Comp Sci, January 17, 2023

Deep learning: NN (FFNN)

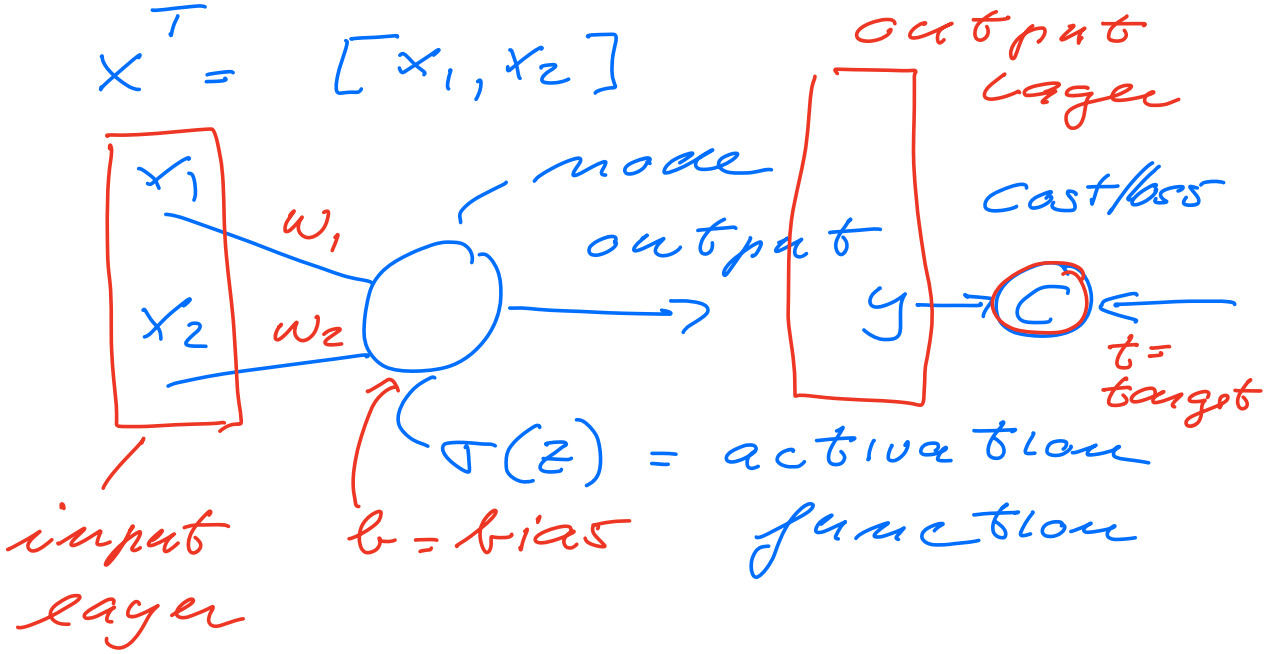
important ingredients

- universal approx theorem
- NN architecture (Model)
  - # hidden layers and #nodes/neurons
  - activation functions
    $\sigma(z)$
- cost/loss function & optimization
  - Type of loss function
  - regularization
  - Gradient methods
    - SGD, batch, epochs

Example of linear regression / classification.

