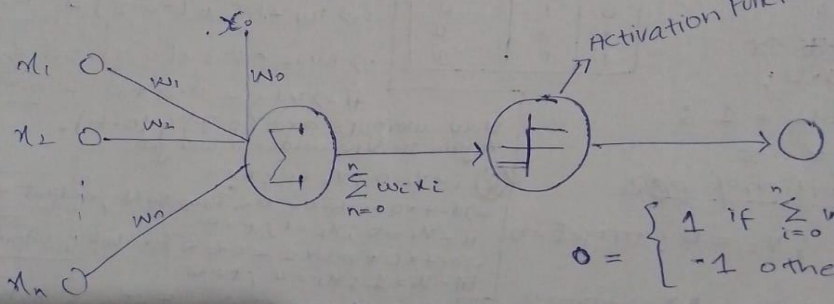
Perceptron And Training Rule (ANN)

→ Unit used to build ANN system.

⇒ It takes vector inputs, calculates Linear combination of inputs, then outputs a '1' if result is greater than threshold eg -1 otherwise

$$o(x_1, \ldots x_n) = \begin{cases} 1 & \text{if } w_0 + w_1 x_1 + w_2 x_2 + \cdots + w_n x_n > 0 \\ -1 & \text{otherwise} \end{cases}$$

→ 'w_i' is a Constant or weight that determines the Contribution of input x_i to the Perceptron output.



$$o = \begin{cases} 1 & \text{if } \sum_{i=0}^{n} w_i x_i > 0 \\ -1 & \text{otherwise} \end{cases}$$

1 way to learn an acceptable weight vector is to begin with random weights, then Iteratively apply perceptron to each training example, modifying the perceptron weights whenever it misclassify an example.

This Process is repeated as needed.

Weights are modified at each step according to the Perceptron training Rule, which revises the weight w_i associated with input x_i according to rule.

$$w_i \leftarrow w_i + \Delta w$$

Where

$$\Delta w = \eta (t - o) x_i$$

(t: target, o: Predicted, η: Learning rate, x_i: input)

AND GATE USING PLR

Let us First choose random weights

$w_1 = 0.2$ & $w_2 = 0.6$;

Threshold = 1 ;

& $\eta = 0.5$

A	B	A x B
0	0	0
0	1	0
1	0	0
1	1	1

1st Iteration of PLR

i- $x_1 = 0$; $x_2 = 0$; target = 0

$w_1 x_1 + w_2 x_2 \Rightarrow 0.2 \times 0 + 0.6 \times 0$

$\Rightarrow 0$ < threshold

So output is '0'

ii- $x_1 = 0$; $x_2 = 1$; target = 0

$\Rightarrow 0.2 \times 0 + 0.6 \times 1 \Rightarrow 0.6 < 1$

output = 0

iii- $x_1 = 1$; $x_2 = 0$; $t = 0$

$\Rightarrow 0.2 \times 1 + 0.6 \times 0 = 1.2 > 1$

output = 1 ✗

Update weights :-)

$w_1 = w_1 + \Delta w_1$ — (i)

$w_2 = w_2 + \Delta w_2$ — (ii)

$\Rightarrow \Delta w_1 = 0.5 \times (0 - 1) * 1$

$\Delta w_1 = -0.5$

i $\Rightarrow w_1 = 1.2 + (-0.5) = 0.7$

$\Rightarrow \Delta w_2 = 0.5 \times (0 - 1) \times 0 \Rightarrow 0$

ii $\Rightarrow w_2 = 0.6 + 0 = 0.6$

New weights are (0.7) & (0.6).

Again Iterate with new weights.

