تقني
معلومة مفيدة .. وقت قليل

ANNs: Weight, Bias, & Activation Functions

مبادئ الشبكات العصبية الاصطناعية

Eng. Mustafa Othman
Data Scientist & Analyst

Today’s Outline:

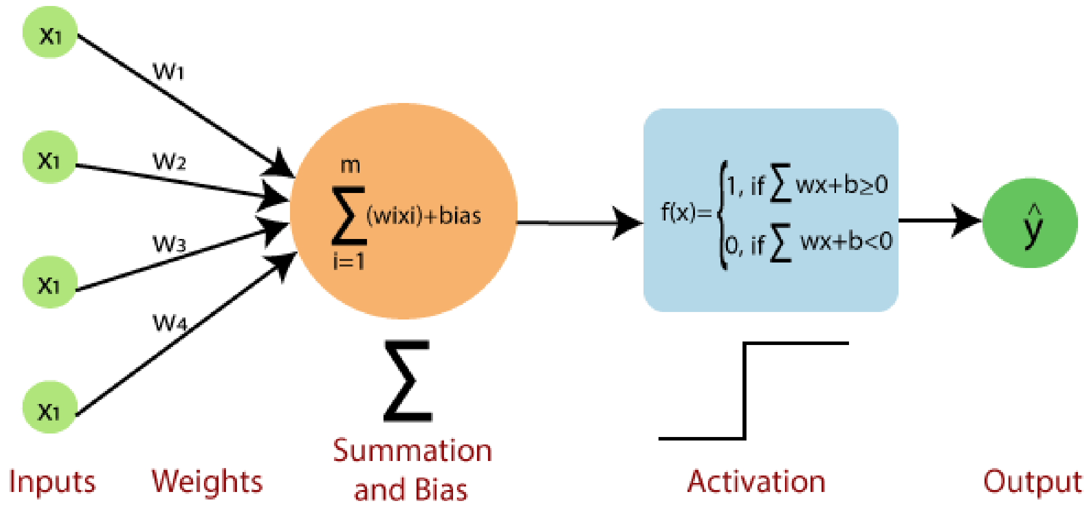
- **Artificial Neural Networks (ANNs) Overview**
  - Building, Training & Optimizing ANNs
- **Artificial Neuron Basic Components**
  - **Weights (w) & Biases (b)**
  - **Activation Functions**
    - Importance of Activation Functions
    - Types of Activation Functions
    - Which one to Choose?
  - **Demo: Simple Artificial Neural Network using Python**

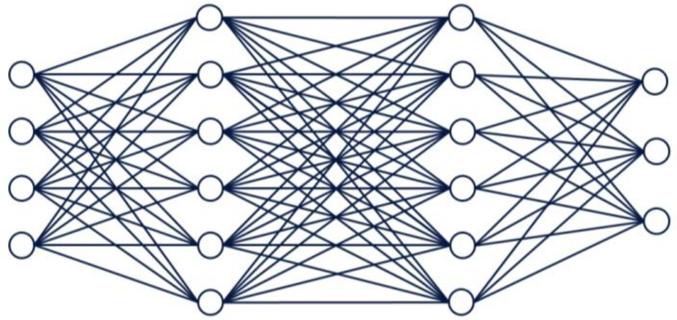
Artificial Neural Networks (ANNs) Overview

“We are all now connected by the Internet, like neurons in a giant brain.” ~ Stephen Hawking

Artificial Neural Networks Overview (1) (Building ANNs)

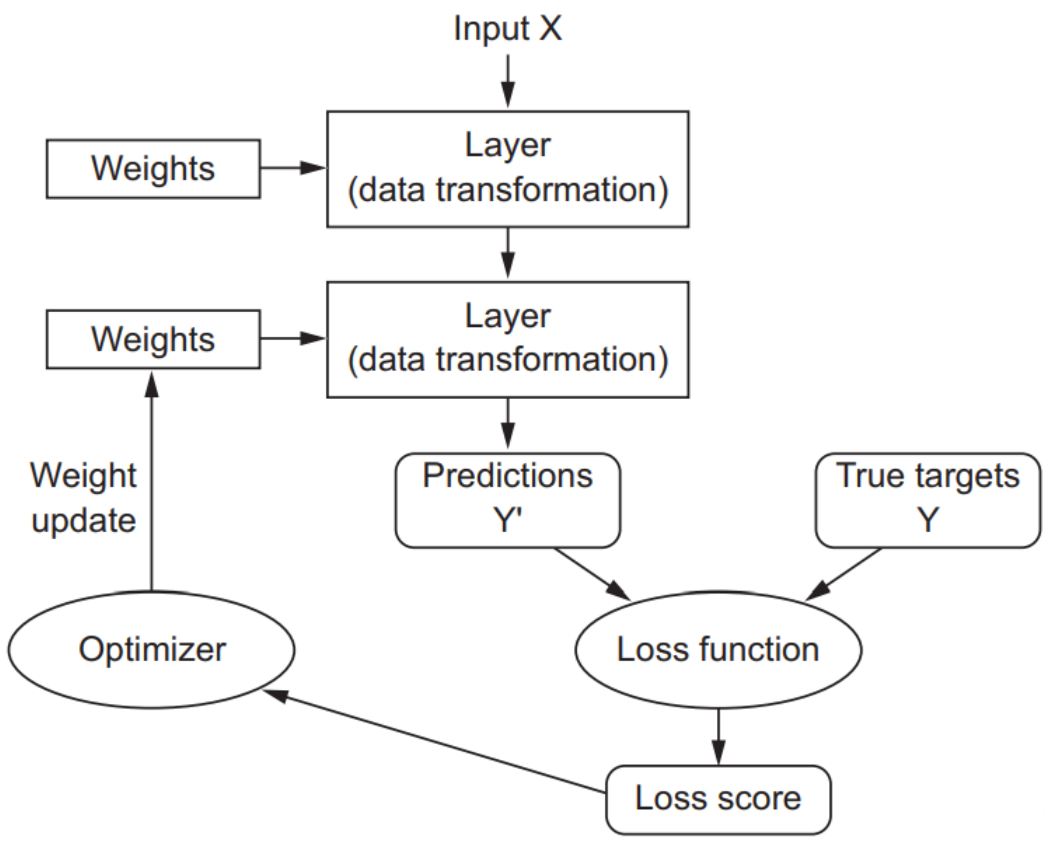
- **Parameters:**
  - Weights (**w**)
  - Bias (**b**)
- **Activation Functions:**
  - Sigmoid
  - ReLU
  - Linear
- **Layers:**
  - Input Layer
  - Hidden Layer(s)
  - Output Layer


