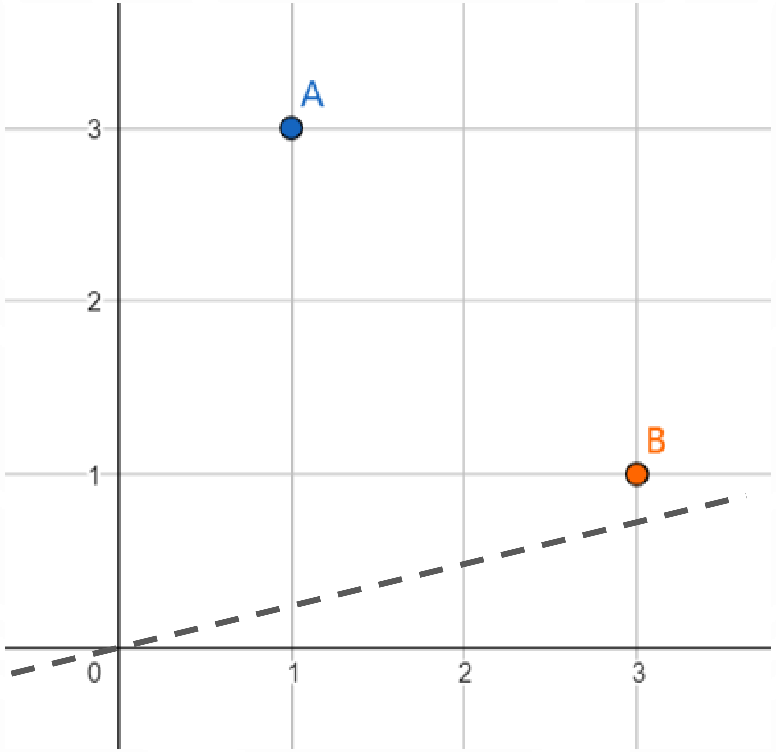
MUDSS
Data Science Society

Core Workshop 8: Artificial Neural Network

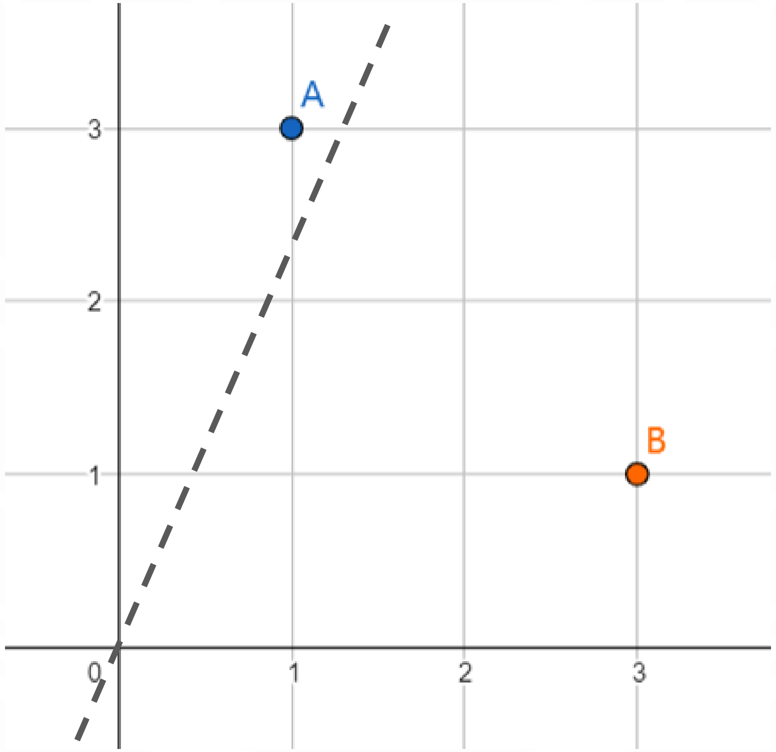
OVERVIEW

- A Simple Linear Classifier
  - One Classifier Is Not Enough
  - Neuron & Activation Function
- Feedforward Signal & Matrix Multiplication
- Backpropagation of Loss
  - Matrix Multiplication Again
  - Update Weights by Gradient Descent
- Input / Output & Initial Weights
- NumPy Code Example

