RNN (Recurrent Neural Network)

-> A type of sequential model to work on sequential data.

Q. What is sequential and Non-sequential data.

Non-sequential data

dataset

iq	marks	gender	placement

↳ In this dataset, while feeding data to ANN, sequence does-not matter, output doesn't depend on sequence.

Sequential data

↳ Ex. text dataset

correct = "Hyy, my name is parth !"

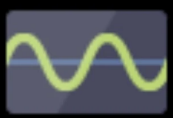
wrong = "Parth...! Hyy my is name"

} Sequence matters...!!

More ex:

- Time series based data

- Speech



- DNA sequence

-> Hence RNNs are highly useful in NLP (Natural Language Processing) tasks.

Q. Why RNN....?

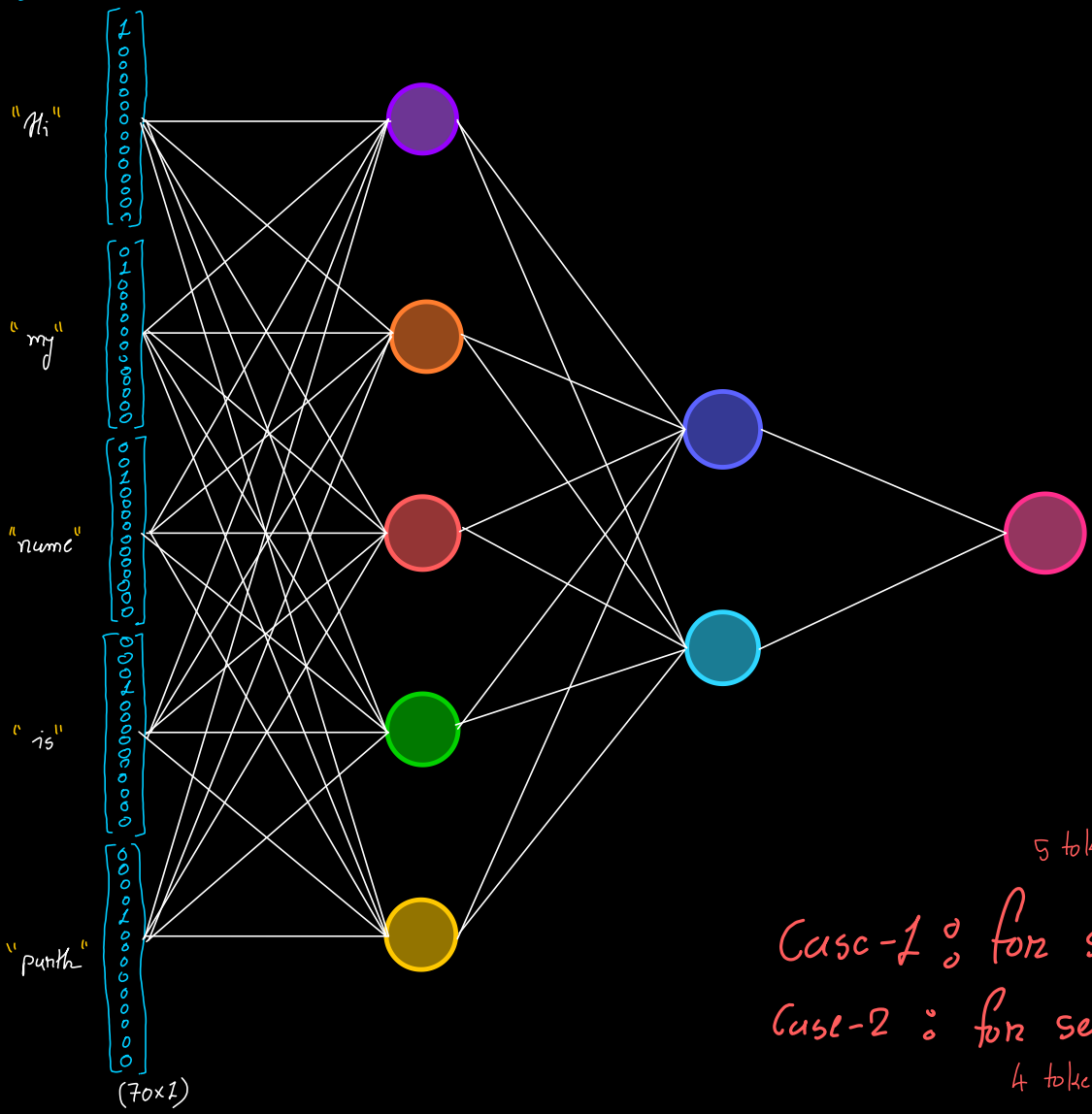
Input	Output	
Hi my name is punth	0	(5)
I love to learn AI/ML	0	(5)
India won the match	1	(4)

total 14 tokens

Vectorize formate

[1,0,0,0,0,0,0,0,0,0,0,0,0,0] → "Hi"

[0,0,0,0,0,0,1,0,0,0,0,0,0,0] → "love"



5 token = 70 i/p

Case-1 : for sentence-1

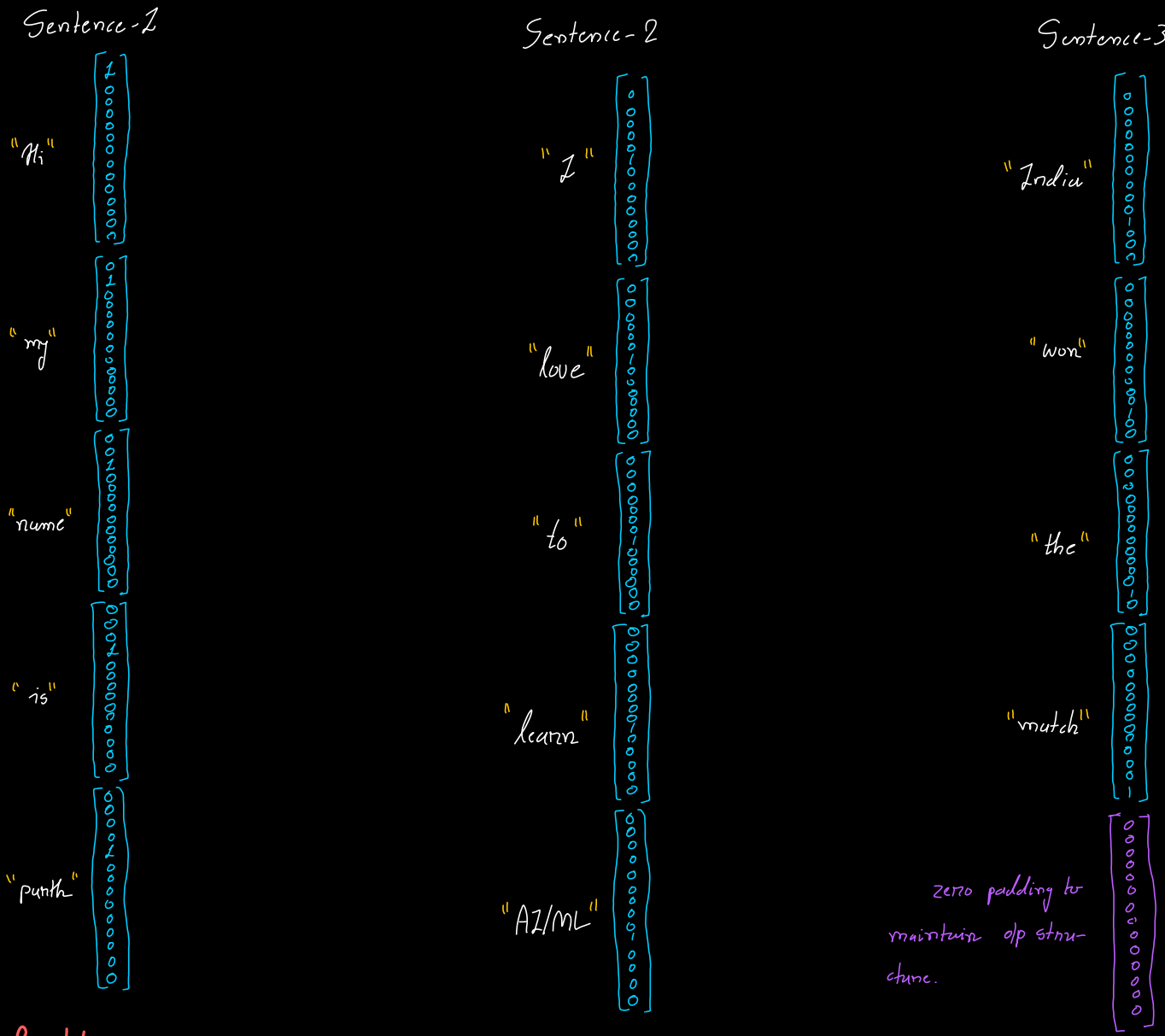
Case-2 : for sentencce-3

4 tokens = 56 i/p

70x5 = 350 weights

⚠ i/p is changing which is not good

Due to this mismatch of sentences, we will use zero padding technique.



