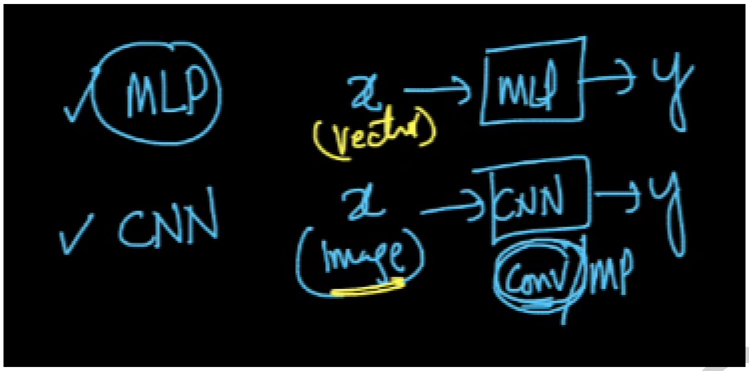
62.1 Why RNNs?

Recurrent Neural network is referred to in short as RNN.

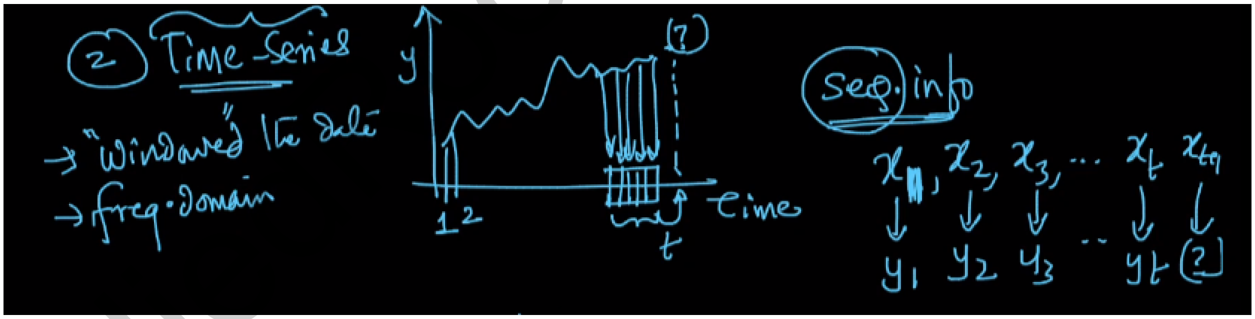
MLP works fairly well when our input data is a vector. And if the input data is an image, CNN works well.



What if the input data is a Sequence..?

Examples of Sequence data:

1. Amazon reviews. Where the sequence is very important.
2. Time series data



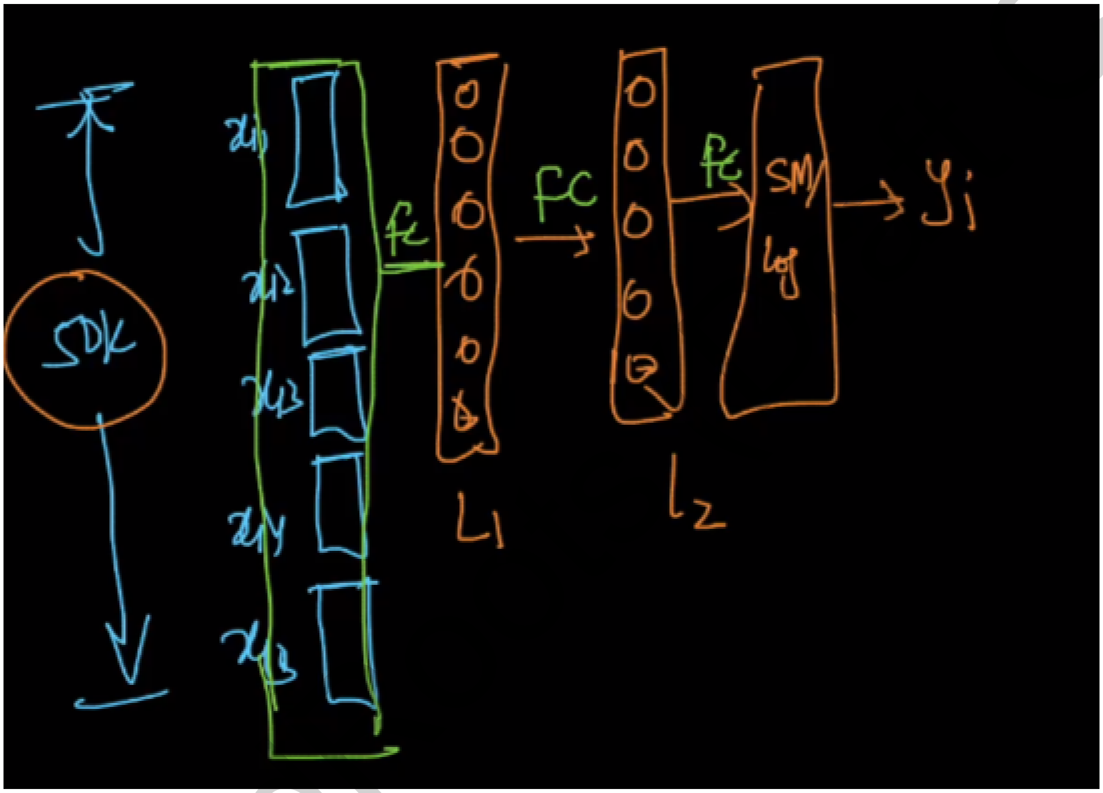
3. Machine Translation.
4. Speech Recognition
5. Image Captioning.

So, If the output depends on the sequence of inputs, then we need to build a different type of neural network that retains and leverages the sequence information. To perform better than the nonsequence type models.

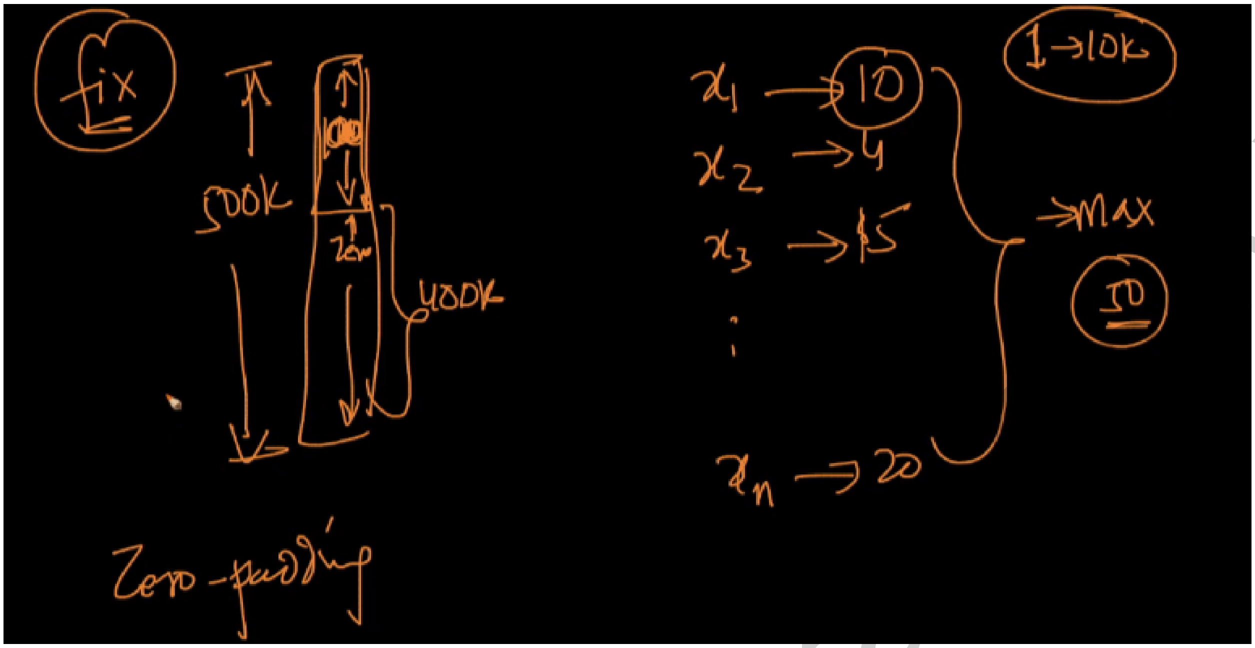
Let us try to solve the problem of Amazon food reviews using MLP.
Let us take the sentence X_i “This phone is very fast.” and yi is the prediction 0 or 1.