(2) i- $x_1 = 0$; $x_2 = 0$; $t = 0$

$\Rightarrow 0.7 \times 0 + 0.6 \times 0 = 0 <$ Threshold ; output = 0

ii- $x_1 = 0$; $x_2 = 1$; $t = 0$

$\Rightarrow 0.7 \times 0 + 0.6 \times 1 = 0.6 <$ Threshold ; output = 0

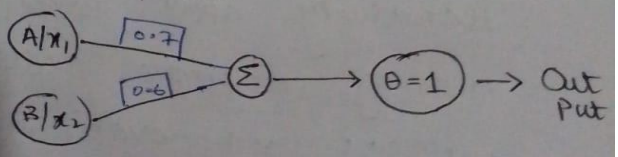
iii- $x_1 = 1$; $x_2 = 0$; $t = 0$

$\Rightarrow 0.7 \times 1 + 0.6 \times 0 \Rightarrow 0.7 <$ Thresh output = 0

iv- $x_1 = 1$; $x_2 = 1$; $t = 1$

$\Rightarrow 0.7 \times 1 + 0.6 \times 1 \Rightarrow 1.3 >$ Threshold ;

So these are final weights & outputnetwork :) output = 1



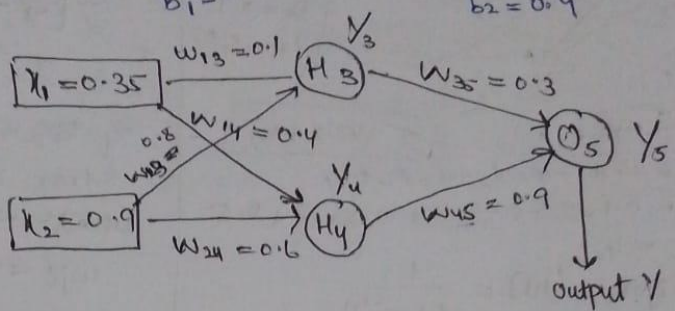
ANN

Example

Activation fun = sigmoid

Y actual = 0.5

$\eta = 0.5$



1

① Forward pass

$\Rightarrow H_3 = w_{13} \times x_1 + w_{23} \times x_2 + 0.5$

$H_3 = 0.1 \times 0.35 + 0.8 \times 0.9 + 0.5$

$H_3 = 1.255$

$Y_3 = \frac{1}{1+e^{-1.255}}$

$\boxed{Y_3 = 0.7781}$

$\Rightarrow H_4 = w_{14} \times x_1 + w_{24} \times x_2 + 0.5$

$= 0.4 \times 0.35 + 0.6 \times 0.9 + 0.5$

$H_4 = 1.18$

$Y_4 = \frac{1}{1+e^{-1.18}}$

$\boxed{Y_4 = 1.44359}$

* Here calculation mistake "0.7649" is correct.

$\Rightarrow O_5 = w_{35} \times Y_3 + w_{45} \times Y_4 + b_2$

$= 0.3 \times 0.7781 + 0.9 \times 1.4435 + 0.4$

$= 1.93258$

$Y_5 = \frac{1}{1+e^{-1.93258}}$

$\boxed{Y_5 = 0.8735}$ } Predicted output.

$\therefore$ ERROR $= Y_t - Y_5$

$= 0.5 - 0.8735$

$\boxed{\text{ERROR} = -0.3735}$

- Backward pass

② ① Compute error at Y_5 & Y_4 & Y_3 which will be δ_5, δ_4 & δ_3 respectively.

$\therefore \Delta w_{ji} = \eta \delta_j O_i$

* if j is output unit

$\boxed{\therefore \delta_j = O_j(1-O_j)(t_j - O_j)}$

* if j is hidden unit

$\boxed{\therefore \delta_j = O_j(1-O_j) \sum_k \delta_k w_{kj}}$

① For Y_5, δ_5?