Consider two inputs x_1, x_2

$$x^T = [x_1, x_2]$$



$$z = x_1 w_1 + x_2 w_2 + b$$

$$\sigma(z) = x_1 w_1 + x_2 w_2 + b = y$$

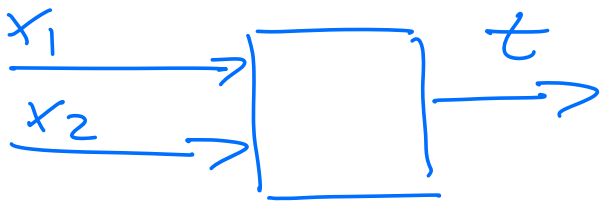
data set t_i

$$y_i = \sigma(x_i ; \Theta) = x_i^T w + b$$

$$w^T = [w_1, w_2]$$

$$\Theta = \{w, b\}$$

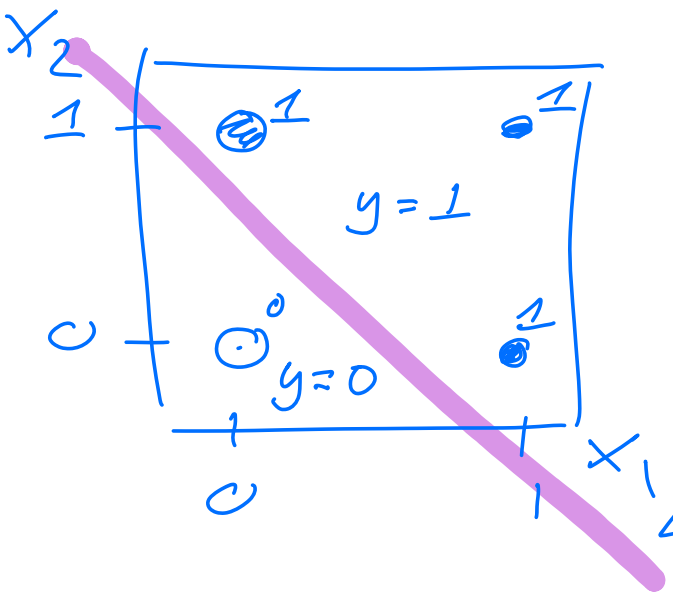
OR-gate



x_1	x_2	t
0	0	0
0	1	1
1	0	1
1	1	1

$$X^T = \{ [0,0], [0,1], [1,0], [1,1] \}$$

$$t = \{0, 1, 1, 1\}$$

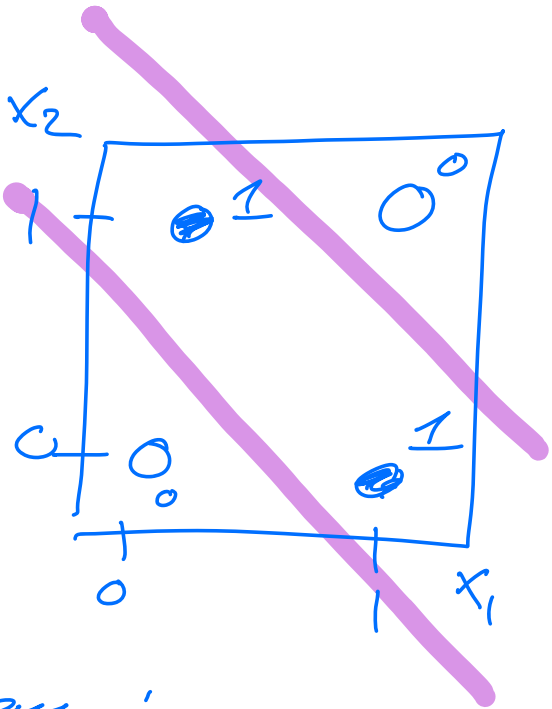


Think of a line fit

$$y = x_1 w_1 + x_2 w_2 + b$$
$$= X^T w + b$$

XOR - gate

x_1	x_2	t
0	0	0
0	1	1
1	0	1
1	1	0



Linear Regression:

$$\begin{bmatrix} 0 & 0 \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \end{bmatrix} + b = y = 0$$

$$\begin{bmatrix} 0 & 1 \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \end{bmatrix} + b = y = \underline{1}$$

$$\begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \end{bmatrix} + b = y = \underline{1}$$

$$\begin{bmatrix} 1 & 1 \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \end{bmatrix} + b = y = \underline{1}$$

$$\Theta = \begin{bmatrix} b \\ w_1 \\ w_2 \end{bmatrix}$$

$$\hat{\Theta} = (X^T X)^{-1} X^T \cdot t$$

$$X = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

$$[X^T X]^{-1} = \begin{bmatrix} 3/4 & -1/2 & -1/2 \\ -1/2 & 1 & 0 \\ -1/2 & 0 & 1 \end{bmatrix}$$

