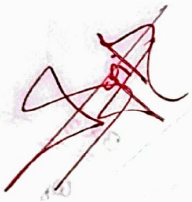
Recurrent Neural Network (RNN)

RNN

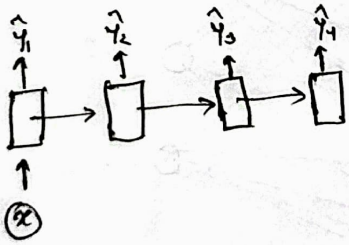
- It is a class of Neural network designed for sequential data like time series, text, audio etc.
- Unlike feedforward network, RNNs have memory - they use previous outputs as input for current processing.
- RNNs maintain a hidden state that carries information from previous time steps to influence the current prediction.



Types of RNN Architectures

① One to one → Regular neural network (eg., image classification)

② One to many → Sequence output (eg. image captioning), single input



③ Many to one → Single output (eg. sentiment analysis), Sequence input

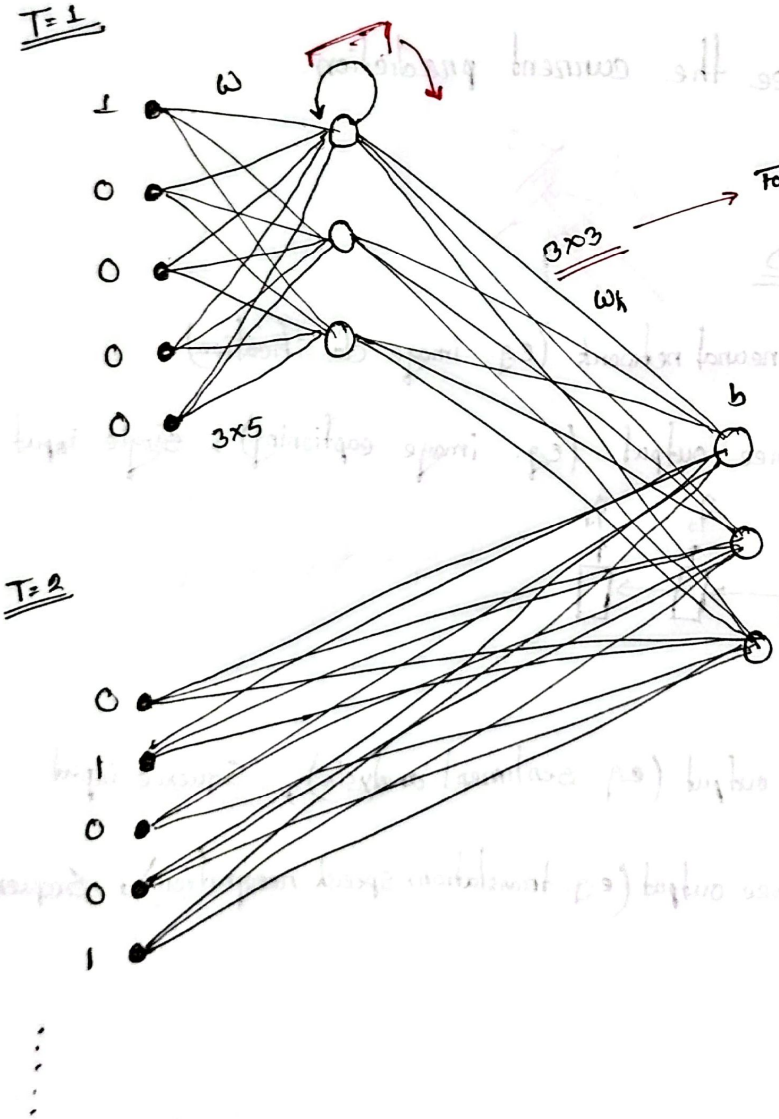
④ Many to Many → Sequence output (e.g. translation, speech recognition), Sequence input

RNN architecture

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

Word to vector (unique)

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]

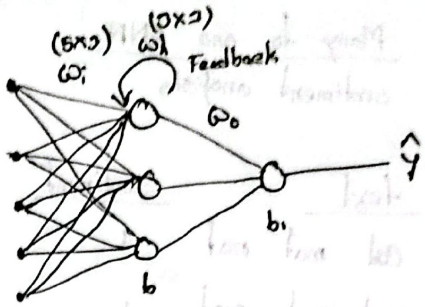


For the t=1, 3x3 weight (W_h) will be random one 0 values.

