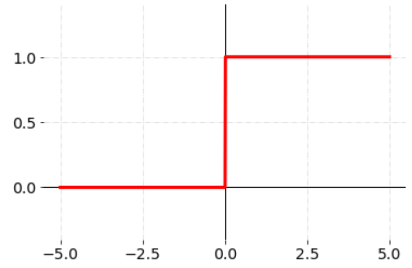
M-P model

- Not Automated
- Parameters computed manually.
- Data should be linearly separable.

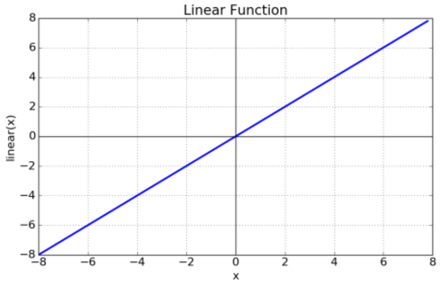
Perceptron

- Automated
- Perceptron Learning rule
- Data should be linearly separable
- Small margins (decision boundary)
- Used step activation function

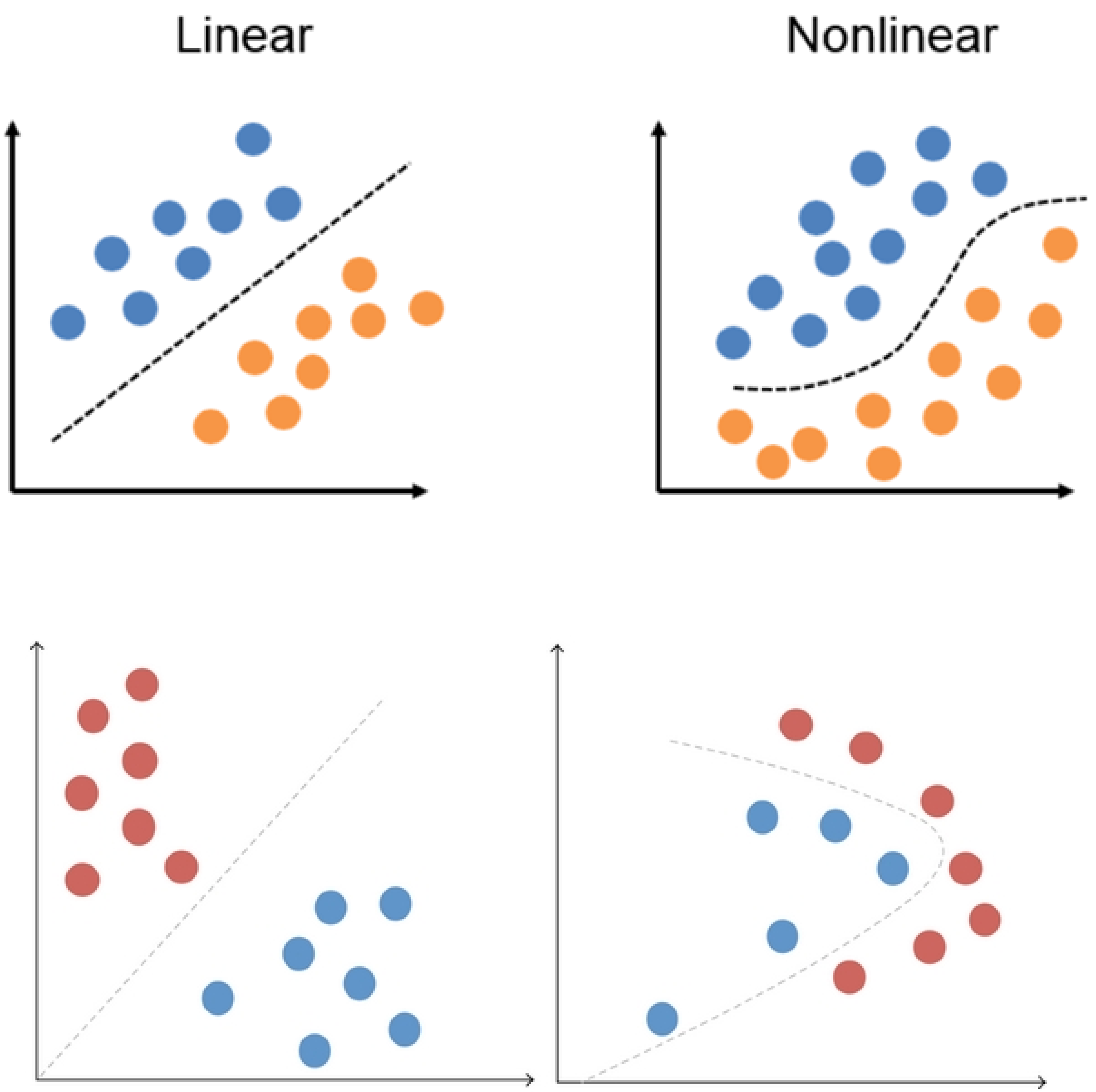


Adaline

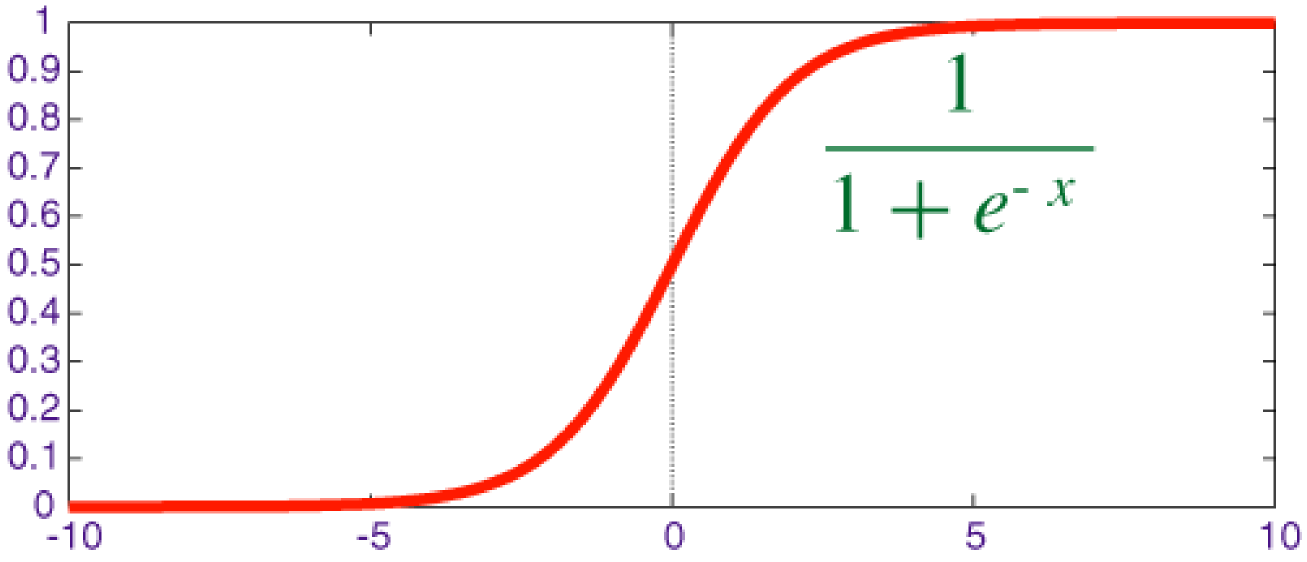
- Automated
- Parameter computing – using gradient
- Data should be linearly separable.
- Large margins (decision boundary)
- Linear activation function (identity function)
- Loss function : MSE
- Data should be linearly separable
- Output : not bounded [-inf to +inf]



Nonlinear activation functions are preferred as they allow the nodes to learn more complex structures in the data.