$\mathbf{x}_{i1}$, $\mathbf{x}_{i2}$, $\mathbf{x}_{i3}$, $\mathbf{x}_{i4}$, $\mathbf{x}_{i5}$

We can represent the words as a one-hot encoded vector.
Let us say we have a vocabulary size of 10k. So $x_{ij} \in R^{10k}$
So we have 5 vectors of size 10k.



The key problem with this Approach is the input size keeps on changing
Eg:
x1: This phone is very fast
x2: This phone is good.
So we can do vector padding to fix this problem:

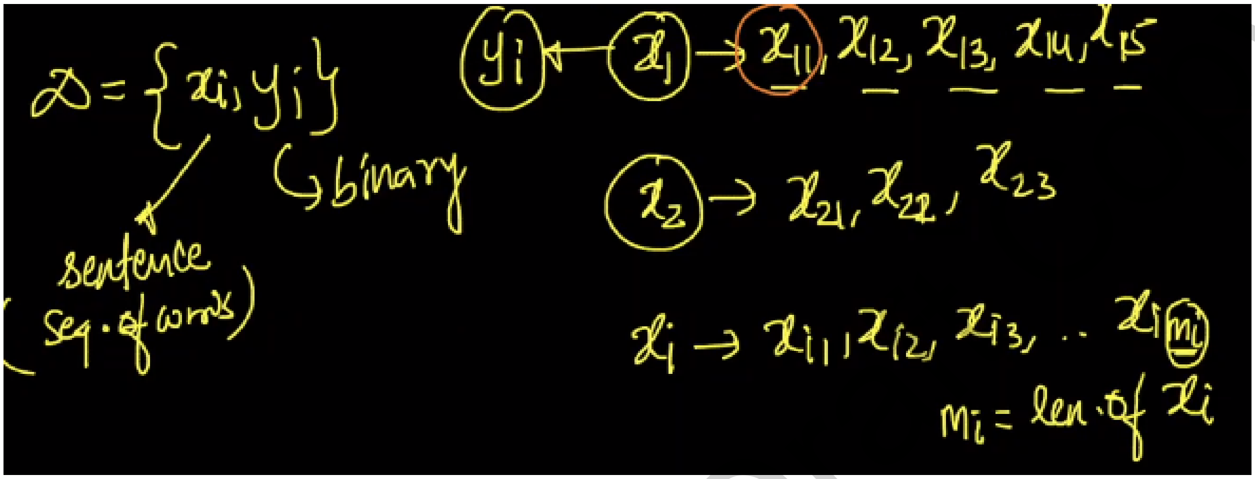


But if the query point size does not match with the input size, the fix fails. And with this zero padding, the network becomes massive which will have numerous unnecessary connections.

So, As a fix for all these problems, we came up with a new network model called **Recurrent Neural Network (RNN)**.

62.2 & 62.3 Recurrent Neural Network

Recurrent refers to repeating. We design a new type of neural network that has a repeating structure. Let us take an amazon food review problem which is a binary classification.



$D = \{x_i, y_i\}$
binary
sentence (seq. of words)
$y_i \leftarrow x_1 \rightarrow x_{11}, x_{12}, x_{13}, x_{14}, x_{15}$
$x_2 \rightarrow x_{21}, x_{22}, x_{23}$
$x_i \rightarrow x_{i1}, x_{i2}, x_{i3}, \ldots x_{im_i}$
m_i = len. of x_i

Here x_{11}, x_{12}… all are words and represented by a one-hot encoding. Let the size of the vocabulary be ‘d.’

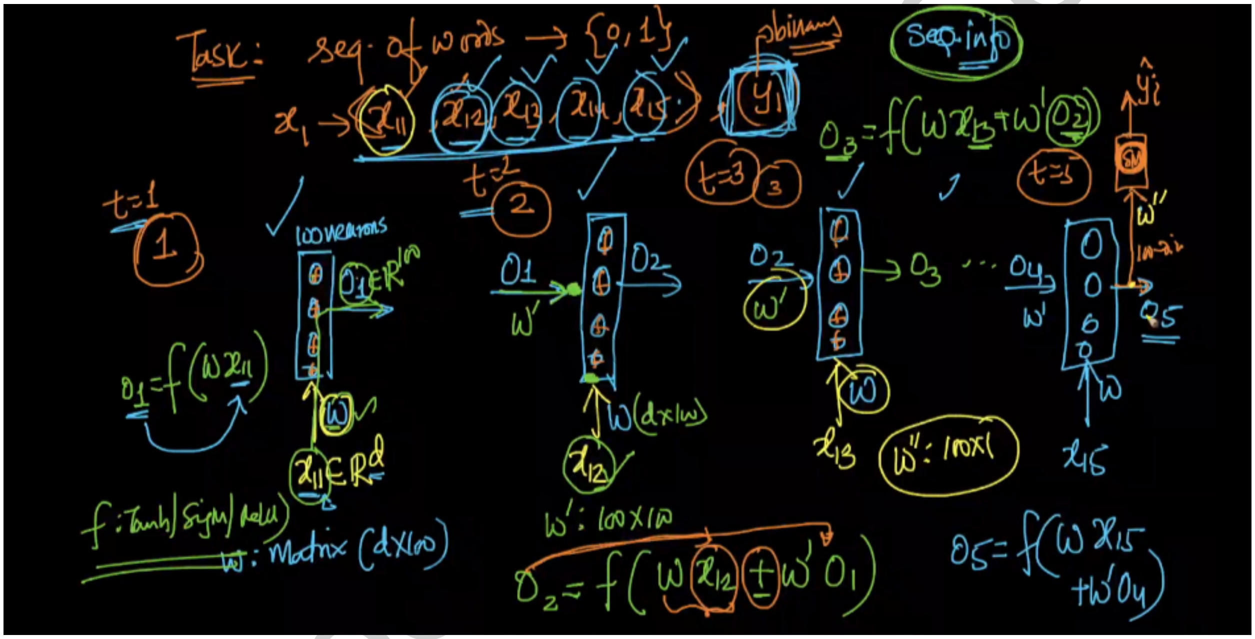
Task: To classify the sequence of words into either 0 or 1.

X_1 :<x_{11},x_{12},x_{13},x_{14},x_{15}> , y_1

1. We process only x_{11} at a time ‘t’ =1.
   At t=1, we input x_{11} into the input layer of 100 neurons and so the weight matrix “W” will be in the size of d*100.
   The activation function “f” might be tanh/sigmoid/softmax
   Let the output from the layer be O_1.
   O_1=f(W*x_{11})

2. At t=2,
   we input x_{12} into the network and the output is O_2.
   And this output O_2 is obtained from the present input word ‘x_{12}’ and the output of the previous word ‘x_{11}’. Let the weight matrix associated with the previous output be w_dash.
   O_2=f(w_dash*O_1+W*x_{12})

