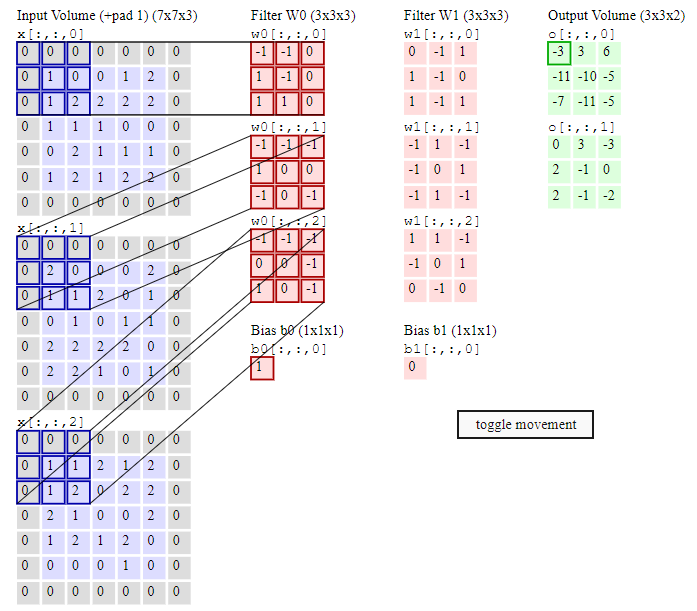
**Convolutional Neural Network**

**Reference:**

<https://cs231n.github.io/convolutional-networks/>



Input: 1

Channel: 3

Spatial dimension : 7\*7 (variables)

Depth dimension : 3 (fixed)

Filters: 2

Output volume 2 because of 2 filters.

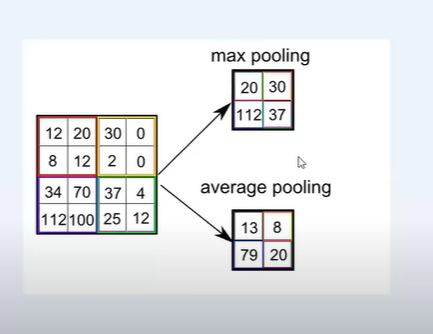
Stride:2

The weight and bias are shared and used to generate the output (3\*3 neurons). I think this is the main difference between CNN and Fully connected neural networks.

I have completely understood the calculations.

Conv layer -> pool layer ( Source: Class Lecture)

Local



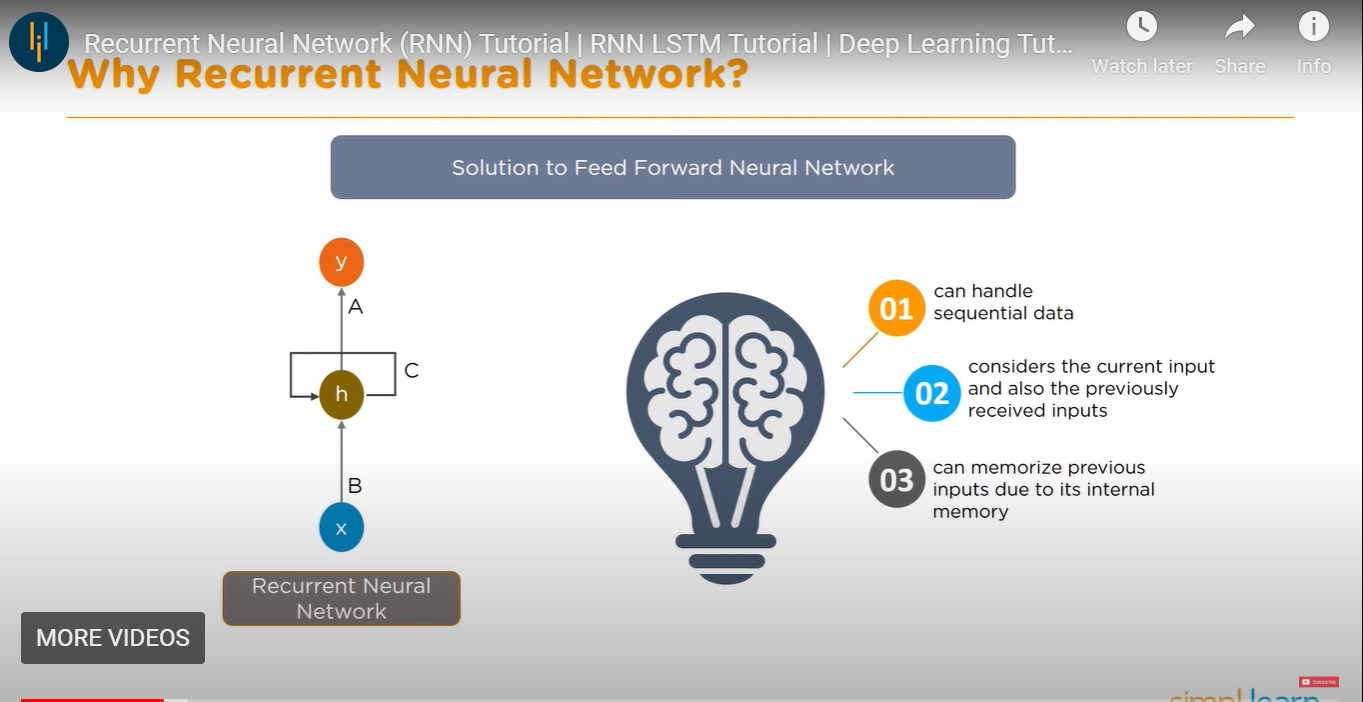
Global (generally used in the last layer)