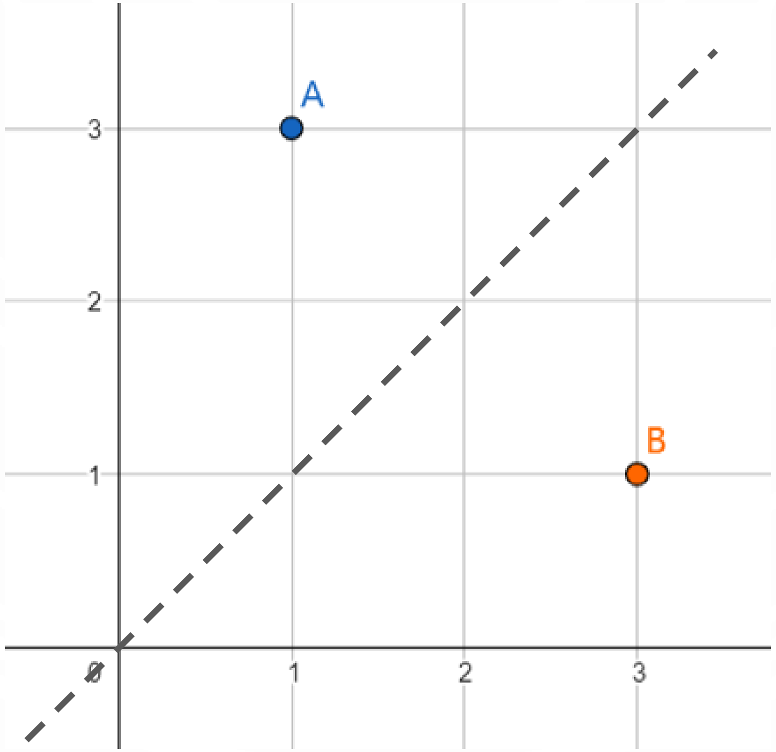
A SIMPLE LINEAR CLASSIFIER



A SIMPLE LINEAR CLASSIFIER

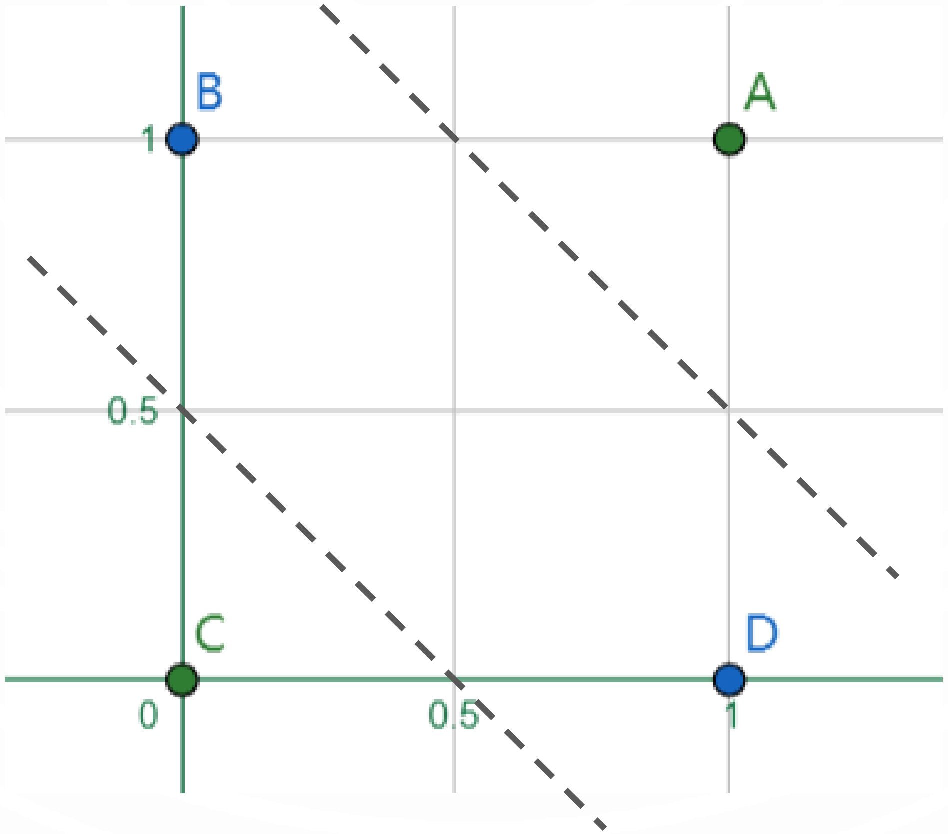


A SIMPLE LINEAR CLASSIFIER

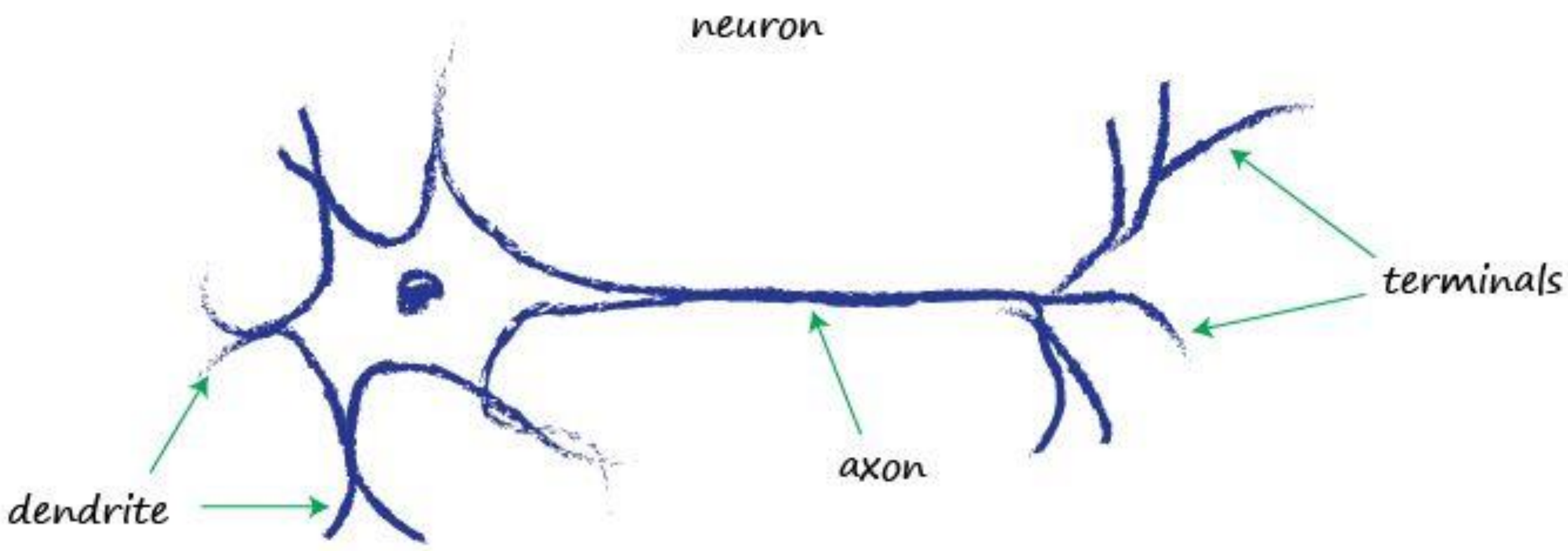


Linear Perceptron

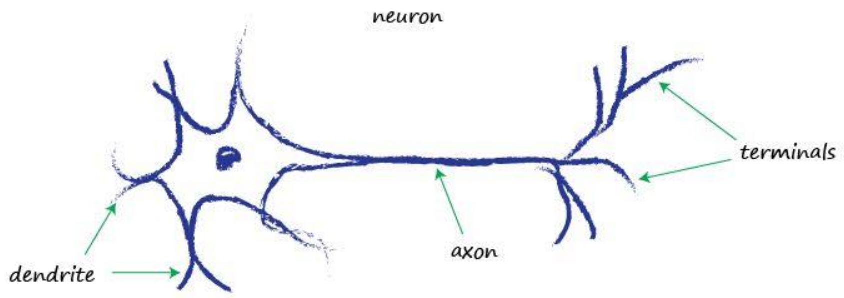
ONE CLASSIFIER IS NOT ENOUGH



NEURONS, NATURAL COMPUTING MACHINE

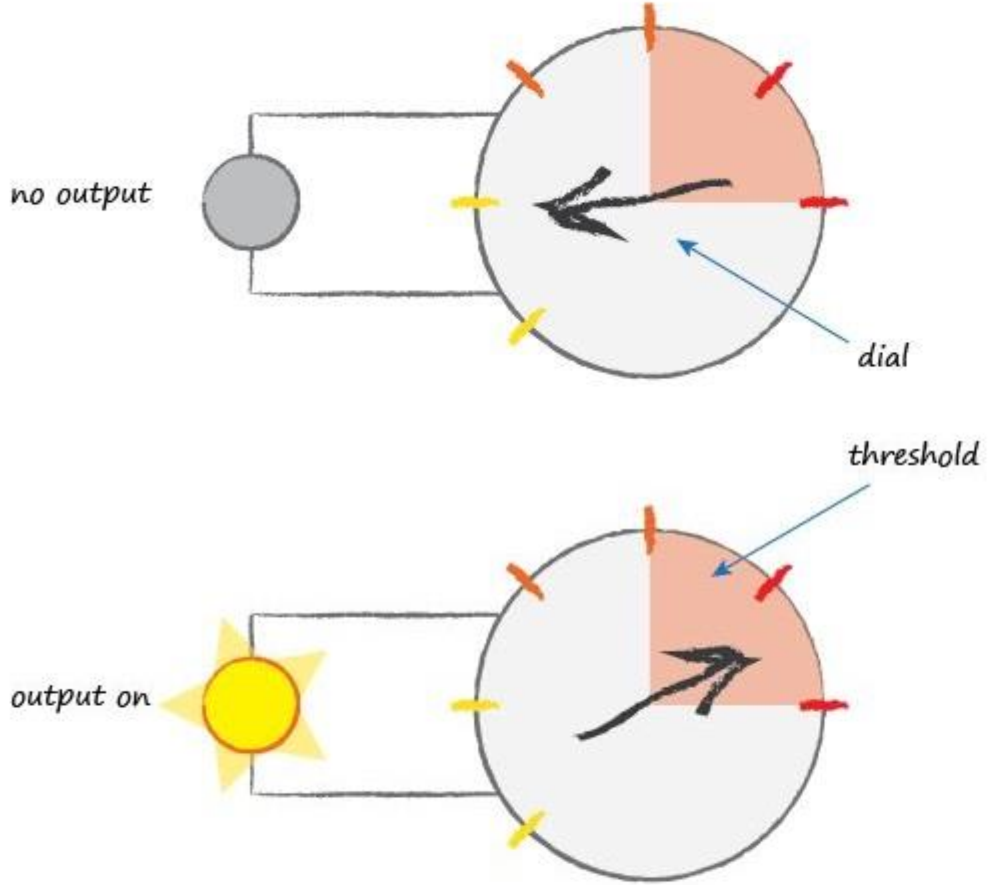


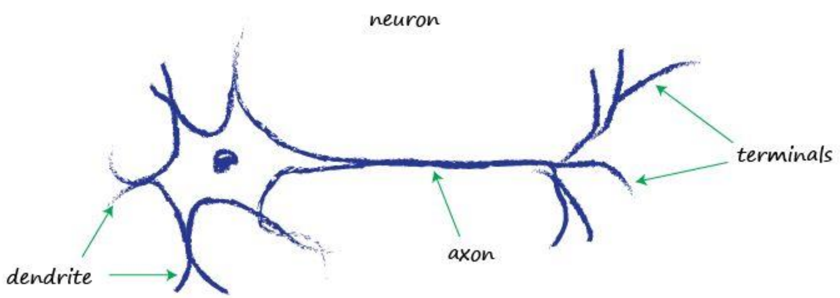
NEURONS, NATURAL COMPUTING MACHINE



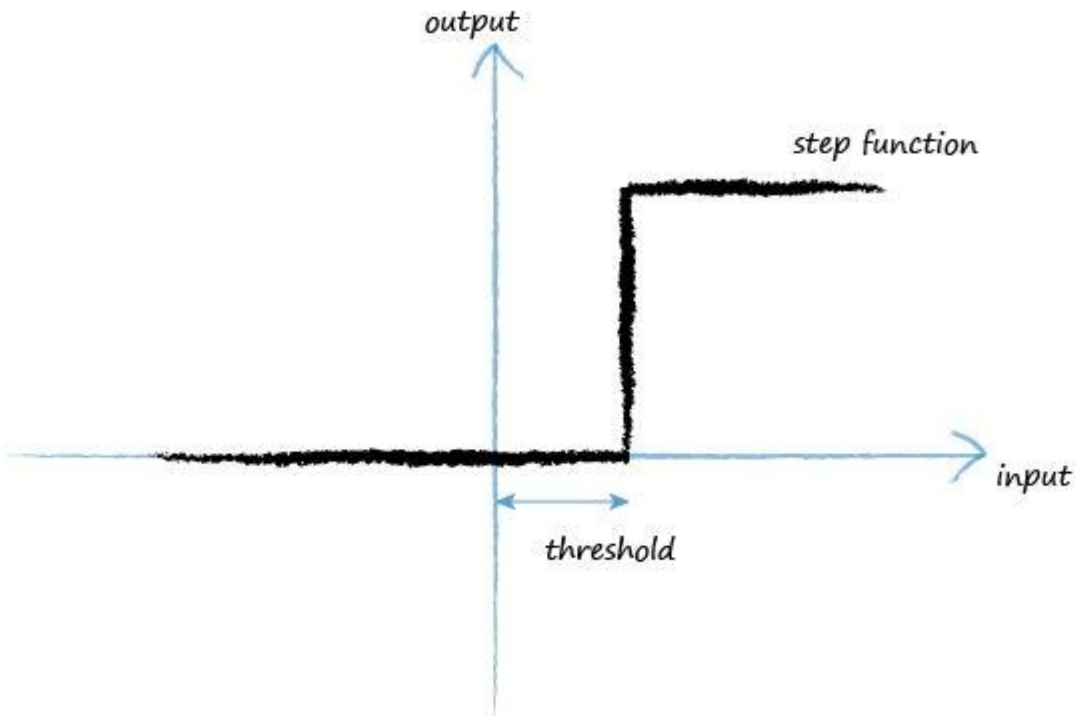
Human Brain	100 billion
Fruit Fly	100,000
Nematode Worm	302

