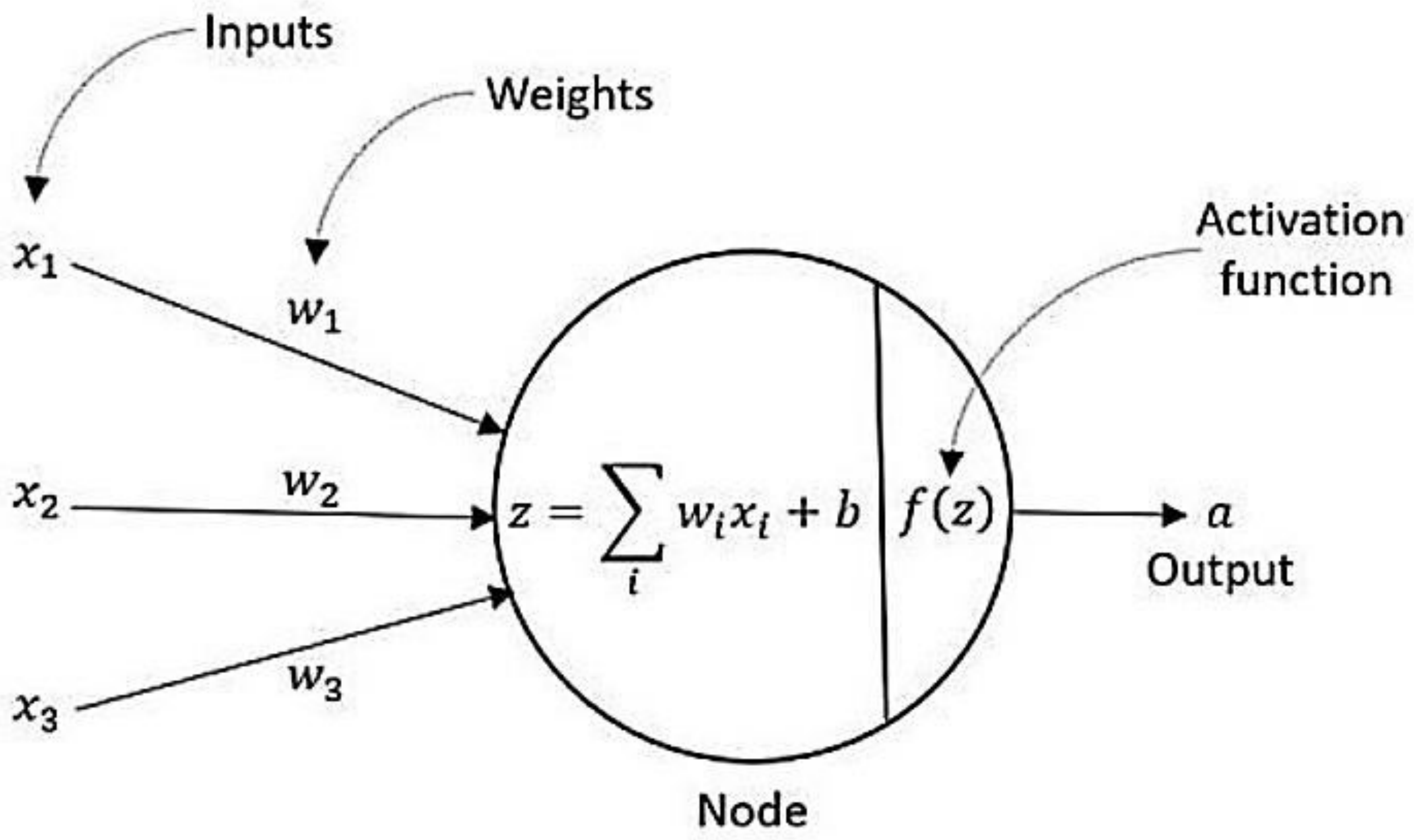
Model of A Neuron

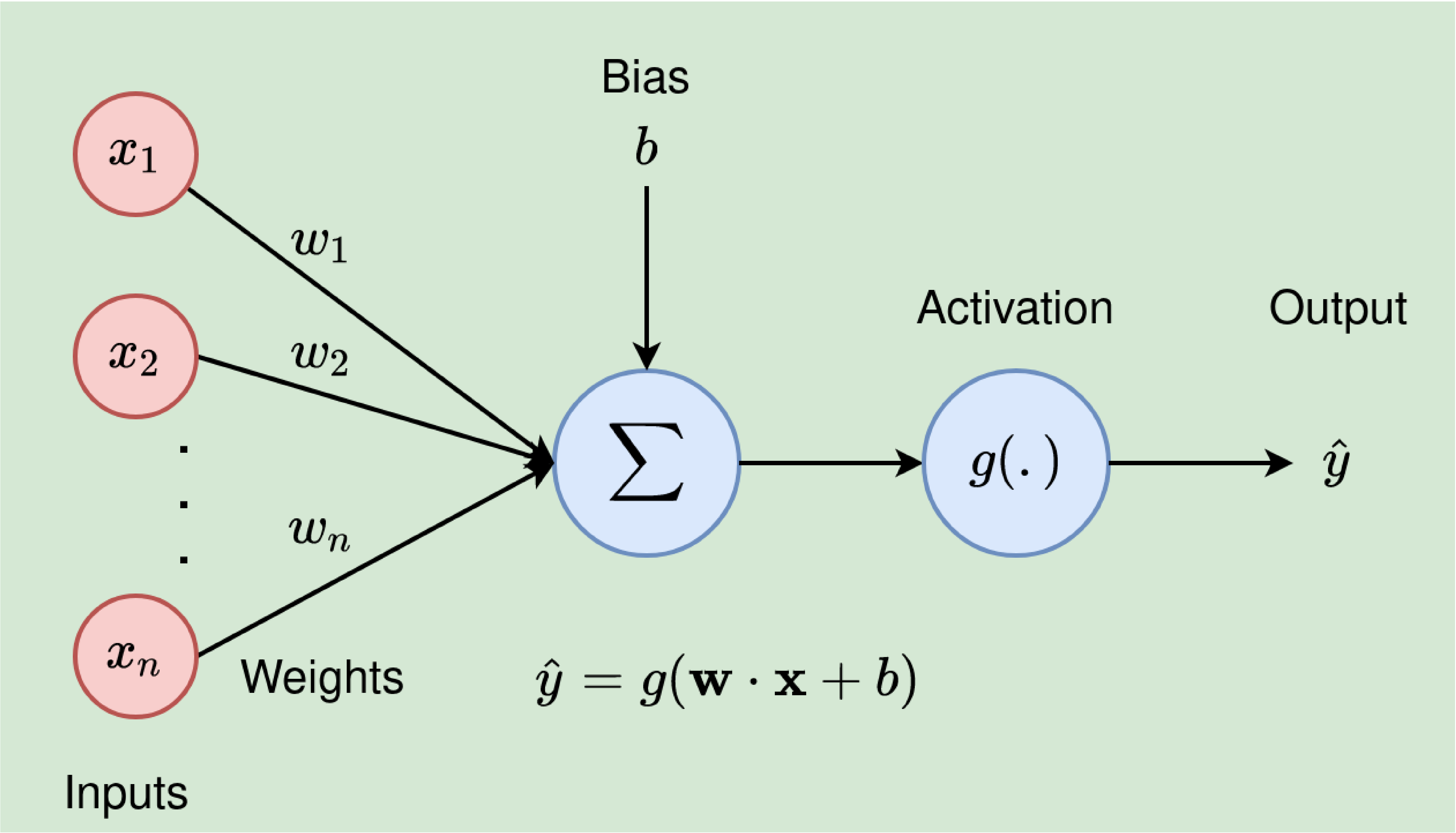


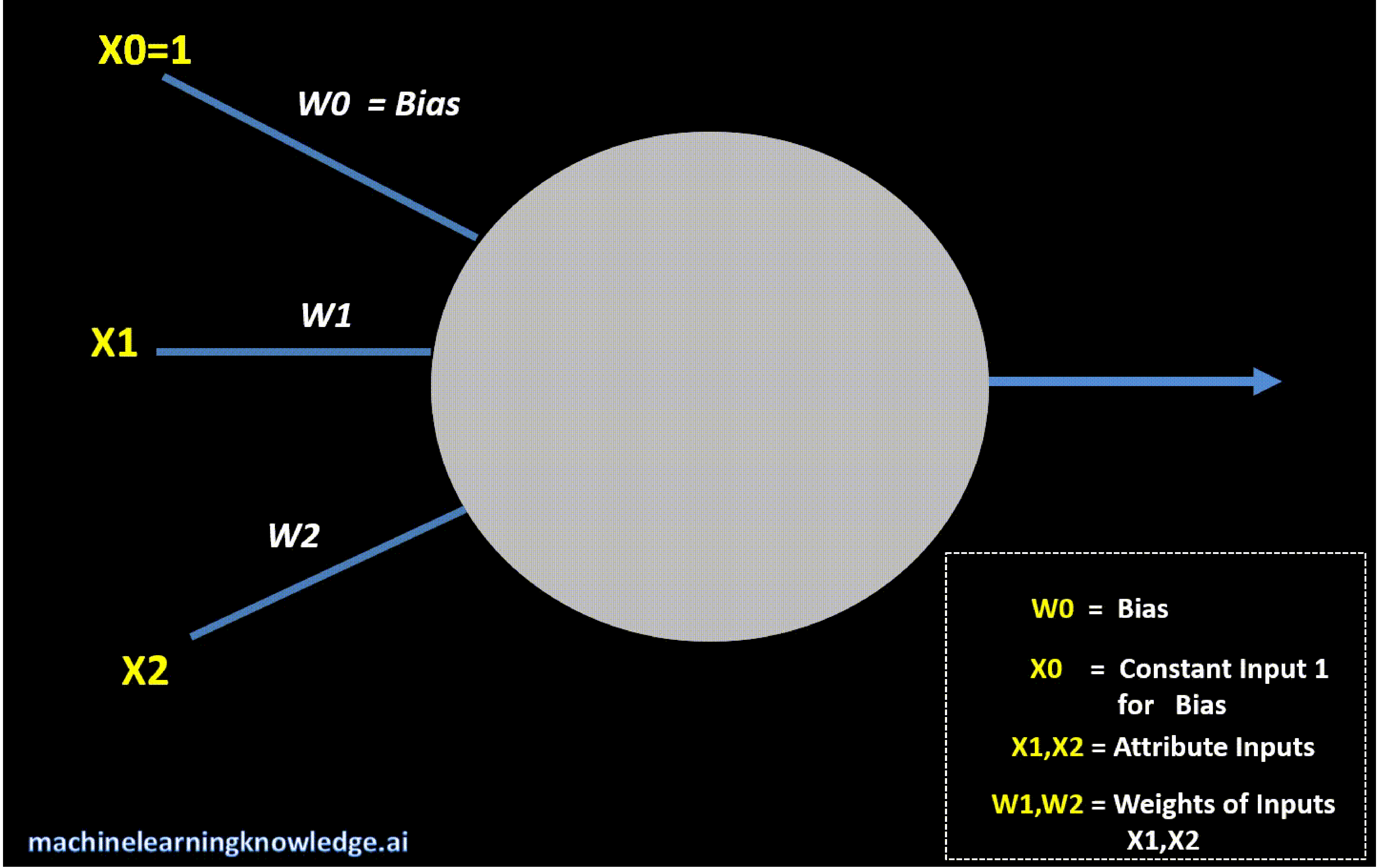
Artificial Neural Network



- A neuron receives input, determines the strength or the weight of the input, calculates the total weighted input, and compares the total weight with a value (threshold)
- The value is in the range of 0 and 1
- If the total weighted input is greater than or equal to the threshold value, the neuron will produce the output, and if the total weighted input is less than the threshold value, no output will be produced.