$\delta_5 = O_5(1-O_5)(t_0 - O_5)$

$= 0.8735(1-0.8735)(0.5 - 0.8735)$

Mistake of sign here it's -ive

$\boxed{\delta_5 = 0.04825}$

① For H_4? $\delta_4 = Y_4(1-Y_4)(w_{45} \times \delta_5)$

$\delta_4 = 1.4435(-0.4435)(0.043425)$

$\boxed{\delta_4 = -0.02780}$

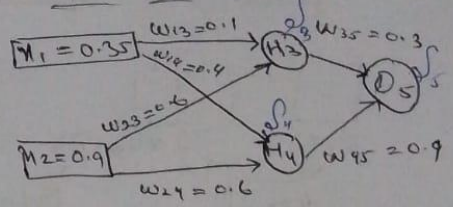
③ Updating weights

$\delta_3 = 0.00249$; $Y_3 = 0.7781$

$\delta_4 = -0.02780$; $Y_4 = 1.44359$

$\delta_5 = 0.04825$; $Y_5 = 0.8735$

Old weights



∴ Formula

$w_{ji} = w_{ji} + \Delta w_{ji}$

$\Delta w_{ji} = \eta \delta_j O_i$

$\Delta w_{45} = \eta \delta_5 O_4 = 0.5(0.04825)(1.44359)$

$\Delta w_{45} = 0.034$

$w_{45} = 0.9 + 0.034$

$\boxed{w_{45} = 0.9348}$

$\Delta w_{35} = \eta \delta_5 O_3 \Rightarrow 0.5(0.04825 \times 0.7781)$

$= 0.01877$

$w_{35} = 0.3 + 0.01877$

$\boxed{w_{35} = 0.3187}$

$\Delta w_{24} = \eta \delta_4 X_2 = 0.5(-0.02780 \times 0.9)$

$\Delta w_{24} = -0.01251$

$\boxed{w_{24} = 0.587}$

$\Delta w_{14} = \eta \delta_4 X_1 = 0.5(0.02780 \times 0.35)$

$\Delta w_{14} = -0.004865$

$\boxed{w_{14} = 0.3951}$

$\Delta w_{13} = \eta \delta_3 X_1 = 0.5(0.00249 \times 0.35)$

$\Delta w_{13} = 0.000435$

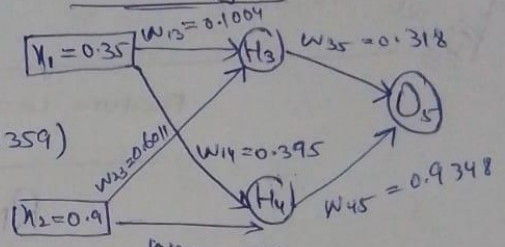
$\boxed{w_{13} = 0.10043}$

$\Delta w_{23} = \eta \delta_3 X_2 \Rightarrow 0.5 \times 0.00249 \times 0.9$

$\Delta w_{23} = 0.0011205$

$\boxed{w_{23} = 0.60112}$

New weights



"AGAIN FORWARD PASS WITH NEW WEIGHTS"

$\Rightarrow H_3 = w_{13} \times X_1 + w_{23} \times X_2 + b_1$

$H_3 = 1.07613$

$O_3 = Sig(H_3) \Rightarrow \boxed{0.7449}$

$\Rightarrow H_4 = w_{14} \times X_1 + w_{24} \times X_2 + b_1$

$H_4 = 1.166$

$O_4 = Sig(H_3) = \boxed{0.7624}$

$\Rightarrow H_5 = w_{35} \times O_3 + w_{45} \times O_4 + b_2$

$H_5 = 1.34956$

$\Rightarrow O_5 = Sig(H_5) \Rightarrow \boxed{0.7940}$

NEW ERRORS

$\delta_5 = O_5(1 - O_5)(t_5 - O_5)$ } error at output layer (t_5 = actual)

$\boxed{\delta_5 = -0.04808}$

$\delta_4 = O_4(1 - O_4)(\delta_5 W_{45})$

$\boxed{\delta_4 = 0.13445}$

$\delta_3 = O_3(1 - O_3)(\delta_5 W_{35})$ } error at Hidden layer Perceptron Neuron

