9/4/18

Course 5 (RNN)

RNN have transformed NLP and speech recognition.
↓
a sequence model

In some sequence problems, both i/p X and o/p Y are sequences and in some cases either of them is a sequence.

e.g. Speech Recognition (Input: Audio Output: text)
Music Generation (Input: Nothing or genre name of music to be generated. Output: Audio)

Sentiment classification (Input: Text Output: Number may be)

Machine translation (" " " Output: Text)
Video activity recognition (I/p: frames of images Output: Activity identified in the i/p)
, etc ... like named entity Recognition (i.e identifying names in an i/p)

e.g. x: Harry Potter and Hermoine Granger invented a new spell.

	$x^{<1>}$	$x^{<2>}$	$x^{<3>}$	$x^{<4>}$	$x^{<5>}$	$x^{<6>}$	$x^{<7>}$	$x^{<8>}$	$x^{<9>}$
y:	1	1	0	1	1	0	0	0	0
	$y^{<1>}$	$y^{<2>}$	$y^{<3>}$	$y^{<4>}$	$y^{<5>}$	$y^{<6>}$	$y^{<7>}$	$y^{<8>}$	$y^{<9>}$

$T_x = 9$ (length of i/p sequence)

$T_y = 9$ {In this e.g. $y^{(i)}$ is if $x^{<1>}$ is a name or not}

$y^{(i)<t>}$ = t^{th} element in the o/p sequence of i^{th} training example.

$T_x^{(i)}$ = input sequence length for training example i.

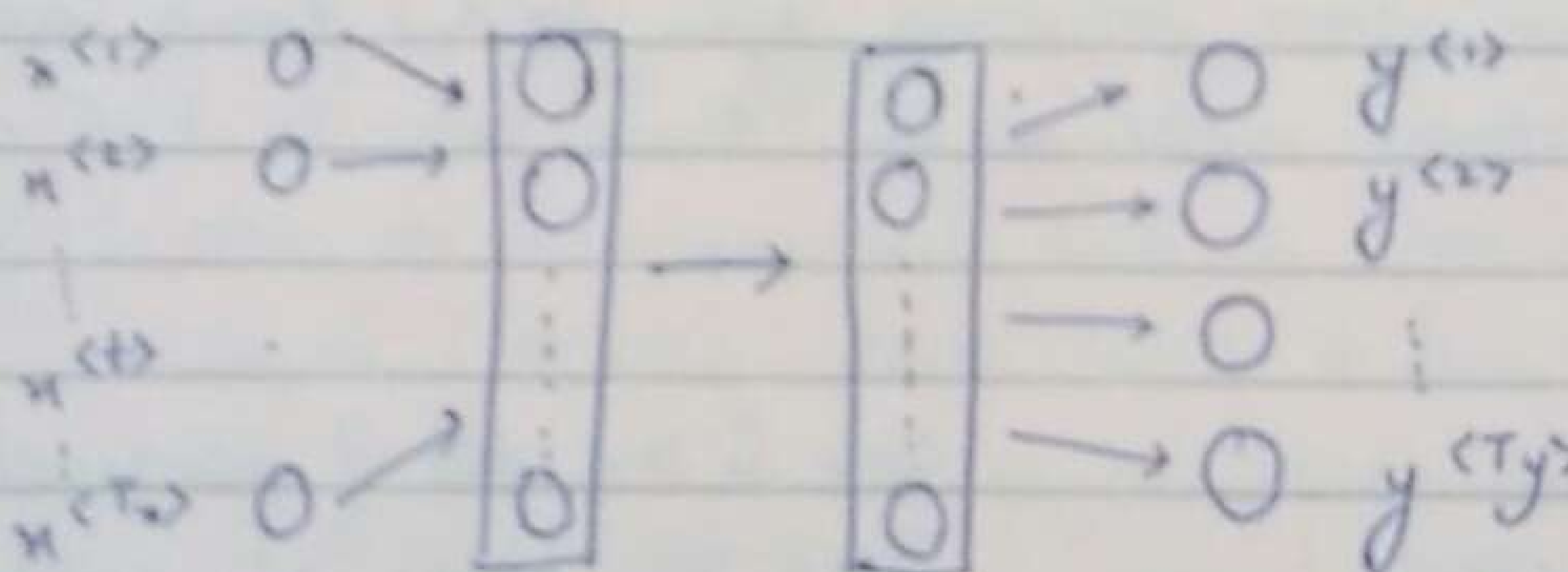
A vocabulary is next made. One way to build this dictionary is to look through the training sets and find the top occurring words or look at some online dictionary to know the most common words in the English language saved.