x1
x1
x1
x1
w1
w2
w3
w4
$\sum_{i=1}^{m}$ (wixi)+bias
$f(x)=\begin{cases}1, \text{if } \sum wx+b\geq 0\\ 0, \text{if } \sum wx+b<0\end{cases}$
$\hat{y}$
Σ
Inputs
Weights
Summation and Bias
Activation
Output



Artificial Neural Networks Overview (2)
(Training & Optimizing ANNs)

- **Forward Propagation**
- **Backward Propagation**
  - Updating Weights
- **Loss Functions:**
  - Regression (Mean Squared Error (**MSE**))
  - Classification (**Cross-Entropy**)
- **Optimizers:**
  - SGD
  - Adam
- **Optimizer Hyperparameters:**
  - Epochs & Batch Size
  - Learning Rate (**η**)

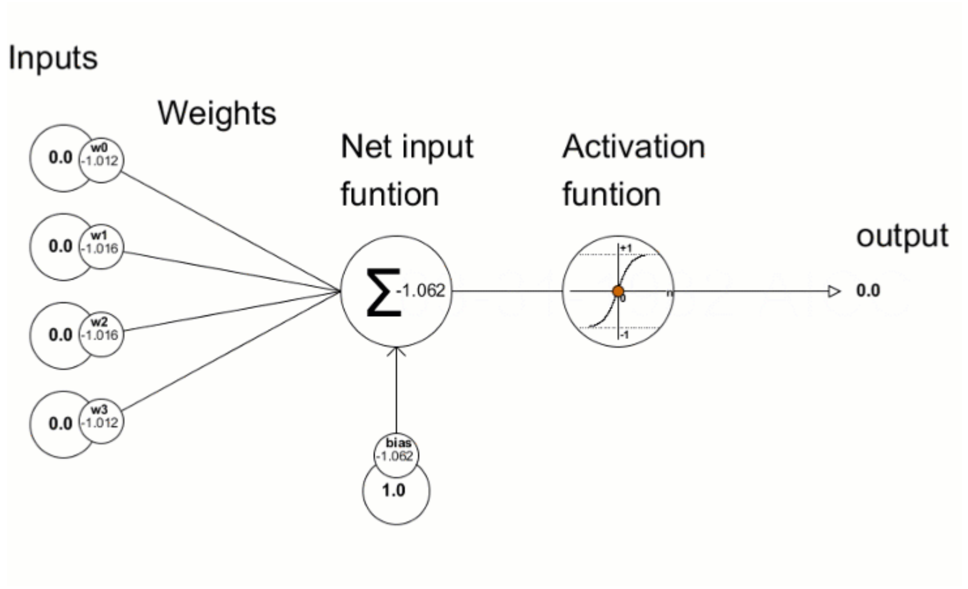


Weights & Biases

“There is no strength without struggle.” ~ Unknown

Weights & Biases (0) (Overview)

- **Weights** and **biases** (commonly referred to as **w** and **b**) are the **learnable parameters** of some machine learning models, including neural networks.
- **Weight** is like **slope** in linear regression, where a weight is multiplied to the input to add up to form the output, they are like the **coefficients** of the equation which you are trying to resolve.
- **Bias** is like the **intercept** added in a linear equation. It is an additional parameter which is used to **adjust** the output along with the weighted sum of the inputs to the neuron.



