Recurrent Neural Network (RNN)

Why do we need RNN?

You might be wondering by now, we have networks like Convolutional ones which perform very well. Why do we need another type of network? There is a very specific use case where RNNs are required. In order to explain RNNs you need to first understand something called a ***sequence***. Let's talk about sequences first.

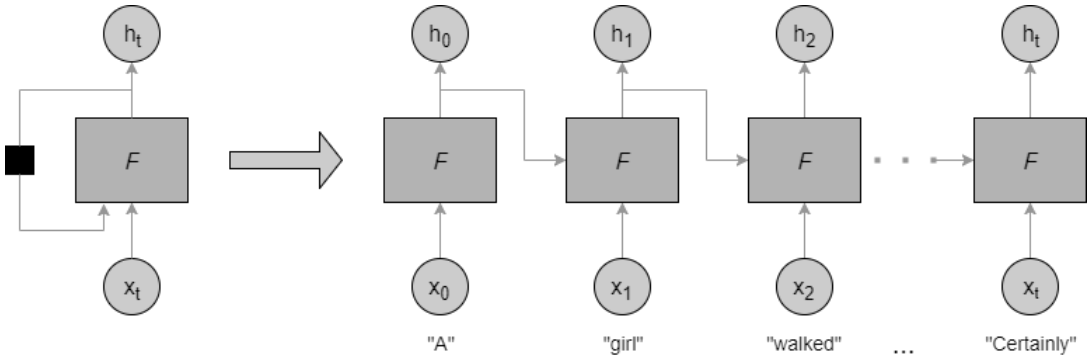
Sequence is a stream of data (finite or infinite) which are interdependent. Examples would be time-series data, informative pieces of strings, conversations, etc. In a conversation a sentence means something but the entire flow of the conversation mostly means something completely different. Also in a time-series data like stock market data, a single tick data means the current price, but a full days data will show movement and allow us to take decision whether to buy or sell.

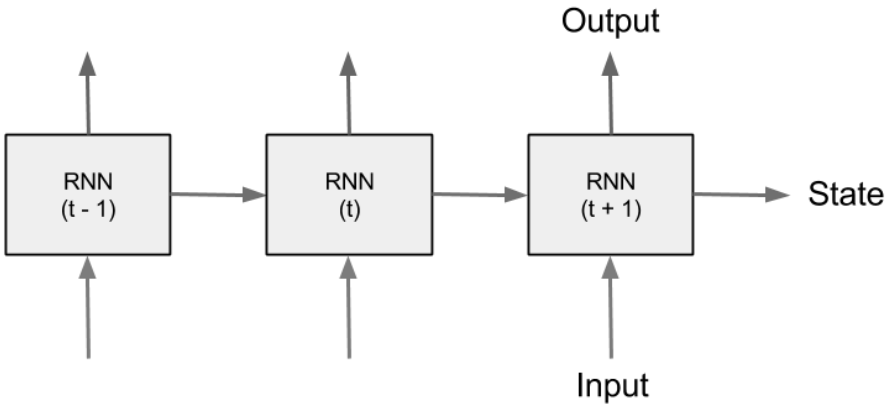
CNNs generally don't perform well when the input data is interdependent in a sequential pattern. CNNs don't have any sort of correlation between previous input to the next input. So all the outputs are self-dependent. CNN takes an input and outputs based on the trained model. If you run 100 different inputs none of them would be biased by the previous output. But imagine a scenario like sentence generation or text translation. All the words generated are dependent on the words generated before (in certain cases, it's dependent on words coming after as well, but we will discuss that later). So you need to have some bias based on your previous output. This is where RNNs shine. RNNs have in them a sense some memory about what happened earlier in the sequence of data. This helps the system to gain context. Theoretically RNNs have infinite memory, meaning they have the capability to look back indefinitely.

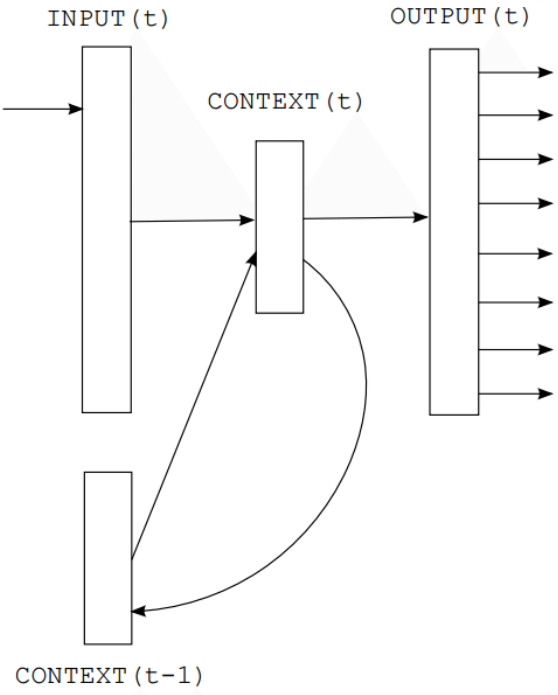
Differences between CNN and RNN:

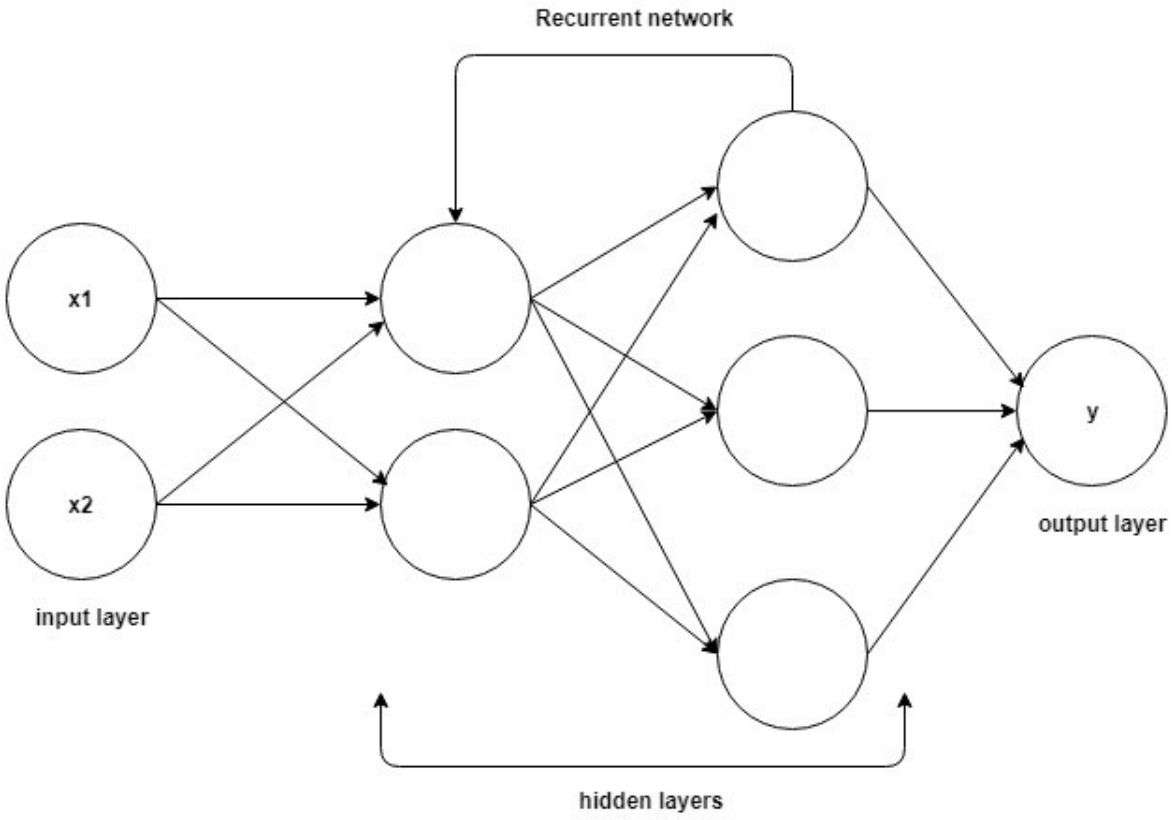
- CNN is a feed-forward neural network that is generally used for image recognition and object classification. While RNN works on the principle of saving the output of a layer and feeding this back to the input in order to predict the output of the layer.
- CNN considers only the current input while RNN considers the current input and also the previously received inputs. It can memorize previous inputs due to its internal memory.
- CNN has 4 layers namely: Convolution layer, ReLU layer, Pooling and Fully Connected Layer. Every layer has its own functionality and performs feature extractions and finds out hidden patterns. There are 4 types of RNN namely: One to One, One to Many, Many to One and Many to Many.
- RNN can handle sequential data while CNN cannot.

