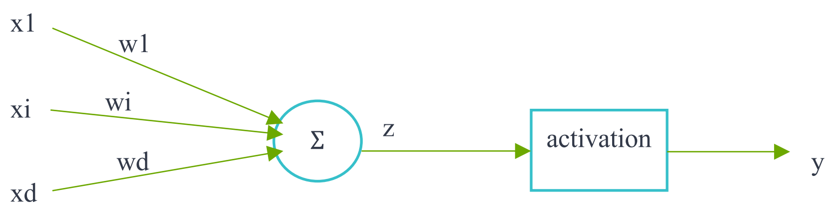
2021/03/01

Artificial Neural Network (ANN)

Perceptron

The perceptron consists a single neuron with adjustable synaptic weights and a hard limiter.



Notation:

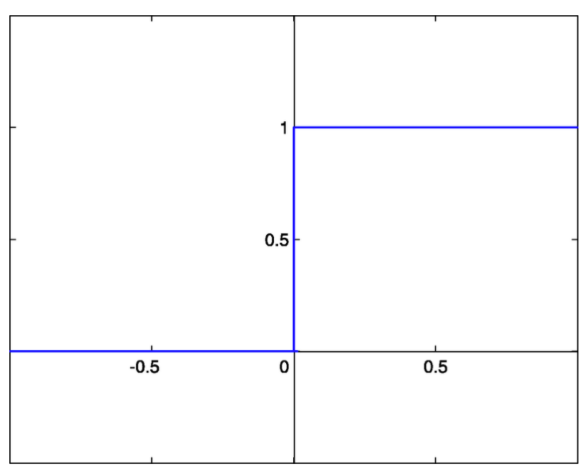
$$x = \begin{bmatrix} x1 \\ x2 \\ x3 \\ \dots \\ xd \end{bmatrix} x \in R^d \qquad w = \begin{bmatrix} w1 \\ w2 \\ w3 \\ \dots \\ wd \end{bmatrix} w \in R^d$$

$$z = w^T x = \sum_{i=0}^{d} x_i * w_i = x_1 * w_1 + x_2 * w_2 + \dots + x_d * w_d$$
$$y = activation\ function(z)$$

Some activation function

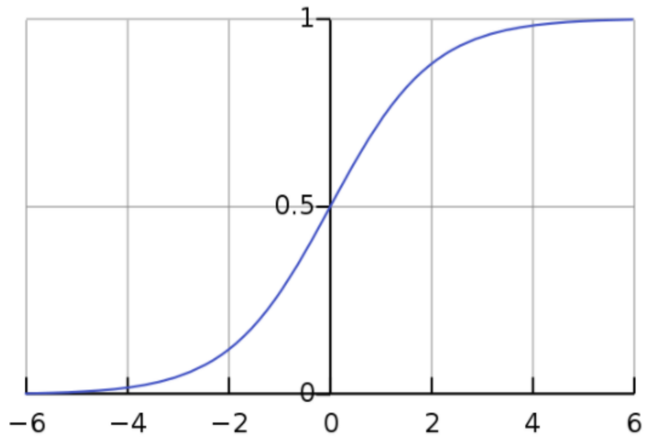
Step function:

$$y = f(x) = \begin{cases} 0, & x < 0 \\ 1, & x \geq 0 \end{cases}$$



Sigmod function

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



Perceptron learning rule

Notation:
α : learning rate
θ : Threshold/Bias
$X^{n \times d}$: train data set with n row, each row is vector d- dimensional

Each row of dataset we will do 3 steps as below:

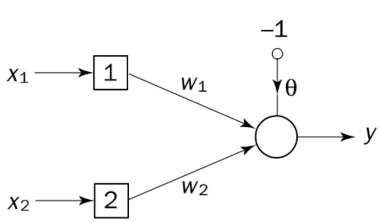
Step 1: initialization
Initial weights w^d / { $w_i \in [-0.5, 0.5]$ }
Initial bias $\theta \in [-0.5, 0.5]$
Initial $\alpha \in [0, 1]$