$$\hat{\Theta} = \begin{bmatrix} 1/4 \\ 1/2 \\ 1/2 \end{bmatrix}$$ OR-gate

$$y^T = [1/4 \quad 3/4 \quad 3/4 \quad 5/4]$$

$$t^T = [0 \quad 1 \quad 1 \quad 1]$$

if $y \geq 1/2$ then $y = 1$

else $y = 0$

XOR

$$\hat{\theta} = \begin{bmatrix} \frac{1}{2} \\ 0 \\ 0 \end{bmatrix}$$

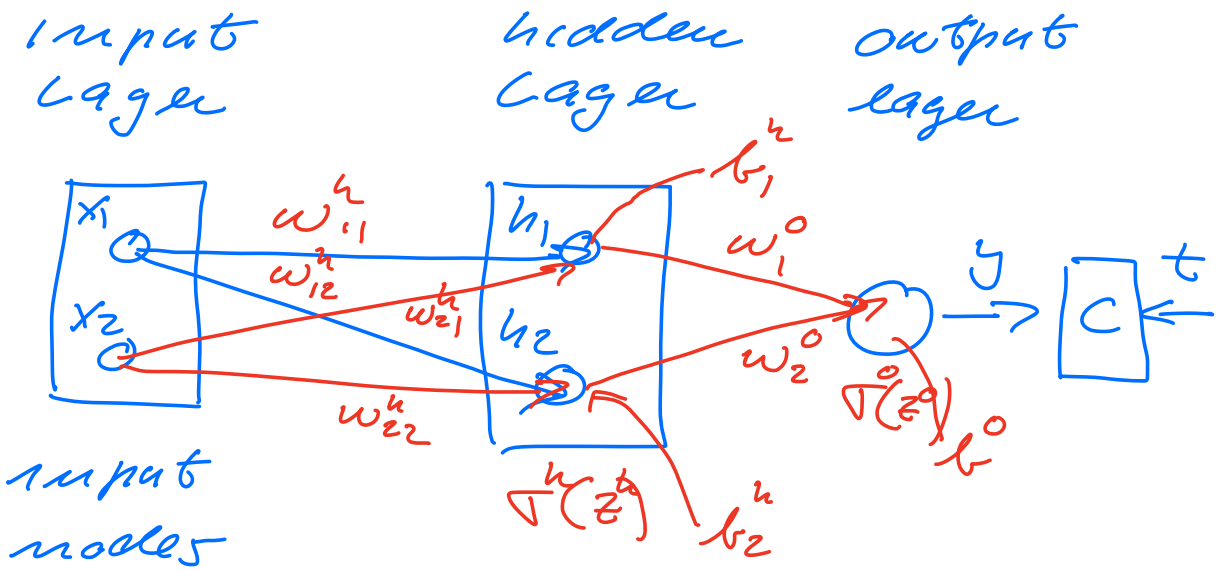
$$y^T = X\hat{\theta} = \begin{bmatrix} \frac{1}{2} & \frac{1}{2} & \frac{1}{2} & \frac{1}{2} \end{bmatrix}$$

fails $t^T = [0 \; 1 \; 1 \; 0]$

$\Rightarrow$ Single perceptron model (no hidden layer) fails in reproducing data which a non-linear description.

