NEURAL NETWORKS

History : Ups, downs

Impact : Image Analysis (Vision) - CNN

Sequence to sequence Translation - RNN
(Speech, Text)

Debate in AI : Model-free (Function / Curve Fitting) - Computation
Labeled data

Model-based (Represent and reason)

BASICS:

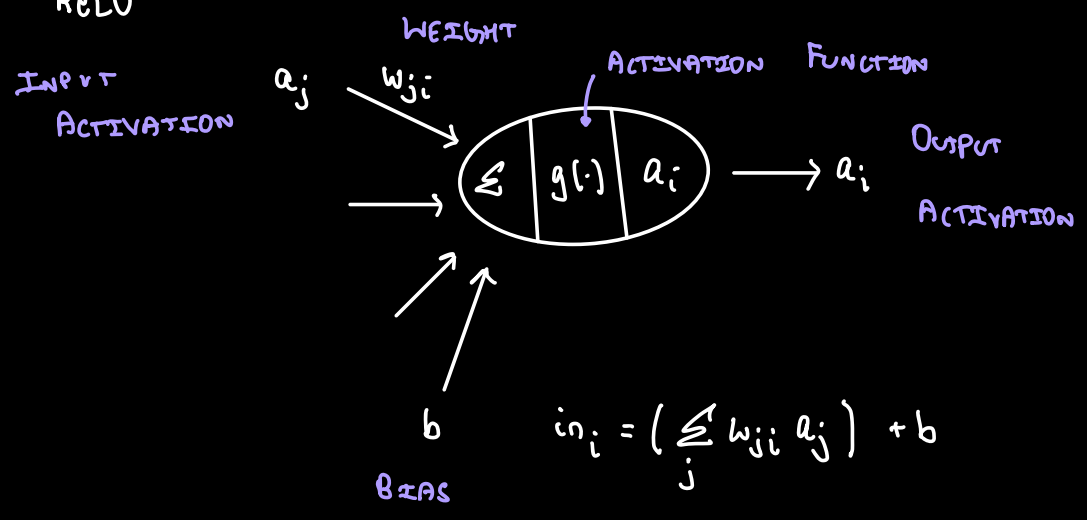
Feedforward NN - trained in supervised fashion using labeled data.

- Neuron
- Neural Networks (Syntax, Semantics)
- How they are trained

- CNN
- Perspective.

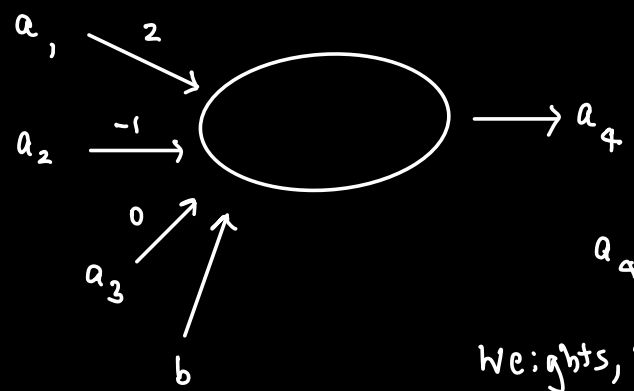
NEURONS:

- Activations (Input, Output)
- Weights
- Bias
- Activation Functions
  - Step
  - Sign
  - Sigmoid
  - ReLU



$$in_i = \left(\sum_j w_{ji}\, a_j\right) + b$$

$$a_i = g(in_i)$$

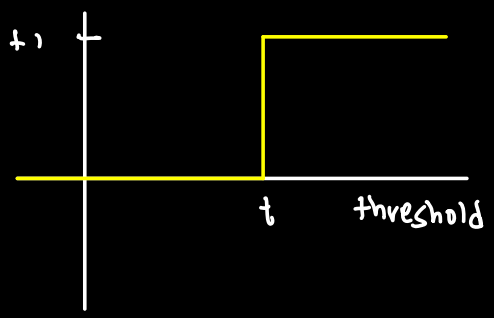


$$a_4 = g(2a_1 - a_2 + b)$$

Weights, bias are learnt.

