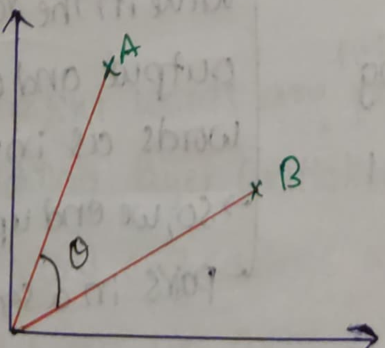
## Word2Vec:

Word2vec uses a neural network model to learn word associations from a large corpus of text. Once trained, such a model can detet synonymous words or suggest additional words for a partial sentence. As name implies word2vec converts word into vector.

→ The vector are chosen carefully such that they capture the semantic and syntactic qualities of words; such as a simple mathematical function [cosine similarity] can indicate the level of semantic similarity b/w the words represented by those vectors.

Cosine similarity:

→ Let's consider 2 points and find the cosine angle between them.



$$\cos\theta = \frac{A \cdot B}{||A|| \cdot ||B||}$$

If $\theta = 45^\circ \Rightarrow \cos 45^\circ = 0.7071$

~~so, points are~~

$$\text{Distance} = 1 - \text{Cosine Similarity}$$
$$= 1 - 0.7071$$
$$\approx 0.29$$

⇒ So, the points are 0.29 (29%) similar

⇒ If $\theta = 0^\circ$, then points are highly [similar ∵ Distance = 1]

→ Word2vec Can utilize either of two model architectures to produce these distributed representations of words

1. Continuous Bag-of-Words (CBOW)

2. Skip-gram

1. Continuous Bag of Words (CBOW):-

This method takes the Context of each word as the input and tries to predict the word Corresponding to the Content.

Example:- Let's consider the sentence.

"Word2vec has a deep learning model woking in the backend".

→ Lets consider Content Window size = 3, we will have pairs like

| | |
|---|---|
| [word2vec, a] | has |
| [has, deep] | a |
| [deep, model] | learning |
| [learning, working] | model |
| ⋮ | ⋮ |
| ⇓ | ⇓ |
| Inputs | outputs |

→ Here we are considering middle value in the range as output and other words as inputs

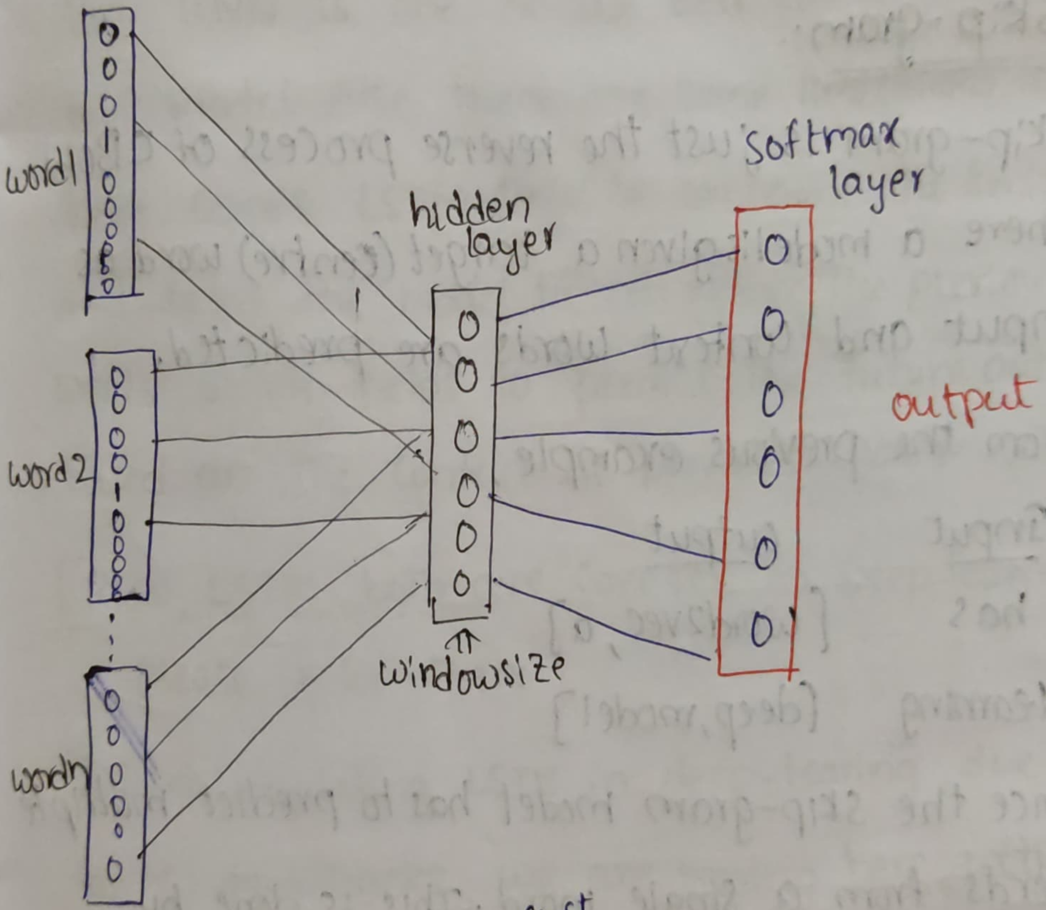
→ so, we end up creating pairs in a sentence.

