Recurrent Neural Networks

...

Or RNNs

Sequential Data

- Sequential Data : in which the order/sequence matters
- A commonly seen example of sequential data is time series
- Time Series: A sequence of data sampled equally spaced over time
- e.g. Stock Market Prediction

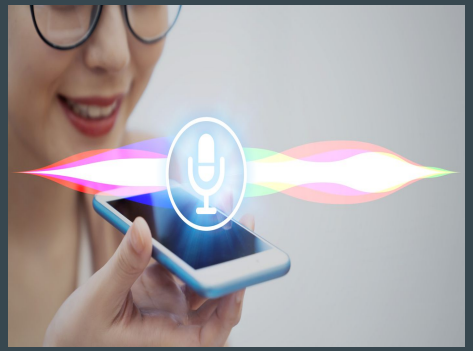
  In stock market prediction, we are given a time series of stock market prices for the last n days, and we are interested in finding the price for next day.

- As observed from the example, the sequence/order

of the data is extremely important.

The R in RNN...

- RNN stands for ‘Recurrent Neural Network’.
- The extra ‘R’ gives these nets the ability to deal with Sequential Data which is not possible with simple Feed-Forward Neural Networks we saw earlier.
- Applications of RNN in which sequential data is important:



Speech Recognition

Text Translation

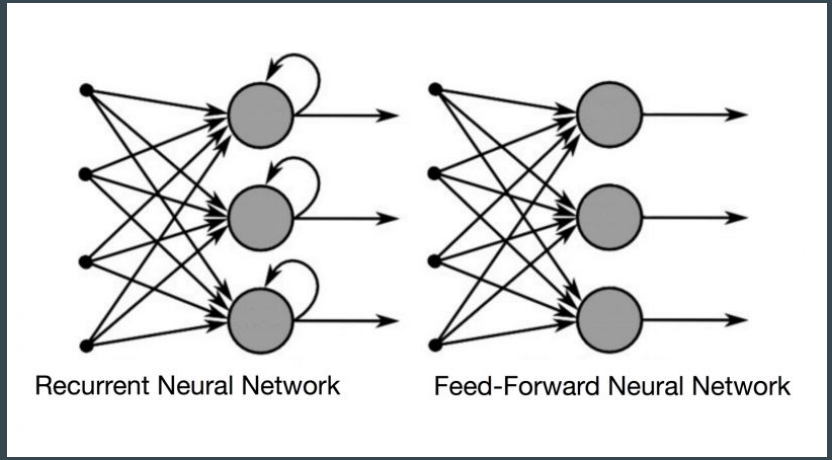
DNA Sequence Classification

Sequential Data and RNN

- RNN is a type of neural network, which is specifically used for dealing with sequential data, i.e. , where the order of data is equally important as the other features.
- The output at a certain time, depends not only on the input at that time, but also on the outputs at the previous time instants.

How is it different from Feed-Forward Network?

- Feed-forward neural networks do not re-use the output from any of the input they receive and are bad at predicting what's coming next
- On the other hand, in RNNs, when it makes a decision, it considers the current input and also what it has learned from the inputs it received previously.



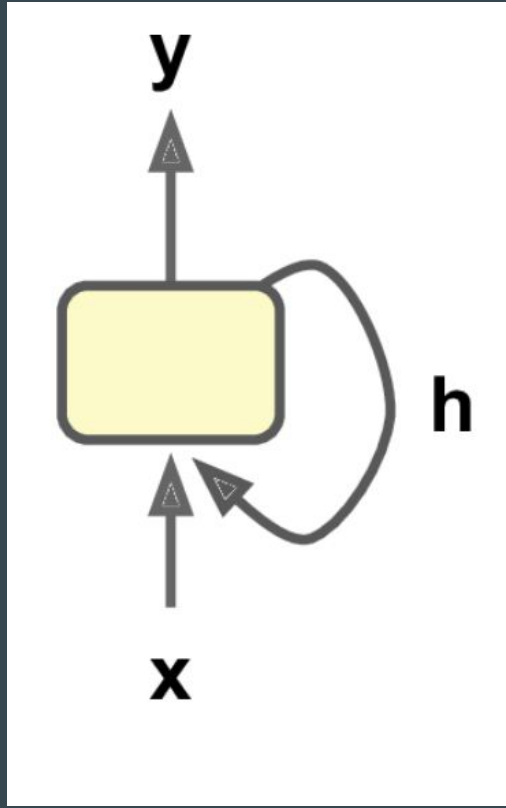
A simple RNN cell

A simple RNN Cell has two inputs:

- The input vector X from the current time-step
- The hidden state vector h, from the previous time step.

The computation in the cell is as follows:

- Each of the inputs is multiplied with a corresponding weight and added along with a bias.
- This is then passed through an activation layer to give the output for that time state as well as give the hidden state vector for the next time-step

