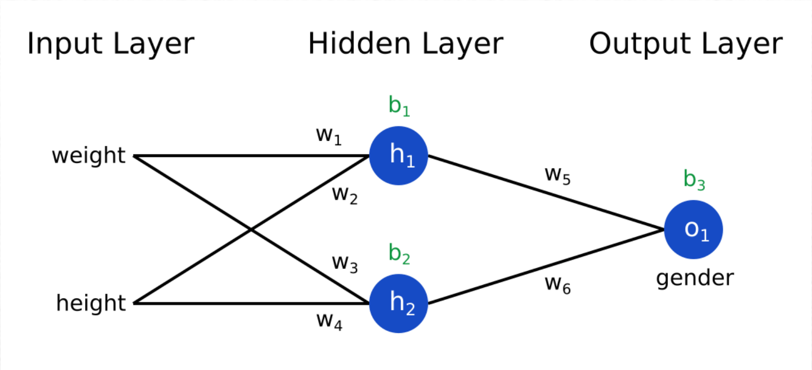
Activation Functions

An Activation Function decides whether a **neuron** should be activated or not. Means decide whether the neuron's input is important or not in the process of prediction.



$$y = \sum_{i=1}^{n} w_i \,.\, x_i + b_i$$

$$a_1 = W_1 weight + W_2 height + b_1$$

$$h_1 = \sigma(a_1)$$

Oversimplified, an activation function is any function that can take one number and return another number.

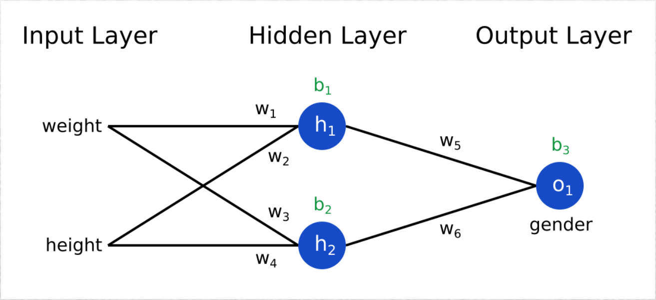
Why We Need an Activation Function?

The purpose of an activation function is to add non-linearity to the neural network.

If there is no activation function:

every neuron will only be performing a linear transformation on the inputs using the weights and biases.

$$y = \sum_{i=1}^{n} w_i \,.\, x_i + b_i$$

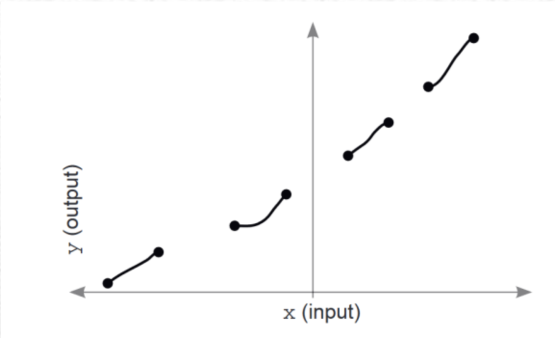


So, all layers will behave in the same way

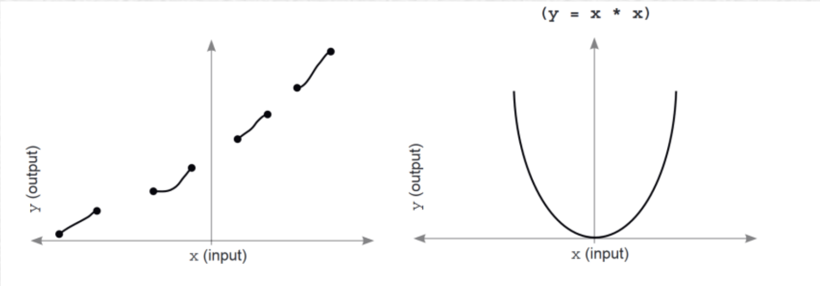
Then, learning any complex task is impossible, and our model would be just a linear regression model.

A Good Activation Function

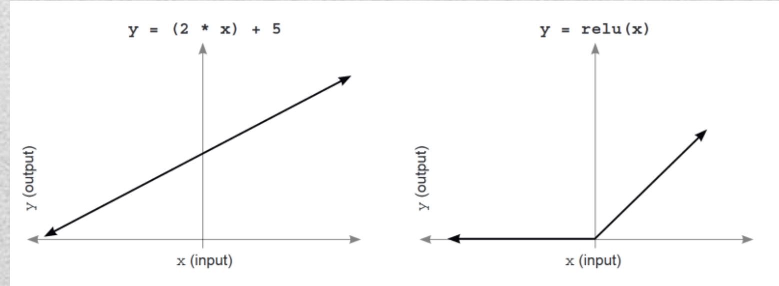
The function must be continuous and infinite in domain



Never changing direction.