→ We will use these inputs & output data into a Deep learning Model (Neural Network) to predict these target words based on the Content words.

→ The input to the hidden layer (Neural network), we use Bag of words to convert the word into vectors

Ex:- "deep" word → to vec → 0 0 0 1 0 0 0 0 0 0

"learning" → 0 0 0 0 1 0 0 0 0 0



→ The Content words are first passed as an input to an embedding layer (initialized with Some random weights)

→ The word embeddings are then passed to a lambda layer where we average out the word embeddings.

→ We then pass these embeddings to a dense Softmax layer that predicts our target word. We match this with our target word and Compute the loss and then we perform backpropagation with each epoch to update the embedding layer in the process.

→ We can extract out the embeddings of the needed words from our embedding layer, once the training is completed.

## 2. Skip-gram:-

Skip-gram is just the reverse process of CBOW.

→ where a model is given a target (centre) word as input and Context words are predicted.

→ From the previous example,

| Input | output |
|---|---|
| has | [word2vec, a] |
| learning | [deep,model] |

→ Since the skip-gram model has to predict multiple words from a single word. This is done by

⇛ creating positive i/p samples & negative i/p samples.

→ Positive i/p samples will have training data in this form [(target, Content),1] where target is the centre word, Context represents Context words and label 1 indicates if it is a relevant pair.

→ Negative i/p samples will have training data in the same form [(target, random), 0]. 0 indicates an irrelevant pair.

* We can use deep neural networks for training the hidden layer of the model in word2vec.

* But RNN is one of the best use case for NLP model. Also, there are some limitations in RNN.

* Here comes LSTM RNN to overcome the limitations and helps the model to remember the previous words which helps to predict the future output based on the content of previous words.

[RNN, LSTM both are covered in Deep learning, please refer there]

* After completing LSTM in deep-learing, due to some drawbacks, we are moving here with Bi-directional LSTM.