Inputs
Weights
x_1
w_1
x_2
w_2
x_3
w_3
$z = \sum_i w_i x_i + b$
$f(z)$
Activation function
a
Output
Node


Bias
b
x_1
x_2
x_n
w_1
w_2
w_n
Σ
Activation
$g(.)$
Output
$\hat{y}$
Weights
$\hat{y} = g(\mathbf{w} \cdot \mathbf{x} + b)$
Inputs


X0=1
W0 = Bias
X1
W1
W2
X2
W0 = Bias
X0 = Constant Input 1 for Bias
X1,X2 = Attribute Inputs
W1,W2 = Weights of Inputs X1,X2
machinelearningknowledge.ai

Characterization

Architecture

–a pattern of connections between neurons

- Single Layer Feedforward
- Multilayer Feedforward
- Recurrent

- Strategy / Learning Algorithm

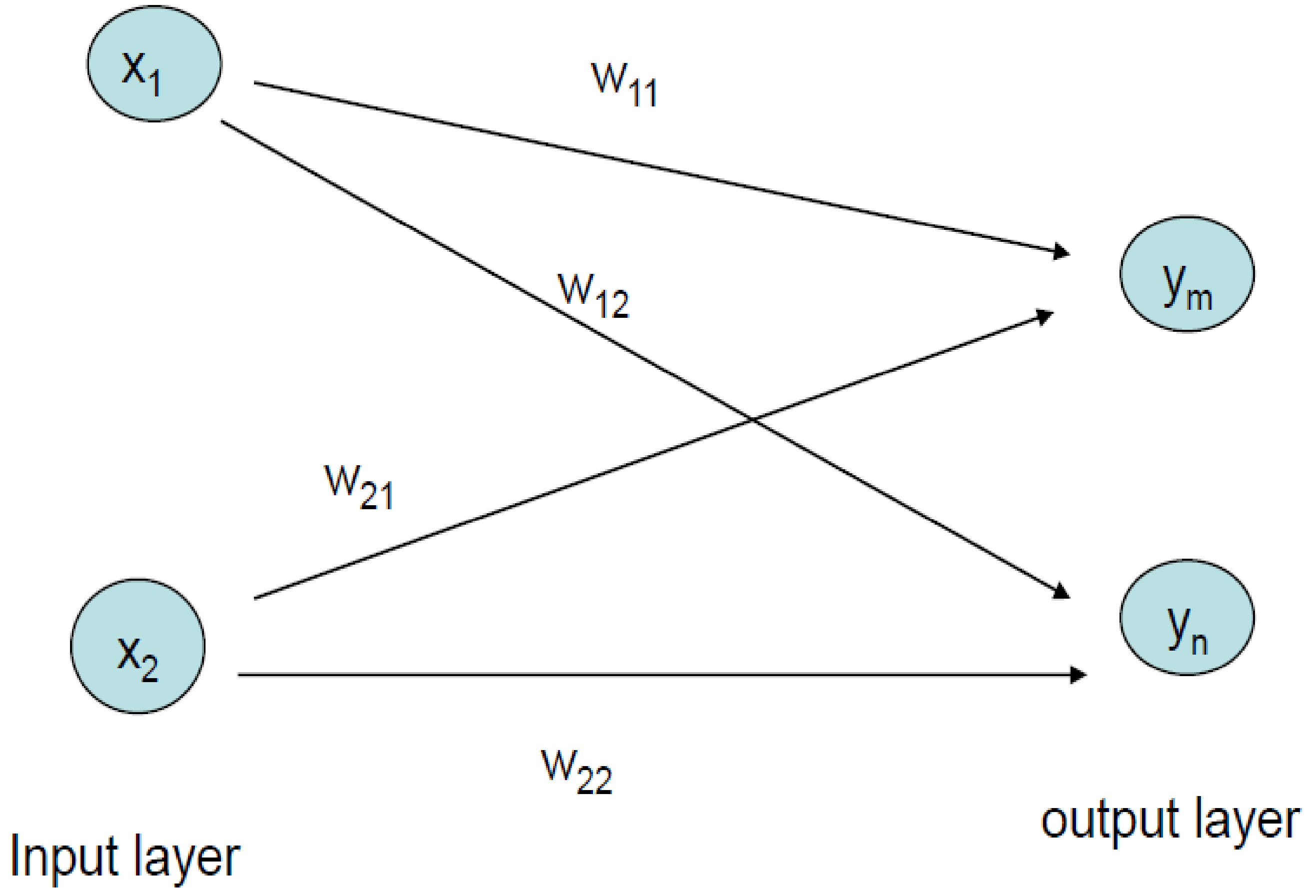
–a method of determining the connection weights

- Supervised
- Unsupervised
- Reinforcement

- Activation Function

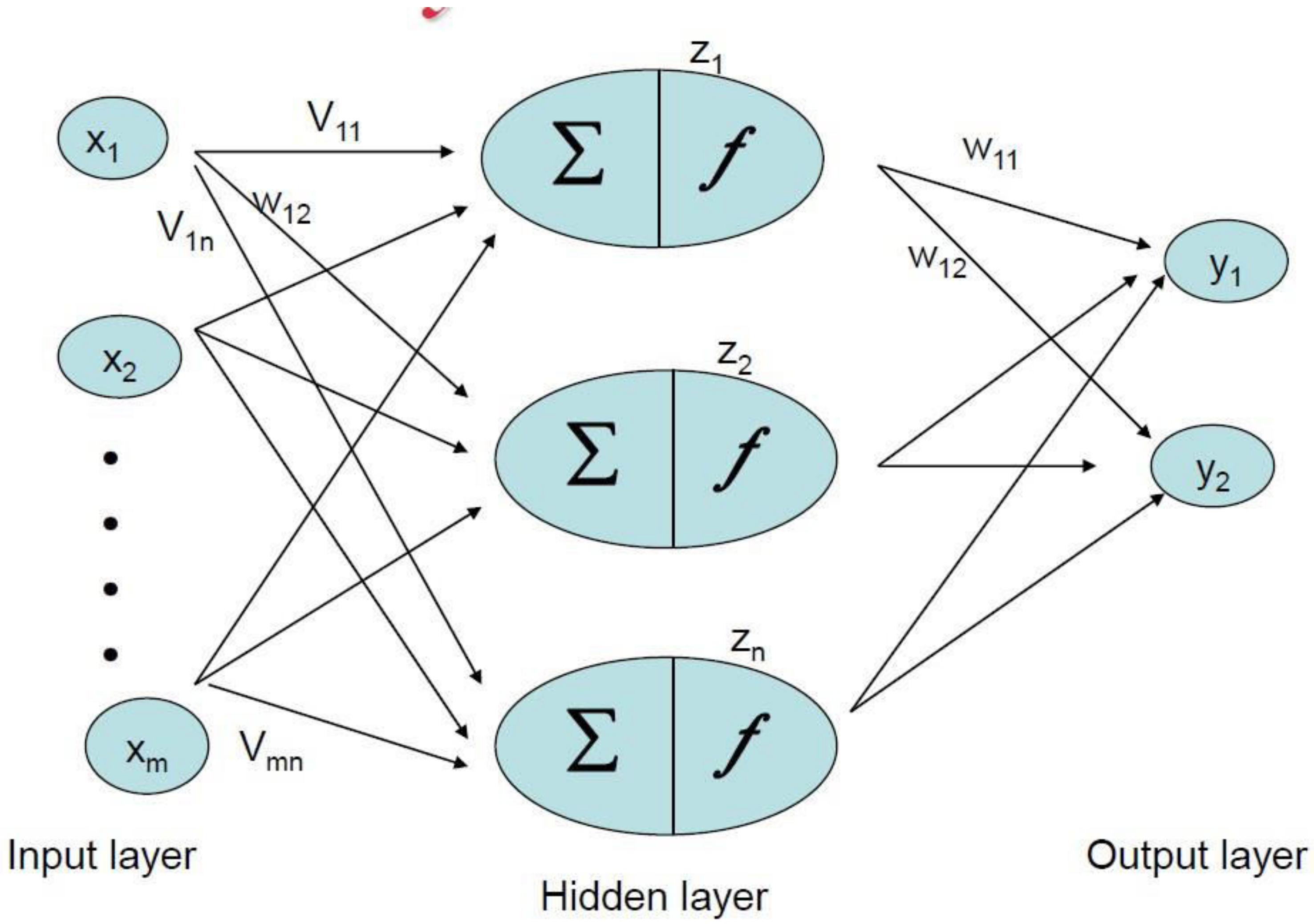
–Function to compute output signal from the input signal

Single Layer Feedforward NN

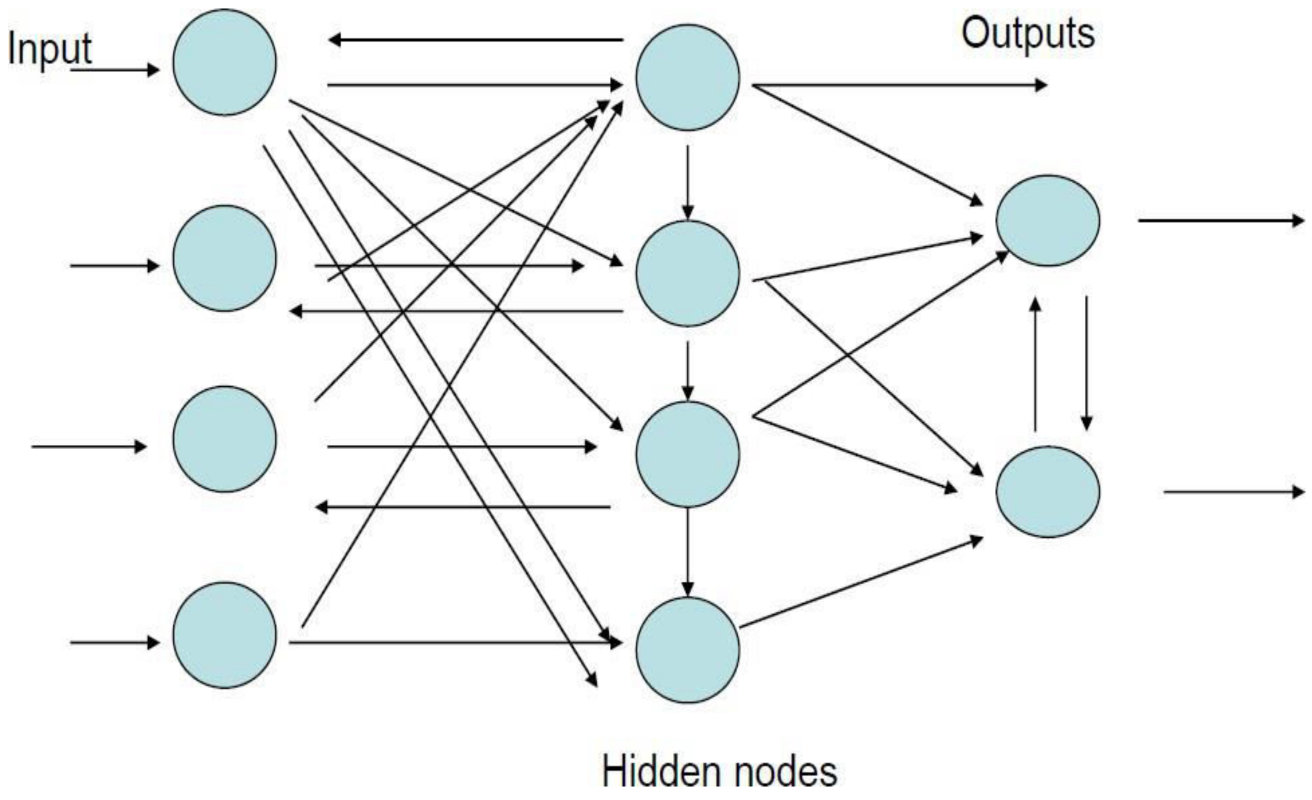


- Self-organization feature map (SOFM)
- Perceptron
- Learning Vector Quantization (or LVQ)
- Hopfield

Multilayer Neural Network



Recurrent N N



- Adaptive Resonance Theory (ART)
- Bidirectional Associative Memory (BAM)
- Brain-State-in-a-Box (BSB)
- Boltzman Machine
- Cauchy Machine

Strategy /Learning Algorithm

Supervised Learning

- Learning is performed by presenting a pattern with a target
- During learning, the produced output is compared with the desired output.