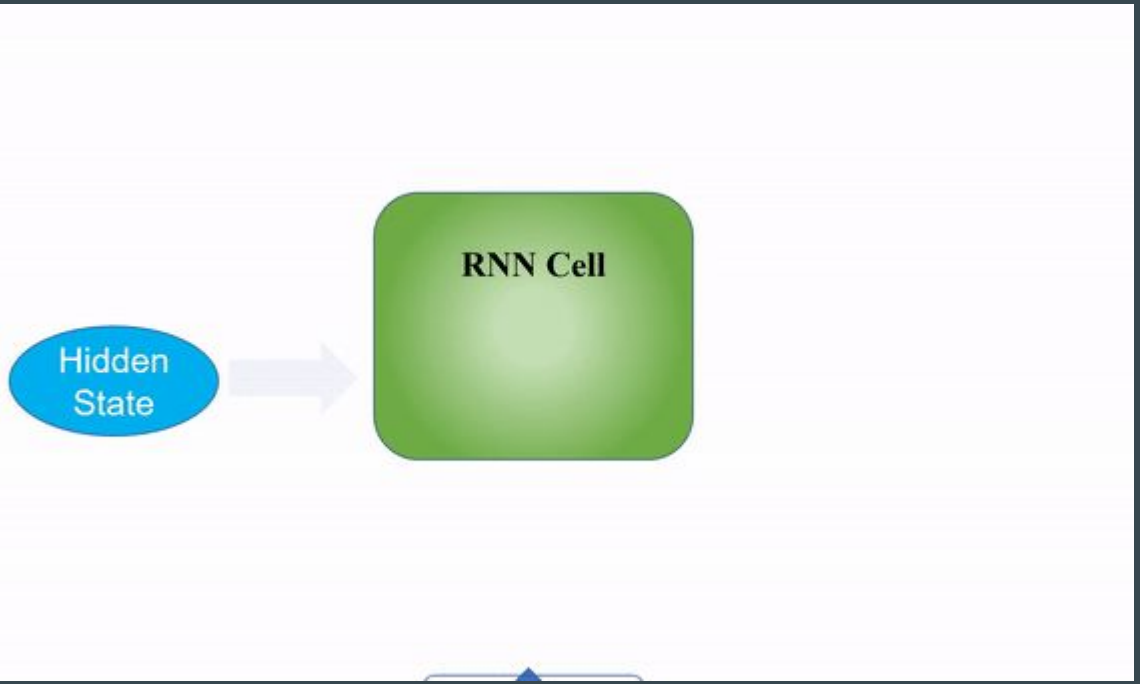


Explaining the technical terms

- **Time-Step :** In an RNN, every time we provide an input, there is an occurence of the RNN cell. A time-step is just a given occurence of the RNN cell.

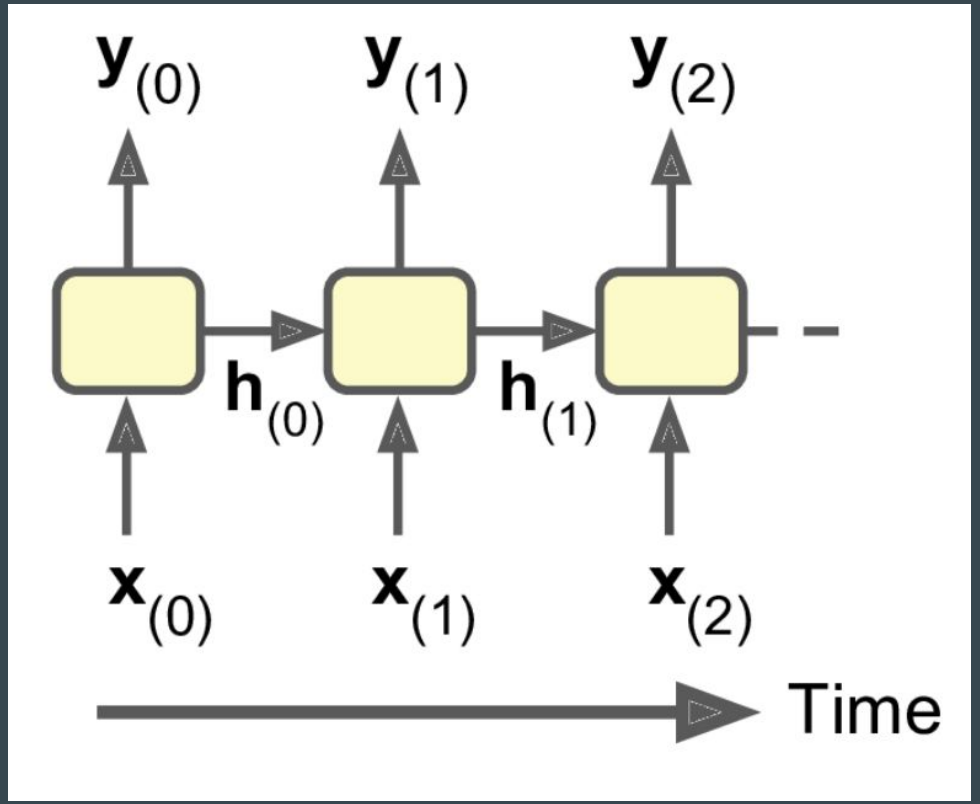
- **Input Vector X** : This is the input that we are providing the RNN at a given time-step.

- **Hidden State Vector h**: This is essentially a copy of the output that was created from the previous time-step.