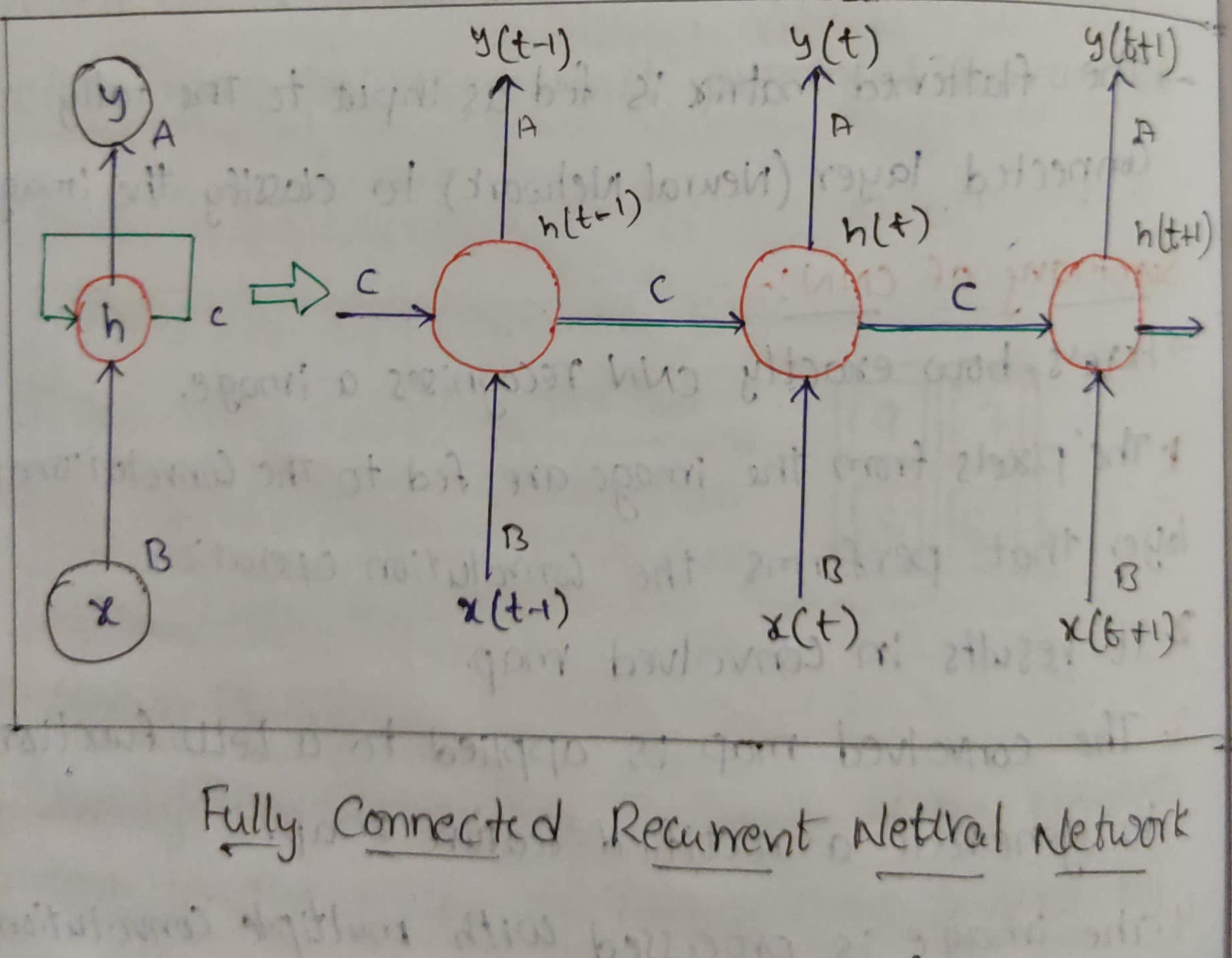
* LSTM's were failing when to try's to get context from the future word.

Ex: Ravi likes to eat ........ in Hyberabad.

→ Here the blank should be filled based on future word content (Hyberabad). So, Biryani is famous in Hyberabad. LSTM Can't predit this types.

# Recurrent Neural Network(RNN)

RNN works on the principle of saving the output of a particular layer and feeding this back to the input in order to predict the output of the layer.



Fully Connected Recurrent Netural Network

→ A, B, C are the parameters of the Network.

→ Here, "x" is the input layer, "h" is the hidden layer, and "y" is the output layer. A, B, C are the network parameters used to improve the output of the model. At any given time t, the current input is a combination of input at $x(t)$ and $x(t-1)$. The output at any given time is fetched back to the network to improve on the output.

$$\boxed{h(t) = f_c(h(t-1), x(t))}$$

$h(t)$ → new state

$f_c$ → function with parameter c.

$h(t-1)$ → old state

$x(t)$ → i/p vector at time step t

*Why RNN's?

RNN were created because there were a few issues in the feed-forward neural network (ANN's).

→ Cannot handle sequential data

→ Considers only the current input.

→ Cannot memorize previous inputs.

→ The solution to these issues is RNN. Which can handle sequential data, accepting the current i/p data and previously received i/ps and also can memorize previous i/p's due to their internal memory.

## * How does RNN work?

In RNN, the information cycles through a loop to the middle hidden layer.

→ The input layer 'x' takes in the input to the neural network and processes it and passes it onto the middle layer.

→ The middle layer 'h' can consist of multiple hidden layers, each with its own activation functions, weights and biases.

→ The RNN will standardize the different activation functions and weights, biases so that each hidden layer has the same parameters. Then, instead of creating multiple hidden layers, it will create one and loop over it as many times as required.

