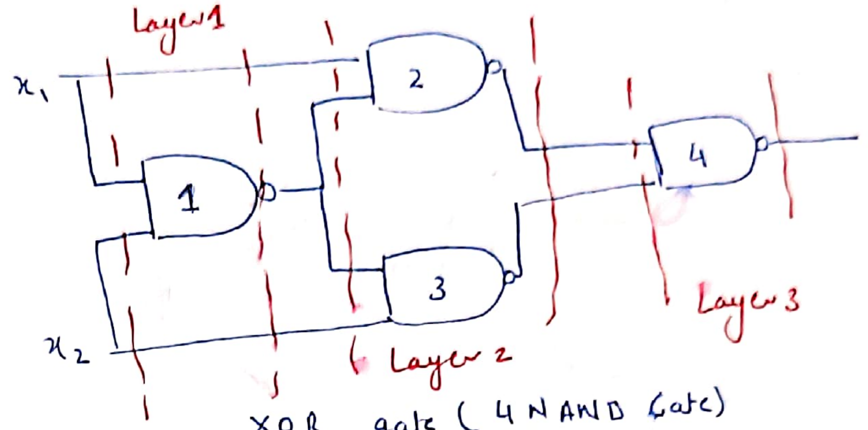
Perceptron : Drawbacks : Only works for linearly separable data

XOR ×

NAND : XOR can also be implemented as a NAND Gate

(1)



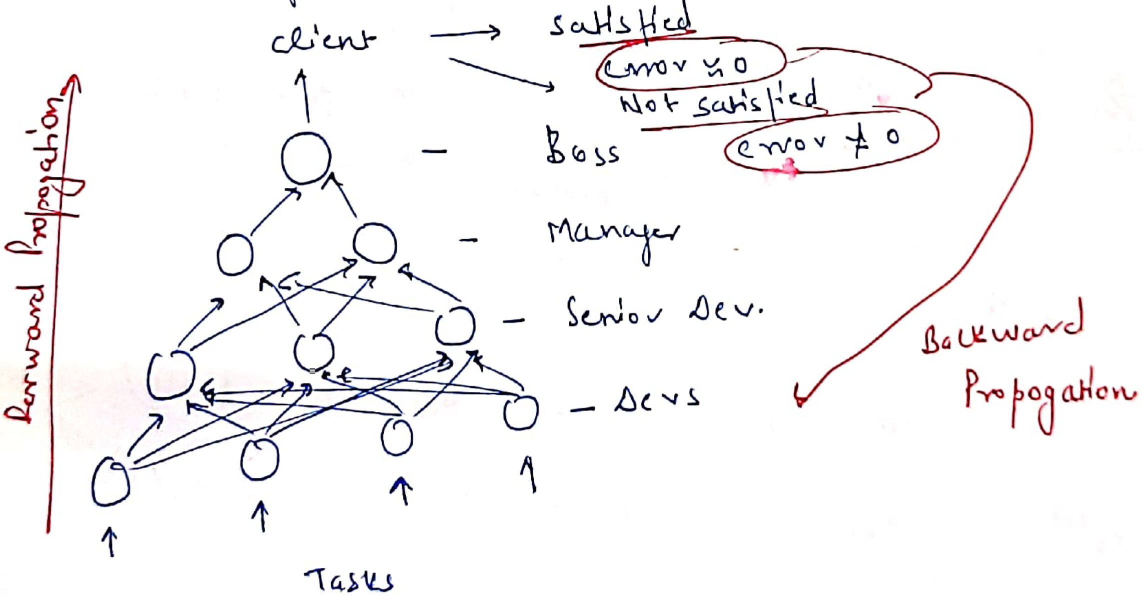
XOR gate (4 NAND Gate)

One NAND gate can be thought of as one perceptron. If we stack multiple perceptron, we can solve a non linear problem as well.

Multilayer perceptron can solve non linear problem as well.