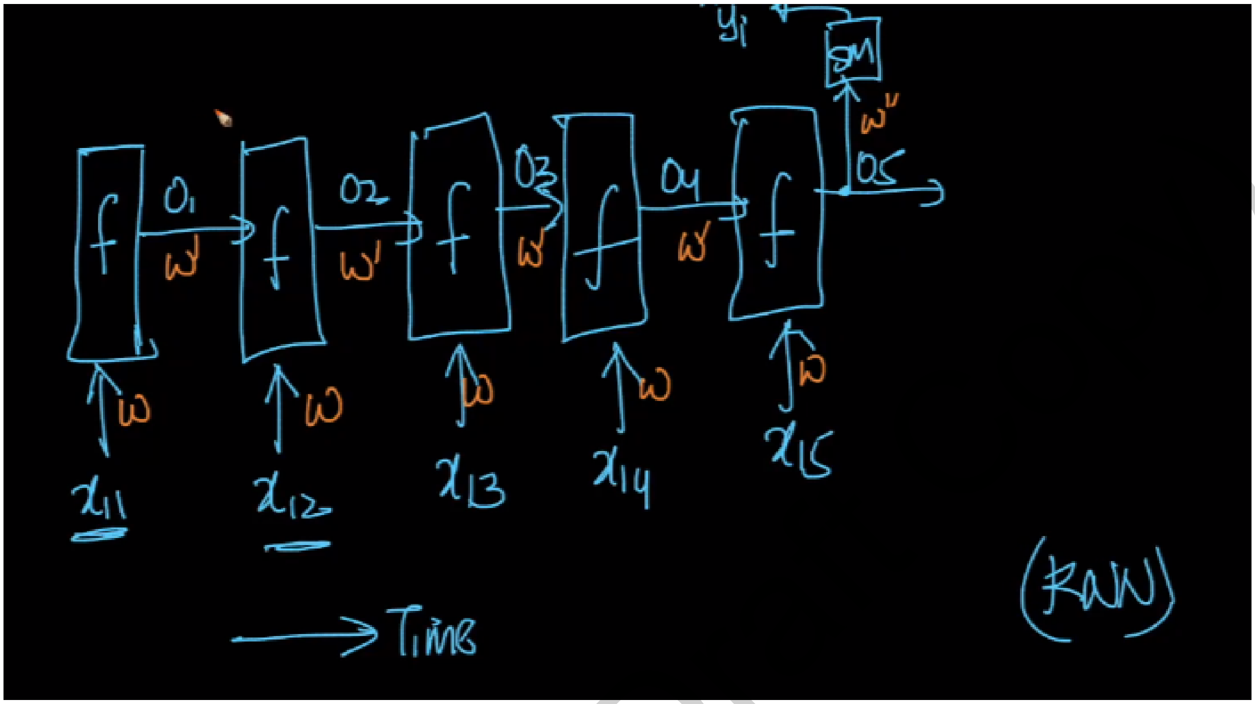
3. At t=3,
   Into the same network, we input the word x_{13} and the output from the network is O_3.
   This O_3 is dependent on x_{13}, and the previous output O_2(which is dependent on x_{12} and O_1). So, Indirectly, the present output is considering the information of all previous words in sequence.
   O_3= f(w_dash*O_2 + W*x_{13})
   This process continues till we reach t=5.



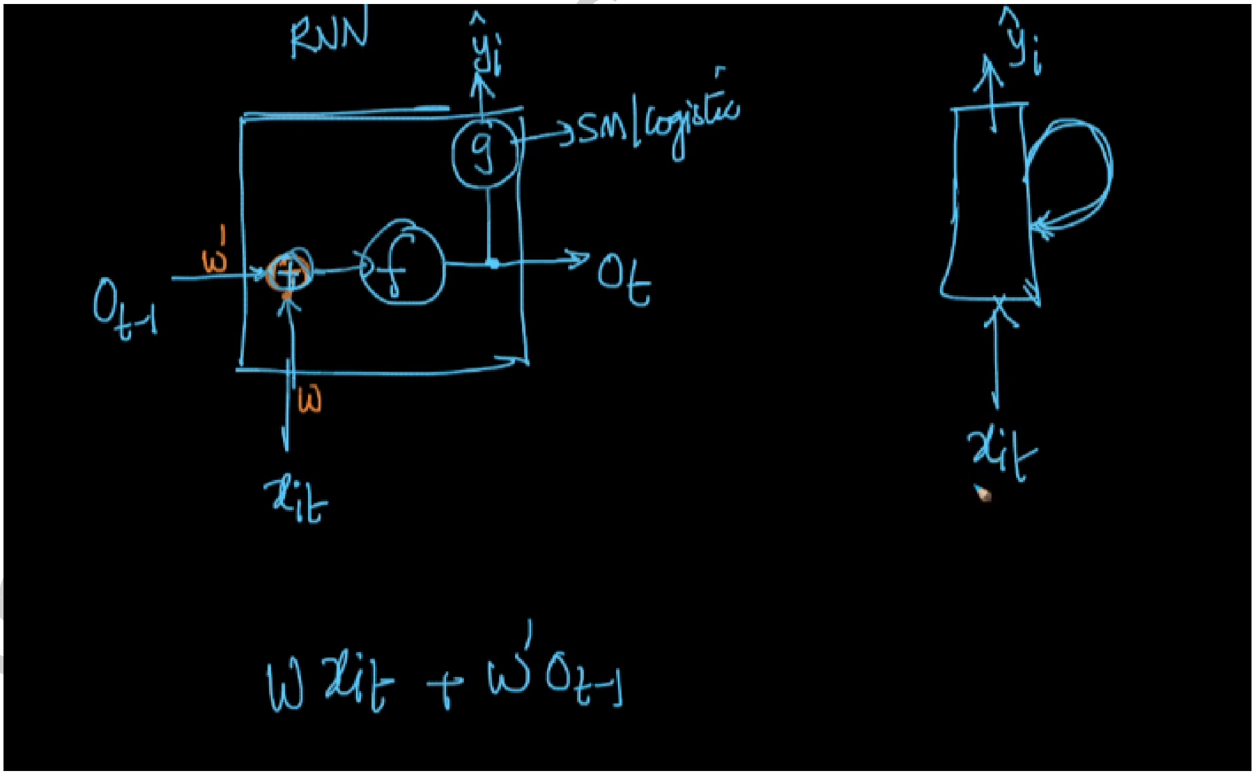
As we have reached the end of a sentence, we generate the y_1.
Here y_1 is the binary output, So, we apply a sigmoid layer on the output of O_5. The weight matrix “w_bar” at the output will be of size 100*1.
y_i=sigmoid(w_bar * O_5)

Note: W_doubledash in videos is represented as w_bar for convenience.

The below image is the summarization of all the processes.
In RNN’s we have 3 weight matrices. W, W_dash, W_bar.
A similar process happens for the following reviews, X_2 and X_3...

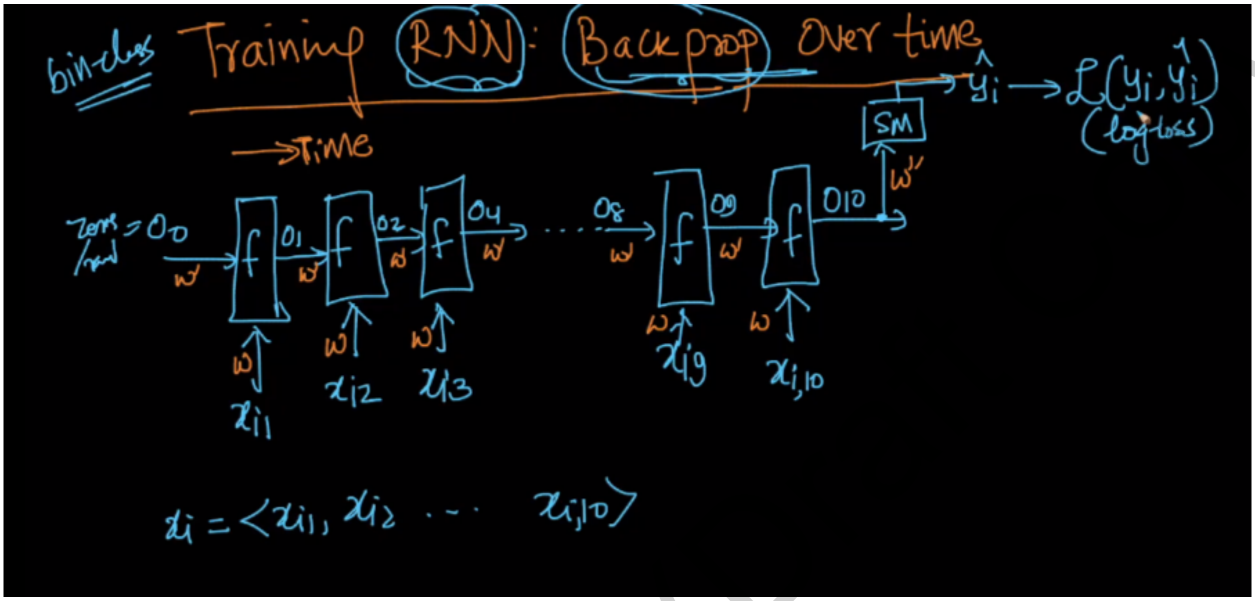


The above image of RNN can be represented in precise as follows:



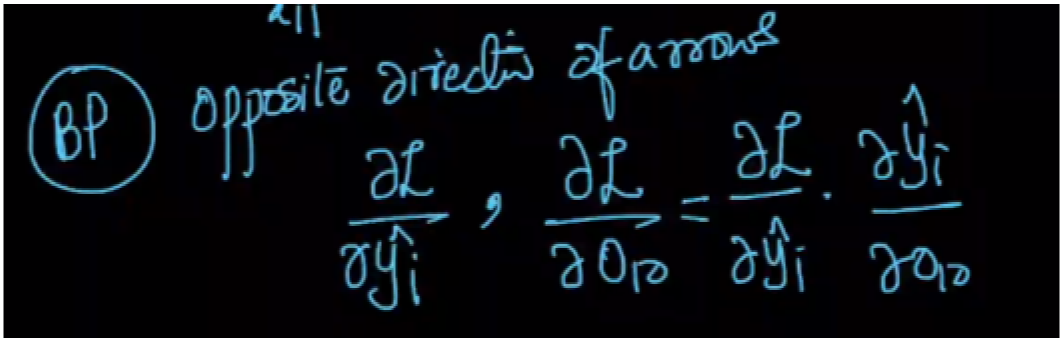
62.4 Training RNNs: Backprop over time

Back propagation used in RNN is called Backpropagation over time.