Forward propagation in RNN

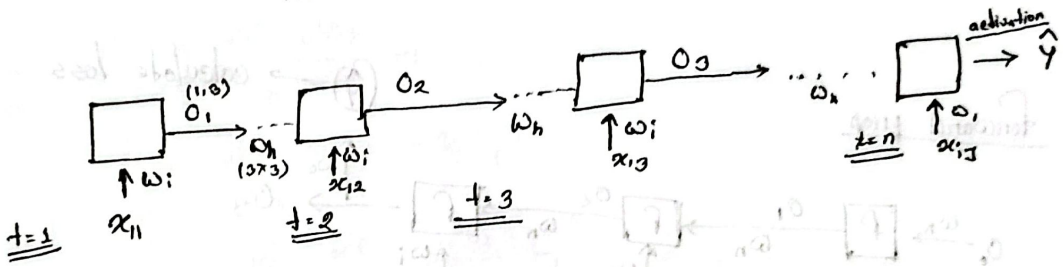
Forward → unfolding through time

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



□ → layer activation - tanh

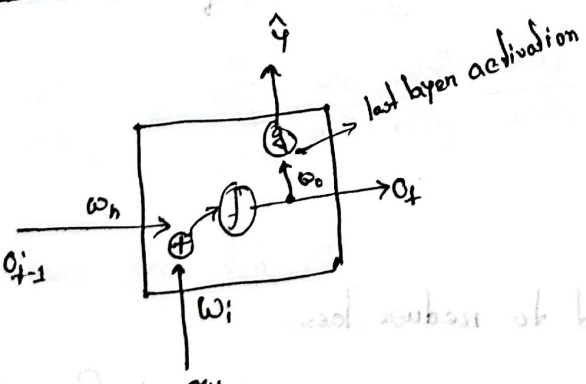
$f(x_{12}w_i + o_1 w_h + b_1)$



$f(x_{11}w_i + b)$

tanh / relu

Simplified representation



Backpropagation in RNN

Many to one RNN

Sentiment analysis

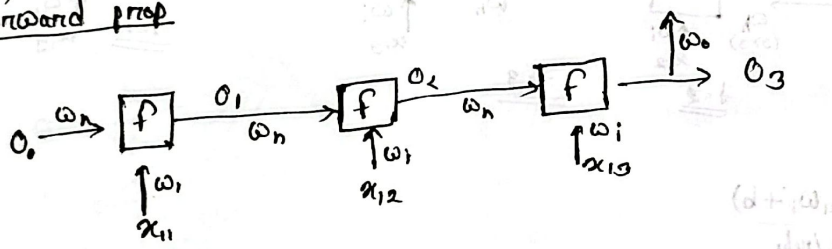
text	sentiment
cat mat rat	1
rat rat mat	1
mat mat cat	0

Vector representation

	y
[1 0 0] [0 1 0] [0 0 1]	1
[0 0 1] [0 0 1] [0 1 0]	1
[0 1 0] [0 1 0] [1 0 0]	0

forward prop

$\hat{y}$ → calculate loss



Here → activation

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

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

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

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

let,

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

Gradient descent is been used to reduce loss.