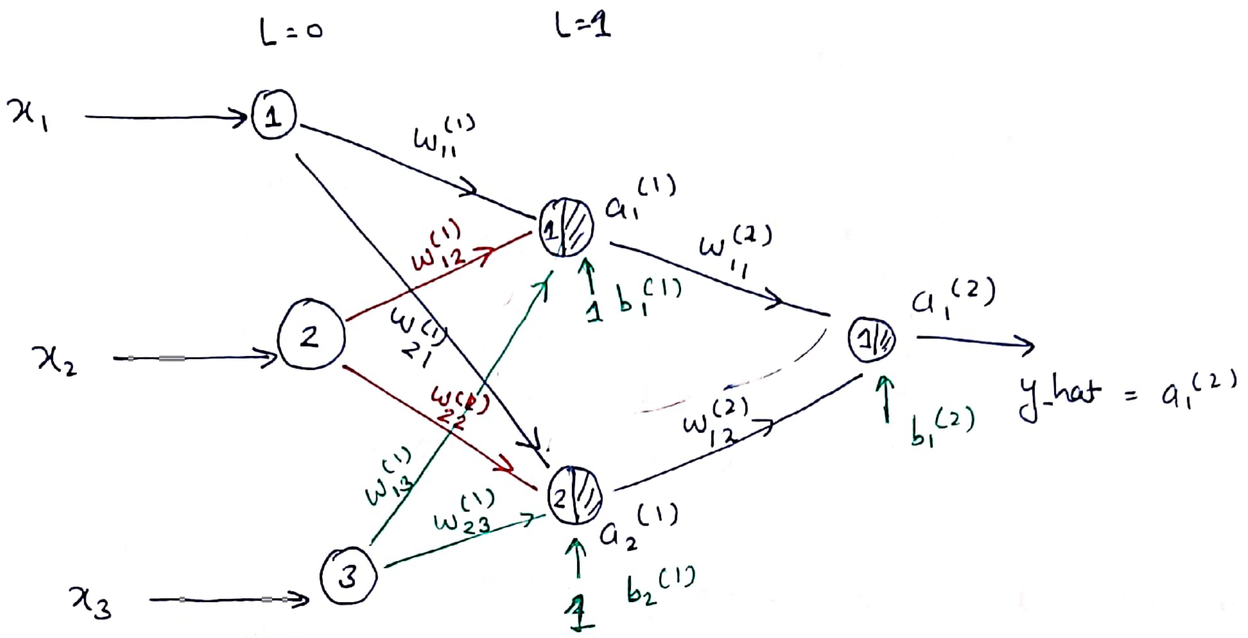
↓

ANN (Artificial Neural Network)



Activation function will filter out the unwanted information and only transmit the most important one to the further layer. There can be many types of activation functions that we can look for.

* forward Propogation



(N) → Nth Layer (L) → ~~L = N-1~~

W_{ni} → source, destination

$L = N-1$

activation functions

$a_i^{(N)}$ → Nth layer

↑ coming from which activation fn

outcome

at layer 1

① – $z_1^{(1)} = w_{11}^{(1)} x_1 + w_{12}^{(1)} x_2 + w_{13}^{(1)} x_3 + b_1^{(1)}$

② – $z_2^{(1)} = w_{21}^{(1)} x_1 + w_{22}^{(1)} x_2 + w_{23}^{(1)} x_3 + b_2^{(1)}$

bias

$$a_1^{(1)} = \sigma(z_1^{(1)}) \approx \begin{cases} 1 & z_1^{(1)} \geq \theta \\ 0 & z_1^{(1)} < \theta \end{cases}$$

↳ sigmoid fn

↳ this type can change

$$a_2^{(1)} = \sigma(z_2^{(1)})$$

final output layer

3 - $z_1^{(2)} = w_{11}^{(2)} a_1^{(1)} + w_{12}^{(2)} a_2^{(1)} + b_1^{(2)}$

adding bias.

$a_1^{(2)} = \sigma(z_1^{(2)}) \rightarrow \hat{y}$

error/loss $= y - \hat{y}$

cost function

$cost(y, \hat{y})$

Back Propogation → Weight update Rule

$W = W + \Delta W$

$\Delta W = -\eta \dfrac{\partial cost}{\partial w}$

change in cost function with respect to small change in weights.

learning rate

Let's introduced the bias.