Applications:-

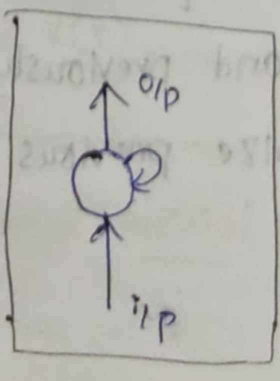
1. Image Captioning
2. Natural Language Processing
3. Time Series prediction.

Types of RNN:-

1. One to One RNN:

Single input and Single output

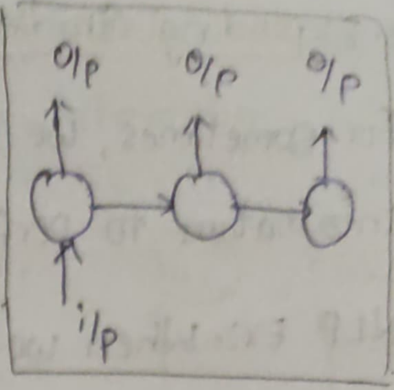
Ex:- Image Classification.



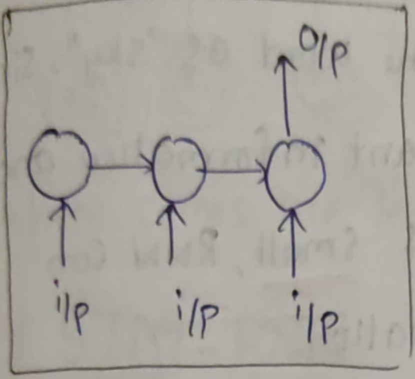
2. One to Many RNN:-

single input and multiple outputs.

Ex:- Image Caption.



3. Many to One RNN:-



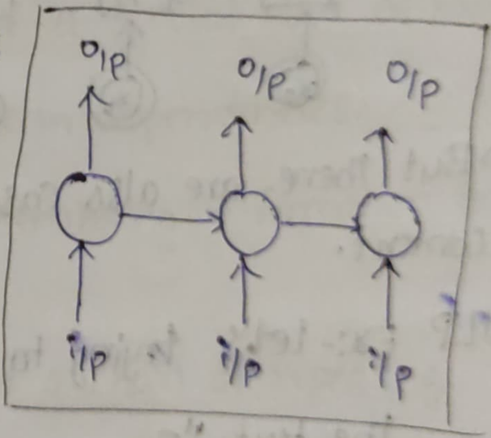
RNN takes a sequence of inputs and generates single output.

Ex:- Sentiment analysis (which takes many inputs and tells whether (+ve) or (-ve) sentiment

4. Many to Many RNN:-

RNN takes sequence of inputs and generates a sequence of outputs.

Ex:- Machine translation.



Issues of Standard RNN's

1. Vanishing Gradient Problem

RNN suffer from the problem of vanishing gradients The gradients carry information used in the RNN, and when the gradient becomes too small, the parameter updates become insignificant. This makes the learning of long data sequences difficult.

——— loss of information through time ———→

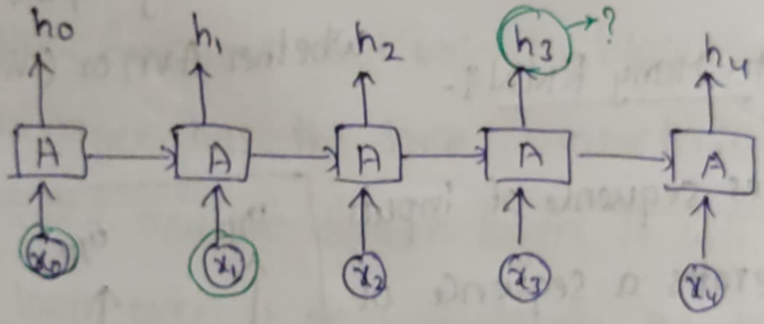
## 2. Exploding Gradient Problem:-

For sometimes, we only need to look at recent information to perform the present task.

NLP Ex:- When we try to predict the last word

"The clouds are in the ____"

→ RNN can able to predict the word as "sky". Since the gap between the relavant information and the place that it's needed is small, RNN can learn to use the past information.

