

DS JULY 2022 Batch
Module 23 : Deep Learning part 4 (RNN)

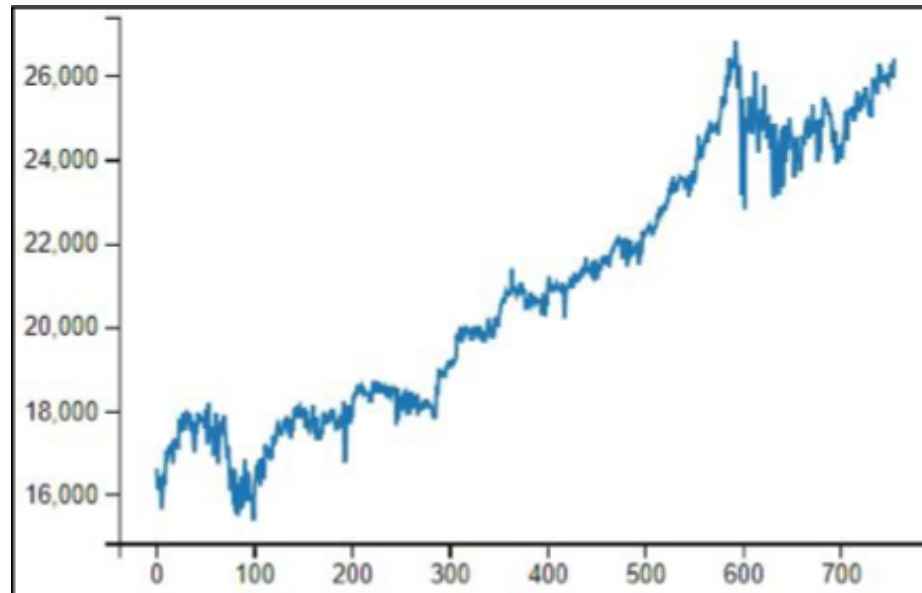
Topics

- Recurrent Neural Network (RNN)
- Back Propagation through time
- Different types of RNN: LSTM, GRU
- Bidirectional RNN
- Seq 2 Seq model (Encoder Decoder)
- Text generation and classification using Deep Learning

RNN

Sequential Data

- The dataset is said to be sequential when the data points are dependent on other data points within a dataset.



Sequential Data

- Consider you have a sequential data that contains temperature and humidity values for everyday.

```
array([[0.23, 0.72],  
       [0.27, 0.54],  
       [0.33, 0.2 ],  
       [0.03, 0.72],  
       [0.27, 0.24],  
       [0.33, 0.29]])
```

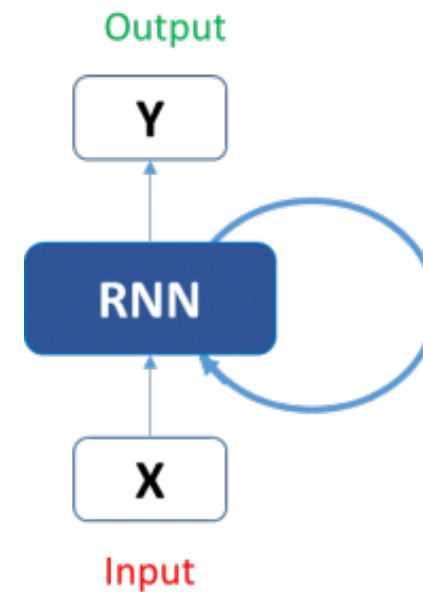



- The data then flows to the hidden layers, where the weights and biases are applied.
- A traditional neural network assumes that the data is non-sequential and each data point is independent of the others.
- The network does not remember what it gives as an output. It just accepts the next data point.

Sequential Data

- In the weather data, there is a strong correlation between the weather from one day and the weather in subsequent days. The former has influence over the latter.
- If it was sunny on one day in the middle of summer, it's easy to presume that it'll also be sunny on the following day.

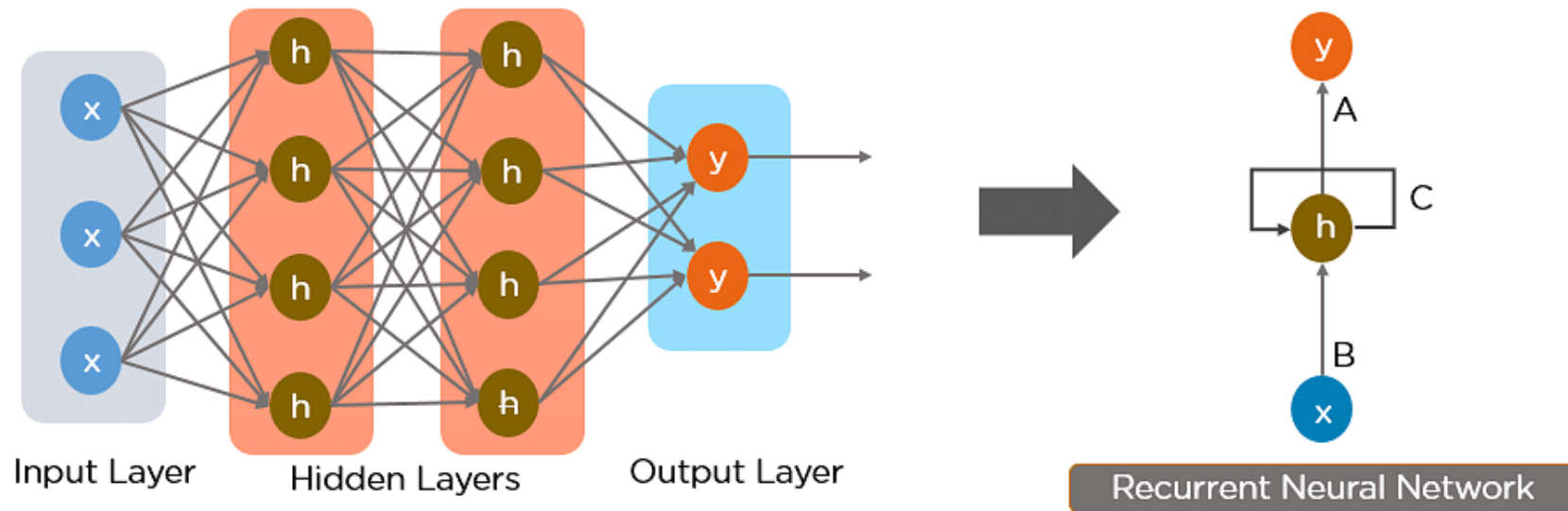
```
array([[0.23, 0.72],  
      [0.27, 0.54],  
      [0.33, 0.2 ],  
      [0.03, 0.72],  
      [0.27, 0.24],  
      [0.33, 0.29]])
```