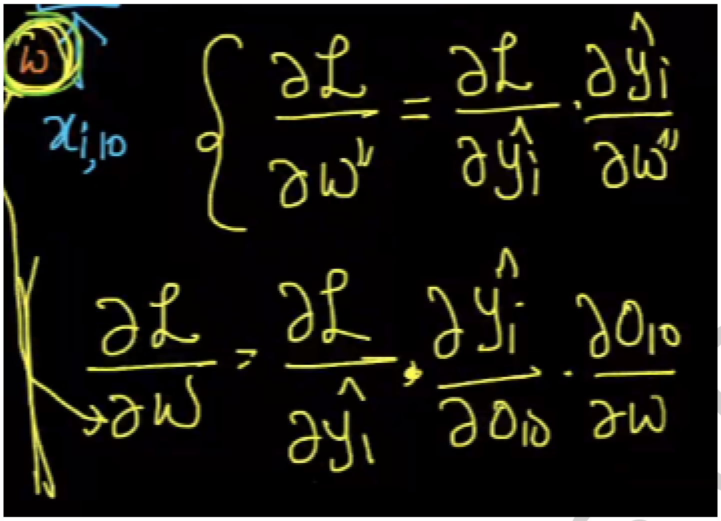
O_1= f(w*x_{i1} + w_dash*O_0)
O_2= f(w*x_{i2} + w_dash*O_1)
.
.
.
O_{10}=f(w*x_{i10} + w_dash *O_9)
yi= g(w_bar*O_{10})

For Forward propagation, we can just follow the direction of arrows to get these equations.
For backward propagation, we follow the opposite direction of arrows.
First we need to compute the derivatives.
dL/dy_i,
dL/dO_{10} = dL/dy_i * dy_i/dO_{10}
dL/dw_bar = dL/dy_i * dy_i/dw_bar
dL/dw = dL/dy_i * dy_i/dO_{10} *dO_{10}/dw



BP opposite directn of arrows

$$\frac{\partial L}{\partial \hat{y}_i}, \quad \frac{\partial L}{\partial O_{10}} = \frac{\partial L}{\partial \hat{y}_i} \cdot \frac{\partial \hat{y}_i}{\partial O_{10}}$$



w

$x_{i,10}$

$$\frac{\partial L}{\partial w'} = \frac{\partial L}{\partial \hat{y}_i} \cdot \frac{\partial \hat{y}_i}{\partial w''}$$

$$\frac{\partial L}{\partial w} = \frac{\partial L}{\partial \hat{y}_i} \cdot \frac{\partial \hat{y}_i}{\partial O_{10}} \cdot \frac{\partial O_{10}}{\partial w}$$

So, we have only three weight matrices for update. By the end of back prop, we will be having the final weights.

But there is a problem with this approach:

As we travel backwards for n times (where n is equal to number of words), by the time we reach the first word, we are easily running into vanishing gradient problems.

This is due to the fact that updates in the weights happen by multiplying the partial derivatives(<1).

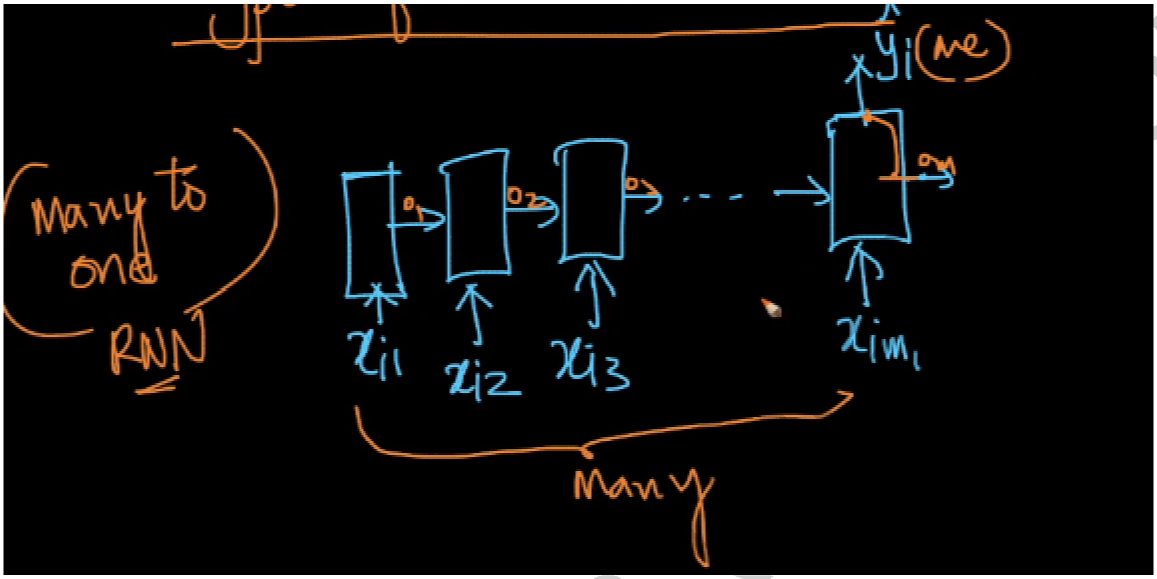
Due to this same reason we might also have an exploding gradient problem if the partial derivatives are greater than 1.

So to overcome this vanishing/ exploding gradient problem, we come up with special RNNs called GRUs and LSTMs.

62.5 Types of RNNs

1. Many to one RNN :

Till now we have seen a regular type of RNN like this.



This is called many to one RNN because the input is a sequence of many values and the output is just one.

The applications of this type of RNNs are

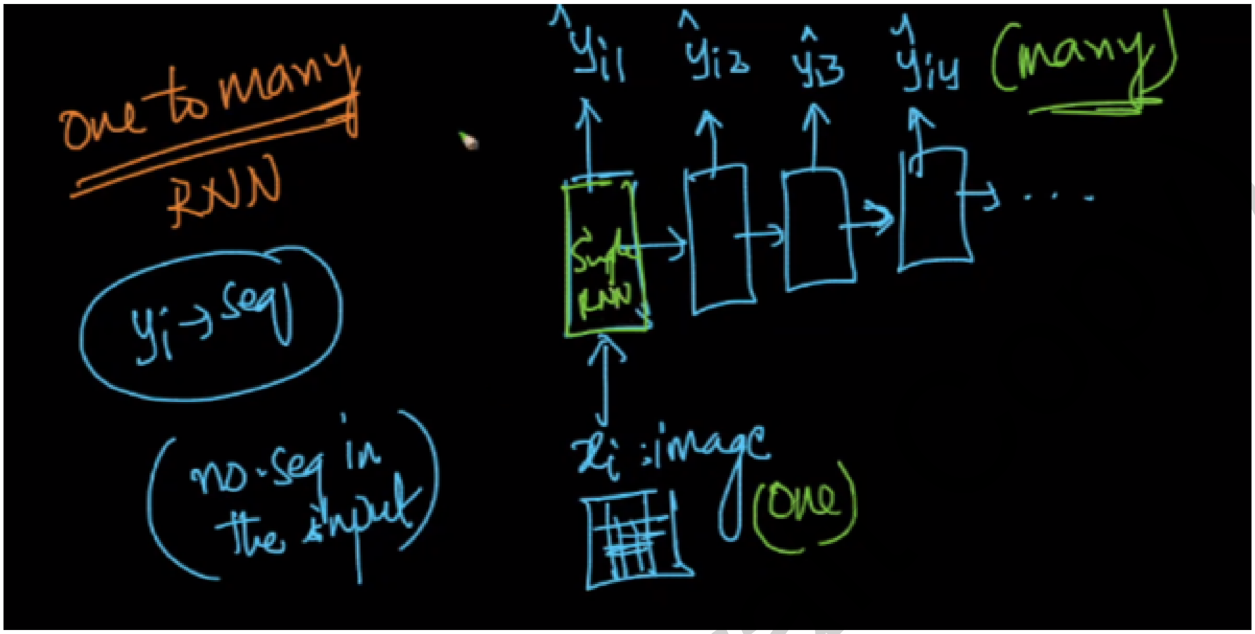
1. Sentiment classification (like Amazon food review)
2. Movie review.