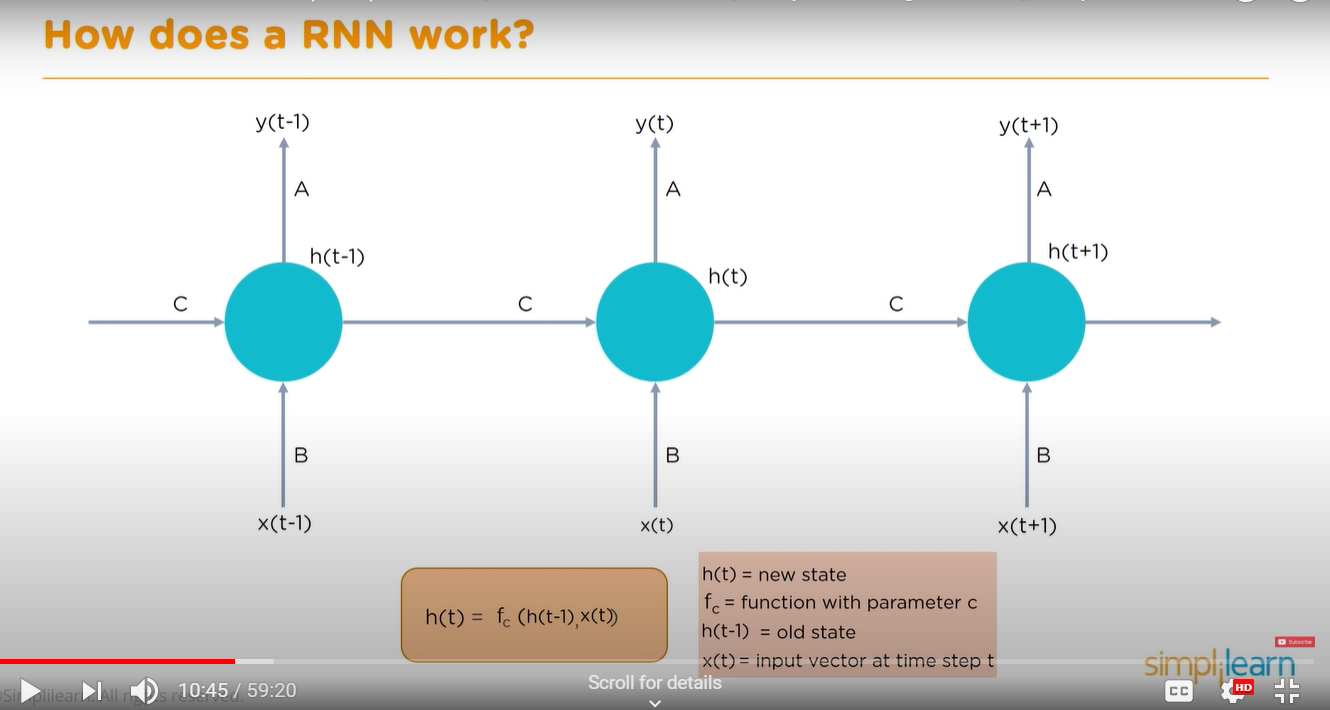
Maximum: Maximum in 4\*4 matrix

Average: Total/16

**Recurrent Neural Network:**

Source: <https://www.youtube.com/watch?time_continue=616&v=lWkFhVq9-nc&feature=emb_logo>





**Difference between CNN and RNN:**

**Source:** <https://www.quora.com/What-is-the-difference-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 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 functionality and performs feature extractions and finds out hidden patterns.

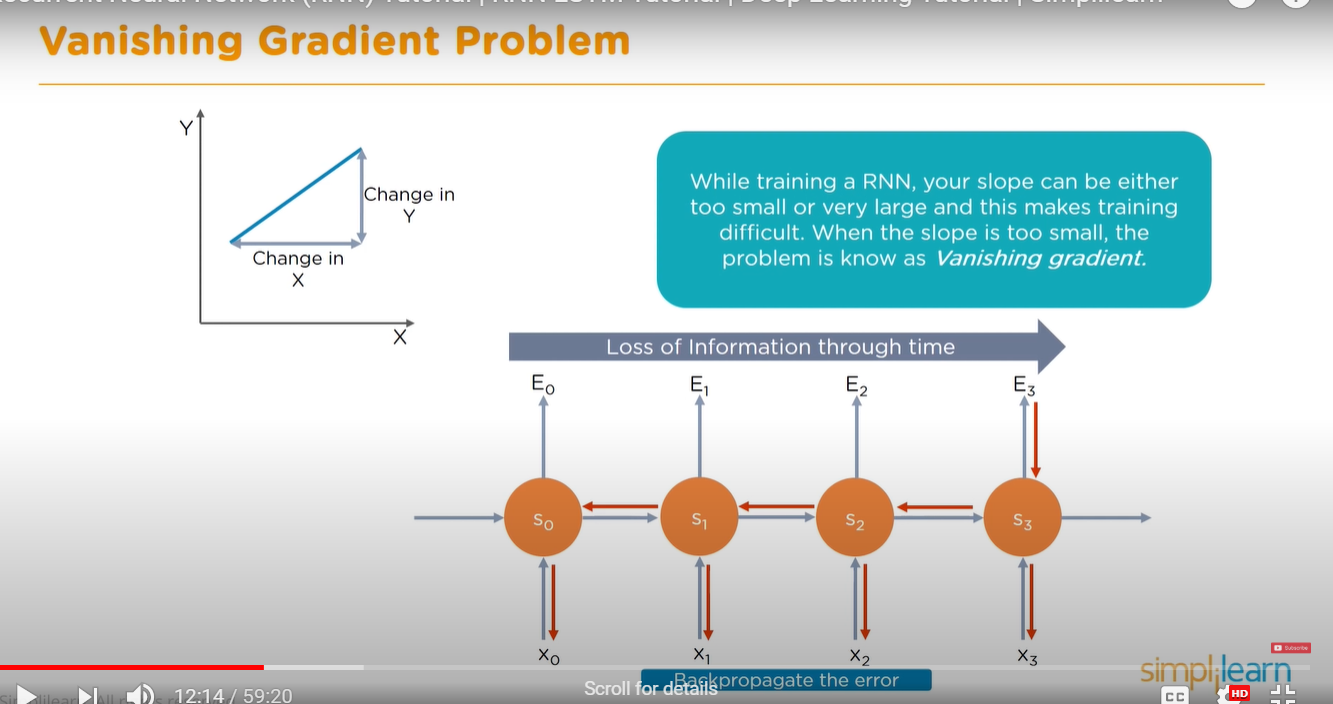
There are 4 types of RNN namely: One to One, One to Many, Many to One, and Many to Many. (didn’t learn much about these)