$$y = f(x) = \sum x_i w_i$$

$$x_1 w_1 + x_2 w_2 + \cdots + x_n w_n + bias$$

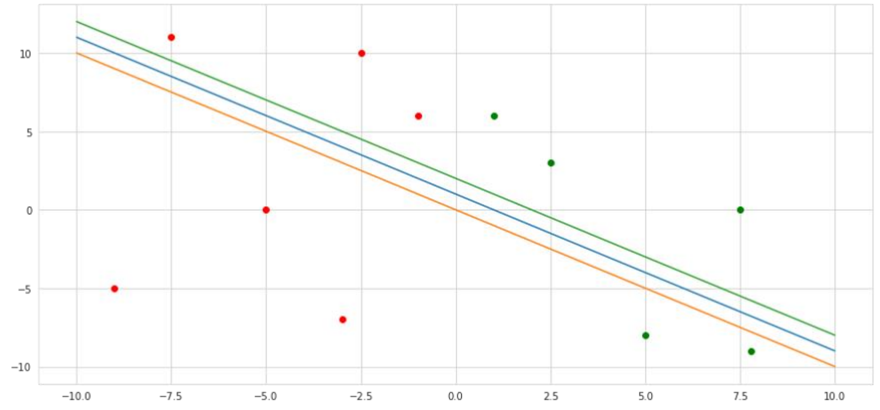
$$output = sum(weights * inputs) + bias$$

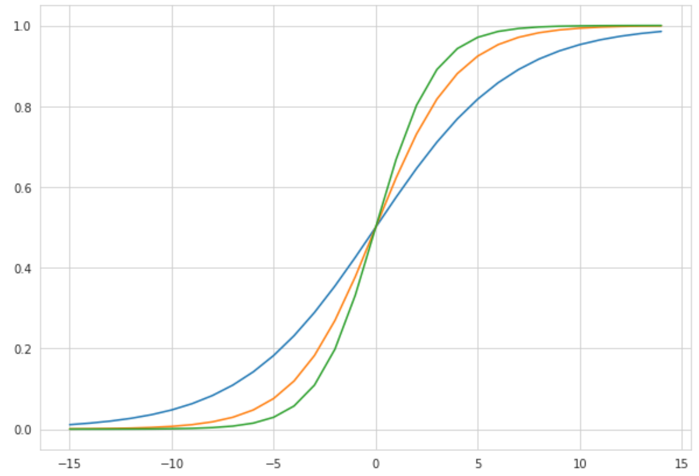
Weights & Biases (1)
(Weights)

- **Weight** decides how much **influence** the **input** will have on the **output**.

$$price\ of\ car = (w_1 x\ Year + w_2 x\ Miles\ )$$

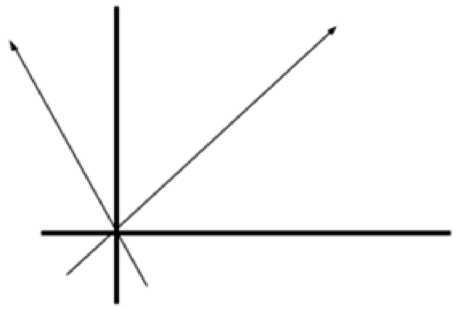
- **Weight** increases the **steepness** of activation function; this means weight decide how **fast** the activation function will **trigger**.
- **What do the weights in a Neuron convey to us?**
  - **Importance** of the feature
  - Tells the **relationship** between a particular feature in the dataset and the target value.

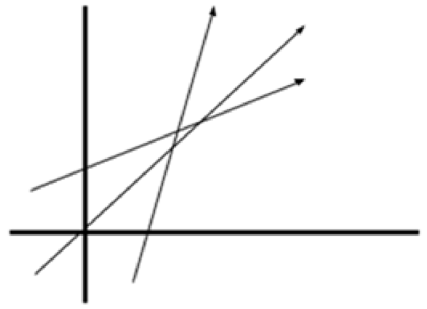


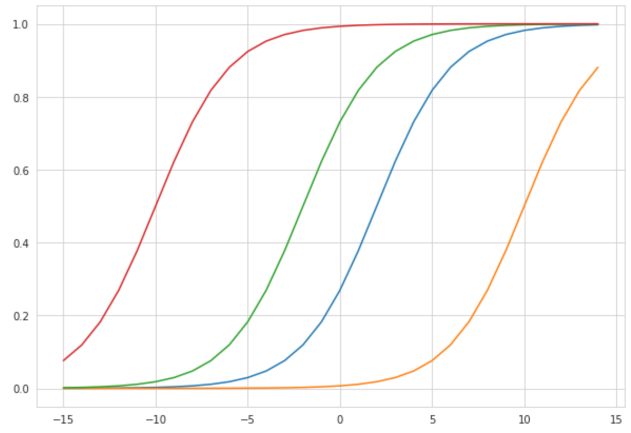


Weights & Biases (2) (Biases)

- Each neuron has a **bias**.
- **Bias** determines if a neuron is **activated** and how much, which **increases** the **model flexibility**.
- Due to **absence** of bias, model will train over point passing through origin only, which is not in accordance with real-world scenario.
- The bias is used to **shift** the result of activation function towards the **positive** or **negative** side.
- Also, **bias** is used to delay the triggering of the **activation function**.