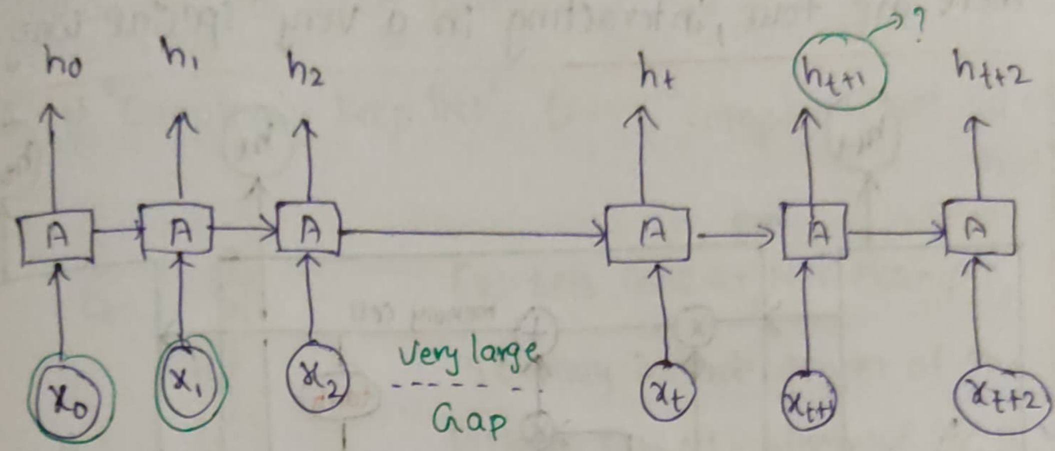


→ But there are also cases where we need more context.

NLP Ex:- Let's trying to predict the last word in the text "I grew up in France. I speak fluent ? "
French

→ The Recent information the next word is probably the name of a language, but if we want which languge, then we need the Context of "France" from previous sentence. Here gap between relavent information and the place it is needed is Very large.

* Unfortunately, as that gap grows, RNNs become unable to learn to connect the information.



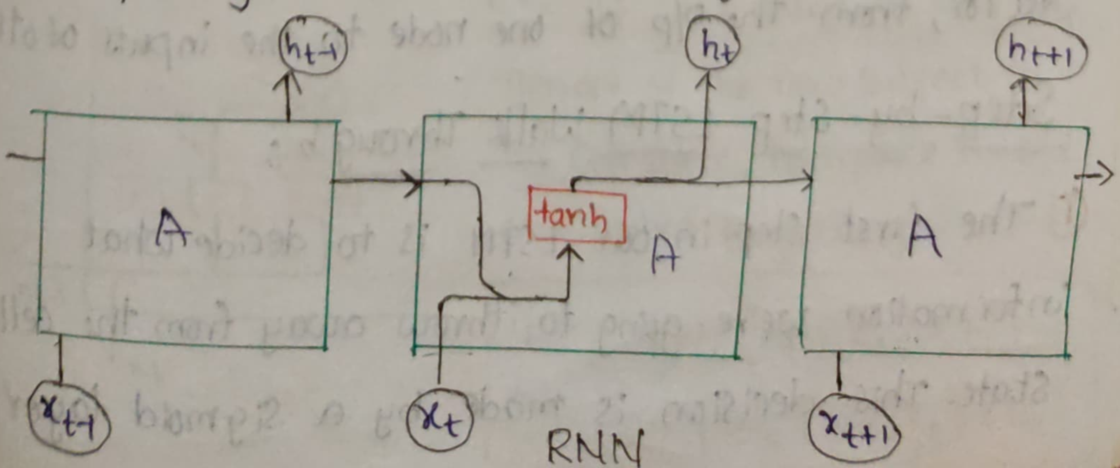
→ Here comes the legend LSTM RNN.