Problems

- Varying text input size
- Unnecessary computation
- Prediction problem.
- Doesn't follows sequence during input.

ANNs are not capable to remember the sequence of incomin -g data. hence we need RNN. because it works on sequenc -e based data.

text2data.com/Demo

prugnakalp.com/demos/BERT-NLP-QnA-Demo/

-> ANN works on fixed i/p size while RNN any length of sequential data.

-> Sequence of data contains some meaning, in ANN while feeding data, i/p data runs forward together, which means there is no sequence will be followed.

-> RNNs have memory feature which means RNNs can remember the past i/p.

-> I/p data have specific formate while feeding it to the network.

form = (timesteps, i/p_features)

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

unique words (vocabulary)

[movie, was, good, bad, not]

[1,0,0,0,0] = "movie"

[0,1,0,0,0] = "was"

[0,0,1,0,0] = "good"

[0,0,0,1,0] = "bad"

[0,0,0,0,1] = "not"

Review_1 (Movie was good)

[[1,0,0,0,0],[0,1,0,0,0],[0,0,1,0,0]]

t=1 t=2 t=3

timesteps = words

(3,5)

No. of timesteps

No. of i/p features (each word can be represented as 5 numbers)

Review_2 (Moview was bad)

[[1,0,0,0,0],[0,1,0,0,0],[0,0,0,1,0]] → (3,5)

Review_3 (Movie was not good)

[[1,0,0,0,0],[0,1,0,0,0],[0,0,0,0,1],[0,0,1,0,0]] → (4,5)

data input → format = (timesteps, i/p_features)

while using keras (SimpleRNN class) ↓

formate = (Batch size, timesteps, i/p_features) — ... reviews can send.

i/p shape = (3, 4, 5) = 3D tensor

- 3 → Number of reviews
- 4 → Max size of review
- 5 → No. of i/p features (OneHotEncoded representation) — each word is represented as 5 numbers.

Q. How RNN works....?

data

	Review				Sentiment
	x_{11}	x_{12}	x_{13}		
x_1	movie	was	good		1
	x_{21}	x_{22}	x_{23}		
x_2	movie	was	bad		0
	x_{31}	x_{32}	x_{33}	x_{34}	
x_3	movie	was	not	good	0

Numeric representation

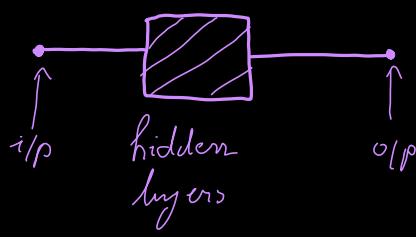
movie [1, 0, 0, 0, 0]

was [0, 1, 0, 0, 0]

good [0, 0, 1, 0, 0]

bad [0, 0, 0, 1, 0]

not [0, 0, 0, 0, 1]



→ I/p will be pass to RNN based on time

time (t)	input
0	x_{11}
1	x_{12}
2	x_{13}

ANN

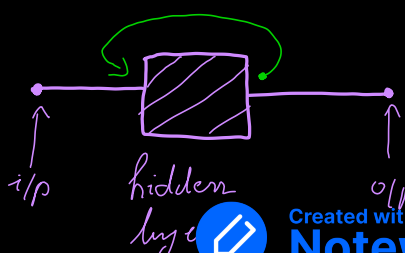
→ like ANN, we don't feed entire data at once.

→ A feed forward NN. (data flows from i/p to o/p direction).

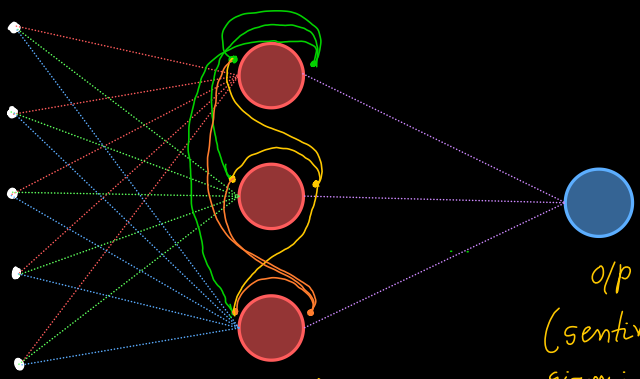
RNN

→ I/p will be given to RNN based on **time steps**.

→ Hidden layer sends feedback on backward direction, which is the major difference between ANN & RNN.



Structure of RNN

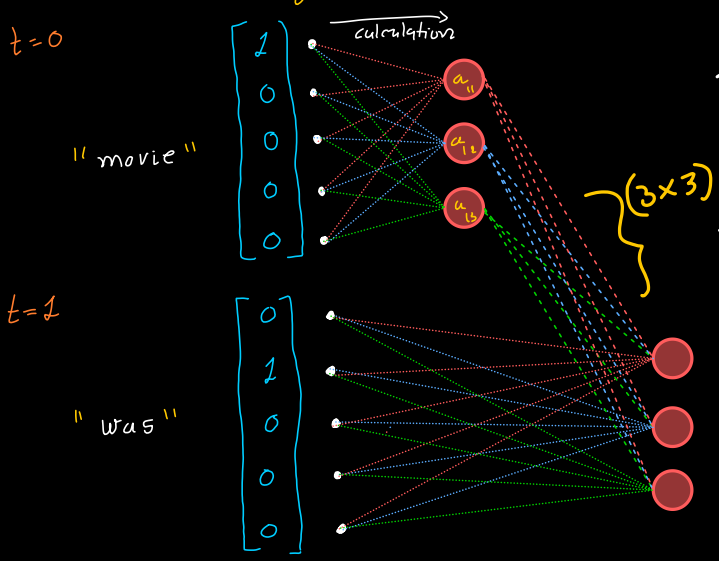


o/p
(sentiment)
sigmoid activation

i/p

-> o/p of each node will be i/p for each of them.

(Because, each token is represented as total 5 binary values)



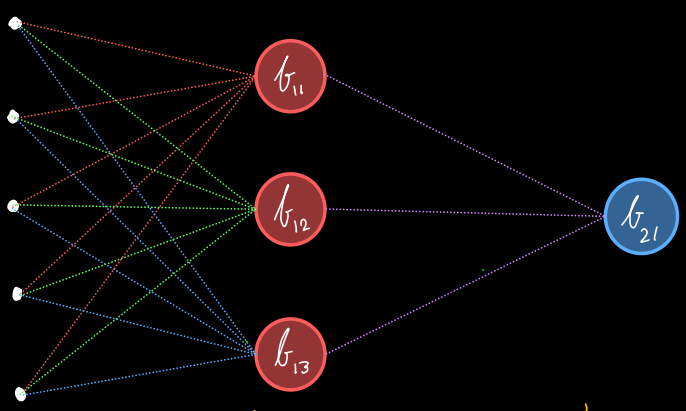
number of weights

i/p to first hidden layer

3 bias & (5,3) = 15 weights

first hidden layer to o/p layer

(3,1) = 3 weights
+1 bias



5×3 + 3×3 + 3×1

15 + 9 + 3 = 27 weights + 1 bias

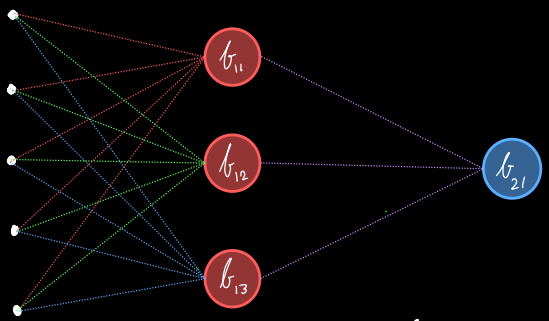
Q. forward Propagation

review	sentiment
x_{11} x_{12} x_{13}	1
x_{21} x_{22} x_{23}	0
x_{31} x_{32} x_{33} x_{34}	0

Where each of x_{ij} is 5D vector

$$[1,0,0,0,0]$$

1×5



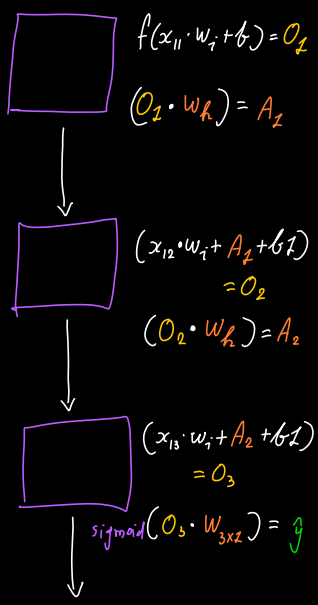
Default activation of each neurons will be tanh.