–The difference between both outputs is used to modify learning weights according to the learning algorithm

- Examples : Recognizing hand-written digits, pattern recognition and etc.
- Neural Network models: perceptron, feed-forward, radial basis function, support vector machine.

Strategy /Learning Algorithm

Unsupervised Learning

- Targets are not provided
- Appropriate for the clustering task

-Find similar groups of documents on the web, content addressable memory, and clustering.

- Neural Network models: Kohonen, self-organizing maps, Hopfield networks.

Strategy /Learning Algorithm

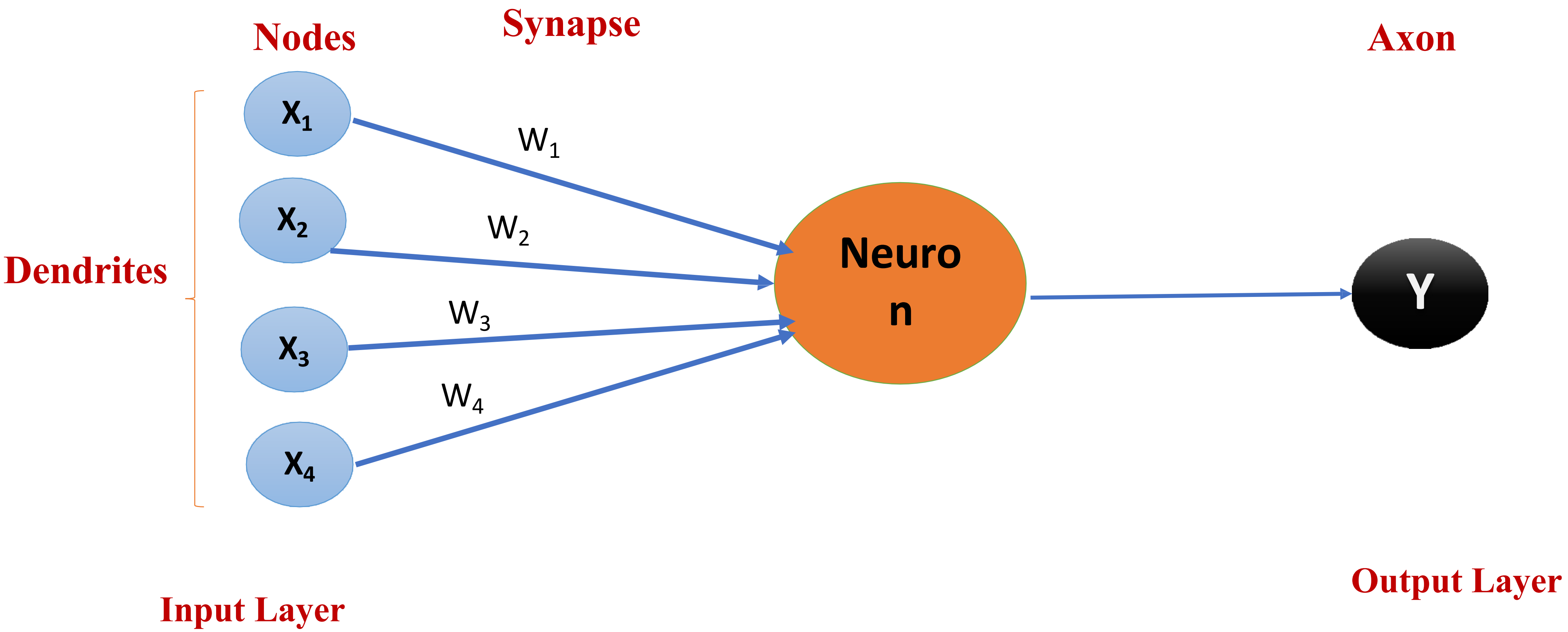
Reinforcement Learning

- Target is provided, but the desired output is absent.
- The net is only provided with guidance to determine whether the produced output is correct or vice versa.
- Weights are modified in the units that have errors.

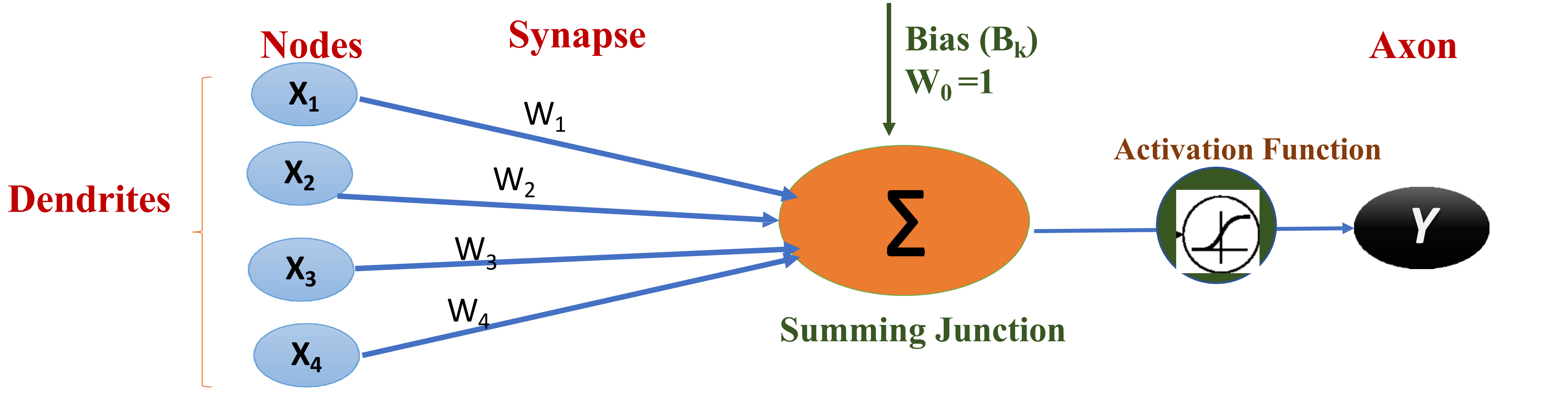
The Perceptron

- Perceptron was introduced by Frank Rosenblatt in 1957
- The basic building block (or) unit of the neural network.
- The perceptron is a network (neural network) at takes a number of inputs carry out some processing on those inputs and produces an output.
- It is also called an Artificial Neuron. It consists of a single neuron with a number of adjustable weights.
- Initially the perceptron was designed to take a number of binary inputs, and produce one binary output.

The Perceptron



It is based on a slightly different artificial neuron called a linear threshold unit (LTU).



Input Layer

Output Layer

X- Input **Y- Output**

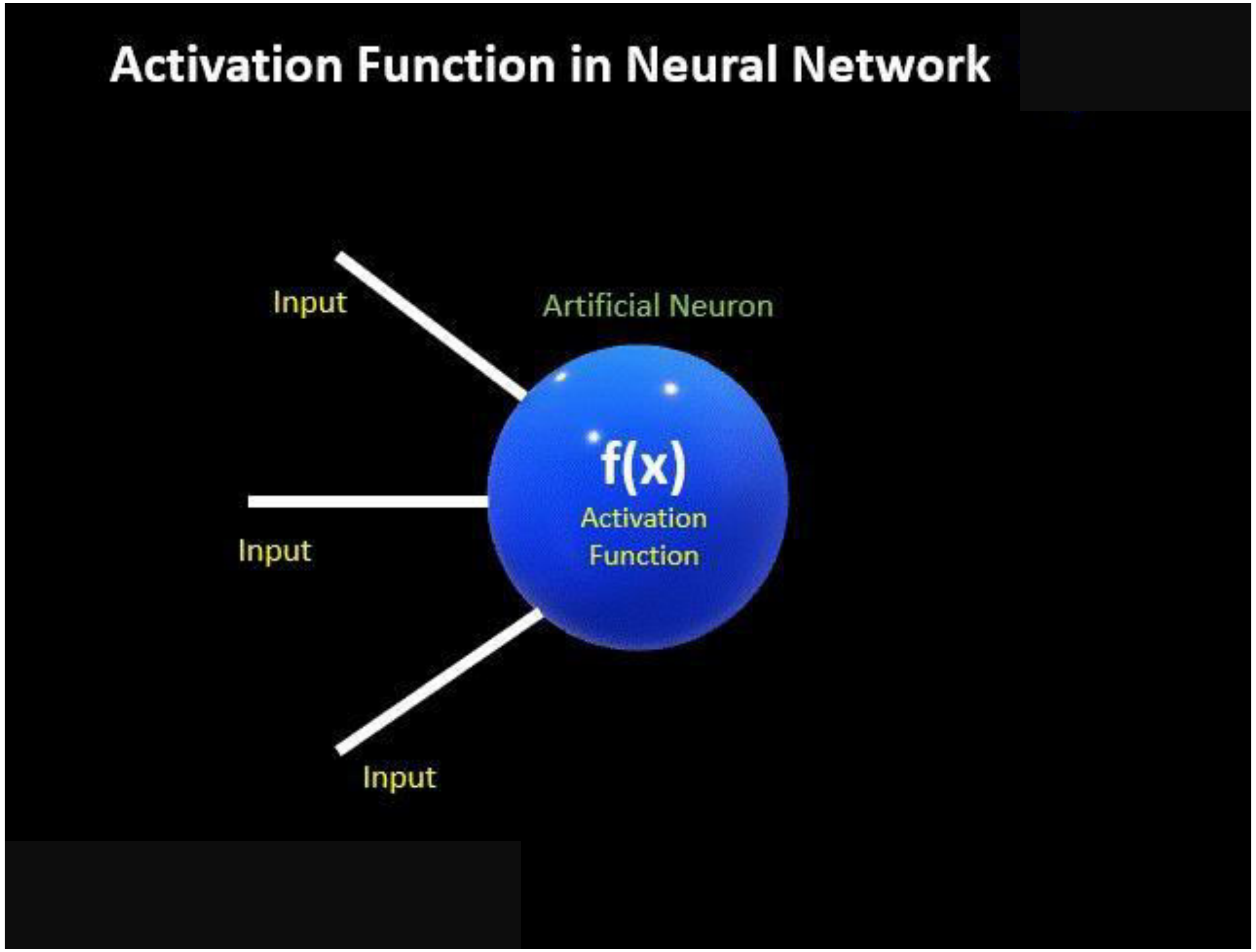
$$Y = X_1 + X_2 + X_3 + X_4$$

$$Y = W_1X_1 + W_2X_2 + W_3X_3 + W_4X_4 + \mathbf{B_k}$$

In mathematical terms, a Neuron *k* can be described as: $U_k = \sigma_{i=0}^{n} W_i X_i$

The output : $y_k = \varphi(u_k + b_k)$

Activation Functions



Activation functions in Neural Networks

- Activation function decides, whether a neuron should be activated or not by calculating the weighted sum and further adding bias with it. The purpose of the activation function is to introduce non-linearity into the output of a neuron.

Can we do without an activation function?

- When we do not have the activation function the weights and bias would simply do a linear transformation.
- A linear equation is simple to solve but is limited in its capacity to solve complex problems.
- A neural network without an activation function is essentially just a linear regression model.
- The activation function does the non-linear transformation to the input making it capable to learn and perform more complex tasks

Why do we use the Activation Function?