Adding hidden layer

XOR-gate



From input to hidden layer

$$\begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

$$w \cdot x$$

$h = \{h_1, h_2\}$ = output from hidden nodes

$$h = \sigma^h(x; W, b) = \sigma^h(x; \Theta)$$

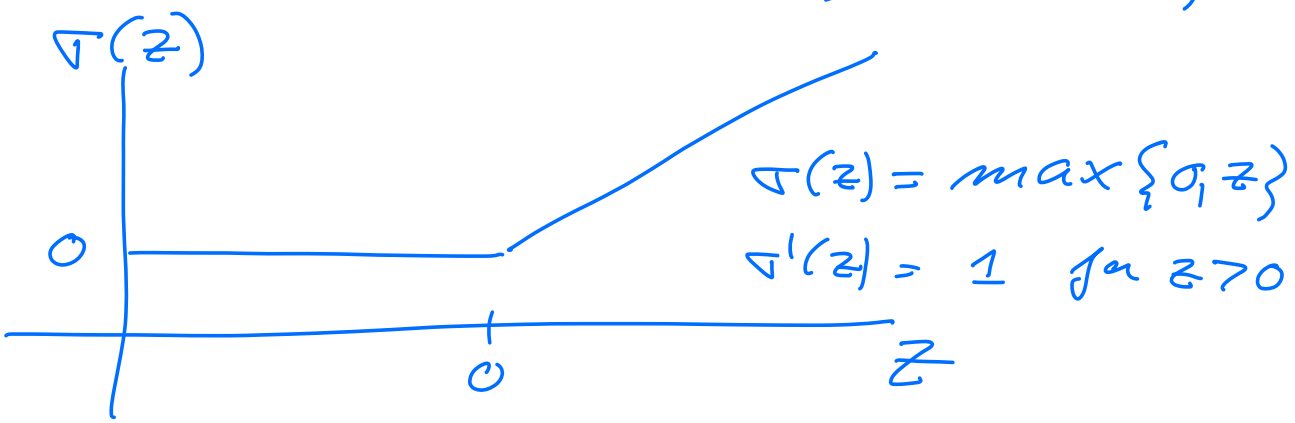
$$b^T = [b^h_1, b^h_2]$$

h needs an activation

$\sigma(z)$; rectified linear unit : ReLU

$z = W \cdot x + b$

$h = \sigma(Wx + b) = \sigma(z)$

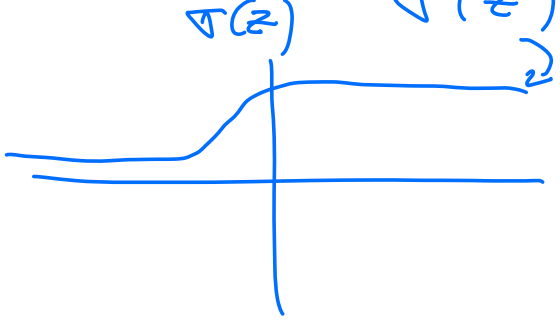


$\sigma(z) = \max\{0, z\}$

$\sigma'(z) = 1$ for $z > 0$

other activation functions

- Sigmoid $\sigma(z) = \dfrac{1}{1+e^{-z}}$



$\sigma'(z) = \sigma(z)(1-\sigma(z))$

$\sigma'(z) \approx 0 \Rightarrow$ vanishing gradients

$z \in (-\infty, \infty)$

- tanh : $\sigma(z) = \tanh(z)$

$\sigma'(z) = 1 - (\tanh(z))^2$

$z \in (-1, 1)$

- ELU : $\sigma(z) = \begin{cases} \alpha [e^{+z} - 1] & z \leq 0 \\ z & z > 0 \end{cases}$

- Leaky ReLU : $\sigma(z) = \begin{cases} \alpha \cdot z & z \leq 0 \\ z & z > 0 \end{cases}$

$\sigma'(z) = \begin{cases} \alpha \overset{\approx 10^{-2}}{} & z \leq 0 \\ 1 & z > 0 \end{cases}$

XOR - gate

From training of a neural network

$$W = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$$

$$b^{\mu} = \begin{bmatrix} 0 \\ -1 \end{bmatrix}$$

$w^0 = \begin{bmatrix} 1 \\ -2 \end{bmatrix}$ $b^0 = 0$

weights from hidden to output

$$X = \begin{bmatrix} 0 & 0 \\ 0 & 1 \\ 1 & 0 \\ 1 & 1 \end{bmatrix} \in \mathbb{R}^{4 \times 2}$$

$$X \cdot W + b = \begin{bmatrix} 0 & -1 \\ 1 & 0 \\ 1 & 0 \\ 2 & 1 \end{bmatrix}$$

$$h = \sigma(XW + b) = \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 1 & 0 \\ 2 & 1 \end{bmatrix}$$

multiply with $w^0 \Rightarrow$

$$y = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \end{bmatrix} = t = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

General FFNN and Basic algorithm

1 Feed Forward stage with a given number of hidden layers, nodes and $\sigma(z)$, with randomly initialized weights and biases

