27/05/2024:-

Recurrent Neural Network:

* why we use RNN, when we have ML models & ANN model.

Sequential Data Handling:-

- RNNs are specifically designed for handling sequential data, where the order of elements matters. Text, time series and speech are common examples of such data.

- unlike traditional machine learning models (like Naive Bayes), RNNs can capture dependencies across time steps, making them suitable for tasks where context matters.

Semantic Relationships:-

- Naive Bayes models treat word as independent features, ignoring their semantic relationships. RNNs, on the other hand, maintain hidden states that allow them to capture context and semantic meaning.

- For example, in language modelling, RNNs can learn that "Apple" refers to the fruit in one context (e.g., "I ate an apple") and the tech company in another (e.g., "Apple Inc").

Variable-length sequences:-

- RNNs handle sequences of varying lengths. This flexibility is crucial for text data, where sentences & documents can have different word counts.

- Traditional Artificial Neural Networks (ANN) require fixed input sizes, which isn't practical for text data without padding. while it's possible to use zero padding to make text data compatible with ANNs, this can lead to inefficiences and overfitting. RNNs with their ability to handle sequences of varying lengths are a better choice for such tasks.

Avoid overfitting :-

- overfitting occurs when a model learns to perform well on the training data but fails to generalize to unseen data.
- ANNs with fixed input sizes (using zero-padding) can lead to overfitting because they learn unnecessary weights for padded tokens. RNNs by contrast, adapt dynamically to the sequence length.

Long short-Term Memory (LSTM) and Gated Recurrent unit (GRU) :-

- RNNs often use variants like LSTMs and GRUs. These architectures address the vanishing gradient problem and allow for better long-term dependencies.
- LSTMs and GRUs have gates the control information flow, making them more effective at capturing context over longer sequences.

Applications :-

- Beyond spam classification and stock price prediction, RNNs excel in machine translation, sentiment analysis, speech recognition and more.
- Their ability to model context and sequential patterns makes them powerful tools in natural language processing (NLP).

In summary, RNNs are essential for handling data, capturing semantic relationships, and adapting to variable-length sequences. They overcome limitations of traditional models and play a crucial role in modern NLP and other time-series tasks.

RNN Forward Propagation :-

Data for RNN :- whenever we sent the data to "RNN" it will be in the form of (Time steps, Input features)

Ex:-

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

* let's say we have the movie data, and the input is in english. out ML, DL models they do not understand english, hence we need to convert the data into numbers / vectors.

* There are many ways to convert words into vectors
* For instance I am using one-hot encoding.
* As we see our data has 3 reviews, 3 reviews were made up of 5 unique words
* Those 5 unique words are making our vocabulary, basically any review in our system will be made out of these 5 words.

Note: In this data our vocabulary is a 5 word corpus. Meaning, I can create any word with 5 number representation

Ex:

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]

→ vectors

* Now, If I want to create a vector for any particular review from the data. It will be this way.

Review 1 [[1 0 0 0 0], [0 1 0 0 0], [0 0 1 0 0]] → Movie was good

Review 3 [[1 0 0 0 0], [0 1 0 0 0], [0 0 1 0 0], [0 0 0 1 0]] → Movie was not good

* Now, I will send every word one by one to RNN model
* when you send first row as input to an RNN, the shape of the i/p would be the length of the first sentence (number of time stamps) and the size of the vocabulary (number of distinct words in all rows). You would similarly for the subset rows with different sentence lengths, create inputs with varying number of time stamps based on the length of each sentence, but the no. of features (size of the vocabulary) would remain the same across all rows.