   We can think of this like a multi class problem as the output ranges from 1 to 5 stars.

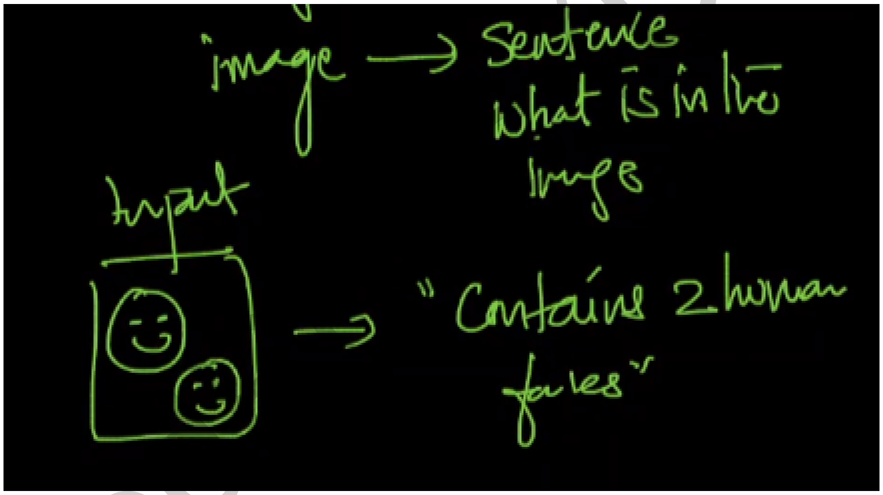
2. One to many: Image captioning:

The input to this kind of RNN must be only one term. And the output will be many values.

The Architecture of such type is shown below.

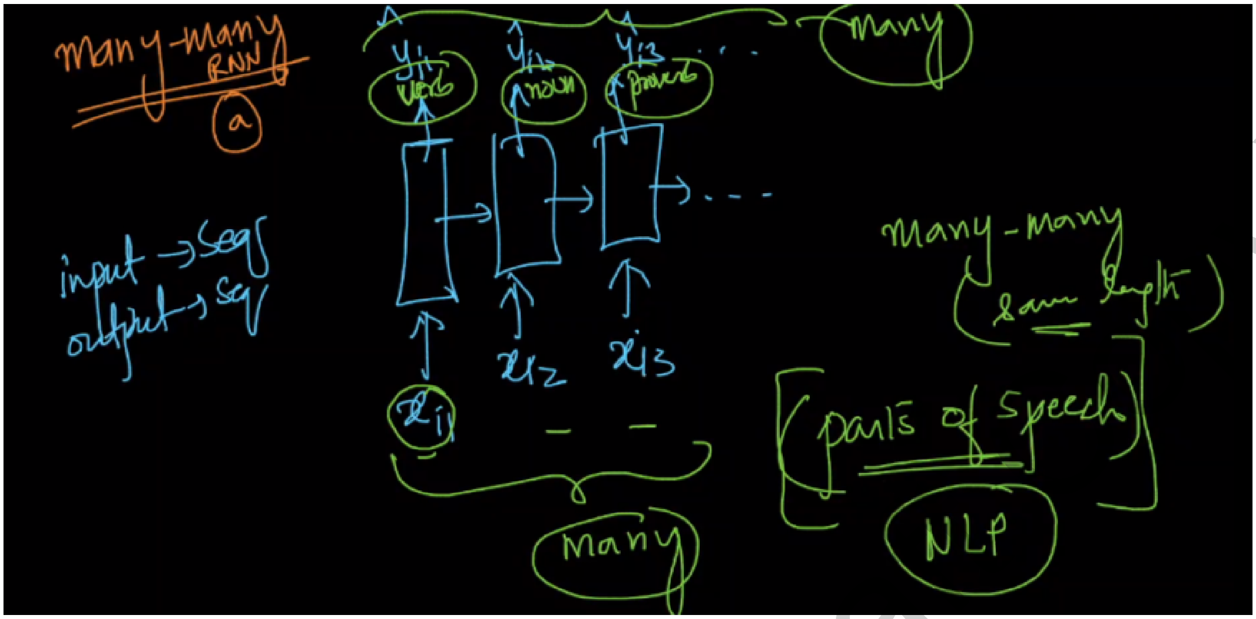


In image captioning, we provide a single image and the output will be sequence.



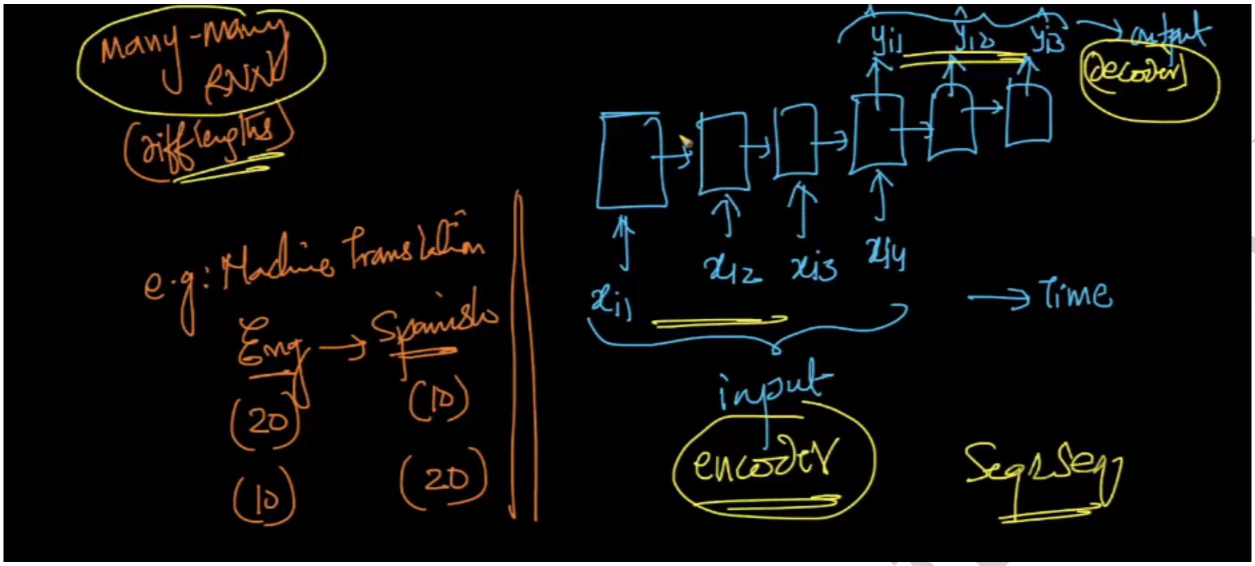
3. Many to many :

In many to many RNN, both the input and output will be the sequences given that the input and outputs are of the same length.
The Application for this is parts of a speech detector.



4. Many to many: different length:

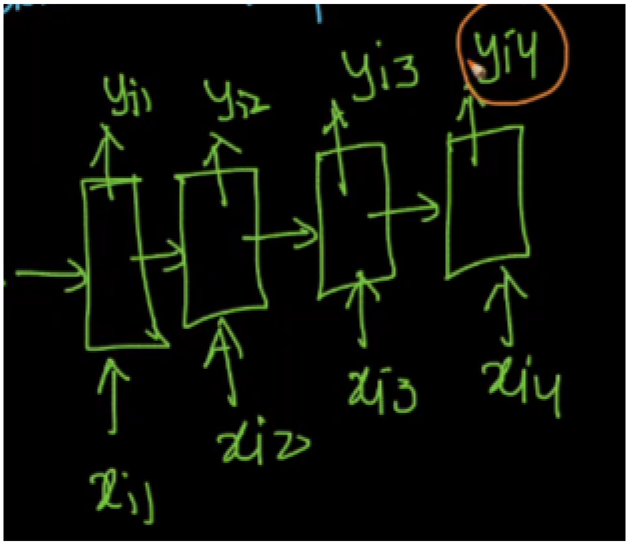
The input and output are of different length.
Popular application is Machine Translation, where we give a sequence of words in one language and the output will be a sequence of words in another language.
Let us take the task of English to Spanish translation. The input in english may be 10 words but the translation may contain only 5 words.
The architecture of this model is shown below:
The output starts only after finishing the inputs. The RNN feeds in all of the inputs and generates the outputs later. This is also called encoder decoder models and sequence- sequence models.