RNN Cell
Hidden State

Unrolling the RNN Cell

- Unrolling an RNN cell means showing all the different time-steps of that cell
- Every time-step outputs some value and the RNN uses this value for the next time-step
- Note that the input for the RNN is different for each time-step
- Also note that this is the same RNN unit, just shown at different time-steps



Some Math for Better Understanding

$$\mathbf{y}_{(t)} = \phi(\mathbf{W}_x{}^T \cdot \mathbf{x}_{(t)} + \mathbf{W}_y{}^T \cdot \mathbf{y}_{(t-1)} + \mathbf{b})$$

- $y_{(t-1)}$ is the output from the previous time step
- W_y is the weights associated with the previous output
- x_t is the input vector for the current time step
- Wx is the weights associated with the current input
- b is the bias term
- Φ is the activation function for the combined input
- y_t is the output for the current time step

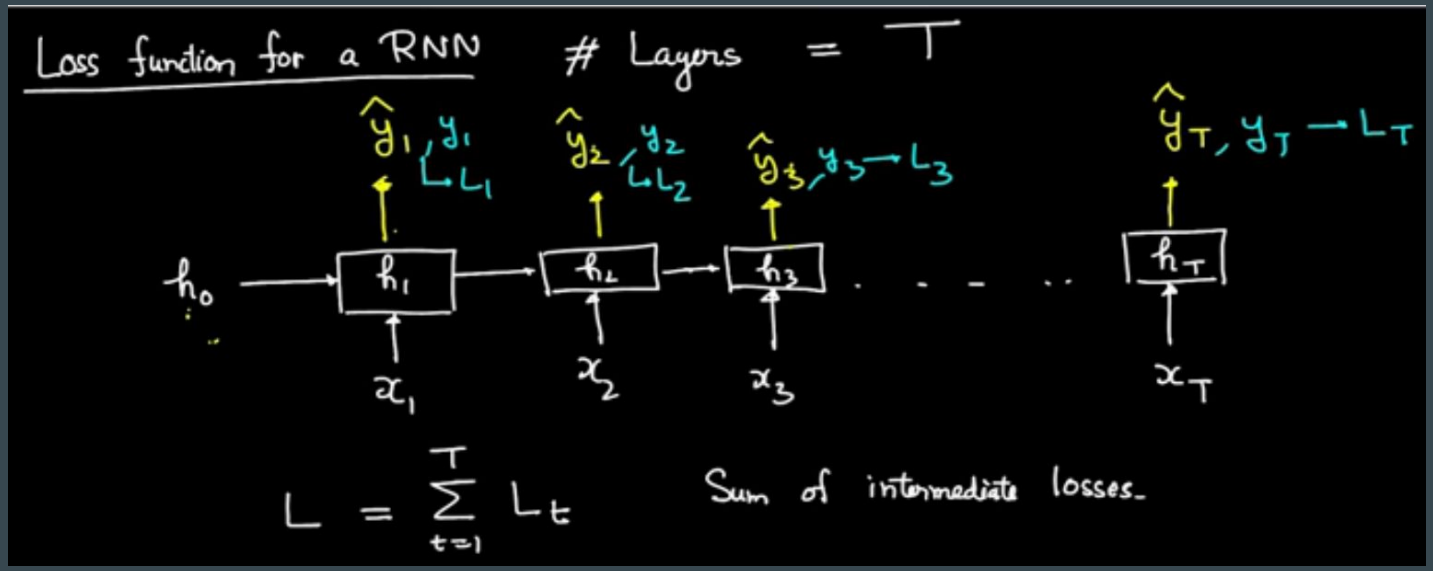
Training an RNN

...

Get ready for some time-travel!!

The Loss Function for RNN

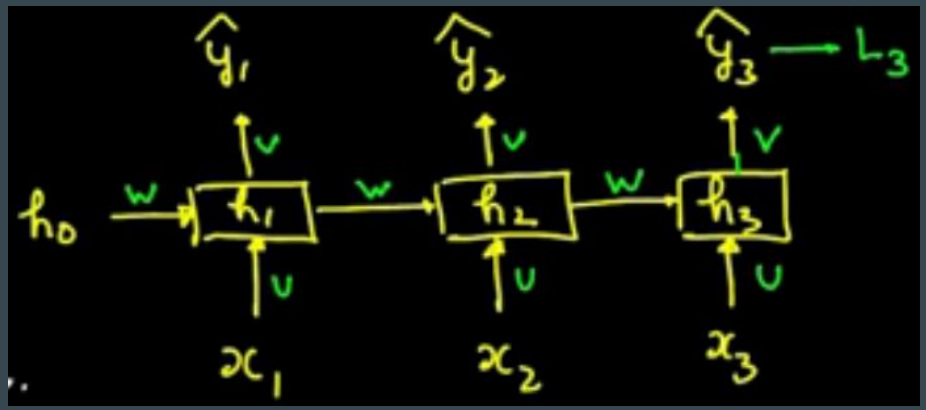
- Since we can have an output corresponding to each time step, the loss function also has to incorporate this.
- The Unrolled RNN helps us understand this better.



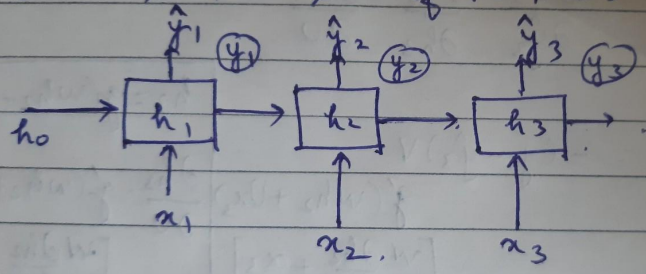
Backprop through time in an RNN

- BPTT stands for backpropagation through time
- ŷ denotes the predicted output, while y denotes ground truth
- Subscript indicates time
- L here is the cost function
- u, v and w are the weight matrices
- Math is similar to regular backprop

Let's look a bit of it...



Let's look at some of the math behind it...



where the subscripts denote time..

y denotes ground truth, $\hat{y}$ denotes predicted value

The weights for the state function and output must remain constant over time for RNNs.

