REVA UNIVERSITY

Bengaluru, India

SCHOOL OF COMPUTING AND INFORMATION TECHNOLOGY

Module 3

Recurrent Neural Networks (RNN)

Dr. Sindhu Menon

Professor and HoD (AI-ML)

REVA University

www.reva.edu.in

Syllabus

Unit 3: Recurrent Neural Networks (RNN)

- LSTM, GRU, Deep RNN, Sequence to Sequence Learning for Machine Translation, The Transformer Architecture, BERT Model, Data Set for Pretraining BERT, Pretraining BERT, Encoder-Decoder Architecture, Multilayer Perceptrons,
- Case study: RNN model implementation.
- (Text 1 Chapter 9,10.1 to 10.3, 10.6,11.7, 15.8-15.10)

LSTM Topics in Deep Learning

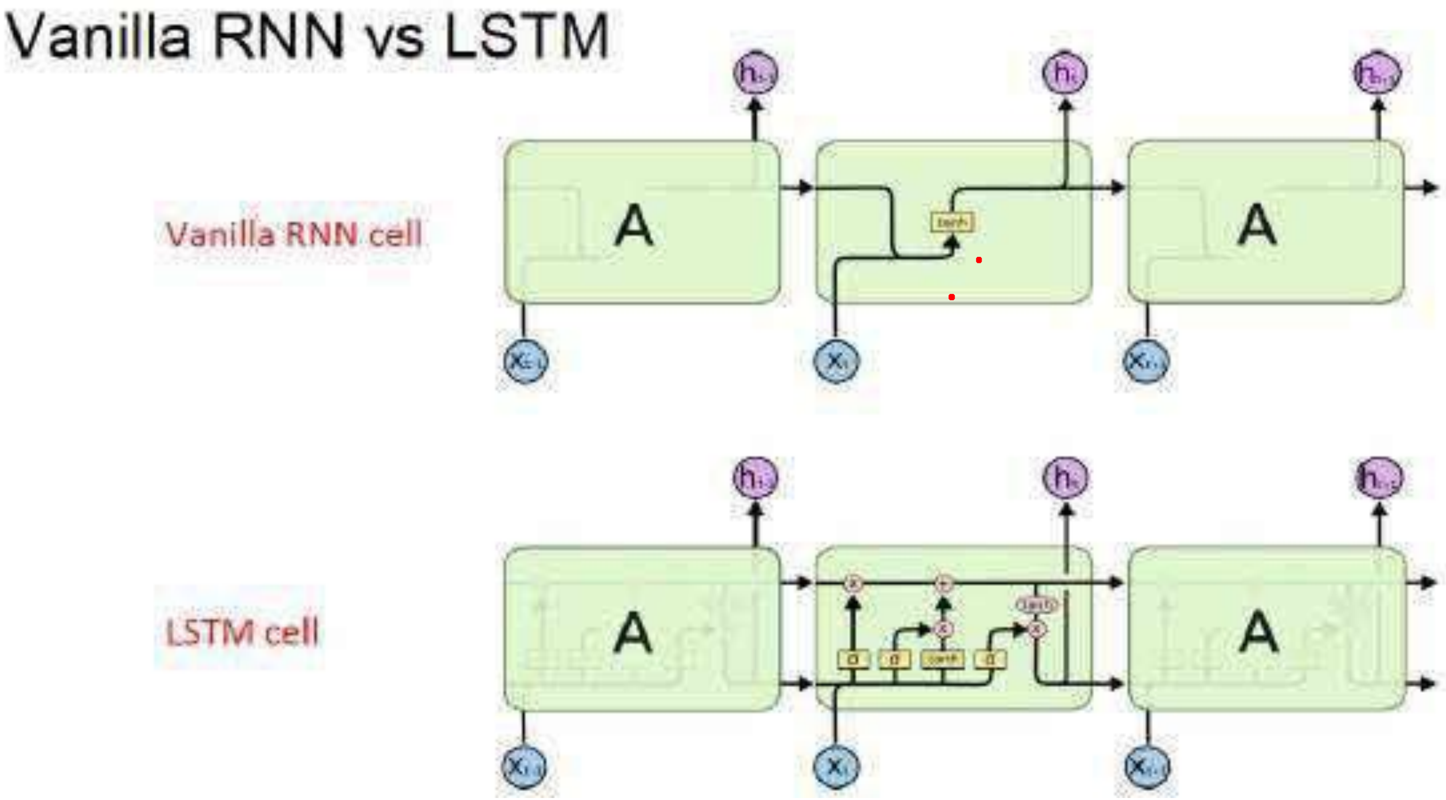
A variant of simple RNN (Vanilla RNN)

- Capable of learning long dependencies.
- Regulates information flow from recurrent units.

LSTM Topics in Deep Learning

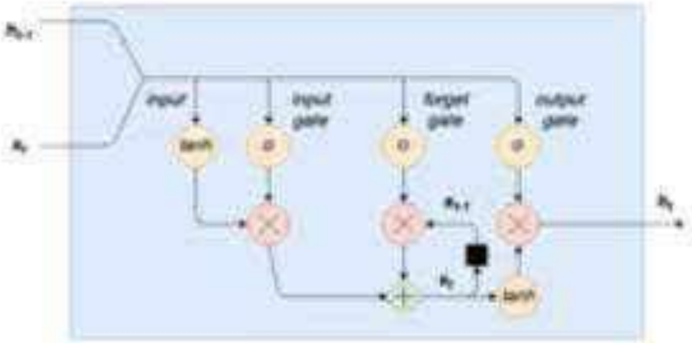
Moving from RNN to LSTM

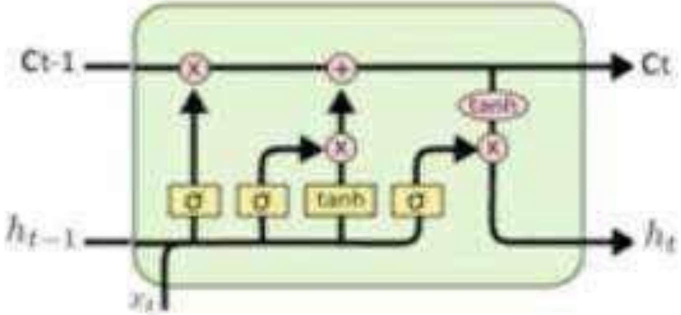
- All RNNs have the form of a chain of repeating modules of neural network.
- In standard RNNs, this repeating module will have a very simple structure, such as a single tanh layer.

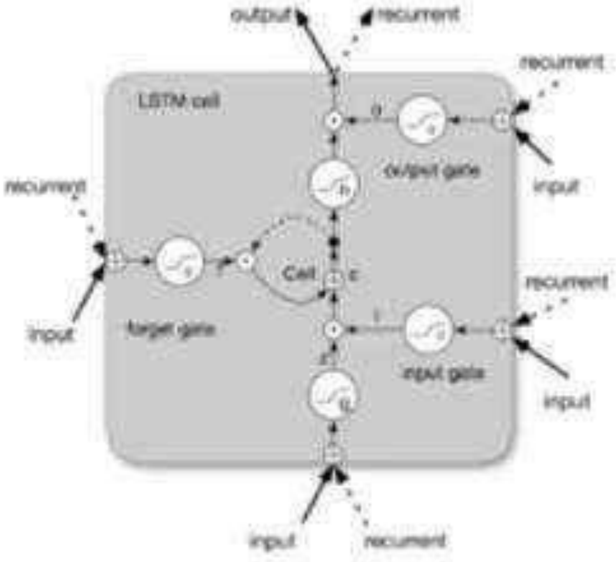


LSTM – Different representations

Same LSTM but…..



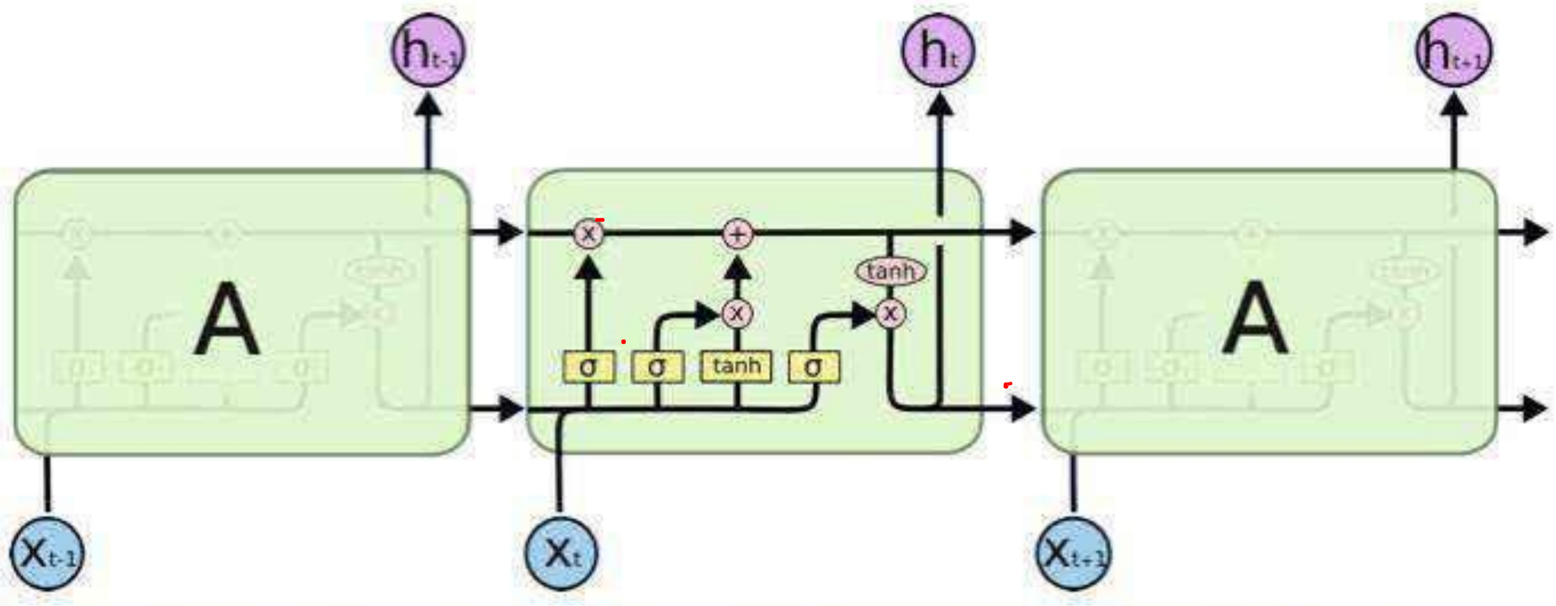




LSTM – Repeating module

Topics in Deep Learning

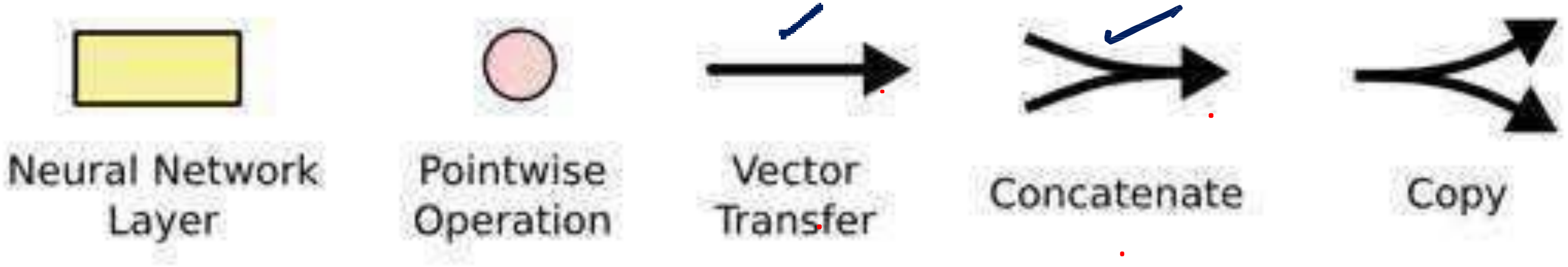
LSTMs also have this chain like structure, but the repeating module has a different structure.
Instead of having a single neural network layer, there are four, interacting in a very special way.



The repeating module in an LSTM contains four interacting layers.

LSTM-Notations

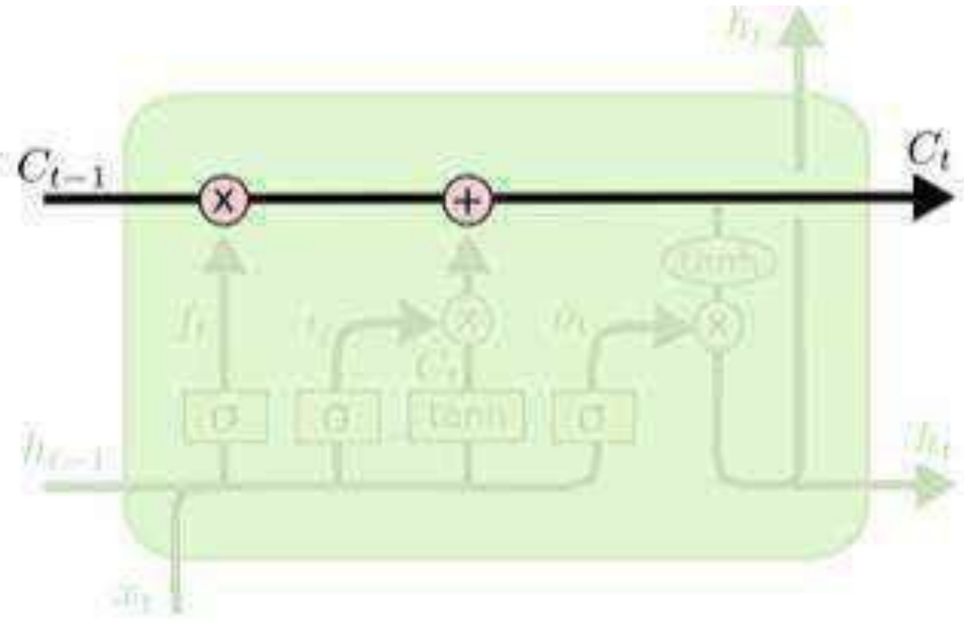
Topics in Deep Learning



In the above diagram, each line carries an entire vector, from the output of one node to the inputs of others.
The pink circles represent pointwise operations, like vector addition, while the yellow boxes are learned neural network layers.
Lines merging denote concatenation, while a line forking denote its content being copied and the copies going to different locations.

The Core Idea Behind LSTMs

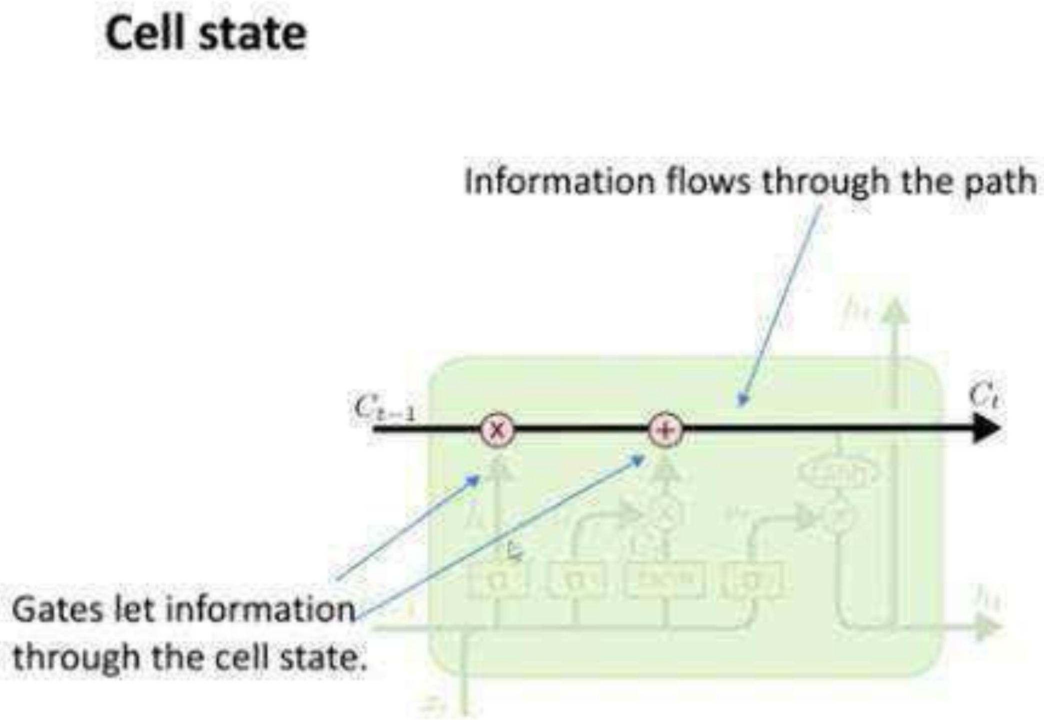
The key to LSTMs is the cell state, the horizontal line running through the top of the diagram.
The cell state is kind of like a conveyor belt.
It runs straight down the entire chain, with only some minor linear interactions.
It's very easy for information to just flow along it unchanged.



