

# Aim

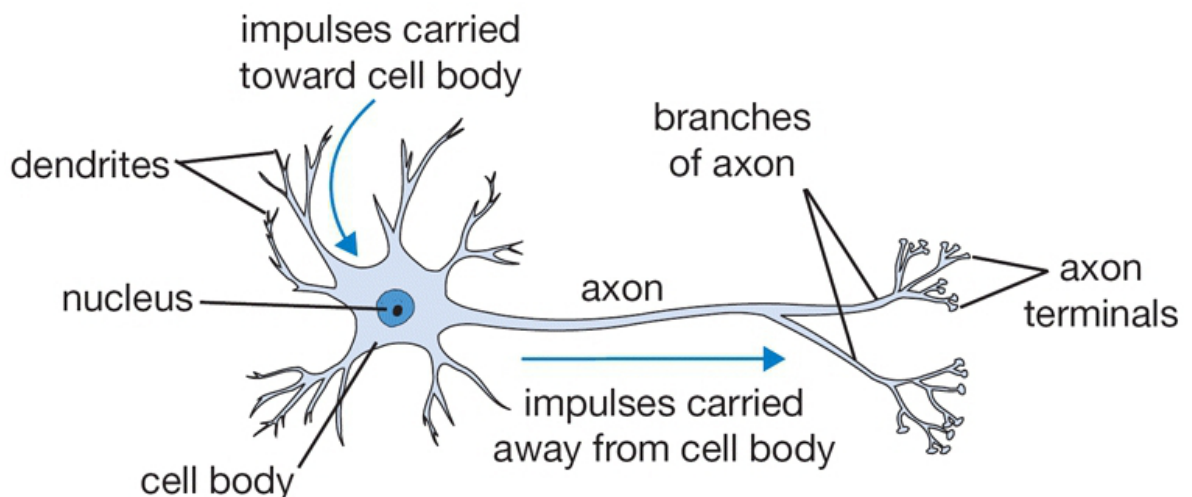
- Understand a neuron

## Modeling one neuron / Biological motivations and connections

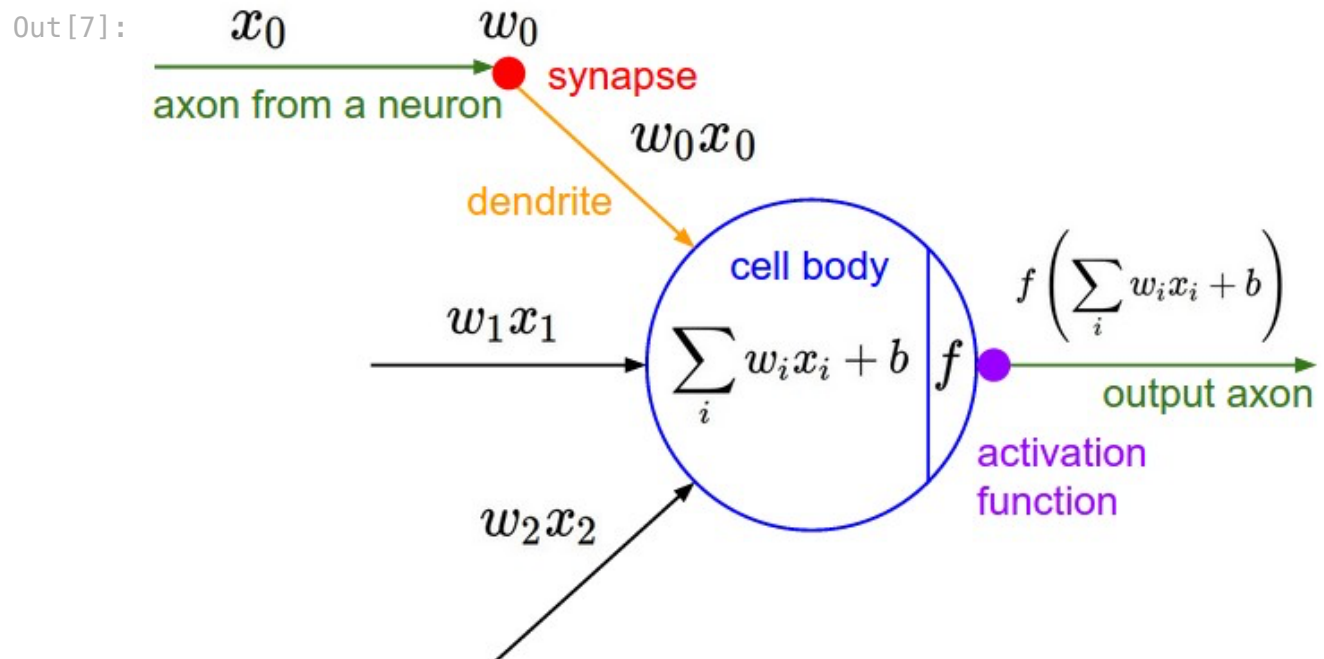
- Neuron is a basic computational unit of a brain
- Each neuron receives input signal from dendrites and produces output signal via axon
- Axon of a neuron connects to dendrites via synapse (weight,  $w$ )
- Signals travel along axons ( $x$ ) and interact multiplicatively with synapse ( $wx$ ) with the dendrite of another neuron
- A neuron sums  $wx$  from all the dendrites-synapse and adds a bias  $b$ . If the sum is above a threshold it will fire sending a spike via axon to the connected neuron.
- The synaptic strengths (i.e.,  $w$ ) and the bias can be learned and we can control the influence of one neuron on another
- Activation function is used to bound the signal on axon within a specific range (e.g., -1 to 1). Commonly used activation functions are sigmoid, tanh, ReLU, Leaky ReLU, Maxout

```
In [6]: from IPython.display import Image
from IPython.core.display import HTML
PATH_NEURON = "/Users/sunil/Yashvi/img/neuron.png"
Image(filename = PATH_NEURON )
```