Weights & Biases (3) (Summary)

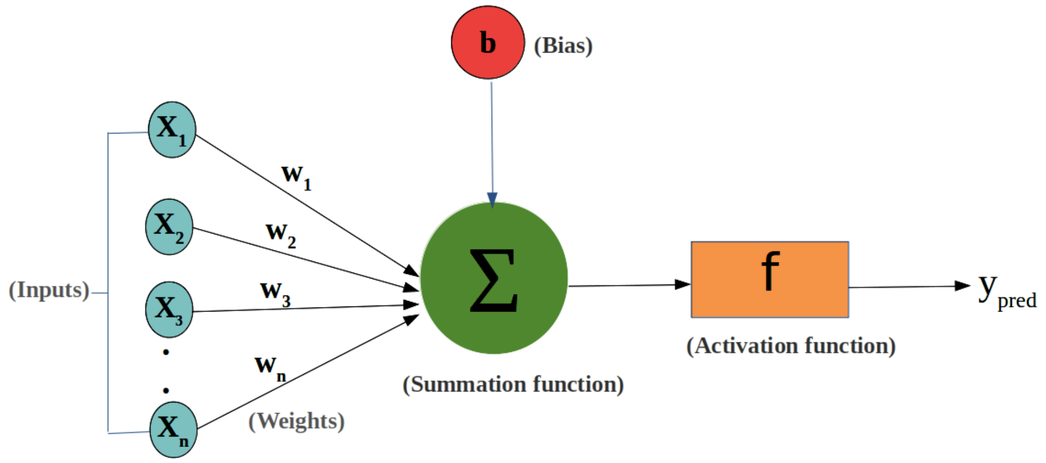
- **Weights** and **bias** are both learnable parameters inside the network.
- A teachable neural network will **randomize** both the weight and bias values **before** learning initially begins.
- As **training** continues, both parameters are **adjusted** toward the desired values and the correct output.
- The two parameters differ in the extent of their **influence** upon the input data.
- Simply, **bias** represents how far off the predictions are from their intended value. Biases make up the difference between the **function's** output and its **intended** output.
- A **low bias** suggest that the network is making **more assumptions** about the form of the output, whereas a **high bias** value makes **less assumptions** about the form of the output.
- **Weights**, on the other hand, can be thought of as the **strength of the connection**. Weight affects the amount of influence a change in the input will have upon the output.
- A **low weight** value will have **no change** on the input, and alternatively a **larger weight** value will **more significantly change** the output.

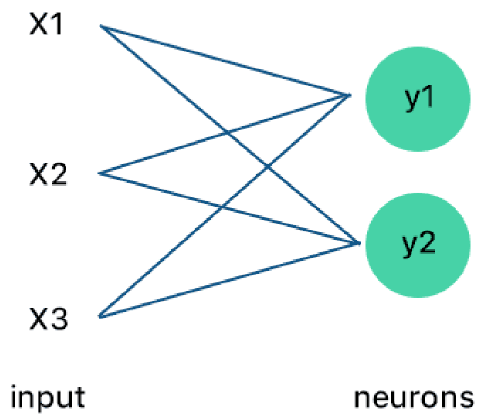
Activation Functions

“Activate yourself to duty by remembering your position, who you are, and what you have obliged yourself to be.” ~ Thomas a Kempis

Activation Functions (0)
(Overview)

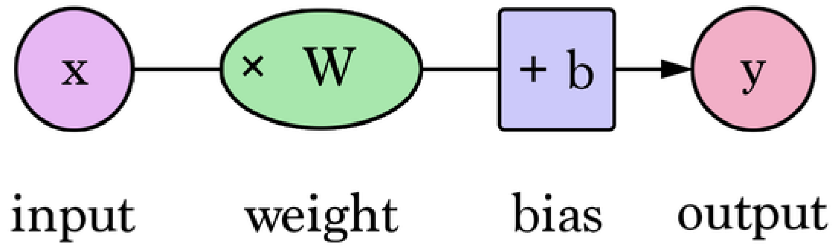
- An **Activation Function** decides whether a neuron should be **activated or not**.
- An **activation function** in a neural network defines how the weighted sum of the input is **transformed** into an output from a node or nodes in a layer of the network.
- The choice of activation function in the **hidden** layer will control how well the network model learns the **training** dataset. All hidden layers typically use the **same** activation function.
- The choice of activation function in the **output** layer will define the type of **predictions** the model can make.





Activation Functions (1) (Why?)