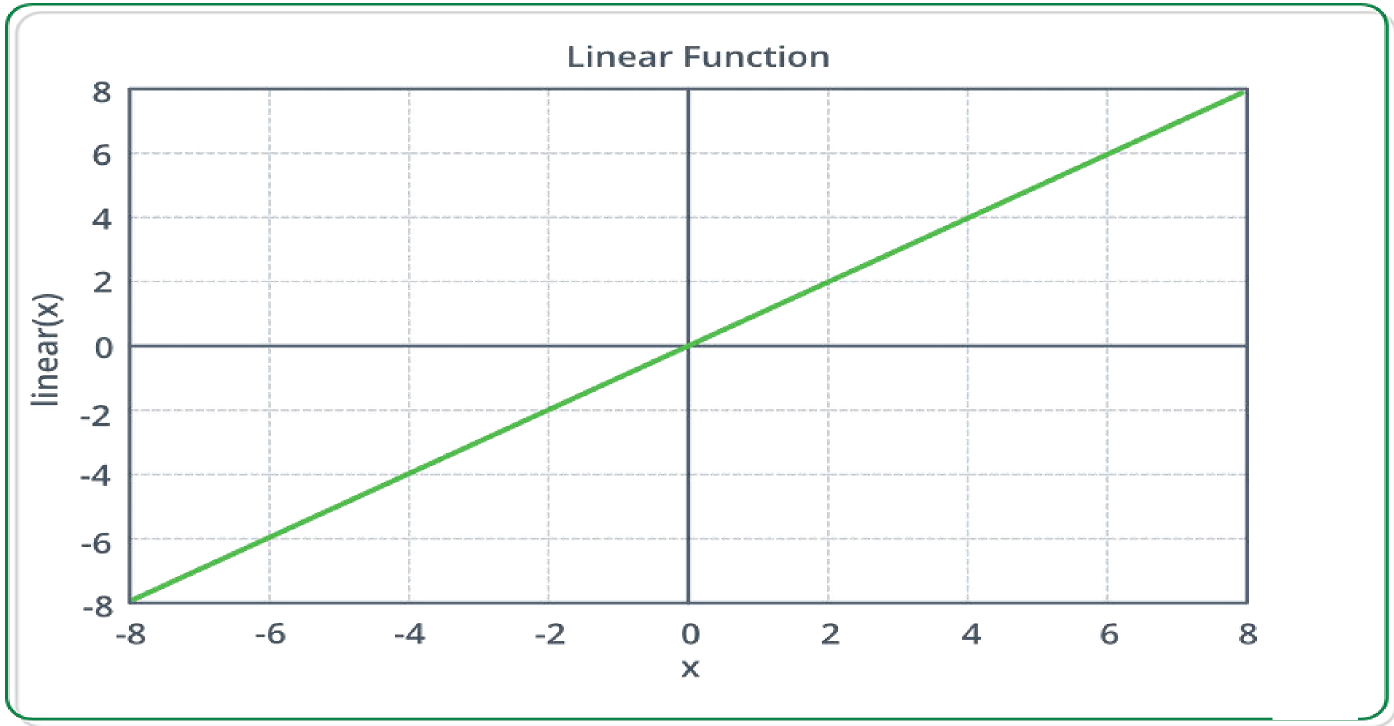
- To normalize data, we employ the activation function .
- To obtain nonlinearity between data in a Feed-Forward Network (FFN), we execute a nonlinearization operation on linear data.

The Activation Functions can be classified into two categories:-

1. Linear Activation Function
2. Non-linear Activation Functions

Linear Activation Function

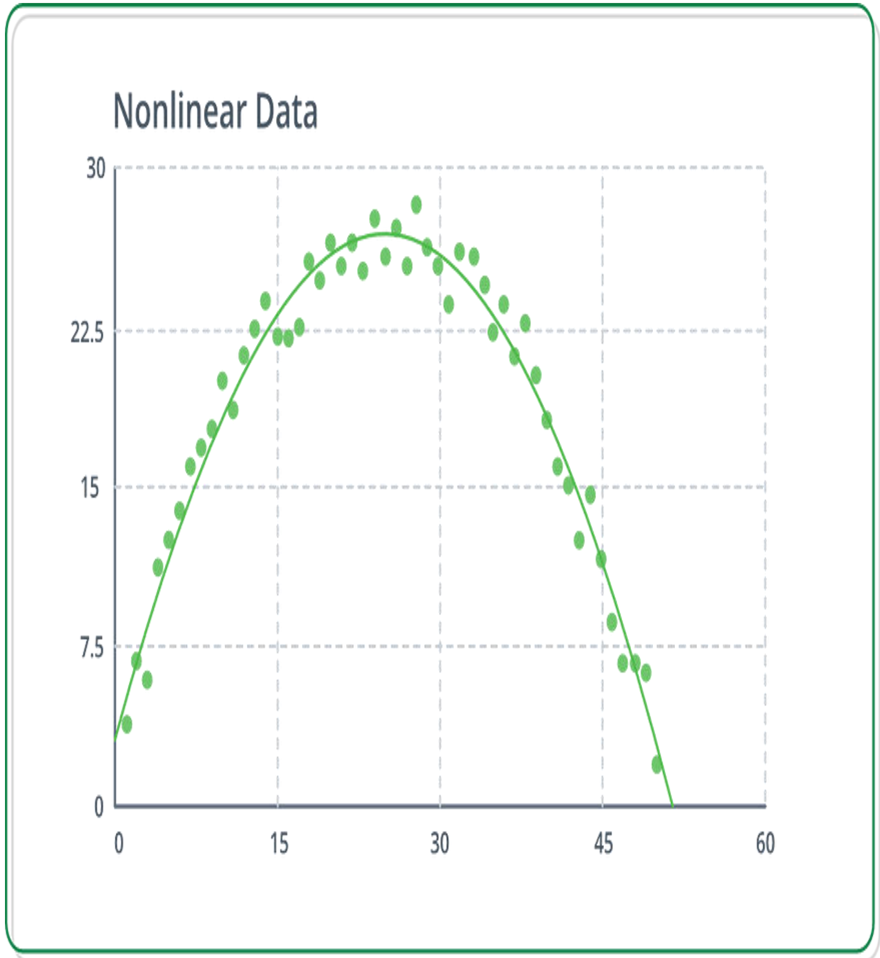
- The function is a line or linear, as you can see. As a result, the functions' output will be unconstrained by any range.
- Equation : $f(x) = x$ and Range : (-infinity to infinity)



- It doesn't help with the complexities of numerous parameters in the data that is normally provided to neural network.

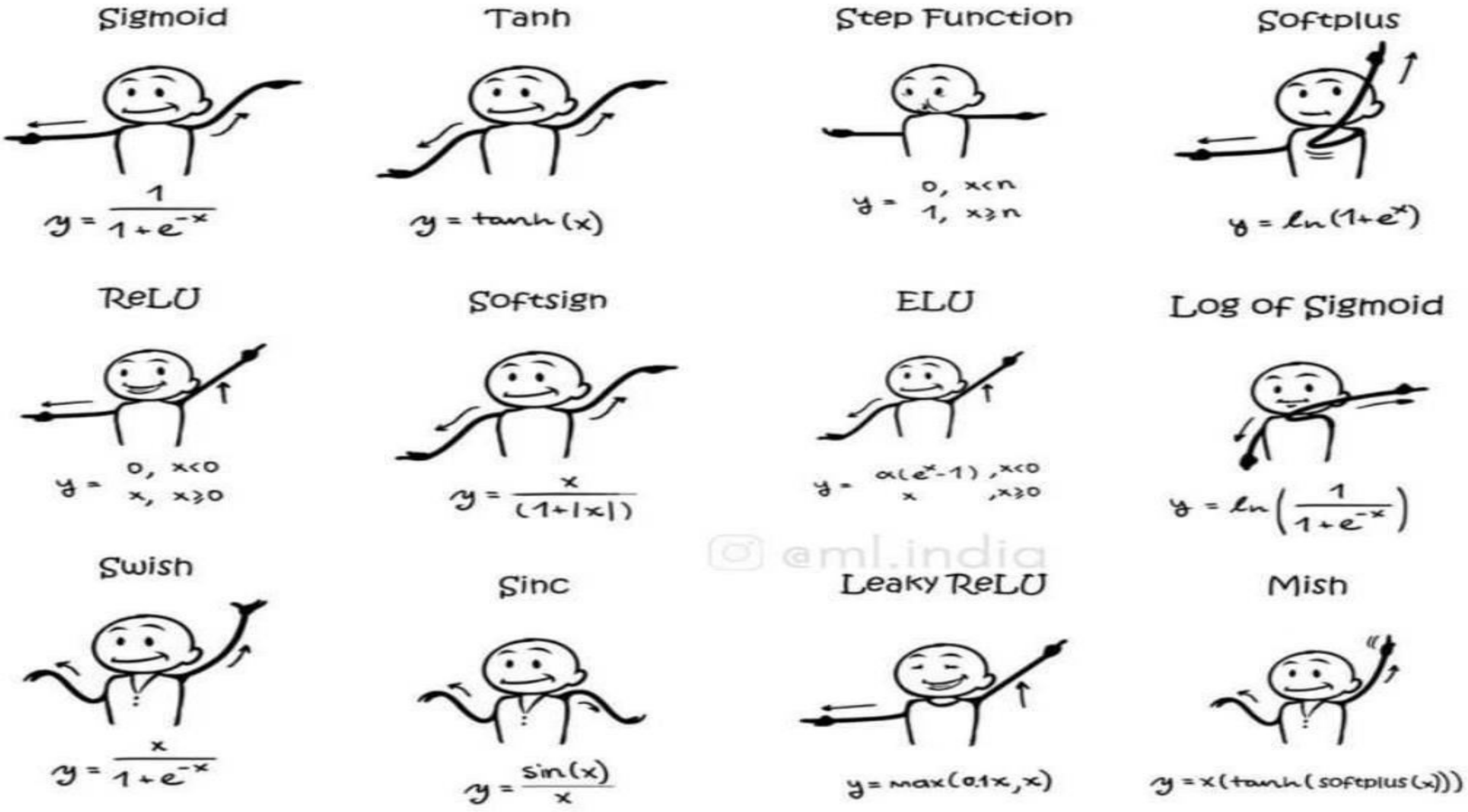
Non-linear Activation Function

- The most commonly utilized activation functions are nonlinear activation functions.

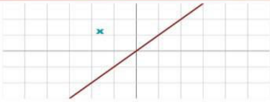
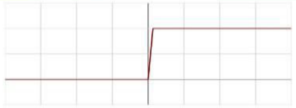
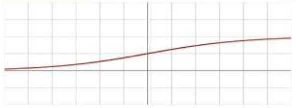
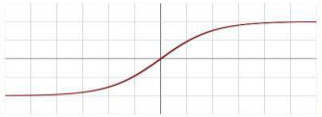
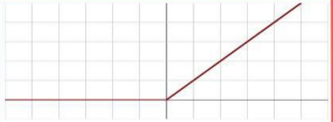
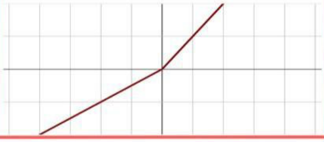
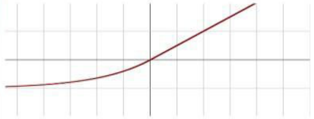


- It allows the model to generalize or adapt to a wide range of data while also distinguishing between the outputs.
- Nonlinear function terms:
  - Derivative or Differential: Change in y-axis w.r.t. change in x-axis. It is also known as slope.
  - Monotonic function: A function that is either completely non-increasing or completely non-decreasing.

