Bidirectional RNNs

Backpropagation in RNN.

→ It is called as backpropagation through time

Eg: Take Many to one RNN for sentiment analysis.

Eg:
R1 = cat mat rat 1
R2 = Rat rat mat 1
R3 = mat mat cat 0.

Step1: Encoding

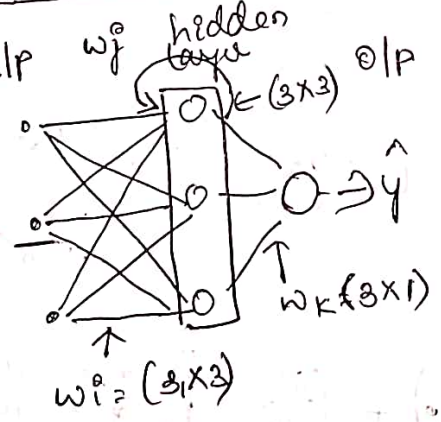
corpus = [cat mat rat]

cat ⇒ [100] [010] [001]

Eg in vector representation y

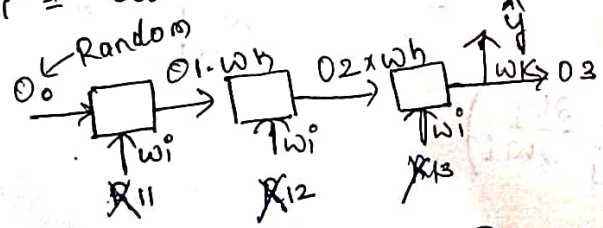
R1 = [100] [010] [001] 1
R2 = [001] [001] [010] 0
R3 = [010] [010] [100] 0

RNN in forward propagation.



we need to send words to RNN in each review.

R1 = at t=1 send [100] ⇒ cat



Equations for forward propagation

$$O_1 = f(x_{11} \cdot w_i + O_0 w_h) + b_1$$

$$O_2 = f(x_{12} \cdot w_i + O_1 w_h) + b_2$$

$$O_3 = f(x_{13} w_i + O_2 w_h) + b_3$$

$$\hat{y} = f(O_3 \cdot w_k) + b_4$$

$$\text{loss} = (y \sim \hat{y})$$

* we need to reduce loss function using gradient descent.
* we need to update w_i, w_h, w_k

Weight updation formula

$$w_i = w_i - \eta \frac{\partial L}{\partial w_i}$$ calculate this using chain rule.

$$w_h = w_h - \eta \frac{\partial L}{\partial w_h}$$

$$w_k = w_k - \eta \frac{\partial L}{\partial w_k}$$

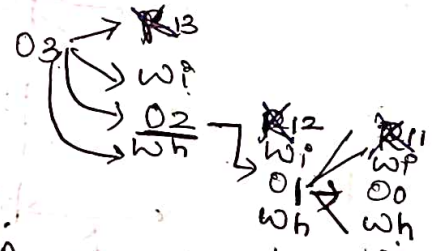
[loss depends on w_k and O_3]

$$\frac{\partial L}{\partial w_k} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial w_k}$$

$$\boxed{\frac{\partial L}{\partial w_k} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial O_3} \cdot \frac{\partial O_3}{\partial w_k}}$$

To update w_i

$L \rightarrow O_3 \rightarrow w_o$



$$\frac{\partial L}{\partial w_i} = \left[\left(\frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial O_3} \cdot \frac{\partial O_3}{\partial w_i}\right) + \left(\frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial O_3} \cdot \frac{\partial O_3}{\partial O_2} \cdot \frac{\partial O_2}{\partial w_i}\right) + \left(\frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial O_3} \cdot \frac{\partial O_3}{\partial O_2} \cdot \frac{\partial O_2}{\partial O_1} \cdot \frac{\partial O_1}{\partial w_i}\right)\right]$$

[↑ in Review 1 there are 3 words hence 3 set]

In simple,

$$\frac{\partial L}{\partial w_i} = \sum_{j=1}^{3} \left(\frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial O_j} \cdot \frac{\partial O_j}{\partial w_i}\right)$$