Note: * Review 1 (3, 5) → timestamps, → # of features

* Review 2 (3, 5)
* Review 3 (4, 5)

- In Keras we have "Batch-Size", "Time steps", "Input features"
- from our example :- Batch size = 3
  Time steps = 4
  Input features = 5   (3, 4, 5) → tensor

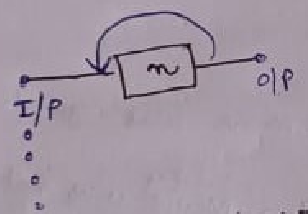
How RNN works:

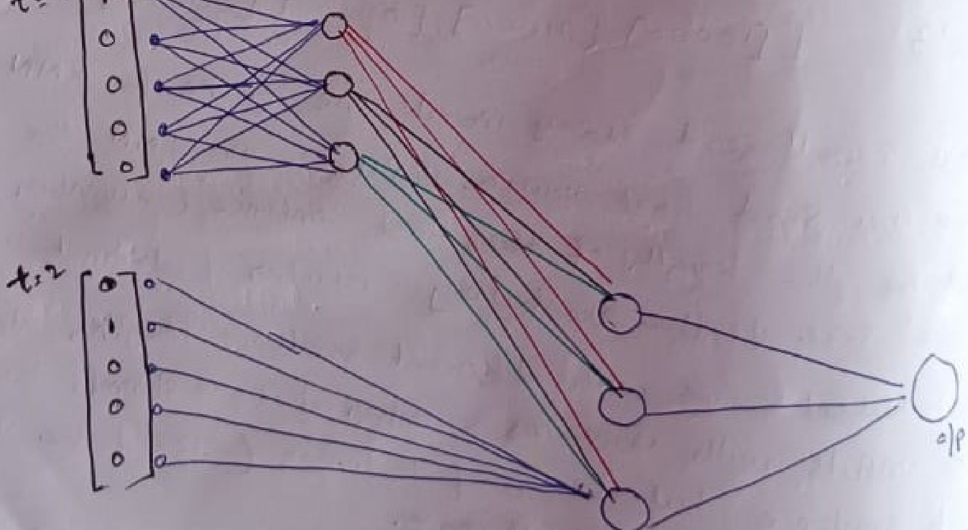
lets consider our example

	X Review	Y Sentiment
X_1 →	x_{11} Movie x_{12} was x_{13} good	1
X_2 →	x_{21} Movie x_{22} was x_{23} bad	0
X_3 →	x_{31} Movie x_{32} was x_{33} not x_{34} good	0

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

28/05/2024 :





1. <u>Input Representation</u> :-

* In a RNN, the input data is processed sequentially, one time step at a time.
* For your example, where you have 10 rows of data (sentences or sequences) and each row contains 3 words, let's consider the following:

  * Assume a vocabulary of 5 unique words, so each word can be represented as a vector (embedding) in a high-dimensional space.

  * The input shape for each time step (each word) will be based on the size of this word embeddings. If each word is represented by a 5-dimensional vector (for simplicity), then the input shape for a single time step is (5,)

2. <u>Time steps</u> and <u>Neurons</u> :-

* Each word in the sequence corresponds to a time step.
* Let's consider the first row (sequence) with 3 words: "word 1", "word2" and "word3".
* The processing proceeds as follows:

  - At time step (for "word 1"), the input is fed into the RNN. The RNN processes it using its hidden state and produces an output.
  - The output from the time step 1 is then used as part of the input for time step 2 (for "word 2")
  - This process continues until all words in the sequence have been processed.
  - The final output (after processing all time steps) can be used for various tasks (e.g., sentiment analysis, language modeling, etc.)

3. <u>Hidden State</u> and <u>Context</u>:

* The key feature of RNNs is their hidden state (also known as memory or context).
* At each time step, the hidden state is updated based on the current i/p and the previous hidden state.

* The hidden state captures information from [illegible]
to maintain context across the sequence.

* This context is crucial to understanding the order and dependencies with the data.

weights:-

* In RNN, each connection b/w nodes (input to hidden, hidden to hidden and hidden to output) has an associated weight.

* Ex:- • Input to hidden layers: 5 i/p nodes X 3 hidden nodes = 15 weights

• Hidden to hidden layer (if applicable): If there's a recurrent connection (e.g., from hidden node t-1 to hidden node t), it adds additional weights.

• Hidden to output layer: 3 hidden nodes X 1 out-put node = 3 weights

• Total weights: 15 (input to hidden) + 3 (hidden to output) = 18 weights

Biases:-

* Each hidden node and the output node typically have a bias term, there is one bias term per node.