LSTM Cell -

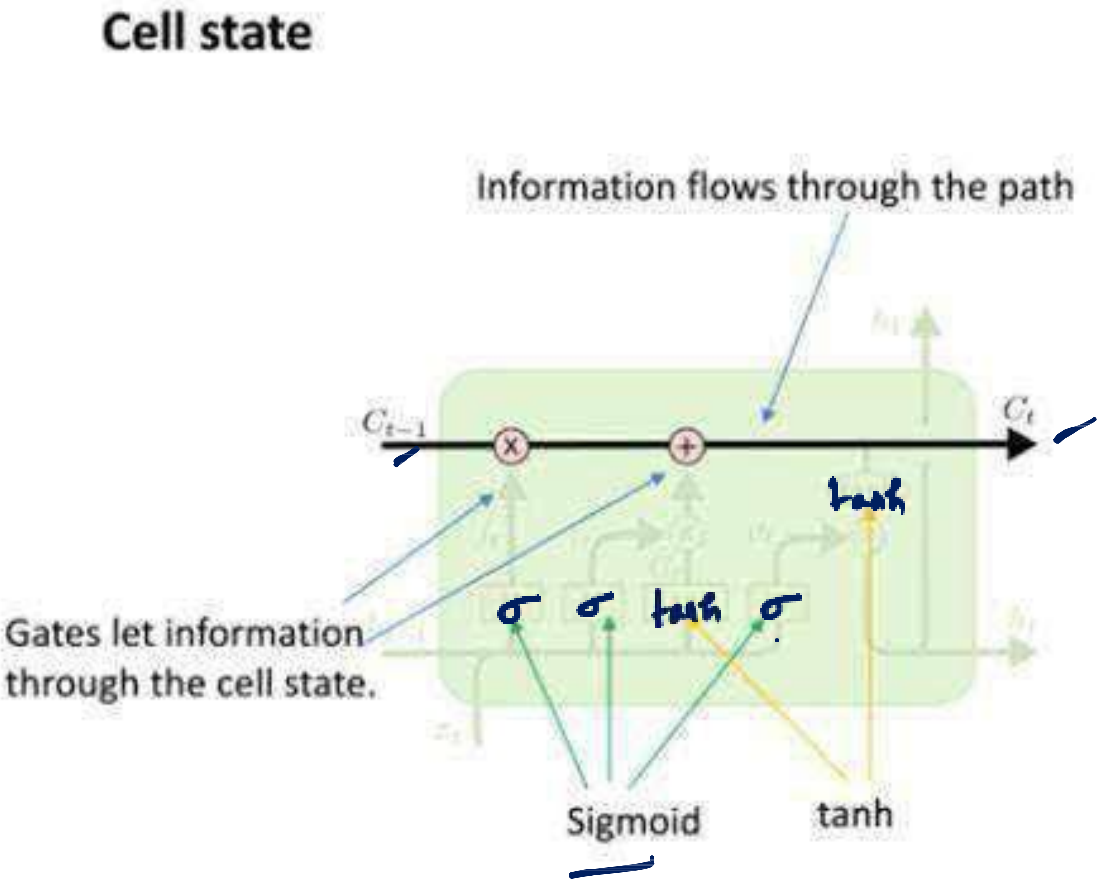
- The LSTM does have the ability to remove or add information to the cell state, carefully regulated by structures called gates.

- Gates are a way to optionally let information through. They are composed out of a sigmoid neural net layer and a pointwise multiplication operation.



Cell state is the long time memory(Conveyor belt)

LSTM – Activation Functions



- Sigmoid can output 0 to 1, it can be used to forget or remember the information.
- Tanh- to overcome the vanishing gradient problem

LSTM Gates

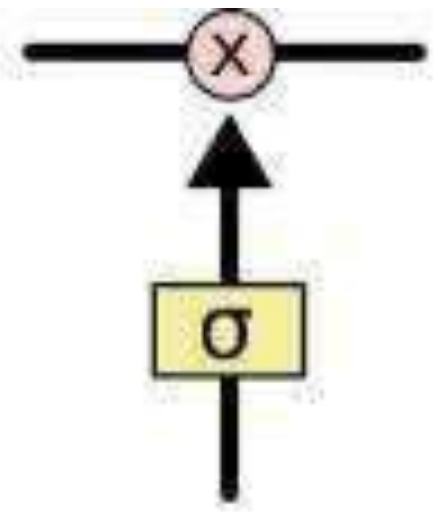
Gates are a way to optionally let information through.

They are composed out of a sigmoid neural net layer and a pointwise multiplication operation.

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!”

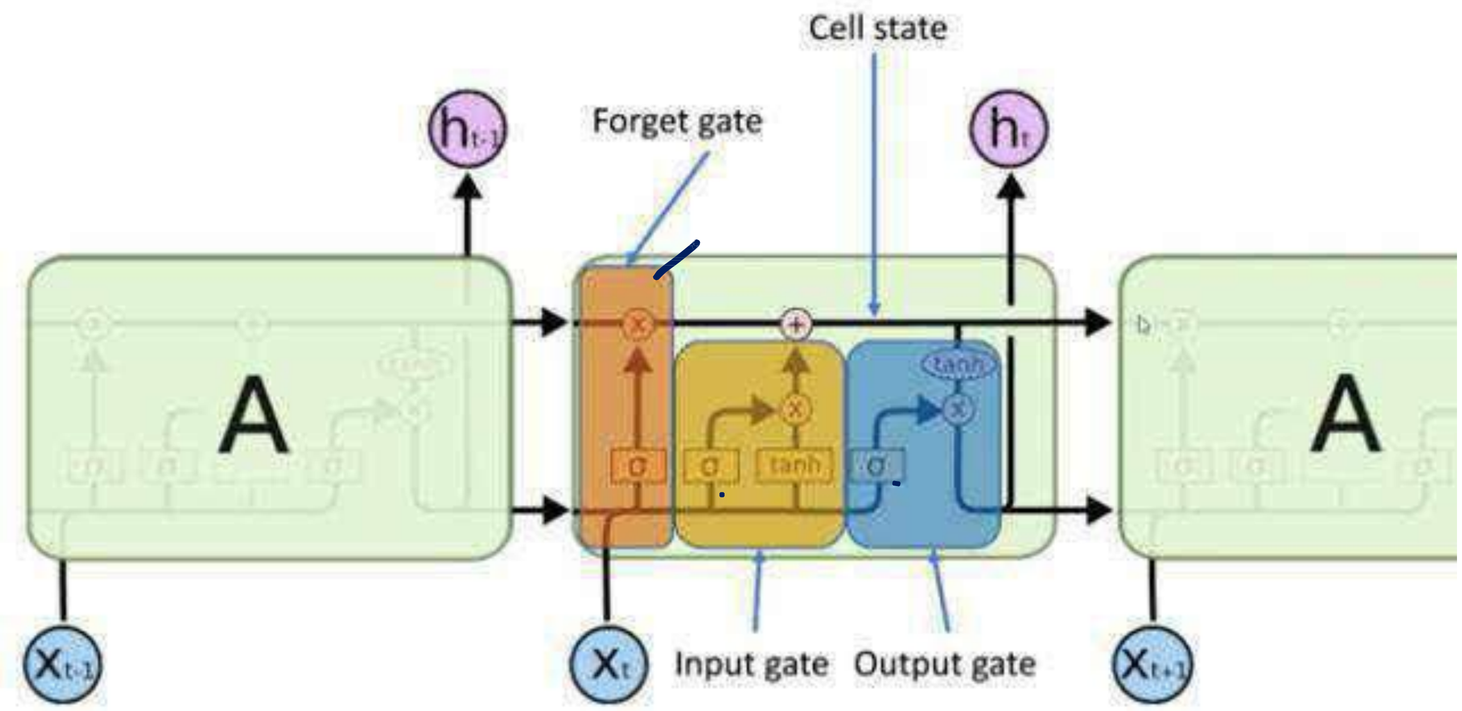
An LSTM has three of these gates, to protect and control the cell state.



LSTM-Gates

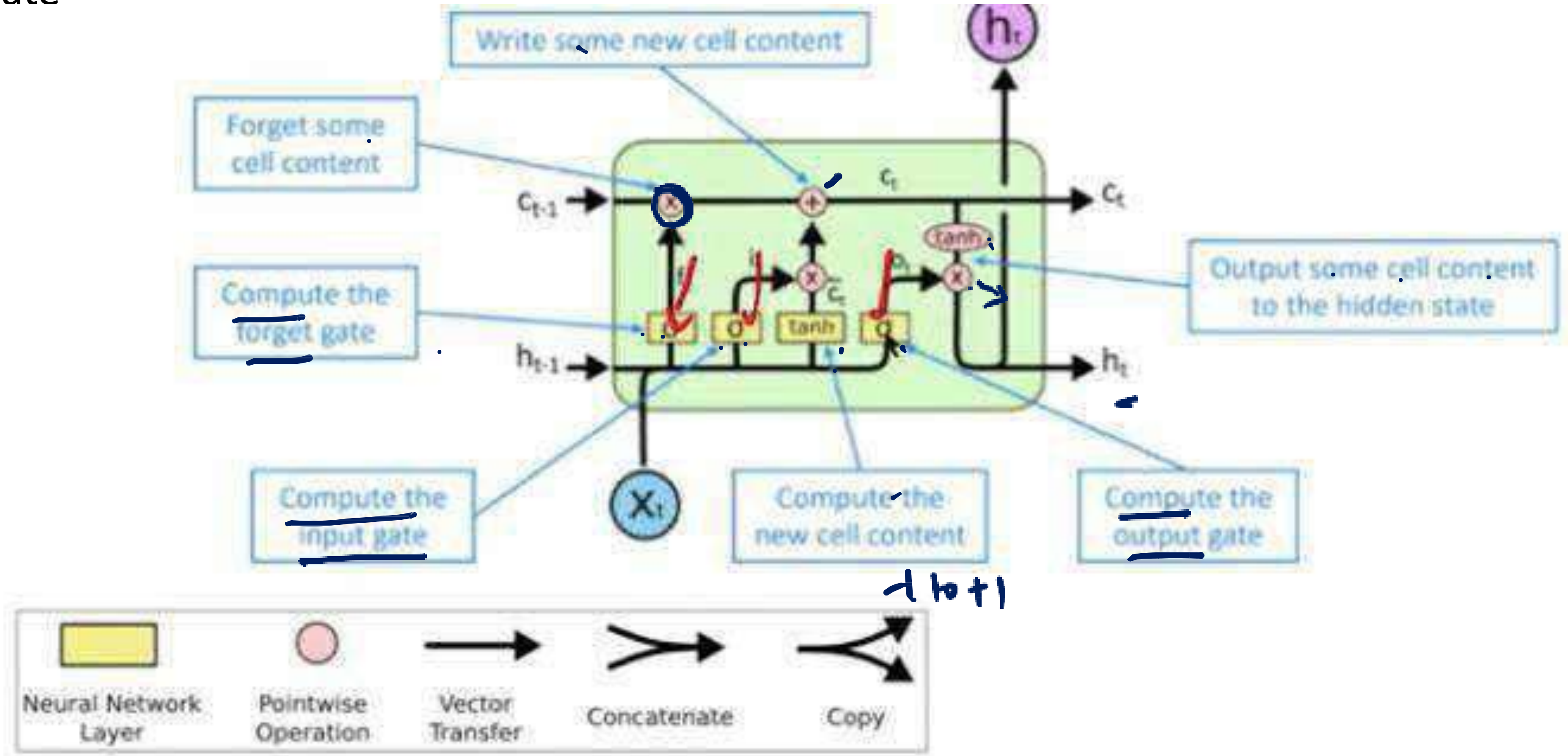
Topics in Deep Learning

1. Forget Gate
2. Input Gate
3. Output Gate



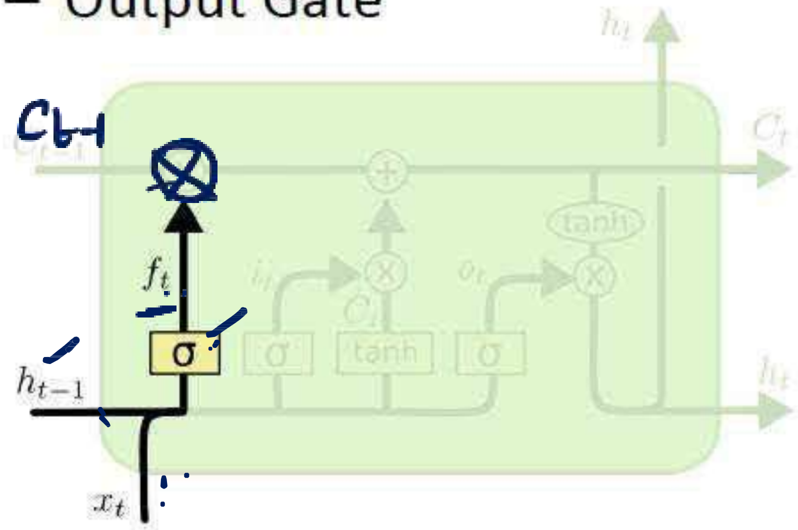
LSTM- Gates

1. Forget Gate
2. Input Gate
3. Output Gate



LSTM- Gates

- Each LSTM unit comprises of three gates.
  - Forget Gate: Amount of memory it should forget.
  - Input Gate
  - Output Gate



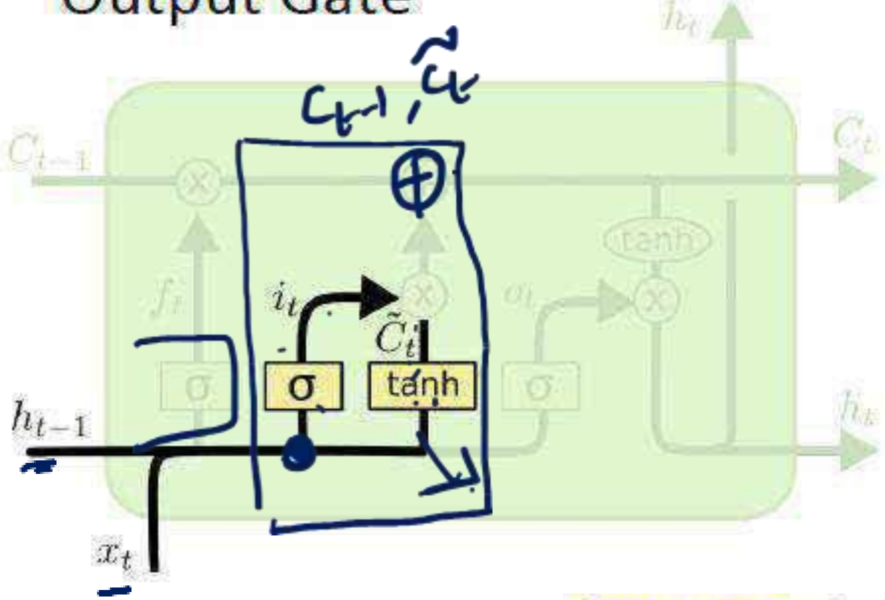
$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] + b_f\right)$$

Concatenating.