$\boxed{\delta_3 = 0.0479}$

& TO BE CONTINUE FOR NEW WEIGHTS & NEW ITERATIONS

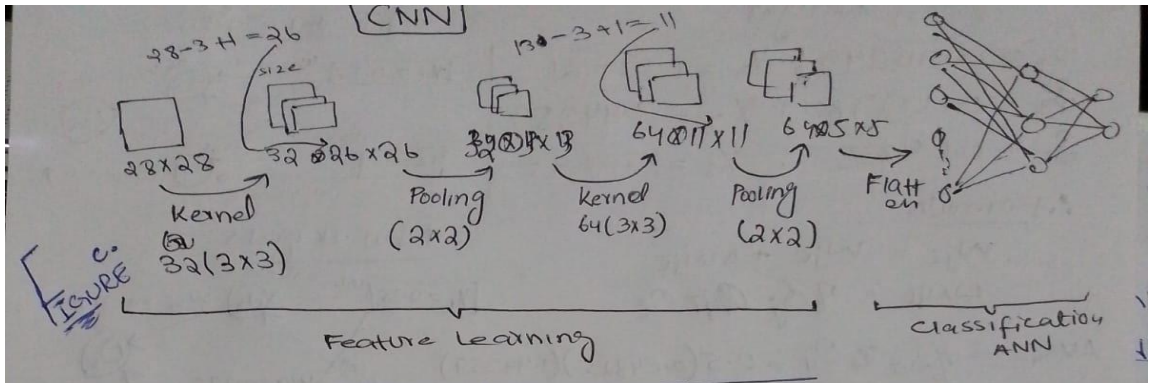


FIGURE C

RNN

Machine Learning → Feed Forward Neural Network ANN → CNN → RNN → LSTM

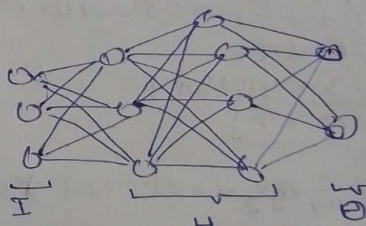
↓ Feature Extract (small data ok)

Problem

→ When DS is huge/ large # of features, manual FE is time consuming & difficult.

Solution

→ ANN
→ Auto FE
→ Activation Function, Bias, weight
→ Called Universal Function Approximation as can learn any non-linear function.



Problem of ANN.

→ Image Recognition
→ Speech Recognition
→ Huge input params 248x248

Solution

→ CNN
→ Apply Convolutions to Reduce the size of input, as shown in FIGURE C.

Problem (2)

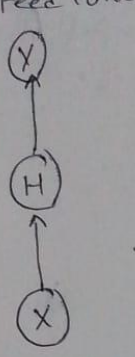
→ No sequence Capture
→ Language Translation no memory

Solution

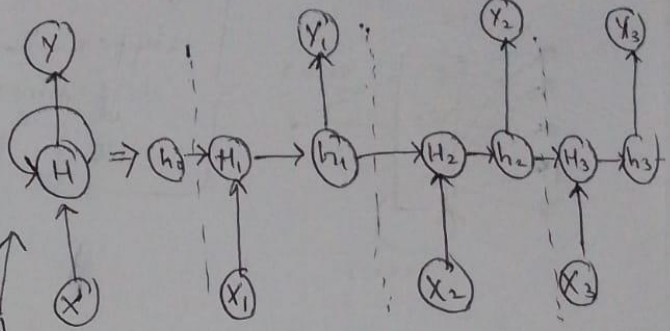
→ RNN

⇒

ANN
Feed Forward Network



RNN



Input Format For RNN:-

(timesteps, Input_Features)

→ If we input review 1

Review

[[10000][01000][00100]]

(3, 5)

no of timesteps

# of features

Review	Sentiment
movie was good	1
movie was bad	0
movie was not good	0

movie was good bad

[10000][01000][00100][00010]

[00001]