Nonlinear



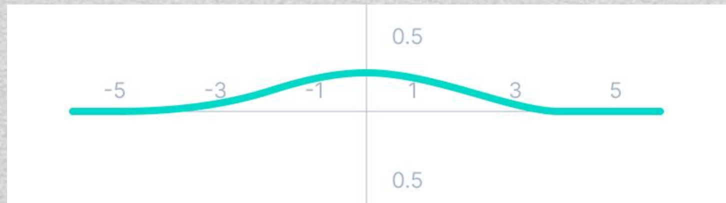
Non-Linear Activation Functions

Sigmoid/Logistic function: takes any real value as input
and outputs values in the range of [0 1].

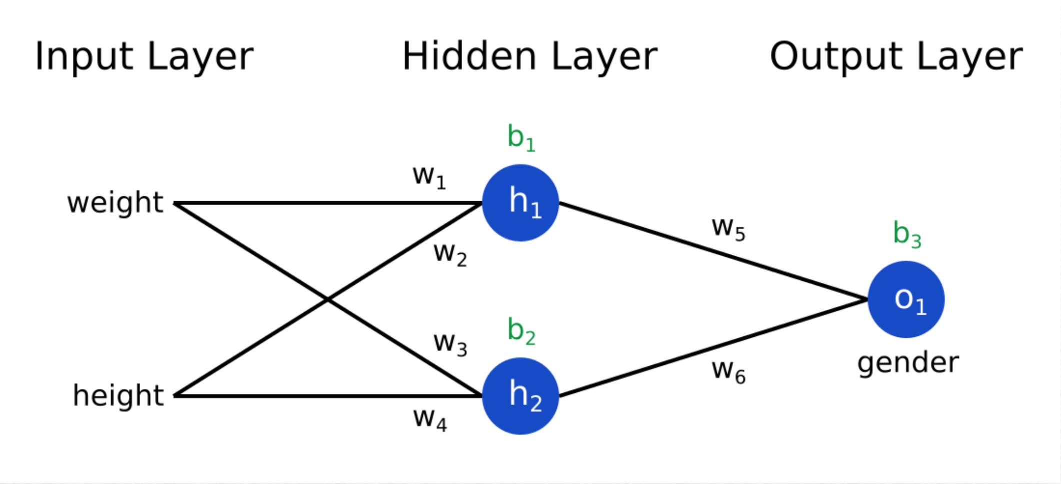
$$f(x) = \frac{1}{1 + e^{-x}}$$

Pro:
- to predict the probability as an output.
- The function is **differentiable.**

Cons:
- The derivative of the function is f'(x) = sigmoid(x)*(1-sigmoid(x)).
- As the gradient value approaches zero, the network ceases to learn and suffers from the *Vanishing gradient* problem



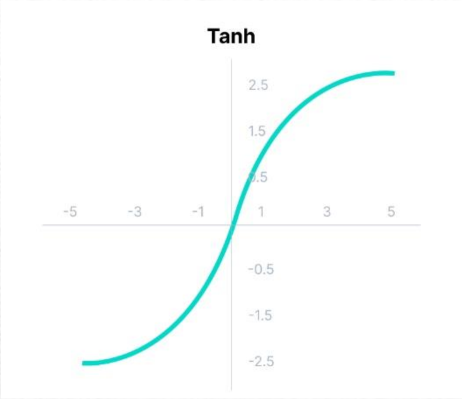
Weight	Height	Class
100	180	Man
70	190	Man
50	160	Woman



$$y = \sum_{i=1}^{n} w_i \, . \, x_i + b_i$$

$$h_1 = \sigma(W_1 weight + W_2 height + b_1)$$
$$= \sigma(0.2 * 100 + 0.1 * 180 + .6)$$
$$= \sigma(20 + 18 + 0.6)$$
$$= \sigma(38.6)$$
$$= \frac{1}{1 + e^{-38.6}}$$
$$h_1 \approx 0.269$$

$$f(x) = \frac{1}{1 + e^{-x}}$$



Tanh (Hyperbolic Tangent) function:
takes any real value as input and outputs values in the range of [-1 1].

