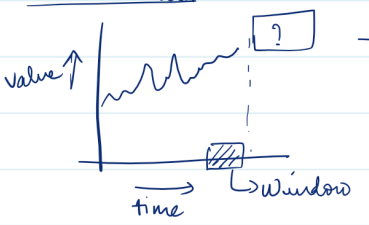
Why RNNs (5.1):-

→ MLP :- $x \rightarrow$ [MLP] $\rightarrow y$ | Sequences ?

CNN :- x (Image) $\rightarrow$ [CNN] $\rightarrow y$

eg:- This phone is very fast } Both have the same TFIDF/BOW representation. Discarded sequence info.
① Phone this is very fast } but both are not the same.

② Time Series Data :-



→ Used windows and fourier transforms previously.

③ Machine Translations :- Language translations. The sentences hold sequential information. Using RNN we work with seq2seq models.

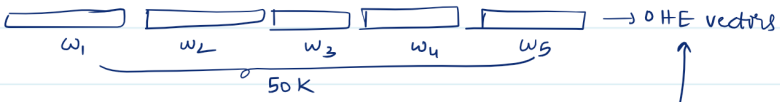
④ Speech Recognition :- Sequence of Audio → English sentence

⑤ Image captions. (Google Image Search)

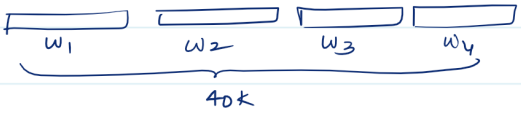
→ The idea is to build a NN that retains & leverages sequential information.

→ Problem with not using RNN & using an MLP directly is inputs are of different sizes

eg:- the review :- This phone is very fast
w_1 w_2 w_3 w_4 w_5



the review :- This phone is good
w_1 w_2 w_3 w_4



We could do zero padding to make all inputs equal but
(i) Does not work for longer test queries
(ii) Unnecessarily increasing dimensionality

Recurrent Neural Network:- (5.2)

→ Recurrent = repeats.