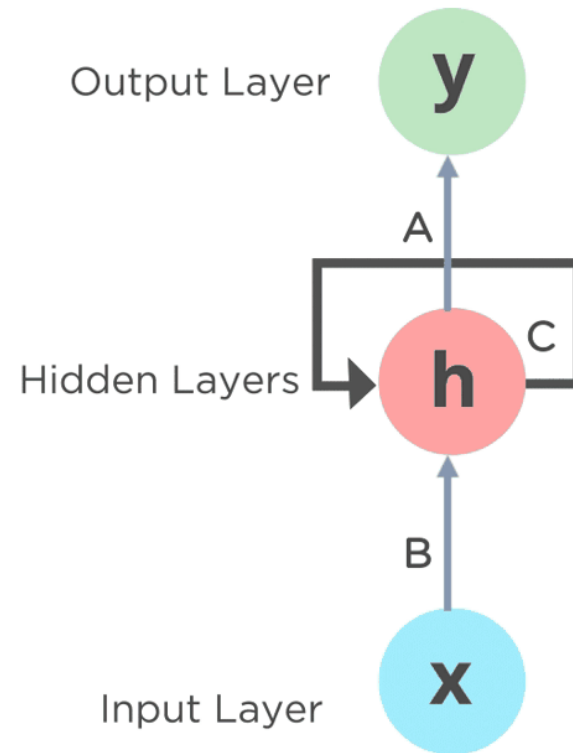
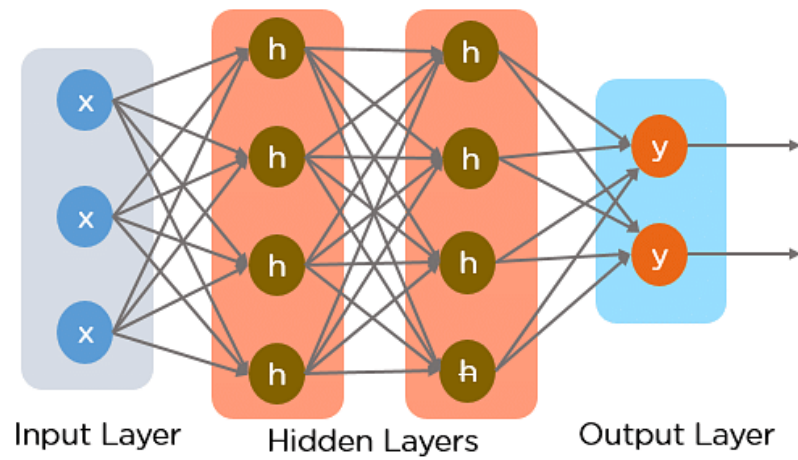
- RNN has a mechanism that can handle a sequential dataset.

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.
- Below is how you can convert a Feed-Forward Neural Network into a Recurrent Neural Network:



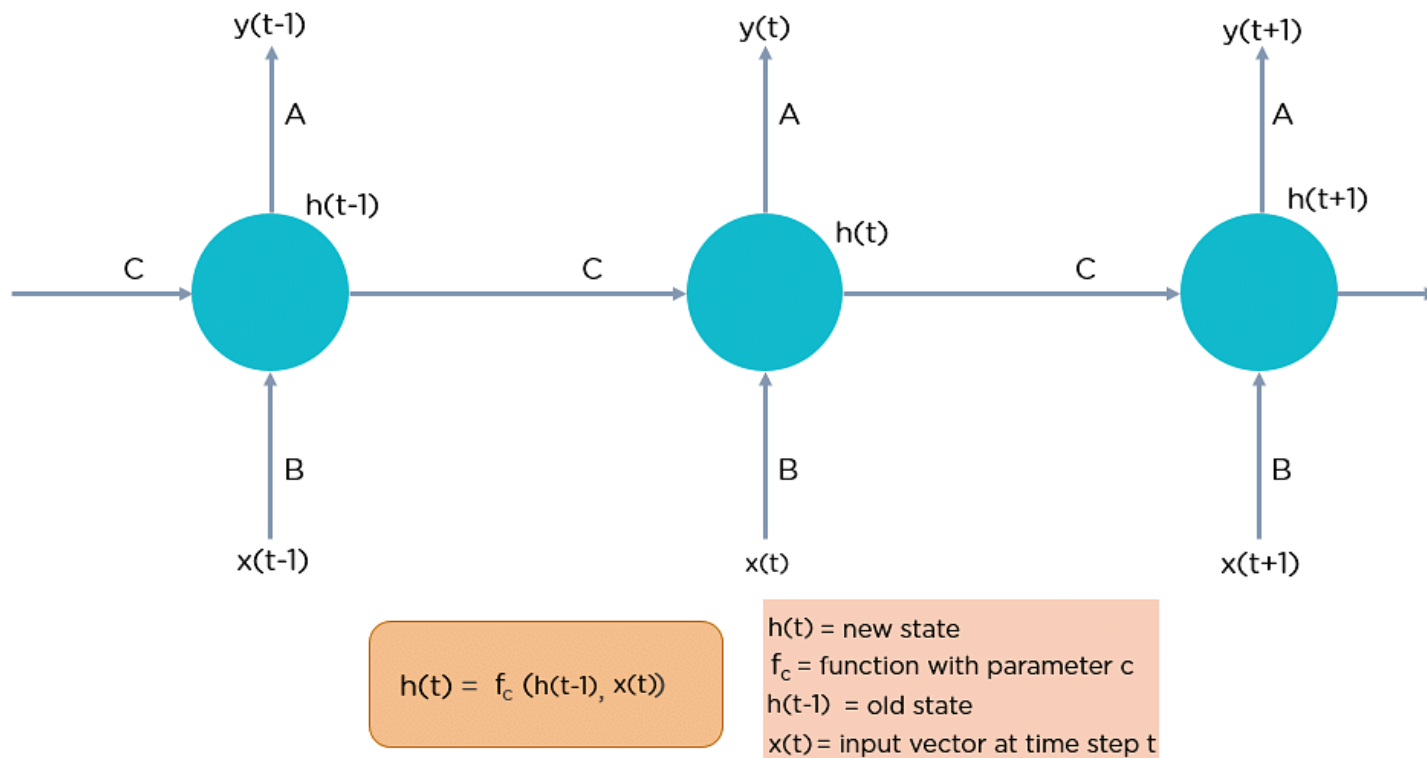
RNN



A, B and C are the parameters

RNN

- Here, “x” is the input layer, “h” is the hidden layer, and “y” is the output layer. A, B, and 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.



Why RNN

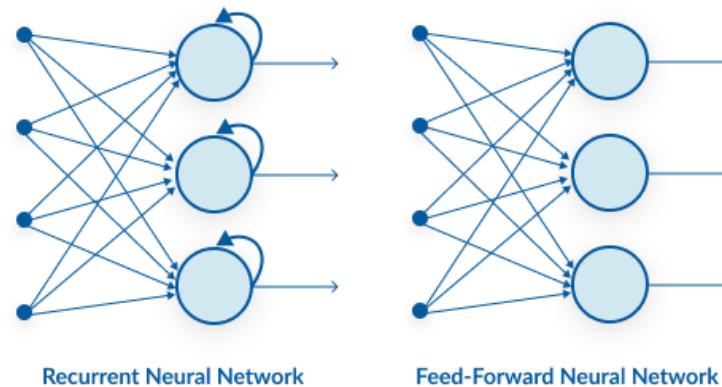
- RNN were created because there were a few issues in the feed-forward neural network:
 - Cannot handle sequential data
 - Considers only the current input
 - Cannot memorize previous inputs
- The solution to these issues is the RNN. An RNN can handle sequential data, accepting the current input data, and previously received inputs. RNNs can memorize previous inputs due to their internal memory.

How RNN works

- In Recurrent Neural networks, 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 and weights and biases. If you have a neural network where the various parameters of different hidden layers are not affected by the previous layer, ie: the neural network does not have memory, then you can use a recurrent neural network.
- The Recurrent Neural Network will standardize the different activation functions and weights and 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.