NEURONS, NATURAL COMPUTING MACHINE

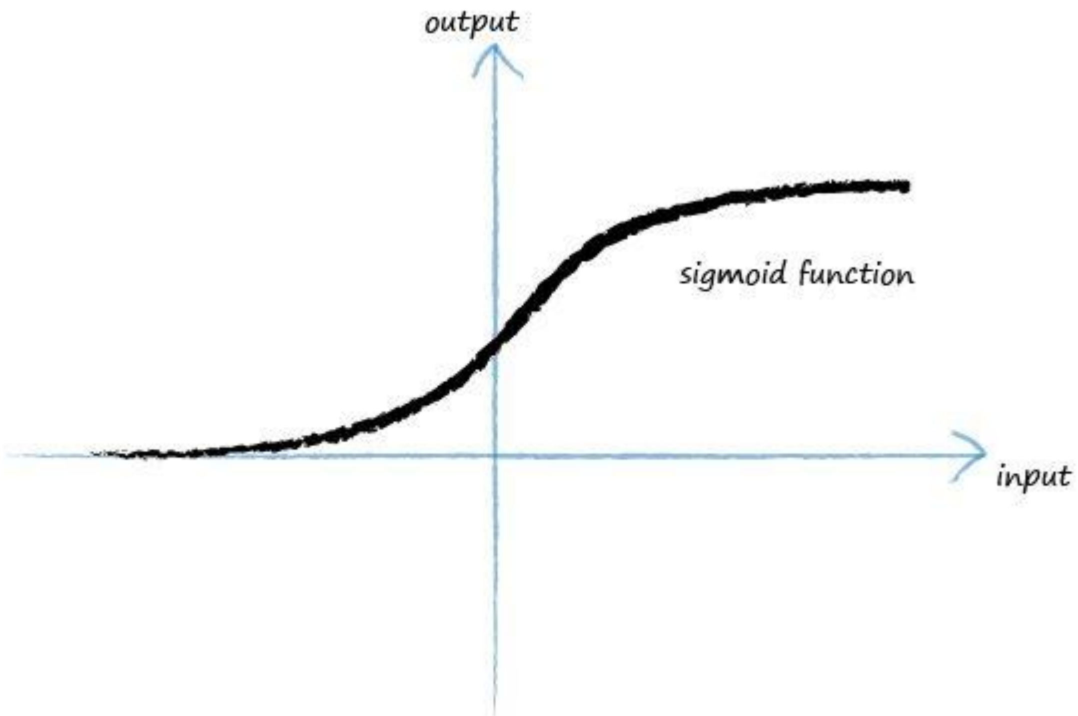




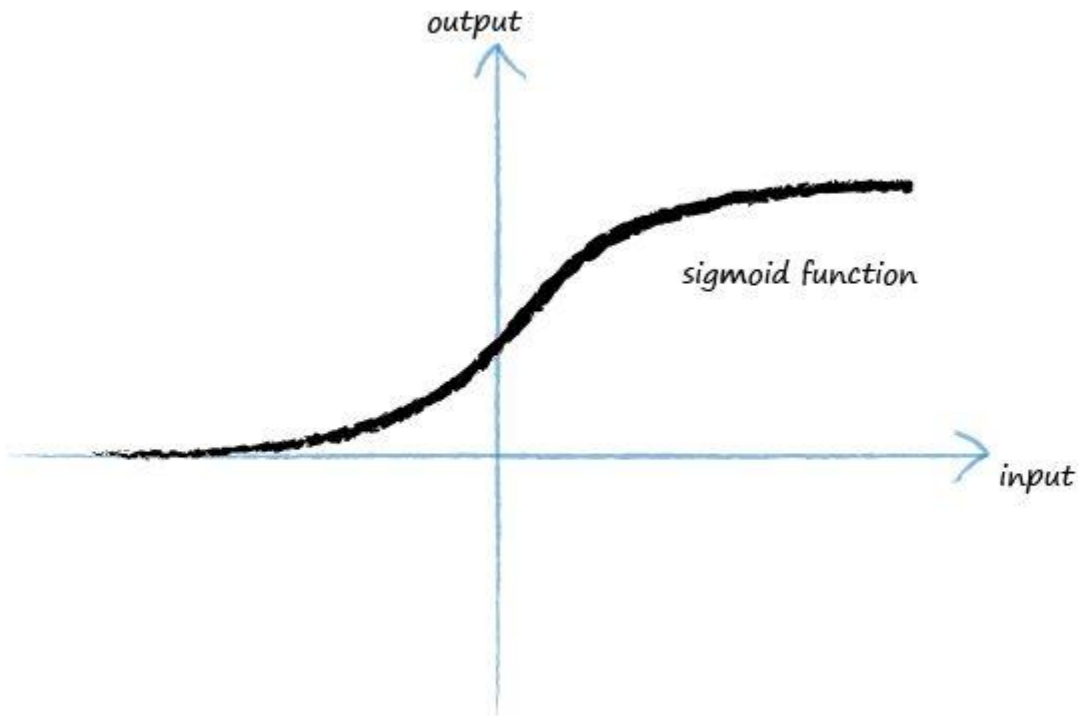
ACTIVATION FUNCTION



ACTIVATION FUNCTION

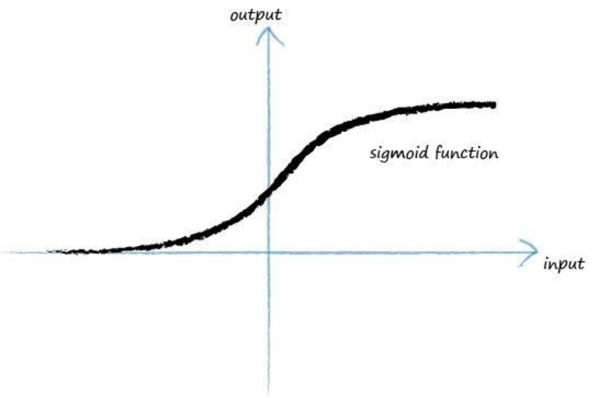


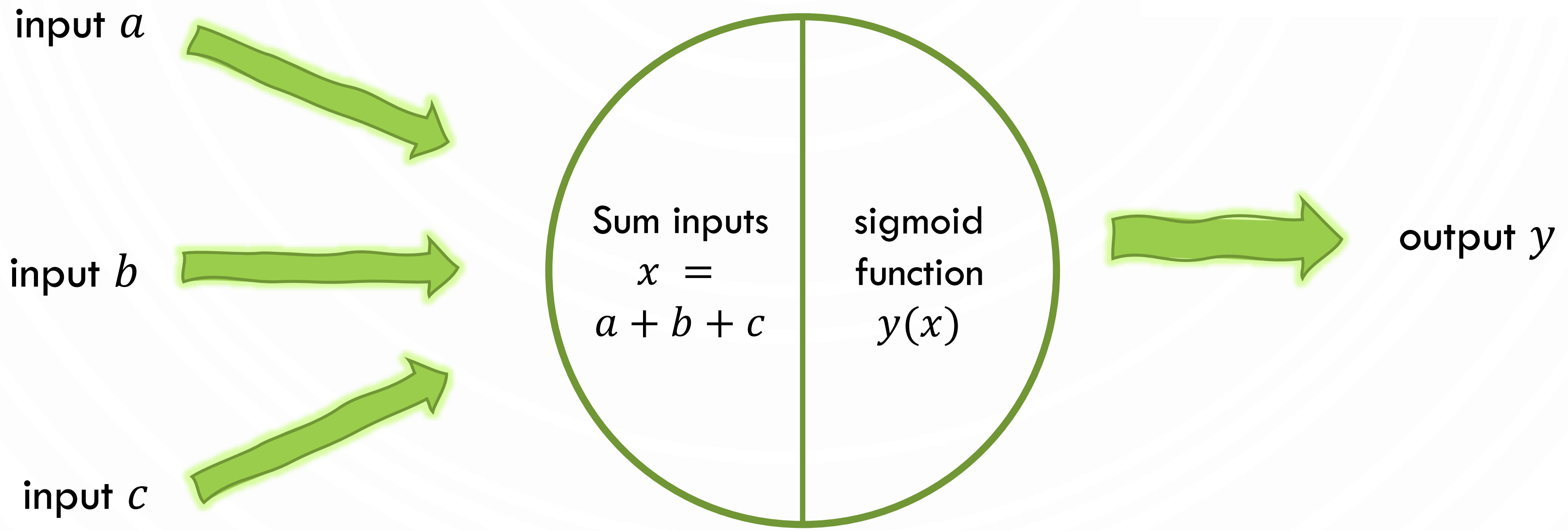
ACTIVATION FUNCTION



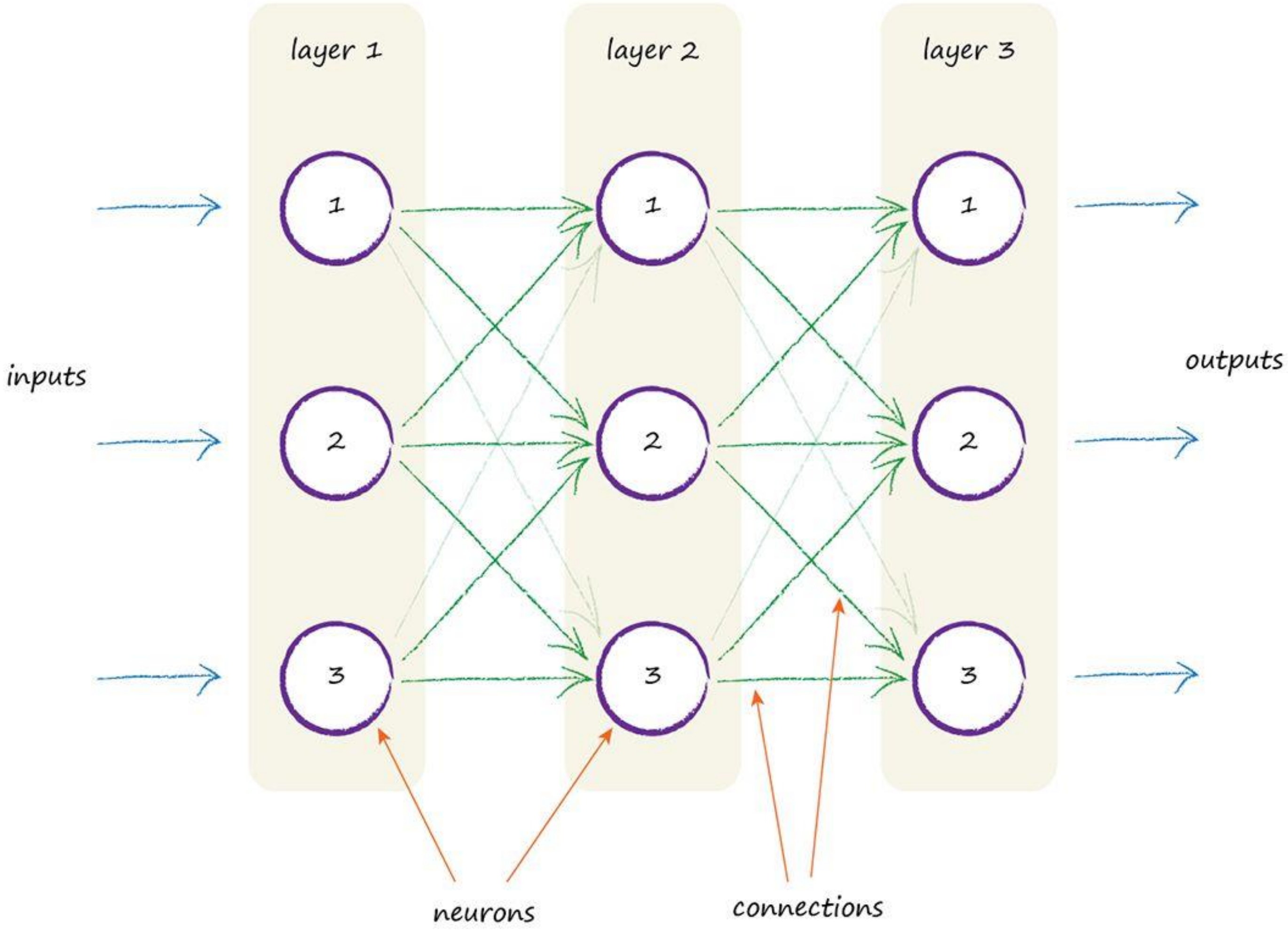
$$y = \frac{1}{1 + e^{-x}}$$