* Biases are learnable parameters associated with each node (neuron) in the hidden layers and the output layer.

* The total no. of biases (excluding input nodes) is equal to the no. of hidden nodes plus output node.

Note:

Biases:- one bias per hidden node and one bias for the output node.

weights:- Sum of weights connecting input to hidden, hidden to hidden and hidden to output.

* Total biases (excluding i/p nodes) = No. of hidden nodes + 1 (for the o/p node)

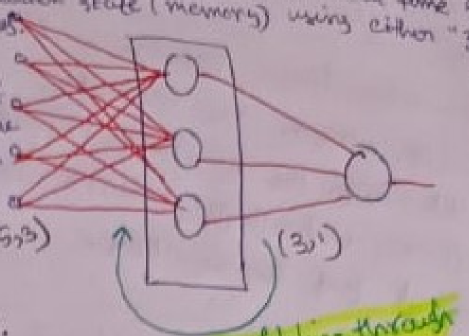
* Total weights = (I/P nodes * Hidden nodes) + (Hidden nodes * Hidden Nodes) + (Hidden node * o/p node)

RNN Forward Propagation :-

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

Note:- Initialization at $t=1$;

* To handle the absence of a previous o/p at the initial time step ($t=1$), we initiate the hidden state (memory) using either "zero" or small random values.
* This initialization allows the RNN to start capturing context from the begining of the sequence.



unfolding through time

- let's understand how RNN calculates the o/p.

* x_{11}, x_{12}, x_{13} are vectors (5 dimensional)
* we'll send them one by one into the Recurrent Neural Network.
* when RNN is doing forward Propagation, it follows one concept called "unfolding through time".

Ex:-

$\therefore$ $\boxed{wh = \text{hidden weights}}$

$x_{11} * w_i \rightarrow (1,3)$

$\boxed{}$ $\cancel{f(x_{11} * w_i)}$ $f(x_{11} + w_i + b_1) \Rightarrow o_1$ $\underset{(1,3)}{}$ --- wh --- $\boxed{}$ $f(x_{12} * w_i + o_1 wh + b_1) \Rightarrow o_2$

$\uparrow w_i$ $\qquad\qquad$ $\uparrow w_i$

$$\underbrace{\underbrace{(1,5)\,(5,3)}_{(1,3)}\;\underbrace{(1,3)\,(3,3)}_{(1,3)}}_{(1,3)}$$

$t=1$ $\quad x_{11}$ $\quad o_1 = f(x_{11} w_i + o_0 wh + b_0)$ $\quad t=2$

$o_2 = f(x_{12} w_i + o_1 wh + b_1)$

$o_3 = f(x_{13} w_i + o_2 wh + b_2) \Rightarrow \hat{y} = \sigma(o_3 . w_o) \Rightarrow y - \hat{y} \Rightarrow L = -y_i \log \hat{y}_i - (1-y_i) \log (1-\hat{y}_i)$

Note: The first word shape is (1,5)

* once t_1 is passed as a input and internally the hidden layer will be created in our case we have 3 nodes in internal hidden layer. So the shape would be (5,3)

* Recurrent layer acts as Artifical Neural Network, in every node it has an activation function. By default "Tanh" function.
* we'll do a dot Product of $\boxed{x_{11} \cdot w_i}$. As the shape of x_{11} is (1,5) and w_i shape is (5,3) it will result a shape (1,3) matrix.
* So, basically we are doing a dot Product within activation function output 1 with a shape (1,3)
  $f(x_{11} * w_i + b_1) \Rightarrow$ This will result to o_1 $\therefore$ output 1
* Now comes $t=2$, In $t=2$ we use the same weights and same network to send $t=2$ x_{12} as input
* This time the difference is we will Provide another input in $t=2$ i.e., This time the difference is we will Provide the previous o/p as current i/p. o_1 the previous o/p of $t=1$; I will send the previous o/p as current i/p.

It will go through a weighted Connection.