Out [6]:



```
In [7]: PATH_NEURON_MODEL = "/Users/sunil/Yashvi/img/neuron_model.jpeg"
Image(filename = PATH_NEURON_MODEL )
```



A basic neuron is expressed as  $f(\sum_i w_i x_i + b)$

Where:

- The inputs  $x_i$  (signals from connected neuron's axon) are multiplied by their corresponding weights  $w_i$  (synapse)
- All weighted inputs are summed together ( $\sum_i w_i x_i$ )
- A bias term  $b$  is added
- The result is passed through an activation function  $f$

## Activation Function

### Sigmoid

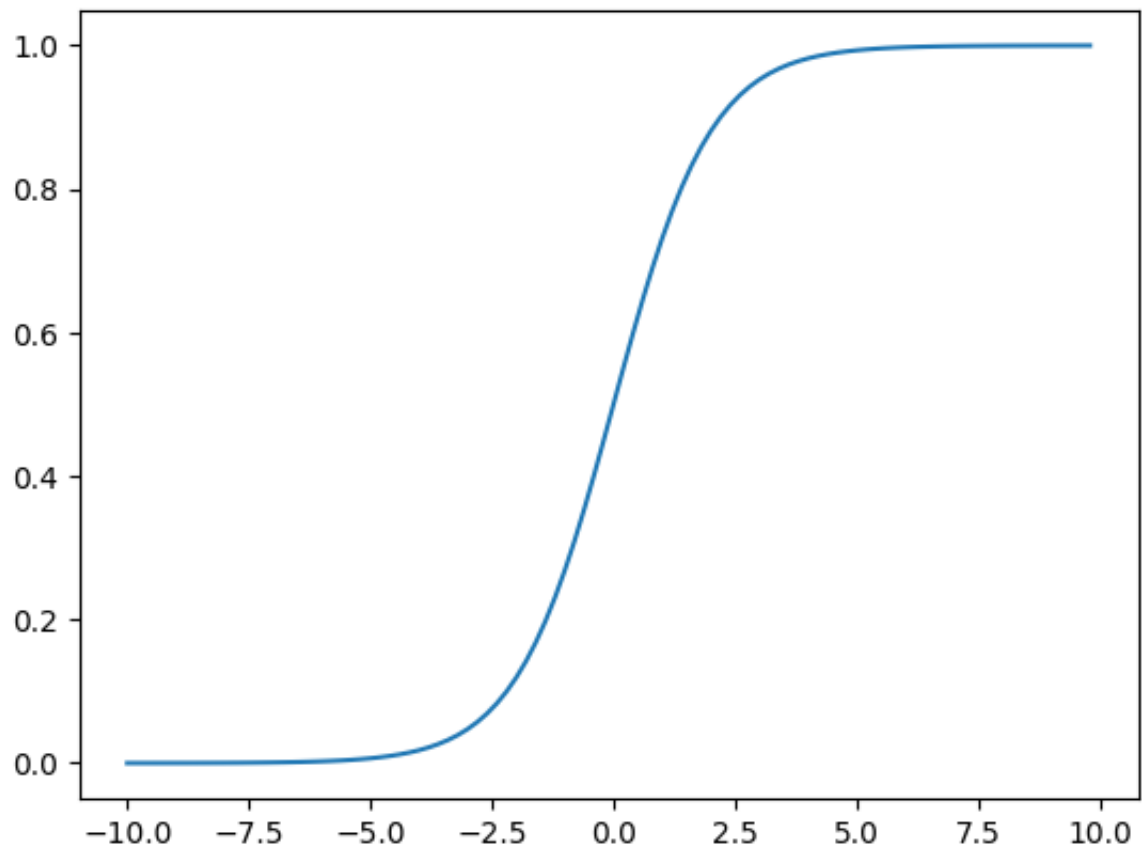
The sigmoid function is defined as:  $\sigma(x) = \frac{1}{1+e^{-x}}$

```
In [ ]: import math
import numpy as np
import matplotlib.pyplot as plt

def sigmoid(x):
    return 1 / (1 + np.exp(-x))

# Example usage
xs = np.arange(-10, 10, 0.2)
ys = sigmoid(xs)
plt.plot(xs, ys)
```