- dot-product (x_{11}, w_i) where x_{11} is 1×5 and w_i is 5×3, hence o/p will be 1×3.
- the o/p of this dot product will be given to an activation function $f(x_{11} \cdot w_i + b1) = O_1$.
- till now, the process was for $t = 0$.
- Now for $t = 1$
- i/p = x_{12}
- in between $O_1 \cdot W_h$ (1×3 · 3×3) will be work as response for hidden layer. → W_{hidden}
- final calculation would be $f(x_{12} \cdot w_i + O_1 w_h + b1) = O_2$

1×5 5×3 1×3 3×3 3×1

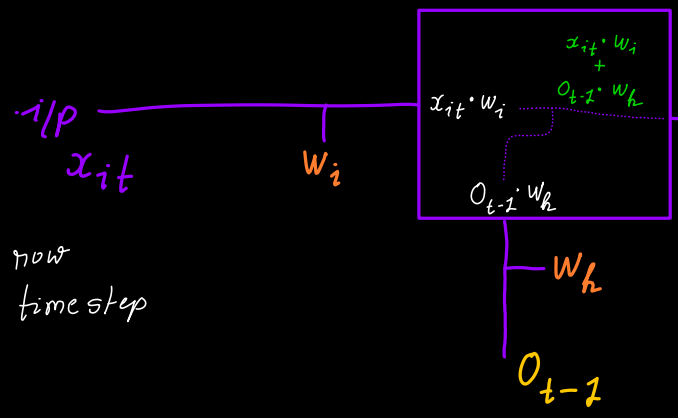
1×3 1×3

$(1 \times 3 + 1 \times 3) = 1 \times 3$



- Now for $t = 2$.
- i/p = x_{13}
- in between $O_2 \cdot W_h$ (1×3 · 3×3)
- final calculation $f(x_{13} \cdot w_i + O_2 \cdot w_h + b1) = O_3$
- Now we will move forward to next layer & multiply O_3 (1×3) with weights of next layer W_k (3×1) hence $O_3 \cdot W_k = \hat{y}$ (1×3 · 3×1 = 1×1) now sigmoid $(\hat{y})$ will give final result.

Simplified Representation



W_o if O_t is last than we will multiply it with weights of last layer.

Now $O_t \cdot W_o$ will be pass to activation function g.

$\hat{y} = g(O_t \cdot W_o)$

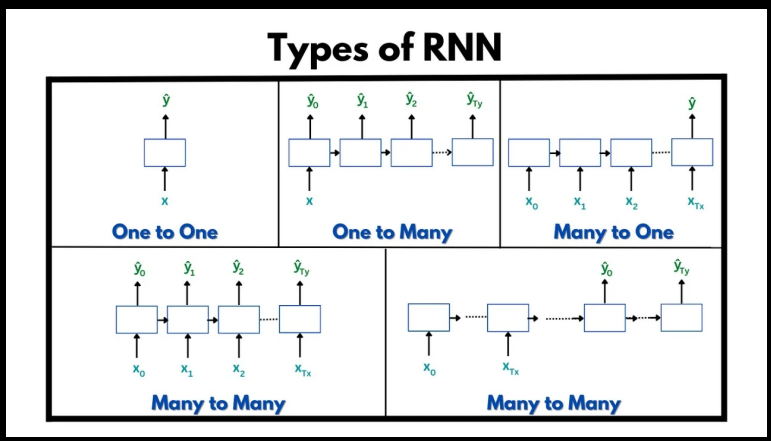
where g may be linear, softmax, sigmoid.

i = row

t = time step

Types of RNNs.

1. One-to-Many
2. Many-to-one
3. Many-to-many
4. One-to-One.



1. Many-to-one

→ input : sequence (sentence, characters, time series).

→ output : Non-sequential output. (Number/Scalar).

2. One-to-many

→ Input : Non-sequential (Numbers/Image)

→ Output : Sequential (sentence, characters, time series)

Input = once, Output = multiple times

Example : Image captioning

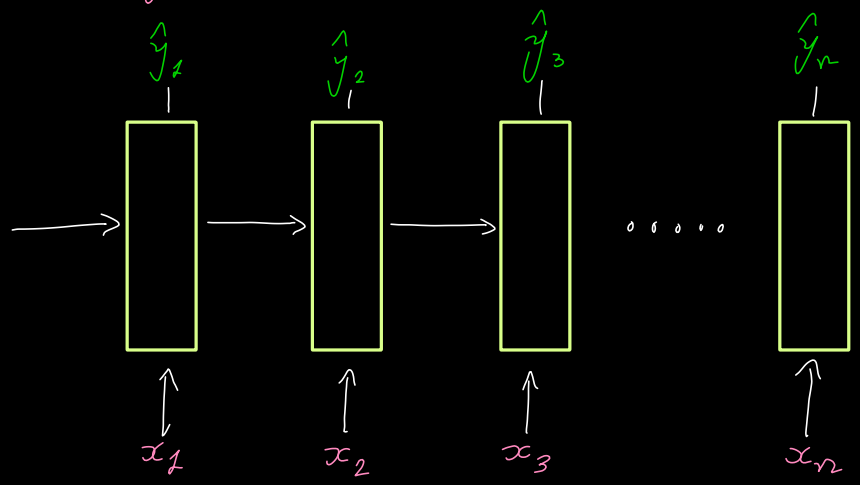
3. Many-to-Many

→ input : Sequential data } Seq 2 Seq.

→ Output : sequential data

Seq 2 Seq has 2 types.

↳ Same length [i/p seq. size = o/p seq. size] ex: Named entity recog.



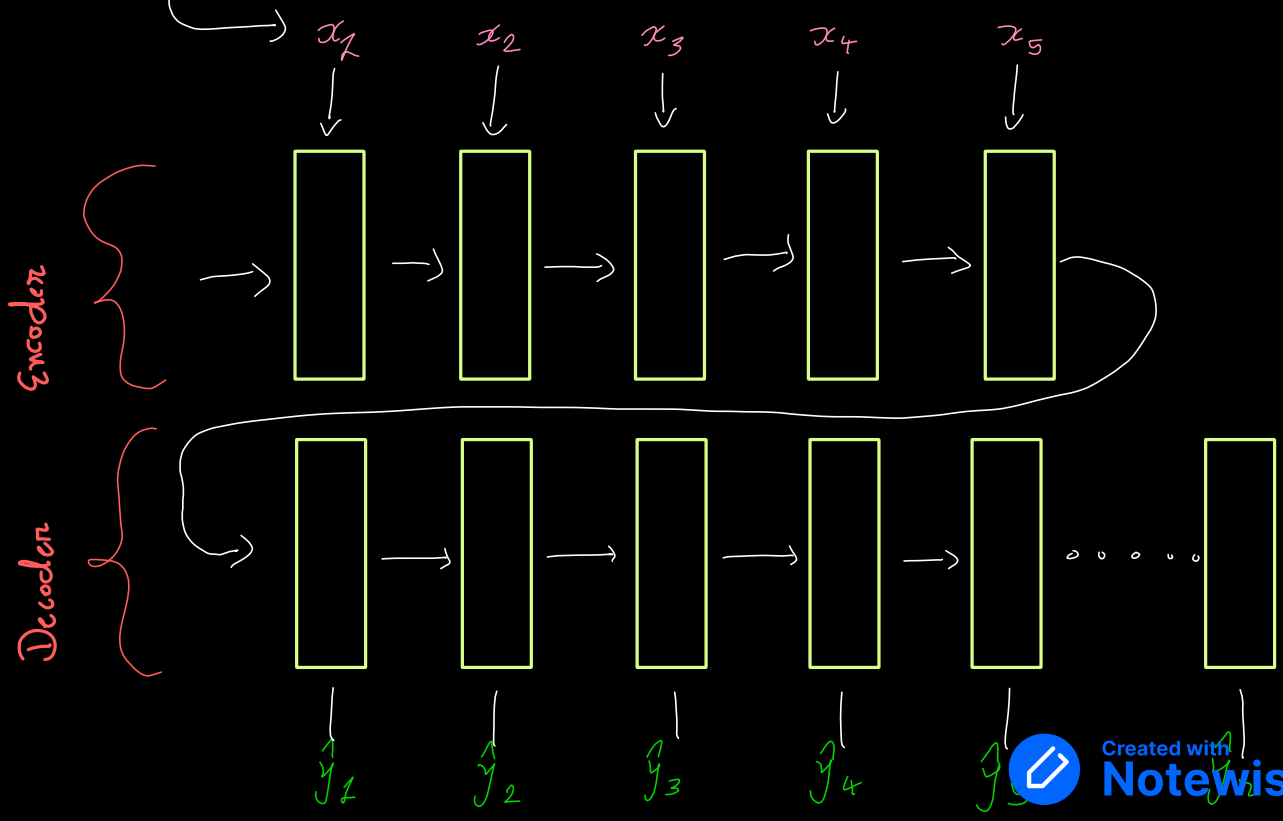
↳ Variable length [Machine translation]

- language translation models.

Example

i/p : Hi, my name is parth.

o/p : ગાર્ગ નામ પાર્થ છે.