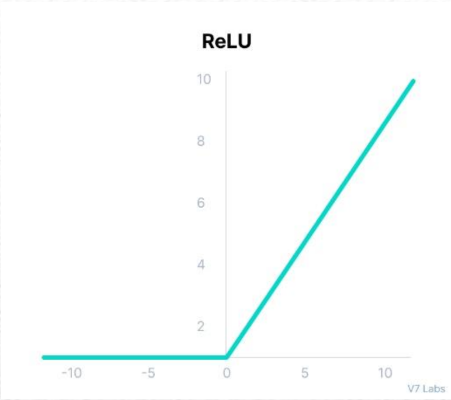
$$f(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

Pro:

- Zero centered; so map the output values as strongly negative, neutral, or strongly positive.
- Usually used in hidden layers.
- It helps in centering the data and makes learning for the next layer much easier.

Cons:

- As the gradient value approaches zero, but better than Sigmod



ReLU Function→ Rectified Linear Unit**:** has a derivative function and allows for backpropagation while simultaneously making it computationally efficient.

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

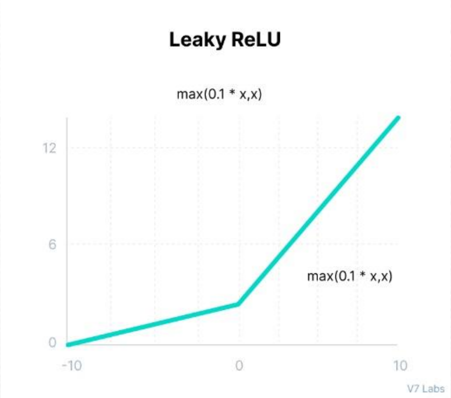
Pro:

- more computationally efficient
- accelerates the convergence of gradient descent towards the global minimum of the loss function.

Cons:

- All the negative input values become zero immediately, which decreases the model's ability to fit the data properly.

Leaky ReLU Function: solve the ReLU problem as it has a small positive slope in the negative area



$$f(x) = \max(0.1 * x, x)$$

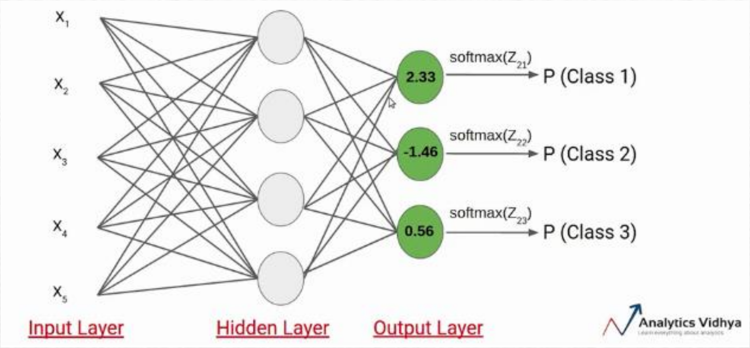
Pro:

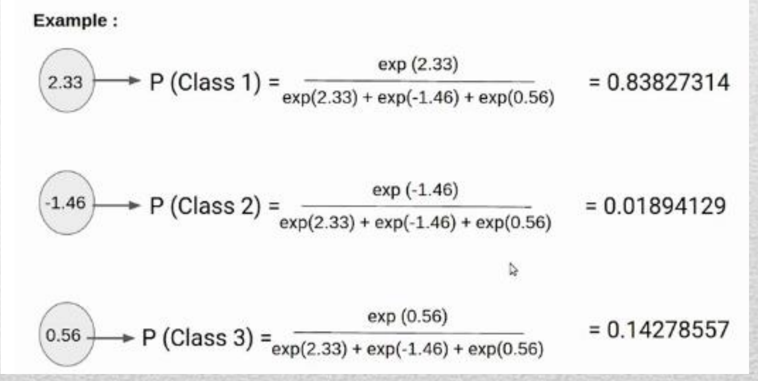
- Leaky ReLU are same as that of ReLU, in addition to the fact that it does enable backpropagation, even for negative input values.
- no longer encounter dead neurons