Out[ ]: [matplotlib.lines.Line2D at 0x10cc17f40>]



- When  $x$  is a large negative number  $\rightarrow$  output approaches 0
- When  $x$  is a large positive number  $\rightarrow$  output approaches 1
- When  $x = 0 \rightarrow$  output = 0.5

## Derivative of sigmoid

$$\sigma'(x) = \sigma(x)(1 - \sigma(x))$$

## Drawbacks of sigmoid function

### The Saturation

When  $x$  is very large positive or negative, the derivative of the sigmoid function is 0. As a result, while doing gradient descent, optimization takes very small steps and learning becomes extremely slow.

### Sigmoid outputs are not zero-centered

TODO: Understand why it is a drawback

## Tanh

The tanh function is defined as:

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

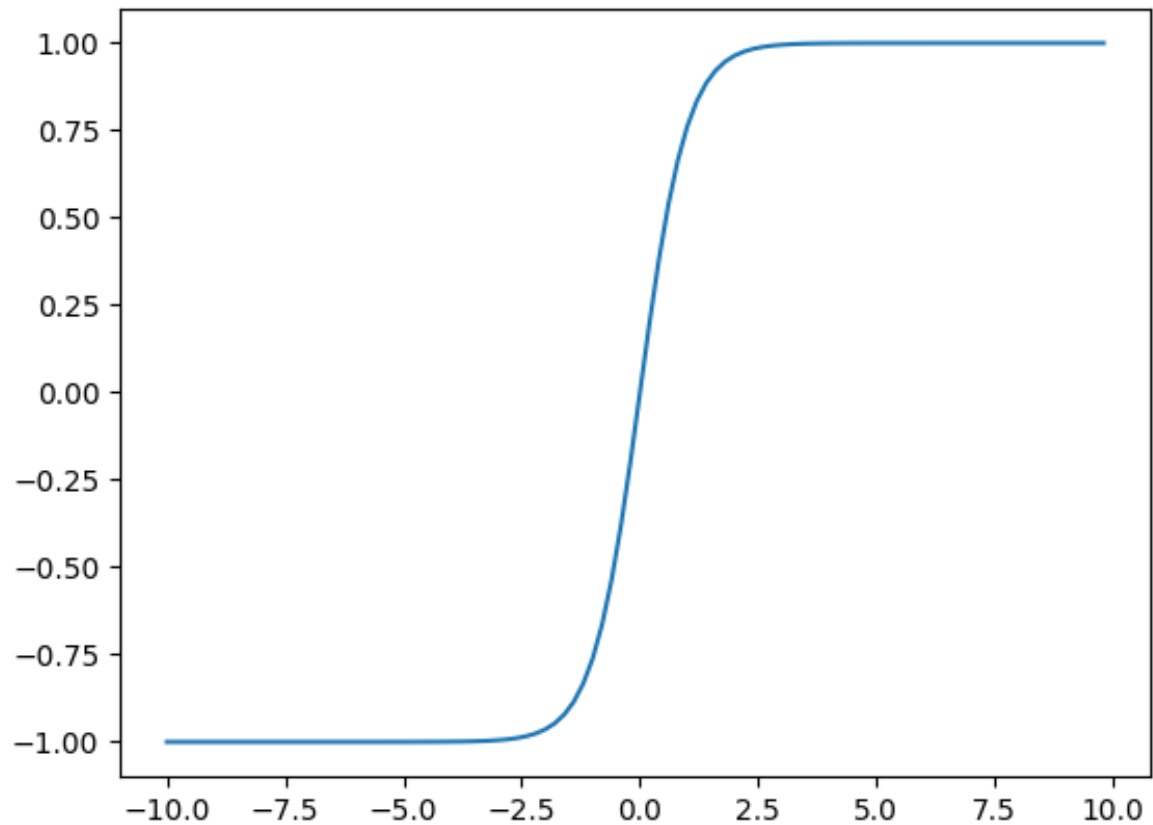
It squashes a real-valued number to the range  $[-1, 1]$ .