ARTIFICIAL NEURON

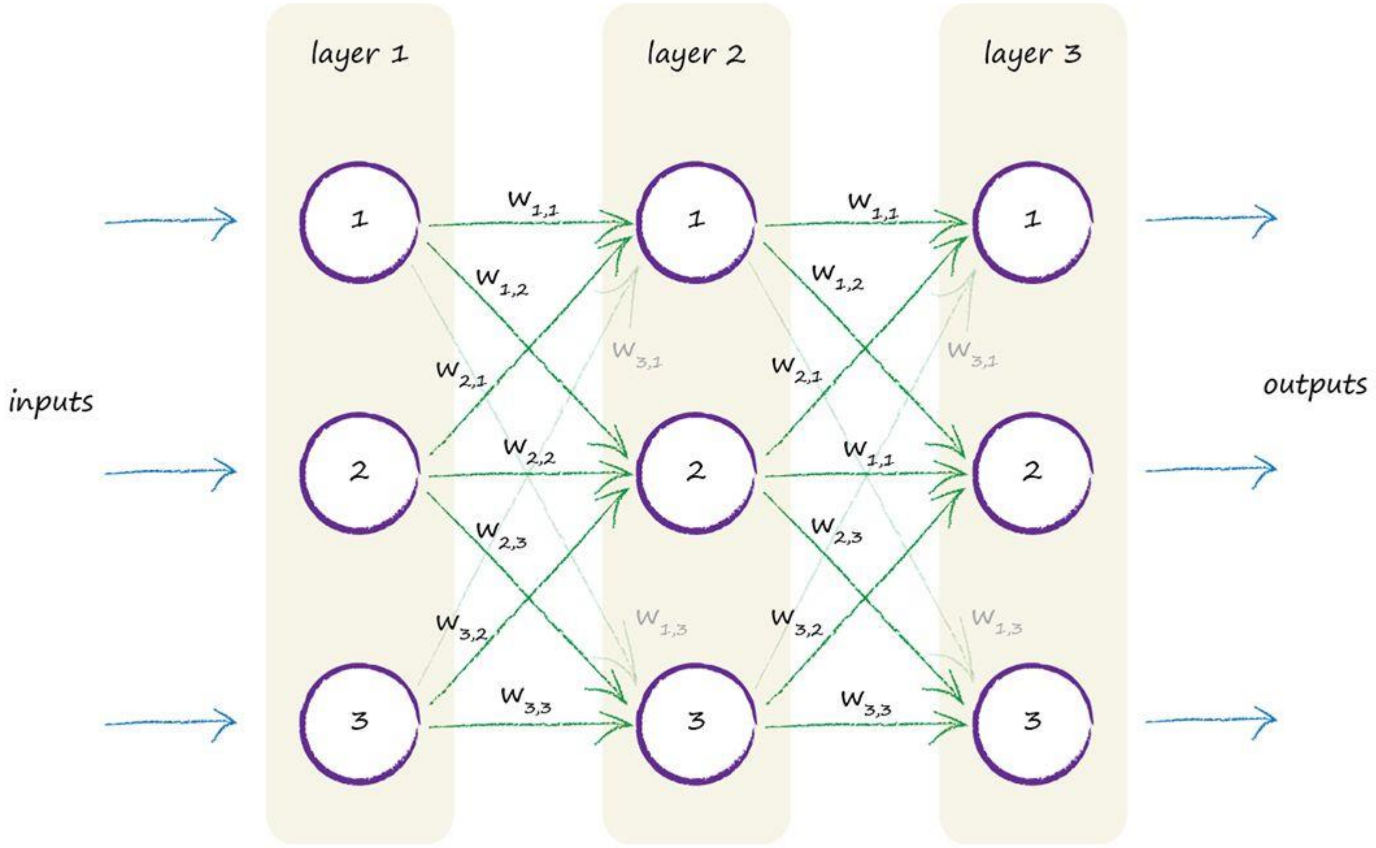




ARTIFICIAL NEURON NETWORK



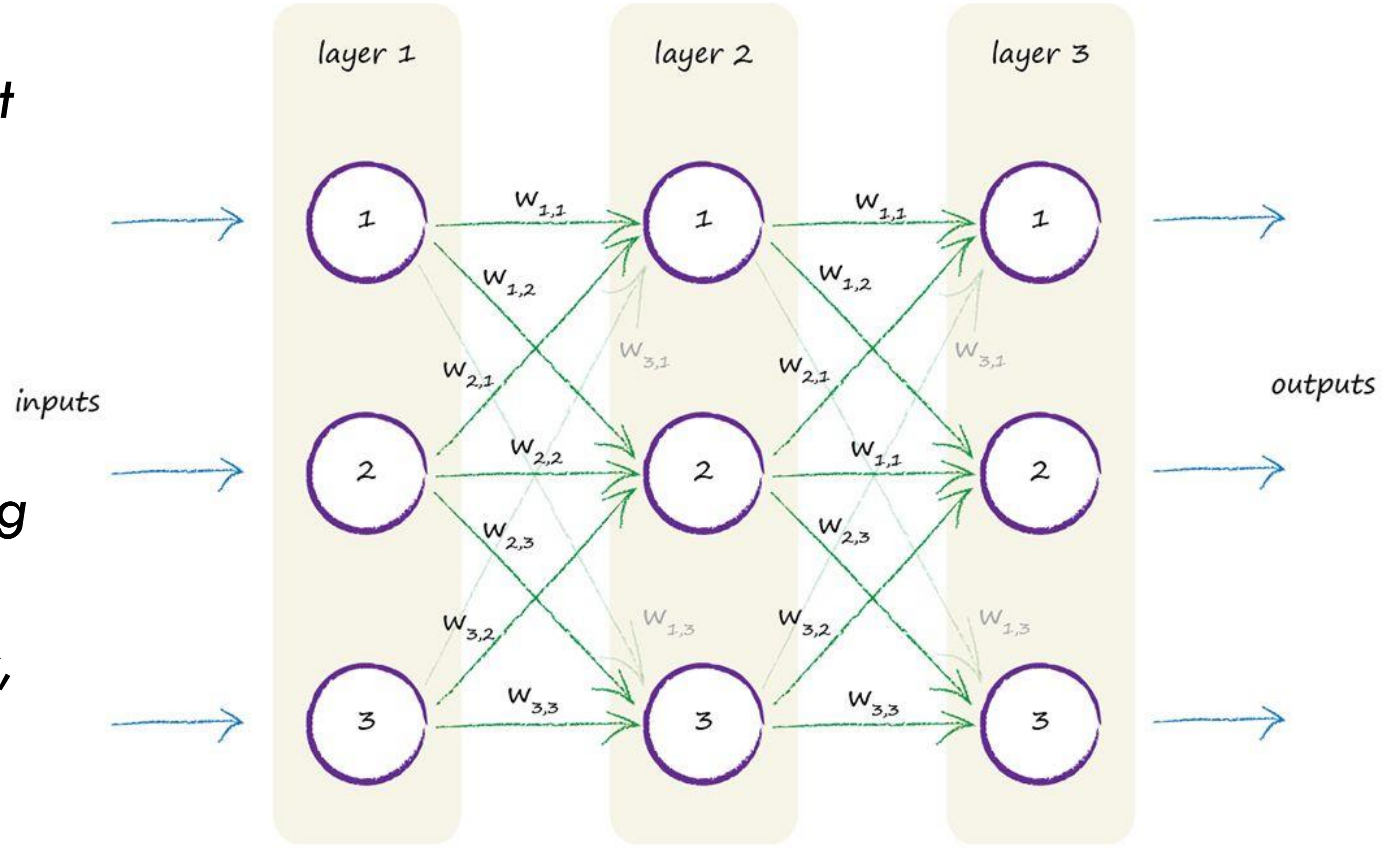
ARTIFICIAL NEURON NETWORK



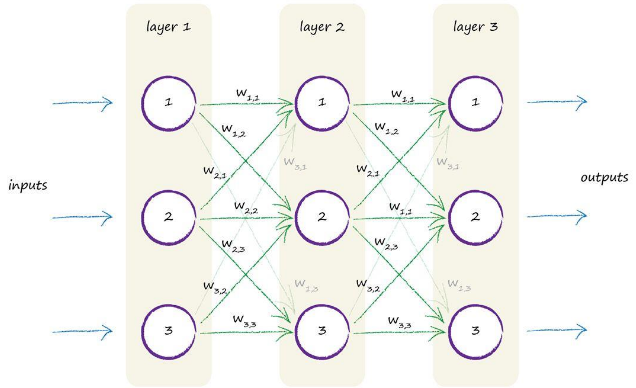
UNIVERSAL APPROXIMATION THEOREM

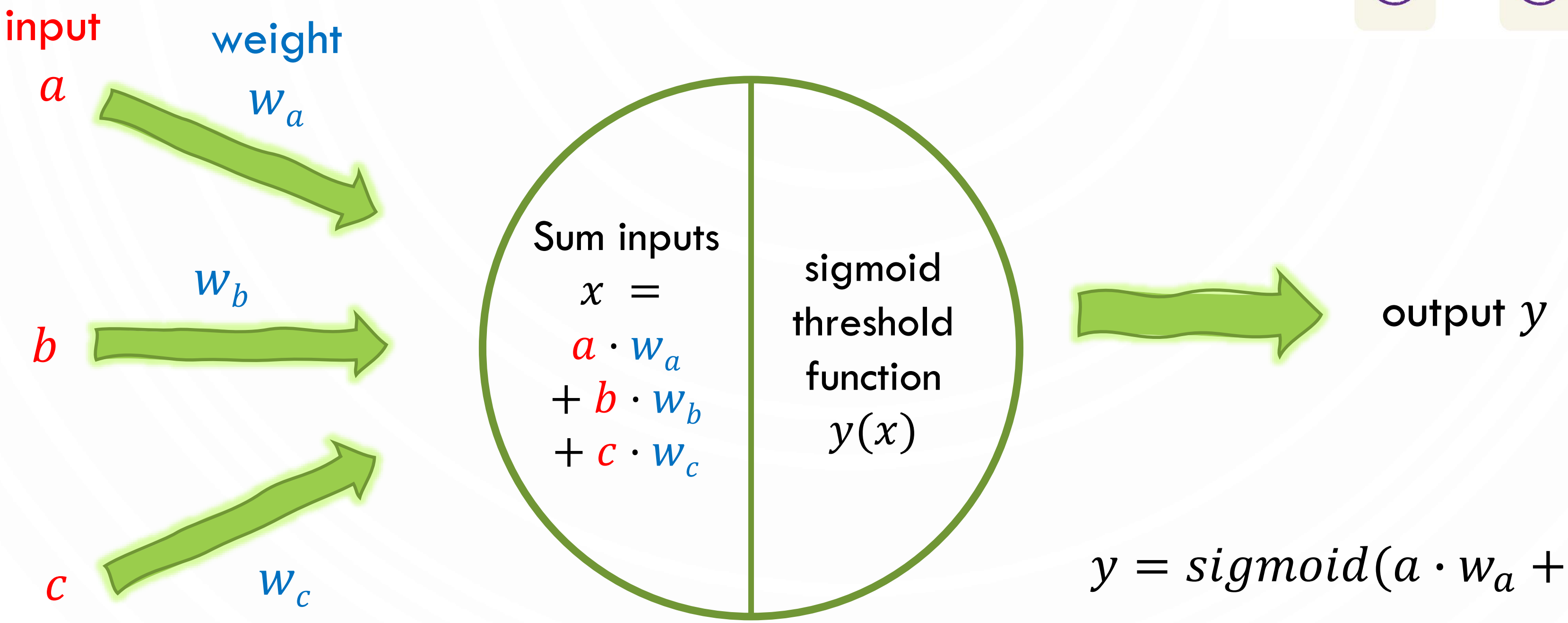
Neural networks can represent a wide variety of interesting functions when given appropriate weights.