1.Composed of a sigmoid neural net layer and a point wise multiplication operation
2.Sigmoid layer outputs numbers between 0 to 1, describing how much information should be let through

LSTM- Gates

- Each LSTM unit comprises of three gates.
  - Forget Gate
  - Input Gate: Amount of new information it should memorize.
  - Output Gate



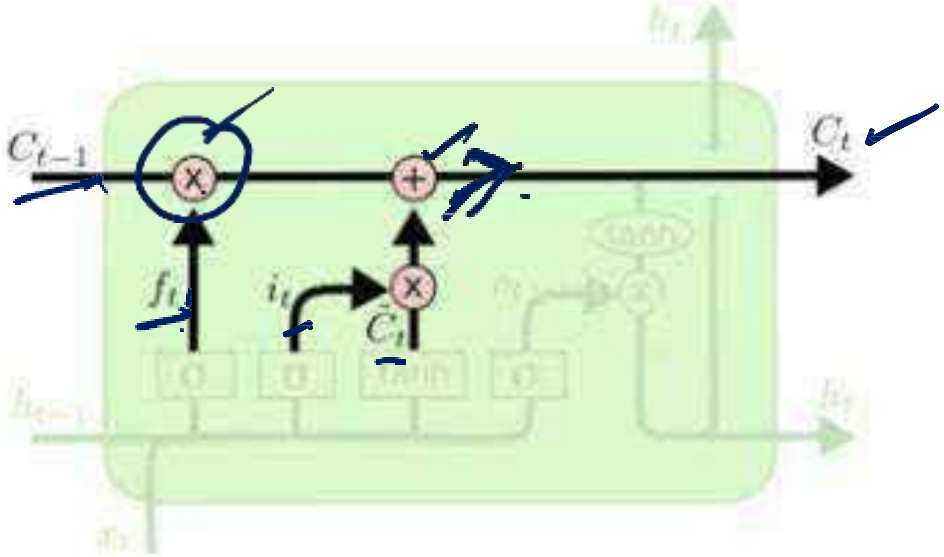
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] + b_i\right)$$

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

Input gate: decides what information to throw away from the cell state
Cell content: new content to be written to cell $\tilde{C}_t$

LSTM- Gates

Cell state: erase ("forget") some content from last cell state, and write ("input") some new cell content

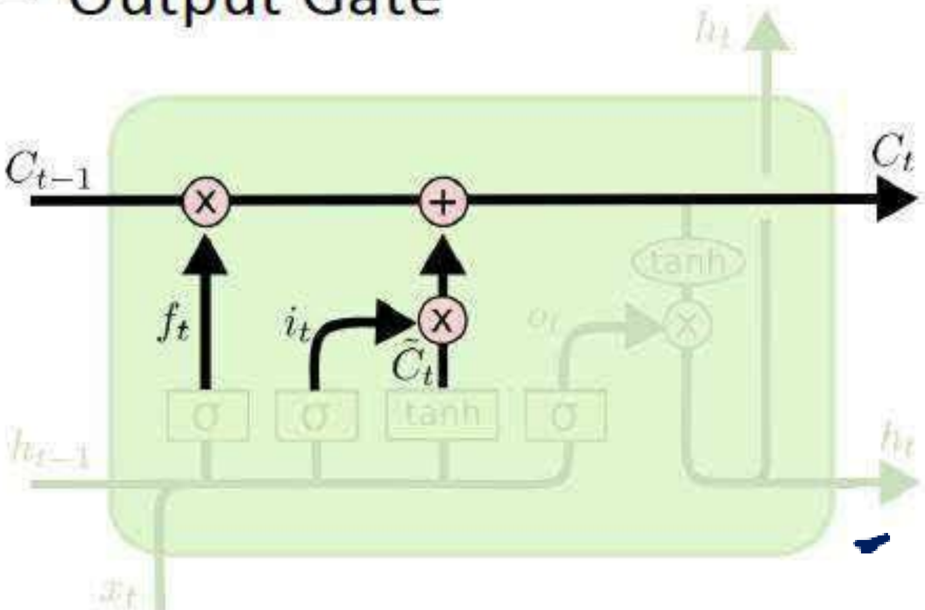


$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Input gate: decides what information to throw away from the cell state
Cell content: new content to be written to cell

LSTM- Gates

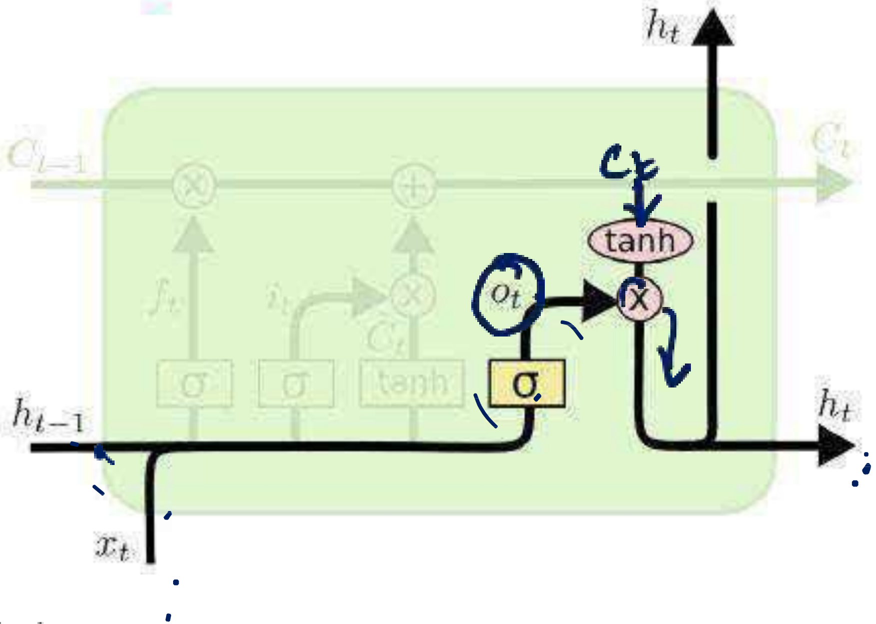
- Each LSTM unit comprises of three gates.
  - Forget Gate: Amount of memory it should forget.
  - Input Gate: Amount of new information it should memorize.
  - Output Gate



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

LSTM- Gates

- Each LSTM unit comprises of three gates.
  - Forget Gate
  - Input Gate
  - Output Gate: Amount of information it should pass to next unit.

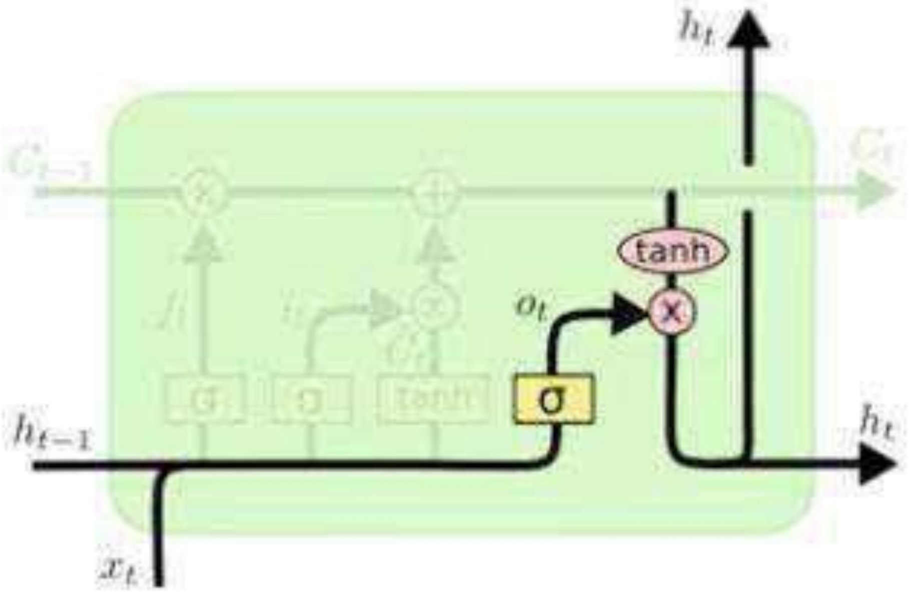


$$o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right)$$

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

LSTM- Gates

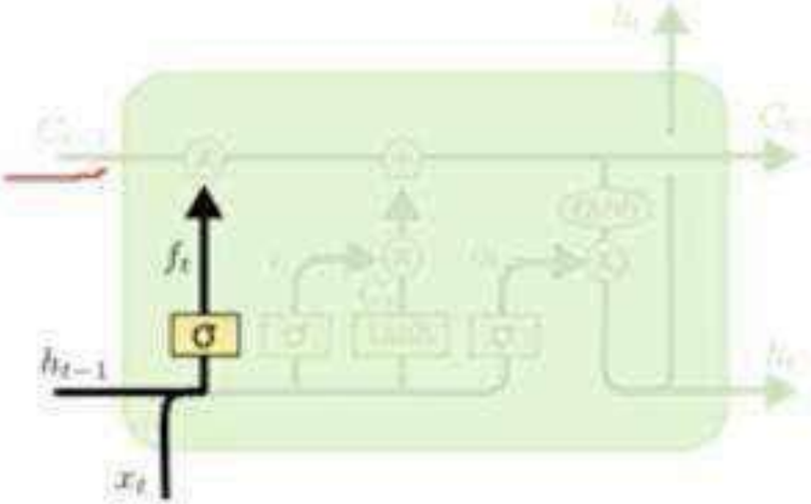
Output gate: controls what parts of cell are output to hidden state
Hidden state: read ("output") some content from cell



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

LSTM- Step-by-Step LSTM Walk Through

- The first step in our LSTM is to decide what information we're going to throw away from the cell state.
- This decision is made by a sigmoid layer called the "forget gate layer." It looks at h_{t-1} and x_t, and outputs a number between 0 and 1 for each number in the cell state C_{t-1}.
- 1 represents "completely keep this" while a 0 represents "completely get rid of this."

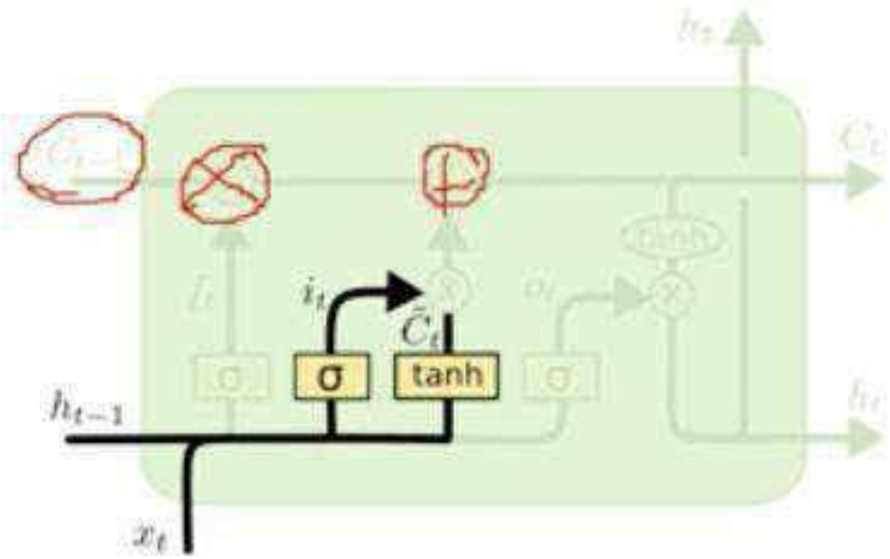


$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] + b_f\right)$$

LSTM- Input Gate(update)

Step-by-Step LSTM Walk Through

- Next, a tanh layer creates a vector of new candidate values, $\tilde{C}_t$, that could be added to the state. In the next step, we'll combine these two to create an update to the state.

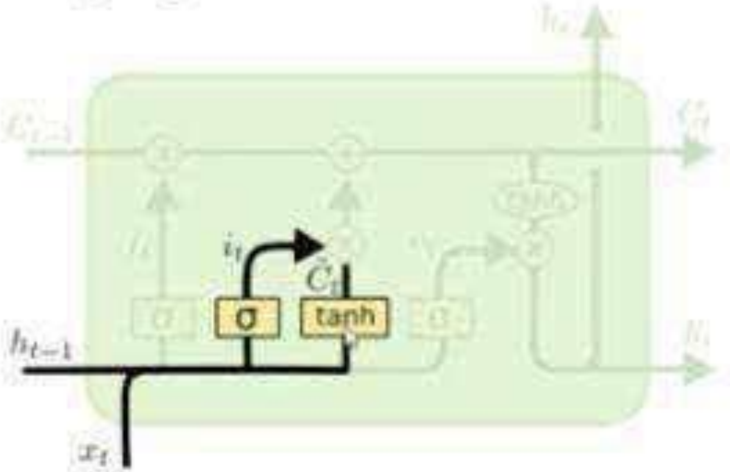


$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] + b_i\right)$$

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

LSTM- Input Gate(update)

Input gate: What new information will be stored in the cell state.



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

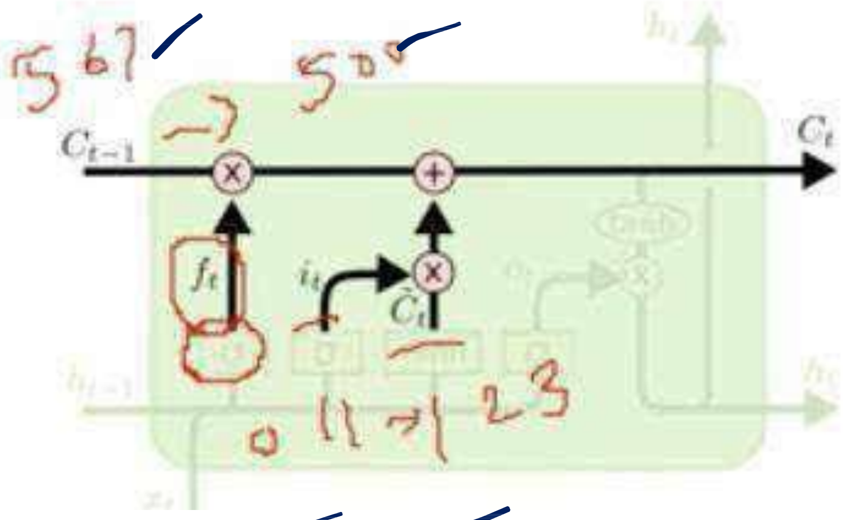
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \ + \ b_C)$$

Sigmoid layer decides which values are updated.

tanh layer gives weights to the values to be added to the state.

LSTM- Input Gate

- It's now time to update the old cell state, C_{t-1}, into the new cell state C_t. The previous steps already decided what to do, we just need to actually do it.
- We multiply the old state by f_t, forgetting the things we decided to forget earlier. Then we add $i_t * \tilde{C}_t$. This is the new candidate values, scaled by how much we decided to update each state value.

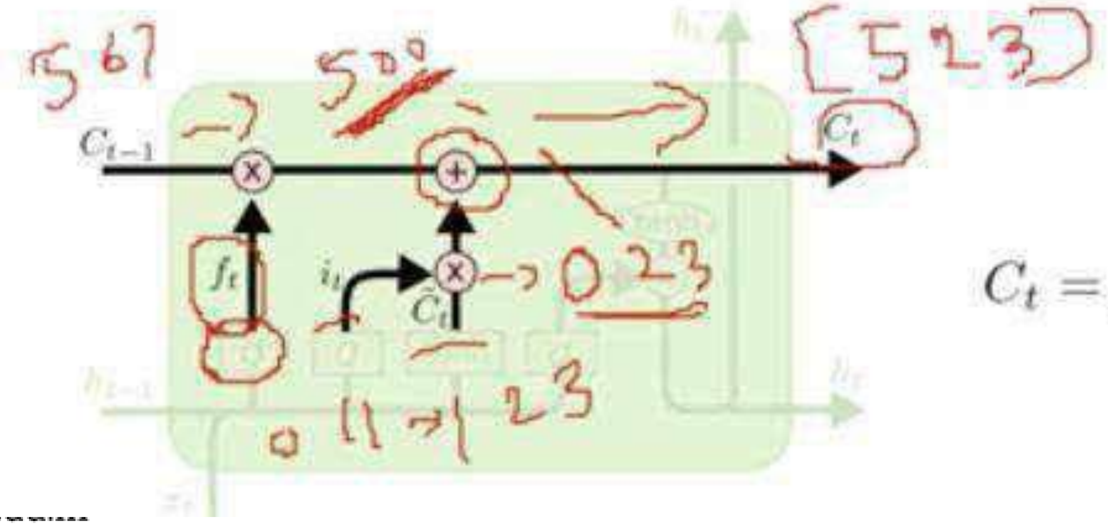


$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

LSTM

Step-by-Step LSTM Walk Through

- It's now time to update the old cell state, C_{t-1}, into the new cell state C_t. The previous steps already decided what to do, we just need to actually do it.
- We multiply the old state by f_t, forgetting the things we decided to forget earlier. Then we add $i_t * \tilde{C}_t$. This is the new candidate values, scaled by how much we decided to update each state value.