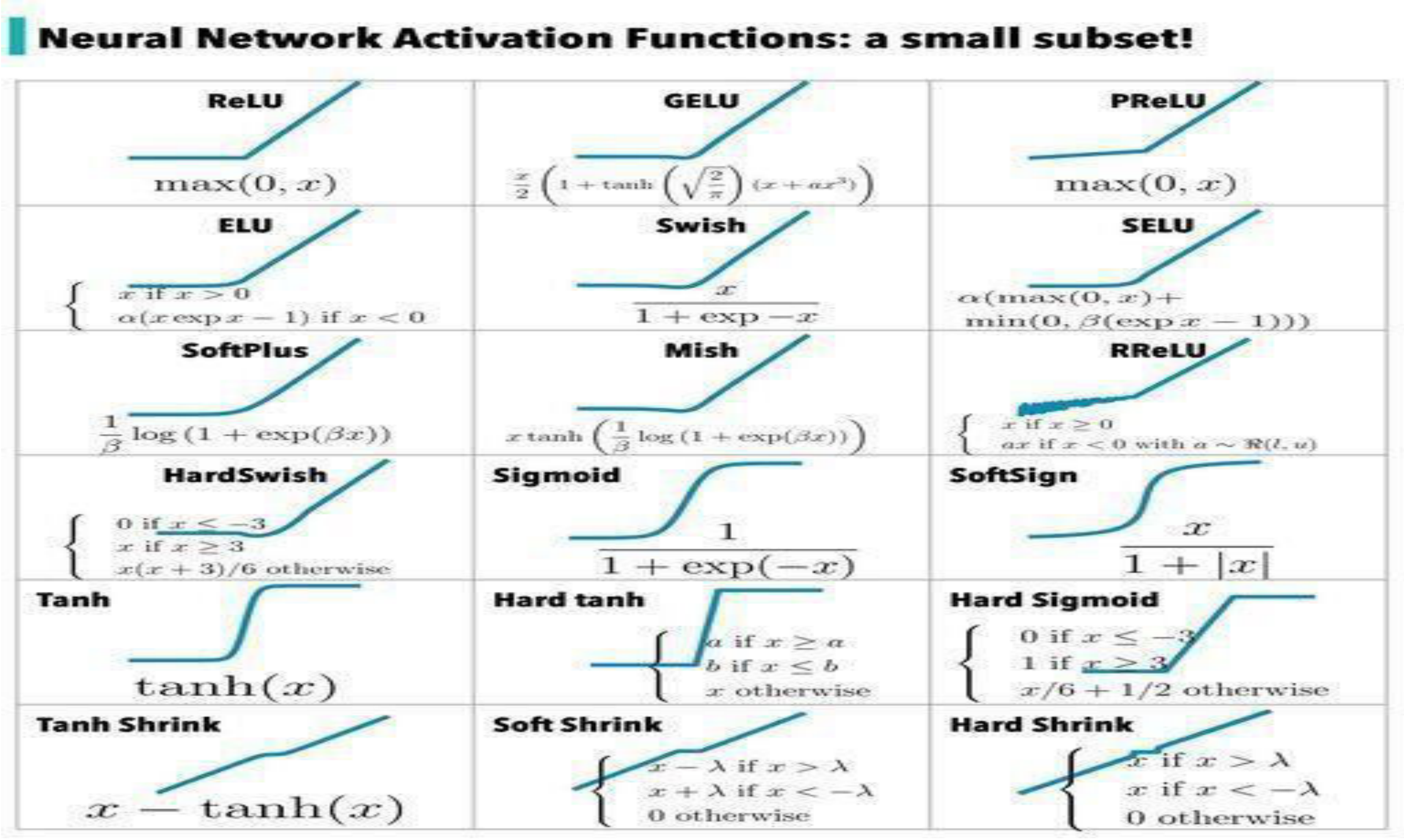
Non-linear Activation Function



Non-linear Activation Function

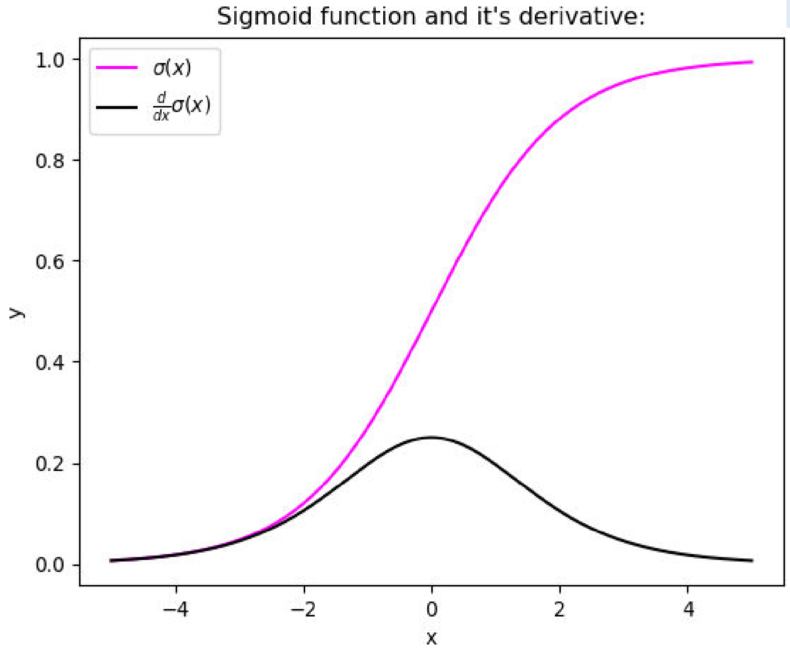
ACTIVATION FUNCTION	PLOT	EQUATION	DERIVATIVE	RANGE
Linear		$f(x) = x$	$f'(x) = 1$	$(-\infty, \infty)$
Binary Step		$f(x) = \begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{if } x \neq 0 \\ \text{undefined} & \text{if } x = 0 \end{cases}$	$\{0, 1\}$
Sigmoid		$f(x) = \sigma(x) = \frac{1}{1+e^{-x}}$	$f'(x) = f(x)(1 - f(x))$	$(0, 1)$
Hyperbolic Tangent(tanh)		$f(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$	$f'(x) = 1 - f(x)^2$	$(-1, 1)$
Rectified Linear Unit(ReLU)		$f(x) = \begin{cases} 0 & \text{if } x<0 \\ x & \text{if } x\geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x > 0 \\ \text{undefined} & \text{if } x = 0 \end{cases}$	$[0, \infty)$
Softplus		$f(x) = \ln(1 + e^x)$	$f'(x) = \frac{1}{1+e^{-x}}$	$(0, 1)$
Leaky ReLU		$f(x) = \begin{cases} 0.01x & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0.01 & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases}$	$(-1, 1)$
Exponential Linear Unit(ELU)		$f(x) = \begin{cases} \alpha(e^x - 1) & \text{if } x \leq 0 \\ x & \text{if } x > 0 \end{cases}$	$f'(x) = \begin{cases} \alpha e^x & \text{if } x < 0 \\ 1 & \text{if } x > 0 \\ 1 & \text{if } x = 0 \text{ and } \alpha = 1 \end{cases}$	$[0, \infty)$

Recent functions



Sigmoid Function

- Sigmoid function is known as the logistic function which helps to normalize the output of any input in the range between 0 to 1.
  - $y = 1/(1+e^{(-x)})$ and $\sigma'(x) = \sigma(x)(1 - \sigma(x))$ (Derivative of Sigmoid Function)
- This function, which is computationally expensive, isn't used in the hidden layers of a convolutional neural network.



Problems of Sigmoid Function are

- *Vanishing gradient*
- *Computationally expensive*
- *The output is not zero centered*
- *The model Learning rate is slow*

TanH (hyperbolic tangent) Activation Function

- TanH is also like a logistic sigmoid but better. The range of the tanh function is from (-1 to 1).

$$tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}.$$

$$\tanh'(x) = 1 - \tanh(x)^2$$

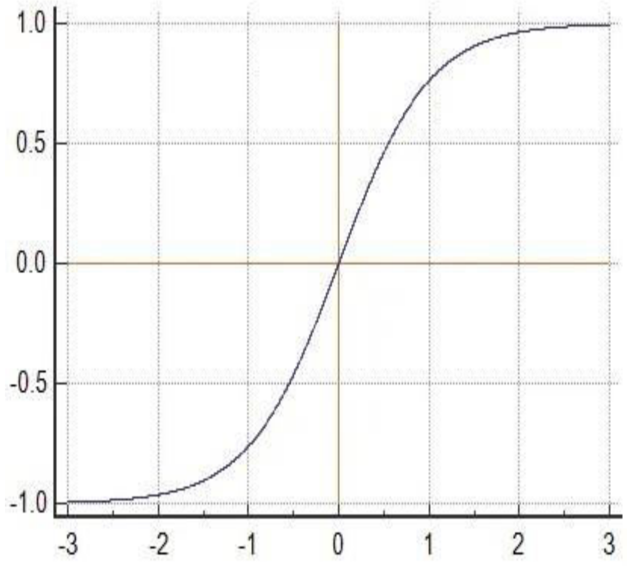
OR

$$f'(x) = 1 - f(x)^2$$

Derivative of TanH

- In Tanh, the larger the input (more positive), the closer the output value will be to 1.0, whereas the smaller the input (more negative), the closer the output will be to - 1.0.

Advantages of TanH function



- We can find the differentiation from this function.
- It's a function that's centered around zero.
- In comparison to sigmoid, optimization is simple.

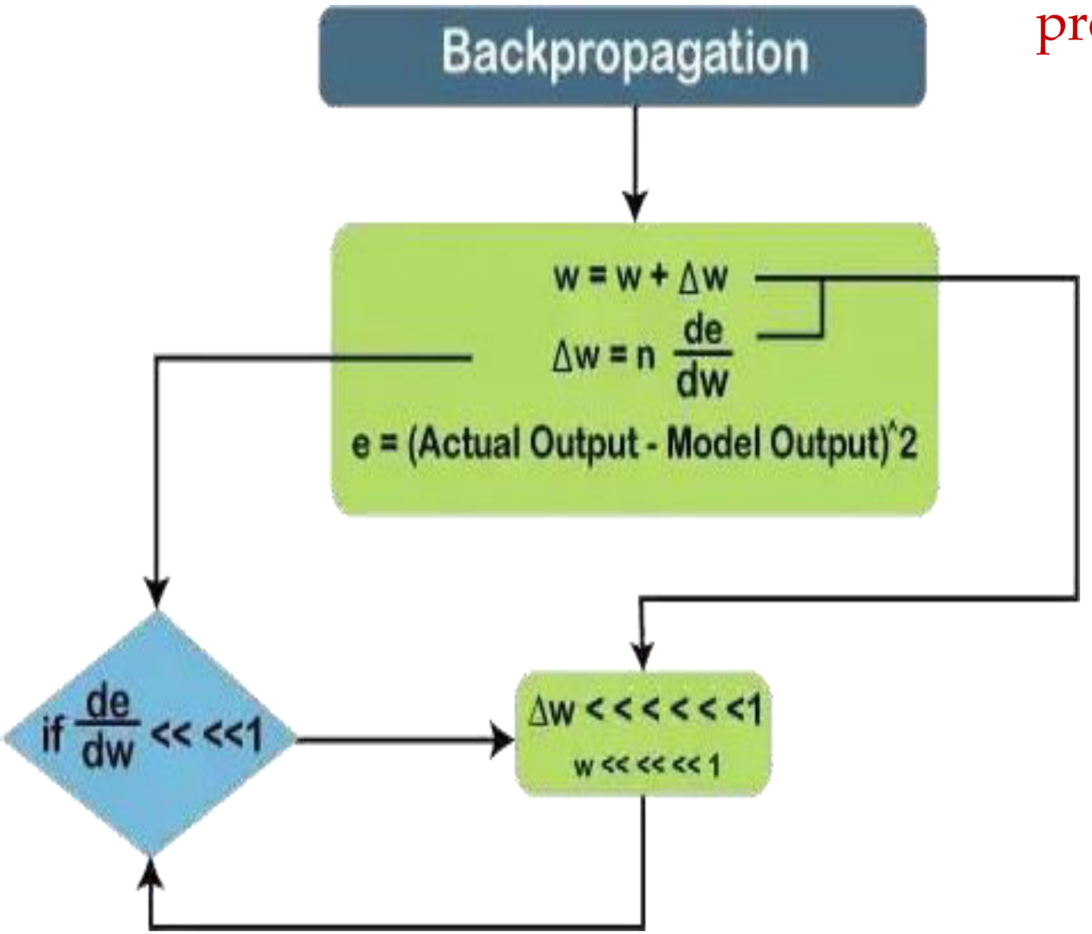
Disadvantages of TanH function

- Because it is a computationally intensive function,
- the conversion will take a long time.
- Vanishing gradients

Vanishing Gradient

- In Deep Learning, During backpropagation, a neural network learns by updating its weights and biases to reduce the loss function.
- In a network with a vanishing gradient, the weights cannot be updated, so the network cannot learn. The performance of the network will decrease as a result.
- As more layers using certain activation functions are added to neural networks, the gradients of the loss function approach zero, making the network hard to train.
- When n hidden layers use an activation like the sigmoid function, n small derivatives are multiplied together. Thus, the gradient decreases exponentially as we propagate down to the initial layers.