It is capable of approximating any continuous functions between two Euclidean spaces, as long as having enough depths and nodes.



ARTIFICIAL NEURON

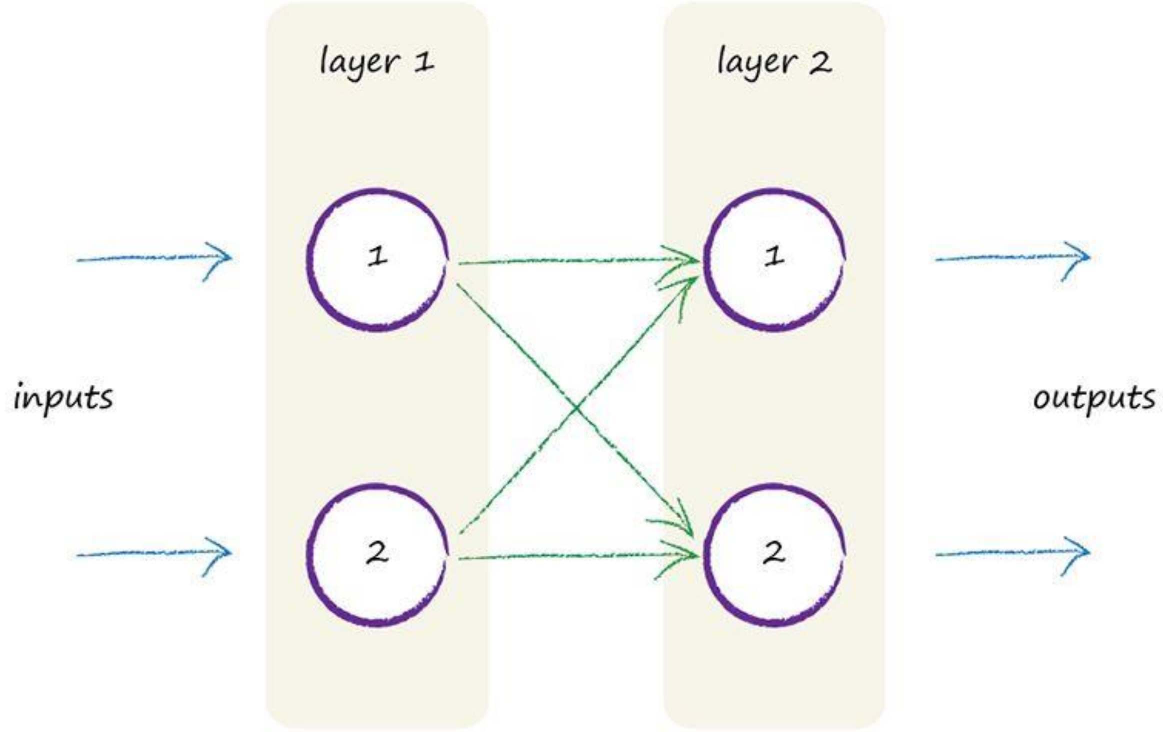




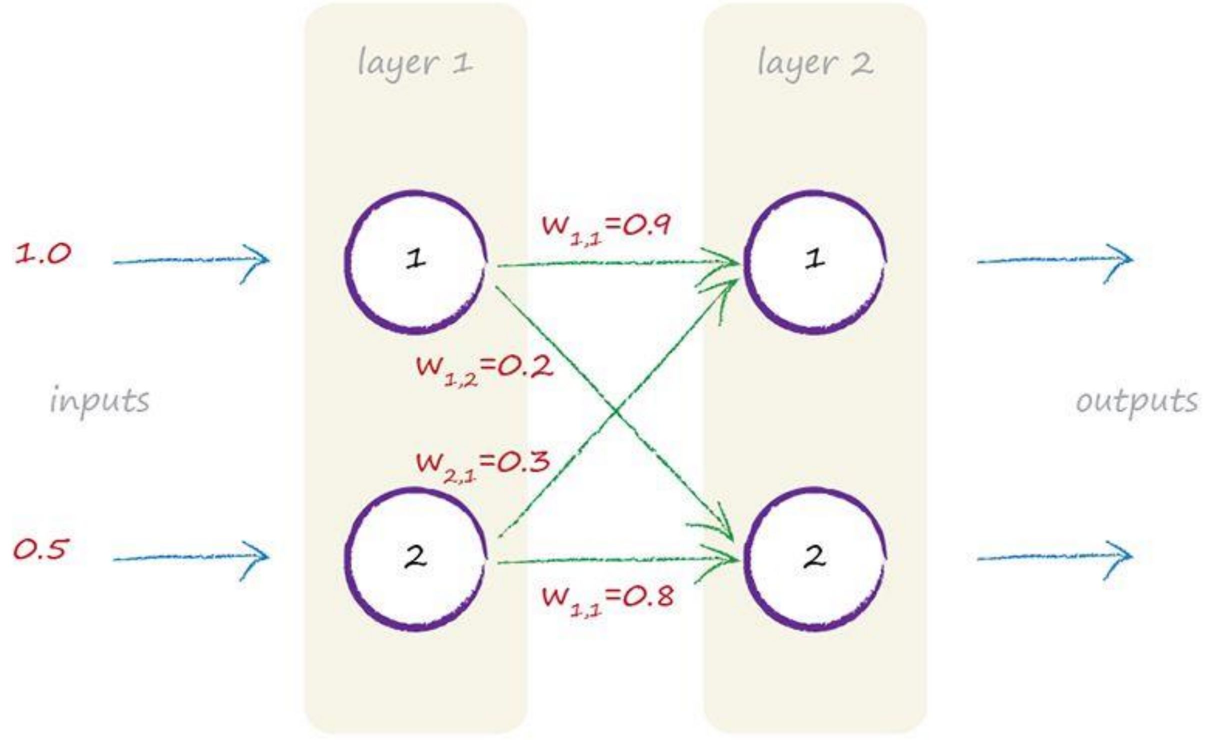
$$y = sigmoid(a \cdot w_a + b \cdot w_b + c \cdot w_c)$$