So, $h_t = g(Wh_{t-1} + Ux_t)$

$\hat{y}_t = g^*(Vh_t)$

Assume, $\hat{y}_t = Vh_t$

where W, U, and V are weight matrices and g & g^* are activation functions.

For Backpropagation, we need $\frac{dL}{dW}$, $\frac{dL}{dU}$ and $\frac{dL}{dV}$.

$$L = \sum_{t=0}^{T} L_t$$

For $\frac{dL}{dW}$, $\frac{dL}{dV}$, $\frac{dL}{dU}$, we have to find $\frac{dL_t}{dW}$, $\frac{dL_t}{dV}$, $\frac{dL_t}{dU}$

For now, let's assume there are only three time steps.

Consider the loss function to be squared error.

$$L_3 = (y_3 - \hat{y}_3)^2.$$

i) $$\frac{dL_3}{dW} = -2(y_3 - \hat{y}_3)\frac{\partial \hat{y}_3}{dW}$$

$$= -2(y_3 - \hat{y}_3)\frac{\partial \hat{y}_3}{\partial h_3}\cdot\frac{\partial h_3}{\partial W}.$$

$\hat{y}_3 = Vh_3$.

$$= -2(y_3 - \hat{y}_3)V\frac{\partial h_3}{\partial W}$$

$$= -2(y_3 - \hat{y}_3)V g'(Wh_2 + Ux_3)\left[h_2 + W\frac{dh_2}{dW}\right]$$

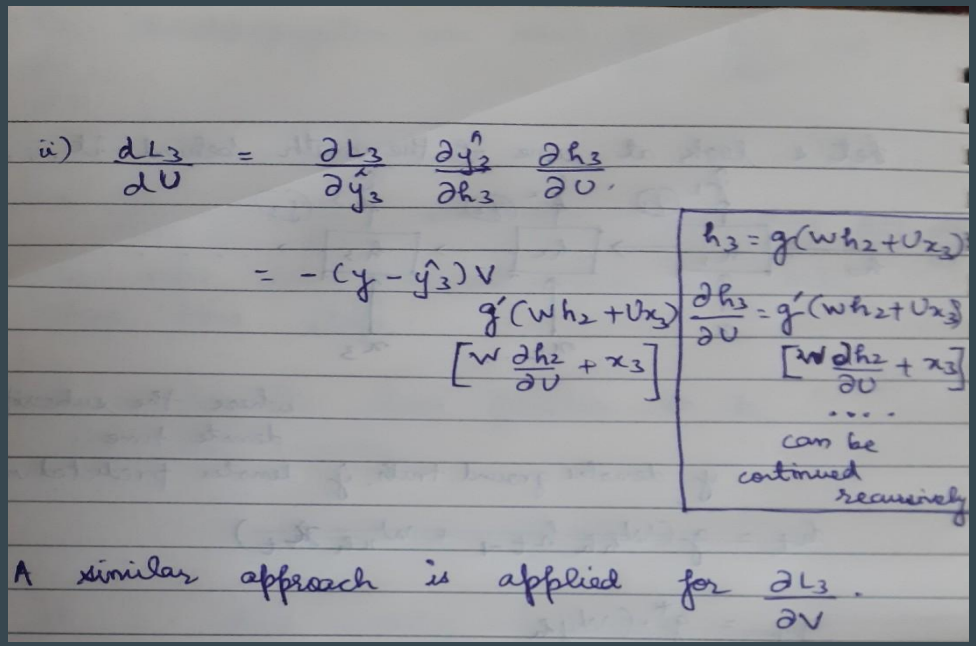
$h_3 = g(Wh_2 + Ux_3)$

$\frac{\partial h_3}{\partial W} = g'(Wh_2 + Ux_3)\left[h_2 + W\frac{dh_2}{dW}\right]$

.....

(can be worked out further)

And a bit more...



So, as we observe, for calculating dL3/ dW and dL3/dU, we need to backpropagate through time...

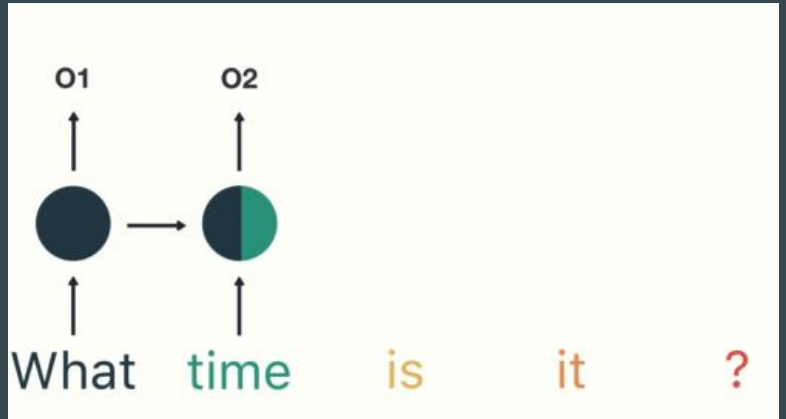
- As discussed earlier, the process of backpropagation aims to update the parameters of the neural network to minimise the total cost using the partial derivatives of the cost function with respect to its parameters.
- As in the case of a simple neural networks, we will try use chain rule to represent the partial derivative in terms of partial derivatives of other parameters across the neural network by traversing backwards, starting from the output layer.
- The key difference between feedforward neural network and RNN is that, in RNNs we sum up the gradients for W at each time step. In a traditional NN we don't share parameters across layers, so we don't need to sum anything.

$$\delta_2^{(3)} = \frac{\partial E_3}{\partial z_2} = \frac{\partial E_3}{\partial s_3}\frac{\partial s_3}{\partial s_2}\frac{\partial s_2}{\partial z_2} \text{ with } z_2 = Ux_2 + Ws_1.$$

Problems faced by RNNs

Vanishing Gradient (or Short Term Memory)

- As the RNN processes more steps, it has troubles retaining information from previous steps. The information from the word “what” and “time” is almost non-existent at the final time step.
- Because of vanishing gradients, the RNN doesn’t learn the long-range dependencies across time steps. That means that there is a possibility that the word “what” and “time” are not considered when trying to predict the user’s intention.