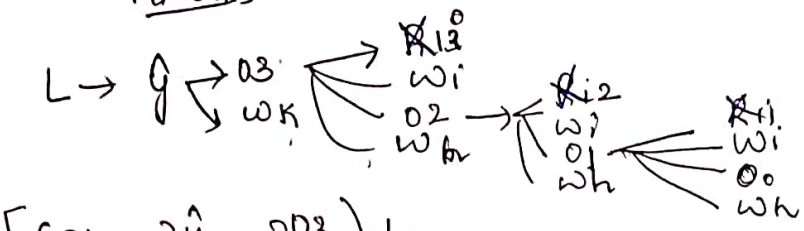
with expansion for $j=3$

$$= \frac{\partial L}{\partial \hat{y}} \left(\frac{\partial \hat{y}}{\partial o_1} \right) \frac{\partial o_1}{\partial w_i}$$

$\hat{y} \to o_3 \to o_2 \to o_1$

$$= \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial o_3} \cdot \frac{\partial o_3}{\partial o_2} \cdot \frac{\partial o_2}{\partial o_1} \cdot \frac{\partial o_1}{\partial w_i}$$

To update $\boxed{\frac{\partial L}{\partial w_h}}$



$$\frac{\partial L}{\partial w_h} = \left[\left(\frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial o_3} \cdot \frac{\partial o_3}{\partial w_h} \right) + \left(\frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial o_3} \cdot \frac{\partial o_3}{\partial o_2} \cdot \frac{\partial o_2}{\partial w_h} \right) + \left(\frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial o_3} \cdot \frac{\partial o_3}{\partial o_2} \cdot \frac{\partial o_2}{\partial o_1} \cdot \frac{\partial o_1}{\partial w_h} \right) \right]$$

$$\frac{\partial L}{\partial w_h} = \sum_{j=1}^{3} \left(\frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial o_j} \cdot \frac{\partial o_j}{\partial w_h} \right)$$

Then we try to reach global minima.

Problems with RNN.

① Problem of long term dependency } Because of
② Problem of ~~unstable~~ stagnated training. } unstable gradients.

① Problem of long term dependency.

→ If the sequence (number of timesteps) is too long it will not remember begining states.

Eg: Maharastra is a beutiful place. I went there last year But I couldnot enjoy properly. Because I don't understand

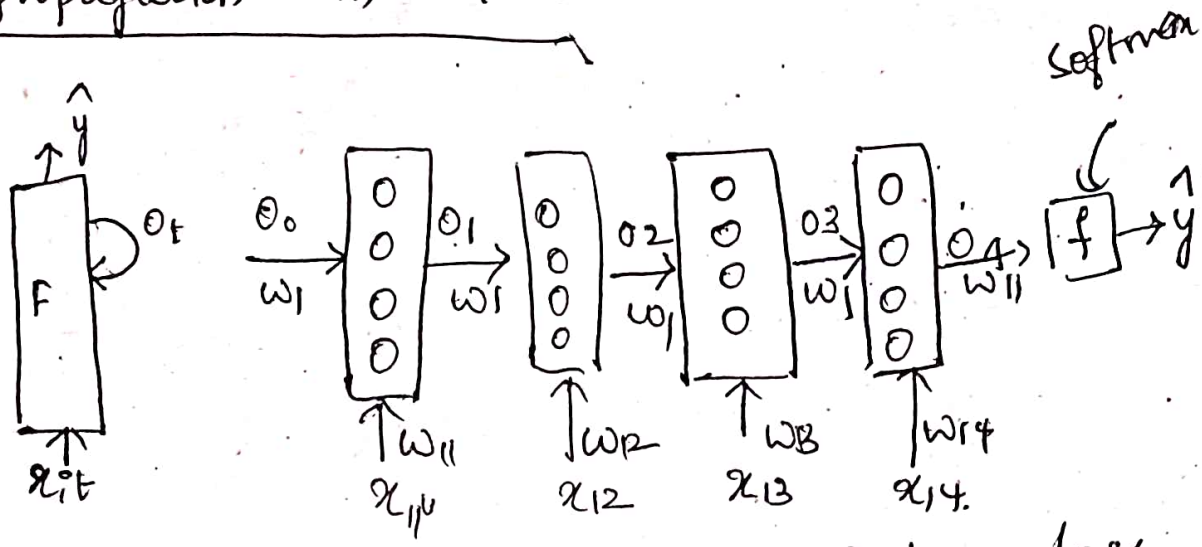
long term → marathi

short → Marathi is spoken in Maharastra

→ RNN is not having that much memory
→ It results in vanishing gradient problem.