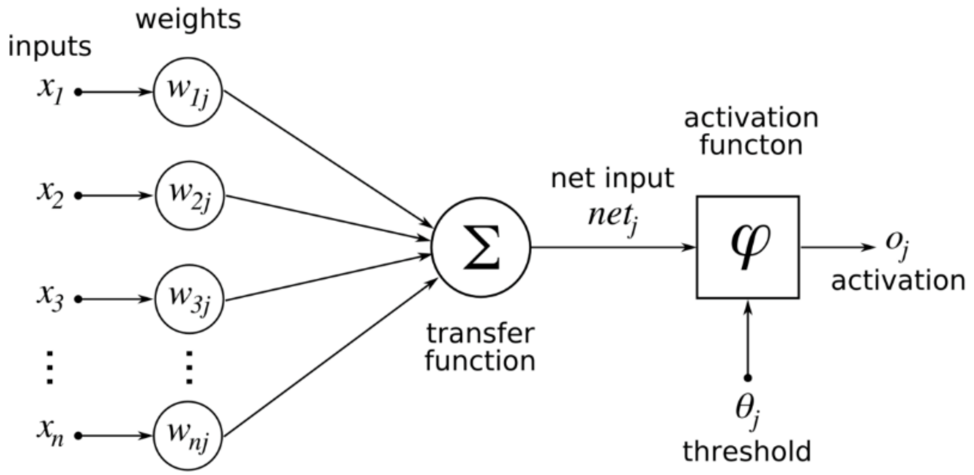
- Imagine a neural network **without** the **activation functions**. In that case, every neuron will only be performing a **linear transformation** on the inputs using the weights and biases.
- Although linear transformations make the neural network **simpler**, but this network would be **less powerful** and will not be able to learn the **complex** patterns from the data.
- A neural network without an activation function is essentially just a **linear regression model**.
- In addition to that, Activation functions are **differentiable** due to which they can easily implement **back-propagations** to measure gradient loss functions in the neural networks.

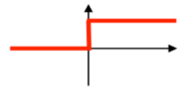
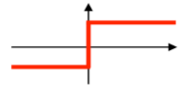
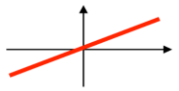
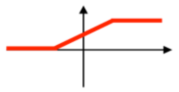
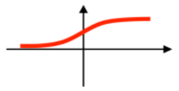
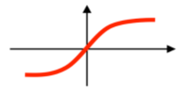
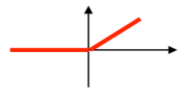
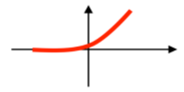


$$output = sum\left(weights * inputs\right) + bias$$

Activation Functions (2) (Types)

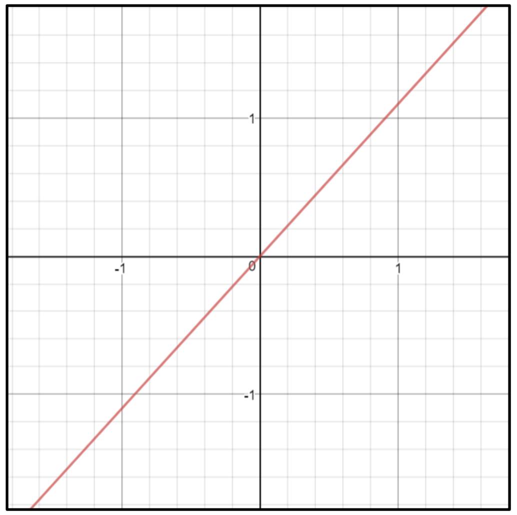
- An **activation function** in a neural network defines how the weighted sum of the **input** is transformed into an **output** from a node or nodes in a layer of the network.



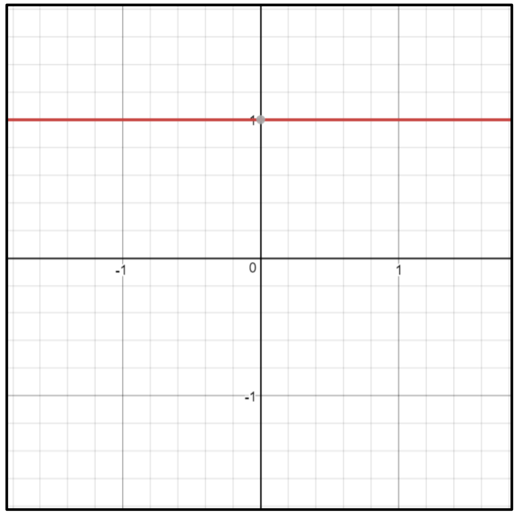
Activation function	Equation	Example	1D Graph
Unit step (Heaviside)	$\phi(z) = \begin{cases} 0, & z < 0, \\ 0.5, & z = 0, \\ 1, & z > 0, \end{cases}$	Perceptron variant	
Sign (Signum)	$\phi(z) = \begin{cases} -1, & z < 0, \\ 0, & z = 0, \\ 1, & z > 0, \end{cases}$	Perceptron variant	
Linear	$\phi(z) = z$	Adaline, linear regression	
Piece-wise linear	$\phi(z) = \begin{cases} 1, & z \geq \frac{1}{2}, \\ z + \frac{1}{2}, & -\frac{1}{2} < z < \frac{1}{2}, \\ 0, & z \leq -\frac{1}{2}, \end{cases}$	Support vector machine	
Logistic (sigmoid)	$\phi(z) = \frac{1}{1 + e^{-z}}$	Logistic regression, Multi-layer NN	
Hyperbolic tangent	$\phi(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$	Multi-layer Neural Networks	
Rectifier, ReLU (Rectified Linear Unit)	$\phi(z) = max(0, z)$	Multi-layer Neural Networks	
Rectifier, softplus Copyright © Sebastian Raschka 2016 (http://sebastianraschka.com)	$\phi(z) = \ln(1 + e^z)$	Multi-layer Neural Networks	

Activation Functions (3)
(Linear Function)

- A straight-line (**Linear**) function where activation is proportional to input (which is the weighted sum from neuron).
- The linear activation function is also called "**identity**" (multiplied by 1.0) or "**no activation**."
- For this function, derivative is a **constant**. That means, the gradient has **no relationship** with X.
- In this scenario, the neural network will not really improve the error since the gradient is the same for every iteration.
- The network will not be able to train well and capture the complex patterns from the data.