3 One-to-One

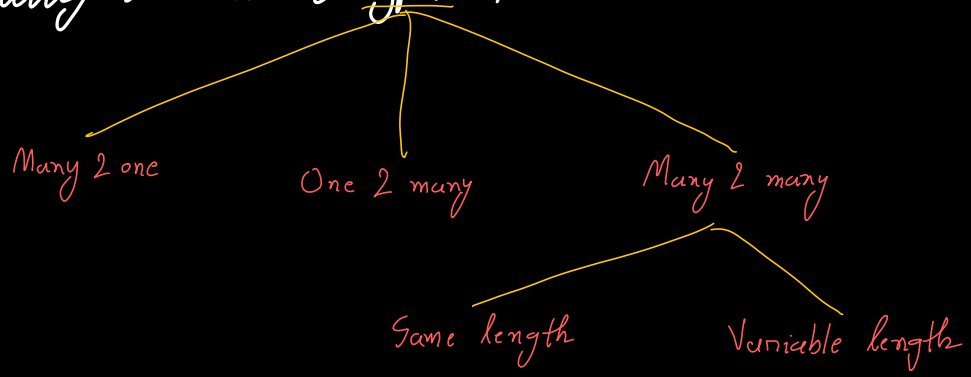
→ input : Non-Sequential

→ output : Non-Sequential

ex. image classification

→ It looks like ANN or CNN.

Hence, technically there is 3 types of RNN.



Backpropagation in RNN

→ Backpropagation in RNN is called as BPTT (Back Propagation Through Time).

Example text

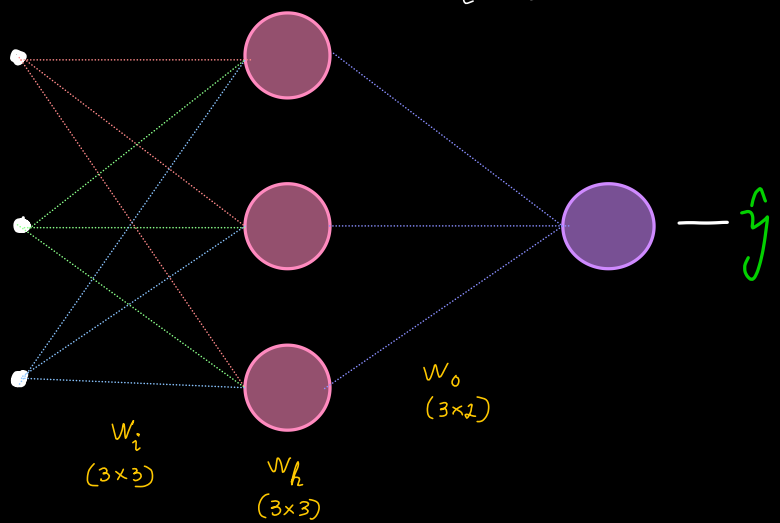
cat	mat	mat	1
mat	mat	mat	1
mat	mat	cat	0

Vocabulary

cat mat mat

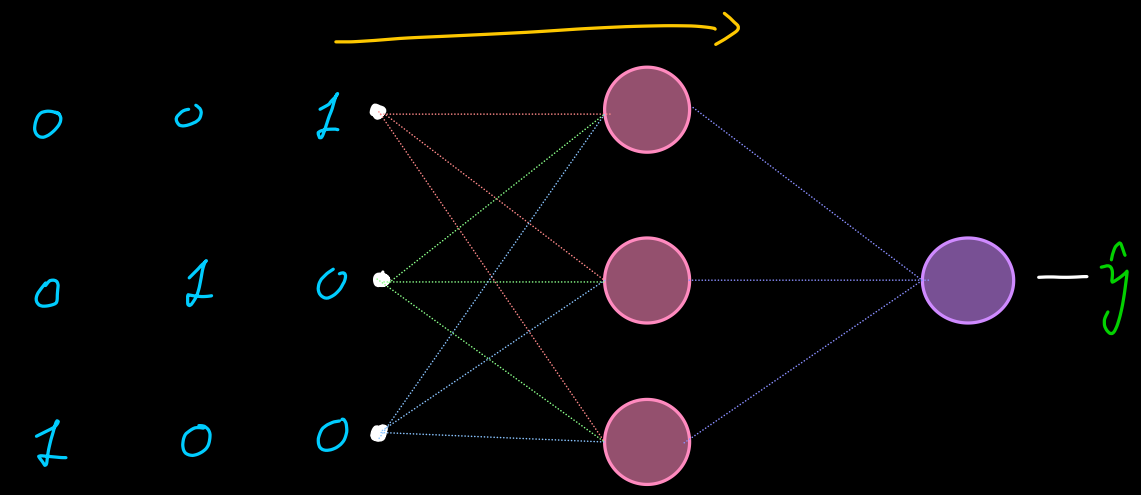
[100] [010] [001]

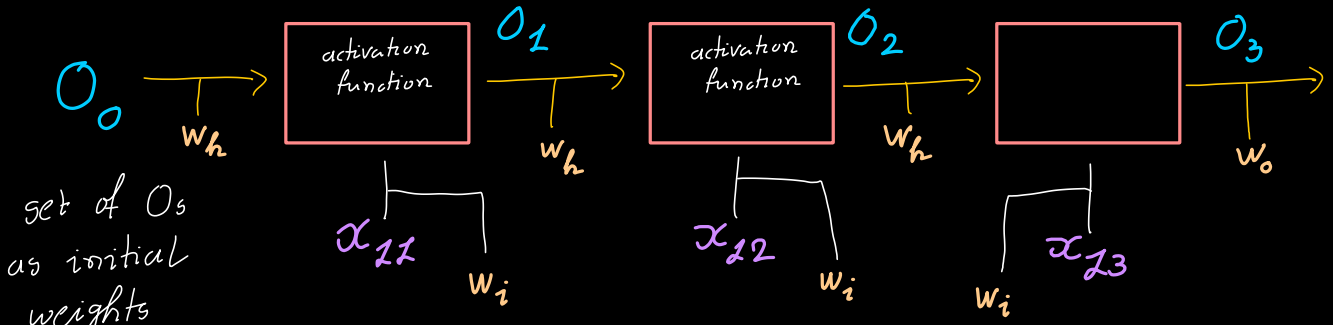
	x			y
x_1	[100]	[010]	[001]	1
x_2	[001]	[001]	[010]	1
x_3	[010]	[010]	[100]	0



forward propagation for....

x_1 [100] [010] [001]





set of 0s as initial weights

where:

$O_1 = f(x_{11} w_i + O_0 w_h)$

$O_2 = f(x_{12} w_i + O_1 w_h)$

$O_3 = f(x_{13} w_i + O_2 w_h)$

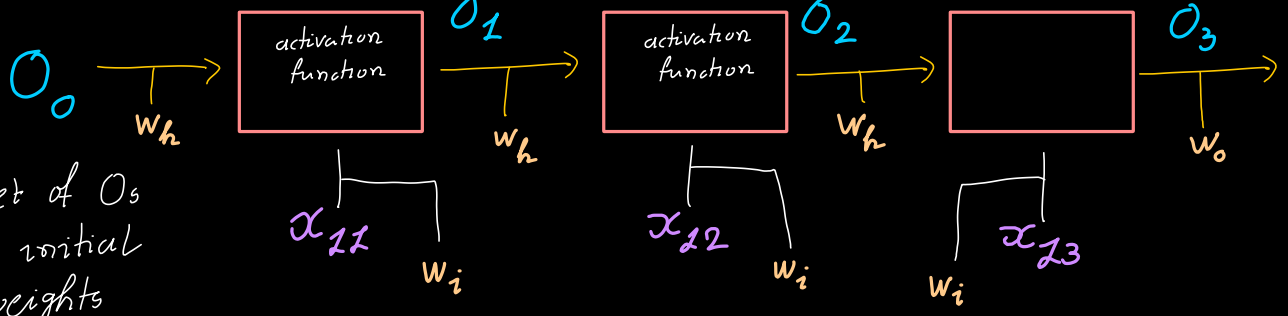
$\hat{y} = \sigma(O_3 w_o)$

$loss = -y_i \log \hat{y}_i - (1-y_i) \log(1-\hat{y}_i)$

→ Till now we have done forward propagation, now we need to find best optimal value of w_i, w_h and w_o

$$w_i = w_i - \alpha \frac{\partial L}{\partial w_i} \quad | \quad w_h = w_h - \alpha \frac{\partial L}{\partial w_h}$$

$$w_o = w_o - \alpha \frac{\partial L}{\partial w_o}$$



set of O_s as initial weights

where:

$$O_1 = f(x_{11} w_i + O_0 w_h)$$

$$O_2 = f(x_{12} w_i + O_1 w_h)$$

$$O_3 = f(x_{13} w_i + O_2 w_h)$$

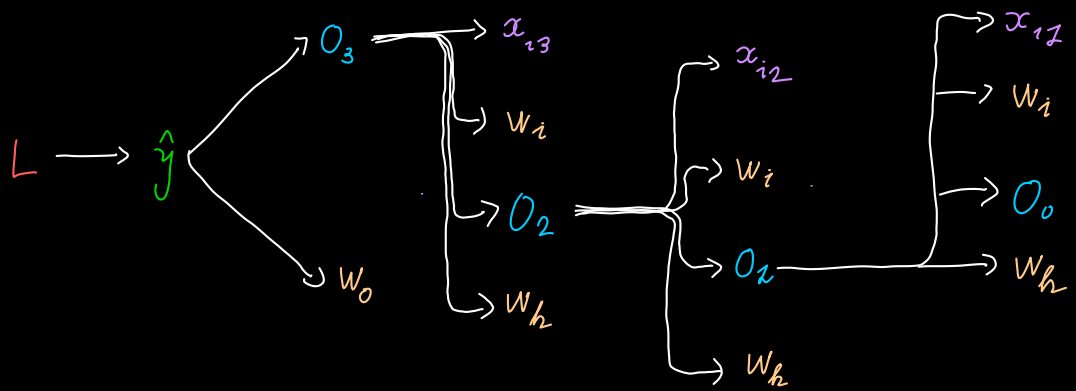
$$\hat{y} = \sigma(O_3 w_o)$$

$$\frac{\partial L}{\partial w_o}, \frac{\partial L}{\partial w_i}, \frac{\partial L}{\partial w_h}$$

$$\frac{\partial L}{\partial w_o} = L \to \hat{y} \to O_3, w_o$$

$$= \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial w_o}$$

full path



for $\frac{\partial L}{\partial w_i}$ we have 3 different paths.