We will do only for weights

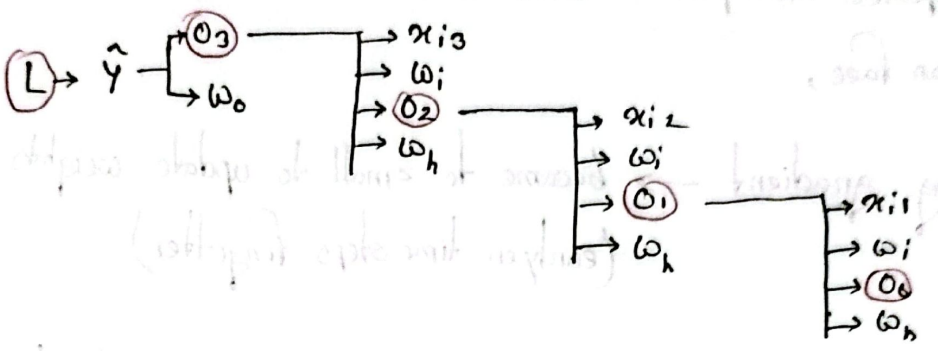
$w_i = w_i - \eta \left[\frac{\partial L}{\partial w_i}\right]$

$w_h = w_h - \eta \left[\frac{\partial L}{\partial w_h}\right]$ ← we have to find

$w_o = w_o - \eta \left[\frac{\partial L}{\partial w_o}\right]$

If we unfold for 3 time,
the flow will look like,

[all dependency of loss]



For, $\frac{\partial L}{\partial w_i} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial O_3} \frac{\partial O_3}{\partial w_i} +$

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

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

$\therefore \frac{\partial L}{\partial w_i} = \sum_{j=1}^{h} \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial O_j} \frac{\partial O_j}{\partial w_i}$

For, $\frac{\partial L}{\partial w_h} = \sum_{j=1}^{h} \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial O_j} \frac{\partial O_j}{\partial w_h}$

The same goes for $\frac{\partial L}{\partial w_o}$

Major Problems with RNN

Due to repeated multiplication while calculation the partial differentiation. ~~the~~ we can face,

① Vanishing gradient → became to small to update weights (earlyer time steps forgotten)

② Exploding Gradient → became to large and destabilize training

Because of such problem, RNNs are been used very less.

For batter performance we use,

⇒ LSTM

⇒ GRU

Long Short-Term Memory (LSTM)

LSTM

It is a special kind of RNN capable of learning long term dependencies.