Vanishing Gradient Cont'd



Methods proposed to overcome the vanishing gradient problem

1. Multi-level hierarchy
2. Long short – term memory (LSTM)
3. Faster hardware
4. Residual neural networks (ResNets)
5. ReLU

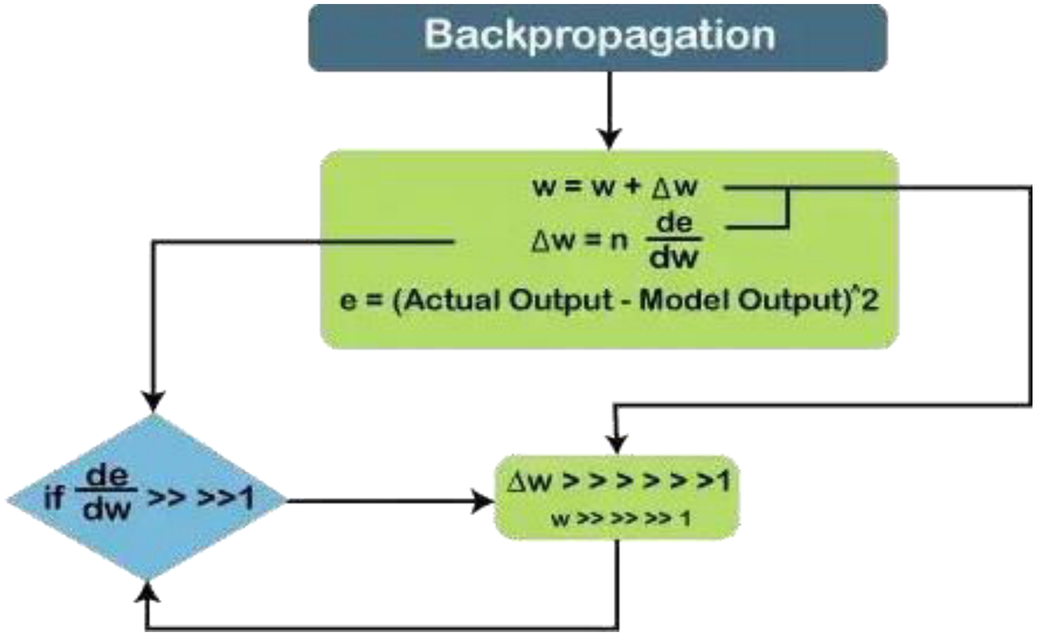
Exploding Gradient

- In some cases, the gradients keep on getting larger and larger as the backpropagation algorithm progresses. This, in turn, causes very large weight updates and causes the gradient descent to diverge. This is known as the *exploding gradients* problem.

- So, The Model will be unstable and unable to learn . Finally, The model weights may become NaN during training.

Precautions Vanishing/ Exploding Gradient

- Proper Weight Initialization
- Using Non-saturating Activation Functions

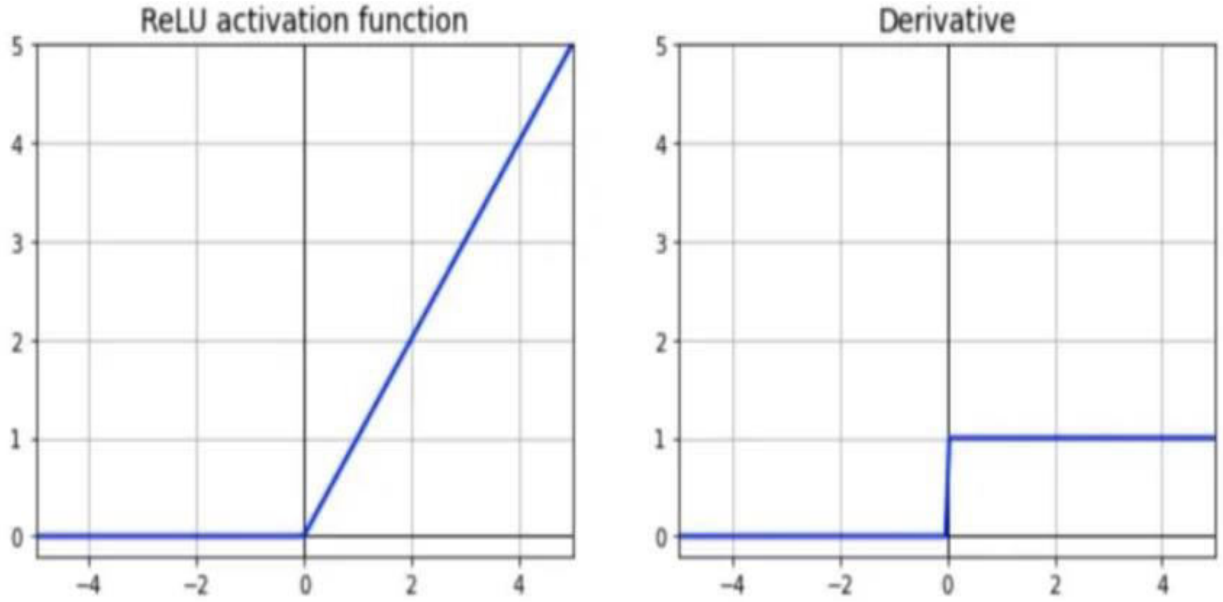


ReLU (Rectified Linear Unit)

- The ReLU is the most widely utilized activation function on the planet.
- It's used in practically all convolutional neural networks and deep learning algorithms.
- Although it gives an impression of a linear function, ReLU has a derivative function and allows for backpropagation while simultaneously making it computationally efficient.

Range: [0 to infinity)

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



- The main catch here is that the ReLU function does not activate all the neurons at the same time.
- The neurons will only be deactivated if the output of the linear transformation is less than 0.

ReLU (Rectified Linear Unit)

Advantages

- We can find a differential.
- Solve the problem of vanishing gradients.
- Because there is no exponential calculation here, the calculation is faster than sigmoid or tanh.
- Linear behavior: A neural network is easier to optimize when its behavior is linear or close to linear

Disadvantages

- It is not a zero-centric function.
- It is fully inactive for -ve input.
- it has a drawback in terms of a problem called as dying neurons.
  - Dead Neurons- they cannot learn from examples for which their activation is zero.

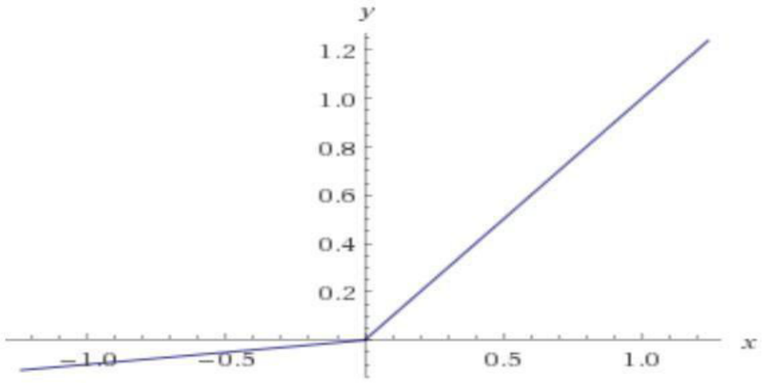
Leaky ReLU Activation Function

- The Leaky ReLU function is superior to the ReLU activation function.
- It has all of the features of ReLU and will never suffer from the Dying ReLU problem.

$$f(x) = \begin{cases} 0.01x & \text{for} \quad x < 0 \\ x & \text{for} \quad x \geq 0 \end{cases}$$

- To address the Dead Neurons problem, Leaky ReLU comes in handy. That is, instead of defining values less than 0 as 0, we instead define negative values as a small linear combination of the input.

- The small value commonly used is 0.01.
- It is represented as \`LeakyReLU(z) = max(0.01 * z, z)\`.
- When a is not 0.01 then it is called Randomized ReLU.



Leaky ReLU Activation Function Cont'd

Advantages

- Performs better as compared to traditionally used activation functions such as Sigmoid and Hyperbolic-Tangent functions and even ReLU.
- It is fast and easy to calculate. The same applies to its derivative which is calculated during the backpropagation.
- It does not saturate for positive values of input and hence does not run into problems related to exploding/vanishing gradients during Gradient Descent.
- Does not suffer from dying ReLU problem.

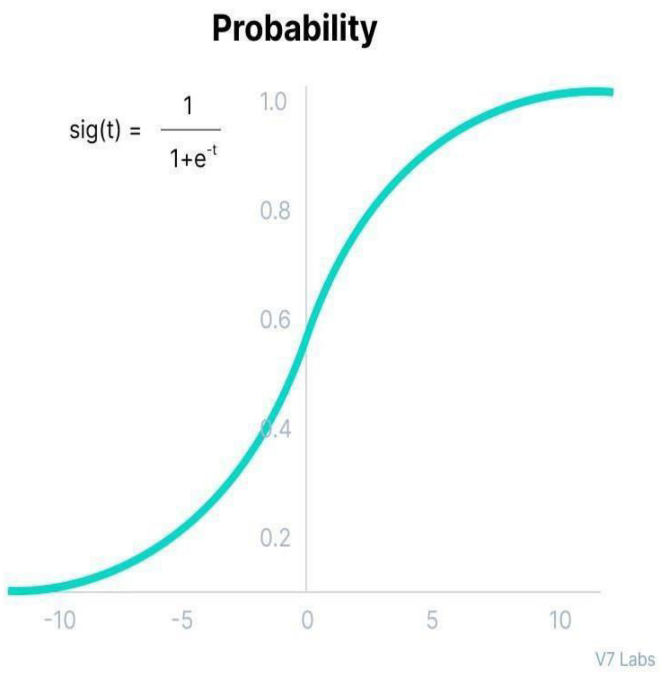
Disadvantages

- Unlike the Parameterized ReLU or PReLU, the value of α is defined prior to the training and hence cannot be adjusted during the training time. The value of α hence chosen might not be the most optimal value.

Softmax

- The SoftMax function is often described as a combination of multiple sigmoids.
- The SoftMax function can be used for multiclass classification problems.
- This function returns the probability for a data point belonging to each individual class.

$$\sigma(\vec{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$



- While building a network for a multiclass problem, the output layer would have as many neurons as the number of classes in the target.
- Suppose you got the output from the neurons as [1.2 , 0.9 , 0.75].
- Applying the softmax function over these values, you will get the following result – [0.42, 0.31, 0.27].

Softmax Cont'd

$\vec{z}$	The input vector to the softmax function, made up of (z0, ... zK)
z_i	All the z_i values are the elements of the input vector to the softmax function, and they can take any real value, positive, zero or negative.
e^{z_i}	The standard exponential function is applied to each element of the input vector. This gives a positive value above 0, which will be very small if the input was negative, and very large if the input was large.
$\sum_{j=1}^{K} e^{z_j}$	The term on the bottom of the formula is the normalization term
K	The number of classes in the multi-class classifier.

Softmax Cont’d

Calculating the Softmax

- Imagine we have an array of three real values.
- These values could typically be the output of a machine learning model such as a neural network.

$$\begin{bmatrix} 8 \\ 5 \\ 0 \end{bmatrix}$$

- First we can calculate the exponential of each element of the input array.

$$e^{z_1} = e^{8} = 2981.0$$
$$e^{z_2} = e^{5} = 148.4$$
$$e^{z_3} = e^{0} = 1.0$$

To obtain the normalization term

$$\sum_{j=1}^{K} e^{z_j} = e^{z_1} + e^{z_2} + e^{z_3} = 2981.0 + 148.4 + 1.0 = 3130.4$$

Softmax Cont'd

Calculating the Softmax

Finally, dividing by the normalization term, we obtain the softmax output for each of the three elements.

$$\sigma(\vec{z})_1 = \frac{2981.0}{3130.4} = 0.9523$$
$$\sigma(\vec{z})_2 = \frac{148.4}{3130.4} = 0.0474$$
$$\sigma(\vec{z})_3 = \frac{1.0}{3130.4} = 0.0003$$

- It is informative to check that we have three output values which are all valid probabilities, that is they lie between 0 and 1, and they sum to 1.