RNN Vs FFNN

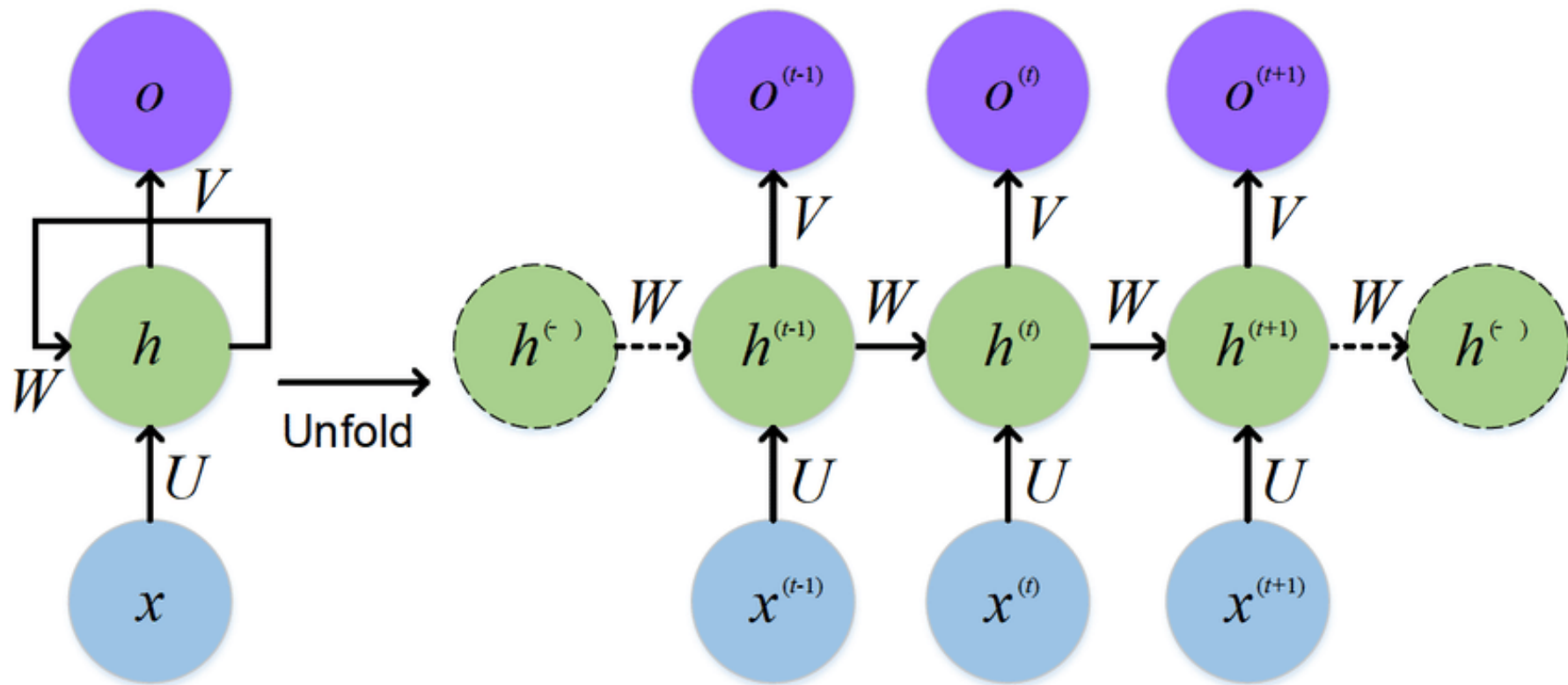
- A feed-forward neural network has only one route of information flow: from the input layer to the output layer, passing through the hidden layers. The data flows across the network in a straight route, never going through the same node twice.
- Feed-forward neural networks are poor predictions of what will happen next because they have no memory of the information they receive. Because it simply analyses the current input, a feed-forward network has no idea of temporal order. Apart from its training, it has no memory of what transpired in the past.



- The information is in an RNN cycle via a loop. Before making a judgment, it evaluates the current input as well as what it has learned from past inputs. A recurrent neural network, on the other hand, may recall due to internal memory. It produces output, copies it, and then returns it to the network.

BACKPROPAGATION THROUGH TIME

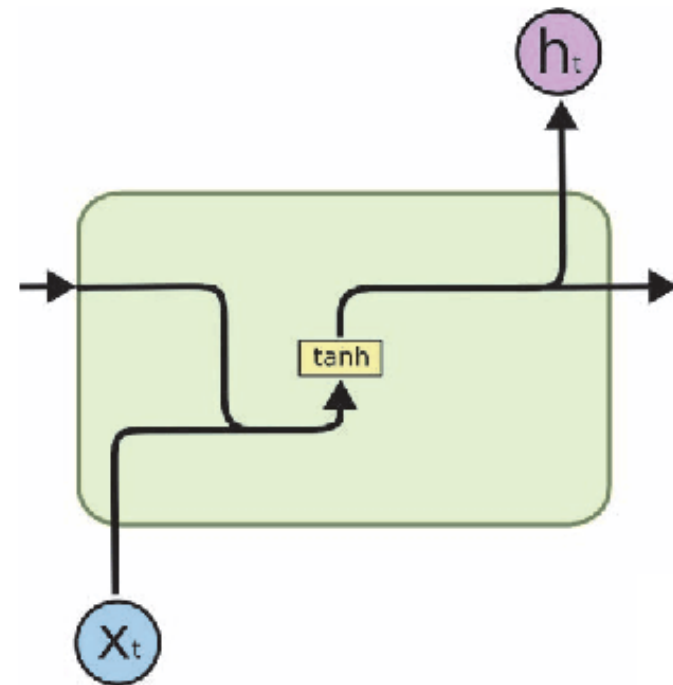
Back Propagation through Time RNN



LSTM
GRU

RNN : Types

- Let's take a look at a single unit of RNN architecture. Where it takes input from the previous step and current state X_t and incorporated with Tanh as an activation function, here we can explicitly change the activation function.
- Sometimes we only need to look at recent information to perform a present task. But this is not the case we face all the time.
- When a standard RNN network is exposed to long sequences or phrases it tends to lose the information because it can not store the long sequences and as the methodology is concerned it focuses only on the latest information available at the node.
- This problem is commonly referred to as Vanishing gradients.