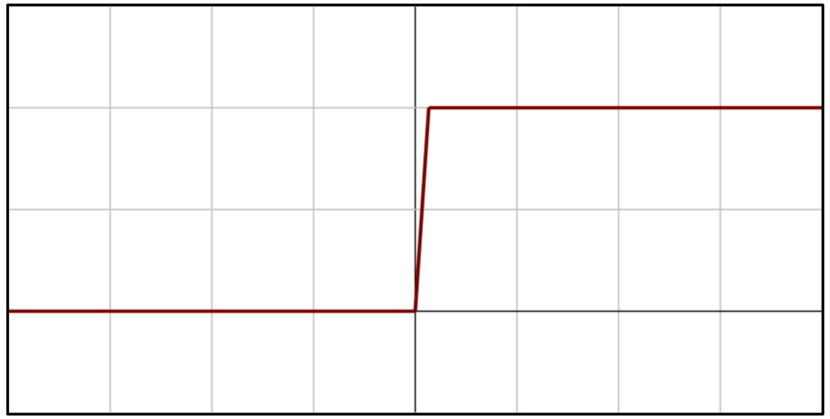
$R(z, m) = \{z * m\}$



$R'(z, m) = \{m\}$

Activation Functions (4)
(Binary Step Function)

- The value of a binary step function is **zero** for **negative** arguments and **one** for **positive** arguments.
- The **binary step function** is very basic, and it comes to mind every time if we try to bound output.
- It can be used as an activation function while creating a **binary classifier**.
- if the input to the activation function is greater than a **threshold**, then the neuron is **activated**, else it is **deactivated**.
- Moreover, the gradient of the step function is **zero** which causes a hindrance in the back propagation process.

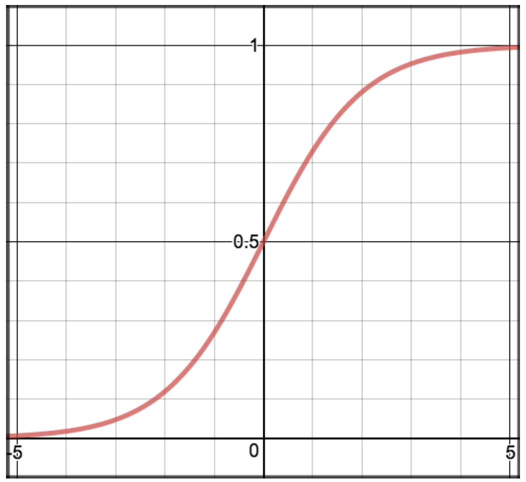


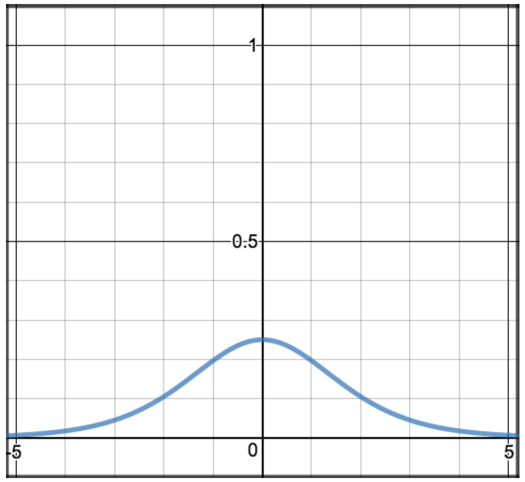
$$\begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases}$$

$$\begin{cases} 0 & \text{if } x \neq 0 \\ \text{undefined} & \text{if } x = 0 \end{cases}$$

Activation Functions (5)
(Sigmoid Function)

- **Sigmoid** takes a real value as input and outputs another value between **0** and **1**.
- 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 **0.0**.
- Its major **drawbacks** are **sharp damp gradients** during backpropagation, gradient saturation, slow convergence, and non-zero centered output thereby causing the gradient updates to propagate in different directions.