Softmax Cont’d

Calculating the Softmax - II

Class	Value	One-Hot Encoding
0		[1, 0, 0]
1		[0, 1, 0]
2		[0, 0, 1]

→ Model (without activation) →

Raw Output

[0.25, 1.23, -0.8]

$$\sigma(\vec{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

$$Pr[y_i = seal] = softmax(\mathbf{z})_0 = \frac{e^{0.25}}{e^{0.25} + e^{1.23} + e^{-0.8}} = 0.249$$

$$Pr[y_i = duck] = softmax(\mathbf{z})_2 = \frac{e^{-0.8}}{e^{0.25} + e^{1.23} + e^{-0.8}} = 0.087$$

$$Pr[y_i = panda] = softmax(\mathbf{z})_1 = \frac{e^{1.23}}{e^{0.25} + e^{1.23} + e^{-0.8}} = 0.664$$

Softmax Cont'd

Calculating the Softmax - II

$$Pr[y_i = seal] = softmax(\mathbf{z})_0$$
$$= \frac{e^{0.25}}{e^{0.25} + e^{1.23} + e^{-0.8}} = 0.249$$

$$Pr[y_i = duck] = softmax(\mathbf{z})_2$$
$$= \frac{e^{-0.8}}{e^{0.25} + e^{1.23} + e^{-0.8}} = 0.087$$

$$Pr[y_i = panda] = softmax(\mathbf{z})_1$$
$$= \frac{e^{1.23}}{e^{0.25} + e^{1.23} + e^{-0.8}} = 0.664$$

$$\begin{bmatrix} 0.25 \\ 1.23 \\ -0.8 \end{bmatrix} \longrightarrow \boxed{\text{Softmax}} \longrightarrow \begin{bmatrix} 0.249 \\ 0.664 \\ 0.087 \end{bmatrix}$$

Softmax Cont'd

Advantages

- The main advantage of using Softmax is the output probabilities range. The range will be **0 to 1**, and the sum of all the probabilities will be **equal to one**.
- If the softmax function is used for multi-classification model it returns the probabilities of each class and the target class will have a high probability.

Softmax Function Usage

- Used in multiple classification logistic regression model.
- In building neural networks softmax functions used in different layer level.

Avoid softmax

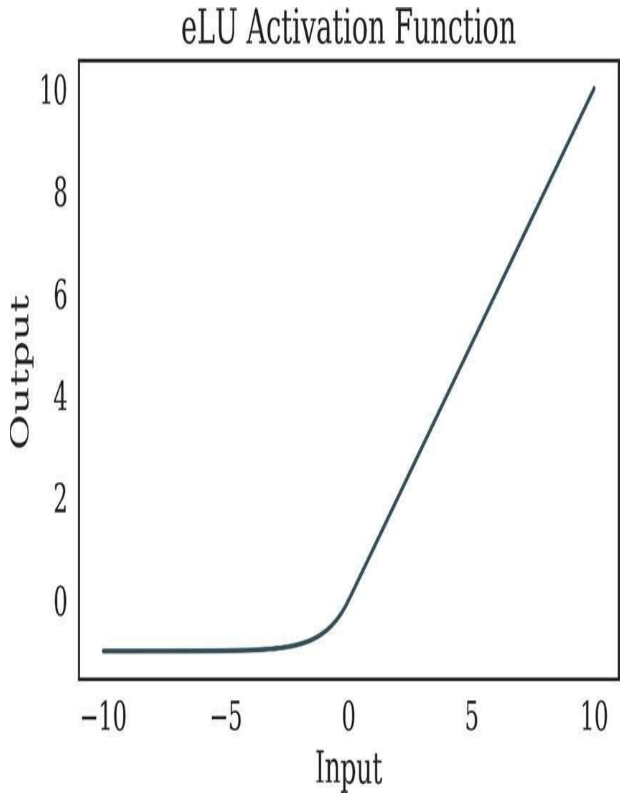
- Many labels in one image, then softmax should not be used.
- In object detection (YOLO(3) models) sigmoid is used.

Exponential Linear Units (ELUs) Function

- Exponential Linear Unit, or ELU for short, is also a variant of ReLU that modifies the slope of the negative part of the function.
- ELU uses a log curve to define the negative values unlike the leaky ReLU and Parametric ReLU functions with a straight line.
- Since ELUs can have negative values it pushes the mean of the activations closer to zero.
- Having mean activations closer to zero also causes faster learning and convergence.

$$f(x) = \begin{cases} x, \text{ if } x > 0 \\ \alpha(e^x - 1), \text{ if } x \leq 0 \end{cases}$$

Exponential Linear Units (ELUs) Function



Advantages

- ELU is a smooth function for negative values, making it more noise-robust.
- For any positive output, it behaves like a step function and gives a constant output.

Disadvantages:

- It saturates for large negative values, allowing them to be essentially inactive.

Summary

Function	Range	0-centered	Saturation	Vanishing Gradient	Computation
Sigmoid	0,1	No	For negative and positive values	Yes	Compute-intensive
Tanh	-1,1	Yes	For negative and positive values	Yes	Compute-intensive
ReLu	0, +∞	No	For negative values	Yes (Better than sigmoid and tanh)	Easy to compute
Leaky ReLu	-∞, +∞	Close	No	No	Easy to compute

Summary Cont'd

Consideration	Activation Function
Non-linearity	Sigmoid, Tanh, ReLU, Leaky ReLU, ELU, SELU
Derivability	Sigmoid, Tanh, ReLU, Leaky ReLU, ELU, SELU
Range of output values	Sigmoid, Softmax
Computational efficiency	ReLU, Leaky ReLU, ELU, SELU
Saturation	ReLU, Leaky ReLU, ELU, SELU

Summary Cont'd

- ReLU activation function should only be used in the hidden layers.
- Sigmoid/Logistic and Tanh functions should not be used in hidden layers as they make the model more susceptible to problems during training since it might be slow(due to vanishing gradients).
- Swish function is used in neural networks having a depth greater than 40 layers.
- Applications such as anomaly detection, recommender systems uses ReLu/Tanh (depends)

Assume we have a 2-input neuron that uses the sigmoid activation function and has the following parameters:

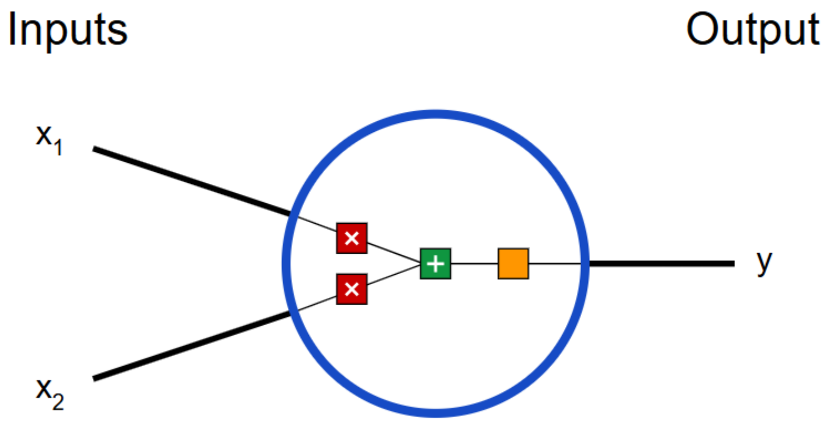
$$w = [0, 1]$$

$$b = 4$$

$w = [0, 1]$ is just a way of writing $w_1 = 0, w_2 = 1$ in vector form. Now, let's give the neuron an input of $x = [2, 3]$. We'll use the dot product to write things more concisely:

$$\begin{aligned}(w \cdot x) + b &= ((w_1 * x_1) + (w_2 * x_2)) + b \\ &= 0 * 2 + 1 * 3 + 4 \\ &= 7\end{aligned}$$

$$y = f(w \cdot x + b) = f(7) = \boxed{0.999}$$

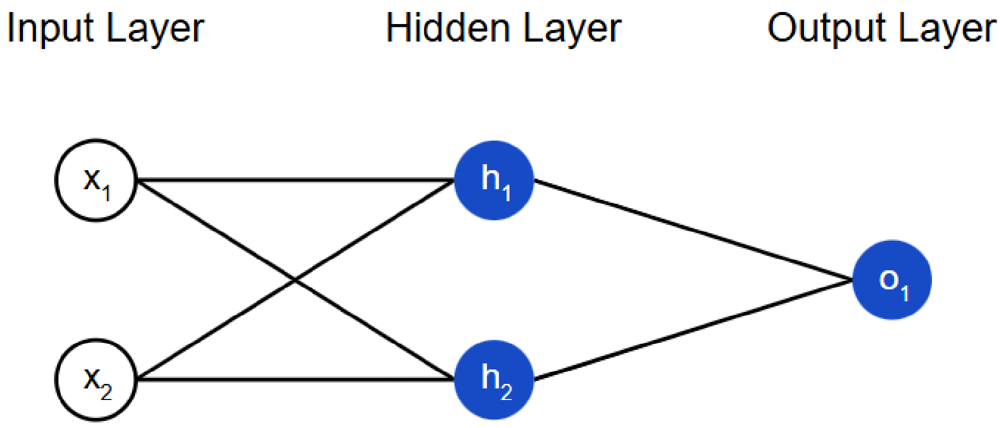


The neuron outputs 0.999 given the inputs $x = [2, 3]$. That's it! This process of passing inputs forward to get an output is known as **feedforward**.

Let's use the network pictured above and assume all neurons have the same weights $w = [0, 1]$, the same bias $b = 0$, and the same sigmoid activation function. Let h_1, h_2, o_1 denote the *outputs* of the neurons they represent.

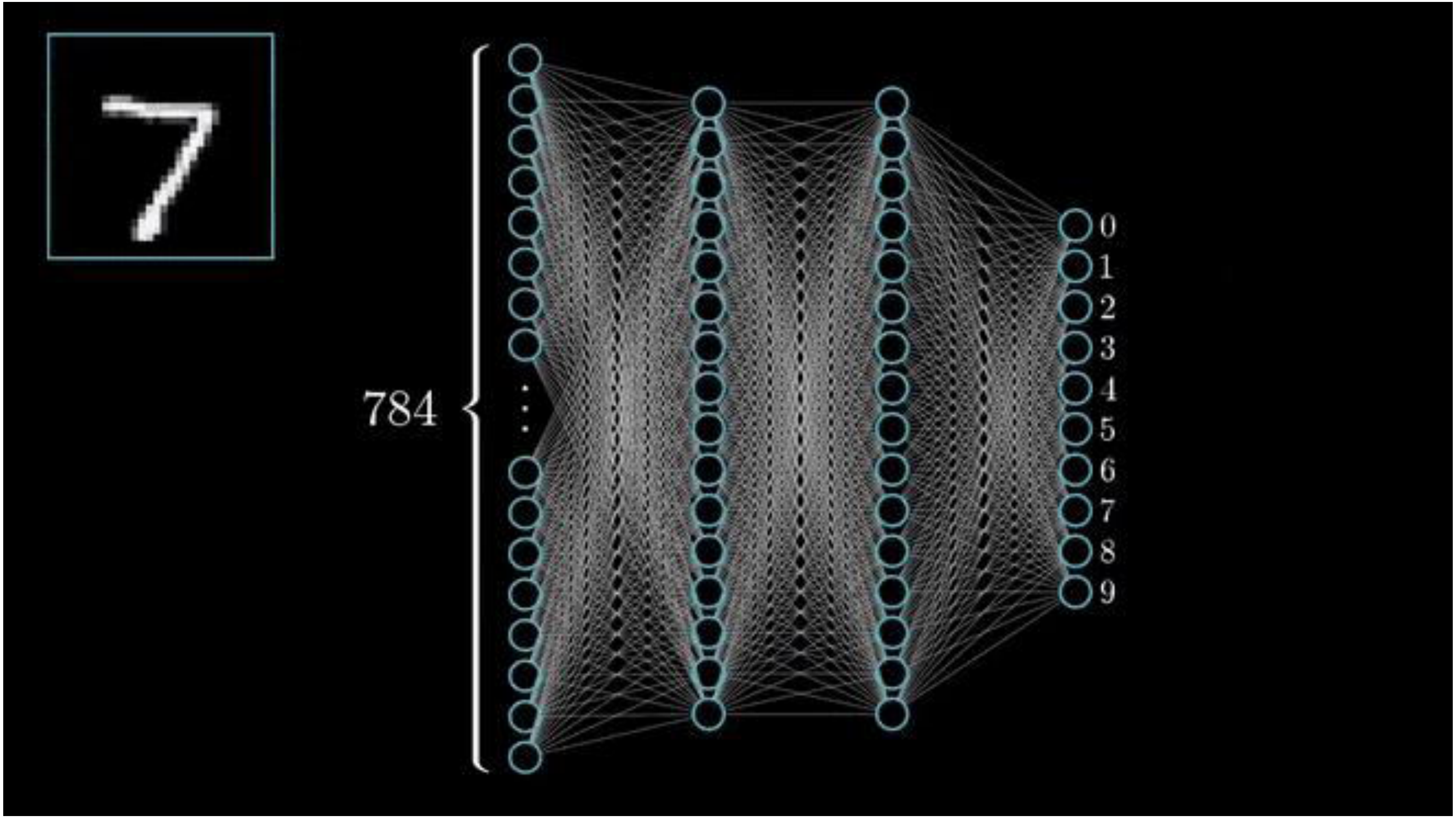
What happens if we pass in the input $x = [2, 3]$?