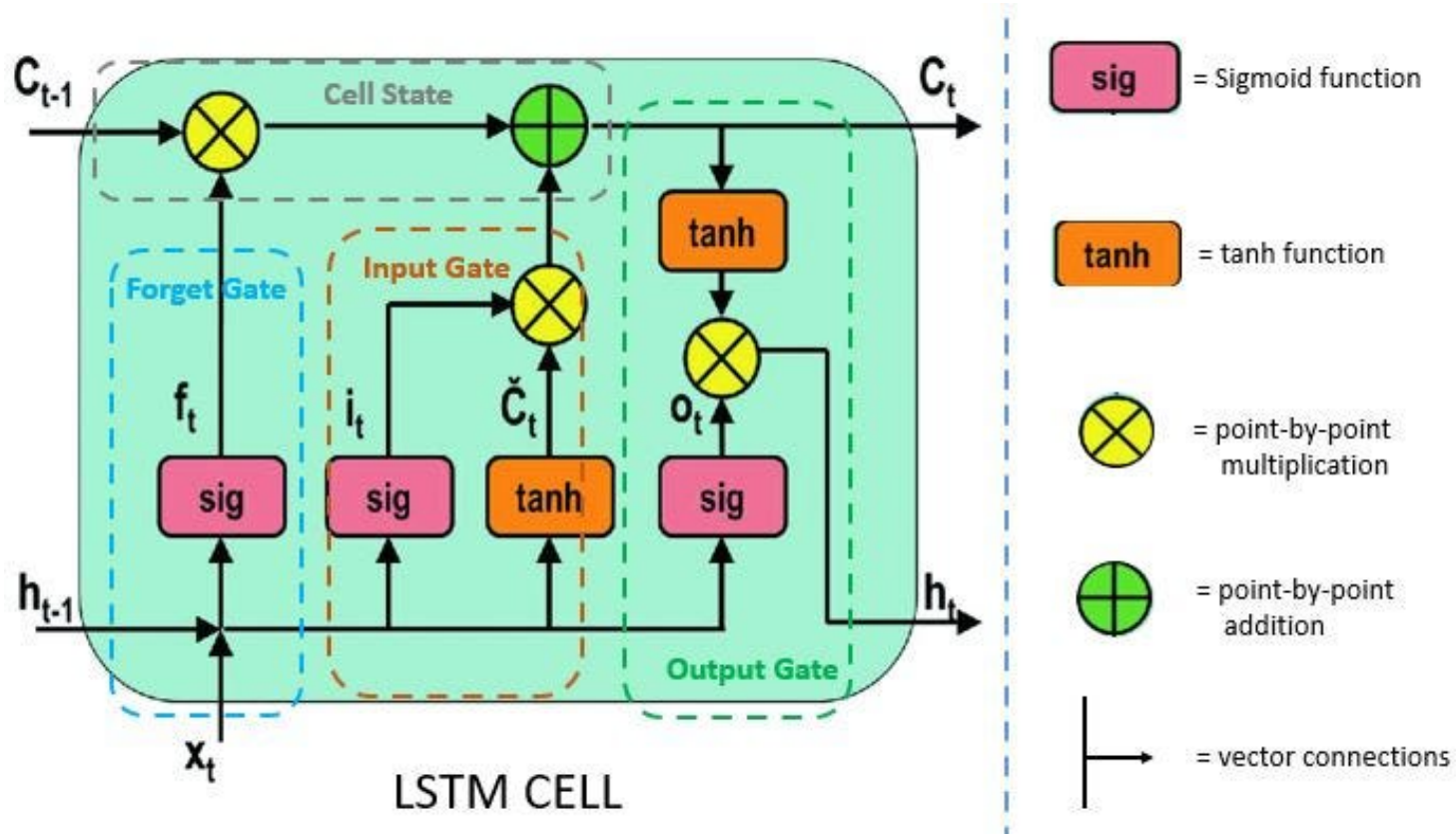


RNN : Types

- In RNN to train networks, we backpropagate through time and at each time step or loop operation gradient is being calculated and the gradient is used to update the weights in the networks.
- Now if the effect of the previous sequence on the layer is small then the relative gradient is calculated small.
- Then if the gradient of the previous layer is smaller then this makes weights to be assigned to the context smaller and this effect is observed when we deal with longer sequences.
- Due to this network does not learn the effect of earlier inputs and thus causing the short term memory problem.
- To overcome this problem specialized versions of RNN are created like LSTM, GRU.

RNN : LSTM

- Long Short Term Memory in short LSTM is a special kind of RNN capable of learning long term sequences. It is explicitly designed to avoid long term dependency problems. Remembering the long sequences for a long period of time is its way of working.



RNN : LSTM

- The popularity of LSTM is due to the Getting mechanism involved with each LSTM cell.
- In a normal RNN cell, the input at the time stamp and hidden state from the previous time step is passed through the activation layer to obtain a new state.
- Whereas in LSTM the process is slightly complex, as you can see in the above architecture at each time it takes input from three different states like the current input state, the short term memory from the previous cell and lastly the long term memory.
- These cells use the gates to regulate the information to be kept or discarded at loop operation before passing on the long term and short term information to the next cell.
- We can imagine these gates as Filters that remove unwanted selected and irrelevant information.
- There are a total of three gates that LSTM uses as Input Gate, Forget Gate, and Output Gate.

RNN : LSTM

- **Input Gate**

- The input gate decides what information will be stored in long term memory. It only works with the information from the current input and short term memory from the previous step. At this gate, it filters out the information from variables that are not useful.