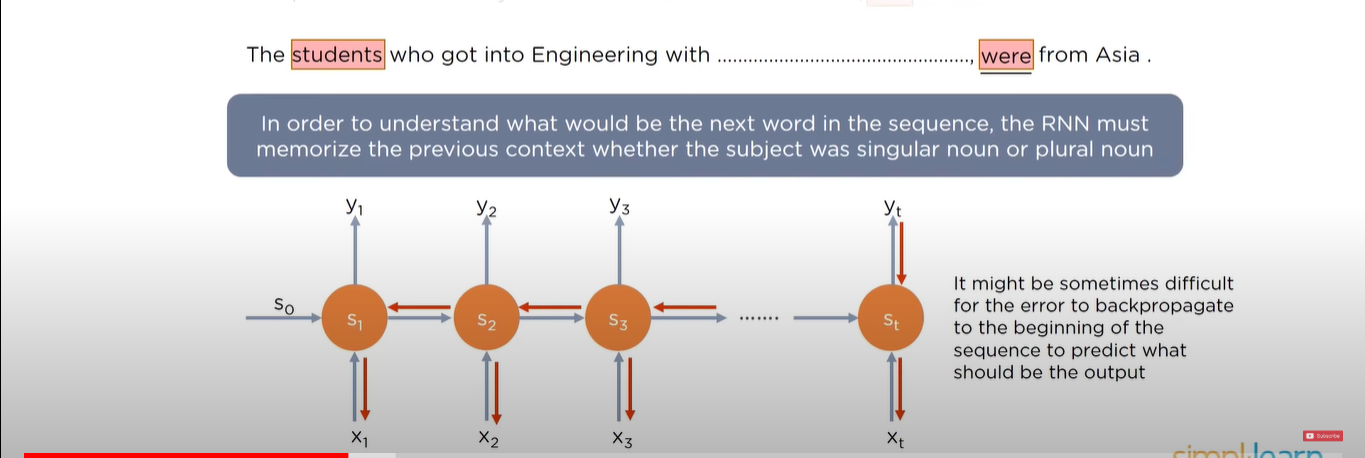
RNN can handle sequential data while CNN cannot. In RNN, the previous states are fed as input to the current state of the network. RNN can be used in NLP, Google search, Time Series Prediction, Machine Translation, etc.