The linear function alone doesn't capture complex patterns



Derivative of Sigmoid function

$$y = \frac{1}{1+e^{-x}}$$

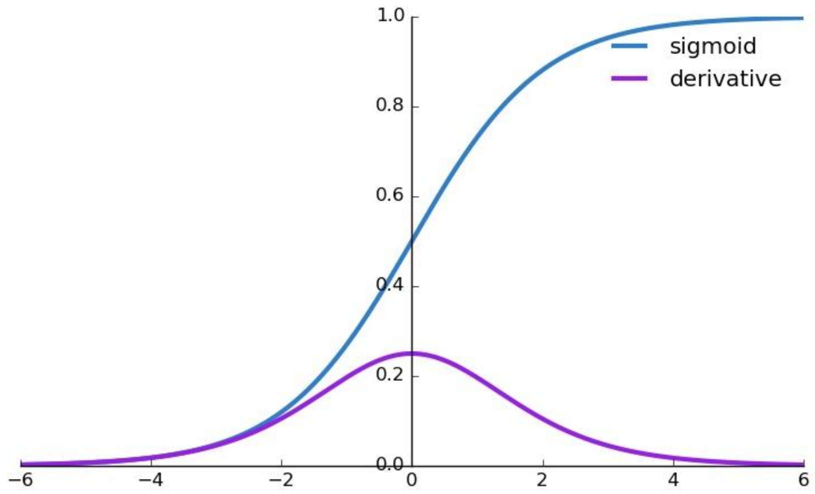
$$\frac{dy}{dx} = -\frac{1}{(1+e^{-x})^2}(-e^{-x}) = \frac{e^{-x}}{(1+e^{-x})^2}$$

$$= \frac{1}{1+e^{-x}}\left(1-\frac{1}{1+e^{-x}}\right) = y(1-y)$$

A **sigmoid function** is a mathematical function having a characteristic "S"-shaped curve or **sigmoid curve.**

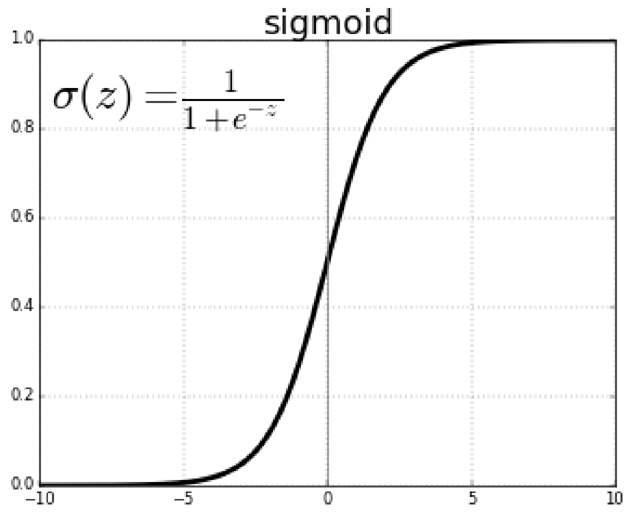
A sigmoid function is a bounded, differentiable, real function that is defined for all real input values and has a non-negative derivative at each point.