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

Ct – long term memory

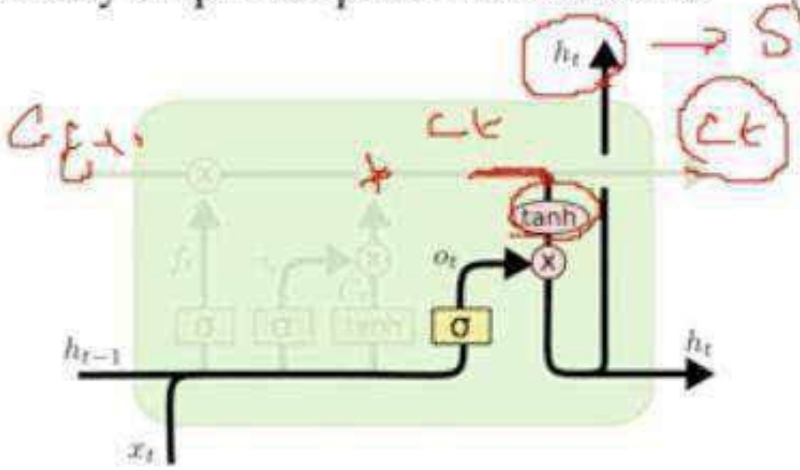
LSTM- output gate

Topics in Deep Learning

Ct – long term memory , ht = short term memory

Step-by-Step LSTM Walk Through

- Finally, we need to decide what we're going to output. This output will be based on our cell state, but will be a filtered version.
- First, we run a sigmoid layer which decides what parts of the cell state we're going to output. Then, we put the cell state through **tanh** (to push the values to be between −1 and 1) and multiply it by the output of the sigmoid gate, so that we only output the parts we decided to.



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

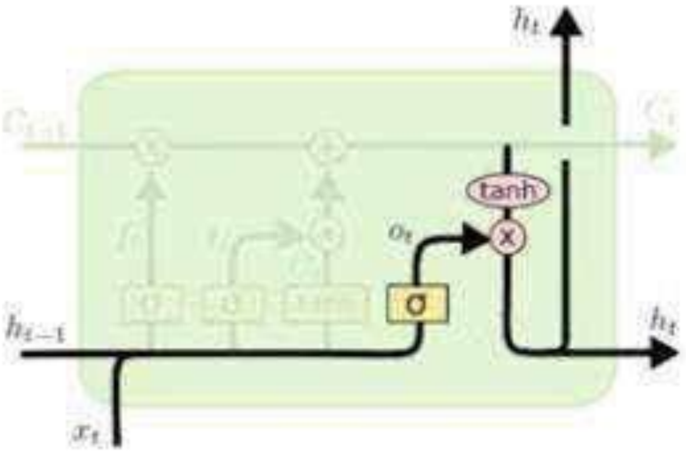
$$h_t = o_t * \tanh\left(C_t\right)$$

LSTM-output gate

Topics in Deep Learning

Ct – long term memory , ht = short term memory

Output gate: Decide what part of current cell makes to the output



Sigmoid layer decides which part of cell state is selected for output.

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

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

tanh layer gives weights to the values (-1 to 1).

LSTM- Summary

Sigmoid function: all gate values are between 0 and 1

Forget gate: controls what is kept vs forgotten, from previous cell state

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

Input gate: controls what parts of the new cell content are written to cell

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

Output gate: controls what parts of cell are output to hidden state

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

New cell content: this is the new content to be written to the cell

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

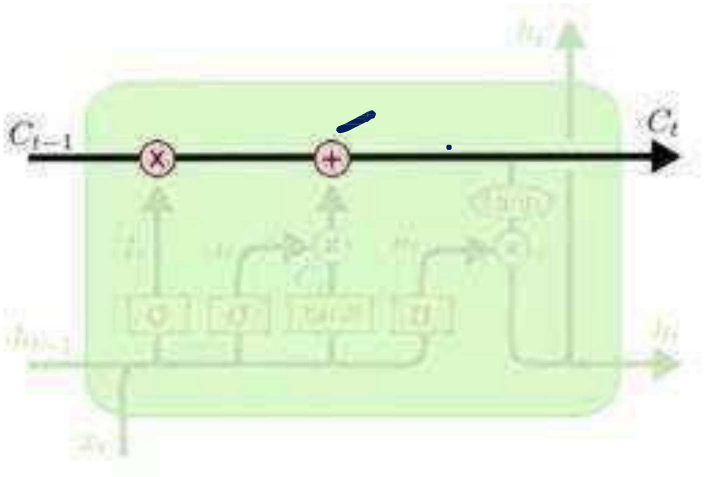
Cell state: erase ("forget") some content from last cell state, and write ("input") some new cell content

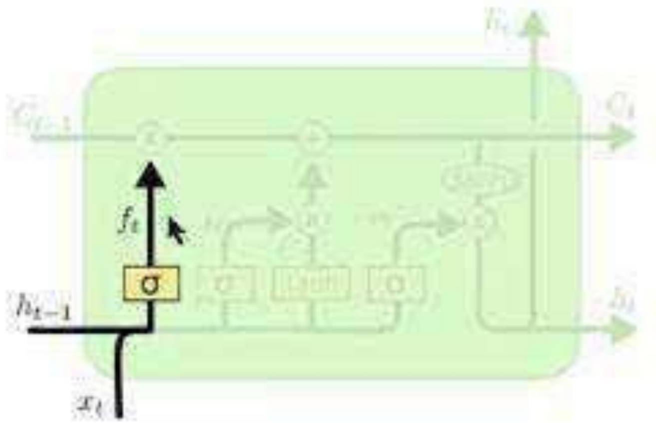
$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Hidden state: read ("output") some content from the cell

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

LSTM- How does it solve the vanishing gradient problem?

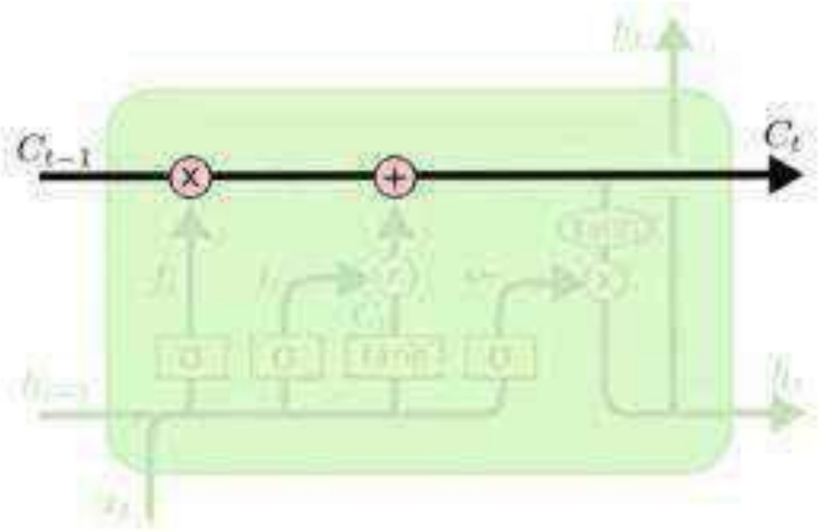


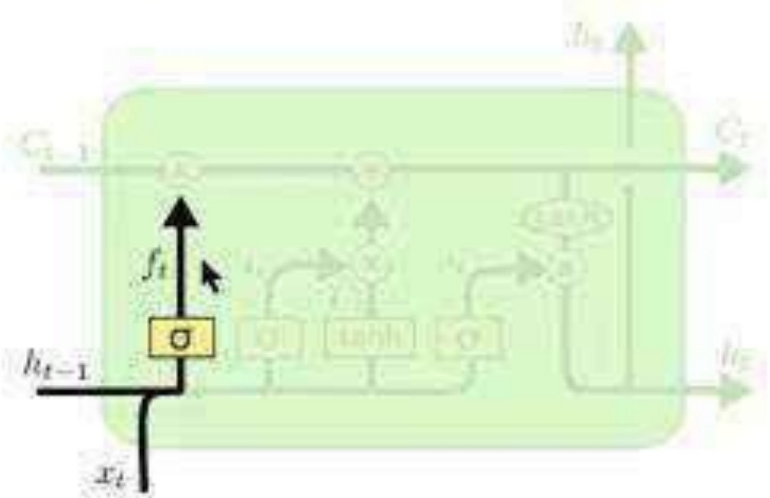


- Gradient "highway"

LSTM- How does it solve the vanishing gradient problem?

Topics in Deep Learning





- Gradient “highway”
- There is no neural network layer between ct-1 to ct and only thing that exists between is sigmoid and the job is “how much information is to retain?
- Forget gate is part of the design, it reduces the gradient where it should.

LSTM- Advantages

- Non-decaying error backpropagation.
- For long time lag problems, LSTM can handle noise and continuous values.
- No parameter fine tuning.
- Memory for long time periods

- LSTM - solves the vanishing gradient and the long memory limitation problem
- LSTM can learn sequences with more than 1000 time steps.

Gated Recurrent Unit, or GRU

- As we have seen in the LSTM, GRU also utilizing the gating mechanisms (Remember the gates used in LSTM – Forget, Input and Output) to manage and as well to control the flow of the information between the cells in the neural network.

- Can we understand the differences between LSTM and GRU in the architectural perspective as well as the functioning?

- Number of gates in LSTM – 3
- Number of gates in GRU – 2
- GRUs do not have the Cell State and Hidden states are used!

- The gates are named as Reset Gate and Update Gate.

Gated Recurrent Unit, or GRU

Update gate: controls what parts of hidden state are updated vs preserved

$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

Reset gate: controls what parts of previous hidden state are used to compute new content

$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

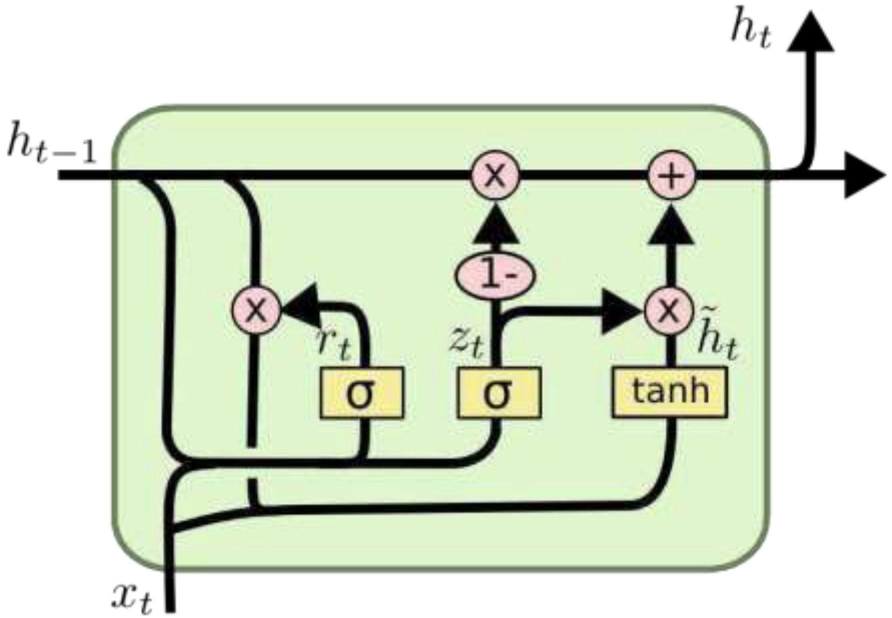
New hidden state content: reset gate selects useful parts of prev hidden state. Use this and current input to compute new hidden content.

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

Hidden state: update gate simultaneously controls what is kept from previous hidden state, and what is updated to new hidden state content

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

Gated Recurrent Unit, or GRU



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

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

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

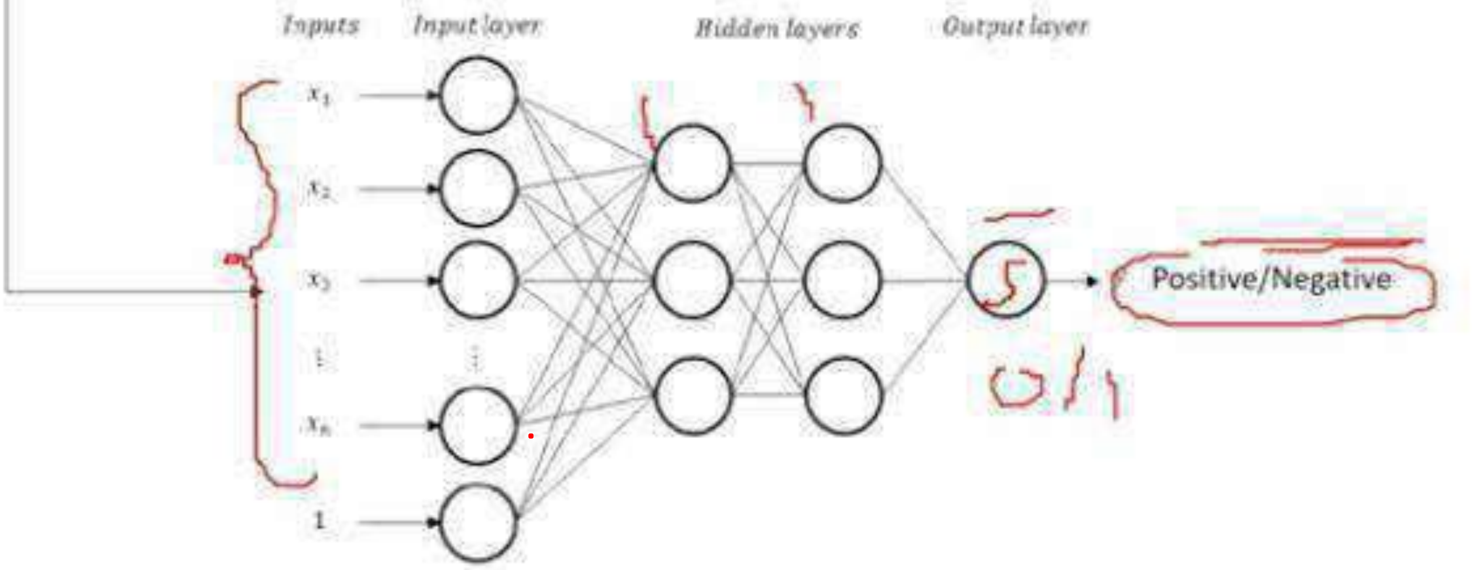
GRU Vs LSTMs : Summary

- Input and forget gates of LSTMs are coupled by an update gate in GRUs; reset gate (GRUs) is applied directly to previous hidden state
- GRU has two gates, an LSTM has three gates; what does this tell you? **Lesser parameters to learn!**
- In GRUs:
  - No internal memory (c_t) different from exposed hidden state
  - No output gate as in LSTMs
- LSTM a good default choice (especially if data has long-range dependencies, or if training data is large); Switch to GRUs for speed and fewer parameters

Why RNN?

Applying ANN

	battery	because	charging	do	good	I	is	like	not	okay	phone	properly	this
Doc 1	0	0	0	0	0	1	0	1	0	1	0	0	1
Doc 2	0	0	0	0	1	0	1	0	0	0	1	0	1
Doc 3	0	0	0	0	0	0	1	0	1	1	1	0	1
Doc 4	0	0	0	0	0	0	0	0	0	0	0	0	0



Why RNN?

Drawback of BoW

- Lets try to represent the following text using Bag-of-Words
  - This phone is no good - Negative
  - No this phone is good - Positive

	good	is	no	phone	this
Doc 1	1	1	1	1	1
Doc 2	1	1	1	1	1

- Feed Forward Neural Networks with Bag-of-words (BoW) model does not consider position of words in input !

Why RNN?

Example : 2, Word Guessing Game

- Guess the missing word in the follow sentence
  - I went to France last year, there the people speak the french ________ language
- Lets find the missing word with the sentence (alphabetically sorted words)
  - Sorted Text: food I is it like Thai so
  - Original Text: I like Thai food it is so ______
  - Answer: I like Thai food it is so ______
- Feed Forward Neural Network will fail for guessing missed words
- Conclusion: Sequence matters !