→ Taking example of Amazon Fine Food Reviews

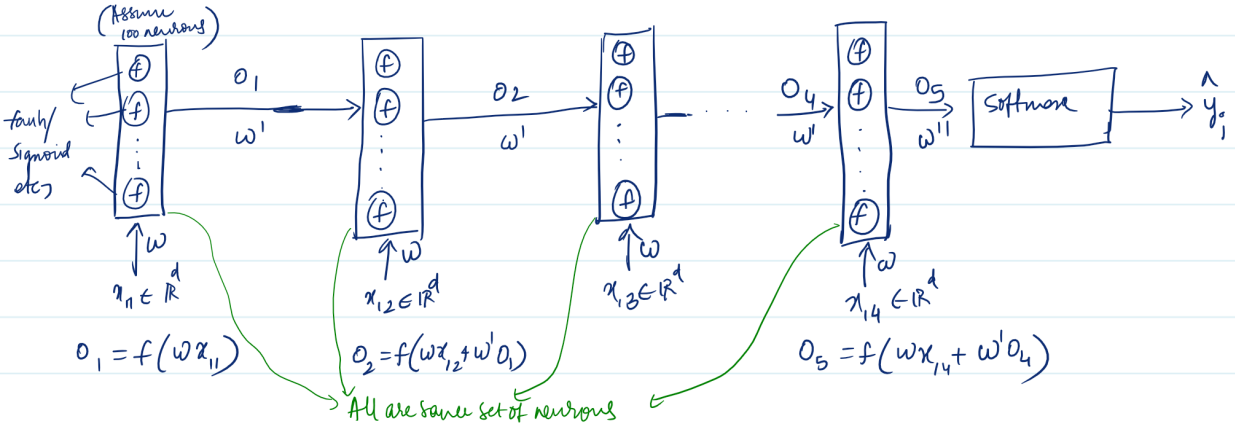
$D = \{x_i, y_i\}$

$x_i \rightarrow x_{i1}, x_{i2}, \ldots x_{im_i}$ where m_i = len of x_i

Assume size of Vocabulary = d

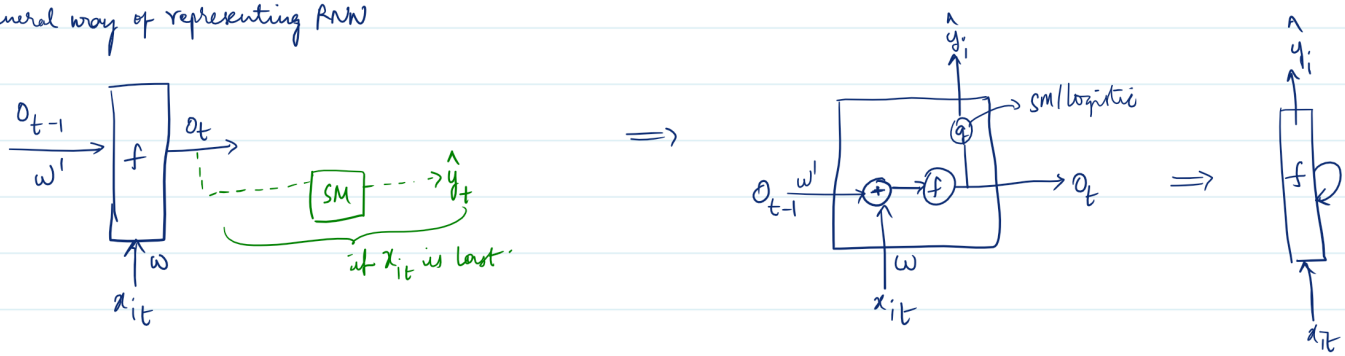
$x_{ij} \in \mathbb{R}^d$ → Binary vectors (one hot encoding) ; $x_1 = \langle x_{11}, x_{12}, x_{13}, x_{14}, x_{15} \rangle$, y_1

At $t=1$ (iteration 1) → At $t=2$ → At $t=3$ ..... At $t=5$ w' is same at all stages.



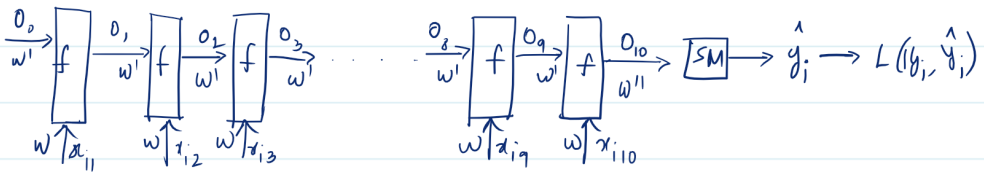
→ For the next sentence the same W are reused.

→ General way of representing RNN



Training RNNs : Backpropogation (5.3) :-

→ Backprop for RNN = Backprop over time



time →

$$o_1 = f(wx_{i1} + w'o_0)$$
$$o_2 = f(wx_{i2} + w'o_1)$$
$$\vdots$$
$$o_{10} = f(wx_{i10} + w'o_9)$$
$$\hat{y}_i = g(w''o_{10})$$

→ Forward propogation = direction of arrows (time)

Back prop : opposite direction.

$$\frac{\partial L}{\partial \hat{y}_i} \quad \left| \quad \frac{\partial L}{\partial o_{10}} = \frac{\partial L}{\partial \hat{y}_i} \cdot \frac{\partial \hat{y}_i}{\partial o_{10}} \right. \quad \left| \quad \frac{\partial L}{\partial w''} = \frac{\partial L}{\partial \hat{y}_i} \cdot \frac{\partial \hat{y}_i}{\partial w''} \right. \quad \left| \quad \frac{\partial L}{\partial w} = \frac{\partial L}{\partial \hat{y}_i} \cdot \frac{\partial \hat{y}_i}{\partial o_{10}} \cdot \frac{\partial o_{10}}{\partial w} \right.$$

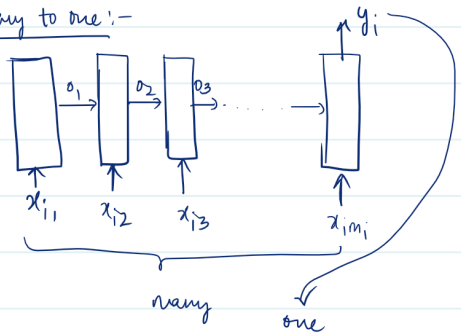
$$\frac{\partial L}{\partial w} = \left(\frac{\partial L}{\partial \hat{y}_i} \cdot \frac{\partial \hat{y}_i}{\partial o_{10}} \cdot \frac{\partial o_{10}}{\partial w}\right) + \left(\frac{\partial L}{\partial \hat{y}_i} \cdot \frac{\partial \hat{y}_i}{\partial o_{10}} \cdot \frac{\partial o_{10}}{\partial o_9} \cdot \frac{\partial o_9}{\partial w}\right) + \left(\frac{\partial L}{\partial \hat{y}_i} \cdot \frac{\partial \hat{y}_i}{\partial o_{10}} \cdot \frac{\partial o_{10}}{\partial o_9} \cdot \frac{\partial o_9}{\partial o_8} \cdot \frac{\partial o_8}{\partial w}\right) + \cdots \quad (\ )$$