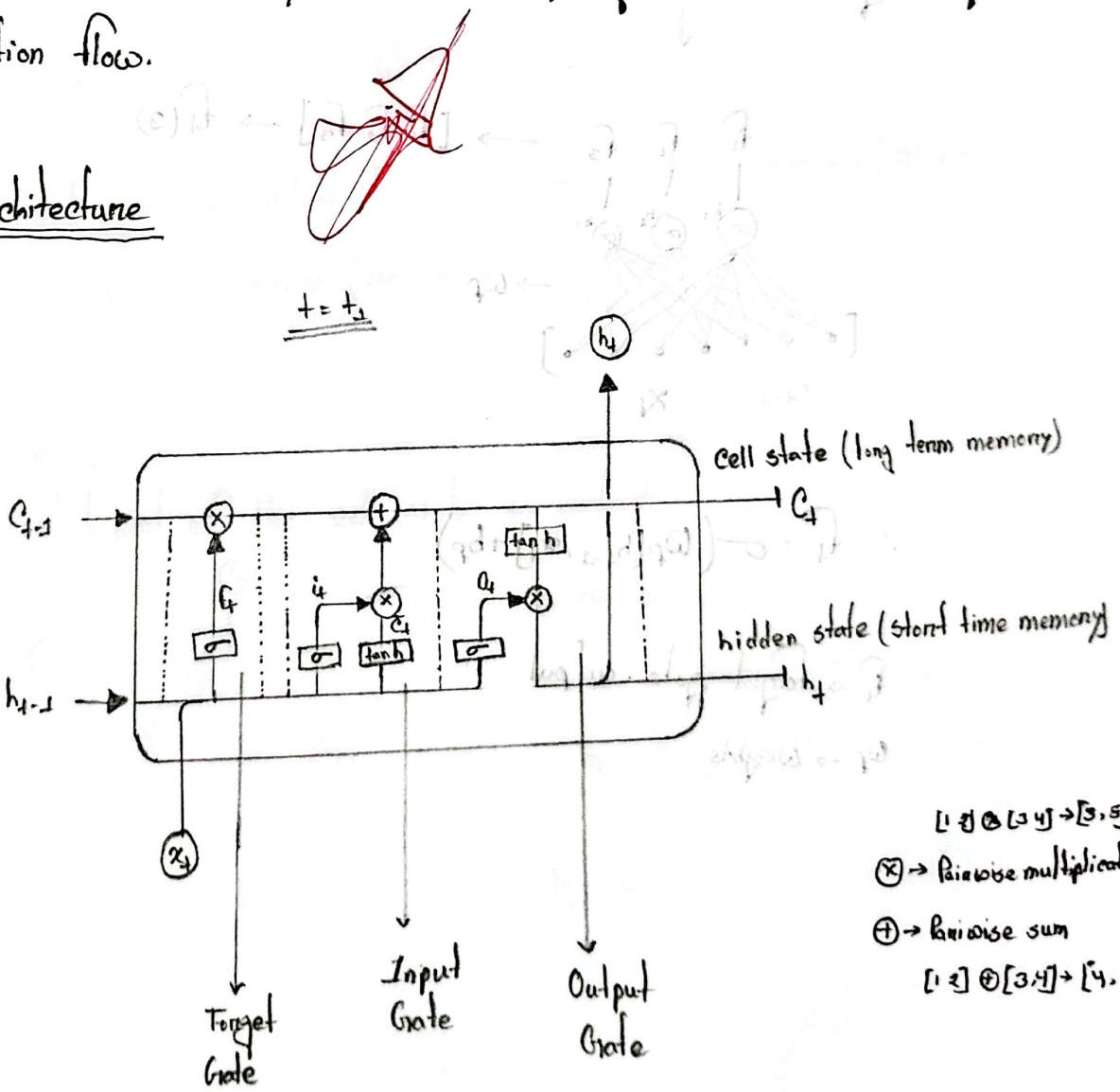
It is mainly designed to solve the vanishing Gradient problem of RNN.

What was the key idea?

=> It uses gates to control how much past information should be remembered or updated.

=> LSTM introduce memory cells and gating mechanisms to regulate information flow.

LSTM Architecture



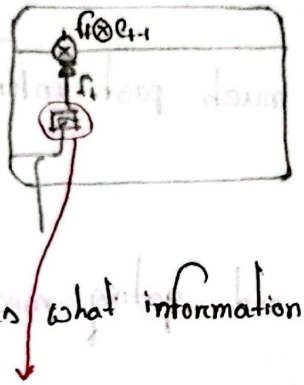
Components of an LSTM Cell

① Forget gate

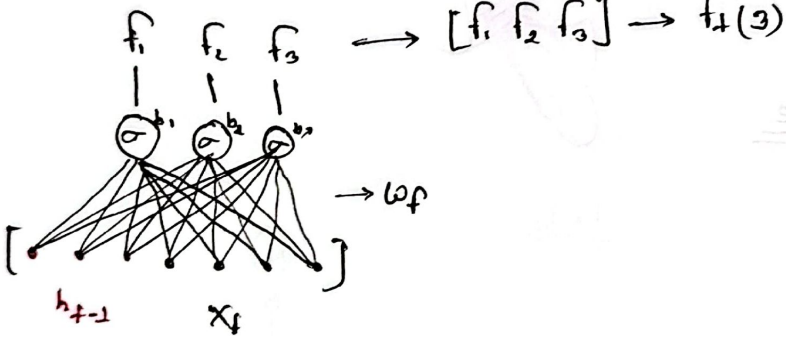
Notes

dimention of $h_t, c_t, f_t, i_t, o_t, \bar{c}_t$ will alaay same.

x_t can deffer.