Problems faced by RNNs

- The gradients shrink when they are multiplied with the activation function derivatives (derivative of layers output wrt. to its input). For example the magnitude of tanh derivative used in RNNs is well below 1.0 in the whole range of the function and makes the gradients vanish quite fast.
- The earlier layers fail to do any learning as the internal weights are barely being adjusted due to extremely small gradients. And that's the vanishing gradient problem.

Exploding Gradients

- When the gradients are being back propagated in time all the way to the initial layer, the gradients coming from the deeper layers have to go through continuous matrix multiplications and as they approach the earlier layers, if they have large values (>1) they get larger and eventually blow up and crash the model.
- LSTMs can be used to avoid the problems of vanishing and exploding gradients.

So as you can see both the derivative terms are a chained multiplication. Now as the number of time steps increases, this gradient could become zero if the multiplicative term is less than 1 - Vanishing gradient, or could blow to infinity if it is more than one - Exploding Gradient

$$\frac{\partial L_t}{\partial U} = \frac{\partial L_t}{\partial y_t}\frac{\partial y_t}{\partial h_t}\left(\sum_{k=1}^{t}\left(\prod_{j=k+1}^{t}\frac{\partial h_j}{\partial h_{j-1}}\right)\frac{\partial^{+} h_k}{\partial U}\right)$$

$$\frac{\partial L_t}{\partial W} = \frac{\partial L_t}{\partial y_t}\frac{\partial y_t}{\partial h_t}\left(\sum_{k=1}^{t}\left(\prod_{j=k+1}^{t}\frac{\partial h_j}{\partial h_{j-1}}\right)\frac{\partial^{+} h_k}{\partial W}\right)$$

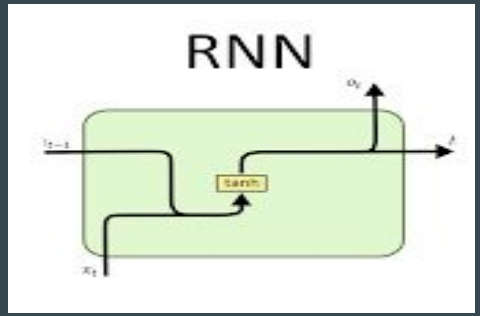
Long Short Term Memory Cells

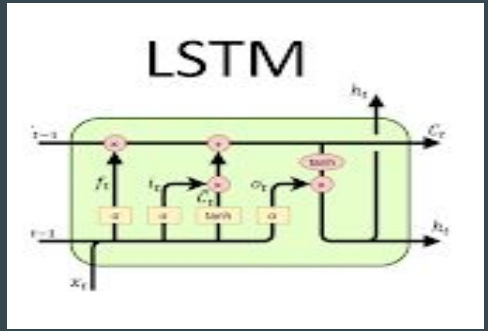
...

Or LSTM Cells

Why Do we use LSTM

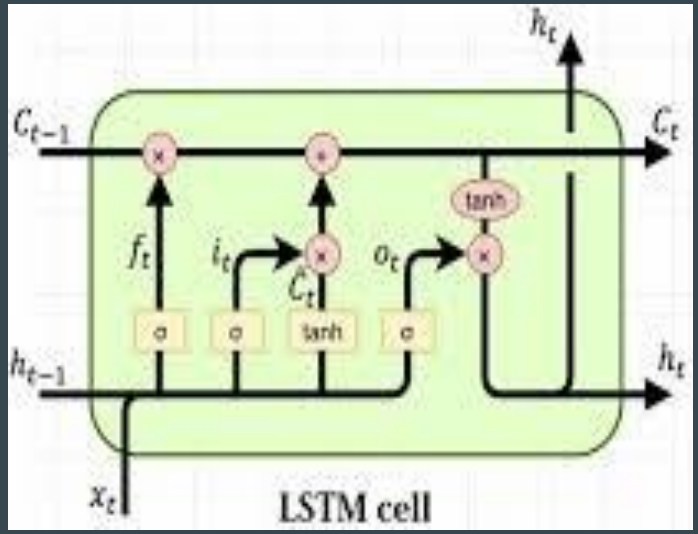
- The advantage of using LSTM is that it solves the problem of vanishing gradient.
- A RNN cell tries to encode all the past data it has seen in the hidden state where the LSTM cell has a separate cell state where only the essential details from the past are chosen and essential details of the current input is stored. The definition of “essential data” is learned by the cell. For accomplishing this task we create three gates which we will see next
- RNNs in total take two input - user input and the previous hidden state
- LSTMs in total take three inputs - user input, previous cell state and previous hidden state