Step 2: Activation
$z = w^T x = \sum_{i=0}^{d} x_i * w_i - \theta = x_1 * w_1 + x_2 * w_2 + \ldots + x_d * w_d - \theta$
$y = \text{activation function}(z)$

Step 3: Weight update
$w_{new}^d = w_{new}^d / \{ w_{i-new} = w_{i-old} + \alpha * (y_{desired} - y) * x_i \}$

Go to next row

Example: Traing AND gate

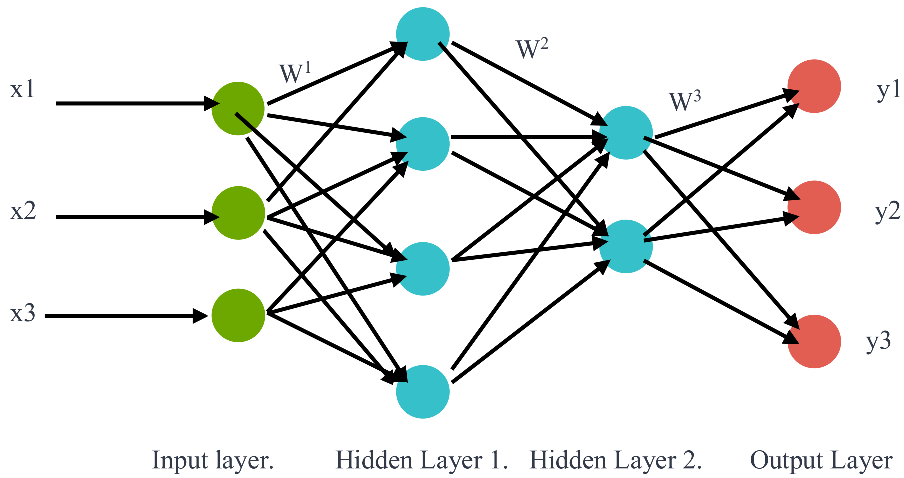


Bias θ = 0.2
Learning rate α = 0.1

x1	x2	$y_{desired}$	w1	w2	z	y	w1-new	w2-new
0	0	0	0.3	-0.1	= 0 * 0.3 + 0 * -0.1 – 0.2 = -0.2	= step(-0.2) = 0	= 0.3 + 0.1 * (0 - 0) * 0 = 0.3	= -0.1 + 0.1 * (0 - 0) * 0 = -0.1
0	1	0	0.3	-0.1	= 0 * 0.3 + 1 * -0.1 – 0.2 = -0.3	= step(-0.2) = 0	= 0.3 + 0.1 * (0 - 0) * 0 = 0.3	= -0.1 + 0.1 * (0 - 0) * 1 = -0.1
1	0	0	0.3	-0.1	= 1 * 0.3 + 0 * -0.1 – 0.2 = 0.1	= step(0.1) = 1	= 0.3 + 0.1 * (0 - 1) * 1 = 0.2	= -0.1 + 0.1 * (0 - 1) * 0 = -0.1
1	1	1	0.2	-0.1	= 0.2 * 1 – 0.1 * 1 – 0.2 = -0.1	= step(-0.1) = 0	= 0.2 + 0.1 * (1 - 0) * 1 = 0.3	= -0.1 + 0.1 * (1 – 0) * 1 = 0

ANN

Insted of having input layer and outpyt layer, The ANN includes mutiple hidder layers



Notation

e_1^{l} $\quad z_1^{l} \rightarrow a_1^{l}$

$e_1^{(l-1)}$ $\quad z_1^{(l-1)} \rightarrow a_1^{(l-1)}$

…

e_2^{l} $\quad z_2^{l} \rightarrow a_2^{l}$

…

$e_i^{(l-1)}$ $\quad z_i^{(l-1)} \rightarrow a_i^{(l-1)}$

…

e_j^{l} $\quad z_j^{l} \rightarrow a_j^{l}$

$e_{d^{(l-1)}}^{(l-1)}$ $\quad z_{d^{(l-1)}}^{(l-1)} \rightarrow a_{d^{(l-1)}}^{(l-1)}$