Example: audio, video, any time series data

Sequence Data

Other situations where sequence matters

1. Stock price data will be more or less similar to yesterday’s price
2. Tomorrow’s temperature will be close to today’s temperature

Examples of sequence data

Speech recognition	 →	"The quick brown fox jumped over the lazy dog."
Music generation	∅ →	
Sentiment classification	"There is nothing to like in this movie." →	★☆☆☆☆
DNA sequence analysis	AGCCCCTGTGAGGAACTAG →	AG**CCCCTGTGAGGAACT**AG
Machine translation	Voulez-vous chanter avec moi? →	Do you want to sing with me?
Video activity recognition	 →	Running
Name entity recognition	Yesterday, Harry Potter met Hermione Granger. →	Yesterday, **Harry Potter** met **Hermione Granger**.

Sequence Data

Sequence Application Variation

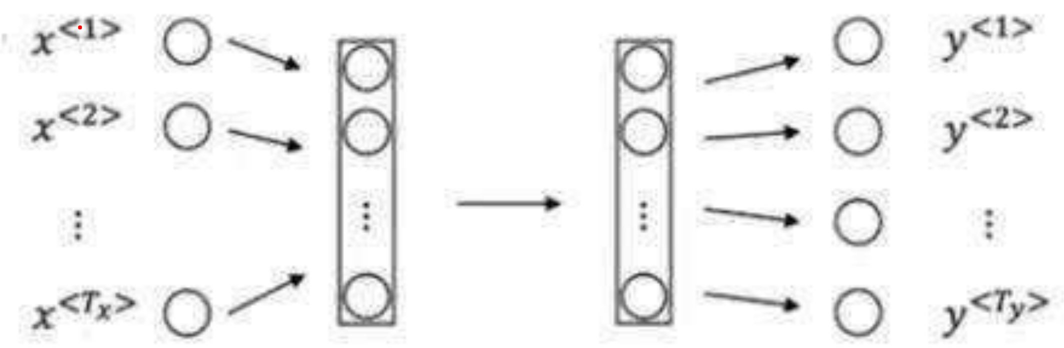
- Audio Signal to Sequence - Speech Recognition
- Nothing to Sequence or Single Parameter to Sequence - Music Generation
- Sequence to Single Output - Sentiment Classification
- Sequence to Sequence - Machine Translation
- Video Frame Sequence to Output - Activity Recognition
- Sub-Sequence from a Sequence - Finding Specific Protein from a DNA Sequence
- Outlining Specific parts of a sequence - Name Entity Recognition

Ex: Siri, Alexa

Ex: Google translator

Drawbacks of Standard Neural Network

Standard Neural Network Does not works out to give a good application for sequence models



Inputs, outputs can be different lengths in different examples.

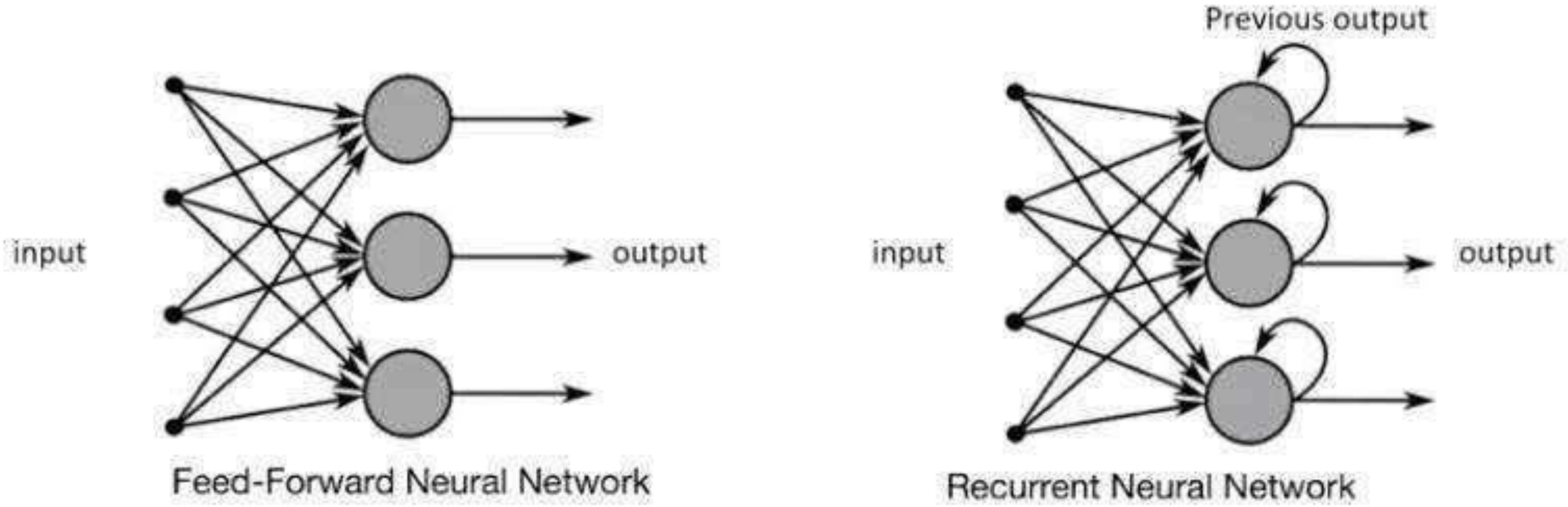
Doesn't share features learned across different positions of text.

1. Feed forward networks accept a fixed-sized vector as input and produce fixed-sized vector as output
2. So, feed forward networks cannot process sequential data containing variable length of data
3. Feed forward networks does not consider sequence in the data.

Solution for Sequence Analysis

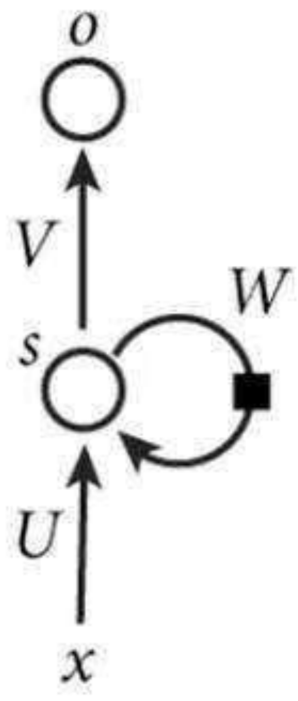
1. Recurrent Neural Network allows us to operate over sequence of vectors.
2. Recurrent, because previous output is also used with current input.
3. RNN also viewed as having a “memory”.
4. Unlike a traditional neural network, RNN shares same parameters across all steps.
5. Greatly reduce the total number of parameters we need to learn.
6. RNN is not feedforward network, as cycle is formed in hidden units.

NN Vs RNN


input
output
Feed-Forward Neural Network
Previous output
input
output
Recurrent Neural Network

RNN Notations

Notations



x : Input

o : Output

s : state of the hidden unit

U, V and W : Weights to be learned

U : weights used for hidden state computation (from input)

V : weights used for output computation

W : weights used for hidden state computation (from previous hidden state)

RNN Notations

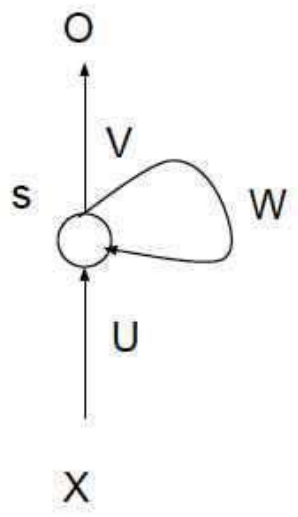
Recurrent Neural Network (RNN)

Basic definition:

A neural network with feedback connections.

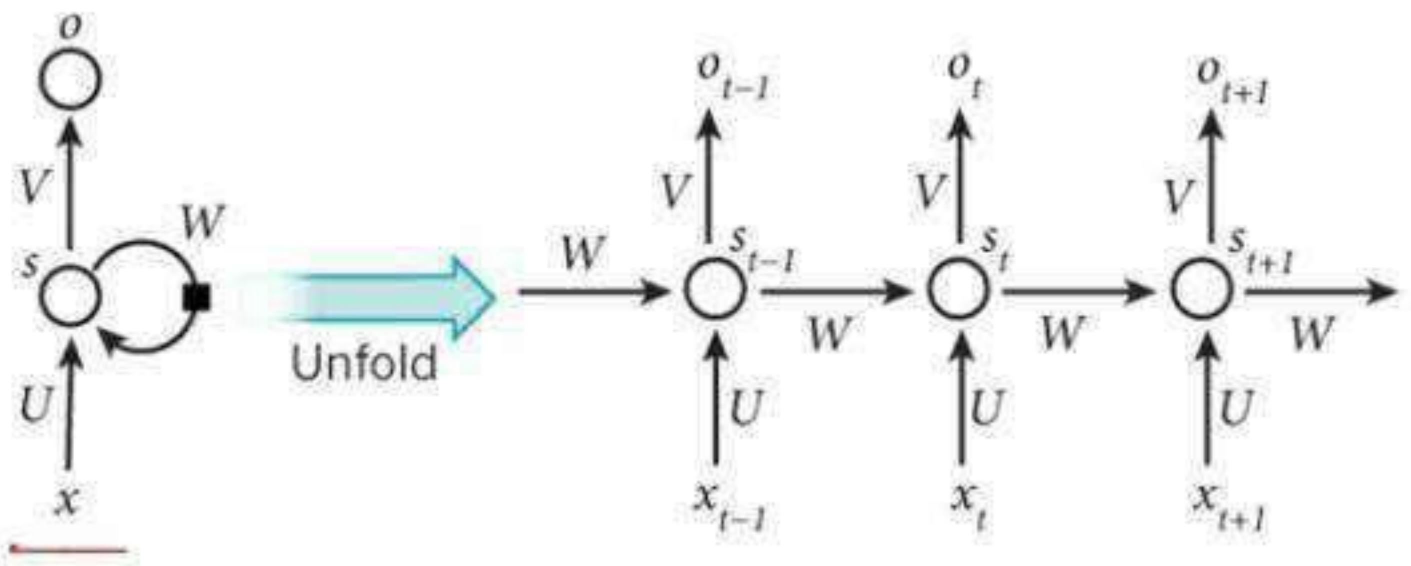
X: Input
O: Ouput
S: Hidden state

Weights: [U,V,W]
Learned during training



RNN Notations

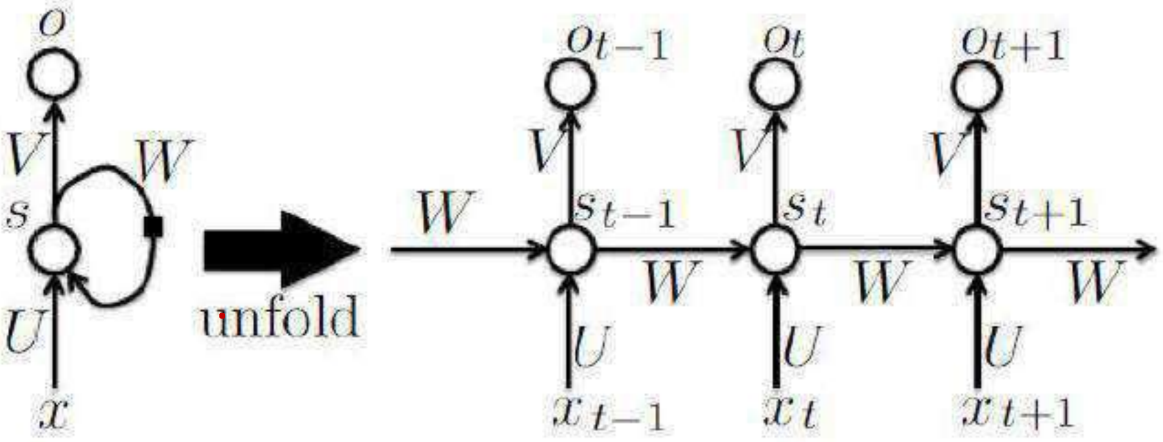
Unrolled RNN with parameters



The recurrent network can be converted into a feed forward network by **unfolding over time**

RNN

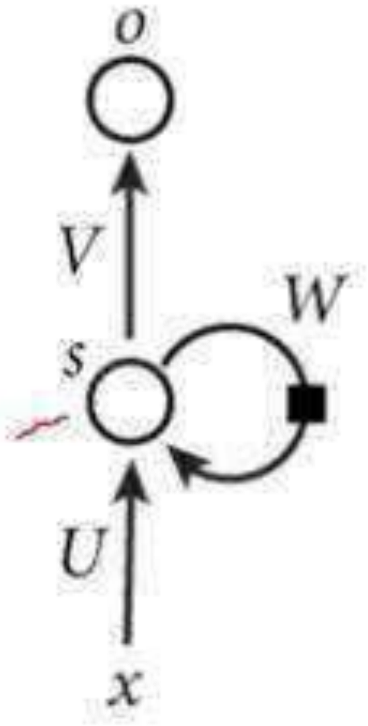
- Enable networks to do temporal processing
- Good at learning sequences
- Acts as memory unit



Memory

$$
\begin{aligned}
a_t &= b + W s_{t-1} + U x_t \\
s_t &= \tanh(a_t) \\
o_t &= c + V s_t \\
p_t &= \mathrm{softmax}(o_t)
\end{aligned}
$$

RNN Forward Pass

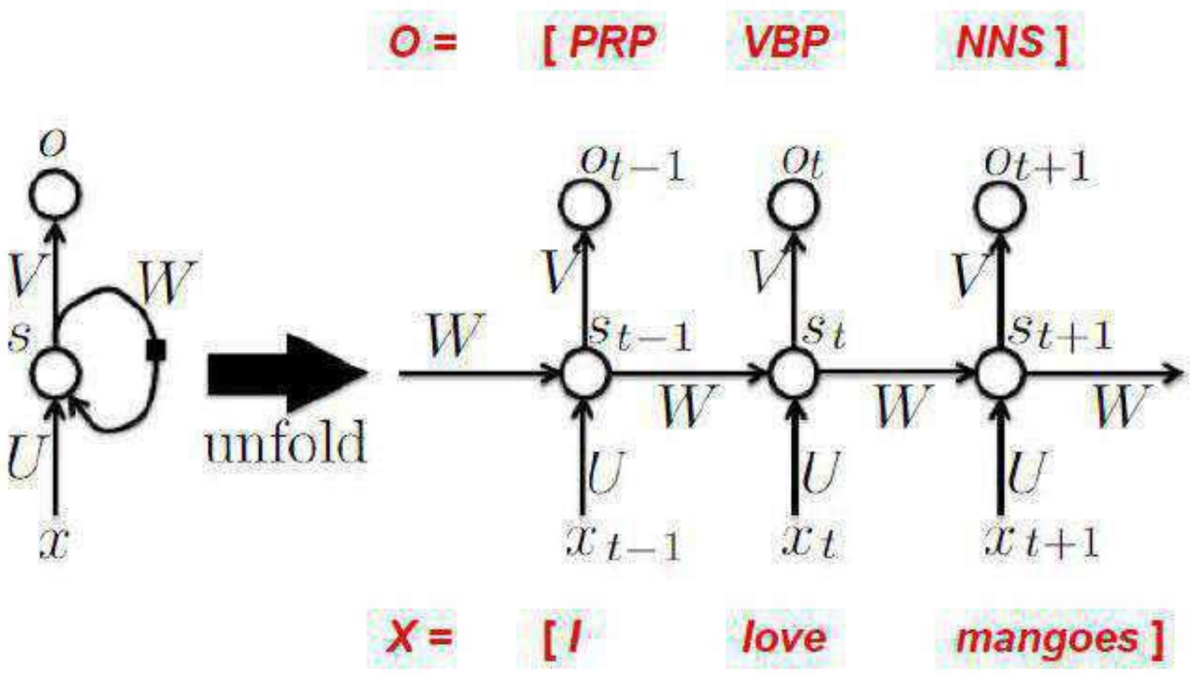


- Step 2: Current hidden state s at time t will be computed using
$$s(t) = f_h(Ux(t) + Ws(t-1))$$
- Step 1: input x will be given at time t
- Step 3: Current output o at time t will be computed using
$$o(t) = f_o(Vs(t))$$
- Note: Output will not necessarily be generated for every t. i.e. it depends upon application. Speech recognition RNN will output words instantly at every iteration. Opinion classification RNN will output label only at the end of sentence.

RNN Example-1

Part-of-speech tagging:

- Given a sentence X, tag each word its corresponding grammatical class.