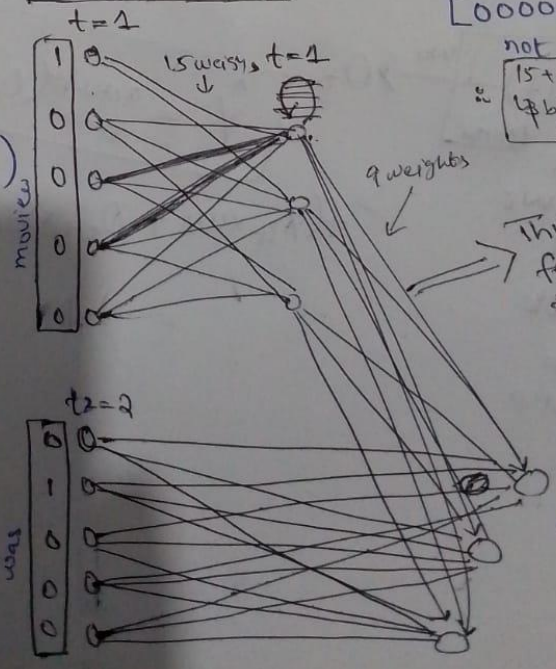
→ ARCHITECTURE

→ For 2nd (3, 5)

→ For 3rd (4, 5)

as 4 word so 4 # of timesteps

ANN needs input of same length but RNN can also handle diff



∴ not 15+9+3 = 27 weights 4 biases from 4 nodes Total 31 params

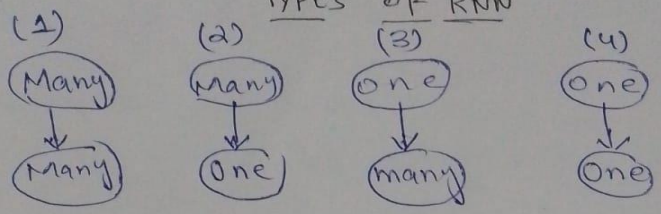
This is memory from previous time_step

output

3 weights

∴ At one timestep only one input come

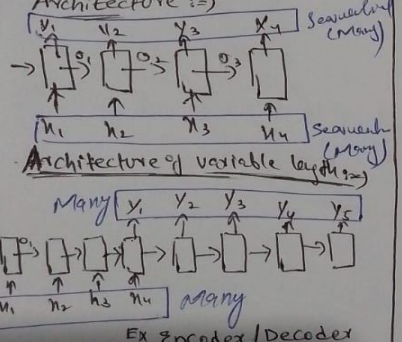
TYPES OF RNN



(1) Many To Many

Example :-
→ Language Translation
Also Called Seq2Seq Model.
Same
NER
Variable
Summarization

Architecture :=)



Ex Encoder / Decoder

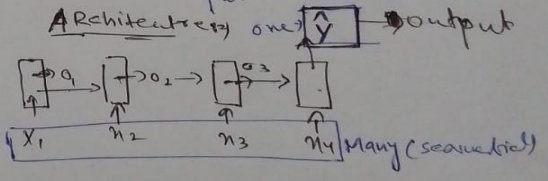
(2) Many To One

input → many i-e: sequence, chars or some.
&
output → 1 i-e:

1) Example :-
input is Review of movies
Output is Rate out of (5).

2) Example :-
input → Tweet
Output → sentiment.

Architecture :=)

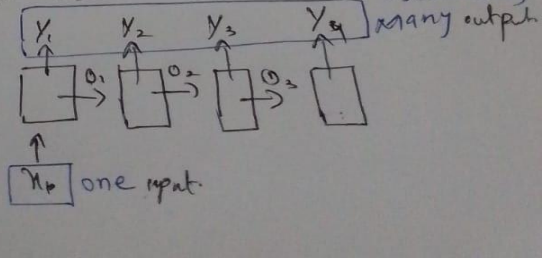


(3) One To Many

Single input & many output (sequential)

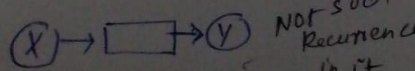
Ex :- → Image Captioning.

Architecture of One To Many :=)



(4) One Two One

Ex :- image classification



Not such Recurrence in it.