(2) * Sometimes we cannot train properly.
* Exploding gradient problem

Back propagation in RNN



* Reduce loss
*

$O_1 = f(x_{11}w + O_0 w_1)$

$O_2 = f(x_{12}w + O_1 w_1)$

$O_3 = f(x_{13}w + O_2 w_1)$

$O_4 = f(x_4 w + O_3 w_1)$

$$\frac{\partial L}{\partial \hat{y}} = \frac{\partial L}{\partial w_4} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial w_{11}}$$

$$\frac{\partial L}{\partial w_{14}} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial O_4} \cdot \frac{\partial O_4}{\partial w_{14}}$$

→ updation of weights in each sequence.

Problems

1. sigmoid → vanishing gradient
2. Relu → if derivative is greater than 1 then exploding

It can be solved by LSTM.

RNN → sequential data,

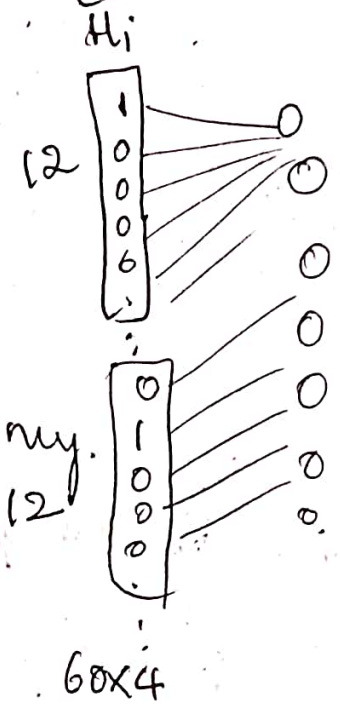
NLP → RNN.

Text → we need to vectorize it.

Hi | my | name

Vectorization → one hot Encoder

[1 0 0 0 0], [0, 1, 0 ...), [0, 0, 1, ...)



2nd
36 ← Input size

3rd
48

* Input size we cannot be vary in RNN

→ Identify which has the highest words
Then put zero padding
⇒ All will be zero.

① text i/p is varying
② zero padding (unnecessary computation)
③ Prediction will be wrong
④ sequence information is lost

RNN forward propagation.

⇒ RNN accepts the data in (timesteps, input_feature)

	Review x_1	sentiment
①	movie was good	1
②	movi was bad	0
③	movi was not good	0

① we need to convert words to vectors.

↑ Sunique words forms a vocabolary ↓ corpus

⇒ Review 1

[[10000], [01000], [00100]]

3 words (3, 5) ← Number of feature

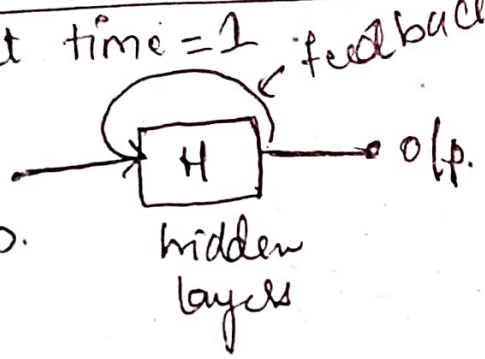
⇒ I can represent movi with 5 number

[10000] movi [01000] was [00100] good [00010] bad [00001] not

Each word will be sent to RNN seperately

ie at time = 1 [10000].

RNN architecture



Review	Sentiment
R_1 movie was good	1
R_2 movie was bad	0
R_3 movie was not good	0

Input for RNN will be in the form

(timesteps, input_features)

Dictionary/corpus: {movie, was, good, bad, not}

Hence

movie = [10000]
was = [01000]
good = [00100]
bad = [00010]
not = [00001]

} using one hot encoding

Hence

Review1 = movie was good

[[10000] [01000] [00100]]

Review2 = movie was bad

[[10000] [01000] [00010]]

Shape (3,5) → (Number of i/p features)