lots of multiplications involved

→ Since we are performing backpropogation over time & $\frac{\partial L}{\partial w}$ will have lots of multiplications based on number of words, we run into vanishing gradient & exploding gradient problem. This is NOT because of many layers but because of sequence (sentence) length. LSTMs & GRUs solve this problem.

Types of RNN Architectures (5.4) :-

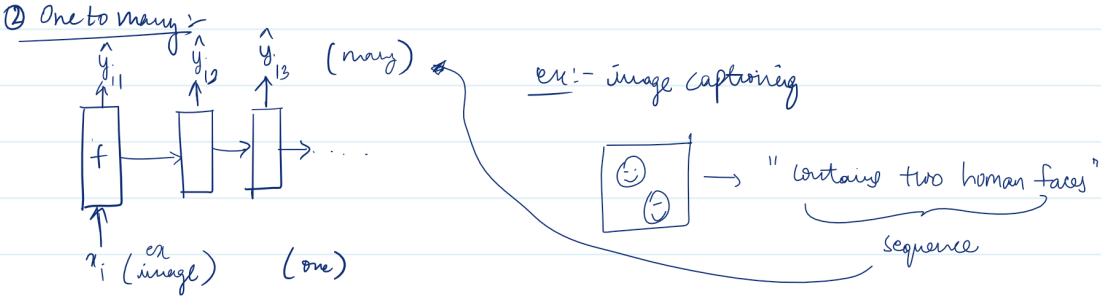
① Many to one :-



→ many to one. example :- ① sentiment classification (food reviews)

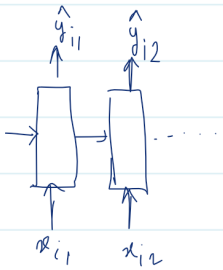
② Movie review → star rating (multiclass classfn)

② One to many :-



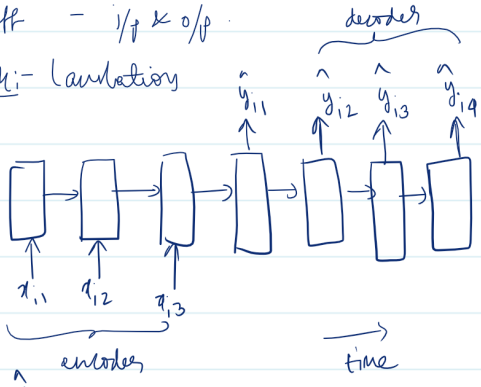
③ Many to Many :- input → seq
output → seq

type① :- input & output same length
ex:- parts of speech of sentence



seq2seq models

type② :- diff - i/p & o/p
ex:- translation

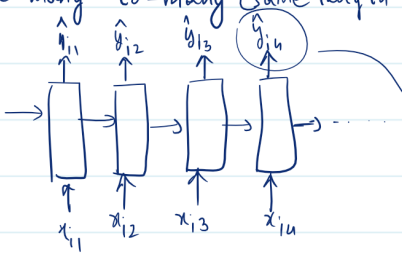


④ one to one :-

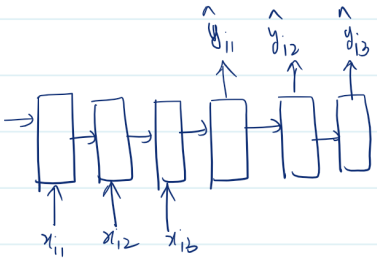
→ simple MLP. No sequence.

Need for LSTM/GRUs (5.5) :-

→ Take many-to-many (same length for example)



$\hat{y}_{i4}$ in this model depends mostly on x_{i4}. It's value does not depend on x_{i1} because the gradient vanishes by the time it reaches the calculation of $\hat{y}_{i4}$.
But in real world there may be cases where $\hat{y}_{i4}$ is more dependent on x_{i1} than x_{i4}.
There's no long term dependancy in this case. (in simple RNN)
Later output depends bit more on earlier inputs.