- **Forget Gate**

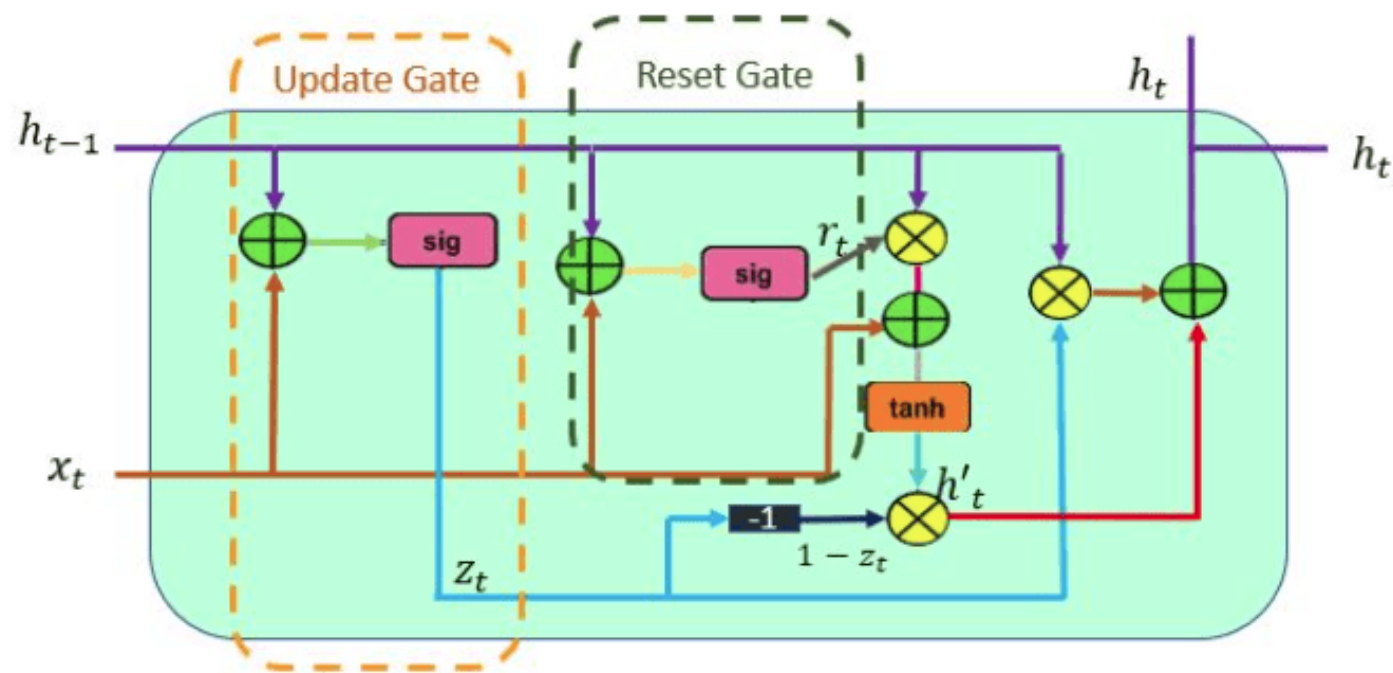
- The forget decides which information from long term memory be kept or discarded and this is done by multiplying the incoming long term memory by a forget vector generated by the current input and incoming short memory.

- **Output Gate**

- The output gate will take the current input, the previous short term memory and newly computed long term memory to produce new short term memory which will be passed on to the cell in the next time step. The output of the current time step can also be drawn from this hidden state.

RNN : GRU

- The workflow of the Gated Recurrent Unit, in short GRU, is the same as the RNN but the difference is in the operation and gates associated with each GRU unit. To solve the problem faced by standard RNN, GRU incorporates the two gate operating mechanisms called Update gate and Reset gate.



RNN : GRU

- **Update gate**
 - The update gate is responsible for determining the amount of previous information that needs to pass along the next state. This is really powerful because the model can decide to copy all the information from the past and eliminate the risk of vanishing gradient.
- **Reset gate**
 - The reset gate is used from the model to decide how much of the past information is needed to neglect; in short, it decides whether the previous cell state is important or not.
- First, the reset gate comes into action it stores relevant information from the past time step into new memory content. Then it multiplies the input vector and hidden state with their weights. Next, it calculates element-wise multiplication between the reset gate and previously hidden state multiple. After summing up the above steps the non-linear activation function is applied and the next sequence is generated.

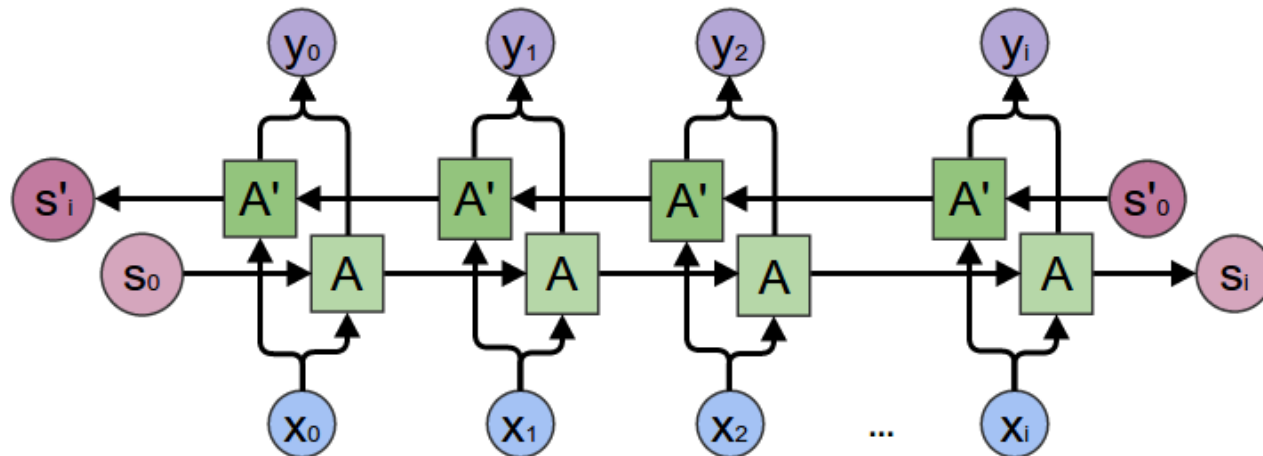
RNN : LSTM :GRU

Feature	RNN	LSTM	GRU
Structure	Simple	More complex	Simpler than LSTM
Training	Can be difficult	Can be more difficult	Easier than LSTM
Performance	Good for simple tasks	Good for complex tasks	Can be intermediate between simple and complex tasks
Hidden state	Single	Multiple (memory cell)	Single
Gates	None	Input, output, forget	Update, reset
Ability to retain long-term dependencies	Limited	Strong	Intermediate between RNNs and LSTMs