↑ Number of timesteps

i.e. 3 words

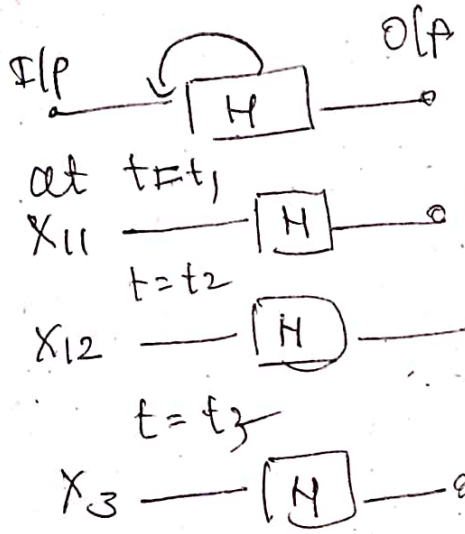
In RNN, each word by word we will send data to RNN.

In keras, (batchsize, timestep, $\hat{y}$ features)

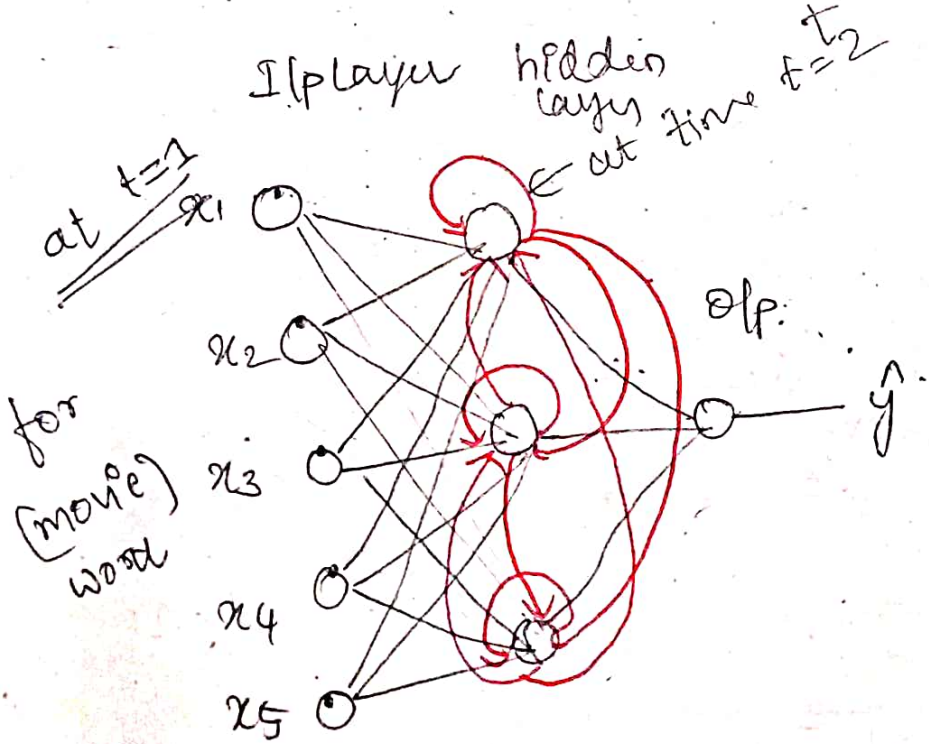
(3, 4, 5)