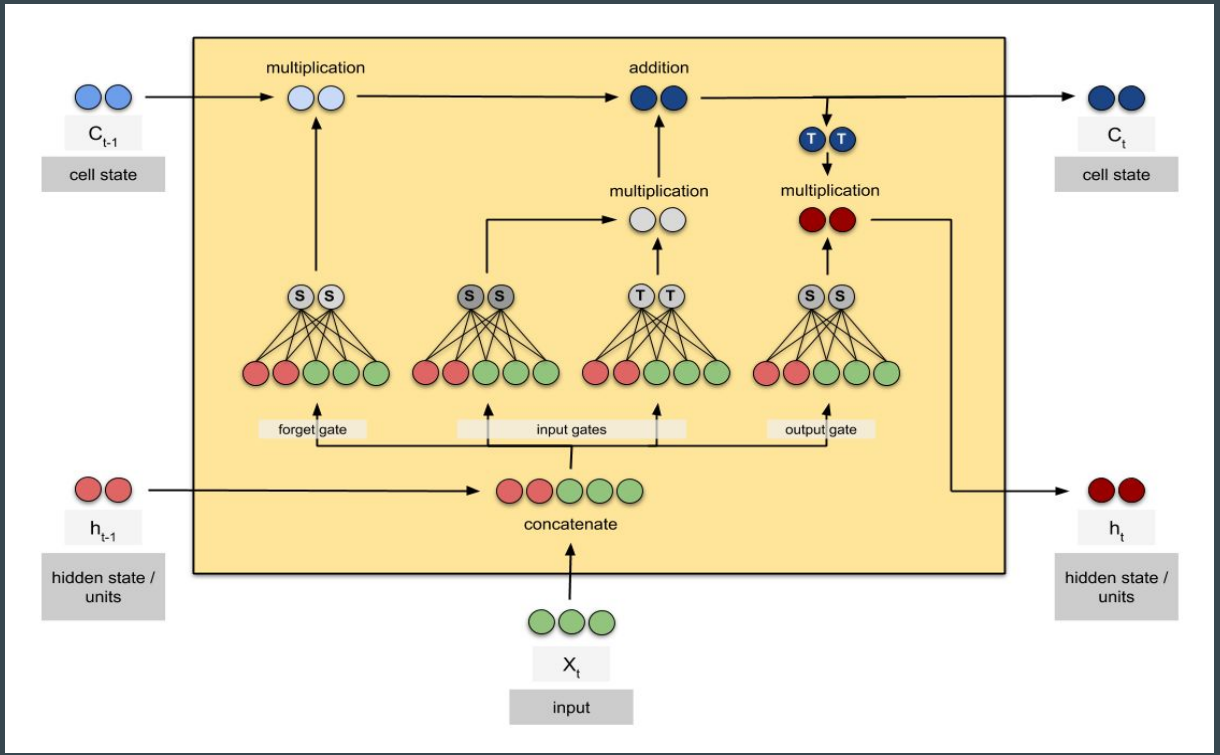
Before Moving on, Lets clear some things up

- Output Hidden state at time step t : h_t
- Output cell state at time step t : C_t
- Input by user at time step t : x_t
- Input hidden state at time step t : h_{t-1}
- Input cell state at time step t : C_{t-1}



LSTM Gates

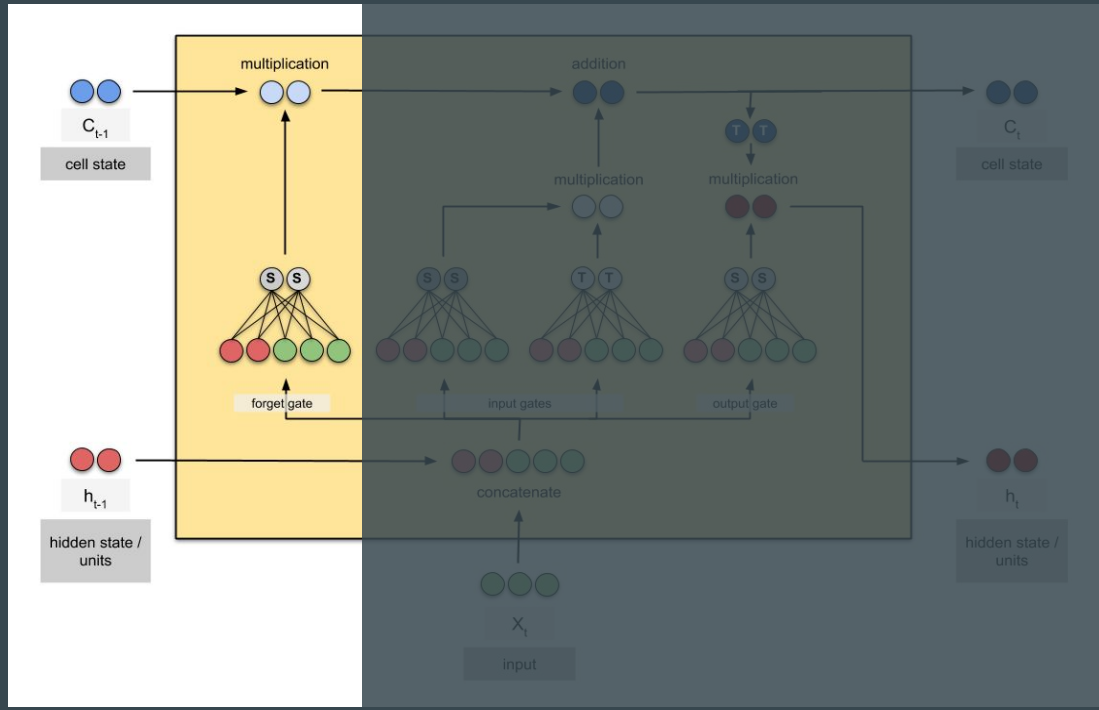
- Forget Gate
- Input Gate
- Output Gate