PRP: personal pronoun, vbp:Verb, present tense, other than third person singular, nns:noun,plural

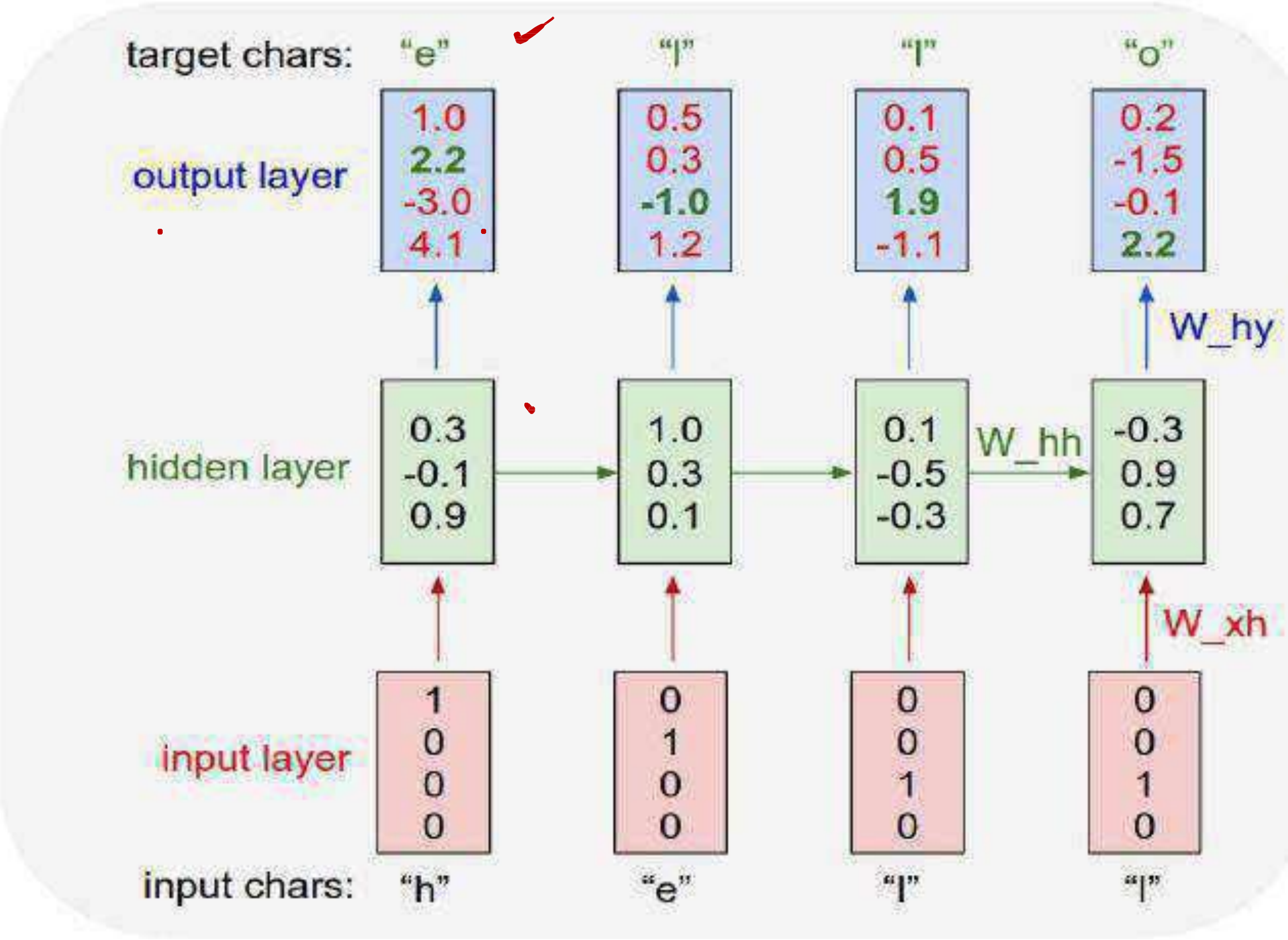
RNN Example-2

Character level language model:

- Given previous and current characters, predict the next character in the sequence.

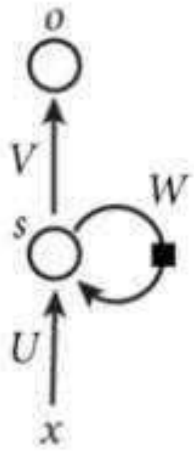
Let

- **Vocabulary: [h,e,l,o]**
- **One-hot representations**
  - h = [1 0 0 0]
  - e = [0 1 0 0]
  - l = [0 0 1 0]
  - o = [0 0 0 1]



RNN Forward Pass with Example

Forward Pass with Example



- The inputs are one hot encoded. Our entire vocabulary is {h,e,l,o} and hence we can easily one hot encode the inputs.

1	0	0	0
0	1	0	0
0	0	1	1
0	0	0	0
h	e	l	l

- Now the input neuron would transform the input to the hidden state using the weight U. We have randomly initialized the weights as a 3*4 matrix –

U			
0.287027	0.84606	0.572392	0.486813
0.902874	0.871522	0.691079	0.18998
0.537524	0.09224	0.558159	0.491528

Forward Pass with Example

Step 1

- Now for the letter "h", for the the hidden state we would need UX_t. By matrix multiplication, we get it as

U			
0.287027	0.84606	0.572392	0.486813
0.902874	0.871522	0.691079	0.18998
0.537524	0.09224	0.558159	0.491528

×

x_t
1
0
0
0
h

=

0.287027
0.902874
0.537524

Forward Pass with Example

Step 2

- Now moving to the recurrent neuron, we have W as the weight which is a 1*1 matrix as **0.427043** and the bias which is also a 1*1 matrix as **0.56700**
- For the letter "h", the previous state is [0,0,0] since there is no letter prior to it.
- So to calculate -> $(W{*}s_{t-1}+\text{bias})$

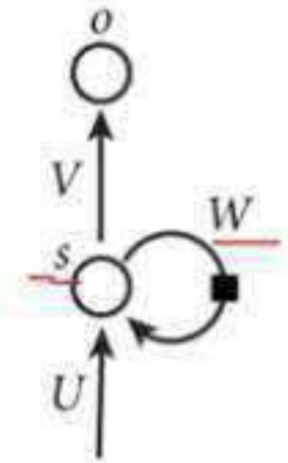
W	bias
0.427043	0.567001

×

0
0
0
s_{t-1}

=

0.567001
0.567001
0.567001



Forward Pass with Example

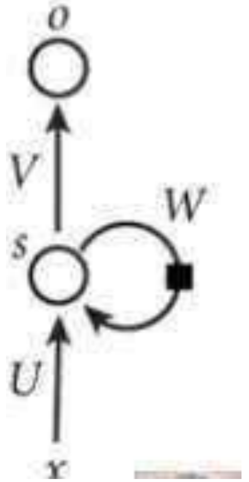
Step 3

- Now we can get the current state as

$$s_t = \tanh(\underline{Ws_{t-1}} + \underline{Ux_t})$$

- Since for h, there is no previous hidden state we apply the tanh function to this output and get the current state s_t

$$\begin{bmatrix} 0.287027 \\ 0.902874 \\ 0.537524 \end{bmatrix} + \begin{bmatrix} 0.567001 \\ 0.567001 \\ 0.567001 \end{bmatrix} = \tanh\left\{ \begin{bmatrix} 0.854028 \\ 1.469875 \\ 1.104525 \end{bmatrix} \right\} = \begin{bmatrix} 0.693168 \\ 0.899554 \\ 0.802118 \end{bmatrix}$$



Forward Pass with Example

Step 4

- Now we go on to the next state. "e" is now supplied to the network. The processed output of s_t, now becomes s_{t-1}, while the one hot encoded e, is x_t. Let's now calculate the current state s_t.

$$h_t = \tanh(Ws_{t-1} + Ux_t)$$

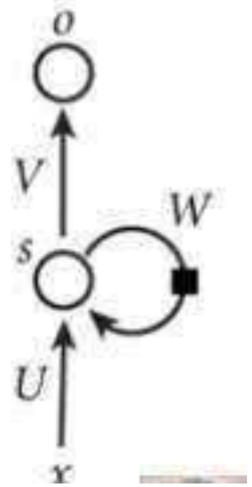
- Ws_{t-1} +bias will be

$$\begin{bmatrix} 0.427043 \end{bmatrix} \times \begin{bmatrix} 0.693168 \\ 0.899554 \\ 0.802118 \end{bmatrix} + \begin{bmatrix} 0.567001 \end{bmatrix} = \begin{bmatrix} 0.863013 \\ 0.951149 \\ 0.90954 \end{bmatrix}$$

- Ux_t will be

U			
0.287027	0.84606	0.572392	0.486813
0.902874	0.871522	0.691079	0.18998
0.537524	0.09224	0.558159	0.491528

$$\times \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ e \end{bmatrix} = \begin{bmatrix} 0.84606 \\ 0.871522 \\ 0.09224 \end{bmatrix}$$

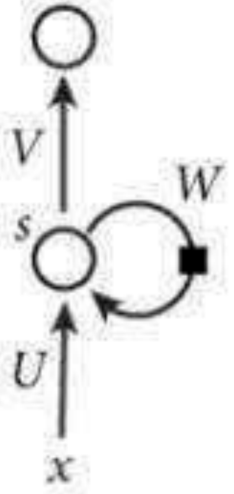


Forward Pass with Example

Step 5

➢ Now calculating s_t for the letter "e",

$$s_t = \tanh\left\{\begin{bmatrix}0.863013\\0.951149\\0.90954\end{bmatrix} + \begin{bmatrix}0.84606\\0.871522\\0.09224\end{bmatrix}\right\} = \begin{bmatrix}0.93653372\\0.94910403\\0.76234056\end{bmatrix}$$



➢ Now this would become s_{t-1} for the next state and the recurrent neuron would use this along with the new character to predict the next one.

Forward Pass with Example

Step 6

- At each state, the recurrent neural network would produce the output as well. Let's calculate y_t for the letter e.

$$Y_t = Vs_t$$

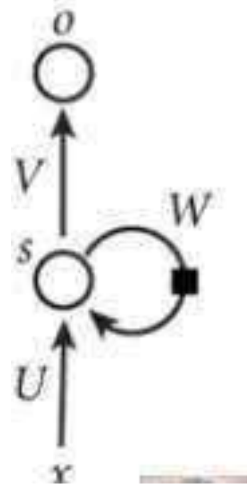
V		
0.37168	0.974829459	0.830034886
0.39141	0.282585823	0.659835709
0.64985	0.09821557	0.334287084
0.91266	0.32581642	0.144630018

×

s_t
0.93653372
0.94910403
0.76234056

=

y_t
1.90607732
1.13779113
0.95666016
1.27422602



TOPICS IN DEEP LEARNING

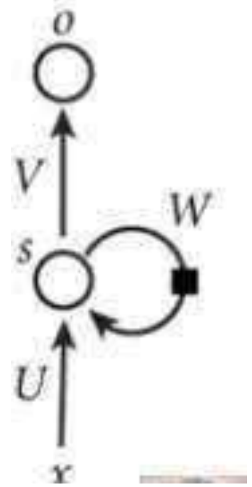
Forward Pass with Example

Step 7

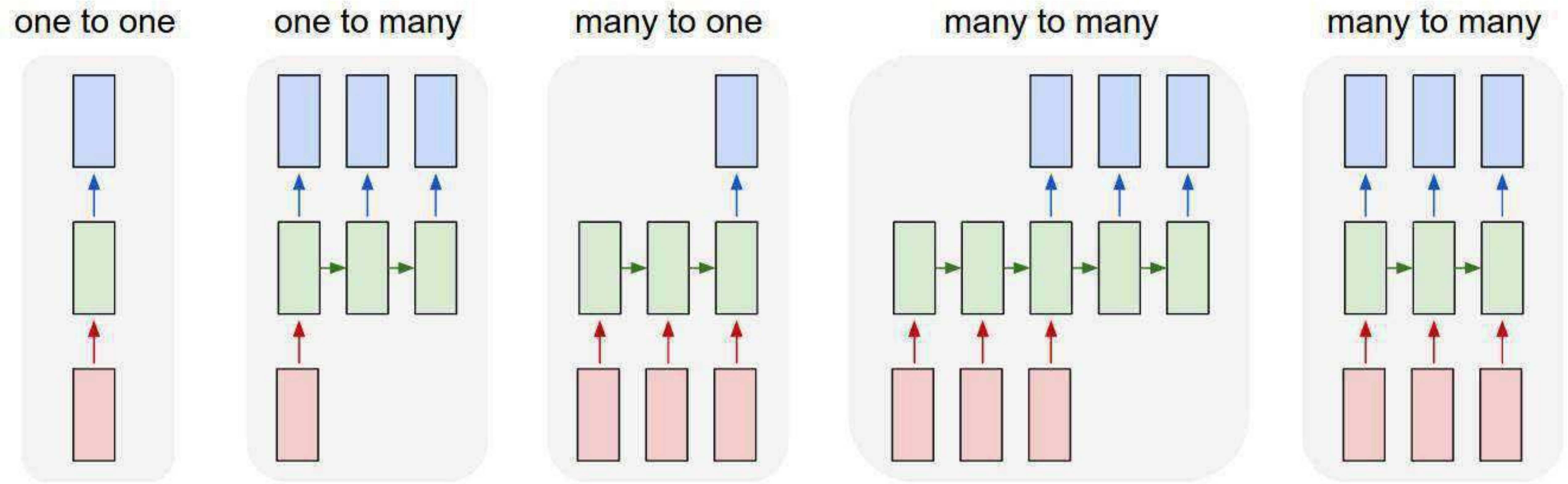
- The probability for a particular letter from the vocabulary can be calculated by applying the softmax function. so we shall have softmax(y_t)

Classwise probabilities for next letter = softmax $\left\{ \begin{array}{|c|} \hline y_t \\ \hline 1.90607732 \\ \hline 1.13779113 \\ \hline 0.95666016 \\ \hline 1.27422602 \\ \hline \end{array} \right\}$ = $\begin{array}{|c|} \hline 0.419748 \\ \hline 0.194682 \\ \hline 0.162429 \\ \hline 0.223141 \\ \hline \end{array}$ Letter h got high probability

- If we convert these probabilities to understand the prediction, we see that the model says that the letter after "e" should be h, since the highest probability is for the letter "h". Does this mean we have done something wrong? No, so here we have hardly trained the network. We have just shown it two letters. So it pretty much hasn't learnt anything yet.
- Now the next BIG question that faces us is how does Back propagation work in case of a Recurrent Neural Network. How are the weights updated while there is a feedback loop?