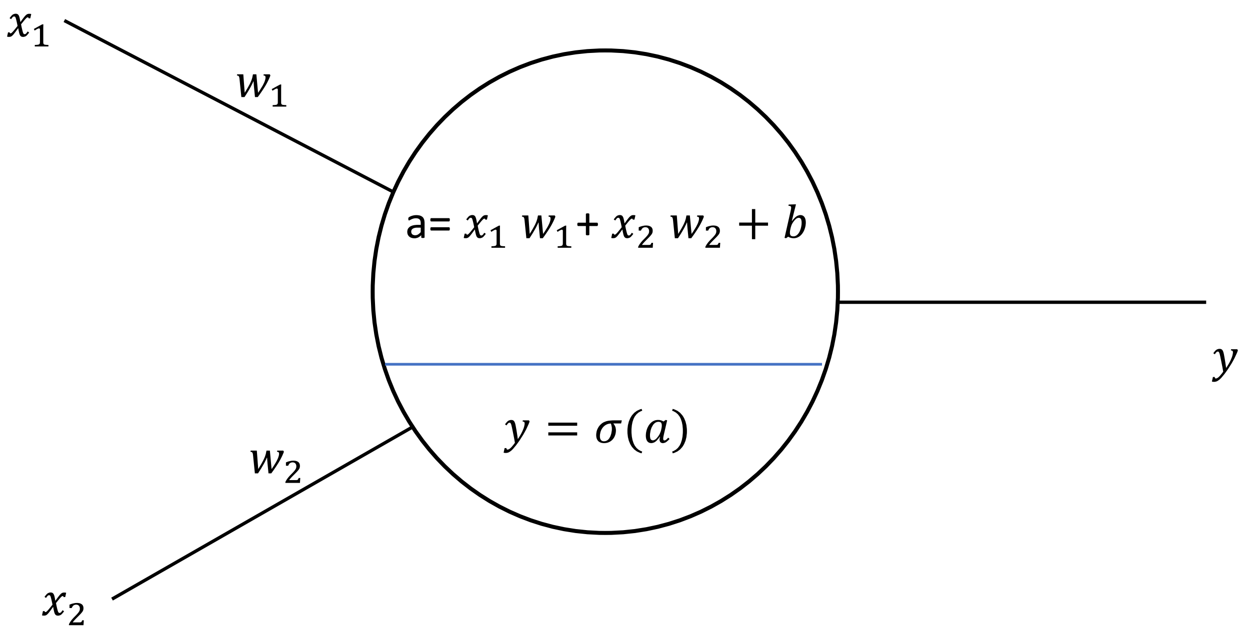
Without a ***non-linear*** activation function in the network, a NN, no matter how many layers it had, would behave just like a single-layer perceptron, because summing these layers would give you just another linear function.



https://en.wikipedia.org/wiki/Sigmoid_function
https://stackoverflow.com/questions/9782071/why-must-a-nonlinear-activation-function-be-used-in-a-backpropagation-neural-net

Sigmoid activation function





$$E = \frac{1}{2}(t - y)^2$$

$$y = \sigma(a)$$

$$\sigma(a) = \frac{1}{1+e^{-a}}$$

$$w_1 = w_1 - \alpha \frac{\partial E}{\partial w1}$$

$$\frac{\partial E}{\partial w1} = \frac{\partial E}{\partial y}\frac{\partial y}{\partial w1}$$

$$\frac{\partial E}{\partial w1} = \frac{\partial E}{\partial y}\frac{\partial y}{\partial a}\frac{\partial a}{\partial w1}$$

$$\frac{\partial E}{\partial y} = \frac{\partial(\frac{1}{2}(t-y)^2)}{\partial y} = -(t - y)$$

$$\frac{\partial y}{\partial a} = \frac{\partial \sigma(a)}{\partial a} = \sigma(a)\,(1 - \sigma(a)) = y\,(1 - y)$$

$$\frac{\partial a}{\partial w1} = \frac{\partial(x_1 w_1 + x_2\, w_2 + b)}{\partial w1} = x_1$$