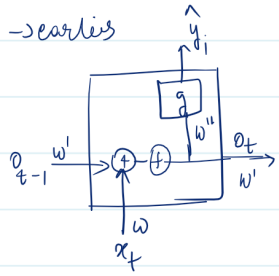


→ $\hat{y}_{i3}$ could depend on x_{i1} in case of translation. Simple RNN can't handle this.

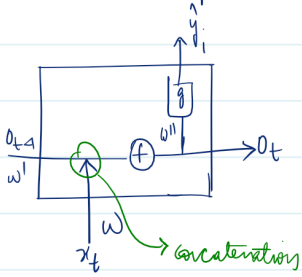
LSTM (5.6) :-

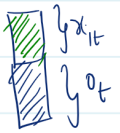
→ It can take care of both long term & short term dependancies.

→ earlier



Now →

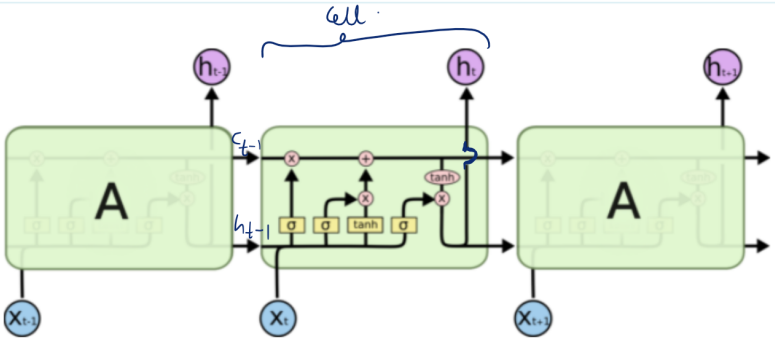




$o_t = f(wx_t + w'o_{t-1})$
$\hat{y}_{it} = g(w''o_t)$

$o_t = f([w', w][x_{it}\ o_t])$
$\hat{y}_{it} = g(w''o_t)$

concatenation operation. Not additions like before.



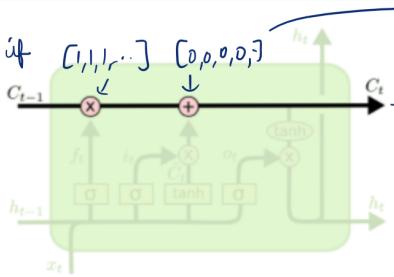
The repeating module in an LSTM contains four interacting layers.

3 outputs.

$h_t = y_t$

$x_t = x_t$

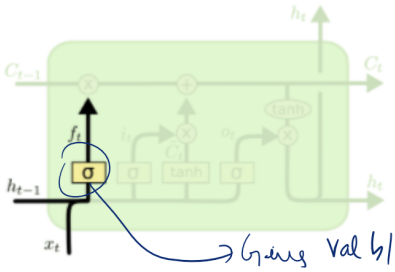
$c_t = o_t$



if [1,1,1,..] [0,0,0,0,] → $c_t = c_{t-1}$ (works like identity function)

→ This handles how much info from prev layer needs to be remembered (×) and how much from current data point is to be added (+)

Forget Gate Layer :-

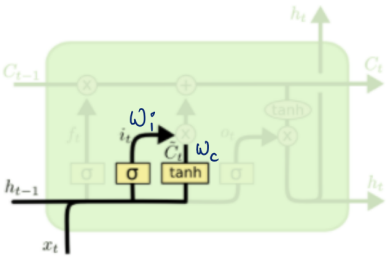


$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

→ Going val b/w 0 & 1.

→ handles how much informations we are going to throw away from the current layer.

Input Gate Layer :-

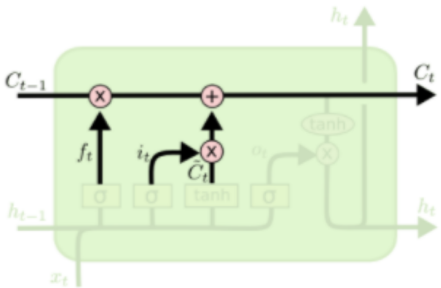


$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

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

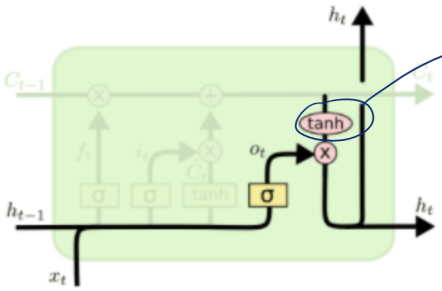
→ handles what new informations we are going to store in cell state.