## LSTM-RNN

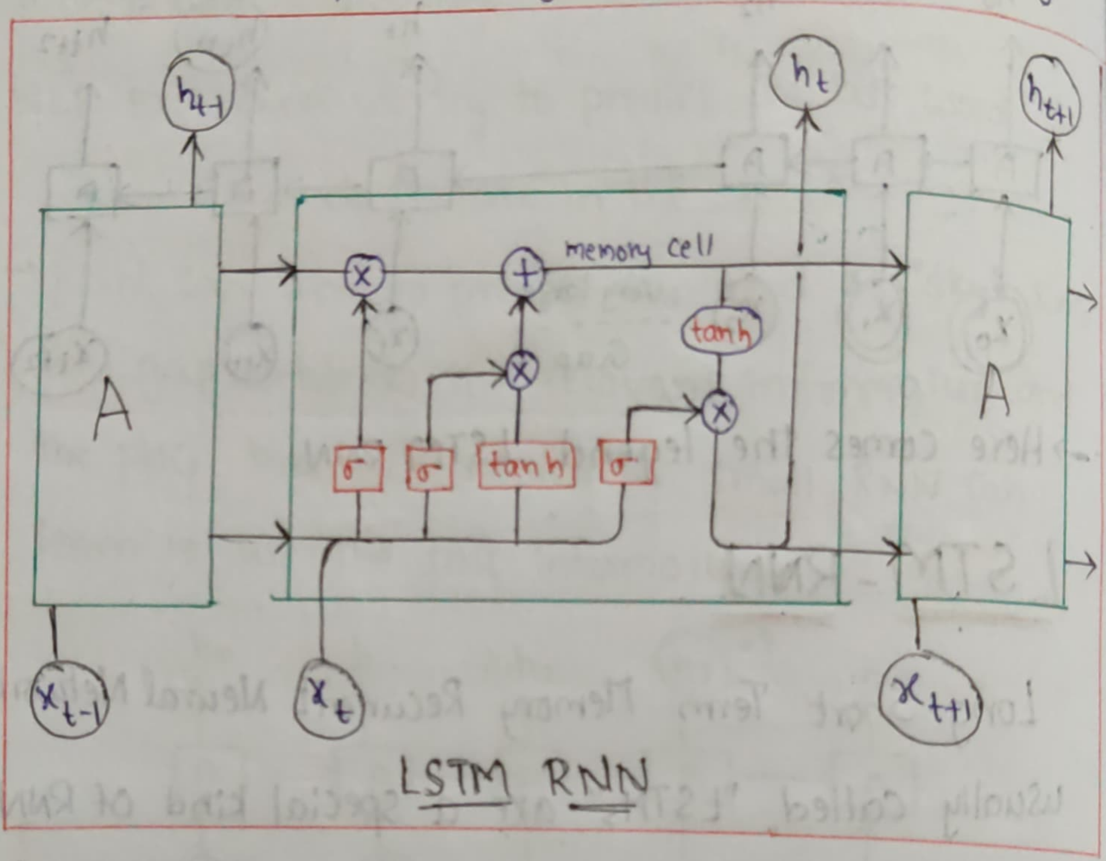
Long Short Term Memory Recurrent Neural Networks usually called "LSTMs" are a special kind of RNN capable of learning long-term dependencies.

→ LSTMs have ability of remembering information for long periods of time.

→ All RNN's have the form of a chain of repeating modules of neural network. In standard RNNs, the repeating module will be a single tanh layer.

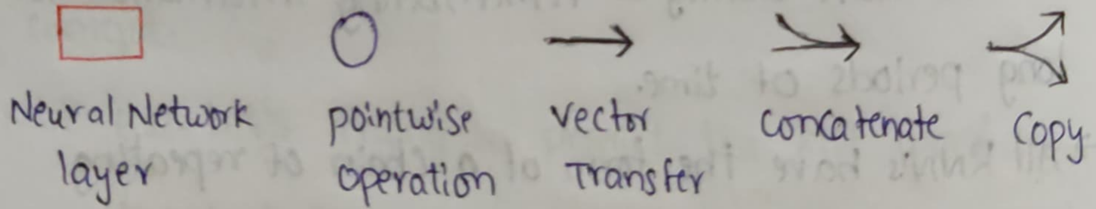


→ LSTMs also have this chain like structure, but the repeating module has a different structure there are four, interacting in a very special way.



LSTM RNN

Notations:

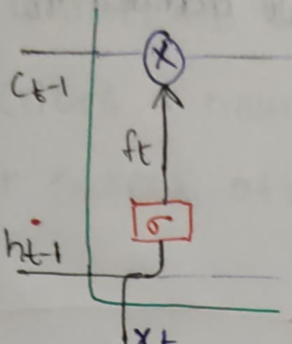


→ In the above diagram, each line carries an entire vector, from the o/p of one node to the inputs of others

Step-by-Step LSTM Walk Through:

① The first step in our LSTM is to decide what information we're going to throw away from the cell state. This decision is made by a sigmoid layer called the "forget gate layer." It looks at $h_{t-1}$ and $x_t$, and outputs a number between 0 and 1 (sigmoid($\sigma$)) for each number in the cell state $C_{t-1}$.