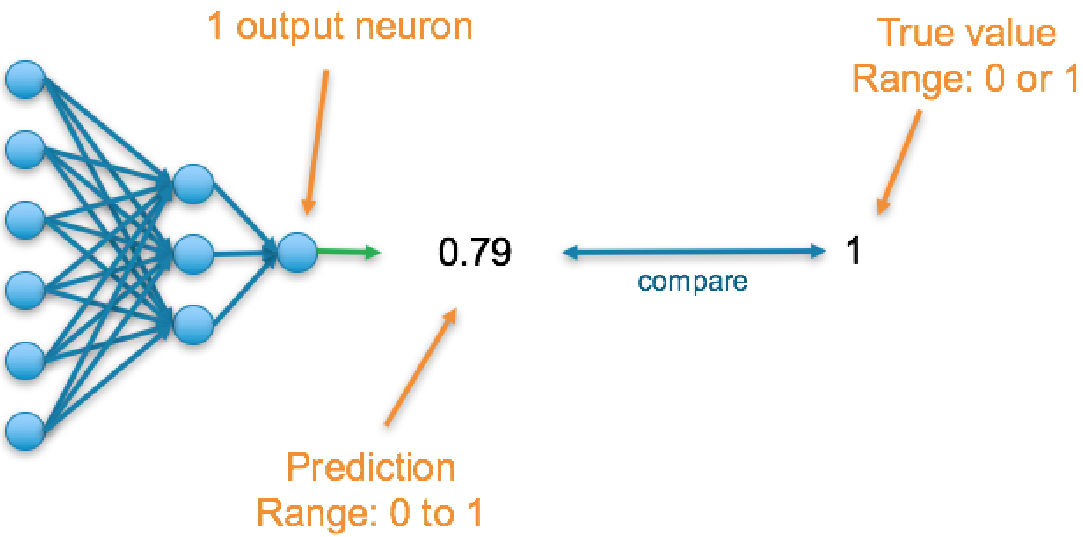
$$\frac{\partial E}{\partial w1} = -(t - y)y\,(1 - y)\,x_1$$

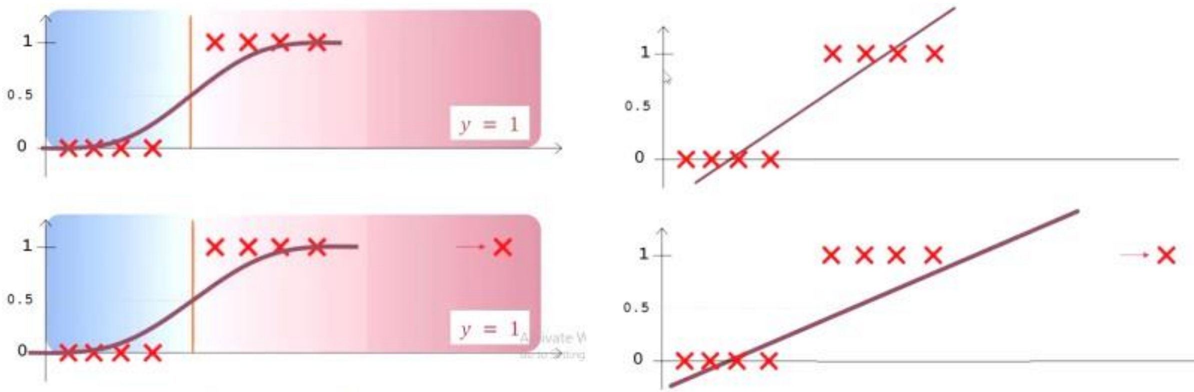
$$w_1 = w_1 + \alpha\,(t - y)y\,(1 - y)\,x_1$$