Activation Functions:

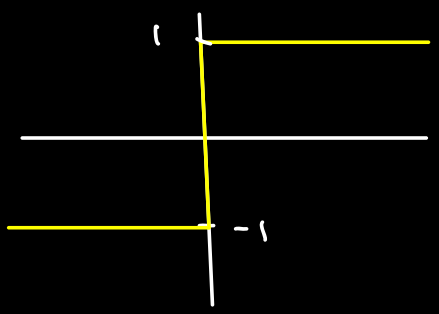
Step Function:



$g(x) = 1 \quad \text{if} \quad x \geq t$

$g(x) = 0 \quad \text{if} \quad x < t$

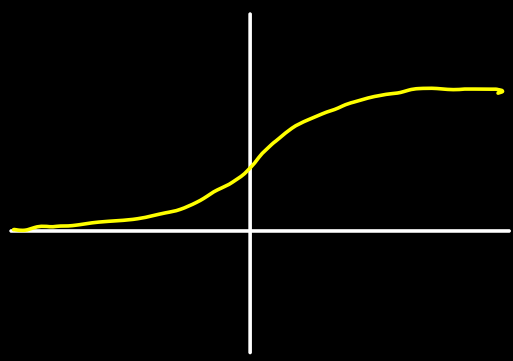
Sign Function:



$g(x) = 1 \quad \text{if} \quad x \geq 0$

$g(x) = -1 \quad \text{if} \quad x < 0$

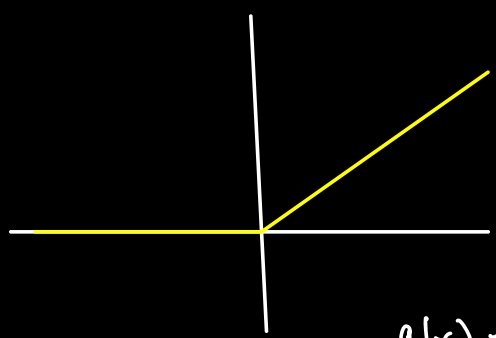
SIGMOID FUNCTION:



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

RELU FUNCTION :

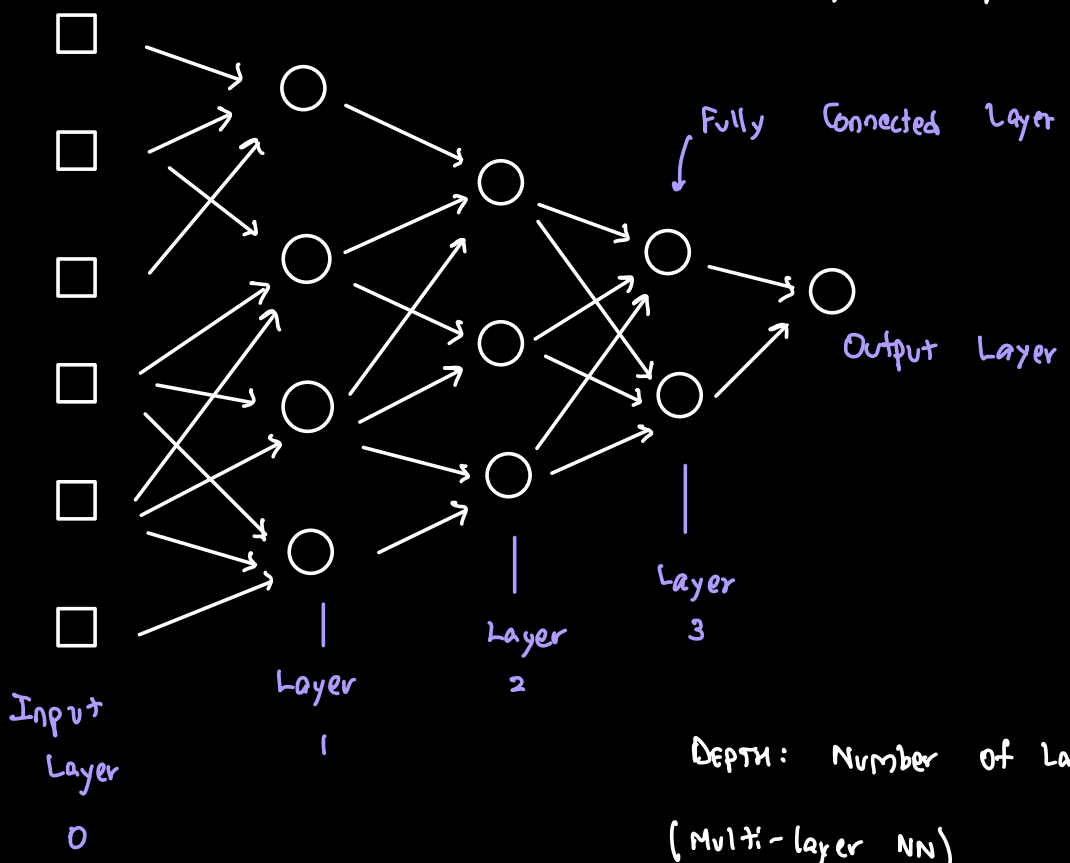
(RECTIFIED LINEAR UNIT)



$$g(x) = \max(0, x)$$

FEEDFORWARD NN:

DAG: Directed Acyclic Graph



DEPTH: Number of Layers.