Suppose, vocabulary is $\begin{array}{c|l} \begin{bmatrix} \text{a} \\ \text{aaron} \\ \vdots \\ \text{and} \\ \vdots \\ \text{harry} \\ \vdots \\ \text{potter} \\ \vdots \\ \text{zulu} \end{bmatrix} & \begin{array}{l} 1 \\ 2 \\ \\ 367 \\ \\ 4075 \\ \\ 6830 \\ \\ \\ \end{array} \end{array}$ (if 10,000 words)

So, $\underbrace{x^{\langle 1 \rangle} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 1 \\ \vdots \\ 0 \\ \vdots \\ 0 \end{bmatrix} \leftarrow 4075 \quad x^{\langle 2 \rangle} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ \vdots \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \leftarrow 6830}_{\text{All are 10,000 dimensional one-hot vector.}}$ and so on.

If some word that doesn't exist in the vocabulary is encountered, then a <u>new token 'unknown' is added to the dictionary</u>.

→ Why not a standard network for such problems?



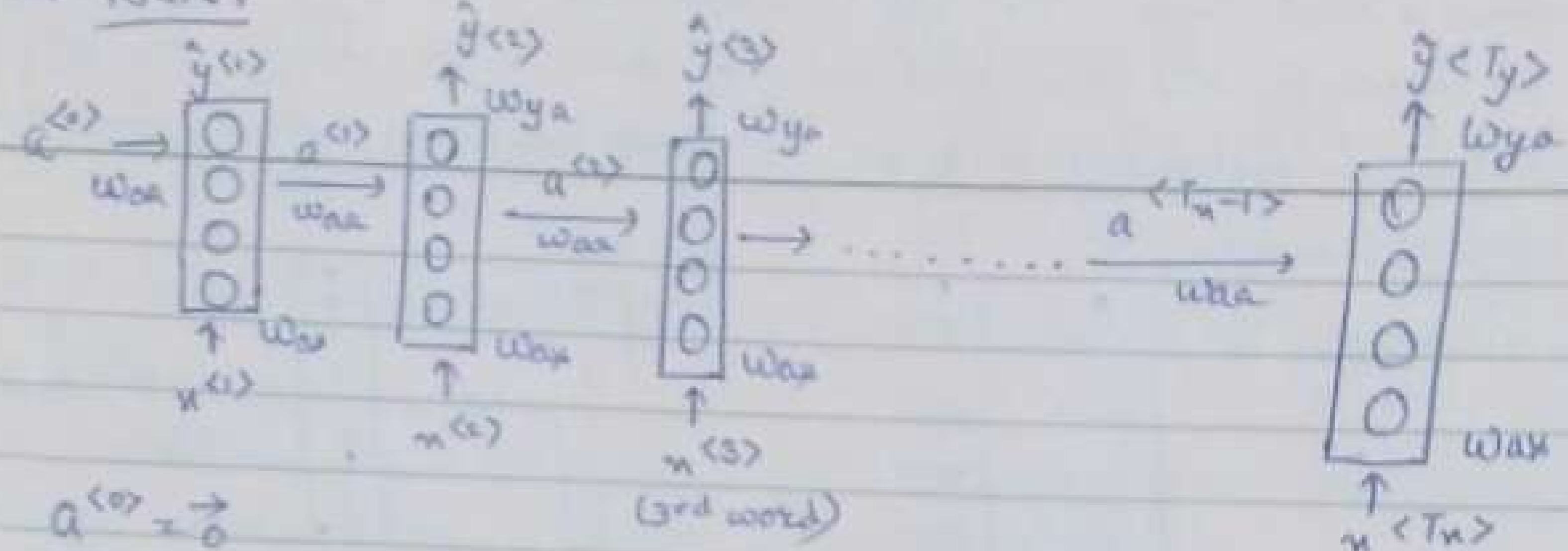
$x^{\langle t \rangle}$ is a one hot vector of 10,000 dimension.