$$w_2 = w_2 + \alpha\,(t - y)y\,(1 - y)\,x_2$$

$$b = b + \alpha\,(t - y)y\,(1 - y)$$

$$w_i = w_i + \alpha\,(t - y)y\,(1 - y)x_i$$





with sigmoid

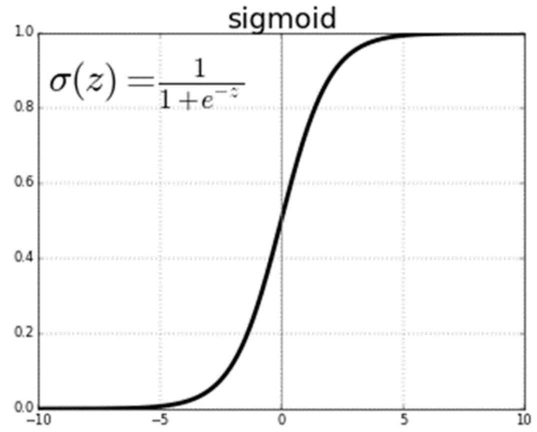
without sigmoid

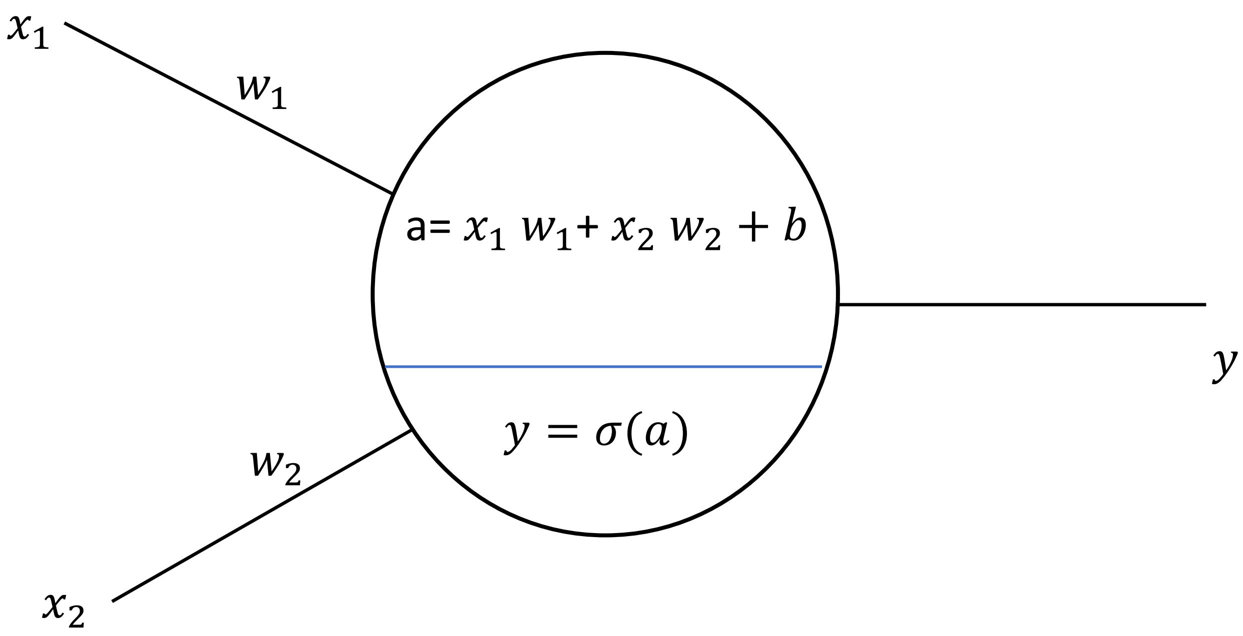
- For **binary classification** , the **typical loss function** is the **binary cross-entropy.**
- Using MSE means that we assume that the underlying data has been generated from a **normal distribution**.
- A dataset that can be classified into two categories (i.e binary) is not from a normal distribution but a **Bernoulli distribution**.

Binary cross entropy = $-(y\log(\hat{y}) + (1-y)\log(1-\hat{y}))$
Where $\hat{y}$ is the predicted value and y is the true value

picture Link : https://pasteboard.co/IgLjcYN.jpg

Sigmoid activation function





$$E = -(t \log(y) + (1-t)\log(1-y))$$

$$y = \sigma(a)$$

$$\sigma(a) = \frac{1}{1+e^{-a}}$$

$$w_1 = w_1 - \alpha \frac{\partial E}{\partial w1}$$

$$\frac{\partial E}{\partial w1} = \frac{\partial E}{\partial y}\frac{\partial y}{\partial w1}$$

$$\frac{\partial E}{\partial w1} = \frac{\partial E}{\partial y}\frac{\partial y}{\partial a}\frac{\partial a}{\partial w1}$$

$$\frac{\partial E}{\partial y} = -\frac{\partial(t\log(y)+(1-t)\log(1-y))}{\partial y}$$

$$\frac{\partial E}{\partial y} = -\left(\frac{t}{y} - \frac{(1-t)}{(1-y)}\right) = \frac{y-t}{y(1-y)}$$

$$\frac{\partial y}{\partial a} = \frac{\partial \sigma(a)}{\partial a} = \sigma(a)\,(1 - \sigma(a)) = y\,(1 - y)$$