(Multi-layer NN)

"Deep Learning".

→ NN are Universal Function Approximators.

$f(x_1 \dots x_6)$ and error ε → we can generate NN that generate output within error.

NEURONS WITH STEP ACTIVATION FUNCTIONS:

AND

OR

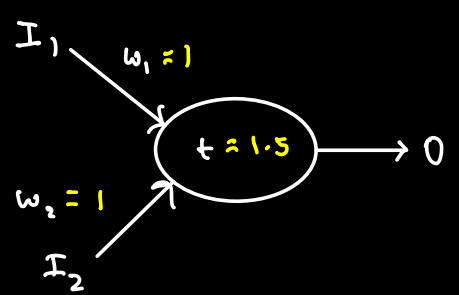
NOT

Minority

Linearly Separable Functions

AND GATE:

I_1	I_2	O
1	1	1
1	0	0
0	1	0
0	0	0

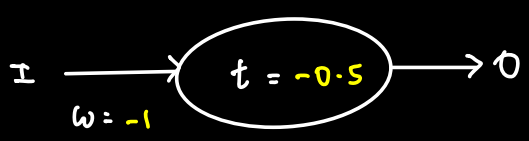


$1 \times 1 + 1 \times 1 = 2 \overset{?}{\geq} 1.5 \rightarrow 1$

$1 \times 1 + 0 \times 1 = 1 \overset{?}{\geq} 1.5 \rightarrow 0$

$0 \times 1 + 0 \times 1 = 0 \overset{?}{\geq} 1.5 \rightarrow 0$

NOT GATE:



$1 \rightarrow 0$

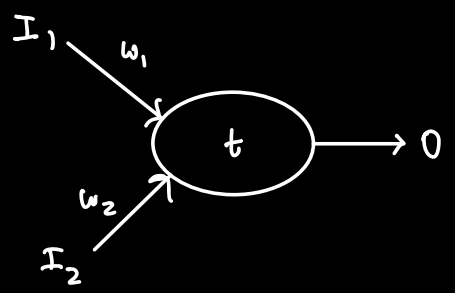
$0 \rightarrow 1$

$I = 1 \quad 1 \times -1 = -1 \geq -1/2 \quad \boxed{0}$

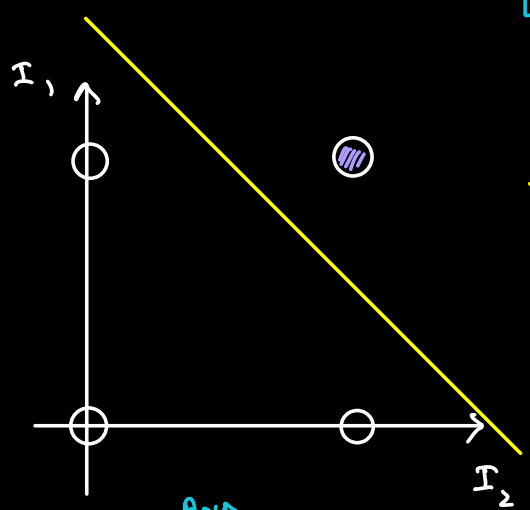
$I = 0 \quad 0 \times -1 = 0 \geq -1/2 \quad \boxed{1}$

XOR GATE:

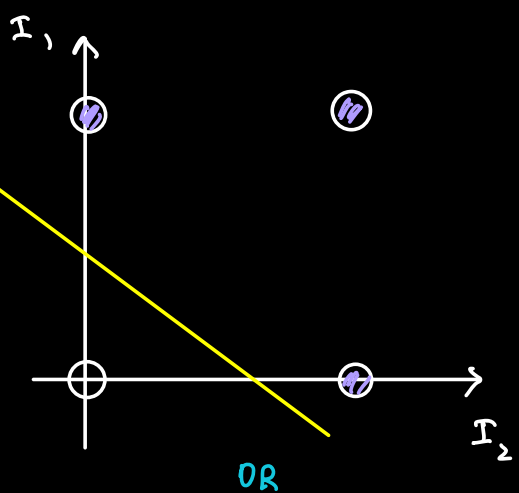
I_1	I_2	O
1	1	0
1	0	1
0	1	1
0	0	0



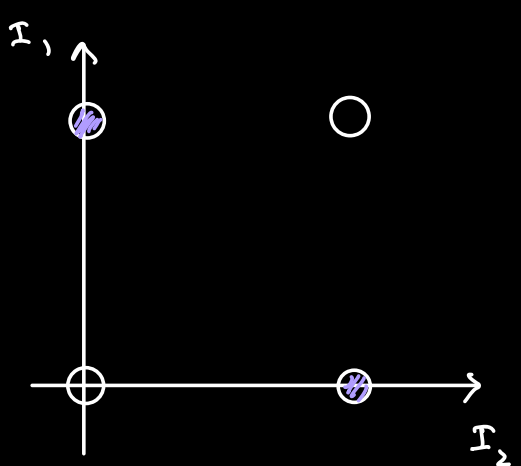
LINEARLY SEPARABLE FUNCTION



AND



OR

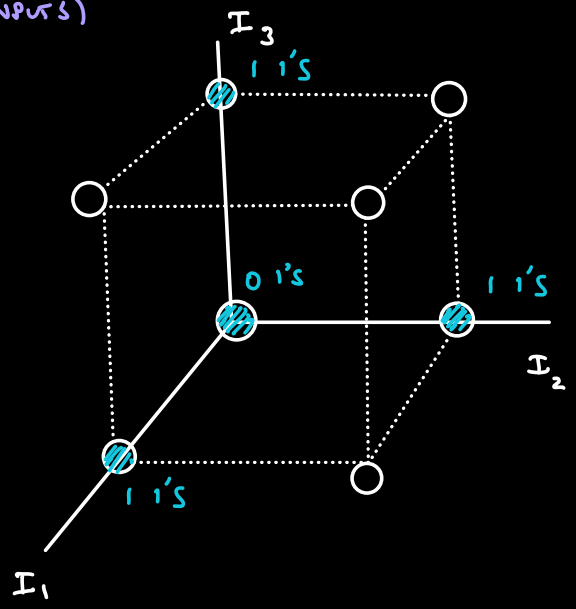


XOR

Not linearly separable → cannot form a neuron, Bunch of neurons needed.

MINORITY FUNCTION:

(3 INPUTS)

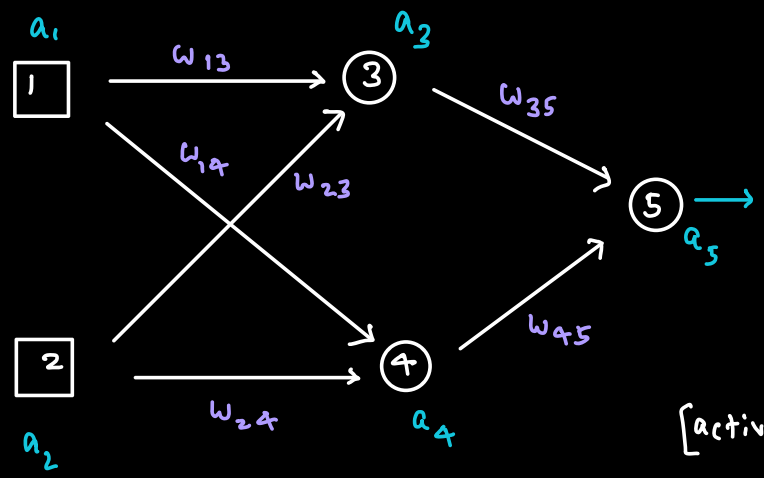


If 0/1 one, then 1 is minority.

We can draw a plane to separate

↳ Linearly Separable!

NEURAL NETWORK AS A FUNCTION:



[activation function g]

$$a_5 = g(w_{35} a_3 + a_4 w_{45})$$

$$= g(w_{35}\, g(w_{13} a_1 + w_{23} a_2) + w_{45}\, g(w_{14} a_1 + w_{24} a_2))$$

$$a_5 = f(\underbrace{a_1, a_2}_{\text{Input}}, \underbrace{w_{13}, w_{14}, \ldots, w_{45}}_{\text{Weight}})$$