$z_1^{(1)} = w_{11}^{(1)} x_1 + w_{12}^{(1)} x_2 + w_{13}^{(1)} x_3 + b_1^{(1)}$

$z_2^{(1)} = w_{21}^{(1)} x_1 + w_{22}^{(1)} x_2 + w_{23}^{(1)} x_3 + b_2^{(1)}$

$z_1^{(2)} = w_{11}^{(2)} a_1^{(1)} + w_{12}^{(2)} a_2^{(1)} + b_1^{(2)}$

Matrix form of same things

from 1 & 2 for layer 1

$$\underset{2\times3}{\begin{bmatrix} w_{11}^{(1)} & w_{12}^{(1)} & w_{13}^{(1)} \\ w_{21}^{(1)} & w_{22}^{(1)} & w_{23}^{(1)} \end{bmatrix}} \underset{3\times1}{\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}} + \underset{2\times1}{\begin{bmatrix} b_1^{(1)} \\ b_2^{(1)} \end{bmatrix}} = \underset{2\times1}{\begin{bmatrix} z_1^{(1)} \\ z_2^{(1)} \end{bmatrix}}$$

$$\begin{bmatrix} z_1^{(1)} \\ z_2^{(1)} \end{bmatrix}_{2\times 1} \xrightarrow[\text{function}]{\text{activation}} \begin{bmatrix} \sigma(z_1^{(1)}) \\ \sigma(z_2^{(1)}) \end{bmatrix} = \begin{bmatrix} a_1^{(1)} \\ a_2^{(1)} \end{bmatrix}_{2\times 1}$$

At final layer

$$\begin{bmatrix} w_{11}^{(2)} & w_{12}^{(2)} \end{bmatrix}_{1\times 2} \begin{bmatrix} a_1^{(1)} \\ a_2^{(1)} \end{bmatrix}_{2\times 1} + \begin{bmatrix} b_1^{(2)} \end{bmatrix}_{1\times 1} = \begin{bmatrix} z_1^{(2)} \end{bmatrix}_{1\times 1}$$

$\downarrow$ activation f^n

$$\hat{y} \leftarrow \begin{bmatrix} a_1^{(2)} \end{bmatrix} \leftarrow \begin{bmatrix} \sigma(z_1^{(2)}) \end{bmatrix}_{1\times 1}$$

Q Why do we need an Activation function?

① It helps achieve non linearity as there are variety of activation functions available

1> sigmoid f^n $\quad \sigma(x) = \dfrac{1}{1+e^{-x}}$

2> $\tanh(x)$ $\quad \dfrac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$

3) ReLU (x) $\quad \max(x, 0)$

② It helps us to converge the solution

1> $\sigma(x)$ $\quad$ ~~Range~~ Input $(-\infty, \infty)$: domain

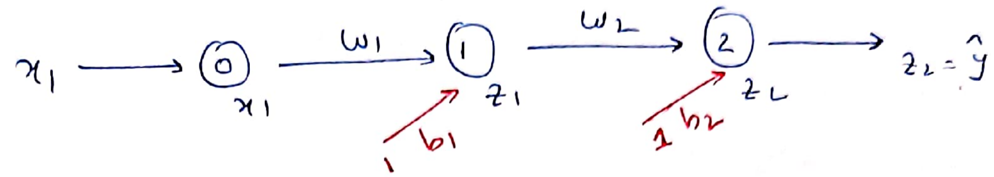
outcome/Range $(0, 1)$

helps reduce the solution space

2) $\tanh(x)$ domain $(-\infty, \infty)$ Range $(-1, 1)$

3) $ReLU(x)$ domain $(-\infty, \infty)$ Range $(0, \infty)$

Assumption (No activation function available)



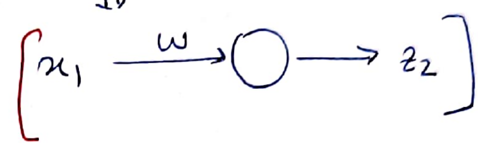
$z_1 = w_1 x_1$

$z_2 = w_2 z_1$

$z_2 = w_2 w_1 x_1$

$z_2 = w x_1$

⇓



↳ entire network boiled down to one single neuron

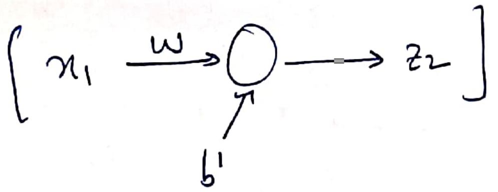
* No benefit of multilayer classification or multilayer effect without an activation function.