$$\frac{\partial a}{\partial w1} = \frac{\partial(x_1 w_1 + x_2\, w_2 + b)}{\partial w1} = x_1$$

$$\frac{\partial E}{\partial w1} = (y-t)x_1$$

$$w_1 = w_1 + \alpha\,(t-y)x_1$$

$$w_2 = w_2 + \alpha\,(t-y)x_2$$

$$b = b + \alpha\,(t-y)$$

$$w_i = w_i + \alpha\,(t-y)x_i$$

Sigmoid

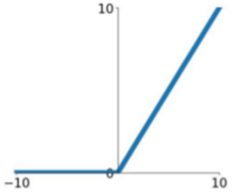
$$\sigma(x) = \frac{1}{1+e^{-x}}$$

tanh

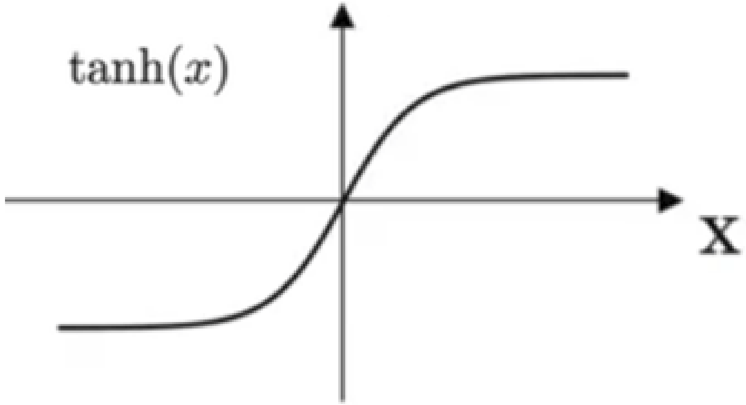
$$\tanh(x) \quad \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