* This Process continues to $t=3$ $\therefore$ it will be $f(x_{13} * w_i + o_2 wh + b_1) \Rightarrow o_3$

$$\underbrace{\underbrace{(1,5)\,(5,3)}_{(1,3)}\;\underbrace{(1,3)\,(3,3)}_{(1,3)}}_{} \;\rightarrow\; (1,3)$$

$o_3 \rightarrow (1,3) * (3,1) = (1,1)$

* This (1,1) is scalar applying Sigmoid we'll get the o/p.

Integer Encoding:- This is a simple mapping of each unique word in corpus to a unique integer. For example, if we have a vocabulary of 10 words, you might assign the numbers 1 to 10 to each word. This encoding doesn't inherently capture relationships between words & their meanings. It's just a way to convert text to numbers.

Word Embeddings:- Embeddings, on the other hand, are Dense vector representations of each words in a continuous vector space. unlike integer encoding, embeddings are learned from data using techniques like "word2vec", "GloVe" & "FastText". these techniques use the context in which words appear in corpus to create vectors where words with similar meanings & usage patterns have similar embeddings. The idea is that words that are semantically similar will be closer together in vector space.

Types of RNN:-

Basically we have 4 types of RNN in the RNNs.

(i) Many - one - RNN:-

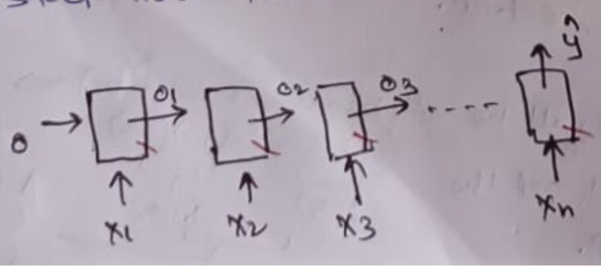
In a Many-to-one RNN architecture, the network processes sequential data, taking into account the information from the previous time steps upto the current time step. This process continues until all the inputs (e.g., words in a sentence) have been processed.

After processing all the inputs, the Many-to-one RNN produces a single output. This output can be scalar, A binary value (e.g., for binary classification tasks), a multi-class output (e.g., for multi-class classification tasks where the output is one of the several possible classes), & continous values (e.g., for regression tasks where the o/p is a continuous value).

Note:- Input → Sequential data → such as sentences, characters, time series data.

o/p :→ Non-sequential o/p

Ex:- Sentiment Analysis, Movie Review Prediction, medical diagnosis, Spam Detection, Stock Price Prediction.



∴ These blocks are called recurrent neural networks.

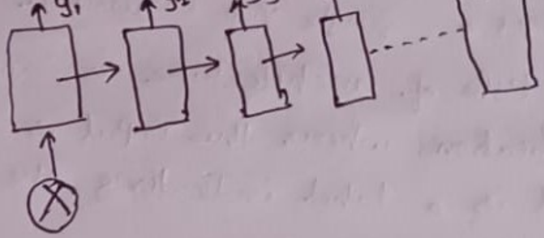
[Many-to-Many RNN]

(ii) One to Many:-

* one-to-Many RNN architecture, the network takes a single non-sequential input and Processes it to Produce an output. This output is then used as the input to multiple layers or blocks of RNN, leading to the generation of multiple outputs.

* one example of a one-to-Many RNN application is "image captioning". In this task, the RNN first Processes the entire image to understand its Content and Context. Then, the output from this initial Processing is fed into another RNN (often a different type, like an LSTM, GRU) to generate a sequence of words that describe the image. The final output is a caption that expliams the content of the image.

* Another example is music generation, where the RNN Processes an initial musical input (such as few notes & chords) and then generates a sequence of musical notes that extend the original input, creating a longer musical piece. Each output from the RNN can represent a musical note & a set of notes, and the sequence of outputs forms a musical composition.



→ Input (normal) Non-sequential
Ex:- (image)

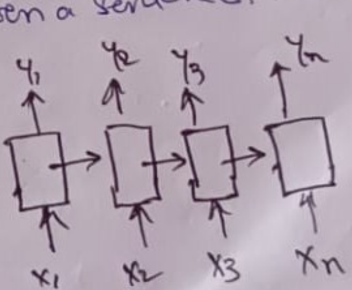
→ output → sequential

(iii) Many-to-Many RNN:-

* In many-to-Many RNN architecture, also known as sequence-to-sequence RNN, there are two main variants based on the length of input and output sequences.