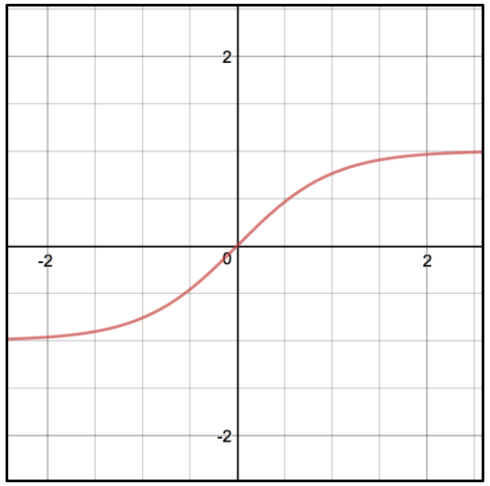


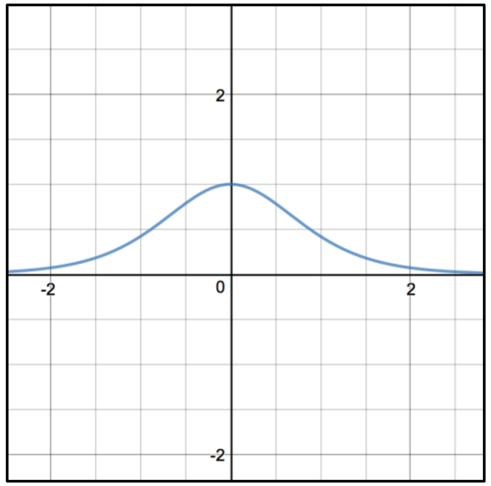
$$S(z) = \frac{1}{1 + e^{-z}}$$

$$S'(z) = S(z) \cdot (1 - S(z))$$

Activation Functions (6)
(Tanh Function)

- **Tanh** squashes a real-valued number to the range [-1, 1].
- It's non-linear, but unlike Sigmoid, its output is **zero-centered**. Therefore, in practice the tanh **non-linearity** is always preferred to the sigmoid nonlinearity.
- The gradient of the tanh function is **steeper** as compared to the sigmoid function. It also faces the problem of **vanishing gradients** like the sigmoid activation function



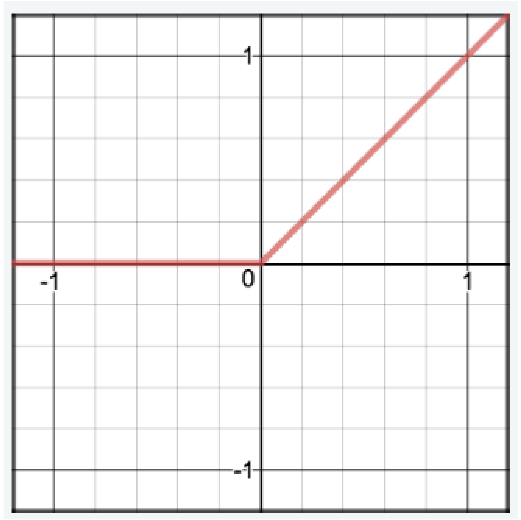


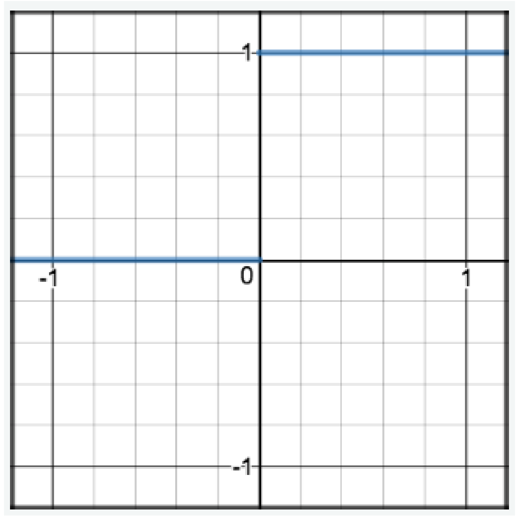
$$tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$$

$$tanh'(z) = 1 - tanh(z)^2$$

Activation Functions (7)
(ReLU Function)

- The **rectified linear Unit (ReLU)** activation function, is perhaps the most common function used for hidden layers.
- The main **advantage** of using the ReLU function over other activation functions is that it **does not activate** all the neurons at the same time. This means that the neurons will only be deactivated if the output of the linear transformation is less than 0.
- For the **negative** input values, the result is zero, that means the neuron does not get activated. Due to this reason, during the backpropagation process, the weights and biases for some neurons are not updated. This can create **dead neurons** which never get activated.