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

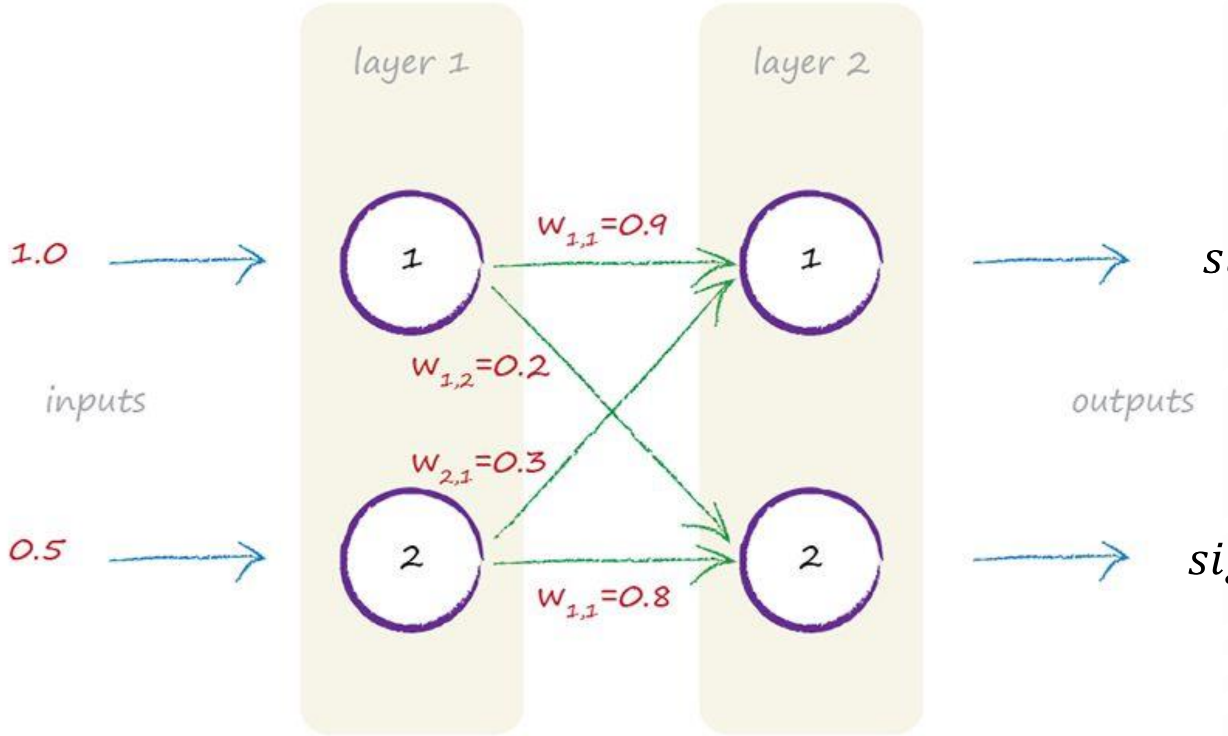
EXAMPLE - FEEDFORWARD SIGNAL



EXAMPLE - FEEDFORWARD SIGNAL



EXAMPLE - FEEDFORWARD SIGNAL

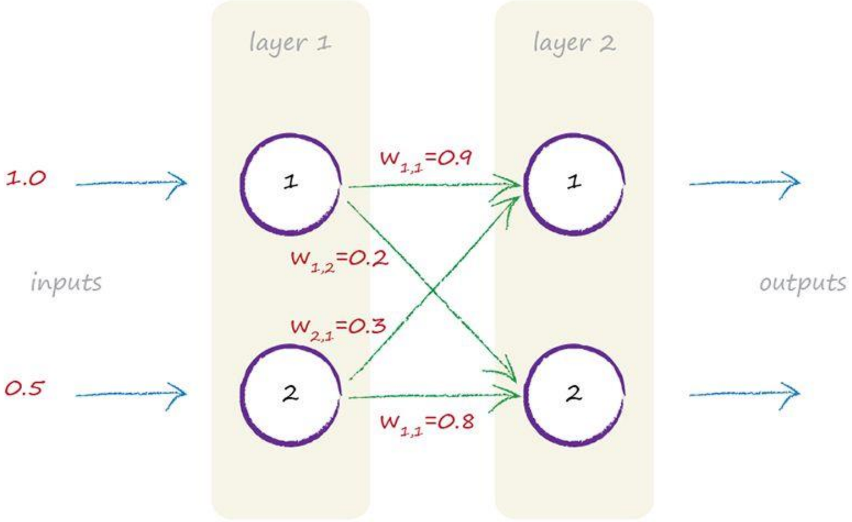


$$sigmoid = \frac{1}{1 + e^{-x}}$$

$$sigmoid(1.0 * 0.9 + 0.5 * 0.3) = 0.74$$

$$sigmoid(1.0 * 0.2 + 0.5 * 0.8) = 0.65$$

USING MATRIX MULTIPLICATION



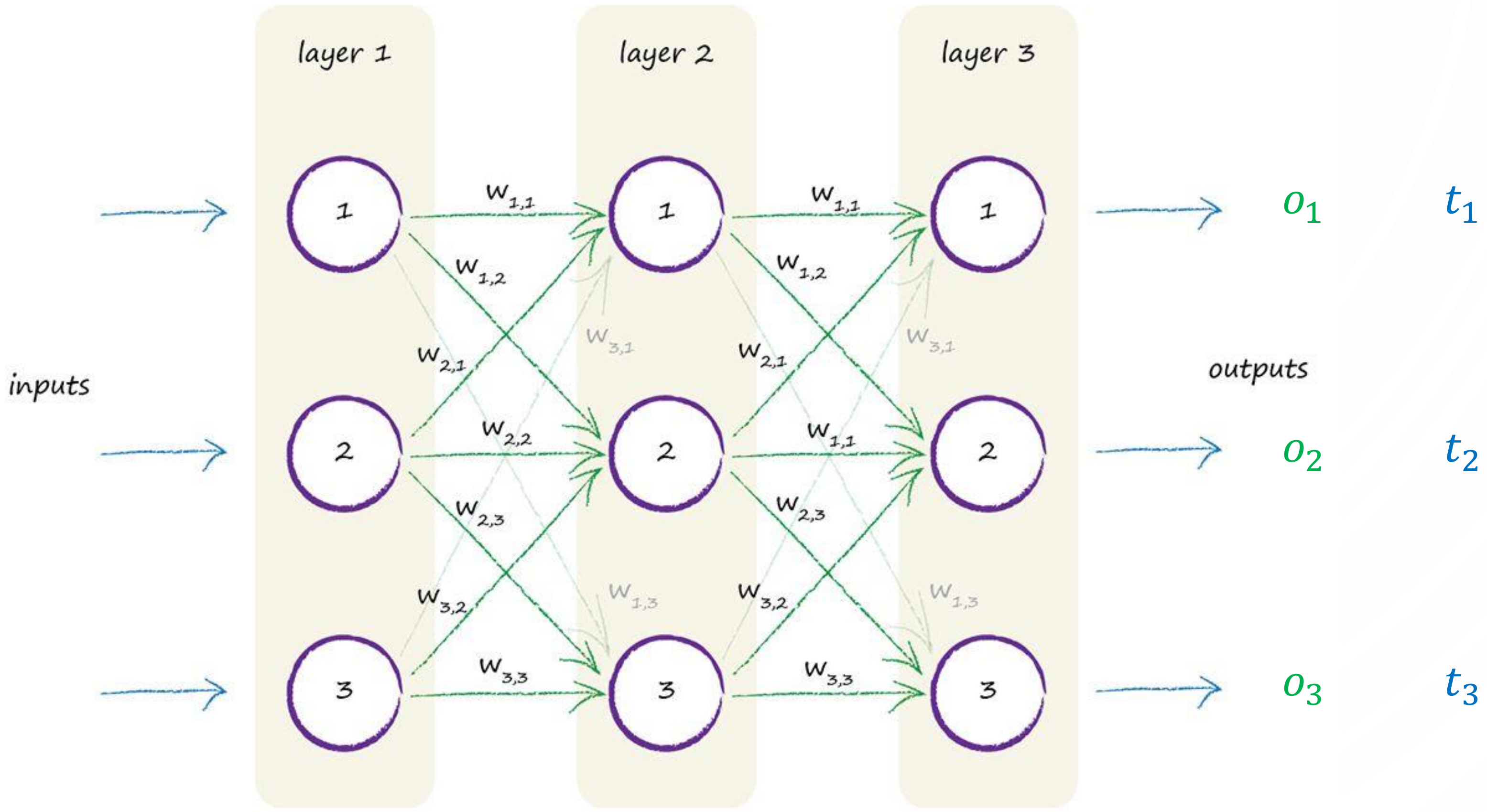
$$\begin{pmatrix} w_{1,1} & w_{2,1} \\ w_{1,2} & w_{2,2} \end{pmatrix} \begin{pmatrix} input\_1 \\ input\_2 \end{pmatrix} = \begin{pmatrix} (input\_1 * w_{1,1}) + (input\_2 * w_{2,1}) \\ (input\_1 * w_{1,2}) + (input\_2 * w_{2,2}) \end{pmatrix}$$