(1) Same-length-Variant:- In this variant, the input and output sequences have same length. The RNN Processes the input sequence and generates an output sequence of the same length.

Ex:- Part-of-Speech Tagging (PoS):- Given a sentence, Predict Part of Speech for each word in a sentence.



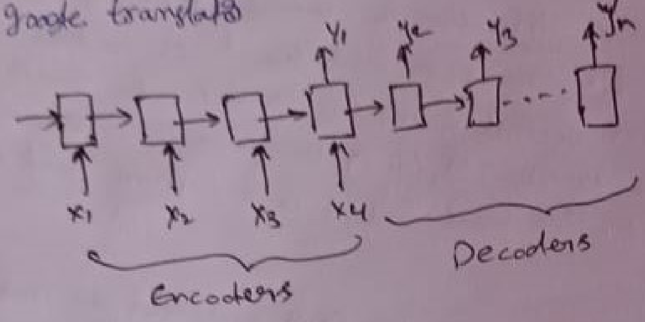
(Input sequence == output sequence)

My	Name	is	Somanath
↓	↓	↓	↓
Pronoun	noun	verb	Noun

(ii) Variable length (Different length Variant) :

* In this Variant, the input and output sequences can have different lengths. This is commonly used in tasks like machine translation, where the length of the input sentence can differ from the length of the output sentence.

Ex:- google translator



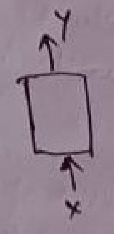
(iv) one-to-one :

* In a one-to-one neural architecture, there is no sequence processing & recurrence involved. The network takes a single non-sequential input and produces a single non sequential output.

This type of architecture is commonly used for tasks such as "image-classification, where the input is an image (non-sequential data) and the output is a label indicating the class of image (also non-sequential data)

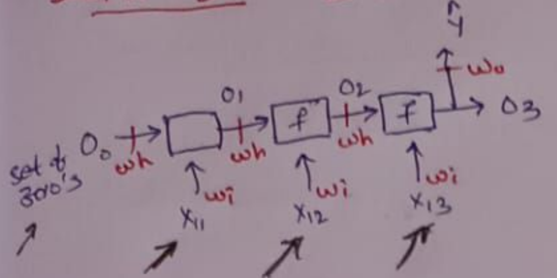
* In Image classification, the neural network processes the entire image at once, without considering any spatial & temporal relationships b/w pixels and produces a prediction for the class of the image.

Input → Non-Sequential
output → Non-Sequential



Back Propagation in RNN through Time

01/06/2024 :-



Sentiment Analysis :- (1/0) Many-to-one RNN

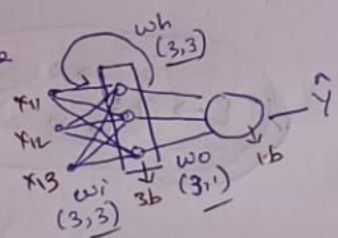
Text	O/P
Cat mat rat	1
rat rat mat	1
mat mat cat	0

$\Rightarrow$

	X	Y
x_1	[1 0 0] [0 1 0] [0 0 1]	1
x_2	[0 0 1] [0 0 1] [0 1 0]	1
x_3	[0 1 0] [0 1 0] [1 0 0]	0
⋮		
x_n		

Text to Numbers

- Count of vocabulary $\Rightarrow$ [cat mat rat] # 3
  [1 0 0] [0 1 0] [0 0 1]
- Convert the words with Number that were assigned
- one by one we will send the review as per the time steps to the as a Input to the machine.