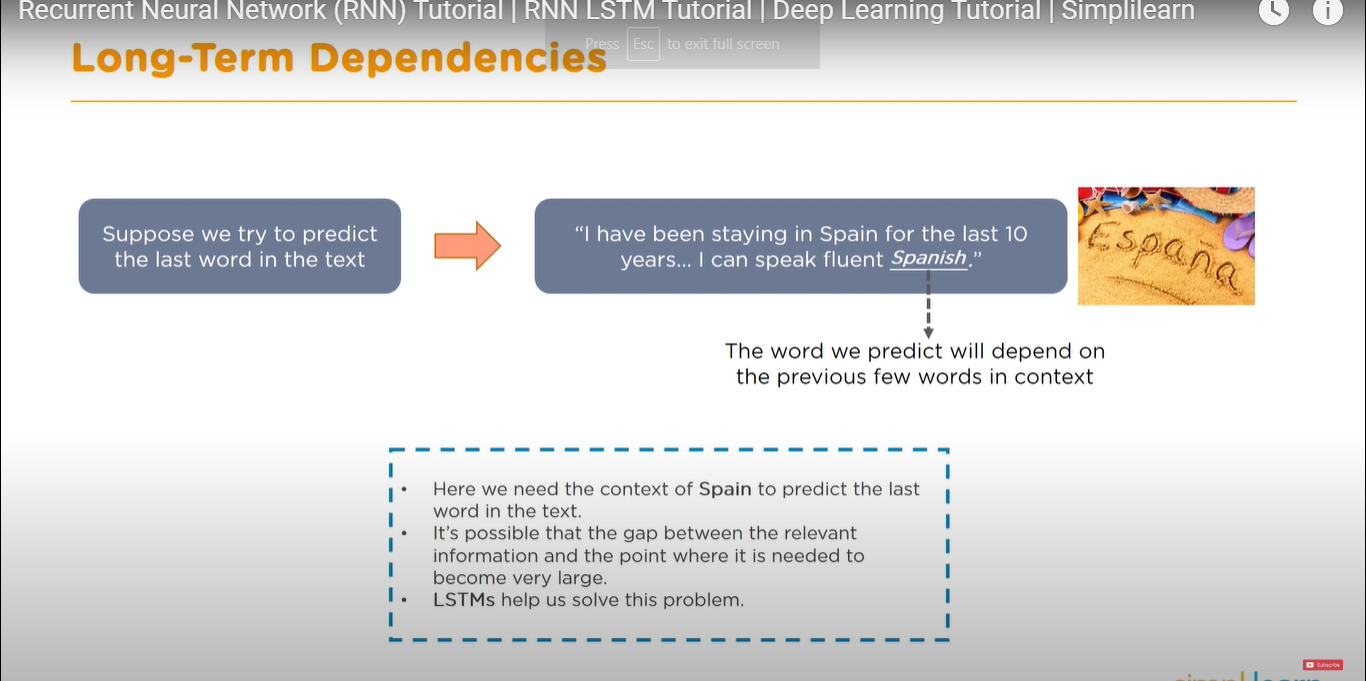
**Why did we use LSTM?**

Source: <https://www.youtube.com/watch?time_continue=616&v=lWkFhVq9-nc&feature=emb_logo>

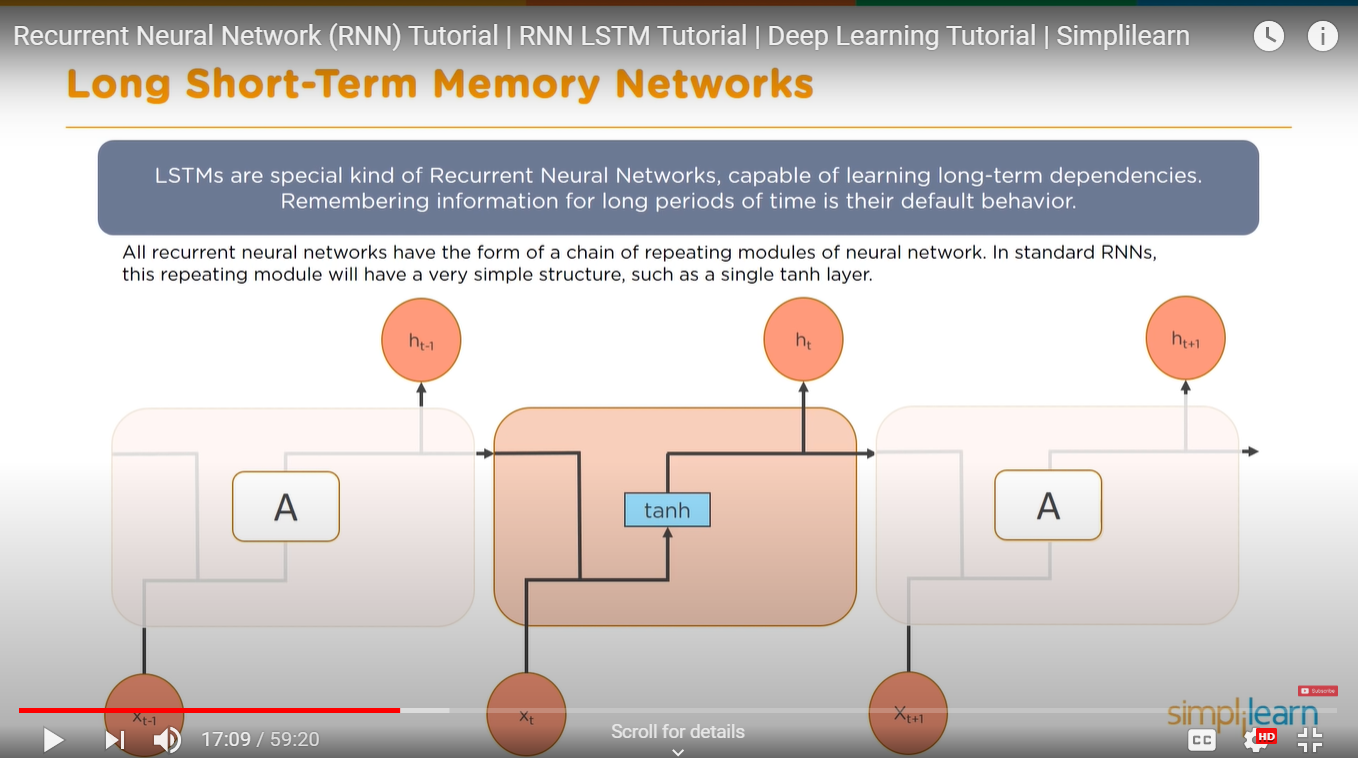
To solve the vanishing gradient problem.