$$X = W \cdot I$$

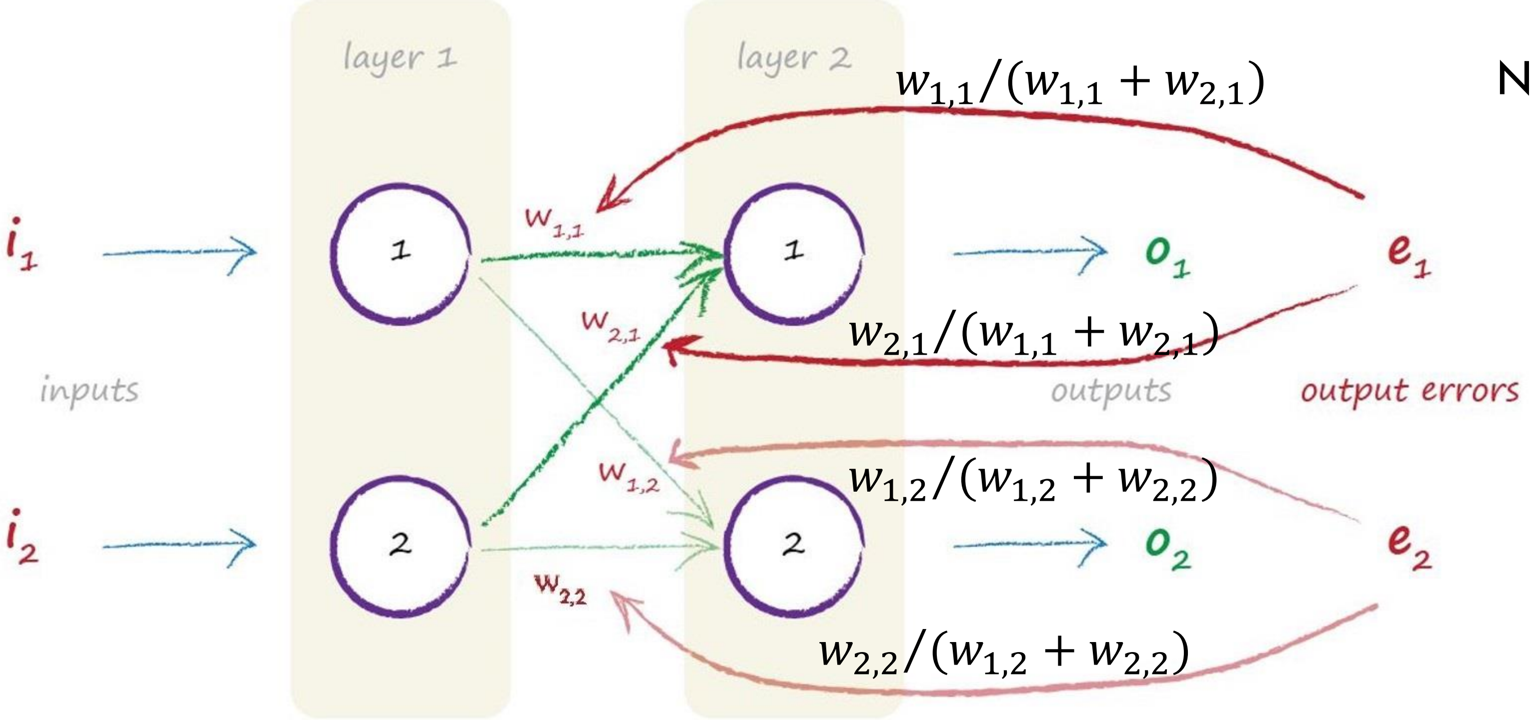
$$O = sigmoid(X)$$

BACKPROPAGATION OF LOSS

$$o_k = \frac{1}{1 + e^{-\sum_{j=1}^{3}\left(w_{j,k} \cdot \frac{1}{1+e^{-\sum_{i=1}^{3}(w_{i,j} \cdot x_i)}}\right)}}$$



BACKPROPAGATION OF LOSS

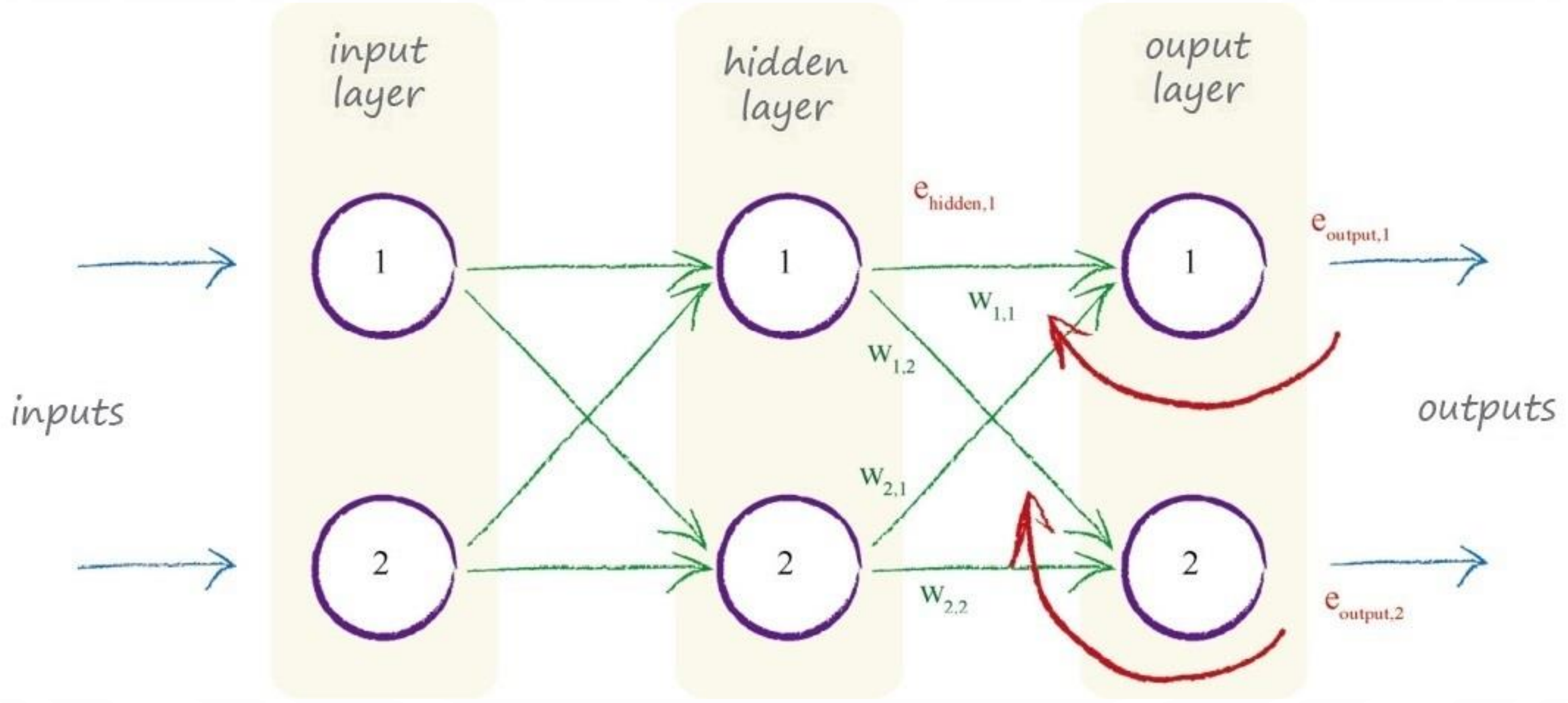


Node Loss = (actual – real)^2

$$e_{2,k} = (t_k - o_k)^2$$

$$e_{1,1} = e_{2,1} * \frac{w_{1,1}}{w_{1,1}+w_{2,1}} + e_{2,2} * \frac{w_{1,2}}{w_{1,2}+w_{2,2}}$$

BACKPROPAGATION OF LOSS



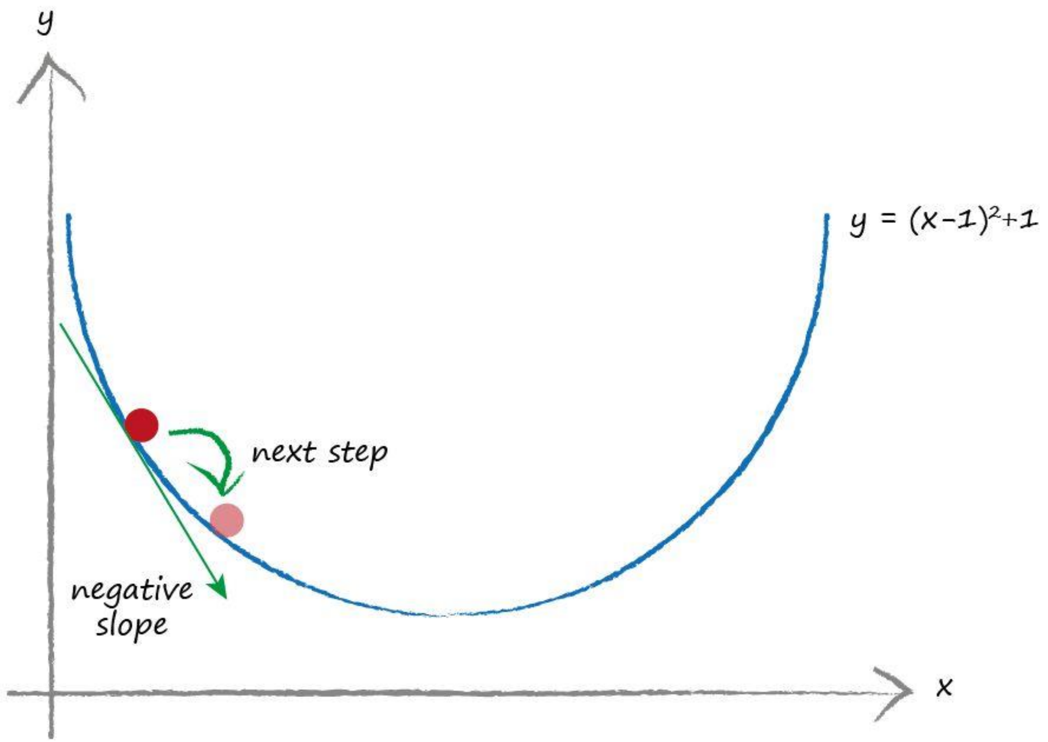
MATRIX MULTIPLICATION AGAIN

$$e_h = \begin{pmatrix} \frac{w_{1,1}}{w_{1,1} + w_{2,1}} & \frac{w_{1,2}}{w_{1,2} + w_{2,2}} \\ \frac{w_{2,1}}{w_{1,1} + w_{2,1}} & \frac{w_{2,2}}{w_{1,2} + w_{2,2}} \end{pmatrix} \cdot \begin{pmatrix} e_{o,1} \\ e_{o,2} \end{pmatrix}$$