$L \to \hat{y} \to O_3 \to w_i$ $\quad \frac{\partial L}{\partial w_i} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial O_3} \frac{\partial O_3}{\partial w_i}$

$L \to \hat{y} \to O_3 \to O_2 \to w_i$ $\quad \frac{\partial L}{\partial w_i} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial O_3} \frac{\partial O_3}{\partial O_2} \frac{\partial O_2}{\partial w_i}$

$L \to \hat{y} \to O_3 \to O_2 \to O_1 \to w_i$ $\quad \frac{\partial L}{\partial w_i} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial O_3} \frac{\partial O_3}{\partial O_2} \frac{\partial O_2}{\partial O_1} \frac{\partial O_1}{\partial w_i}$

hence, final path is...

$$\frac{\partial L}{\partial w_i} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial O_3} \frac{\partial O_3}{\partial w_i} + \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial O_3} \frac{\partial O_3}{\partial O_2} \frac{\partial O_2}{\partial w_i} + \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial O_3} \frac{\partial O_3}{\partial O_2} \frac{\partial O_2}{\partial O_1} \frac{\partial O_1}{\partial w_i}$$

$$\frac{\partial L}{\partial w_i} = \sum_{j=1}^{3} \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial o_j} \frac{\partial o_j}{\partial w_i}$$

: where 3 is number of words

for $j=1$: $\frac{\partial L}{\partial w_i} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial o_1} \frac{\partial o_1}{\partial w_i}$

$\hookrightarrow \frac{\partial \hat{y}}{\partial o_1} = \frac{\partial \hat{y}}{\partial o_3} \frac{\partial o_3}{\partial o_2} \frac{\partial o_2}{\partial o_1}$

for $j=2$

$\frac{\partial L}{\partial w_i} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial o_2} \frac{\partial o_2}{\partial w_i}$

$\hookrightarrow \frac{\partial \hat{y}}{\partial o_2} = \frac{\partial \hat{y}}{\partial o_3} \frac{\partial o_3}{\partial o_2}$

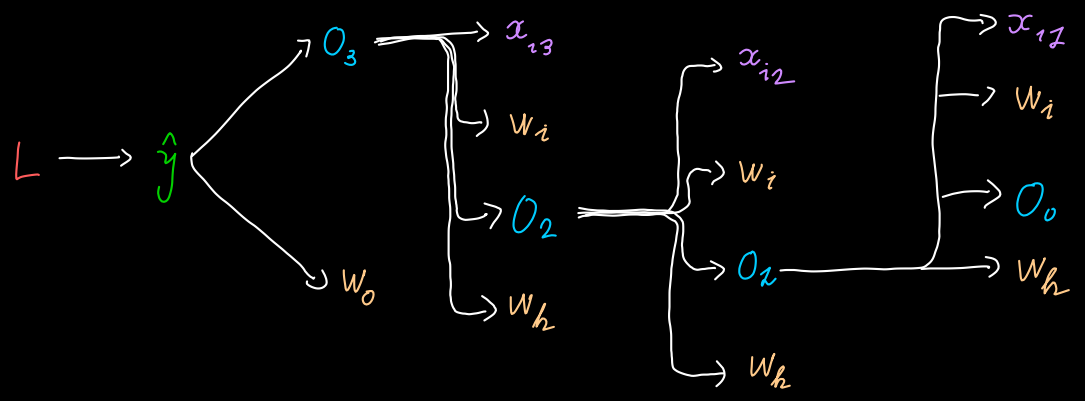
for $j=3$

$\frac{\partial L}{\partial w_i} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial o_3} \frac{\partial o_3}{\partial w_i}$

$\hookrightarrow \frac{\partial \hat{y}}{\partial o_3}$

hence $\frac{\partial L}{\partial w_i} = \sum_{j=1}^{n} \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial o_j} \frac{\partial o_j}{\partial w_i}$

full path



for $\frac{\partial L}{\partial W_h}$ we have two different paths.

$L \to \hat{y} \to O_3 \to W_h$

$L \to \hat{y} \to O_3 \to O_2 \to W_h$

$L \to \hat{y} \to O_3 \to O_2 \to O_1 \to W_h$

hence.....

$$\frac{\partial L}{\partial W_h} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial O_3} \frac{\partial O_3}{\partial W_h} + \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial O_3} \frac{\partial O_3}{\partial O_2} \frac{\partial O_2}{\partial W_h} + \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial O_3} \frac{\partial O_3}{\partial O_2} \frac{\partial O_2}{\partial O_1} \frac{\partial O_1}{\partial W_h}$$

$$\frac{\partial L}{\partial W_h} = \sum_{j=1}^{n} \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial O_j} \frac{\partial O_j}{\partial W_h}$$

Where n = number of timesteps / words.

Problems with RNNs.

- RNN suffers from few problems, hence they are not widely used at that level.
- Hence we uses some other architectures like LSTMS.
- Problems
  1. Problem of long term dependency.
  2. Stagnated training

  } Unstable gradients

- RNN is useful for sequential data. where each term is depend on its previous term.
- If the sequence / sentence gets too longer than the last term can't remember the initial term. which is like short term memory loss.
- Example
  1. gujarati is spoken in gujarat.

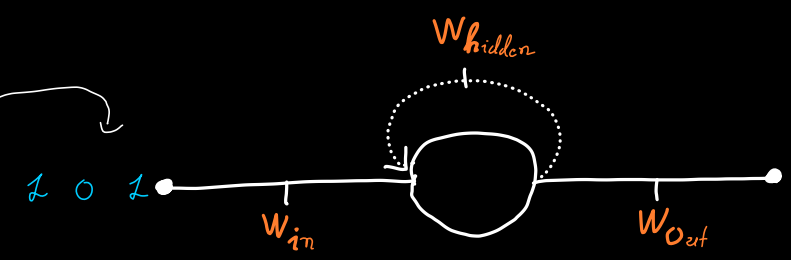
  → Here word gujarat is depend on another previous word gujarati, which is not far from it. so the word gujarat is predictable based on its near by word gujarati.

  2. Germany is beautiful place. I went there last year, But i couldn't enjoyed because i don't understand german.

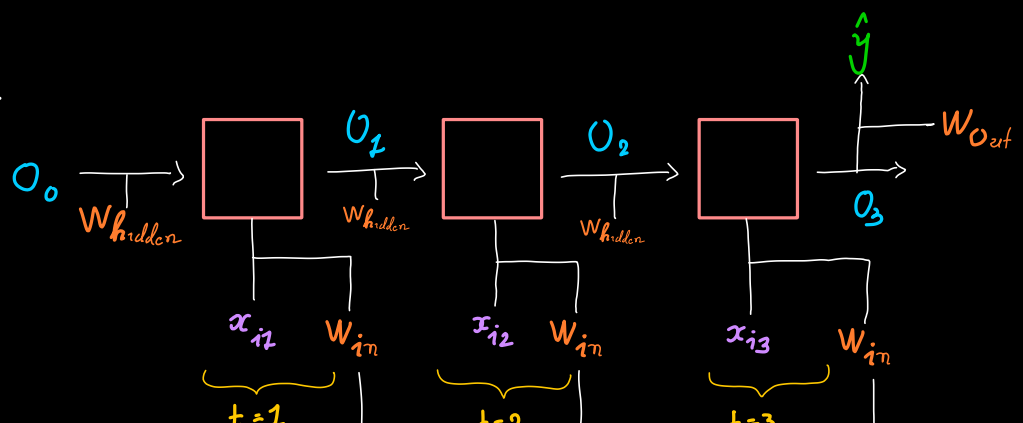
  → As we can see above example of long term dependency where german word depends on Germany.
  → This kinds of situations RNN can't work properly.
  → This problem is called as long term dependency problem. which happens due to Vanishing Gradient Problem.