Many-Many RNN
(diff lengths)
e.g: Machine translation
Eng → Spanish
(20) (10)
(10) (20)
y_i1 y_i2 y_i3 → output
Decoder
x_i1 x_i2 x_i3 x_i4
→ Time
input
encoder
Seq2Seq

62.6 Need for LSTM/GRU.

In this chapter we learn about the different types of architecture like LSTM/GRUs.

Let's take many-many RNN of the same length.



The output y_{i4} depends a lot on x_{i4} and the dependence decreases gradually while moving to the earlier words.

That is, y_{i4} depends less on x_{i3} and much less on x_{i2} and less on x_{i1}.

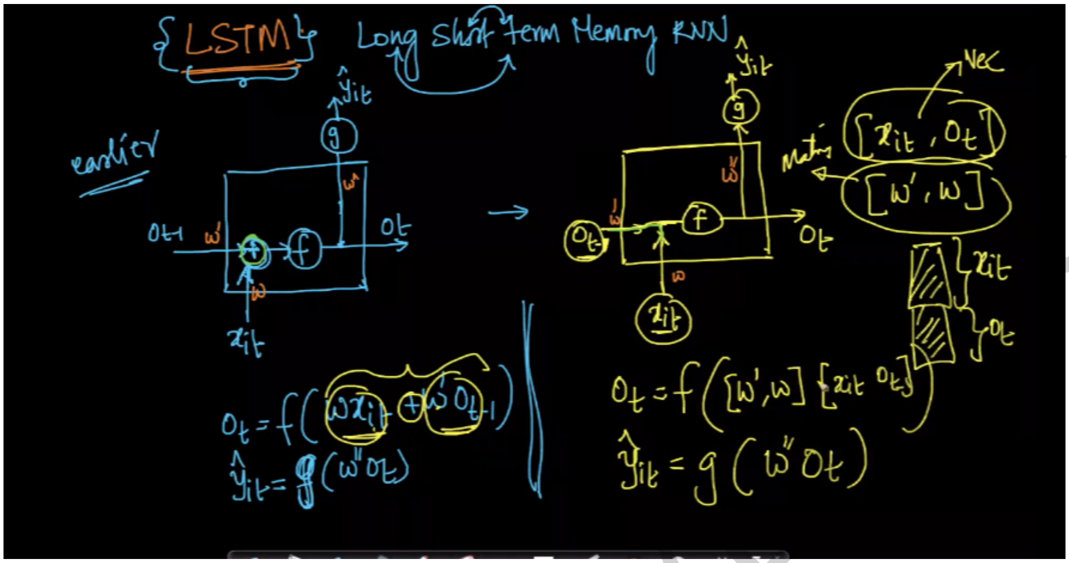
But in the real world, we may have applications like y_{i4} depending more on x_{i1} and less on x_{i2} and x_{i3}.

Both in forward pass and backward pass, if the later outputs are dependent on earlier words, It is very hard to make it work for backpropagation. That is the simple RNN cannot handle or work well for long term dependencies.

In order to work for long term dependencies, we came up with the idea of LSTM/GRUs.

62.7 LSTM

LSTM is an abbreviation for Long Short Term Memory RNN



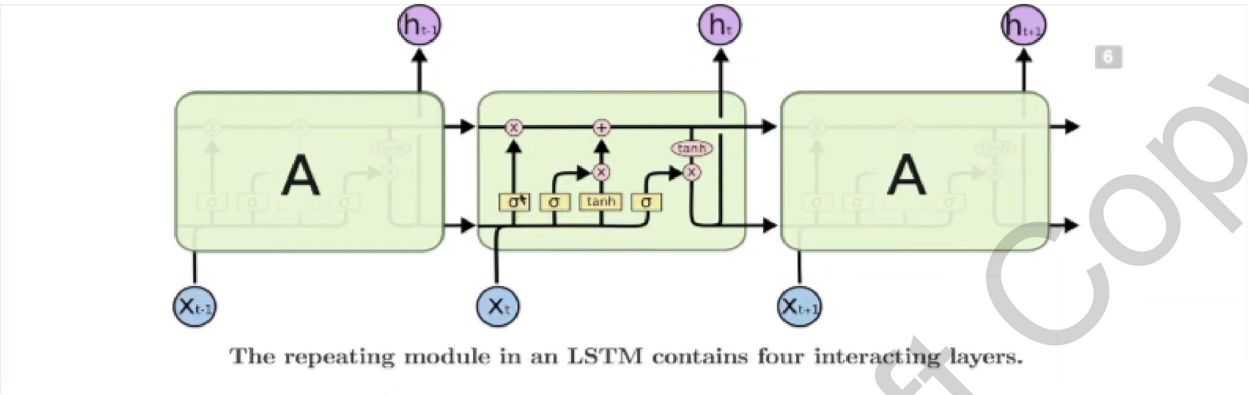
Simple RNN (SUM) :

$O_t = f(Wx_{it} + W'O_t-1)$
y_{it} hat = $g(W'' O_t)$

Modified RNN (Concatenation) :

$O_t = f([W,W'] [x_{it},O_t-1])$
y_{it} hat = $g(W'' O_t)$

Notation in LSTM



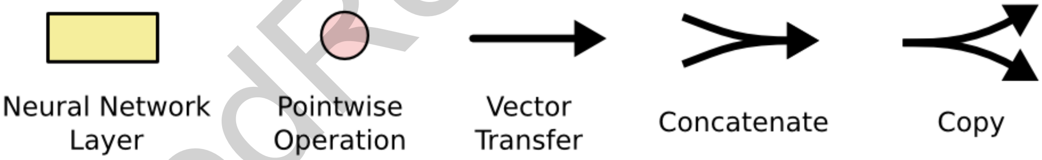
The repeating module in an LSTM contains four interacting layers.

X_t input to LSTM at timestamp t.

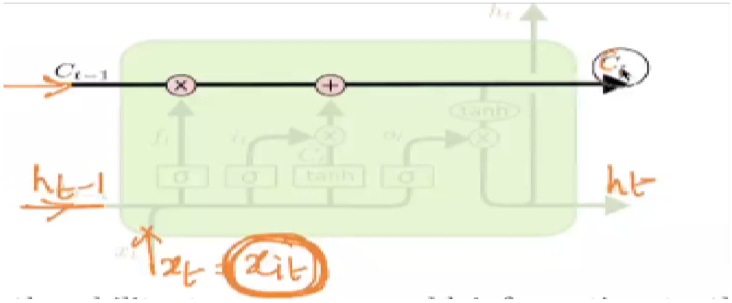
C_t (Cell state) is the state of the cell that is passing from one time stamp to another.

H_t (Hidden state): It is output from the cell at timestamp t.

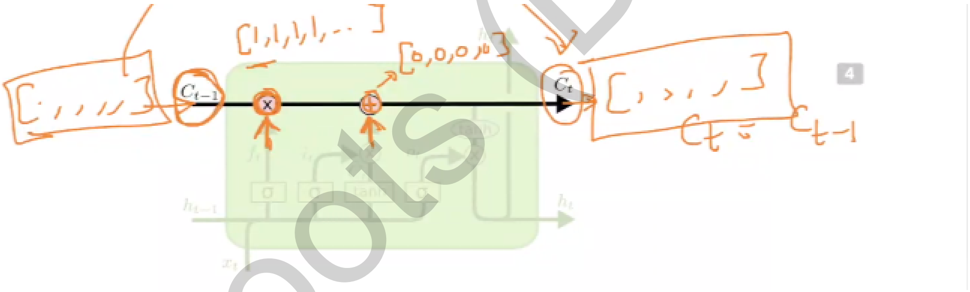
Operations used in LSTM



LSTM Walk Through :



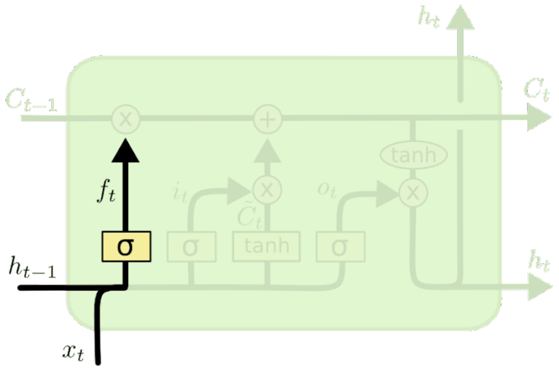
Lstm has three inputs and two outputs:
X_t , C_{t-1}, H_{t-1} are input to the current cell, whereas C_t and H_t are outputs



The LSTM does have the ability to remove or add information to the cell state and is also possible to pass information unchanged.