Cell state :-



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

Output layer :-



→ to get val b/w −1 & +1 so we can multiply with o/p of sigmoid gate & only o/p the parts we want to.

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

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

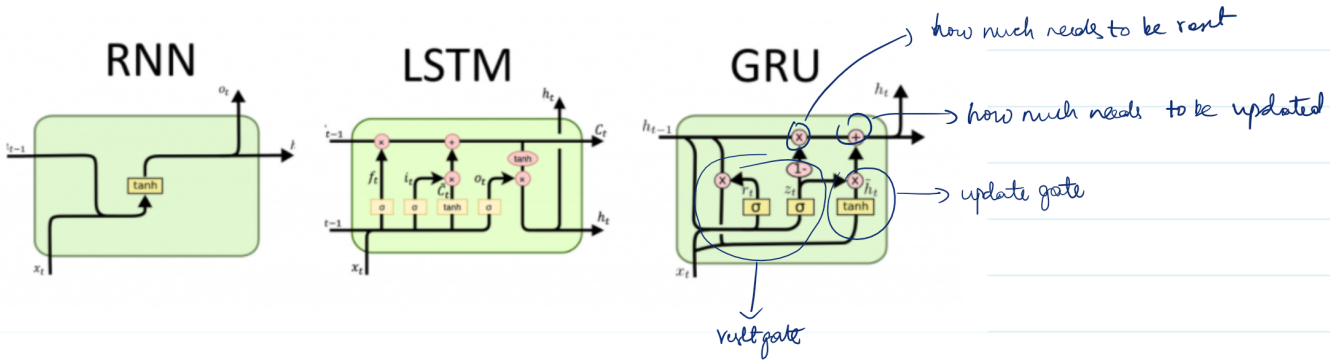
→ 4 weights. All are concatenations of x_t & h_{t-1}. All learn diff weights through.

GRU (Gated Recurrent Unit) (5.7):-

LSTM - 1997 , GRU - 2014.

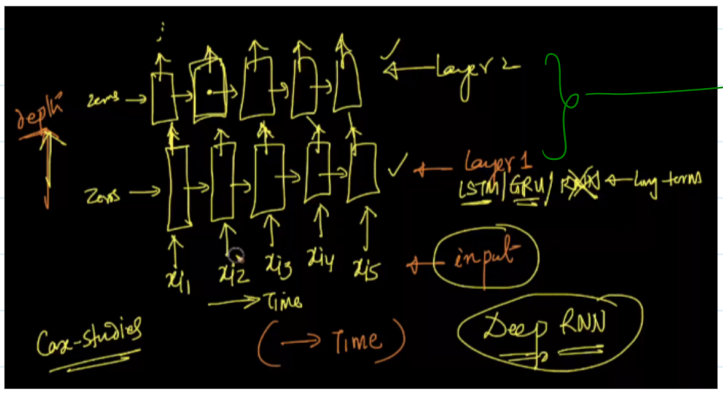
→ simplified & faster to train versions of LSTMs. As powerful as GRU. Merges the cell state & hidden state.

2 gates → reset gate, update gate.



fewer equations ⇒ fewer updates ⇒ fewer backpropagations ⇒ faster.

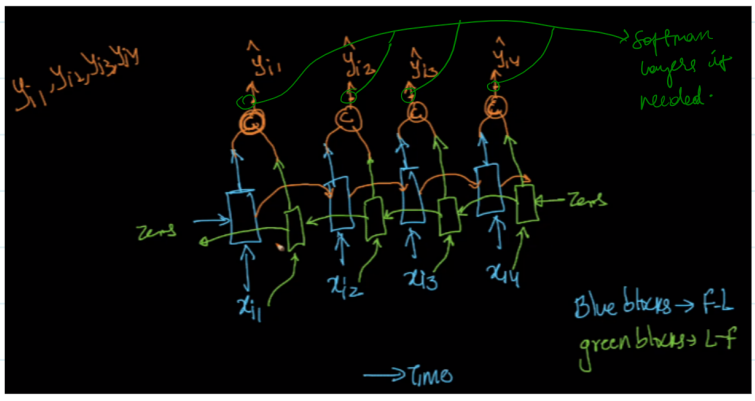
Deep RNN (5.8):-



→ Not always equal. Number of cells is a hyperparameter like neurons in MLP.