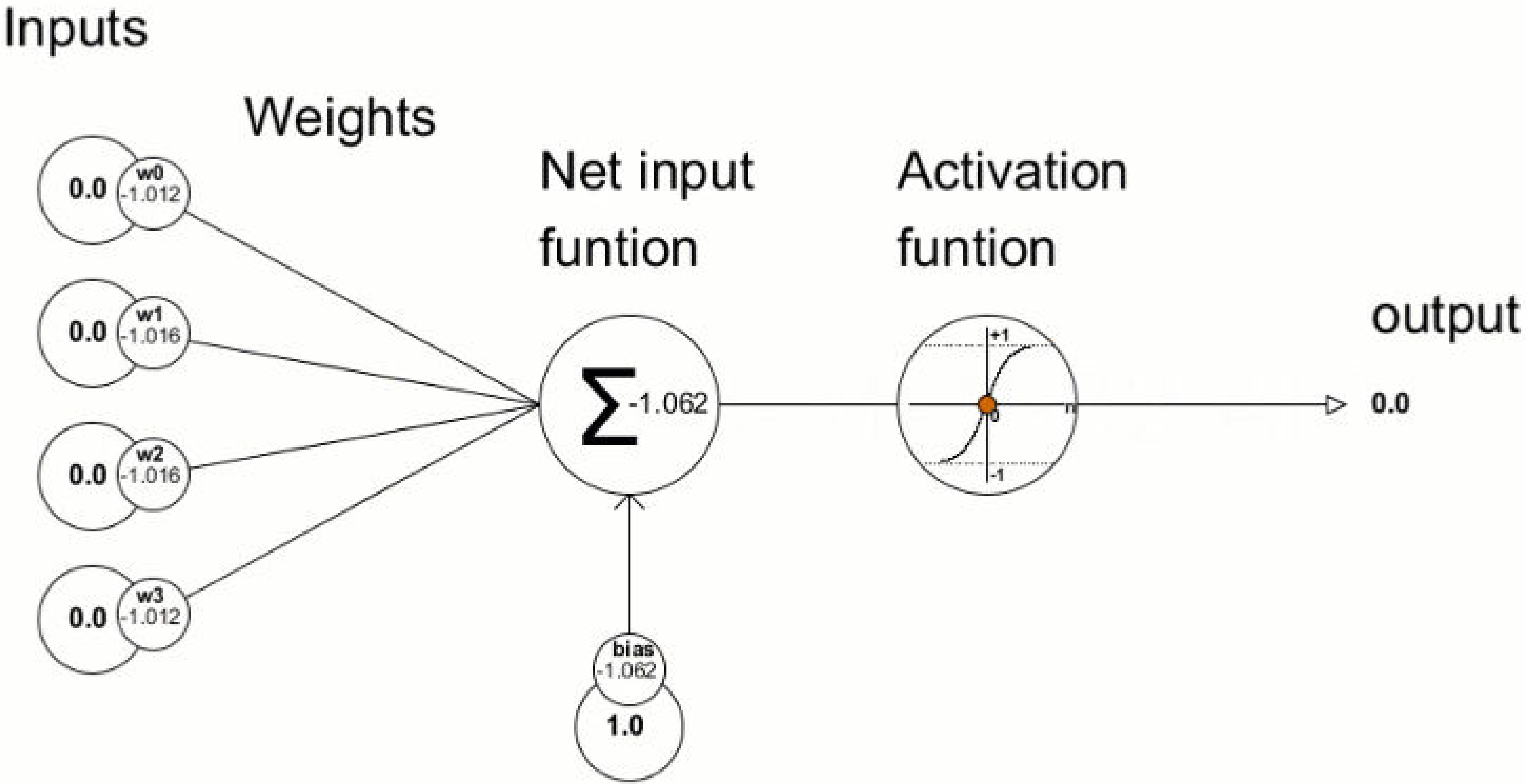
$$\begin{aligned} h_1 = h_2 &= f(w \cdot x + b) \\ &= f((0 * 2) + (1 * 3) + 0) \\ &= f(3) \\ &= 0.9526 \end{aligned}$$



$$\begin{aligned} o_1 &= f(w \cdot [h_1, h_2] + b) \\ &= f((0 * h_1) + (1 * h_2) + 0) \\ &= f(0.9526) \\ &= \boxed{0.7216} \end{aligned}$$

The output of the neural network for input $x = [2, 3]$ is 0.7216. Pretty simple, right?


7
784
0
1
2
3
4
5
6
7
8
9



Inputs
Weights
Net input
funtion
Activation
funtion
output
0.0
w0
-1.012
0.0
w1
-1.016
0.0
w2
-1.016
0.0
w3
-1.012
Σ -1.062
+1
0
-1
bias
-1.062
1.0
0.0

Q 2. Consider a task of image classification. There are 10 million images belonging to 1000 classes. Each image has resolution 32 x 32.

1. **Construct an ANN with 3 hidden layers (hidden layer 1 - 5 neurons, hidden layer 2 - 10 neurons and hidden layer 3 - 10 neurons) which takes input as image and classifies it to one of the class**
2. **Calculate total number of learnable parameters in each layer**
3. **Which activation functions you will consider in each layer of the network? justify your answer** **10 Marks**

- Input:
  - Each image has a resolution of 32×32, meaning there are $32 \times 32 = 1024$ input features (neurons in the input layer).
- Hidden layers:
  - **Hidden layer 1**: 5 neurons
  - **Hidden layer 2**: 10 neurons
  - **Hidden layer 3**: 10 neurons
- Output:
  - The output layer has 1000 neurons (one for each class).

$$\text{Learnable Parameters} = (\text{Number of Input Neurons} \times \text{Number of Output Neurons}) + \text{Number of Output Neurons (Bias Terms)}$$

1. **Input to Hidden Layer 1**:
   - Input neurons: 1024
   - Output neurons: 5
   - Parameters:

$$(1024 \times 5) + 5 = 5120 + 5 = 5125$$

2. **Hidden Layer 1 to Hidden Layer 2**:
   - Input neurons: 5
   - Output neurons: 10
   - Parameters:

$$(5 \times 10) + 10 = 50 + 10 = 60$$

3. **Hidden Layer 2 to Hidden Layer 3**:
   - Input neurons: 10
   - Output neurons: 10
   - Parameters:

$$(10 \times 10) + 10 = 100 + 10 = 110$$

4. **Hidden Layer 3 to Output Layer**:
   - Input neurons: 10
   - Output neurons: 1000
   - Parameters:

$$(10 \times 1000) + 1000 = 10000 + 1000 = 11000$$

Total Parameters:

$$5125 + 60 + 110 + 11000 = 16295$$

1. **Hidden Layers**:
    - Use **ReLU (Rectified Linear Unit)**:

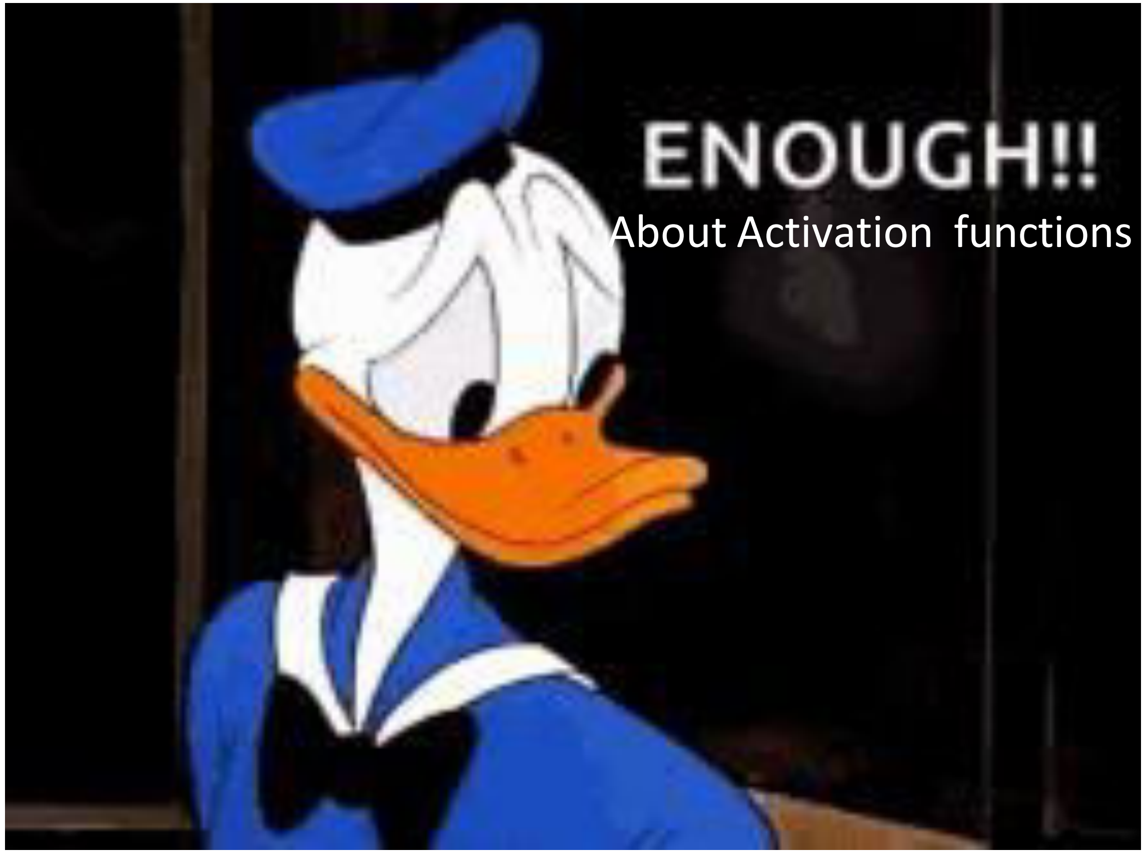
$$\text{ReLU}(x) = \max(0, x)$$

        - Justification:
            - ReLU is computationally efficient.
            - It mitigates the vanishing gradient problem.
            - Encourages sparsity in activations, which helps reduce overfitting.
2. **Output Layer**:
    - Use **Softmax**:

$$\text{Softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{C} e^{z_j}}$$

        - Justification:
            - Softmax is suitable for multi-class classification.
            - It converts logits into probabilities, ensuring the sum of outputs is 1.

1. **Structure**:
    - Input Layer: 1024 neurons
    - Hidden Layer 1: 5 neurons
    - Hidden Layer 2: 10 neurons
    - Hidden Layer 3: 10 neurons
    - Output Layer: 1000 neurons
2. **Learnable Parameters**:
    - Total: 16295
3. **Activation Functions**:
    - Hidden layers: ReLU
    - Output layer: Softmax


ENOUGH!!
About Activation functions