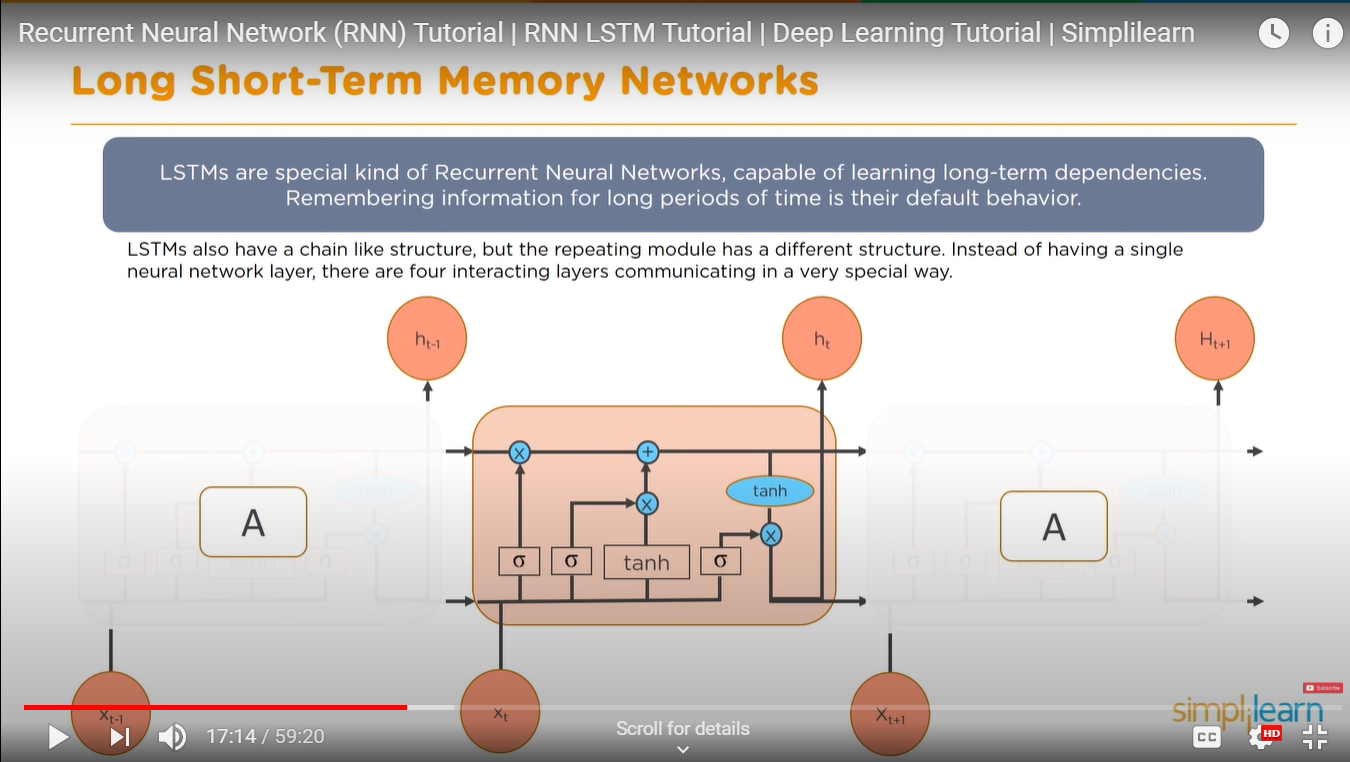






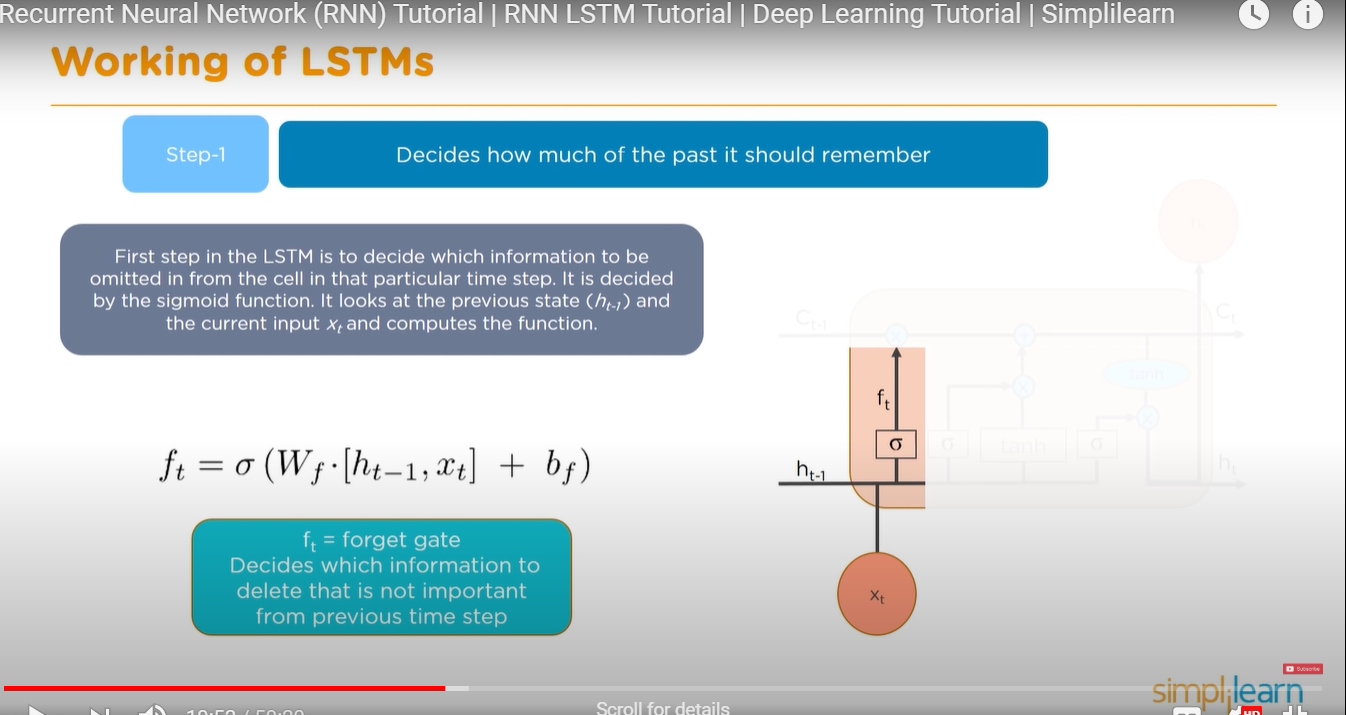
**RNN structure :**