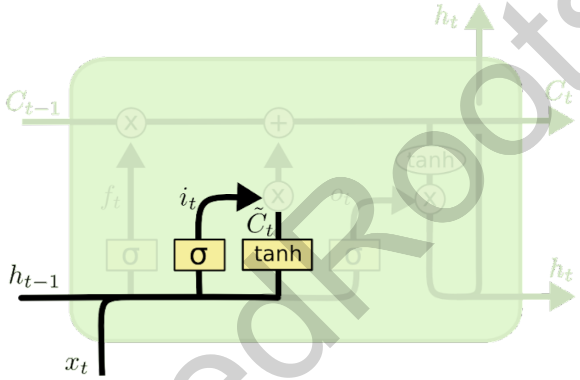
The sigmoid layer outputs numbers between zero and one, describing how much of each component should be let through. A value of zero means “let nothing through,” while a value of one means “let everything through!”

If pointwise multiplication has all ones [1,1,1,1,1….] and point wise addition has all zeros [0,0,0,0,0…..] then C_t = C_{t-1}. We can observe similar skip connections in RESNETS.



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

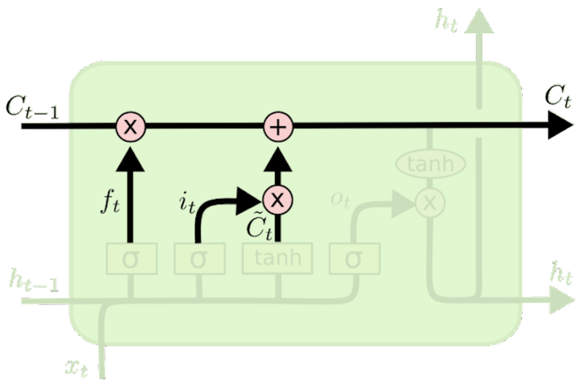
In this step output from the previous state(h_{t-1}) and input (X_t) are concatenated and sent to Feed Forward Neural Network with sigmoid activation. The decision is made by a sigmoid layer called the “forget gate layer.” As it will tell how much we can forget or retain information from the previous cell state to the next cell state.



$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \ + \ b_i\right)$$

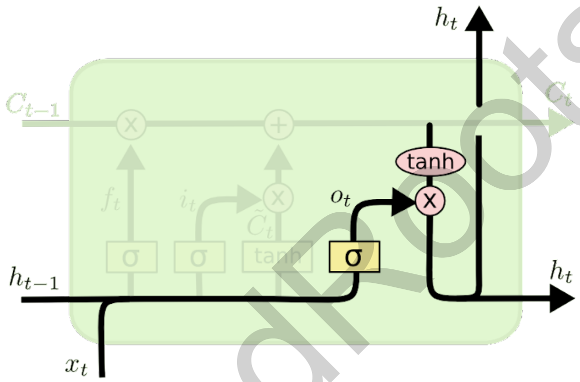
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \ + \ b_C)$$

The next layer is called the “input gate layer” as it will decide what new information we’re going to store in the cell state. This has two parts. First, a sigmoid layer decides which values we’ll update. Next, a tanh layer creates a vector of new candidate values. We will do element-wise multiplication to output from these two parts.



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

While calculating present cell state value we consider values from forget gate and input gate . Forget gates can forget or retain information from the previous cell state to the next cell state and the input gate will decide what new information we’re going to store in the cell state based on present state inputs.



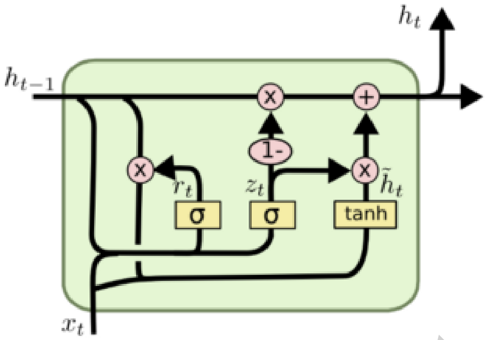
$$o_t = \sigma\left(W_o \left[h_{t-1}, x_t\right] + b_o\right)$$

$$h_t = o_t * \tanh\left(C_t\right)$$

The final layer in LSTM is called the “output gate layer”. This output will be based on our Current cell state , Previous output and input (X_t) . We will send the current cell state through Tanh to push the values to be between −1 and 1 and multiply it by the output of the sigmoid gate.

62.8 GRUs

LSTMs were created in 1997. And GRUs are modern and simplified versions created in 2014.
GRU stands for Gated Recurrent Unit.
Though this is a simplified version of LSTM, It is as powerful as LSTM and is faster to train.
GRUs have only 2 gates instead of 3 gates like in LSTM. The two gates in GRU are Reset gate and Update gate.



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$
$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$
$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$
$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

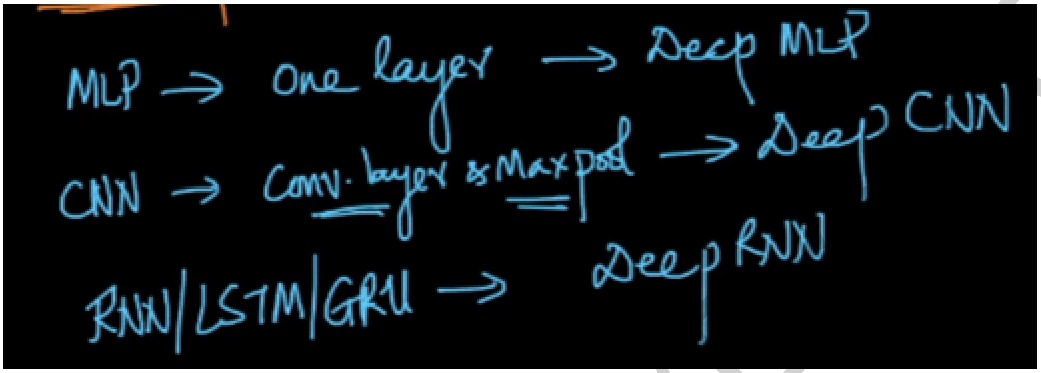
We have very fewer equations in GRU, which gives fewer partial derivatives and thus faster backpropagation.

Please go through the following blogs for better understanding about the GRUs:

Refer: https://www.slideshare.net/hytae/recent-progress-in-rnn-and-nlp-63762080
Refer: https://towardsdatascience.com/understanding-gru-networks-2ef37df6c9be

62.9 DEEP RNN

Till now we have discussed the simple RNN.Now we extend it to Deep RNN.
We have first discussed MLP and CNN and then extended it to deep MLP and Deep CNN. Similarly we extend the simple RNNs to Deep RNNs.