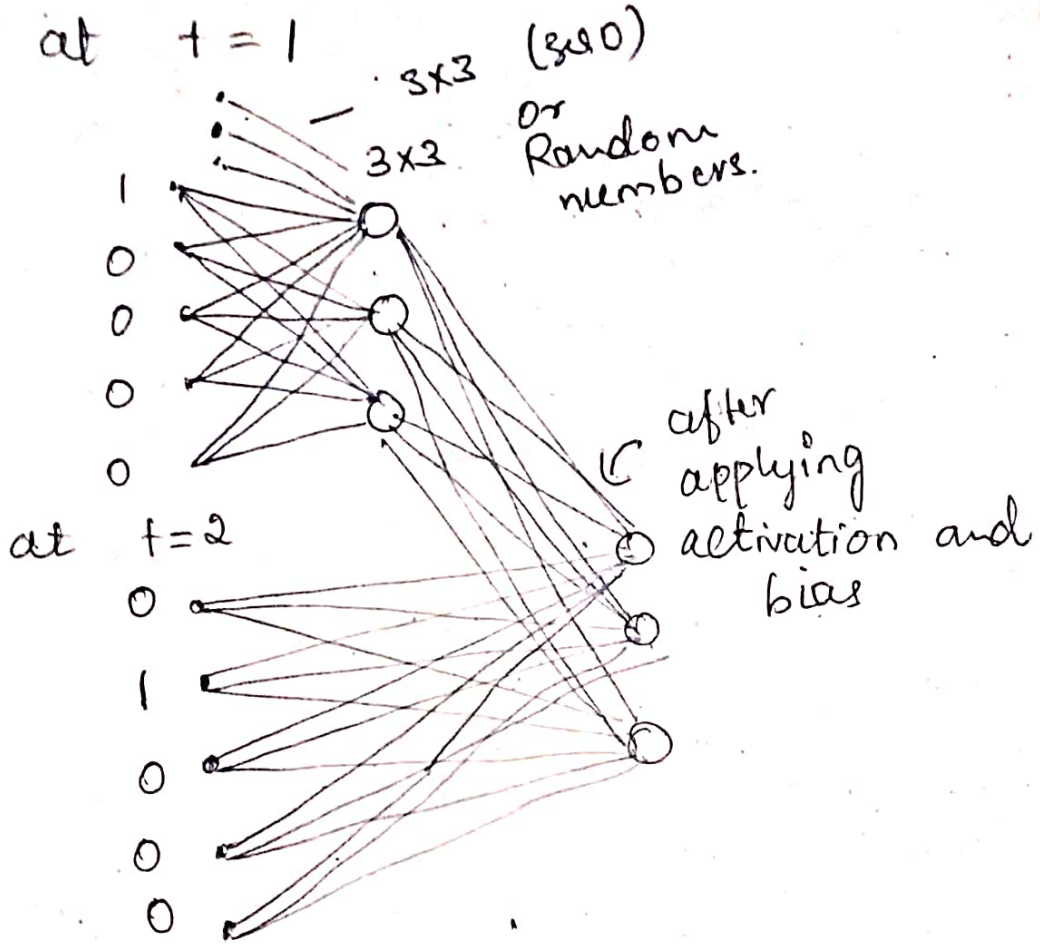
- 3 reviews are sent together
- max length of words
- corpus length.

RNN architecture

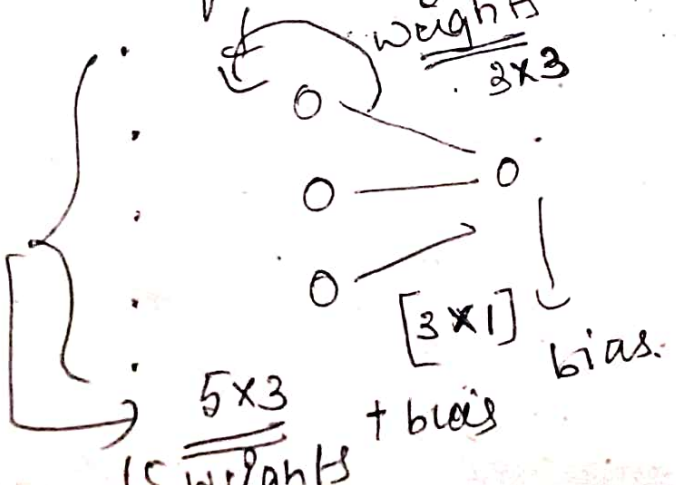


* RNNs are not feedforward
* There is a concept of state.
* Hidden layer sends a feedback back..





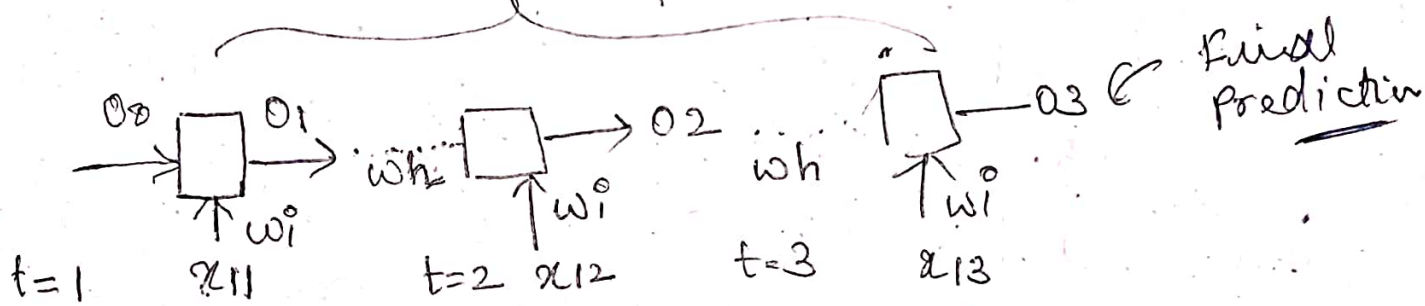
How many weights and bias?