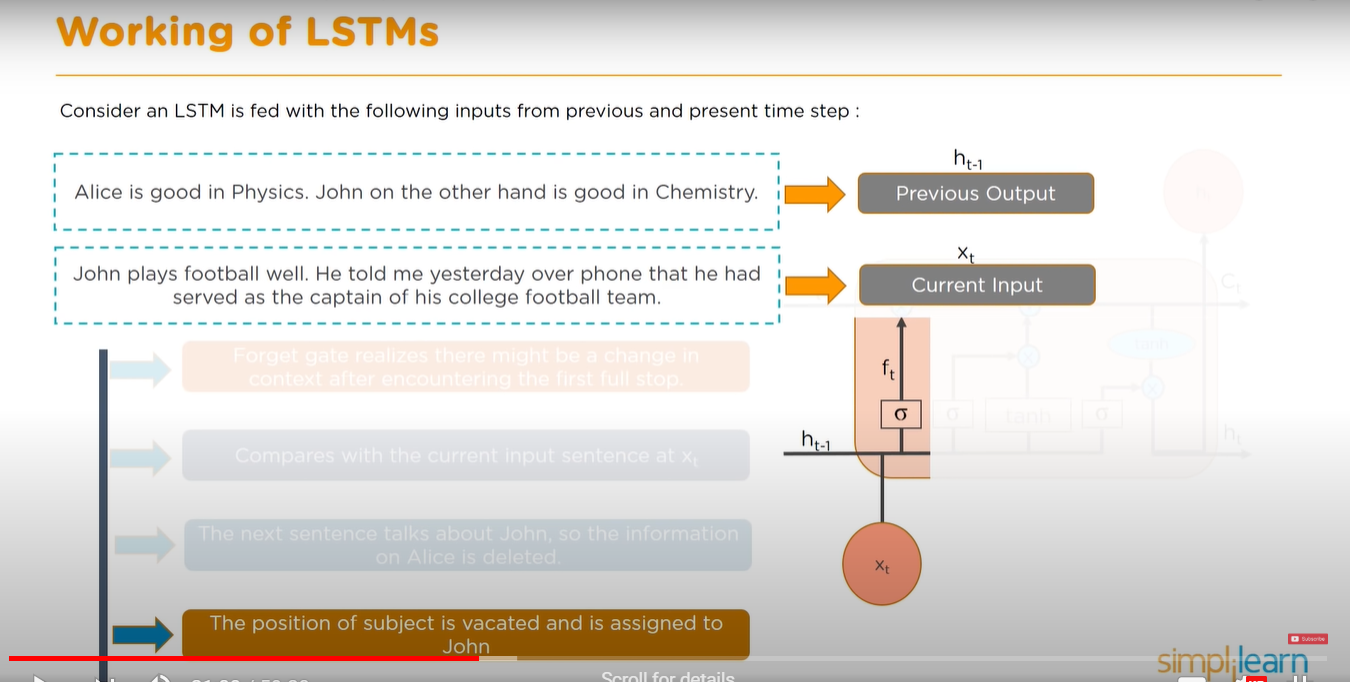


**LSTM structure:** 

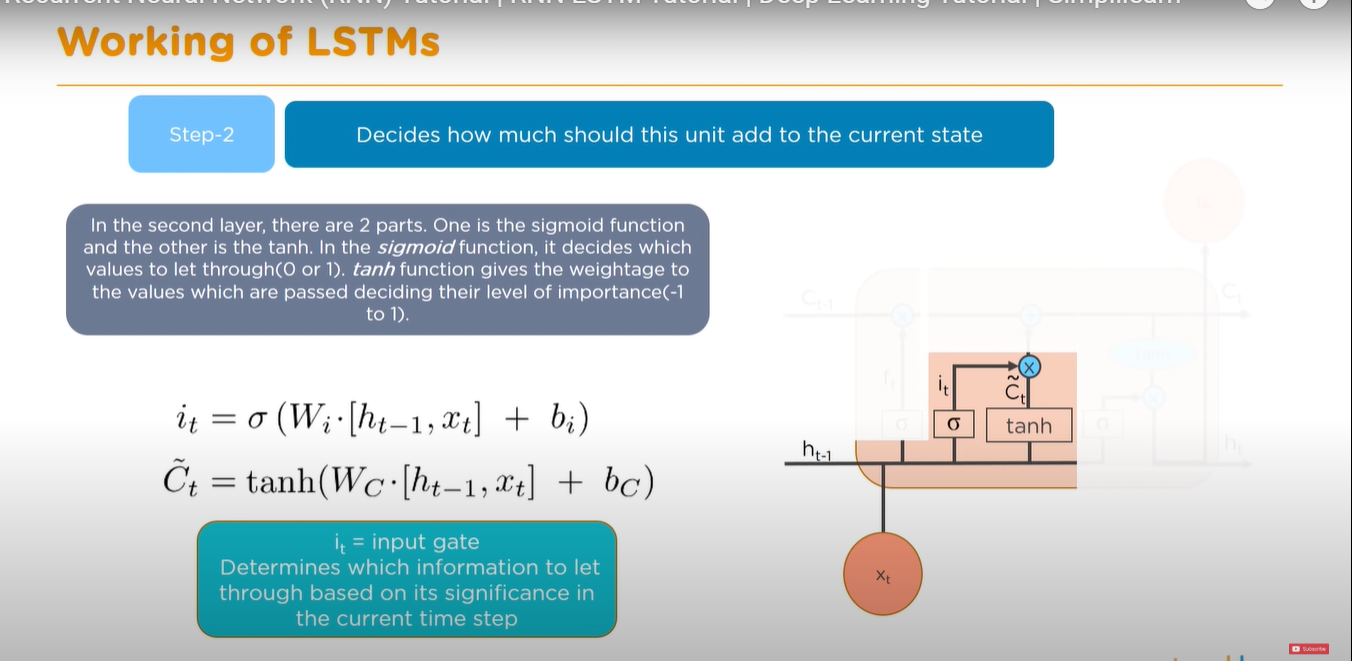
**How LSTM works:** Total 3steps.