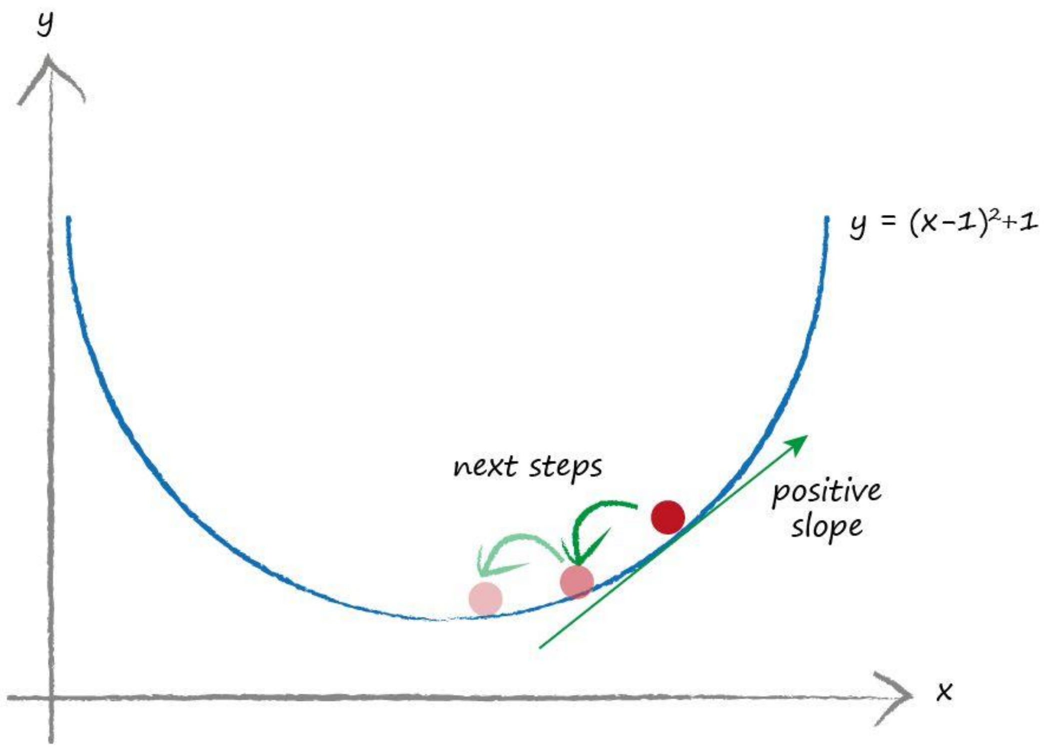
$$\Downarrow$$

$$e_h = \begin{pmatrix} w_{1,1} & w_{1,2} \\ w_{2,1} & w_{2,2} \end{pmatrix} \cdot \begin{pmatrix} e_{o,1} \\ e_{o,2} \end{pmatrix} = w^T \cdot e_o$$

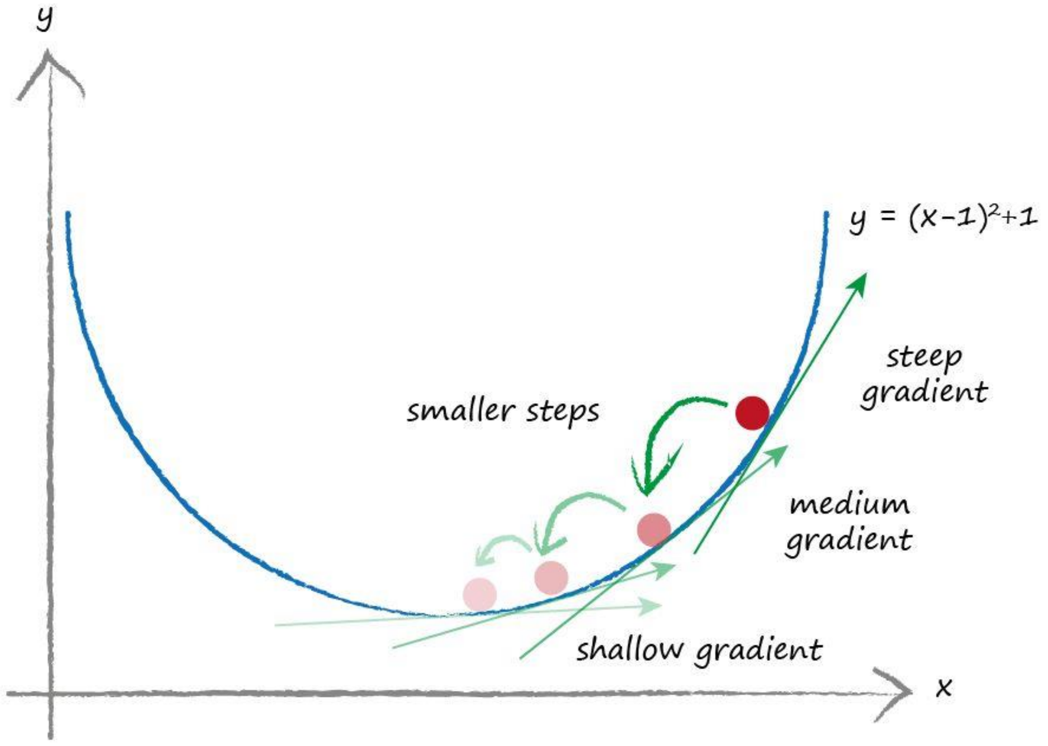
UPDATE WEIGHTS - GRADIENT DESCENT



UPDATE WEIGHTS - GRADIENT DESCENT



UPDATE WEIGHTS - GRADIENT DESCENT



UPDATE WEIGHTS - GRADIENT DESCENT

$$\frac{\partial E}{\partial w_{j,k}} = \frac{\partial (t_k - o_k)^2}{\partial w_{j,k}} = \frac{\partial (t_k - o_k)^2}{\partial o_k} \cdot \frac{\partial o_k}{\partial w_{j,k}} = -2(t_k - o_k) \cdot \frac{\partial o_k}{\partial w_{j,k}}$$

$$= -2(t_k - o_k) \cdot \frac{\partial sigmoid\left(\sum_j w_{j,k} \cdot o_j\right)}{\partial w_{j,k}}$$

$$= -2(t_k - o_k) \cdot sigmoid\left(\sum_j w_{j,k} \cdot o_j\right)\left(1 - sigmoid\left(\sum_j w_{j,k} \cdot o_j\right)\right) \cdot \frac{\partial\left(\sum_j w_{j,k} \cdot o_j\right)}{\partial w_{j,k}}$$

$$= -2\color{purple}{(t_k - o_k)} \cdot \color{red}{sigmoid\left(\sum_j w_{j,k} \cdot o_j\right)}\left(1 - \color{red}{sigmoid\left(\sum_j w_{j,k} \cdot o_j\right)}\right) \cdot \color{green}{o_j}$$

$$\Delta w_{j,k} = -\alpha \cdot \frac{\partial E}{\partial w_{j,k}} = -\alpha \cdot (-\color{purple}{E_k} \cdot \color{red}{o_k}(1 - \color{red}{o_k}) \cdot \color{green}{o_j^T})$$

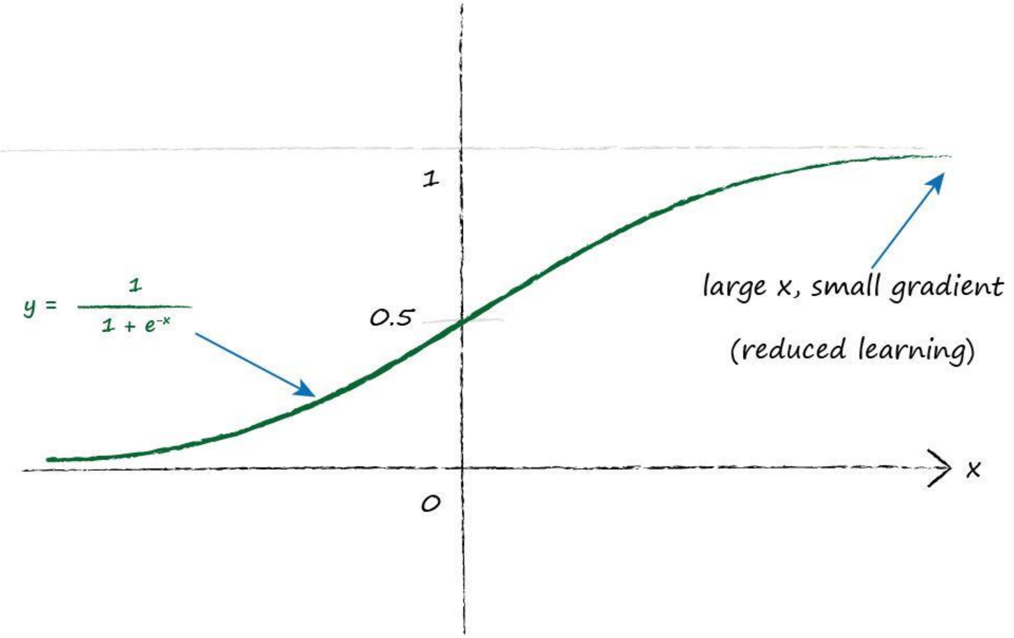
INPUT / OUTPUT & INITIAL WEIGHTS

$$-2(t_k - o_k) \cdot sigmoid\left(\sum_j w_{j,k} \cdot o_j\right)\left(1 - sigmoid\left(\sum_j w_{j,k} \cdot o_j\right)\right) \cdot o_j$$

Input/Output 0~1

Initial Weights $\frac{-1}{\sqrt{n}} \sim \frac{1}{\sqrt{n}}$

n = number of nodes in target layer



** Break Symmetry: never set initial weights to the same constant value, especially no zero.*

CODE EXAMPLE - NUMPY

MUDSS

Data Science Society

THANK YOU!

References:

- Rashid, Tariq. *Make your own neural network*. CreateSpace Independent Publishing Platform, 2016.