…

$e_{d^l}^{l}$ $\quad z_{d^l}^{l} \rightarrow a_{d^l}^{l}$

z_{index}^{layer} : sum of input * weight of particular layer of an particular unit
a_{index}^{layer}: the actual output after applied activation function of particular layer of a particular unit
e_{index}^{layer}: the amount of error of particular layer of particular unit
y_{index}: desired output with particular index

Feed forward neural network

$w_j^l \in R^{d^{(l-1)}}$

$z_j^l = w_j^{lT} * a^{(l-1)}$

$a_j^l = f(z_j^l)$

Backpropagation neural network

Output Layer Error

$e_j^{output\ layer} = (y_{index} - a_j^l) * (a_j^l * (1 - a_j^l))$

$w_{ij}^l = w_{ij}^l + \sigma * a_i^{(l-1)} * e_j^{output\ layer}$

Hidden layer error

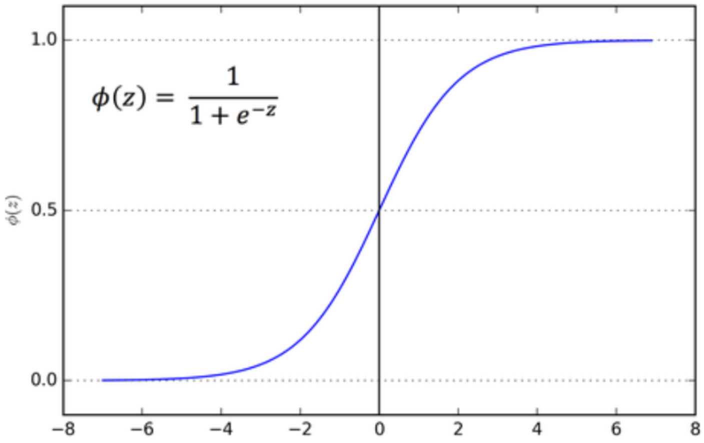
$$e_i^{(l-1)} = a_i^{(l-1)} * (1 - a_i^{(l-1)}) * \sum_{j=1}^{j=d^l} e_j^l * w_{ij}^l$$

$w_{ij}^{(l-1)} = w_{ij}^{(l-1)} + \sigma * a_i^{(l-2)} * e_i^{(l-1)}$

Activation function

Sigmoid or Logistic Activation function

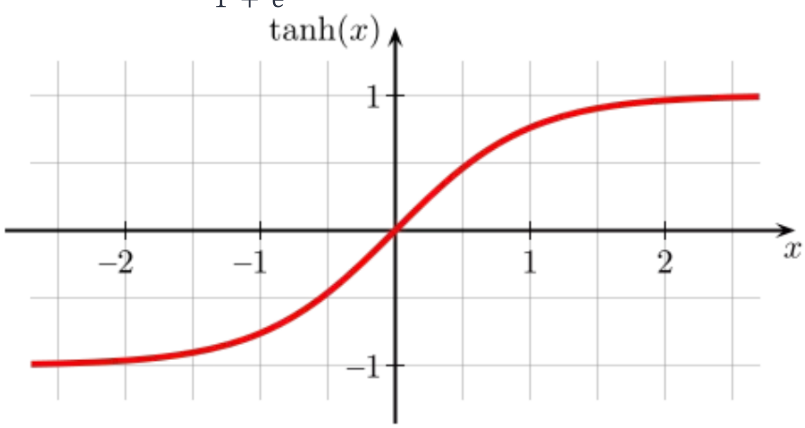
$$y = f(x) = \frac{1}{1 + e^{-x}}$$



Range: [0, 1]
It is especially used for models where we have to predict the probability as an output. Since probability of anything exists only between the range of 0 and 1, sigmoid is right choice.

Tanh

$$f(x) = \tanh(x) = \frac{2}{1 + e^{-2x}} - 1$$



Range: [-1, 1]

ReLU

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

Range: [0, infinity]

refs: https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6