This gate decides what information to remove from the cell state.

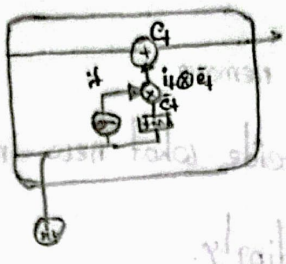


$$\therefore f_t = \sigma\left(\omega_f [h_{t-1}, x_t] + b_f\right)$$

$f_t \rightarrow$ forget gate output

$\omega_f \rightarrow$ weights

② Input Gate



It decides what new information to added to the cell state.

$$C_t = f_t \otimes C_{t-1} \oplus i_t \otimes \bar{C}_t$$

$$i_t = \sigma(W_i [h_{t-1}, x_t] + b_i)$$

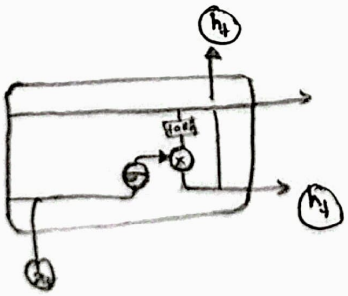
$$\bar{C}_t = \tanh(W_c [h_{t-1}, x_t] + b_c)$$

i_t = input gate activation (controls how much to update) (filter)

$\bar{C}_t$ = Candidate cell state (new potential memory)

③ Output Gate

Decides what part of the cell state to output.



- $\tanh(-1, 1)$

$$O_t = \sigma(W_o [h_{t-1}, x_t] + b_o)$$

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

h_t = new hidden state (output)

Complete Flow

① Input x_t and previous hidden state h_{t-1} enter the cell.

② Forget gate f_t decides which past to remove.

③ Input gate i_t and candidate $\tilde{c}_t$ decide what new info to add.

④ Cell state c_t is updated accordingly.

⑤ Output gate o_t decides what to output as h_t.

Gated Recurrent Unit (GRU)

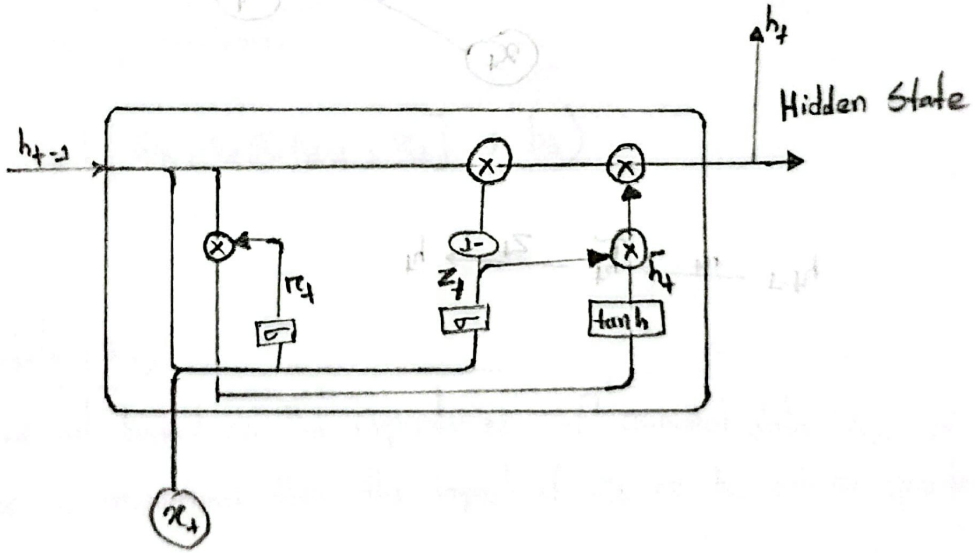
GRU

It's a varient of LSTM, designed to make training simplier and faster, while still solving vanishing gradient problem.

Key idea

GRU combines the forget gate and input gate of LSTM into a single update gate, reducing the number of parameters.