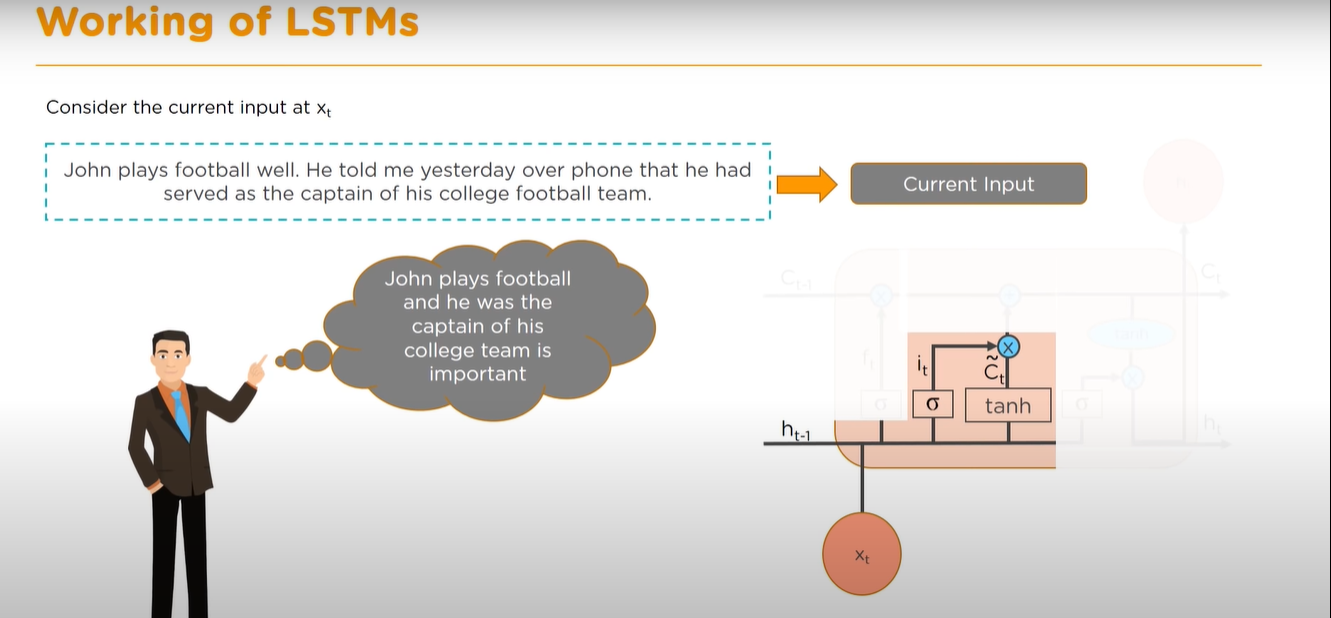
**Step 1:**



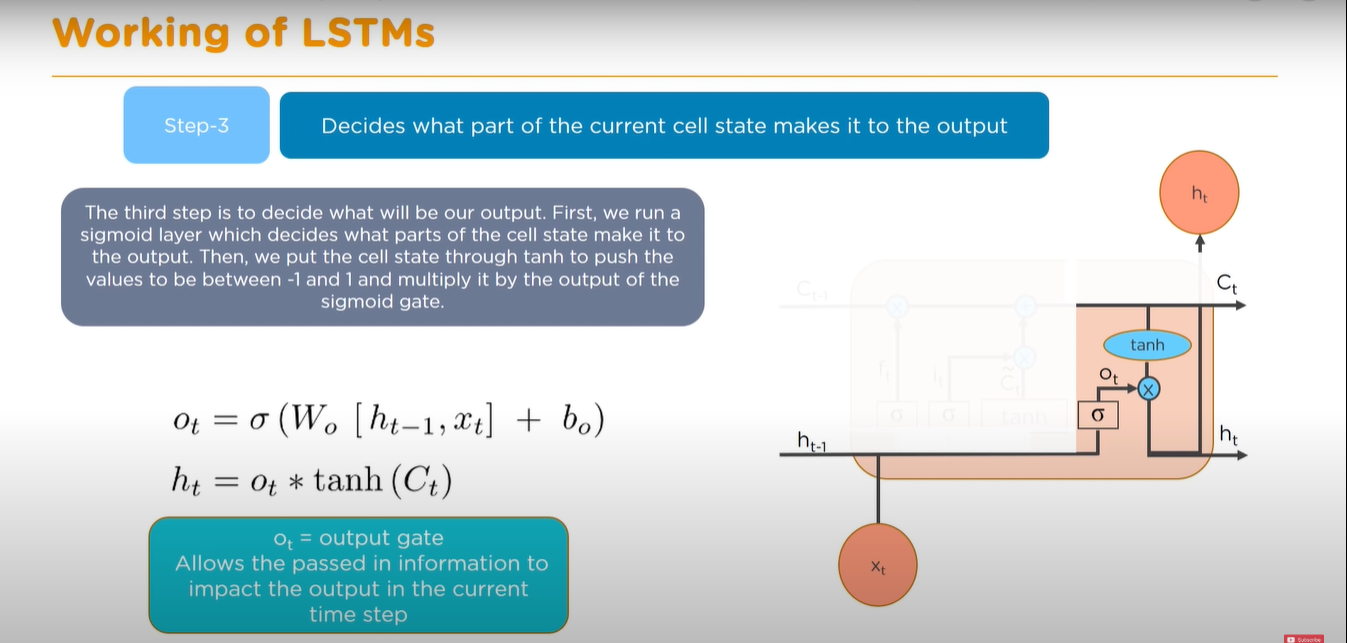


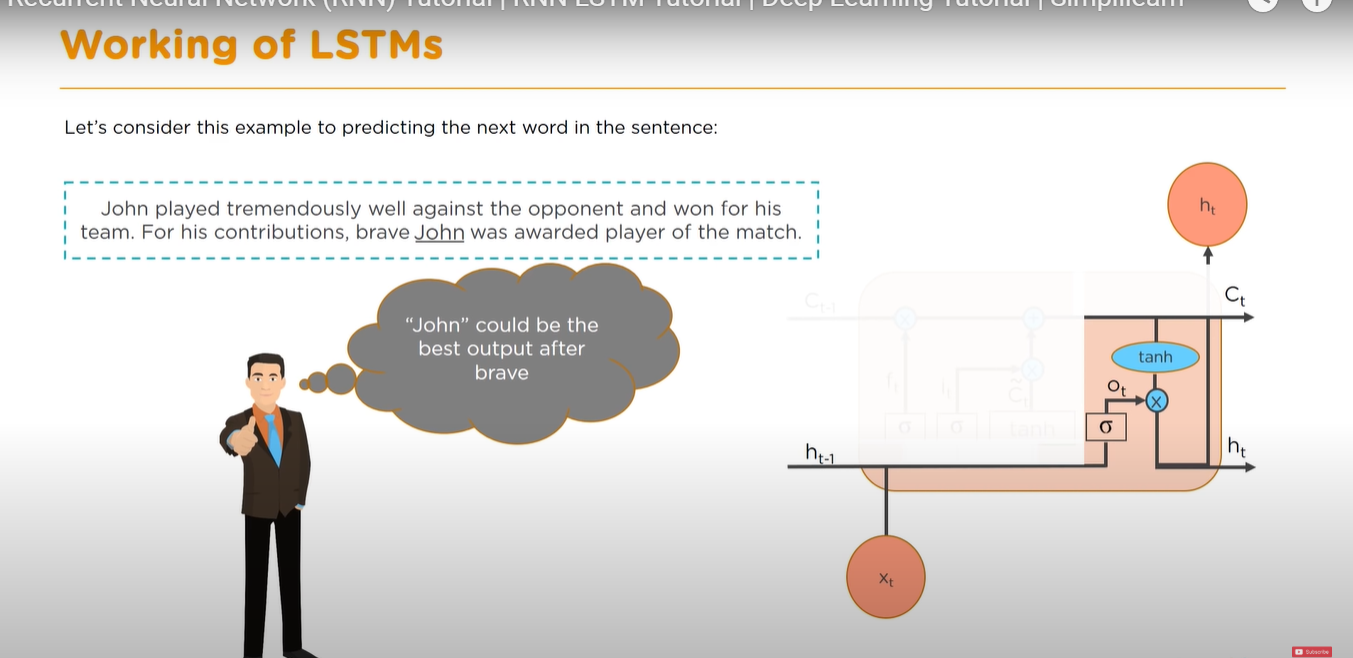
**Step 2:**





**Step 3:**





Bravae= adjective

So in the blank it should be a noun.

**Why do we come up with ROW LSTM?**

**Source:** <https://www.quora.com/Why-does-a-Row-LSTM-have-a-triangular-context>

You could run LSTMs on images even before row LSTMs were around. However, they were Sloooow to train. (I tried training one once, took a week and still did not train). The reason was they were pixel wise, trained one pixel at a time, that would be really slow given the number of pixels your dataset will have. So these LSTMs did the following: h(i,j) = lstm(h(i-1,j),h(i,j-1),p(i,j)), which is taking the hidden states of top and left pixel of a pixel to get hidden state of that pixel. The process slow because it was very hard to parallelize the training of pixels as they were all interdependent.

Row LSTMs tried to solve the slow training problem as well as the parallelization problem. The concept is what if we can take an input of an entire row and predict the next row. This would decrease the training time (You’re putting one row at a time instead of one pixel) and training of pixels can be parallelized.

To do this, they don't take the top and left pixel of the image like in normal 2D LSTM but both top diagonal elements. hence h(i,j) = lstm(h(i-1,j-1),h(i-1,j+1),p(i,j)). Thus a pixel is dependent upon the hidden state of pixels from only the row above or never dependent on the same row. So you can train an entire row in parallel.

* If you think about it, you lose the context of two pixels per row at each end recursively. This is what gives rise to a triangular context. So in row 2 the first pixel will be dependent on -1th and 2nd pixel (which is a non-existing combination and hence the hidden state is set to zero). in row 3, the second pixel will be dependent upon the row 2 first pixel, so will be zero, as well as row3 first pixel, which is also zero being dependent on -1th pixel of row 2. This tapering of contexts takes place at both ends of the row if you think carefully. ( This is the problem of ROW LSTM I guess. This is solved using Diagonal Bi LSTM I guess so ) .

**Diagonal BiLSTM : Future study**