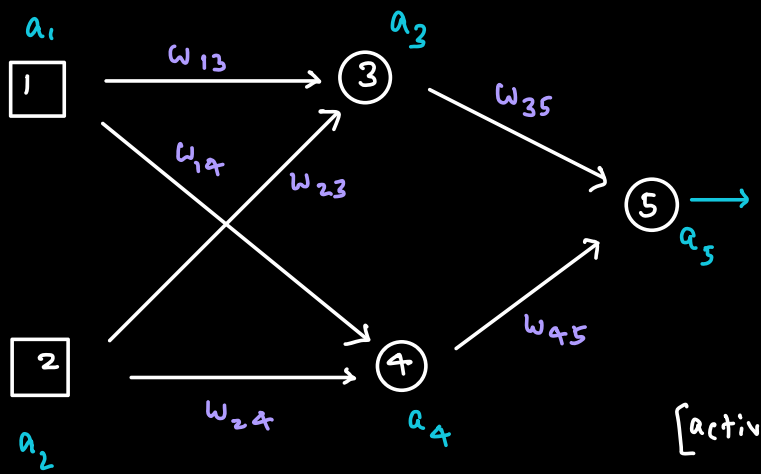
COMPLEX NON-LINEAR FUNCTION

Value for a_1, a_2 (i)

(1) $a_5 = f_1(w_{13}, w_{14}, \dots, w_{45})$

(2) $a_5 = f_2(w_{13}, w_{14}, \dots, w_{45})$

TRAINING NEURAL NETWORK



[activation function g]

$$a_5 = g(w_{35} a_3 + a_4 w_{45})$$

$$= g(w_{35}\, g(w_{13} a_1 + w_{23} a_2) + w_{45}\, g(w_{14} a_1 + w_{24} a_2))$$

DATASET:

	a_1	a_2	a_5	Data Label
I_1	:	:	:	L_1
I_2	:	:	:	L_2
:	:	:	:	:

Loss Function:

(Optimize)

CE : Cross Entropy

MSE : Mean - Square error.

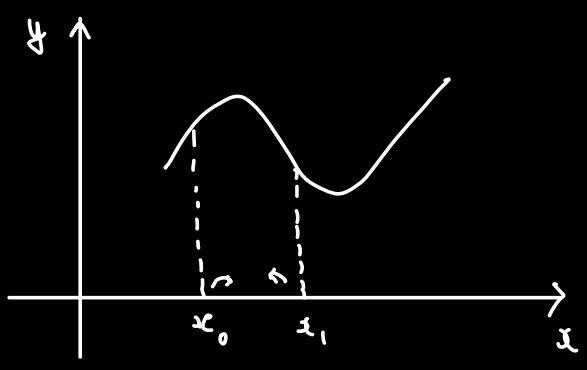
$$MSE = \frac{1}{N} \sum_{i=1}^{N} \underbrace{(NN(I_i) - L_i)^2}_{f_i(w_1 \dots w_k)}$$

N → number of examples

MSE is a function of weights.

So choose weights that minimize MSE!

Gradient Descent:



$y = f(x)$

* $\frac{\partial f}{\partial x}$ (Partial Derivative)

* Step

$$y = f(\bar{w}_1, \bar{w}_2, \cdots, \bar{w}_k)$$

Loss function

Gradient: $\left(\partial f/\partial w_1, \partial f/\partial w_2, \cdots, \partial f/\partial w_k\right)$

→ Adam Optimizer.

(Optimization)

How to check if it is optimized?

....	...
	yes
	no

2000
↓
1600 - train
400 - test

So train and then test on testing data.

Say Accuracy = 98%

Performance Metric

Loss Function

Performance Metric

- Epoch
- Batch → Stochastic Gradient Descent (SGD)
  → Parallel Execution
- Train / Validate / Test (data)
- Stopping Criterion

Data → Training 80% (Optimize loss)
Data → Testing 20% (Performance Metric)

Epoch:

One pass through data!

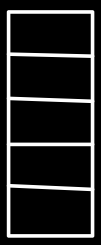
loss metric

batch 1 ← epoch 1 .....
epoch 2 .....
⋮

epoch k ..... Stop

Divide Data into batches **randomly**.

training data



→ batch (64 examples)

↳ BATCH SIZE

Apply one step of gradient descent on one batch

↳ 1 Epoch : all the batches run once.

WHEN TO STOP ?

→ Number of epochs - No

→ As long as loss decreases - No - Overfitting.

Training → Training 80 % → Validation 20 %

} For each epoch, loss from training, metric from validation.

loss metric → Validation.

- epoch 1
- epoch 2
  ⋮
- epoch $_k$ Stop

Stop when metric stops improving or decreases.
(Patience factor - say no improvement for 10 epochs - stop)

Validation
Testing

} Metric measured but validation helps in training.