Bidirectional RNN (5.9):-

→ What if y_{it} is dependent on $x_{i,t+k}$. Simple RNN can't work with such data.



Use cases:-

→ NLP tasks

→ Speech to text.

IMDB Text Classification (5.10):-

→ IMDB keras $x_1 = <x_{11}, x_{12}, \ldots>, y_1$

$x_2 = <x_{21}, x_{22}, \ldots>, y_2$

V = set of all the words in all the reviews.

Step ①:-

word	freq
The	100000
a	20000
⋮	⋮

Step ②:- sort by frequency

Step ③:-

	word	freq
1	a	20000
2	an	18000
3	the	1000
⋮	⋮	⋮

give words numbers

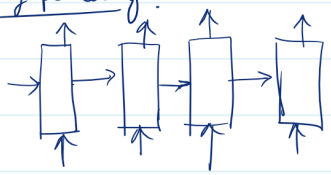
Step ④:- x_1 = This phone is a good phone

$< 81, \dots, 1, \dots >$

$< x_{11}, \dots x_{14}, \dots >$

→ padding :- convert all reviews to the same length by inserting zeros at the beginning.

Why padding ?



$x_1 = < x_{1,1}, x_{1,2}, \dots x_{1,150} >$

$x_2 = < x_{2,1}, x_{2,2}, \dots, x_{2,100} >$

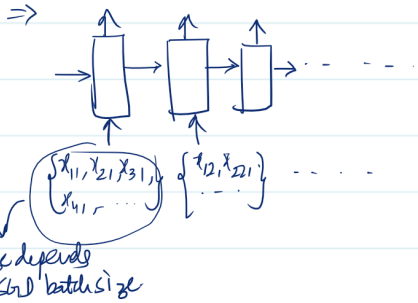
→ Normally, sent one i/p after another.

↓

Similar to SGD batch size = 1.

with padding, all elements have the same size.

⇒



as possible. Speeds up training.

size depends on SGD batch size

→ embedding takes a +ve integer & returns a vector.

eg:- 1 → embedding → [. . . . .]

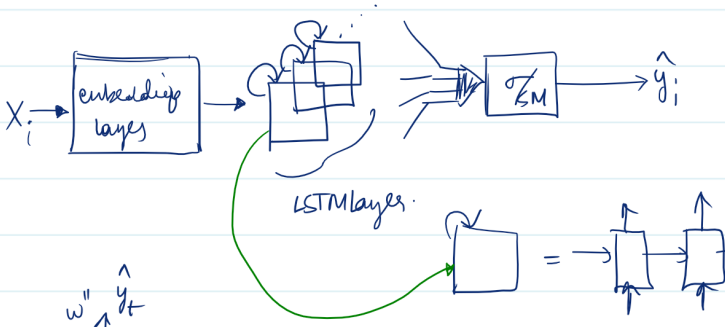
embedding vector length.

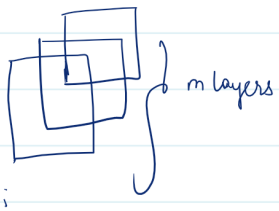
→ if there are m inputs & n outputs to LSTM,

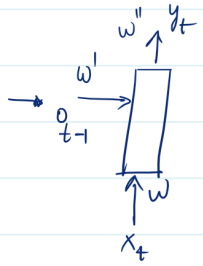
$\#\text{params} = 4(nm + n^2 + n)$

↳ bias (Keras by default adds bias)

→







$\hat{y}_t$ = Scalar

$w'' = m \times 1$

$w' = m \times m$

$w = n \times m$

$x_t = 1 \times n$

$o_{t-1} = 1 \times m$

4 gates exist (3 σ, 1 tanh)

each = $m \times m + n \times m = m^2 + nm = 4(m^2 + nm + m)$

↳ Bias

m = # of neurons in layer = # of "outputs"

n = # of dimensions of x_t

2)Also I'm finding it difficult to understand top level on how the process happens, so here the input we'll pass in x_train[0] right? It will be a vector of 600 dimension because of pad sequence . Then first embedding layer converts each feature of 600 features to 32 dimensions right? So it will be like a list of lists. Then LSTM take in each word(of 32 dim) i.e (first list in list of lists) and then takes the second word and so on and performs forward and backward prop and outputs the final binary decision for one sentence right? Is this how it works internally from a top level point of view. Please correct me if I'm mistaken anything.

Reply

Jul 02, 2019 02:46 AM

AppliedAI

yes, you are correct.