Bidirectional RNN

BRNN

- Bi-directional recurrent neural networks (Bi-RNNs) are artificial neural networks that process input data in both the forward and backward directions.
- They are often used in natural language processing tasks, such as language translation, text classification, and named entity recognition. They can capture contextual dependencies in the input data by considering past and future contexts.
- In a Bi-RNN, the input data is passed through two separate RNNs: one processes the data in the forward direction, while the other processes it in the reverse direction. The outputs of these two RNNs are then combined in some way to produce the final output.



BRNN

- Consider an example where we could use the recurrent network to predict the masked word in a sentence.
 - Apple is my favorite _____.
 - Apple is my favourite _____, and I work there.
 - Apple is my favorite _____, and I am going to buy one.
- In the first sentence, the answer could be fruit, company, or phone. But in the second and third sentences, it can not be a fruit.
- A Recurrent Neural Network that can only process the inputs from left to right might not be able to accurately predict the right answer for sentences discussed above.
- To perform well on natural language tasks, the model must be able to process the sequence in both directions.

BRNN

- During the **forward pass** of the RNN, the forward RNN processes the input sequence in the usual way by taking the input at each time step and using it to update the hidden state.
- The updated hidden state is then used to predict the output at that time step.
- Backpropagation through time (**BPTT**) is a widely used algorithm for training recurrent neural networks (RNNs).
- It is a variant of the backpropagation algorithm specifically designed to handle the temporal nature of RNNs, where the output at each time step depends on the inputs and outputs at previous time steps.
- In the case of a bidirectional RNN, BPTT involves two separate Backpropagation passes: one for the forward RNN and one for the backward RNN.
- During the forward pass, the forward RNN processes the input sequence in the usual way and makes predictions for the output sequence.
- These predictions are then compared to the target output sequence, and the error is backpropagated through the network to update the weights of the forward RNN.

BRNN

- During the **backward pass**, the backward RNN processes the input sequence in reverse order and makes predictions for the output sequence.
- These predictions are then compared to the target output sequence in reverse order, and the error is backpropagated through the network to update the weights of the backward RNN.
- Once both passes are complete, the weights of the forward and backward RNNs are updated based on the errors computed during the forward and backward passes, respectively.
- This process is repeated for multiple iterations until the model converges and the predictions of the bidirectional RNN are accurate.
- This allows the bidirectional RNN to consider information from past and future time steps when making predictions, which can significantly improve the model's accuracy.

BRNN: Merging modes

- There are several ways in which the outputs of the forward and backward RNNs can be merged, depending on the specific needs of the model and the task it is being used for. Some common merge modes include:
 - Concatenation: In this mode, the outputs of the forward and backward RNNs are simply concatenated together, resulting in a single output tensor that is twice as long as the original input.
 - Sum: In this mode, the outputs of the forward and backward RNNs are added together element-wise, resulting in a single output tensor that has the same shape as the original input.
 - Average: In this mode, the outputs of the forward and backward RNNs are averaged element-wise, resulting in a single output tensor that has the same shape as the original input.
 - Maximum: In this mode, the maximum value of the forward and backward outputs is taken at each time step, resulting in a single output tensor that has the same shape as the original input.

SEQ2SEQ MODEL

Seq2Seq Model

- A Seq2Seq model is a model that takes a sequence of items (words, letters, time series, etc) and outputs another sequence of items.
- Sequence to Sequence (often abbreviated to seq2seq) models is a special class of Recurrent Neural Network architectures that we typically use (but not restricted) to solve complex Language problems like Machine Translation, Question Answering, creating Chatbots, Text Summarization, etc.

Machine Language Translation

*Les modèles de séquence
sont super puissants*

Sequence Model

*Sequence models are super
powerful*

Text Summarization

*A strong analyst have 6
main characteristics. One
should master all 6 to be
successful in the industry :*
1.
2.

Sequence Model

*6 characteristics of
successful analyst*

Chatbot

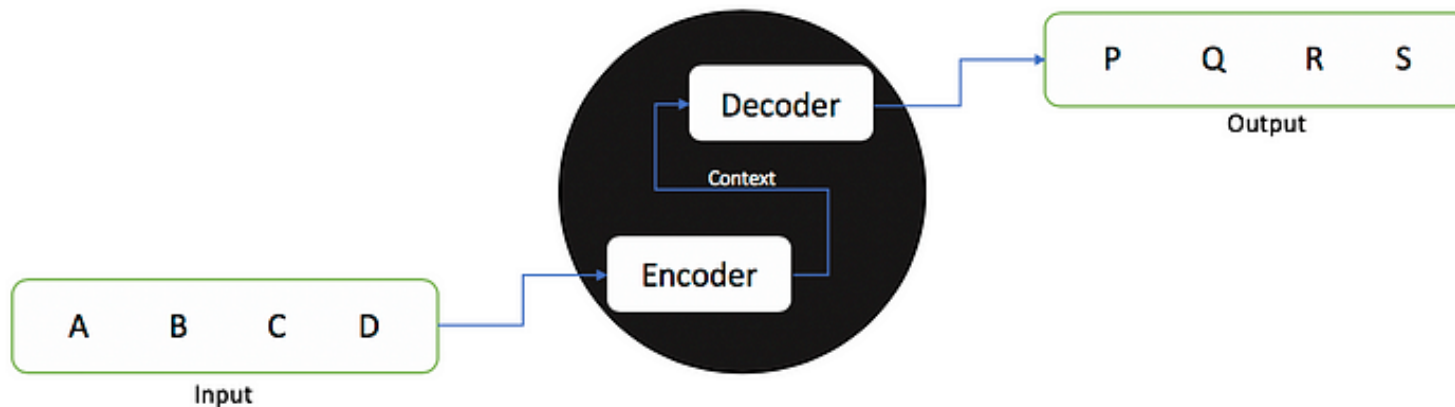
How are you doing today?