Architecture of GRU



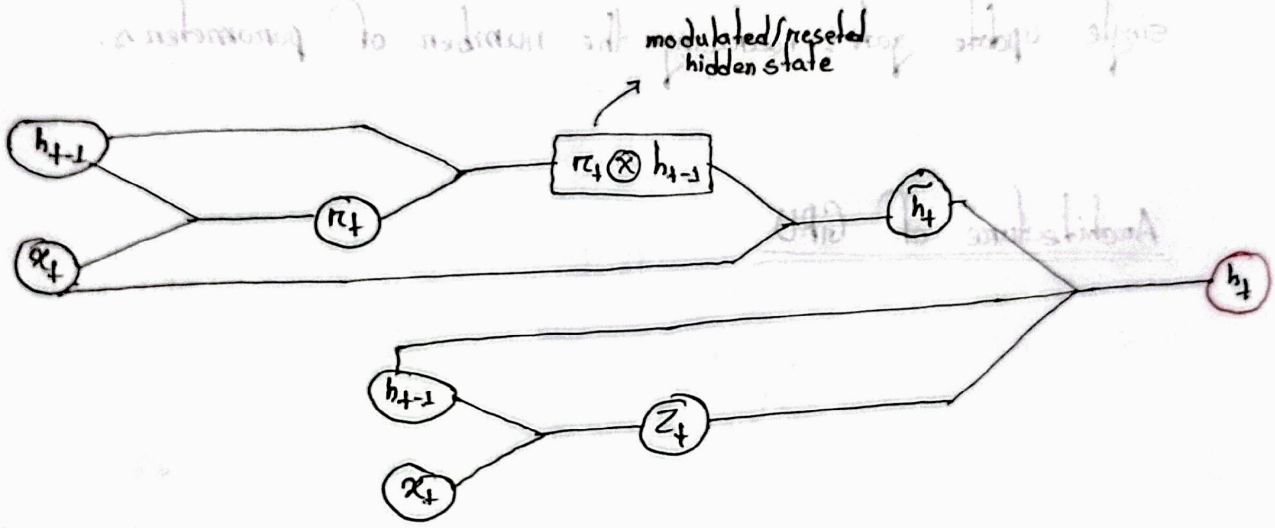
Flow of calculating h_t

① Calculate reset gate (r_t)

② Calculate candidate hidden state ($\tilde{h}_t$)

③ Calculate Update gate (z_t)

④ Calculate current hidden state (h_t)



$h_{t-1} \xrightarrow{r_t} \tilde{h}_t \xrightarrow{z_t} h_t$

① Update gate (z_t)

Controls how much past information to carry forward.

$$z_t = \sigma(W_z [h_{t-1}, x_t] + b_z)$$

② Reset gate (r_t)

Controls how much past information to forget on re-set.

$$r_t = \sigma(W_r [h_{t-1}, x_t] + b_r)$$

③ Candidate hidden state ($\tilde{h}_t$)

Creates new memory content.

$$\tilde{h}_t = \tanh(W_h [r_t \otimes h_{t-1}, x_t] + b_h)$$

④ Final hidden state (h_t)

Here it balanced out based on the importance of current data (z_t). If current information is important then the impact of z_t on h_t will be greater or vice versa.

$$h_t = (1 - z_t) \otimes h_{t-1} \oplus z_t \otimes \tilde{h}_t$$

LSTM vs GRU

Feature	LSTM	GRU
Gate	3	2
Memory cell	Yes (c_t)	No (only hidden state)
Parameter	More	Fewer
Training Speed	Slower	Faster
Performance	Slightly better for long dependency	Nearly same
Complexity	High	Simpler
Suitable for	Complex, long term sequence, bit data	Simpler, small data