1 → "Completely keep this" 0 → "completely get rid of this"

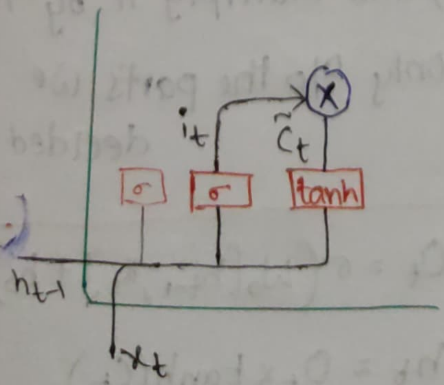


forget-gate

Ex:- Lets consider prev. NLP example, It may include gender of the present subject. When we see a new subject we want to forget gender (old subject).

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

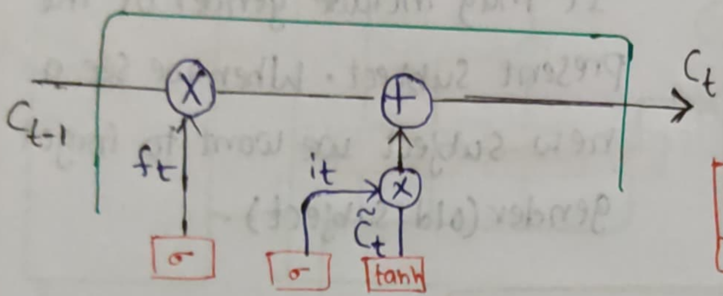
② The next step is to decide what new information we're going to store in the cell state. This has two parts. A sigmoid layer called "input gate layer" decides which values we'll update. Next, a tanh layer creates a vector of new candidate values, $\tilde{C}_t$ that could be added to the state. In next step, we'll combine these two to create an update to the state.



Ex:- If we'd want to add the gender of the new subject to the cell state, to replace the old one we're forgetting.

$$\boxed{\begin{aligned} i_t &= \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \\ \tilde{C}_t &= \tanh(W_C \cdot [h_{t-1}, x_t] + b_C) \end{aligned}}$$

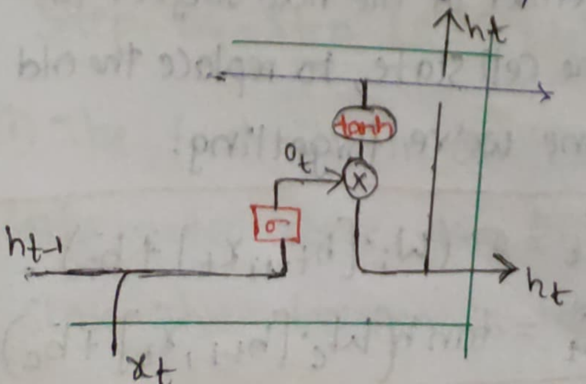
③ Now, we will update the Old Cell State, $C_{t-1}$ into the new cell state $C_t$. We multiply the old state $f_t$, forgetting the things we decide to forget earlier. Then we add $i_t \times \tilde{C}_t$. This is the new Candidate Values, scaled by how much we dedided to update each state value.



$$\boxed{C_t = f_t \times C_{t-1} + i_t \times \tilde{C}_t}$$

Ex:- Where we'd actually drop the info old Subject (gender) and add the new information, as we decided prev.

④ Finally, we need to decide what we're going to output. This output will be based on ourcell state, but will be a filtered version. First, we run a Sigmoid layer which decides what parts of the cell state we're going to output. Then, we put the cell state through tanh (to push values between -1&1) and multiply it by o/p of the sigmoid gate, so we only o/p the parts we decided to.



$$\boxed{\begin{aligned} O_t &= \sigma(W_o[h_{t-1}, x_t] + b_o) \\ h_t &= O_t \times \tanh(C_t) \end{aligned}}$$

Ex:- NLP → "John played tremendously well and won for his team. For his contributions, brave ----- was awarded player of the match.

→ There could be many choices for empty space. The current i/p brave is adjective, and adjective describes a noun(John). So, "John" could be the best output after brave.