ie w_1 = [15 weights]
w_2 = (3×3) = 9 ⇒ 32 Trainable parameters
w_3 = 3
bias = 4

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



RNN forward propagation.

Step 1: at t=1

① send 1st word x_{11}
② multiply with all weights w_i
③ Then activation function will act upon it. $O_1 = f(x_{11} w_i) + b_1$
④ Then add bias to it

at t=2

① we use the same network with same weights.

(2) previous O_1 is given as input with weighted connection.

$$O_2 = f(x_{12} w_i + O_1 w_h + b_2)$$

at t_3

$$O_3 = f(x_{13} w_i + O_2 w_h + b_3)$$

Note: Here weight sharing will be done.

---

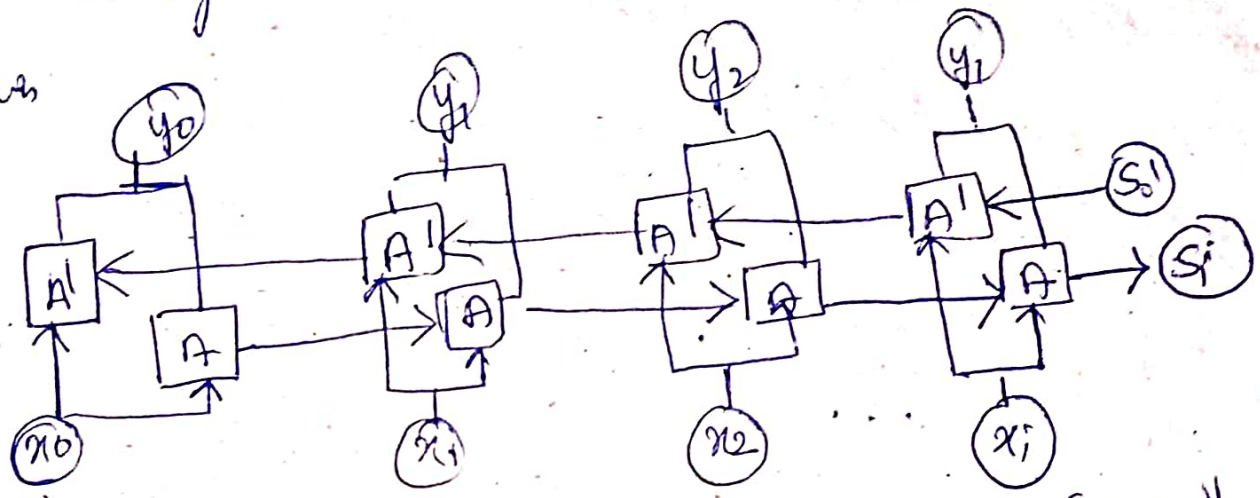
1) A bidirectional RNN is a type of neural network architecture that allows for processing of i/p sequence in both foward and backward directions.

2) → This is achieved by having two seperate hidden layers for each time step: one for proceing the sequence from left to right (forward) and one for proceing the sequence from right to left (backward)

→ The o/p of these two hidden layers are then combined to generate a final o/p which takes into account information from both directions of the i/p sequence.

This approach is particularly useful for tasks where context from both past and future time steps is important.

→ During training, the weights of both forward and backward RNNs are updated using backpropagation through time (BPTT) to minimize the loss function.



① → Bidirectional recurrent neural networks (RNN) are really just putting two independent RNNs together.

→ The i/p sequence is fed in normal time order for one network, and in reverse time order for another. The o/p of the two networks are usually concatenated at each time.

→ This structure allows the networks to have both backward and forward information about the sequence at every time step.

Problems of RNN

① Vanishing Gradient problem: RNNs can have difficulty learning long-term dependencies in sequences due to the vanishing gradient problem.