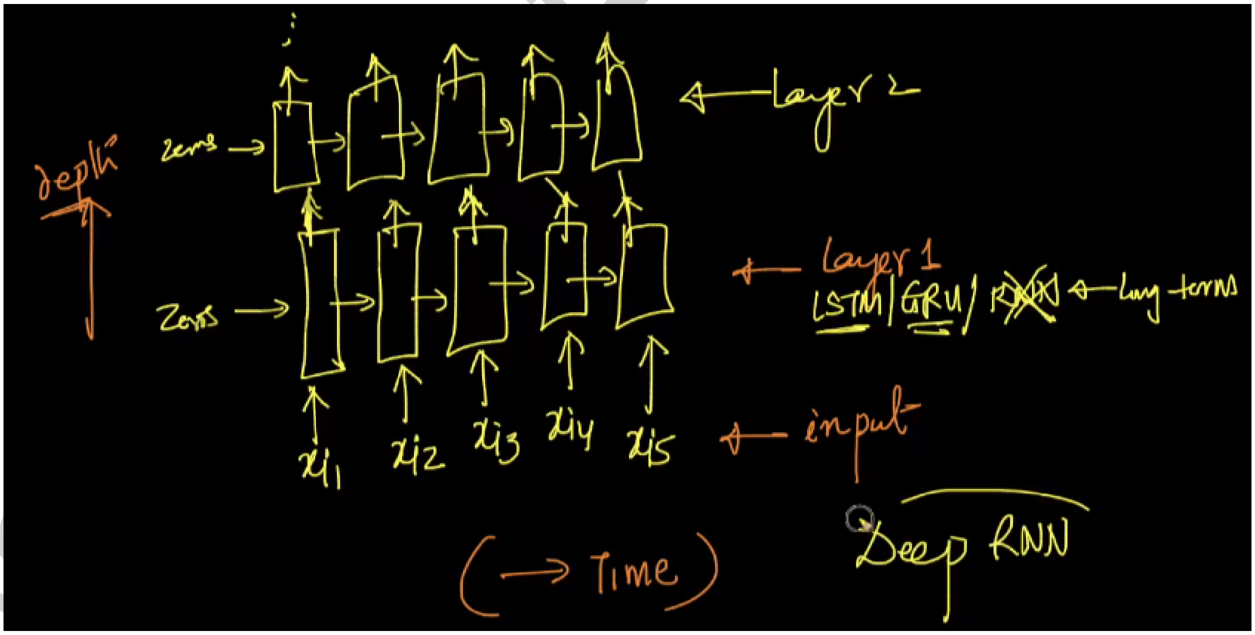


MLP → One layer → Deep MLP

CNN → Conv. layer & Max pool → Deep CNN

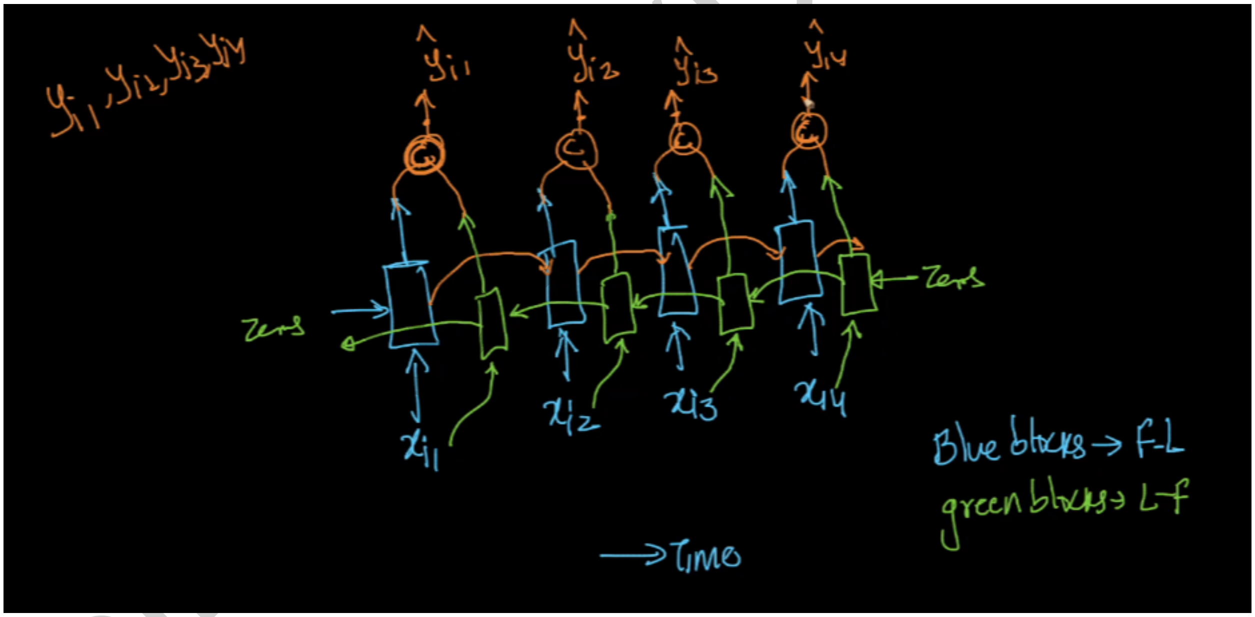
RNN/LSTM/GRU → Deep RNN

DEEP RNN:
We can have a number of layers on top of the other.



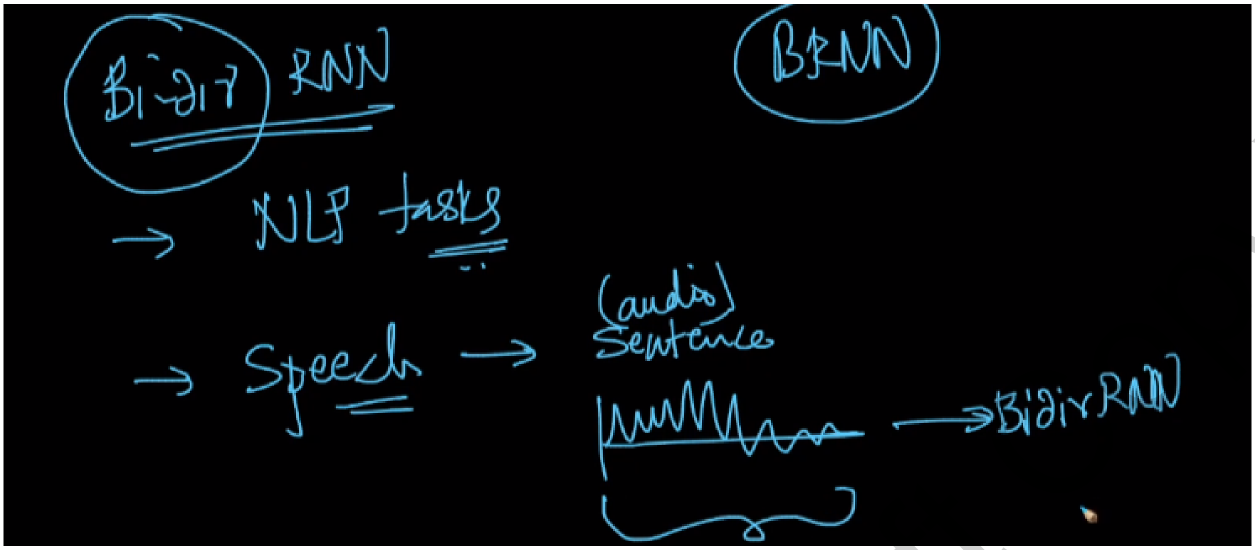
62.10 Bidirectional RNN

This is another special type of RNN called Bidirectional RNN.
These are used for certain types of NLP tasks.
Example:
Let us take words x_{11} x_{12} x_{13} x_{14} x_{15} and the output is
y_{11} y_{12} y_{13} y_{14} y_{15}
Here if we consider y_{13}, in regular RNNs like LSTM/GRU, y_{13} is dependent on previous outputs. But what if in the sentence, a word is connected more to the word which occurs in future. In unidirectional RNN, the sequence is being processed from the first word to the last word. So we lost the information of the future word dependencies.
So, we came up with the idea of Bidirectional RNN.



As shown the figure,
y_{i3} is getting impacted by x_{i1},x_{i2} and x_{i3} through the forward connection and by x_{i4} through the backward connection.
So, this way, we are able to consider the information of all words in the sentence for more effective predictions.

Applications of Bidirectional RNN:



Bi-dir RNN

BRNN

→ NLP tasks

→ Speech → (audio) Sentence

→ Bidir RNN

62.11 Code example : IMDB Sentiment classification

As the topic 62.11 is associated with the code walkthroughs (or) code discussions, we are not providing any notes for them, For any queries, please feel free to post them in the comments section (or) you can mail us at mentors.diploma@appliedroots.com

62.12 Code Walkthrough: Time Series forecasting using LSTMs/GRUs

As the topic 62.12 is associated with the code walkthroughs (or) code discussions, we are not providing any notes for them, For any queries, please feel free to post them in the comments section (or) you can mail us at mentors.diploma@appliedroots.com