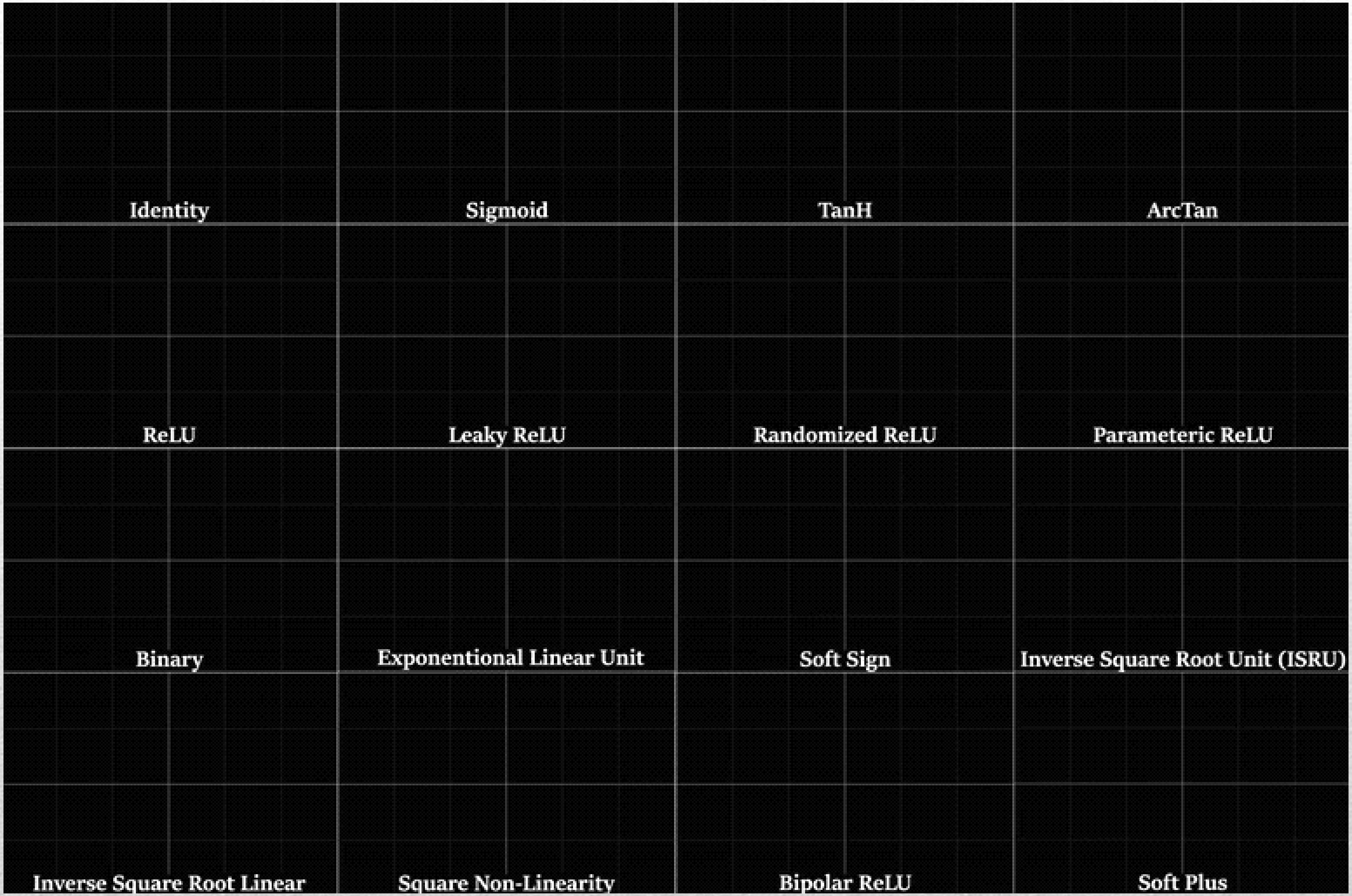
Softmax Function:

It calculates the relative probabilities. Similar to the sigmoid/logistic activation function, the SoftMax function returns the probability of each class.

$$\text{Softmax}(z_i) = \frac{\exp(z_i)}{\sum_j \exp(z_i)}$$






Identity
Sigmoid
TanH
ArcTan
ReLU
Leaky ReLU
Randomized ReLU
Parameteric ReLU
Binary
Exponentional Linear Unit
Soft Sign
Inverse Square Root Unit (ISRU)
Inverse Square Root Linear
Square Non-Linearity
Bipolar ReLU
Soft Plus

Problem Type	Output Type	Final Activation Function
Regression	Numerical value	Linear
Classification	Binary outcome	Sigmoid
Classification	Single label, multiple classes	Softmax