Sample Diagram to understand RNN:

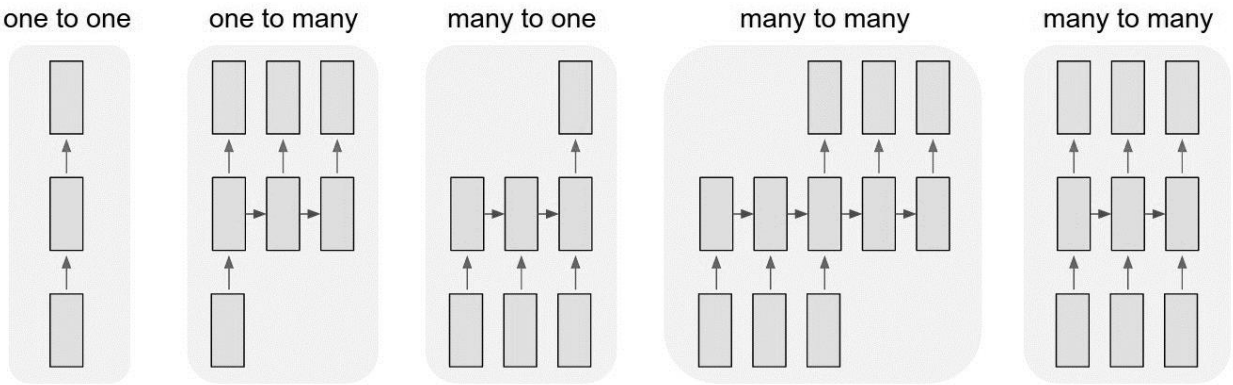








Different Types of RNN Structure:



Where to use an RNN? / Applications of RNN:

RNNs can be used in a lot of different places. The following are a few examples where a lot of RNNs are used.

1. **Language Modelling and Generating Text:**
   Given a sequence of the word, here we try to predict the likelihood of the next word. This is useful for translation since the most likely sentence would be the one that is correct.

2. **Machine Translation:**
   Translating text from one language to another uses one or the other form of RNN. All practical day systems use some advanced version of an RNN.

3. **Speech Recognition:**
   Predicting phonetic segments based on input sound waves, thus formulating a word.

4. **Generating Image Descriptions:**
   A very big use case is to understand what is happening inside an image, thus we have a good description. This works in a combination of CNN and RNN. CNN does the segmentation and RNN then used the segmented data to recreate the description. It's rudimentary but the possibilities are limitless.

5. **Video Tagging:**
   This can be used for video search where we do image description of a video frame by frame.

Limitations of RNN:

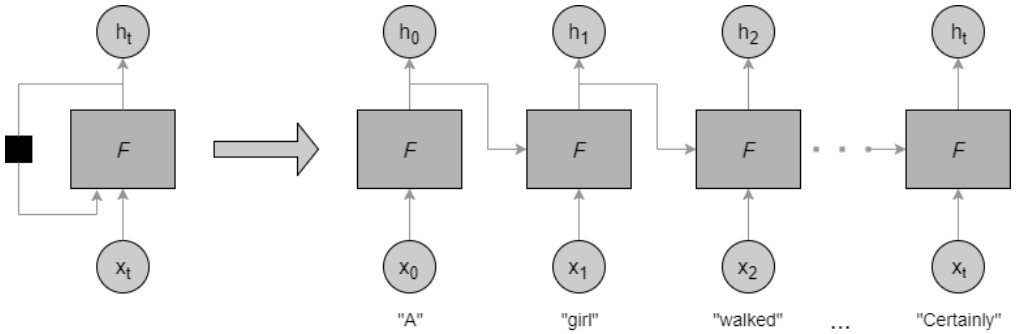
RNN is supposed to carry the information up to time. However, it is quite challenging to propagate all this information when the time step is too long. When a network has too many deep layers, it becomes untrainable. This problem is called: **vanishing gradient problem**. The neural network updates the weight using the gradient descent algorithm. The gradients grow smaller when the network progress down to lower layers.

When the gradients stay constant meaning there is no space for improvement. The model learns from a change in the gradient; this change affects the network's output. However, if the difference in the gradient is too small (i.e., the weights change a little), the network can't learn anything and so the output. Therefore, a network facing a vanishing gradient problem cannot converge toward a good solution.