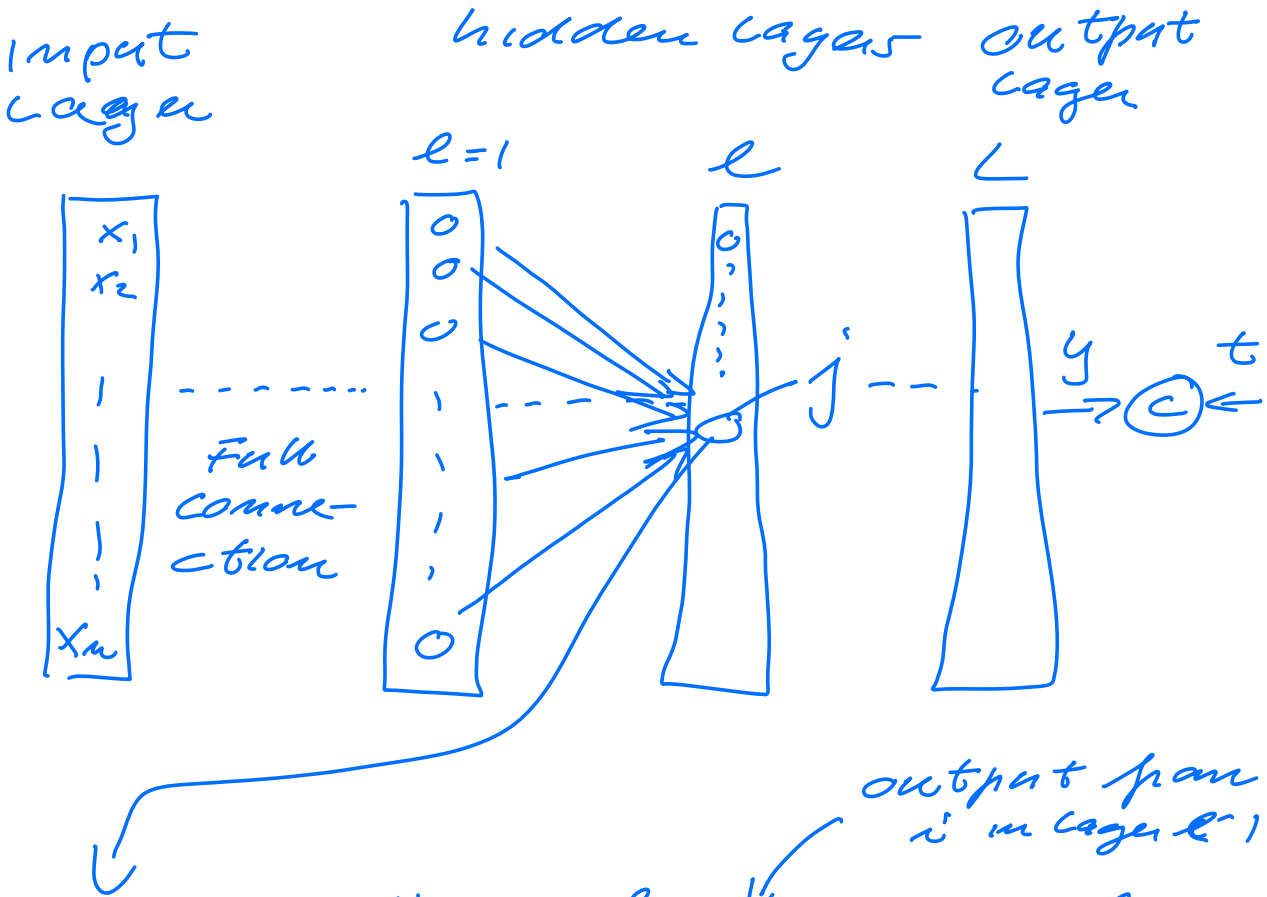
- produces an output which we feed in to a cost function and compare with the targets - t -

2 Back propagation stage (reverse mode in automatic differentiation)

- algorithm (Back propagation algo) to calculate gradients of weights & biases.
- use gradient descent to train weights & biases

3 repeat 1 and 2 till score is acceptable

Back propagation algo :



output from i in layer $l-1$

$$z_j^l = \sum_{i=1}^{N_{l-1}} w_{ij}^l a_i^{l-1} + b_j^l$$

w_{ij}^l = weights for layer l connecting with $l-1$

a_j^l = output from node j in layer l

$$a_j^l = \sigma^l(z_j^l)$$

Define a cost function.

Example is MSE

$$C(\hat{e}) = \frac{1}{2} \sum_{i=1}^{n} (y_i - t_i)^2$$

Typical activation function

$$a_j^l = \sigma^l(z_j^l) = \frac{1}{1 + e^{-z_j^l}}$$