Forget Gate

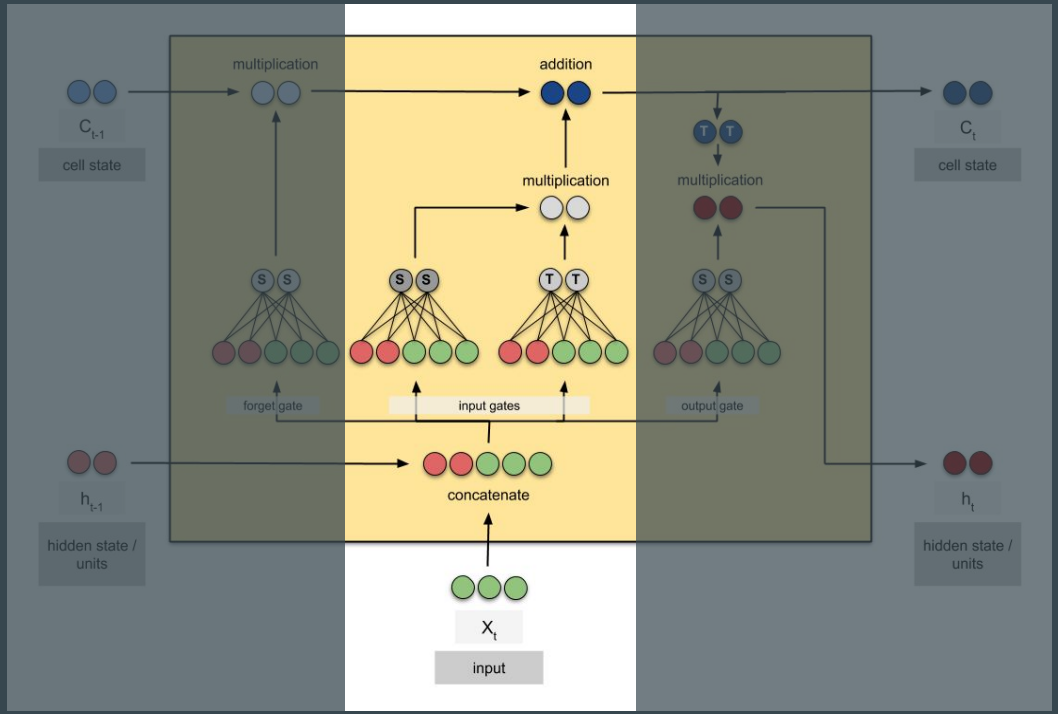
- As the name suggests it “forgets” some parts of the long term memory or the cell state

- The decision on what to forget made based on the current input and the output of the previous time step using an internal neural network