ReLU

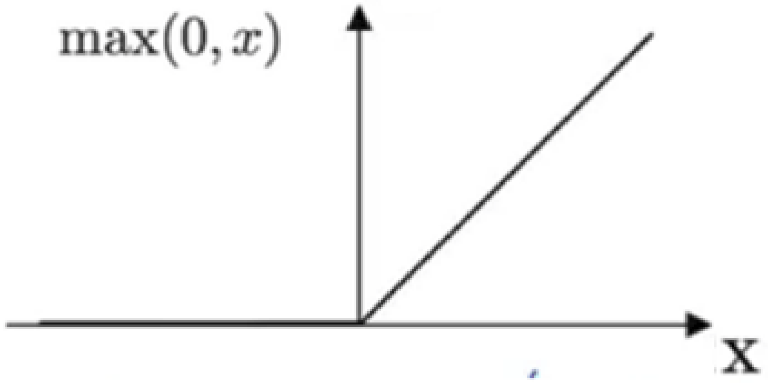
$$\max(0, x)$$



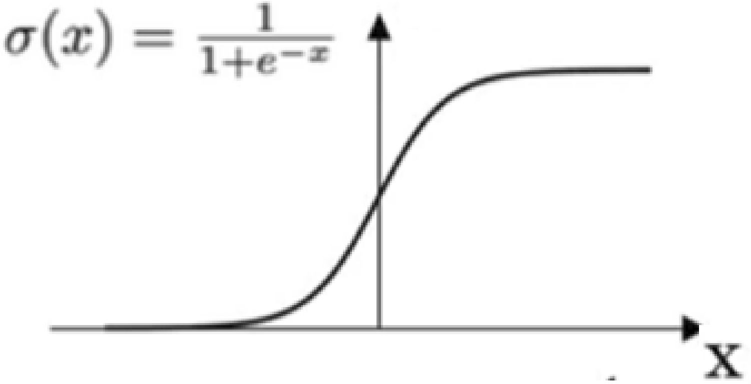
Hyper Tangent Function



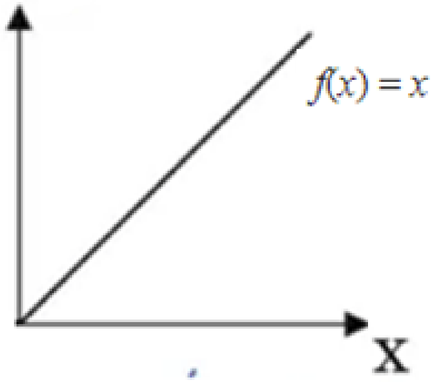
ReLU Function

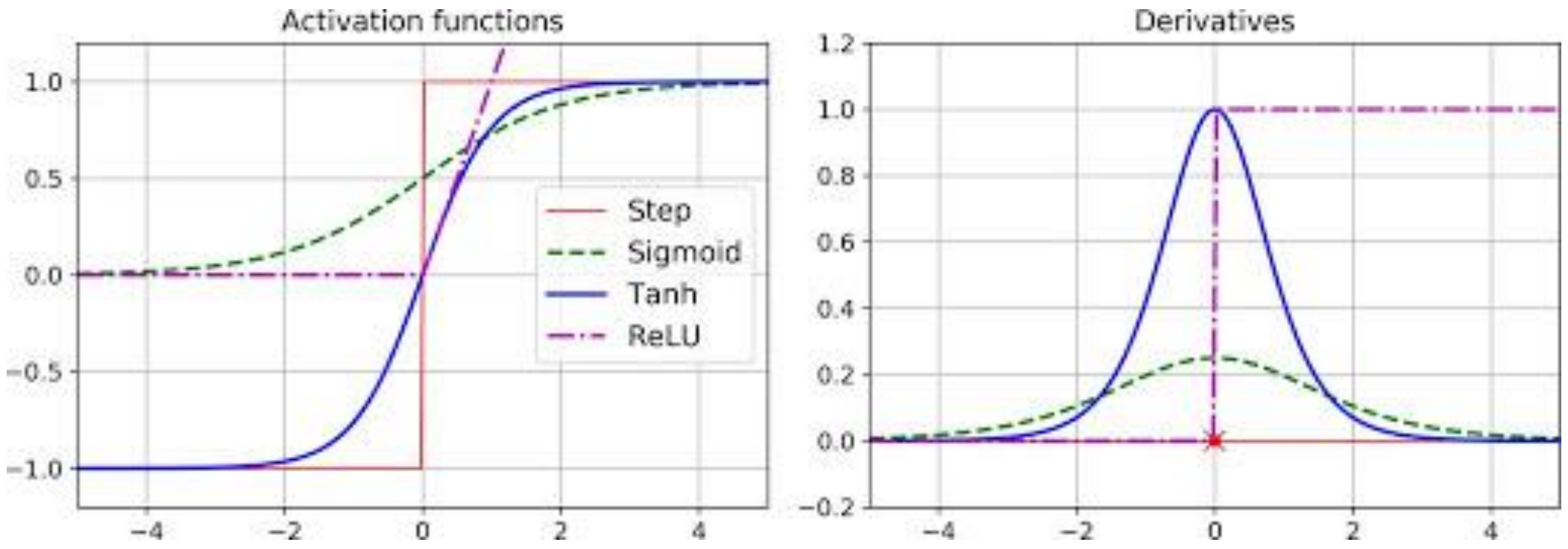


Sigmoid Function



Identity Function





- Convergence is usually faster if the average of each input variable over the training set is close to zero.

- When all of the components of an input vector are positive, all of the updates of weights that feed into a node will be the same sign

- The main advantage provided by the tanh function is that it produces zero centered output.

- ReLU activation function was first introduced to a dynamical network by Hahnloser et al. in 2000 with strong biological motivations and mathematical justifications.

- It has been demonstrated for the first time in 2011 to enable better training of deeper networks, compared to the widely used activation functions prior to 2011.

- It’s sparsely activated.

- Adoption of ReLU may easily be considered one of the few milestones in the deep learning revolution.