Sequence Model

*I am doing well. Thank you.
How are you doing today?*

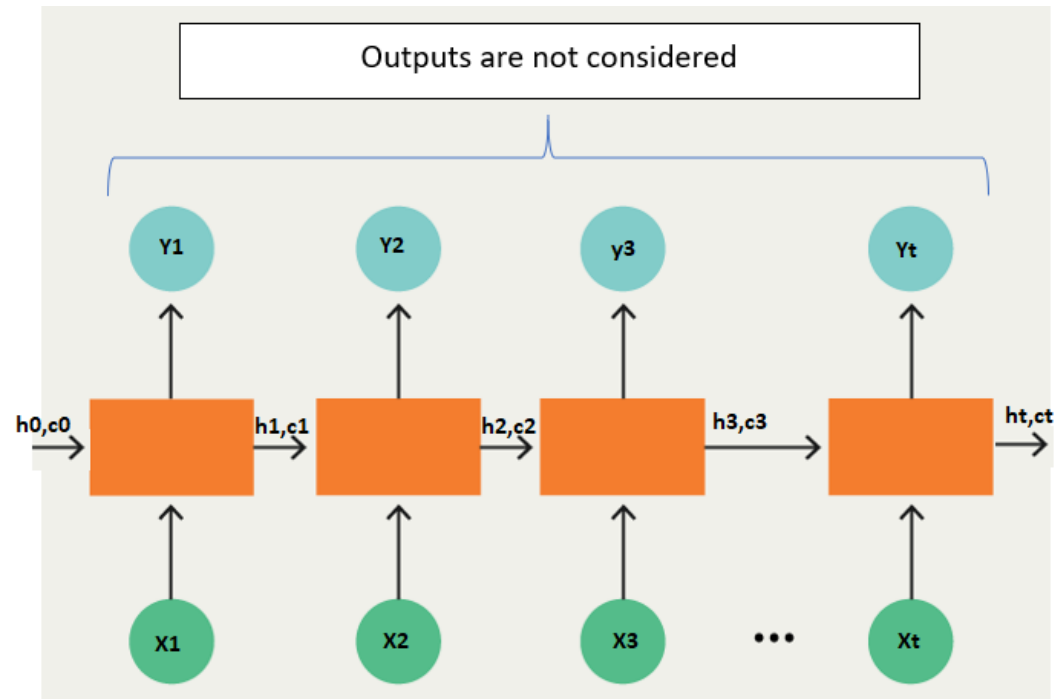
Seq2Seq Model

- The model is composed of an encoder and a decoder. The encoder captures the context of the input sequence in the form of a hidden state vector and sends it to the decoder, which then produces the output sequence.
- Since the task is sequence based, both the encoder and decoder tend to use some form of RNNs, LSTMs, GRUs, etc.



Seq2Seq Model : Encoder

- Both encoder and the decoder are LSTM models (or sometimes GRU models)
- Encoder reads the input sequence and summarizes the information in something called the internal state vectors or context vector (in case of LSTM these are called the hidden state and cell state vectors).
- We discard the outputs of the encoder and only preserve the internal states.
- This context vector aims to encapsulate the information for all input elements in order to help the decoder make accurate predictions.



$$h_t = f(W^{(hh)}h_{t-1} + W^{(hx)}x_t)$$

Seq2Seq Model : Encoder

- The LSTM reads the data, one sequence after the other. Thus if the input is a sequence of length 't', we say that LSTM reads it in 't' time steps.
- 1. X_i = Input sequence at time step i.
- 2. h_i and c_i = LSTM maintains two states ('h' for hidden state and 'c' for cell state) at each time step. Combined together these are internal state of the LSTM at time step i.
- 3. Y_i = Output sequence at time step i. Y_i is actually a probability distribution over the entire vocabulary which is generated by using a softmax activation. Thus each Y_i is a vector of size "vocab_size" representing a probability distribution.

Seq2Seq Model : Decoder

- The decoder is an LSTM whose initial states are initialized to the final states of the Encoder LSTM, i.e. the context vector of the encoder's final cell is input to the first cell of the decoder network. Using these initial states, the decoder starts generating the output sequence, and these outputs are also taken into consideration for future outputs.
- A stack of several LSTM units where each predicts an output y_t at a time step t .
- Each recurrent unit accepts a hidden state from the previous unit and produces an output as well as its own hidden state.
- Any hidden state h_t is computed using the formula:

$$h_t = f(W^{(hh)}h_{t-1})$$

- The output y_t at time step t is computed using the formula:

$$y_t = \text{softmax}(W^S h_t)$$

Seq2Seq Model : Decoder

- We calculate the outputs using the hidden state at the current time step together with the respective weight $W(S)$. Softmax is used to create a probability vector which will help us determine the final output (e.g. word in the question-answering problem).