* Lets introduce the bias without activation function

$z_1 = w_1 x_1 + b_1$

$z_2 = w_2 z_1 + b_2 = w_2 w_1 x_1 + w_2 b_1 + b_2$

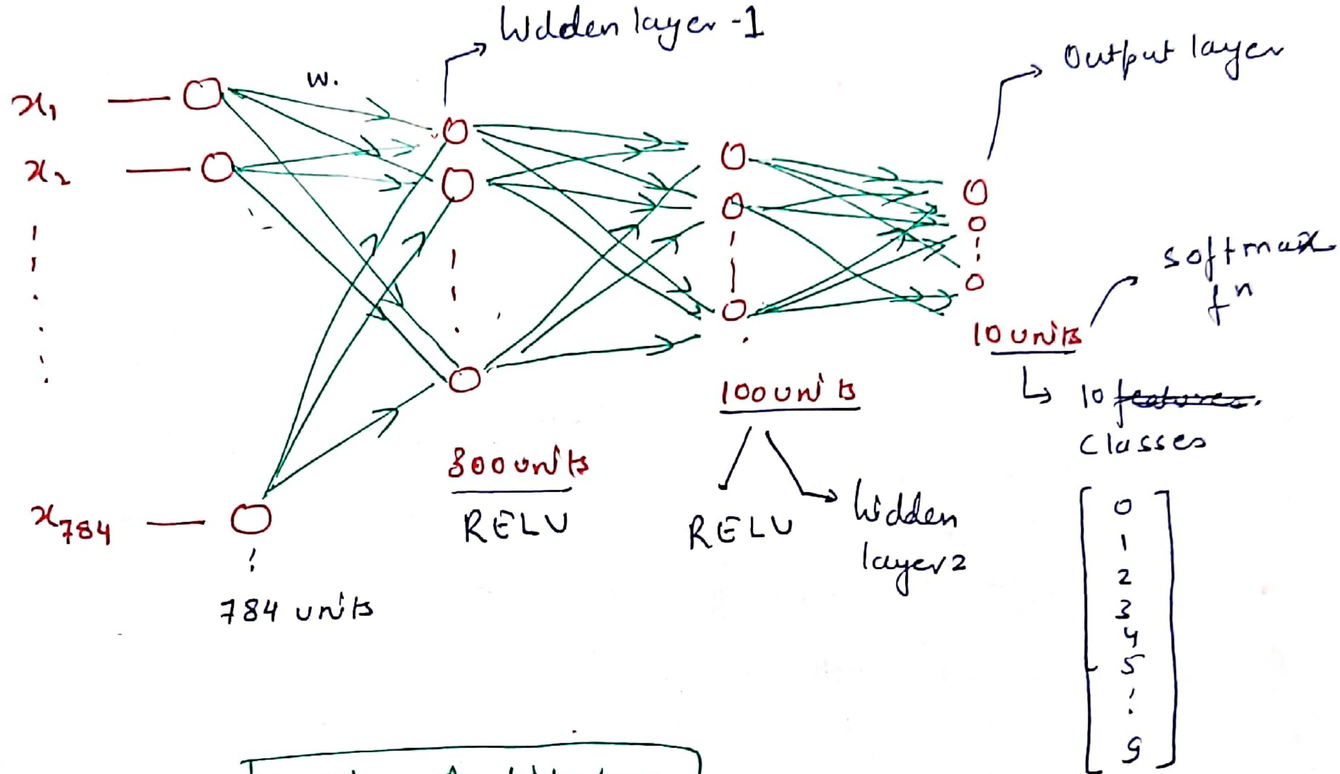
$\simeq w x_1 + b'$



MNIST data analysis

1 Data point - 28×28 ⟶ 1D array
2 D array ↓ 784

flattening operation
1st layer flattening layer



$\boxed{\text{ANN Architecture}}$

Fully connected ANN where every datapoint is connected to other data point.

RELU $\max(x, 0)$

Softmax f^n

multi-class classification with probability distinction.

Total number of weights = $784 \times 300 + 300$ (bias units)

$= 235500$