— Parameters

$$O_1 = f(O_0 w_h + x_{11} \cdot w_i + b_0)$$

$$O_2 = f(x_{12} w_i + O_1 w_h + b_1)$$

$$O_3 = f(x_{13} w_i + O_2 w_h + b_2)$$

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

$$L = Y - \hat{Y}$$

$$L = -Y_i * \log \hat{Y}_i - (1 - Y_i) * (1 - \log \hat{Y}_i)$$

- Since loss was determined, now we have to minimize the loss using Back Propagation using some optimizing techniques.
- using optimizers we have to find those values of "wo wi wh" so that the loss is minimum. $L\downarrow$

Traditionally we can use Gradient Descent

$$\boxed{\begin{aligned} w_{i\,new} &= w_{i\,old} - \eta \frac{\partial L}{\partial w_{i\,old}} \\ w_{h\,new} &= w_{h\,old} - \eta \frac{\partial L}{\partial w_{h\,old}} \\ w_{o\,new} &= w_{o\,old} - \eta\, \partial L / \partial w_{o\,old} \end{aligned}}$$

w_o - ~~bias~~ output
w_i - weights
w_h - hidden state

* $w_{o_{new}} = w_{o_{old}} - \eta \frac{\partial L}{\partial w_{o_{old}}}$

* $w_{h_{new}} = w_{h_{old}} - \eta \frac{\partial L}{\partial w_{h_{old}}}$

* $w_{i_{new}} = w_{i_{old}} - \eta \frac{\partial L}{\partial w_{i_{old}}}$

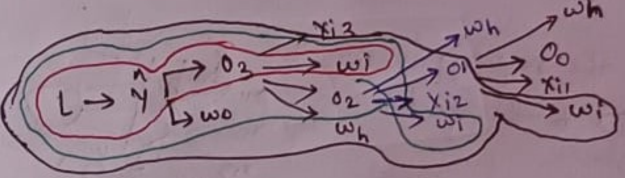
$\frac{\partial L}{\partial w_o}$ ⇒ find derivative

⇒ $w_{new} = w_{old} - \frac{\partial L}{\partial w_{old}}$

⇒ $\frac{\partial L}{\partial w_{o_{new}}} = \frac{\partial L}{w_o} \frac{\partial o_3}{\partial w_o} \cdot \frac{\partial w_o}{\partial w_h} \cdot \frac{\partial w_h}{\partial w_i}$

⇒ we can assume answer as it depends on the weight Initialization.

∴ $\frac{\partial L}{\partial w_i}$



Note: L & wi there is 3 path relationship

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

"3" units is because our corpus is of 3 units/words and so, these many outputs was sent.

- In the text we cannot ~~Predict~~ say how many words it be! hence we write the formula as.

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

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

$$\Rightarrow \boxed{\frac{\partial L}{\partial w_h} = \sum_{j=1}^{n} \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial o_j} \frac{\partial o_j}{\partial w_h}}$$

Limitations of Simple RNN :-

RNNs are used for sequential data like text & time series, where the current output depends on previous steps in the sequence. The two main issues with basic RNNs are "long-term dependencies" and "vanishing/exploding gradients" during training.

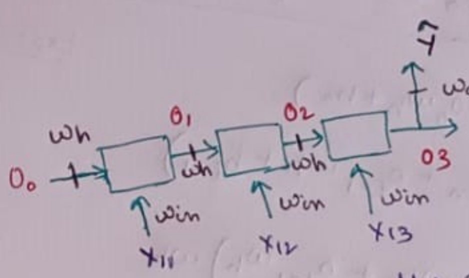
1. Long - Term Dependencies :-
   RNNs can have difficulty learning long-term dependencies, meaning they might struggle to connect information from earlier steps to later ones in a sequence. This can lead to information loss & inefficiency in learning long-range patterns.

2. Exploding / Vanishing Gradients :-
   During Back Propagation through time (BTT) gradients in RNN can either explode (become too large) & vanish (become too small) as they are propagated through many time steps. This can make learning difficult especially for long sequences.

Problem # 1 → Problem of long term dependency (vanishing) :-

Input	output
1 0 1	1
0 0 1	0
1 1 1	0
0 0 0	0

· 3 timestamps



* BackPropagation is basically done to reduce the loss so that the L will be minimum.