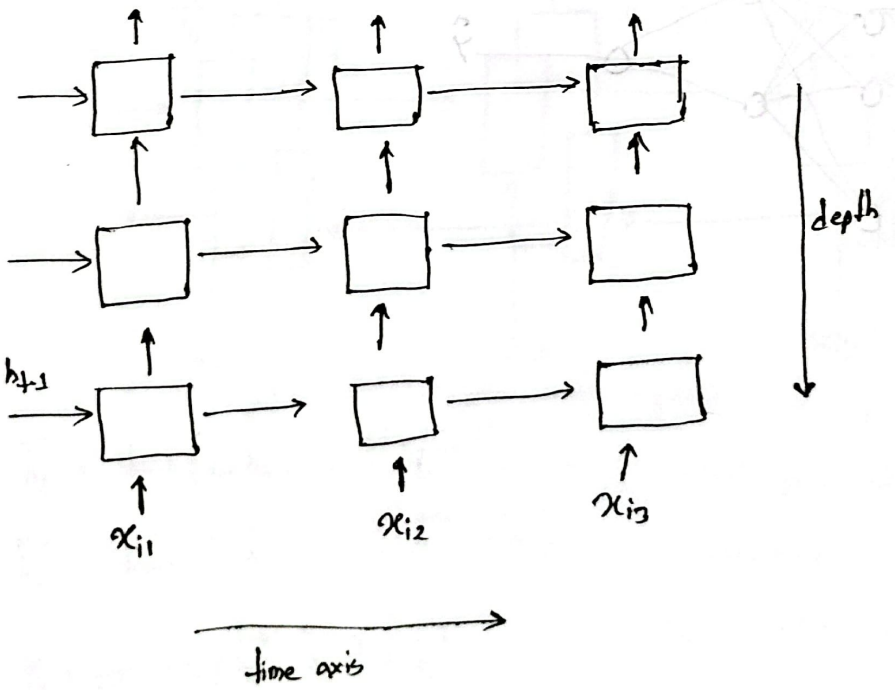
Deep RNN

=> An extension of a standard RNN where multiple recurrent layers are stacked on top of each other.

=> Each layer processes the sequential data and passes its hidden states to the next layer.

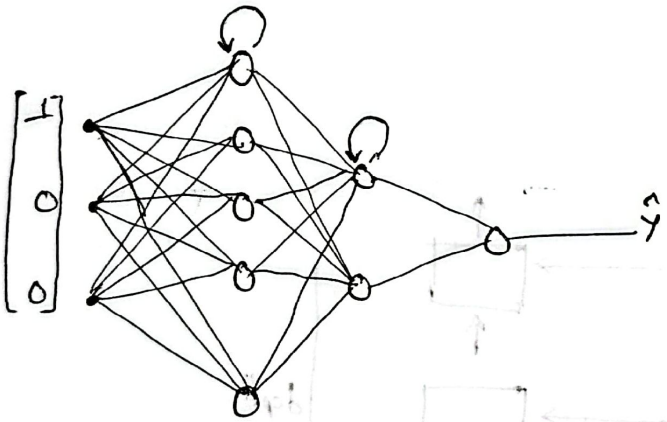
Key idea

Deep RNNs learn hierarchical ~~represented~~ temporal representations, where lower layers capture short term patterns and higher layers capture long term dependencies.



```
model = Sequential([
    Embedding(10000, 32, input_length=100),
    SimpleRNN(5, return_sequences=True),  # This will be true except the last layer
    SimpleRNN(2),
    Dence(1, activation='sigmoid')
])
```

SimpleRNN(...)
LSTM(...)
GRU(...)



Bidirectional RNN (BiRNN)

It is an extension of RNN that can process data in both forward and backward direction.