Problem #1 Long term dependency

input	output
1 0 1	1
0 0 1	0
0 0 0	0
1 1 1	1



for 3 time steps



Lets imagine for 100 time-steps.

$$\frac{\partial L}{\partial W_{in}} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial O_{100}} \frac{\partial O_{100}}{\partial O_{99}} \frac{\partial O_{99}}{\partial O_{98}} \frac{\partial O_{98}}{\partial O_{97}} \cdots\cdots\cdots \frac{\partial O_2}{\partial O_1} \frac{\partial O_1}{\partial W_{in}}$$

→ As we go to calculate gradient, these long term dependencies gets smaller. in this situation short-term dependencies gives more contribution to culculate $\frac{\partial L}{\partial W_i}$.

→ As time steps increases (as the term gets bigger) long-term dependencies will get more small en that's why the responsibility of derivative culculation will be on short-term dependencies.

$$\frac{\partial L}{\partial W_{in}} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial O_{100}} \frac{\partial O_{100}}{\partial O_{99}} \frac{\partial O_{99}}{\partial O_{98}} \frac{\partial O_{98}}{\partial O_{97}} \cdots\cdots\cdots \frac{\partial O_2}{\partial O_1} \frac{\partial O_1}{\partial W_{in}}$$

Let's generalize this equation.

$$\frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial O_{100}} \prod_{t=2}^{100} \left(\frac{\partial O_t}{\partial O_{t-1}} \right) \frac{\partial O_1}{\partial W_{in}}$$

Where:

$$O_t = \tanh(x_{it} W_{in} + O_{t-1} W_h)$$

derivative of tanh

$$\frac{\partial O_t}{\partial O_{t-1}} = \tanh'(x_{it} W_{in} + O_{t-1} W_h) W_h$$

derivative of tanh will be between 0 to 1.

Now vum long-term dependency.

$$\frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial O_{100}} \prod_{t=2}^{100} \left(\tanh'(x_{it} W_{in} + O_{t-1} W_h) W_h \right) \frac{\partial O_1}{\partial W_{in}}$$

→ As timestamp increases, gradients which are at very long distance gets more smaller. this is called as vanishing gradient problem

Solution

- Use different activation function.
- Better weight initialization techniques.
- Use Skip RNNs.
- LSTM

Problem #2 Unstable training (Exploding Gradient)

→ What if, our long term dependency get such bigger that it dominate short-term dependency or became infinite.

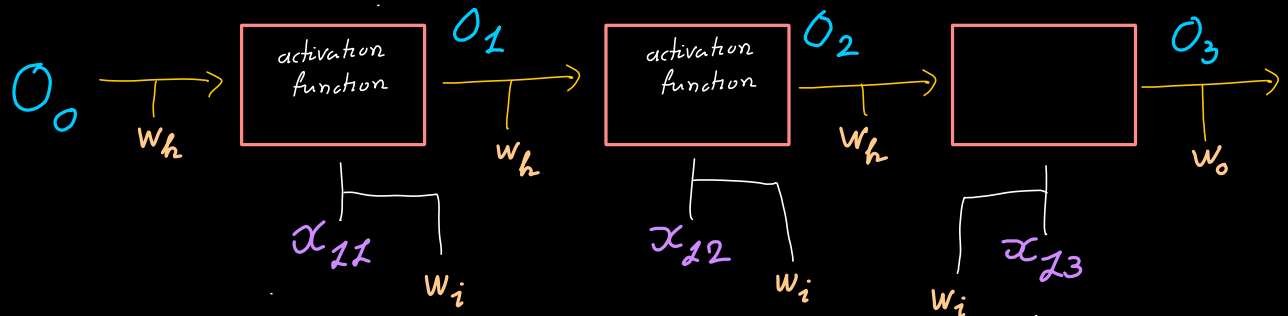
→ Hence gradient update will became infinite & training won't occur.

→ Suppose we have initialized our weight as 1. with each multiplication this number will become more larger.

Solution

- Gradient Clipping (set maximum limit)
- Controlled learning-rate
- LSTM

LSTM (Long Short Term Memory)



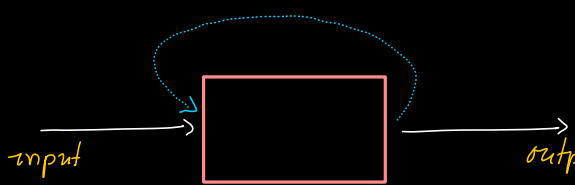
Above shown is normal workflow of RNN which can only hold short term context.

→ To store long term context, RNN also maintains a **long term memory** called as **cell state.**

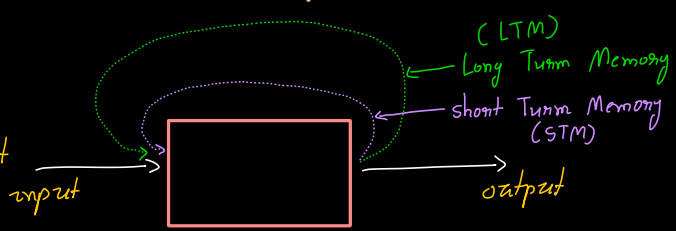
→ RNN has simple architecture, while LSTM has a little bit complex architecture.

cell state

→ Simple RNN architecture.

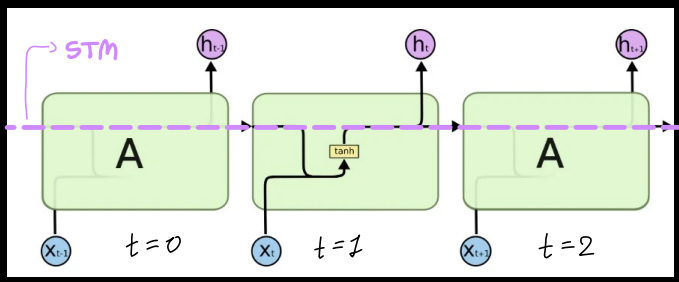


→ LSTM architecture

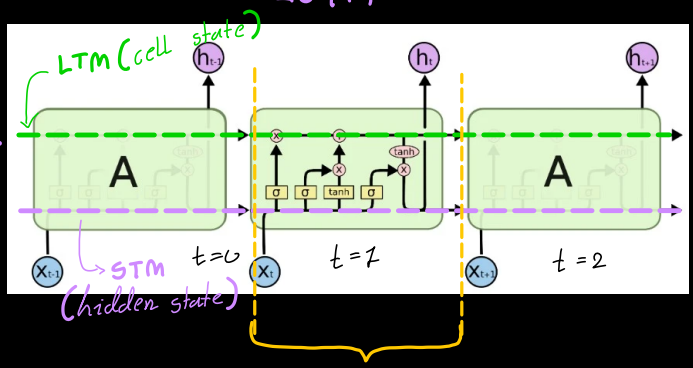


→ To maintain state we need a complex architecture which can maintain communication between LTM and STM.

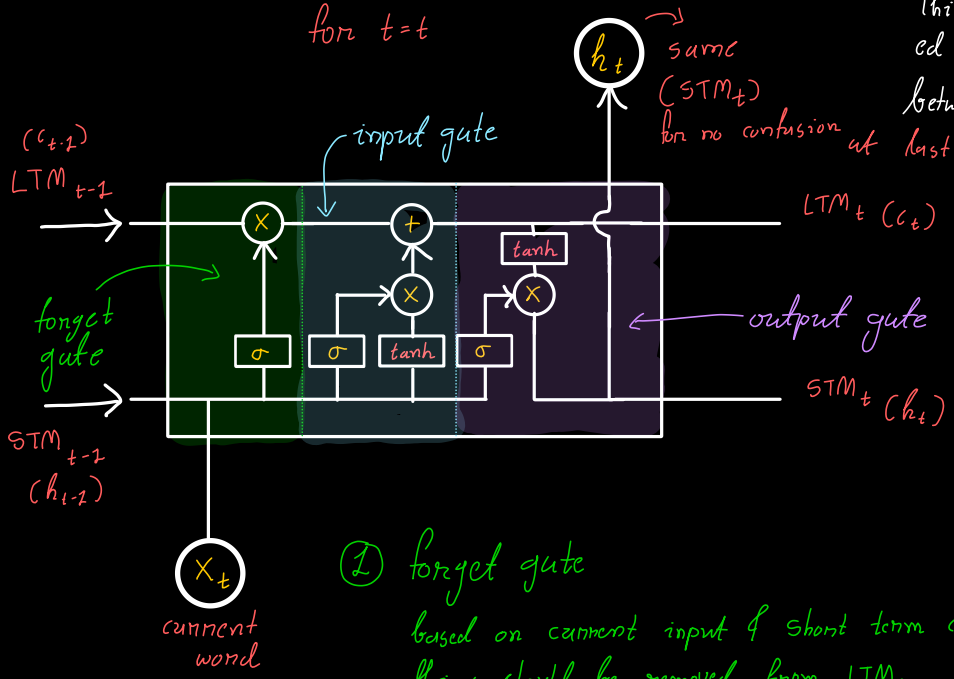
RNN



LSTM



This circuit is specially designed to establish communication between STM and LTM.



① **forget gate**

based on current input & short term context, it decides that which things should be removed from LTM.