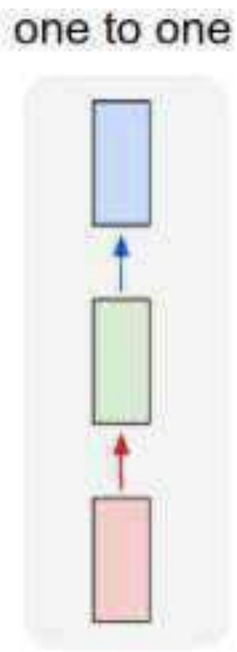


RNN- Variants



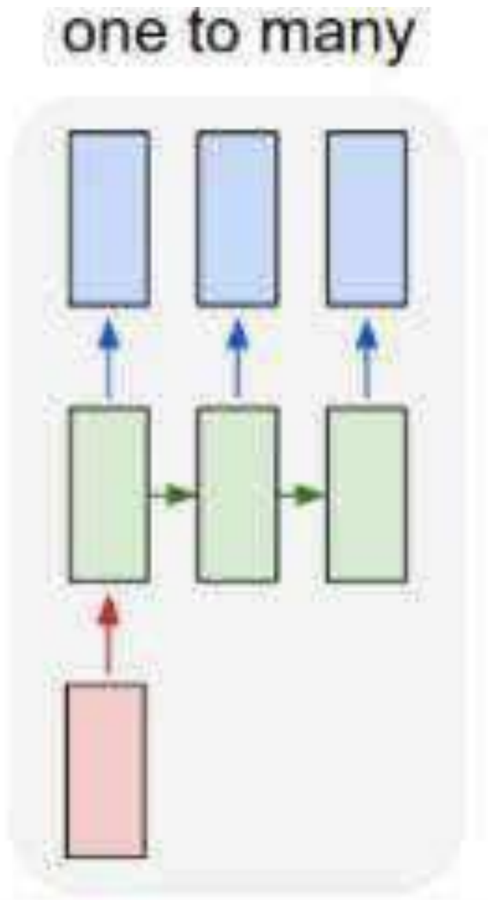
Input vectors are in red, output vectors are in blue and green vectors hold the RNN's state

RNN- Variant-1



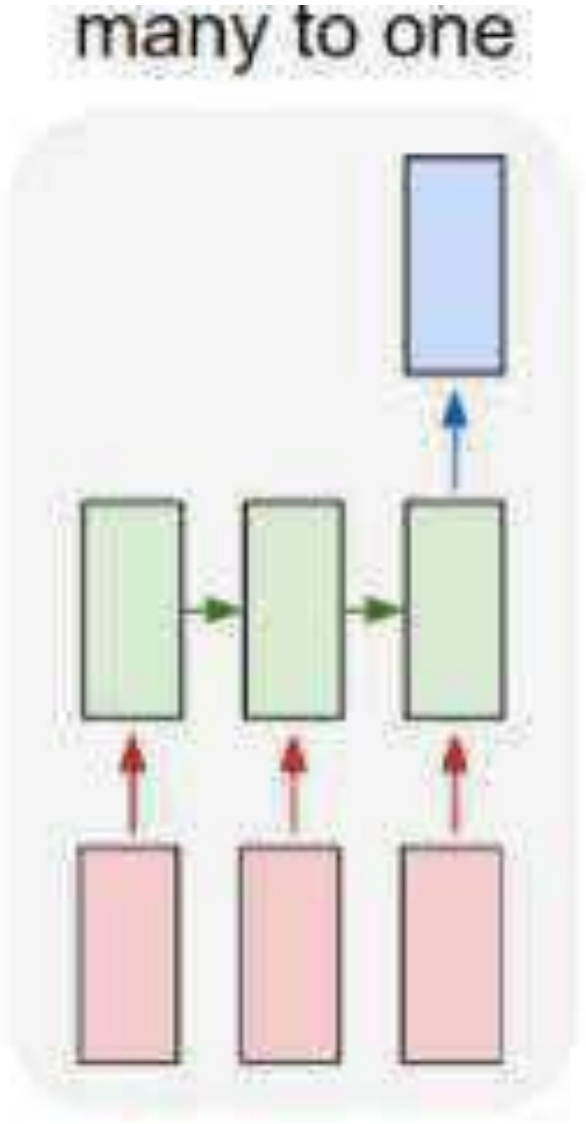
(1) Vanilla mode of processing without RNN, from fixed-sized input to fixed-sized output (e.g. image classification).

RNN- Variant-2



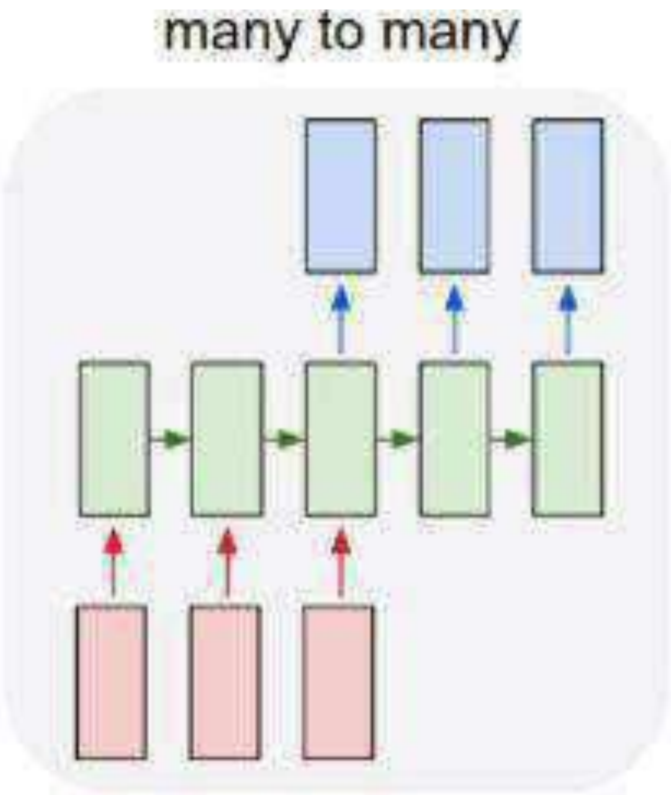
(2) Sequence output (e.g. image captioning takes an image and outputs a sentence of words).

RNN- Variant-3



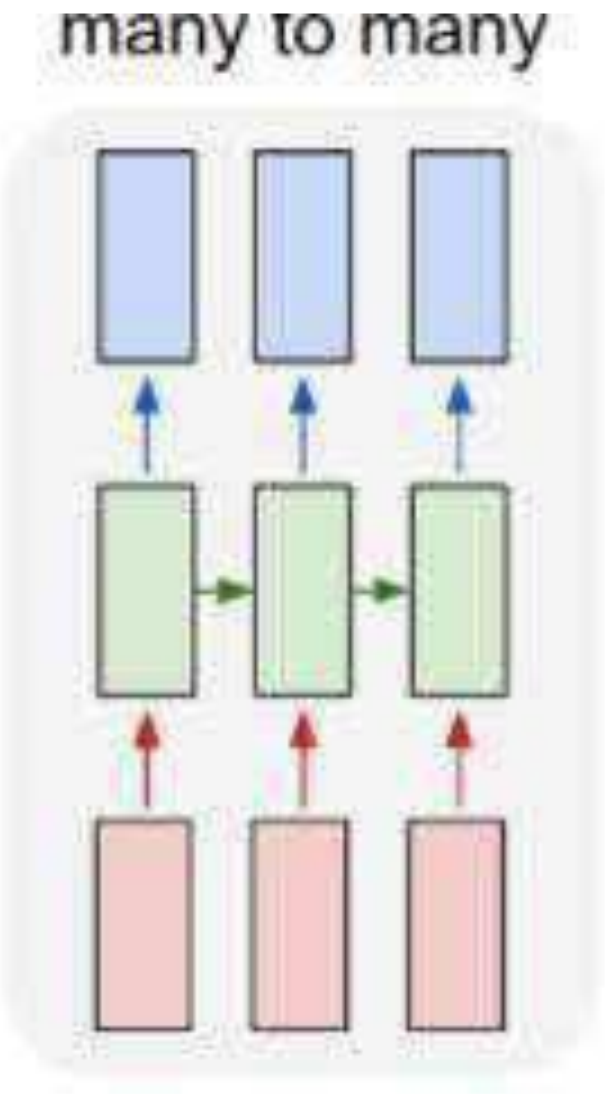
(3) Sequence input (e.g. sentiment analysis where a given sentence is classified as expressing positive or negative sentiment).

RNN- Variant-4



(4) Sequence input and sequence output (e.g. Machine Translation: an RNN reads a sentence in English and then outputs a sentence in French)

RNN- Variant-5

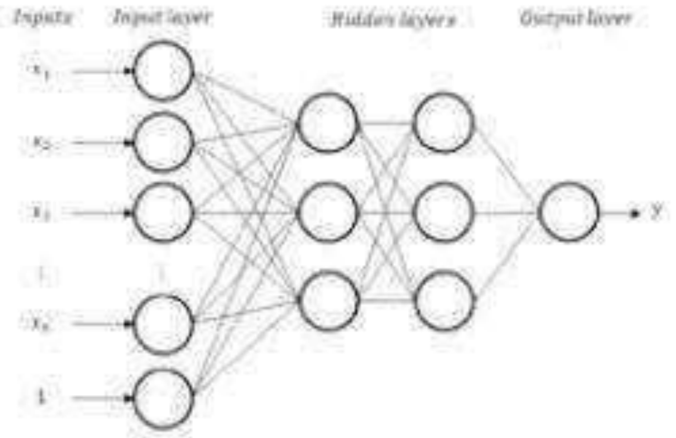


(5) Synced sequence input and output (e.g. video classification where we wish to label each frame of the video).

Why RNN ?

Sentiment Analysis

- Let us try to classify following text as positive or negative
  - I like this phone – Positive
  - This phone is good – Positive
  - This phone is not okay – Negative
  - I do not like this phone because battery is not charging properly - Negative



Feed forward networks accept a fixed-sized vector as input !

Why RNN ?

Solution 1: Using Bag-of-Words

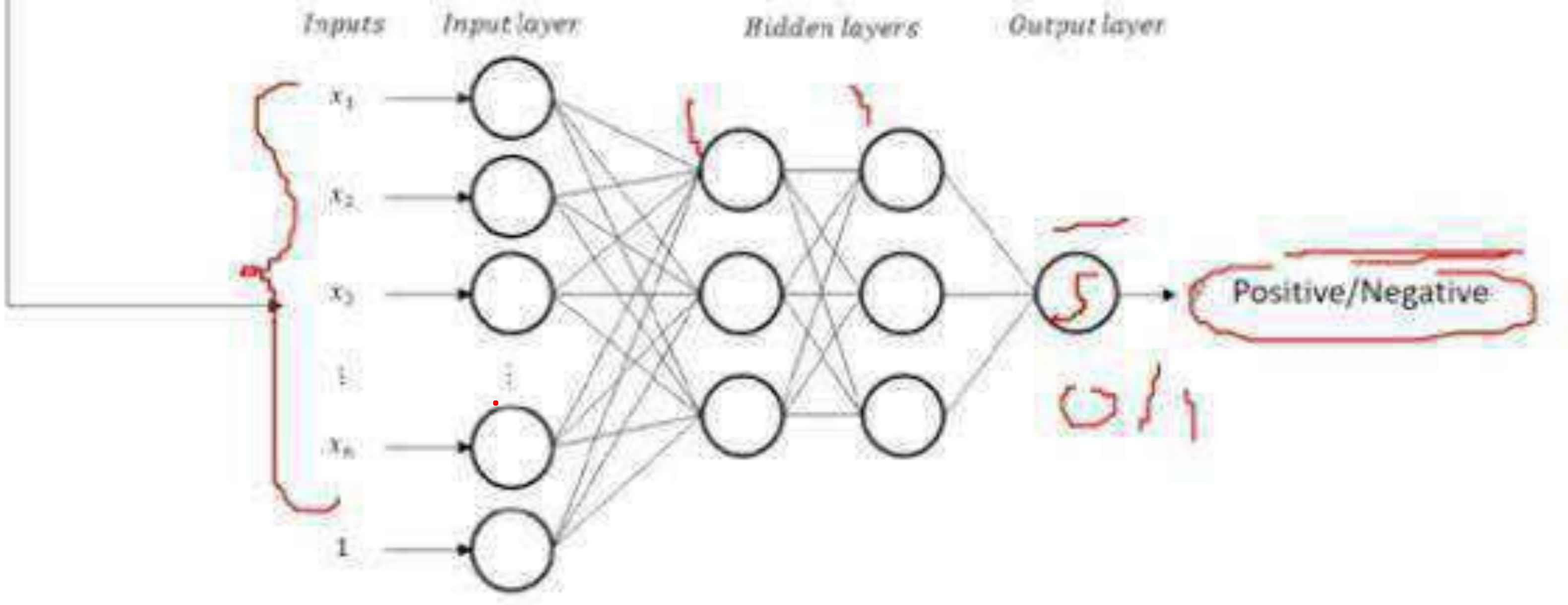
- Represent the text using Bag-of-Words
  - I like this phone
  - This phone is good
  - This phone is not okay
  - I do not like this phone because battery is not good

	battery	because	charging	do	good	I	is	like	not	okay	phone	properly	this
Doc 1	0	0	0	0	0	1	0	1	0	1	0	0	1
Doc 2	0	0	0	0	1	0	1	0	0	0	1	0	1
Doc 3	0	0	0	0	0	0	1	0	1	1	1	0	1
Doc 4	0	0	0	0	0	0	0	0	0	0	0	0	0

Why RNN?

Applying ANN

	battery	because	charging	do	good	I	is	like	not	okay	phone	properly	this
Doc 1	0	0	0	0	0	1	0	1	0	1	0	0	1
Doc 2	0	0	0	0	1	0	1	0	0	0	1	0	1
Doc 3	0	0	0	0	0	0	1	0	1	1	1	0	1
Doc 4	0	0	0	0	0	0	0	0	0	0	0	0	0



Why RNN?

Drawback of BoW

- Lets try to represent the following text using Bag-of-Words
  - This phone is no good - Negative
  - No this phone is good - Positive

	good	is	no	phone	this
Doc 1	1	1	1	1	1
Doc 2	1	1	1	1	1

- Feed Forward Neural Networks with Bag-of-words (BoW) model does not consider position of words in input !

TOPICS IN DEEP LEARNING

Why RNN?

Example : 2, Word Guessing Game

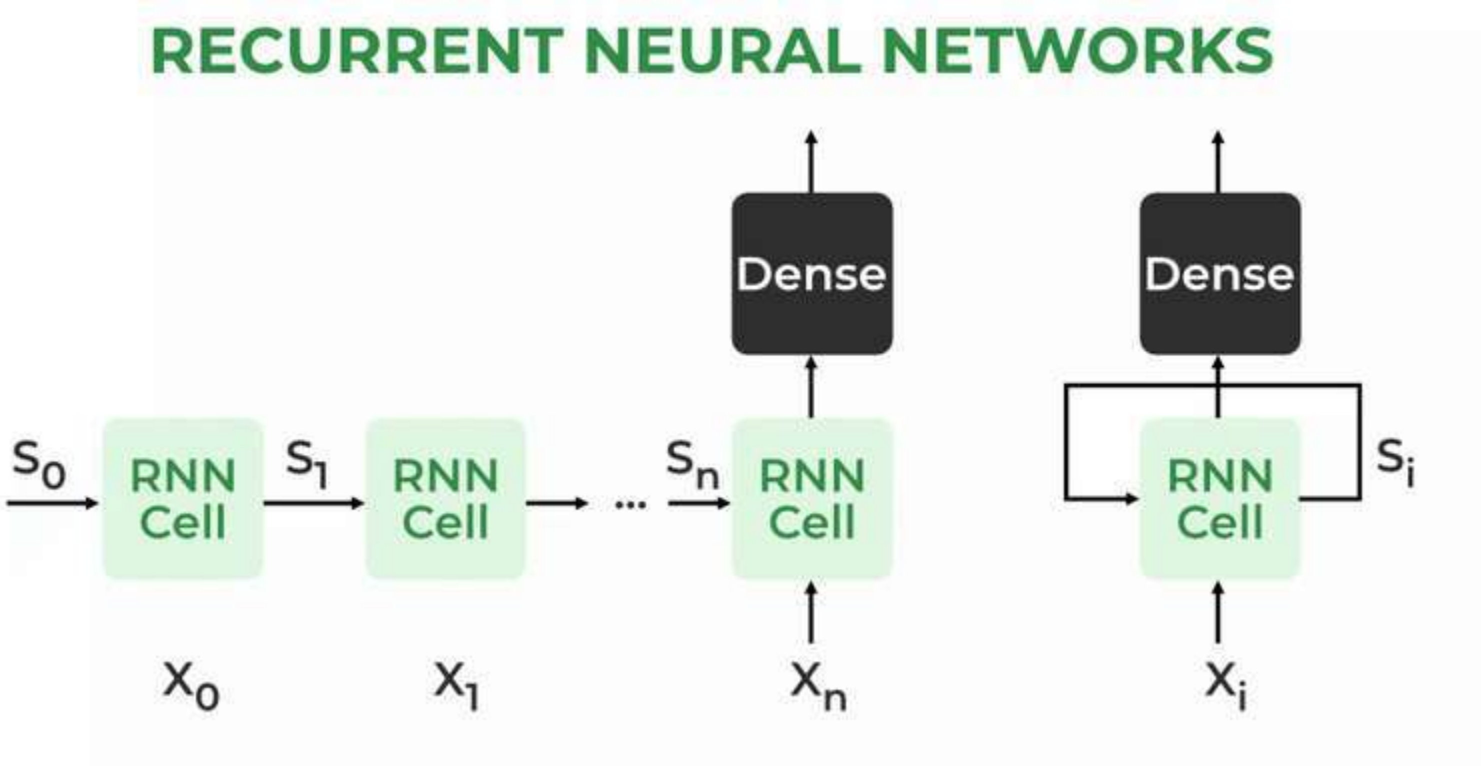
- Guess the missing word in the follow sentence
  - I went to France last year, there the people speak the french ________ language
- Lets find the missing word with the sentence (alphabetically sorted words)
  - Sorted Text: food I is it like Thai so
  - Original Text: I like Thai food it is so ________
  - Answer: I like Thai food it is so ________
- Feed Forward Neural Network will fail for guessing missed words
- Conclusion: Sequence matters !