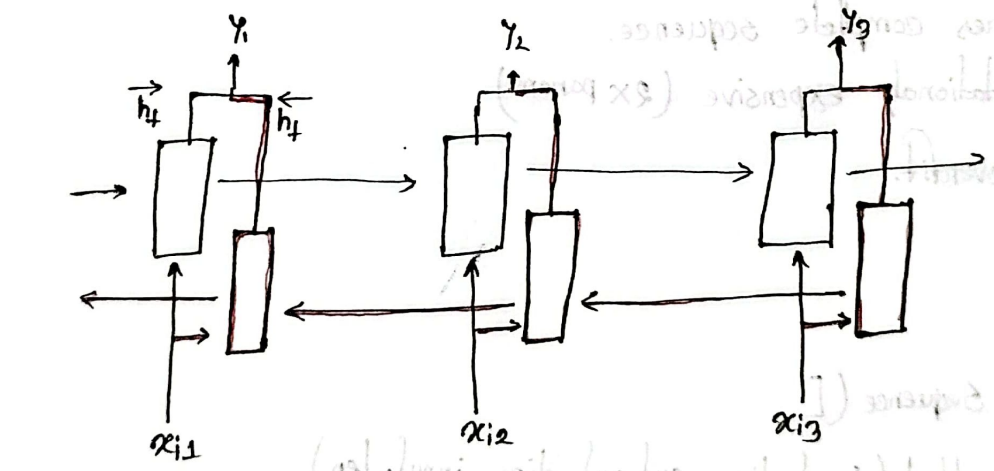
⇒ Useful for the tasks where the entire input sequence is available before prediction.

⇒ When the current prediction can depend of the future. like,

Amazon, a great website.

Amazon, a big river.

Amazon, a huge jungle.



$$\overrightarrow{h_t} = \tanh\left(\overrightarrow{W}\overrightarrow{h_{t-1}} + \overrightarrow{U}x_t + \overrightarrow{b}\right)$$

$$\overleftarrow{h_t} = \tanh\left(\overleftarrow{W}\overleftarrow{h_{t-1}} + \overleftarrow{U}\overleftarrow{x_t} + \overleftarrow{b}\right)$$

$$y_t = \sigma\left(V\left[\overrightarrow{h_t}, \overleftarrow{h_t}\right] + b\right)$$

Mostly used with LSTM (BiLSTM) and (BiGRU)

⇒ Uses LSTM and GRU for both direction

⇒ In BiLSTM, handels long term dependency better.

⇒ In BiGRU, Faster and computationaly lighter.

Adv

⇒ Use both past and future context.

⇒ Improve accuracy in sequence pred. tasks.

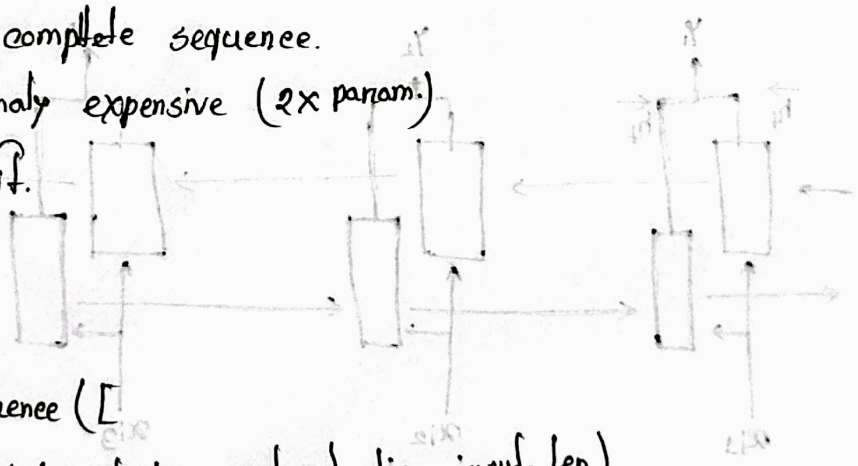
⇒ Works well in NLP and speech recognition.

Dis adv

⇒ Requires complete sequence.

⇒ Computationaly expensive (2x param.)

⇒ Can overfit.



Code

```
model = Sequence([
    Embedded(input_dim, output_dim, input_len),
    Bidirectional(SimpleRNN(5, return_sequences = True)),
    Bidirectional(SimpleRNN(2)),
    Dense(1, activation = 'sigmoid')
])
```