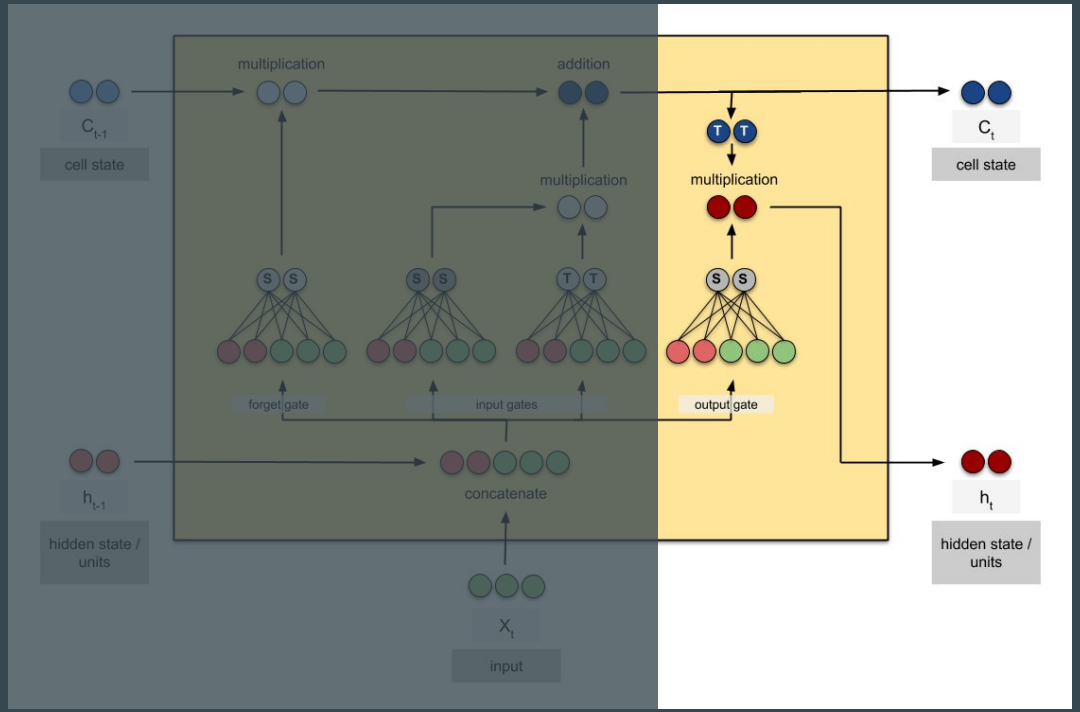
Input Gate

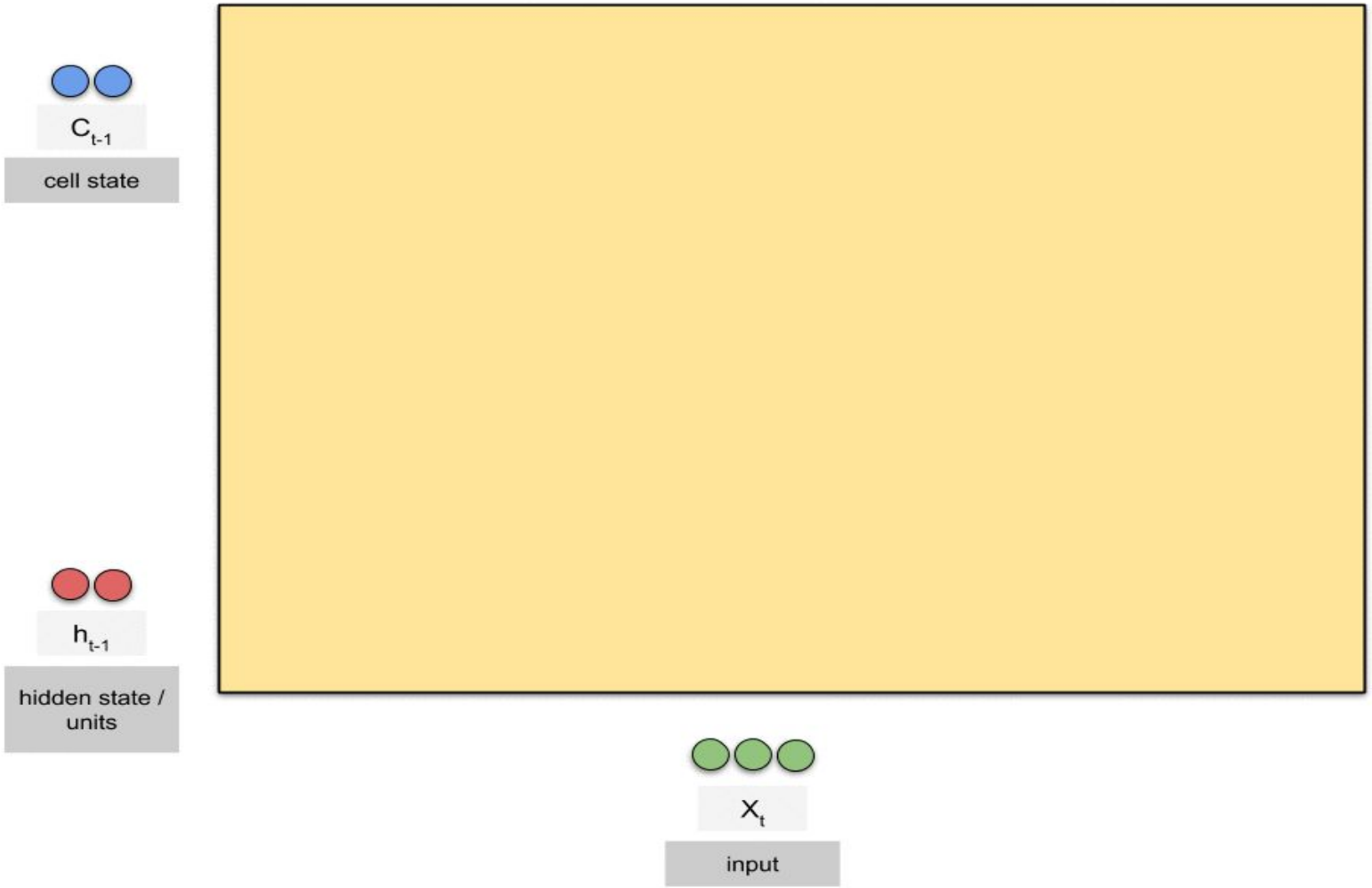
- It chooses to remember only some parts of the current input and the output of the previous time step
- This decision is again made based on the current input and the output of the previous time step using an internal neural network
- After the essential parts of the current input is extracted, this is added to the cell state after the cell state goes through the forget gate



Output Gate

- After saving all the relevant details in the cell state, this value is used to output the final hidden state of the current time step. The output gate decides how much of the information is to be used
- This decision is (as you would have guessed) is made using the current input and the output of the previous timestep using an internal neural network




C_{t-1}
cell state
h_{t-1}
hidden state / units
X_t
input

Let's look at the math

Here

- f_t: value of forget gate at time step t
- C-dash$_t$: value of activation at time step t
- I_t : value of input gate at time step t
- O_t : value of the output gate at time step t

$$
\begin{aligned}
f_t &= \sigma\,(X_t * U_f + H_{t-1} * W_f) \\
\bar{C}_t &= \tanh\,(X_t * U_c + H_{t-1} * W_c) \\
I_t &= \sigma\,(X_t * U_i + H_{t-1} * W_i) \\
O_t &= \sigma\,(X_t * U_o + H_{t-1} * W_o)
\end{aligned}
$$

$$
\begin{aligned}
C_t &= f_t * C_{t-1} + I_t * \bar{C}_t \\
H_t &= O_t * \tanh\,(C_t)
\end{aligned}
$$

Some more Applications of LSTMs

- Short term Traffic Forecasting - How many people will pass through a junction at a given time? Will they all be in a bus? Or will they all have their own cars? Or is it some combination of both and if it is then how exactly is it distributed?

- Human Action Recognition - Recognising what a human is doing by feeding it a sequence of frames (fancy name for a video :P). An extended application of this could be to convert sign language to speech.

That’s all for now!!!
See you guys in the next session!!!

...

For any doubts or queries feel free to reach out to any of the club coordinators