② Exploding Gradient Problem: In addition to the vanishing gradient problem, RNN can also suffer from the opposite problem, This occurs when the gradients become too large during backpropagation

③ Memory Limitations;

④ Computationally Expensive

⑤ Overfitting.

Bidirectional RNN

→ A typical state in an RNN relies on the past and present events..

→ There can be situations where a prediction depends on the past present and future events.

→ Take speech recognition, when you use a voice assistant, you initially utter a few words after which the assistant interprets and responds

→ This interpretation may not entirely depend on the preceding words; the whole sequence of words can make sense only when succeding words are analyzed.

→ To enable the past and future traversal of i/p bidirectional RNNs are used.

→ Bidirectional RNN → combination of 2 RNNs one RNN moves forward begining from the start of the data sequence and other moves backward begin from end of data sequence

→ A BRNN has an additional hidden layer to accomodate the backward training process.

At any time t.

$$O_f = \phi(X_t * \overset{\text{Forward}}{\vec{w}_i} + O_b \times \overset{\text{Forward}}{\vec{w}_{b_0}} + b_i) \rightarrow \text{Forward.}$$

$$O_n = \phi(x_t \times w_i^{\text{backward}} + O_n \times \overset{\text{Backward}}{\vec{w}_{h_N}} + b_n) \rightarrow \text{Backward.}$$

where ϕ = Activation function

b = bias.

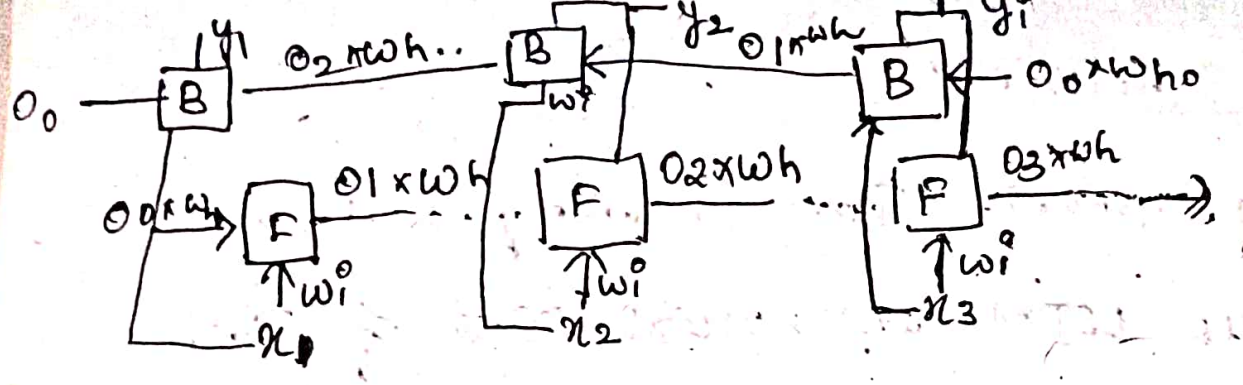
The hidden state at t is given by a combination of θ_1 (Forward) and On (Backward).

The o/p at any hidden state is defined as

$$O_h = H_t \times w_j + b_j$$

↑ new weight

In BRNN, since there is forward and backward passes happening simultaneously, updating the weights for two processes could happen at the same point of time. This leads to erroneous results.



→ optimization: Techniques to handle long term dependencies in CNN.

Forward RNN.

at time t.

$$O_1 = f(x_{11} \times w_i^o + O_0 who + b_1)$$
$$O_2 = f(x_{12} \times w_i^o + O_1 \times wh + b_2)$$
$$O_3 = f(x_{13} \times w_i^o + O_2 \times wh + b_3)$$

} Forward

Backward RNN.

~~$O_3 = f(x_{13} \times w_i^o + \ldots$~~

$$O_1 \leftarrow (x_{13} \times w_i^o + O_0 who + b_1)$$
$$O_2 = (x_{12} \times w_i^o + O_1 wh + b_2)$$
$$O_3 = (x_{11} \times w_i^o + O_2 wh + b_3)$$

A special hidden state which concatenates the o/p of FRNN & BRNN.

~~$y_3 = h_3 \times w_j + \ldots$~~

$$y_3 = (h_3 \times w_j + b_4)$$
$$y_2 = (h_2 \times w_j + b_5)$$
$$y_1 = (h_1 \times w_j + b_6)$$