* L ↓

* w_h, w_{in}, w_{out} → find these values of these so that the Loss will be minimum.

$$\ast \quad w_{in_{new}} = w_{in_{old}} - \eta \frac{\partial L}{\partial w_{in_{old}}}$$

$$\ast \quad w_{out_{new}} = w_{out_{old}} - \eta \frac{\partial L}{\partial w_{out_{old}}}$$

$$\ast \quad w_{h_{new}} = w_{h_{old}} - \eta \frac{\partial L}{w_{h_{old}}}$$

$$\frac{\partial L}{\partial w_{in}} = \underbrace{\boxed{\frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial o_3} \frac{\partial o_3}{\partial w_{in}}}}_{\text{Short term dependency}} + \boxed{\frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial o_3} \frac{\partial o_3}{\partial o_2} \frac{\partial o_2}{\partial w_{in}}} + \underbrace{\boxed{\frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial o_3} \frac{\partial o_3}{\partial o_2} \frac{\partial o_2}{\partial o_1} \frac{\partial o_1}{\partial w_{in}}}}_{\text{long term dependency}}$$

- The above derivative is for 3 time steps. what if we have 100 time steps i.e.,

$$\boxed{\frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial o_{100}} \frac{\partial o_{100}}{\partial o_{99}} \cdots \frac{\partial o_2}{\partial o_1} \cdot \frac{\partial o_1}{\partial w_{in}}}$$

As the number of time steps increases in an RNN, the challenge of capturing long-term dependencies increases. RNNs have a tendency to focus more on short-term dependencies, as they are updated more frequently and have a more direct impact on the current prediction.

During backpropagation, the model updates its weights based on the loss calculated at the end of the sequence. If the model struggles to capture long-term dependencies, it may not be able to accurately update its weights to improve performance over long sequences. This can lead to difficulties in learning and memorizing information from earlier steps, especially the sequence length grows.

$$\boxed{\frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial o_{100}} \prod_{t=2}^{100} \left(\frac{\partial o_t}{\partial o_{t-1}}\right) \frac{\partial o_1}{\partial w_{in}}}$$

Ex:1 Marati is spoken in Maharastra

Short term dependency

Ex:2 Maharastra is a beautiful place. I went last year. But, I couldn't enjoy properly because I don't understand marathi

long term dependency.

$$o_1 = \tanh(x_{i1} w_{in} + o_0 w_h + b_0)$$

$$o_t = \tanh(x_{it} w_{in} + o_{t-1} w_h + b_{t-1})$$

$$\frac{\partial o_t}{\partial o_{t-1}} = \underbrace{\tanh\left(x_{it} w_{in} + o_{t-1} w_h\right)}\, w_h$$

∴ As we derivative of tanh is $0 \leftrightarrow 1$

Substitute this value in the above compact derivative

$$\frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial o_{100}} \prod_{t=2}^{100} \left(\tanh(x_{it} w_{in} + o_{t-1} w_h) w_h\right) \frac{\partial o_1}{\partial w_{in}}$$

Note: Assume for a while we have considered a value for "w_h" b/w "0 to 1" meaning

$$\frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial o_{100}} \boxed{\prod_{t=2}^{100} \left(\underbrace{\tanh\left(x_{it} w_{in} + o_{t-1} w_h\right)}_{0 \text{ to } 1} \underbrace{w_h}_{0 \text{ to } 1}\right)} \frac{\partial o_1}{w_{in}}$$

time step ≥ 100

→ we will multiply more than 100 times then the entire thing will be a very small number close to "0"

So, that the remaining terms will become minimal and the total equation will be close to "0".

Solutions :.

1) we can try different activation functions like "ReLU, leaky ReLU" as their derivatives are not limited b/w 0 to 1.

2) Better weight initialisation techniques.

3) Skip RNNs

4) LSTM (Long Short term Memories)

Problem #2 unstable training (Exploding Gradients)

Solutions :.

1) Gradient clipping :. Meaning, we are clipping the Gradient to a maximum value, so that the gradients will not above the threshold.

2) Controlled learning Rate

3) LSTMs