- Unlike sigmoid, tanh is zero-centered.

```
In [9]: def tanh(x):
        return (np.exp(x) - np.exp(-x)) / (np.exp(x) + np.exp(-x))

xs = np.arange(-10, 10, 0.2)
tanhy = tanh(xs)
plt.plot(xs, tanhy)
```

```
Out[9]: [matplotlib.lines.Line2D at 0x10cd23c10>]
```



## Derivative of tanh

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

## ReLU

Rectified Linear Unit is defined as:

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

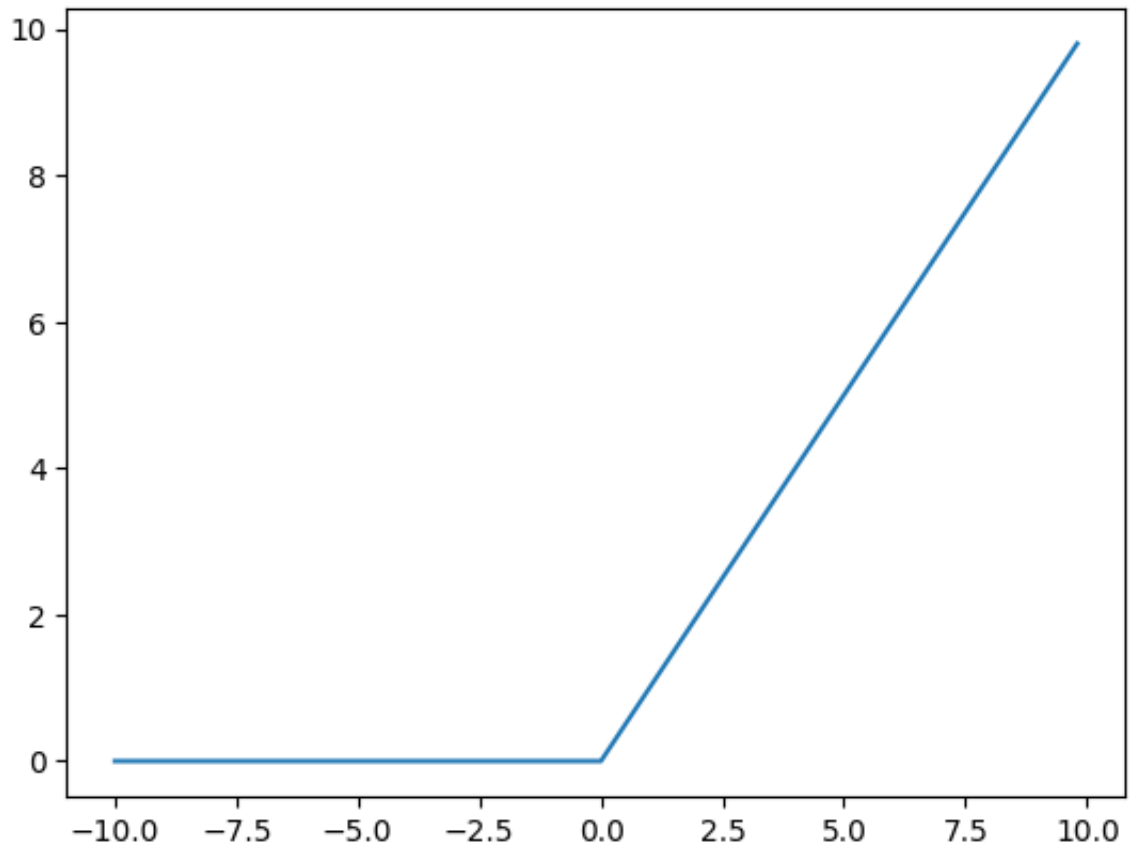
## Derivative of ReLU

$$\text{The derivative of ReLU: } f'(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{if } x < 0 \\ \text{undefined} & \text{if } x = 0 \end{cases}$$

ReLU is computationally very simple.

```
In [10]: def draw_relu():  
         xs = np.arange(-10, 10, 0.2)  
         ys = np.maximum(xs, 0)  
         plt.plot(xs, ys)
```

```
draw_relu()
```



## Refereces

- Neuron : <https://cs231n.github.io/neural-networks-1>
- Sigmoid function : <https://youtu.be/3nmAA30MDFg?si=Z3oeZXm8NjN2FQg0>