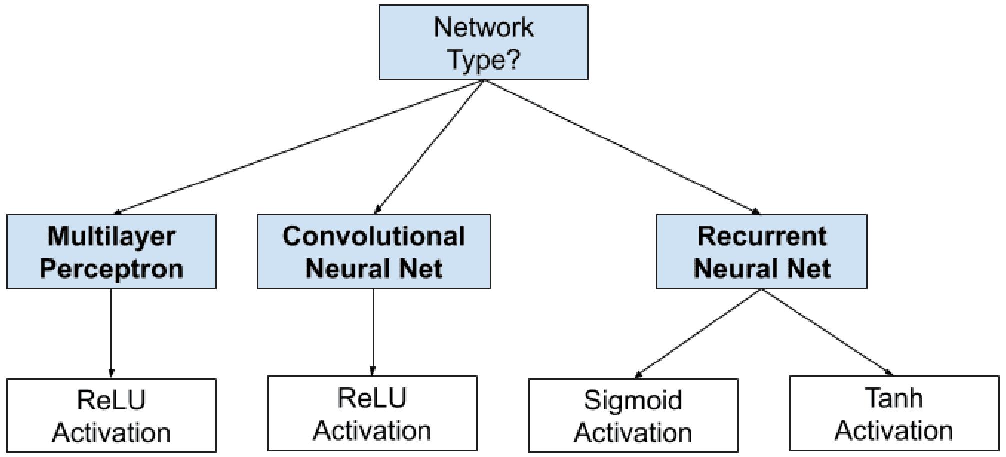


$$R'(z) = \begin{Bmatrix} 1 & z > 0 \\ 0 & z < 0 \end{Bmatrix}$$

$$R(z) = \begin{Bmatrix} z & z > 0 \\ 0 & z <= 0 \end{Bmatrix}$$

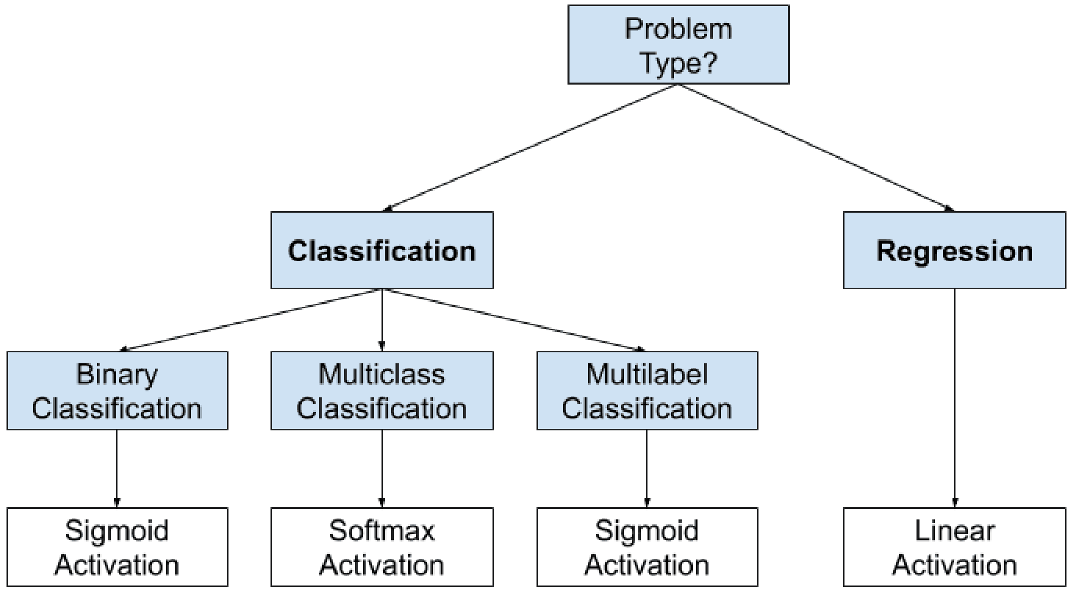
Activation Functions (8)
(Activation Functions for Hidden Layer(s))

- A neural network will almost always have the **same** activation function in all hidden layers.
- The **activation function** used in **hidden** layers is typically chosen based on the type of neural network **architecture**.
- **Multilayer Perceptron (MLP):**
  - **ReLU** activation function.
- **Convolutional Neural Network (CNN):**
  - **ReLU** activation function.
- **Recurrent Neural Network (RNN):**
  - **Tanh** and/or **Sigmoid** activation function.



Activation Functions (9)
(Activation Functions for Output Layer)

- You must choose the activation function for your output layer based on the **type of prediction** problem that you are solving.
- **Regression:**
  - One node, **linear** activation.
- **Binary Classification:**
  - One node, **sigmoid** activation.
- **Multiclass Classification:**
  - One node per class, **softmax** activation.
- **Multilabel Classification:**
  - One node per class, **sigmoid** activation.



Activation Functions (10) (Summary)

- **Activation Functions** are used to introduce **non-linearity** in the network.
- A neural network will almost always have the **same** activation function in all **hidden** layers. This activation function should be **differentiable** so that the parameters of the network are learned in **backpropagation**.
- **ReLU** is the most used activation function for hidden layers.
- While selecting an activation function, you must consider the **problems** it might face **vanishing and exploding gradients.**
- If we encounter a case of **dead neurons** in our networks the **Leaky ReLU** function is the best choice.
- Regarding the output layer, we must always consider the expected value range of the predictions. If it can be any numeric value (as in case of the regression problem) you can use the linear activation function or ReLU.
- Use **Softmax** or Sigmoid function for the **classification** problems.
- As a rule of thumb, you can **begin** with using **ReLU** function and then move over to other activation functions in case ReLU doesn't provide with optimum results

Further Readings

- Deep Learning Illustrated, Jon Krohn
  - Chapter 6
- Deep Learning with Python, François Chollet
  - Chapters 2, 3

تقــني
معلومة مفيدة .. وقت قليل

THANKS

Keep Moving Forward! ☺

Eng. Mustafa Othman
Data Scientist & Analyst