Sequence Data

Other situations where sequence matters

1. Stock price data will be more or less similar to yesterday’s price
2. Tomorrow’s temperature will be close to today’s temperature

Recurrent Neural Network



Updating the Hidden State in RNNs

1. **State Update:**

$$h_t = f(h_{t-1}, x_t)$$

2. **Activation Function Application:**

$$h_t = \tanh(W_{hh} \cdot h_{t-1} + W_{xh} \cdot x_t)$$

3. **Output Calculation:**

$$y_t = W_{hy} \cdot h_t$$

Working with Sequences

- We sometimes wish to predict a fixed target y given sequentially structured input (e.g., sentiment classification based on a movie review).
- At other times, we wish to predict a sequentially structured target ($y_1, \ldots, y_T$) given a fixed input (e.g., image captioning).
- Still other times, our goal is to predict sequentially structured targets based on sequentially structured inputs (e.g., machine translation or video captioning).
- Such sequence-to-sequence tasks take two forms:

(i) *aligned*: where the input at each time step aligns with a corresponding

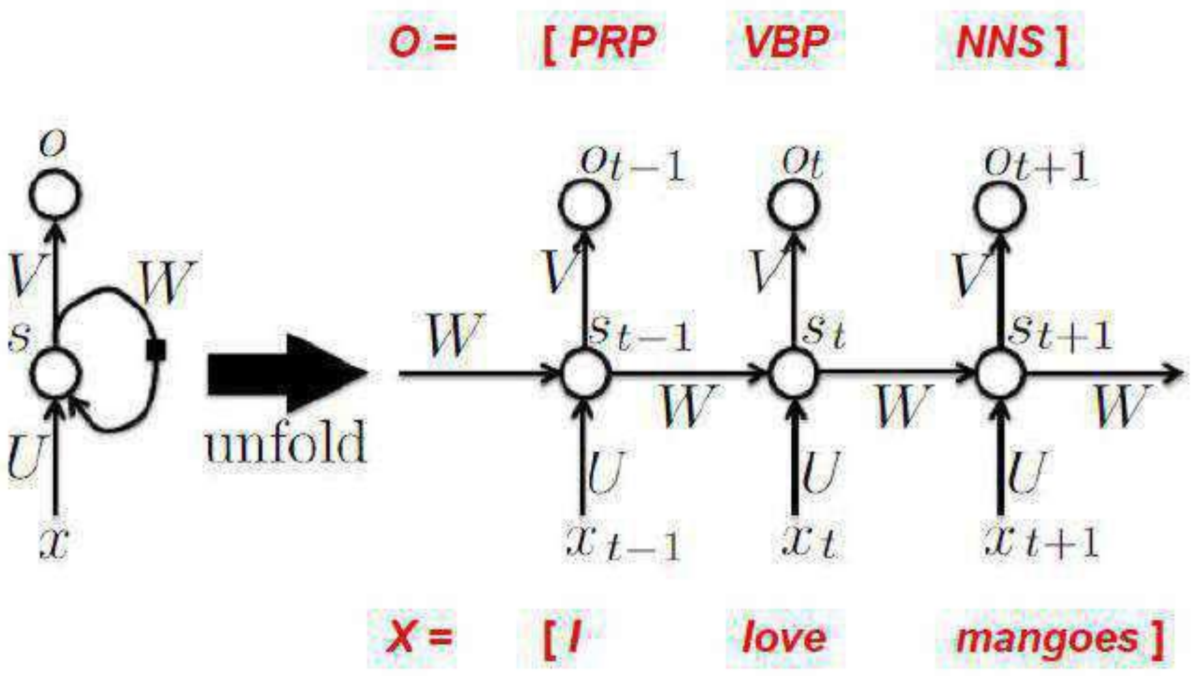
target (e.g., part of speech tagging);

(ii) *unaligned*: where the input and target do not necessarily exhibit a step-for-step correspondence (e.g., machine translation).

RNN Example-1

Part-of-speech tagging:

- Given a sentence X, tag each word its corresponding grammatical class.



PRP: personal pronoun, vbp:Verb, present tense, other than third person singular, nns:noun,plural

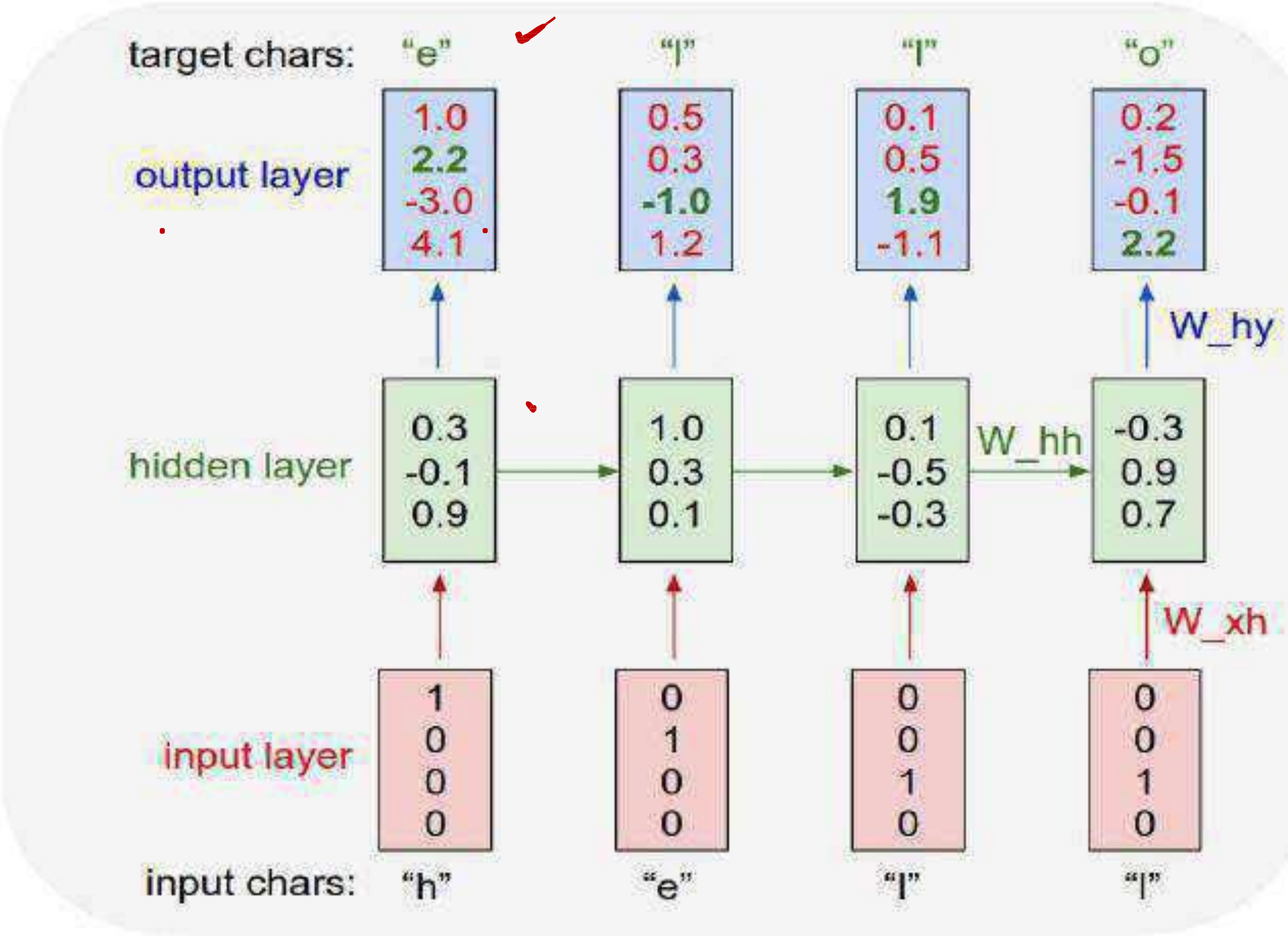
RNN Example-2

Character level language model:

- Given previous and current characters, predict the next character in the sequence.

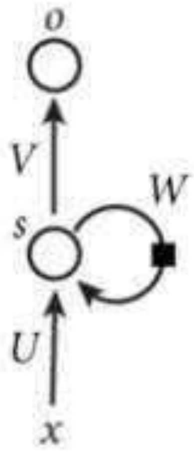
Let

- **Vocabulary: [h,e,l,o]**
- **One-hot representations**
  - h = [1 0 0 0]
  - e = [0 1 0 0]
  - l = [0 0 1 0]
  - o = [0 0 0 1]



RNN Forward Pass with Example

Forward Pass with Example



- The inputs are one hot encoded. Our entire vocabulary is {h,e,l,o} and hence we can easily one hot encode the inputs.

1	0	0	0
0	1	0	0
0	0	1	1
0	0	0	0
h	e	l	l

- Now the input neuron would transform the input to the hidden state using the weight U. We have randomly initialized the weights as a 3*4 matrix –

U			
0.287027	0.84606	0.572392	0.486813
0.902874	0.871522	0.691079	0.18998
0.537524	0.09224	0.558159	0.491528

Forward Pass with Example

Step 1

- Now for the letter "h", for the the hidden state we would need UX_t. By matrix multiplication, we get it as

U			
0.287027	0.84606	0.572392	0.486813
0.902874	0.871522	0.691079	0.18998
0.537524	0.09224	0.558159	0.491528

×

x_t
1
0
0
0
h

=

0.287027
0.902874
0.537524

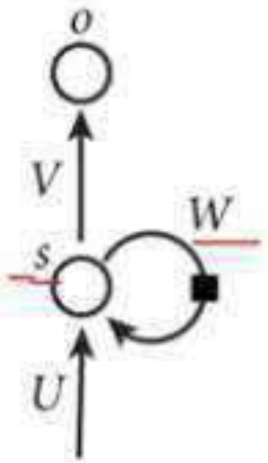
Forward Pass with Example

Step 2

- Now moving to the recurrent neuron, we have W as the weight which is a 1*1 matrix as **0.427043** and the bias which is also a 1*1 matrix as **0.56700**
- For the letter “h”, the previous state is [0,0,0] since there is no letter prior to it.
- So to calculate -> $(W * s_{t-1} + bias)$

W	bias
0.427043	0.567001

$$\times \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}_{s_{t-1}} = \begin{bmatrix} 0.567001 \\ 0.567001 \\ 0.567001 \end{bmatrix}$$



Forward Pass with Example

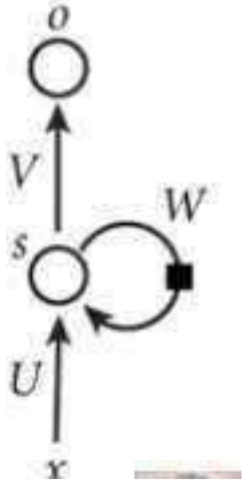
Step 3

- Now we can get the current state as

$$s_t = \tanh(\underline{Ws_{t-1}} + \underline{Ux_t})$$

- Since for h, there is no previous hidden state we apply the tanh function to this output and get the current state s_t

$$\begin{bmatrix}0.287027\\0.902874\\0.537524\end{bmatrix} + \begin{bmatrix}0.567001\\0.567001\\0.567001\end{bmatrix} = \tanh\left\{\begin{bmatrix}0.854028\\1.469875\\1.104525\end{bmatrix}\right\} = \begin{bmatrix}0.693168\\0.899554\\0.802118\end{bmatrix}$$



Forward Pass with Example

Step 4

- Now we go on to the next state. "e" is now supplied to the network. The processed output of s_t, now becomes s_{t-1}, while the one hot encoded e, is x_t. Let's now calculate the current state s_t.

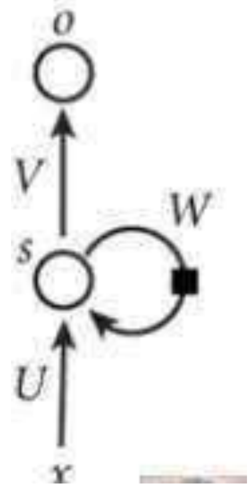
$$h_t = \tanh(Ws_{t-1} + Ux_t)$$

- Ws_{t-1} +bias will be

$$\begin{bmatrix} 0.427043 \end{bmatrix} \times \begin{bmatrix} 0.693168 \\ 0.899554 \\ 0.802118 \end{bmatrix} + \begin{bmatrix} 0.567001 \end{bmatrix} = \begin{bmatrix} 0.863013 \\ 0.951149 \\ 0.90954 \end{bmatrix}$$

- Ux_t will be

$$\begin{array}{c} \text{U} \\ \begin{bmatrix} 0.287027 & 0.84606 & 0.572392 & 0.486813 \\ 0.902874 & 0.871522 & 0.691079 & 0.18998 \\ 0.537524 & 0.09224 & 0.558159 & 0.491528 \end{bmatrix} \end{array} \times \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ \text{e} \end{bmatrix} = \begin{bmatrix} 0.84606 \\ 0.871522 \\ 0.09224 \end{bmatrix}$$

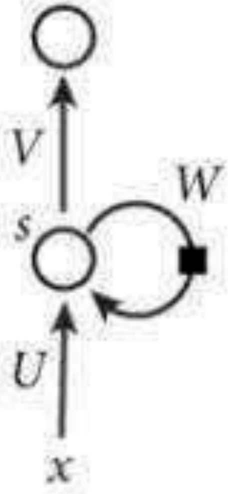


Forward Pass with Example

Step 5

- Now calculating s_t for the letter "e",

$$s_t = \tanh\left\{ \begin{bmatrix} 0.863013 \\ 0.951149 \\ 0.90954 \end{bmatrix} + \begin{bmatrix} 0.84606 \\ 0.871522 \\ 0.09224 \end{bmatrix} \right\} = \begin{bmatrix} 0.93653372 \\ 0.94910403 \\ 0.76234056 \end{bmatrix}$$



- Now this would become s_{t-1} for the next state and the recurrent neuron would use this along with the new character to predict the next one.

Forward Pass with Example

Step 6

- At each state, the recurrent neural network would produce the output as well. Let's calculate y_t for the letter e.

$$Y_t = Vs_t$$

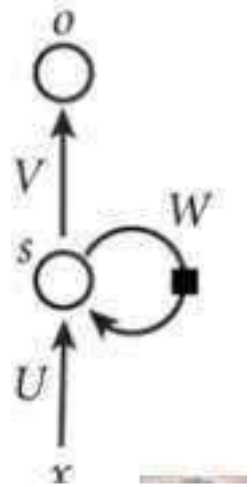
V		
0.37168	0.974829459	0.830034886
0.39141	0.282585823	0.659835709
0.64985	0.09821557	0.334287084
0.91266	0.32581642	0.144630018

✖

s_t
0.93653372
0.94910403
0.76234056

=

y_t
1.90607732
1.13779113
0.95666016
1.27422602



Forward Pass with Example

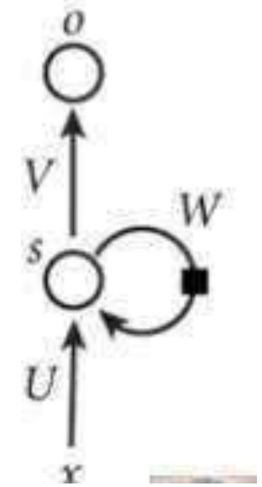
Step 7

- The probability for a particular letter from the vocabulary can be calculated by applying the softmax function. so we shall have softmax(y_t)

$$\text{Classwise probabilities for next letter} = \text{softmax}\left\{\begin{array}{|c|}\hline y_t \\ \hline 1.90607732 \\ \hline 1.13779113 \\ \hline 0.95666016 \\ \hline 1.27422602 \\ \hline\end{array}\right\} = \begin{array}{|c|}\hline 0.419748 \\ \hline 0.194682 \\ \hline 0.162429 \\ \hline 0.223141 \\ \hline\end{array}$$

Letter h got high probability

- If we convert these probabilities to understand the prediction, we see that the model says that the letter after "e" should be h, since the highest probability is for the letter "h". Does this mean we have done something wrong? No, so here we have hardly trained the network. We have just shown it two letters. So it pretty much hasn't learnt anything yet.
- Now the next BIG question that faces us is how does Back propagation work in case of a Recurrent Neural Network. How are the weights updated while there is a feedback loop?



Thank You