② **input gate**

based on current arrived input, it decides that which things should be added in LTM.

③ **Output gate**

Based on current input, it decides to return LTM as output after removing something unnecessary things from LTM.

→ Now at last stage & based on current cell state (content in LTM), it will decide the output.

→ At each given stage, it creates new STM.

Task-1

Update LTM by removing unnecessary thing & adding important things.

Task-2

Create STM for next stage.

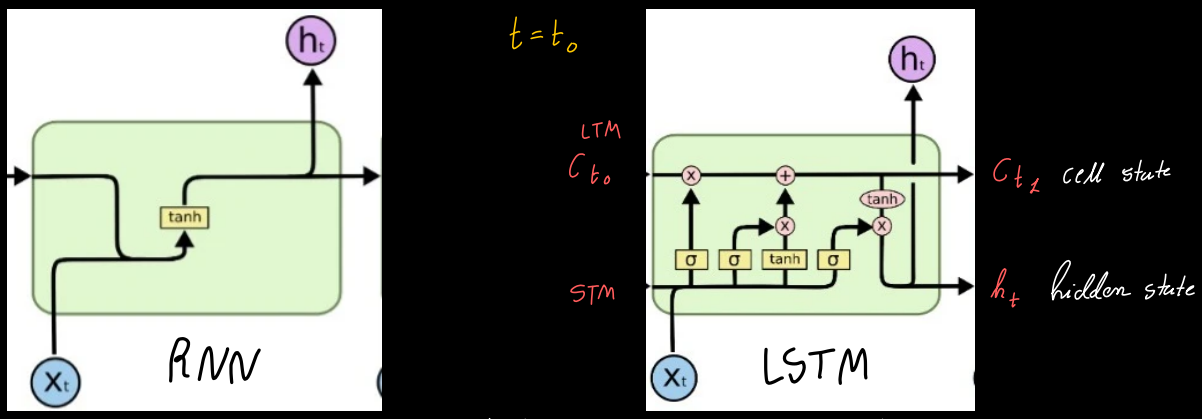
Input

- → C_{t-1} LTM of previous state
- → h_{t-1} previous hidden state
- → x_t Current word

Output

- → C_t current cell state
- → h_t Current hidden state

LSTM Architecture



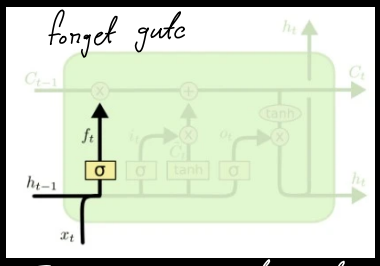
→ LSTM Maintains 2 different/interconnected state/Memory at a time called as
1. LTM (Long Term Memory) also called as **Cell State**.
2. STM (Short Term Memory) also called as **Hidden State**.

→ LSTM keeps interaction alive between LTM and STM.

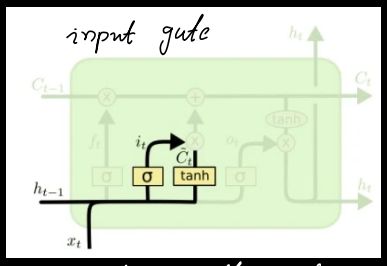
→ Within this process, 2 task is happening.
1. Update cell state. — add / remove
2. Calculating hidden state (h_t)

→ based on X_t, it decides that which thing should / should-not be in cell state.

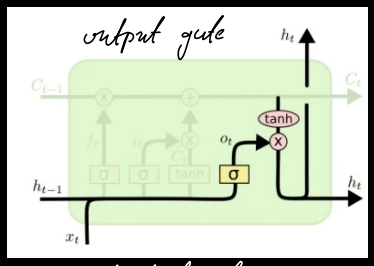
The Gates in LSTM



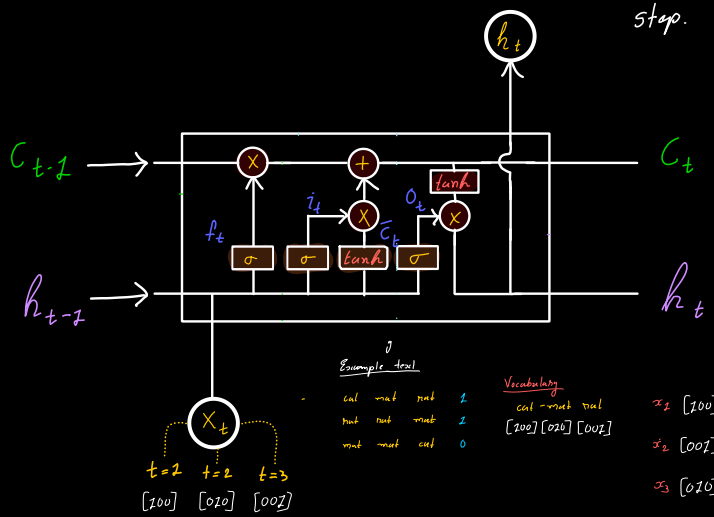
→ To remove something from cell state.



→ To add something to cell state.



→ To calculate hidden-state (h_t) for next time step.



$\bar{C}_t$ = potential important info that should be added in long term context.

→ Mathematically h_t and C_t are vectors of 3 dimention [0.1, 0.3, 0.2].
→ h_t and C_t will always have same dimention but may contain different numbers.
→ X_t is set of input words (already seen in RNN)
→ f_t, i_t, $\bar{C}_t$ and O_t are part of LSTM cell.
↳ candidate cell state.
all of them are vectors. and dimention of all of 4 them will be same. this is a rule.

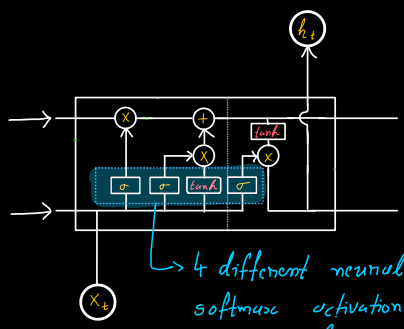
Pointwise operation
→ Major 3 operations are happening in cell.

1. ⊗
↳ example
$C_{t-1} = [1, 2, 3]$ $f_t = [4, 5, 6]$
o/p = [4, 10, 18]

2. ⊕
↳ example
$C_{t-1} = [1, 2, 3]$ $f_t = [4, 5, 6]$
o/p = [5, 7, 9]

3. tanh
↳ example
$C_{t-1} = [1, 2, 3]$ $f_t = [4, 5, 6]$
tanh tanh
[0.76159, 0.96402, 0.99505] [0.99932, 0.99990, 0.99998]

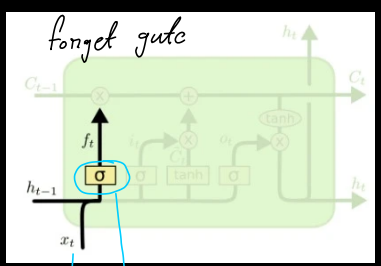
Neural Network Layers



↳ 4 different neural network architectures. where 3 ANN with sigmoid activation & 1 with softmax activation. Number of neurons is hyper parameter, but once we finalize the number of neurons, each ANN will have same number of neurons.

Now let's understand each gate.

#1 The Forget Gate

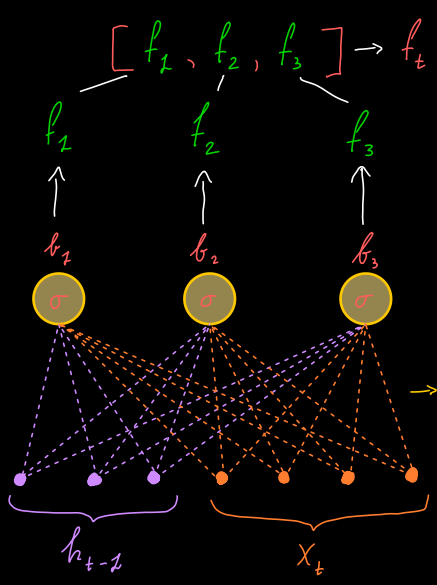


input
1. C_{t-1} : Previous cell state
2. h_{t-1} : Previous hidden state
3. X_t : Current word

Output
→ updated cell-state after removing unnecessary content.

→ suppose there is 3 units/neurons & activation = sigmoid
→ At current time-step X_t is 4 dimentional vector. $[x_{i1}, x_{i2}, x_{i3}, x_{i4}]$
→ h_{t-1} and C_{t-1} will be of 3 dimention. because the dimention of h_{t-1} & C_{t-1} will always be equal to the number of neurons in ANN layer.
→ This gate will work on 2 stages.
Stage-1 : calculate f_t
Stage-2 : $C_{t-1} \times f_t$ (element wise multiplication)
↳ removing something from cell state.

Now let's see the ANN architecture of forget gate.
as we discussed that ANN has 3 neurons with each has sigmoid as activation function.