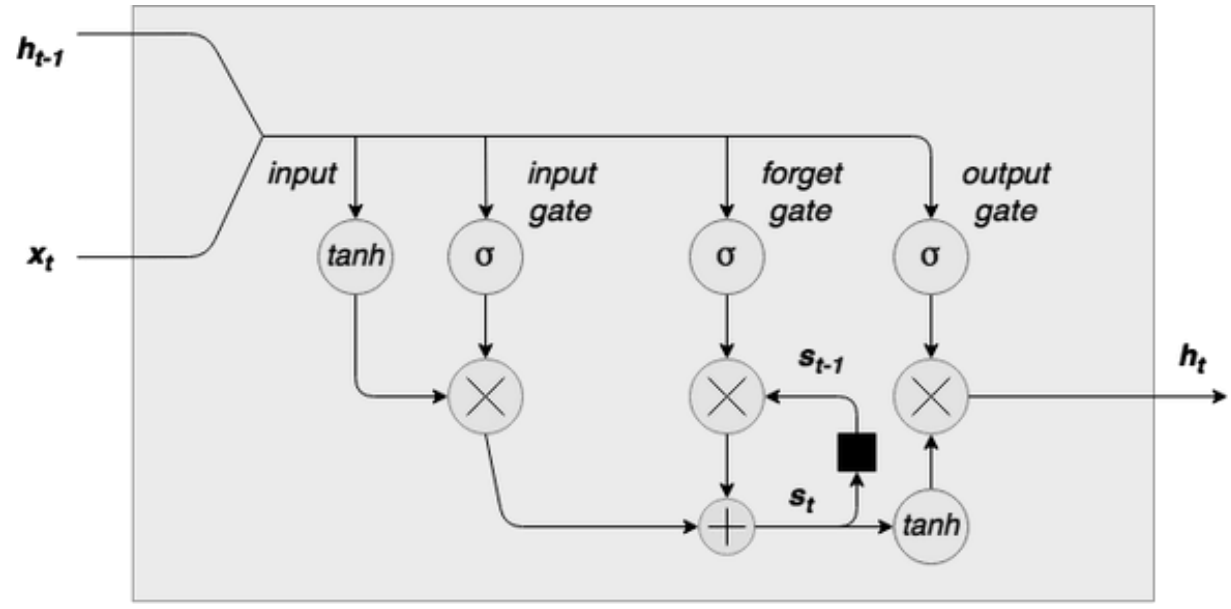
How does LSTM Overcome the Limitations of RNN?

RNN is supposed to carry the information up to time. However, it is quite challenging to propagate all this information when the time step is too long. When a network has too many deep layers, it becomes untrainable. This problem is called: **vanishing gradient problem**. The neural network updates the weight using the gradient descent algorithm. The gradients grow smaller when the network progress down to lower layers.

When the gradients stay constant meaning there is no space for improvement. The model learns from a change in the gradient; this change affects the network's output. However, if the difference in the gradient is too small (i.e., the weights change a little), the network can't learn anything and so the output. Therefore, a network facing a vanishing gradient problem cannot converge toward a good solution.



To overcome the potential issue of vanishing gradient faced by RNN, three researchers, Hochreiter, Schmidhuber and Bengio improved the RNN with an architecture called Long Short-Term Memory (LSTM). An LSTM network is a recurrent neural network that has LSTM cell blocks in place of our standard neural network layers. Here is a graphical representation of the LSTM cell:



On the left hand side, we have our new word/sequence value x_t being concatenated to the previous output from the cell h_{t-1}. The first step for this combined input is for it to be squashed via a *tanh* layer. The second step is that this input is passed through an input gate. An input gate is a layer of sigmoid activated nodes whose output is multiplied by the squashed input. These input gate sigmoids can act to "kill off" any elements of the input vector that aren't required. A sigmoid function outputs values between 0 and 1, so the weights connecting the input to these nodes can be trained to output values close to zero to "switch off" certain input values (or, conversely, outputs close to 1 to "pass through" other values).

The next step in the flow of data through this cell is the internal state / forget gate loop. LSTM cells have an internal state variable s_t. This variable, lagged one time step i.e. s_{t-1} is added to the input data to create an effective layer of recurrence. This ***addition operation, instead of a multiplication operation, helps to reduce the risk of vanishing gradients***. However, this recurrence loop is controlled by a forget gate – this works the same as the input gate, but instead helps the network learn which state variables should be "remembered" or "forgotten".

Finally, we have an output layer *tanh* squashing function, the output of which is controlled by an output gate. This gate determines which values are actually allowed as an output from the cell h_t.

In brief, whereas an RNN can overwrite its memory at each time step in a fairly uncontrolled fashion, an LSTM transforms its memory in a very precise way: by using specific learning mechanisms for which pieces of information to remember, which to update, and which to pay attention to. This helps it keep track of information over longer periods of time. This is how LSTM overcome the limitations of RNN.

RNN Vs. LSTM:

RNN stands for Recurrent Neural Network.
An LSTM is a Long Short-Term Memory layer, which is a type of RNN.

These layers, built into a network, function as some sort of memory that allows the network to infer from not only the present, but also past events. They do this by keeping a sort of "internal memory" that they modify to keep track of meaningful events that happened.

whereas an RNN can overwrite its memory at each time step in a fairly uncontrolled fashion, an LSTM transforms its memory in a very precise way: by using specific learning mechanisms for which pieces of information to remember, which to update, and which to pay attention to. This helps it keep track of information over longer periods of time. This is how LSTM overcome the limitations of RNN.