$\rightarrow W_{7\times3}$ (21 weights, 3 biases)

matrix multiplication of (7×3) and (7×1) matrix. and adding (3×1) biases.

$$f_t = \sigma(W_f [h_{t-1}, X_t] + b_f)$$

3×7 $\quad 7\times1$ $\quad 3\times1$

3×1 $\quad 3\times1$

3×1

elementwise multiplication

$$f_t \otimes C_{t-1}$$

3×1 $\quad 3\times1$

3×1

This operation removes unnecessary things from cell state.

example	f_t	C_{t-1}	O/P
	$[\frac{1}{2}, \frac{1}{2}, \frac{1}{2}]$	$[4, 5, 6]$	$[2, 25, 3]$ → cell state is reduced by half
	$[1, 1, 1]$	$[4, 5, 6]$	$[4, 5, 6]$ → No changes occurred.
	$[0, 0, 0]$	$[4, 5, 6]$	$[0, 0, 0]$ → cell state is empty now.

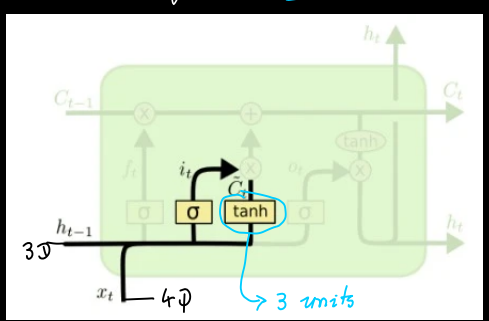
#2 The Input Gate

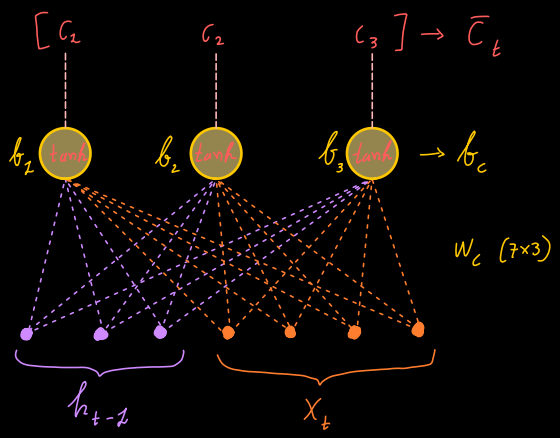
→ This gate works on 3 stages.

Stage-1 calculate C_t candidate cell state.

Stage-2 calculate i_t. where i_t decides that which value will be added in cell state from candidate cell state.

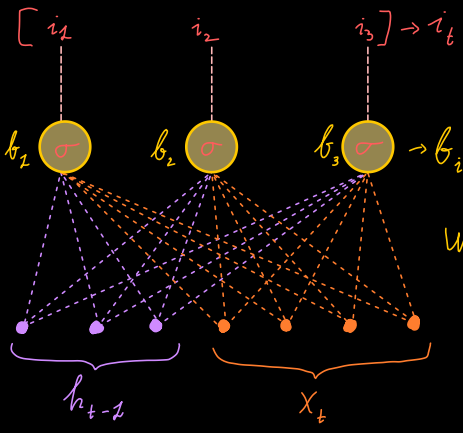
Stage-3 Calculate C_t from previous incomming state $[f_t \otimes C_{t-1}]$.





Now let's calculate i_t which filters the content from $\bar{c}_t$ & adds final filtered o/p in cell state.

→ ANN structure to calculate will be same as all previous architecture with sigmoid activation.



$$\tanh\left(W_c \cdot [h_{t-1}, x_t] + b_c\right)$$

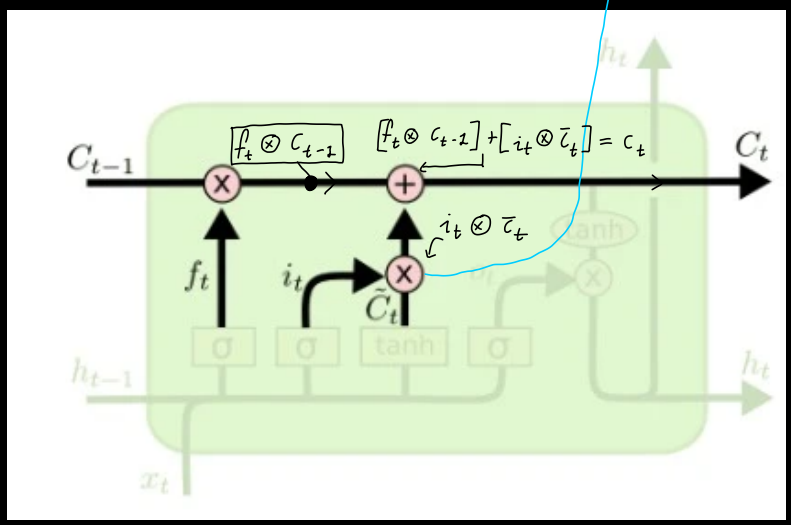
3×7, 7×1, 3×1 → 3×1 → 3×1 → tanh → 3×1

$$\text{sigmoid}(\sigma)\left(W_i \cdot [h_{t-1}, x_t] + b_i\right)$$

3×7, 7×1, 3×1 → 3×1 → σ(3×1) → 3×1

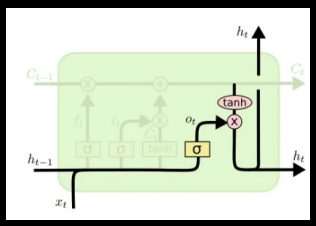
Let's check point wise operations.

$$i_t \otimes \bar{c}_t \rightarrow \text{filtered } \bar{c}_t$$



final cell state $(C_t) = [f_t \otimes C_{t-1}] + [i_t \otimes \bar{c}_t]$

#3 The Output Gate



Task : to decide the o/p or also we can say "decide the value of hidden state (h_t)" at current time stamp.

Here, main point is h_t is calculated by applying some mathematical operation on final cell state.

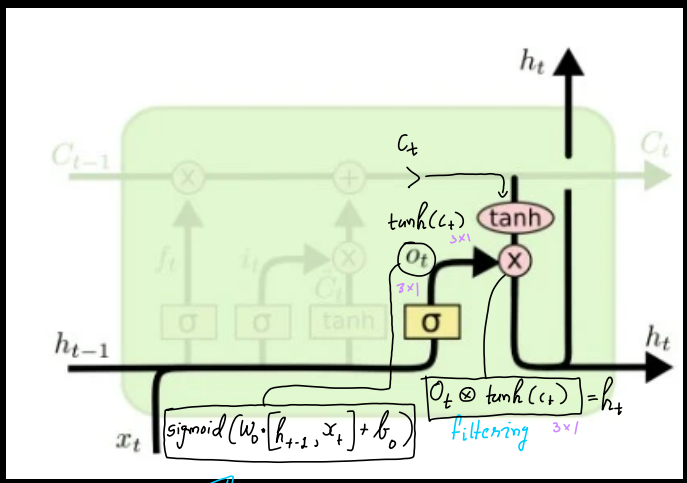
→ There are 2 steps to calculate hidden state of current timestamp by using current cell-state.

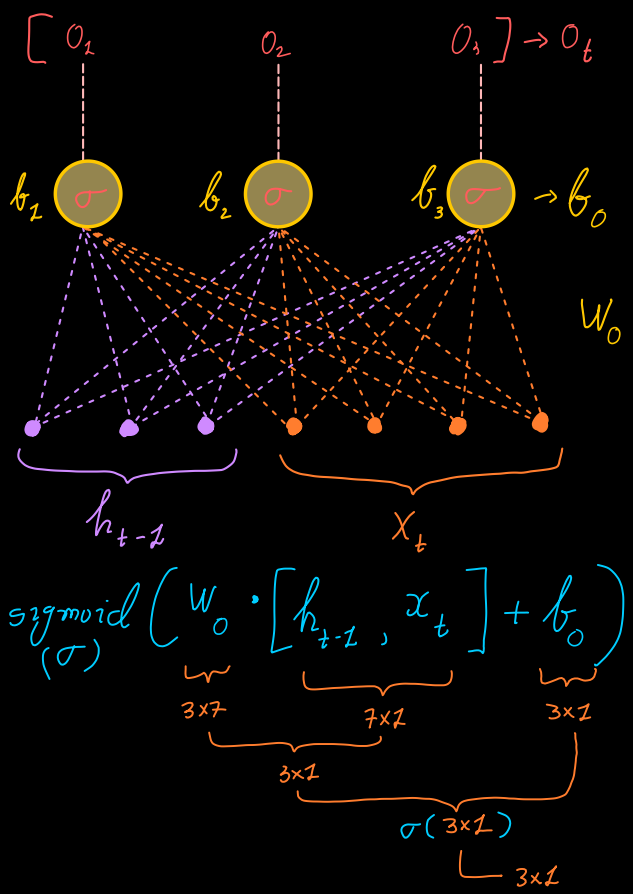
Step-1 :
- Apply tanh operation on each element of C_t that brings them between -1 to +1.

$$\tanh(C_t)$$

Step-2 :
- Apply filtering using O_t and $\tanh(C_t)$.

$$h_t = [O_t \otimes \tanh(C_t)]$$





And that is the output of current state.....