Disadvantages of using this model for such data:

- Inputs, Outputs can be different lengths in different examples and padding can't always be done.
- Doesn't share features learned across different positions of text.
- Too large input size $[10,000 \times T_x]$ for just one input sentence, which would lead to an enormous no. of parameters in the weight matrix.

<u>RNN</u> does not have these disadvantages.

⇒ RNN:



$a^{<0>} = \vec{0}$

$a^{<1>} = g(W_{aa} a^{<0>} + W_{ax} X^{<1>} + b_a)$ ← tanh/Relu

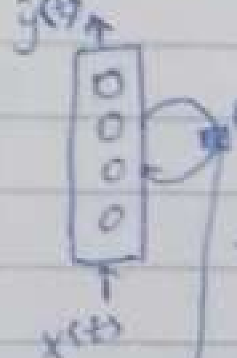
$\hat{y}^{<1>} = g'(W_{ya} a^{<1>} + b_y)$ ← Sigmoid in case of classifica^ problem

$a^{<t>} = g(W_{aa} a^{<t-1>} + W_{ax} X^{<t>} + b_a)$

$\hat{y}^{<t>} = g'(W_{ya} a^{<t>} + b_y)$

* W_{pq} means some q type quantity is used to calculate some p type quantity.

At every step RNN passes activa^ from previous layer to the next layer to use. $a^{<0>}$ is a vector of zeros made up at time zero to kick up the whole thing.



Recurrent loop drawn at each time to show that the layer feeds back to itself.

↳ shaded box to represent time delay of 1 step.

W_{ax} is the set of parameters used by RNN at each time step.
Horizontal connections are governed by the set of parameters W_{aa} used at every time step.
Similarly W_{ya} governs the output predictions.

* Limita^ of this network: It uses only the information from previous layers i.e. only the info earlier in the sequence to

make a prediction, i.e while predicting $\hat{y}^{<3>}$, it doesn't use info about the words $x^{<4>}$, $x^{<5>}$ or any later word in the sequence.

E.g.

i) He said, "Teddy Gupta was a great man".

ii) He said, "Teddy bears are on sale".

In i) and ii), to know whether Teddy is the name of a person, knowing just 1st. 3 words is not sufficient. Info about the later words is necessary to decide if Teddy is a name or not. In i) it is and in ii) its not, this is decided on the basis of info from later words only.

For this we will use <u>B-RNN</u> (Bidirectional RNN).

Also, $a^{<t>} = g(W_{aa} a^{<t-1>} + W_{ax} x^{<t>} + b_a)$ — ①

$$\boxed{a^{<t>} = g(W_a [a^{<t-1>}, x^{<t>}] + b_a)} \quad — ②$$

where $W_a = [W_{aa} \vdots W_{ax}] = (100, 10100)$ dimensional if 1

If a was 100 dimensional and x 1000 dimensional, then W_{aa} is $(100, 100)$ and W_{ax} is $(100, 10,000)$

$[a^{<t-1>}, x^{<t>}]$ means $= \begin{bmatrix} a^{<t-1>} \\ x^{<t>} \end{bmatrix} \updownarrow 10100$ justifying ① = ②.

So, $\hat{y}^{<t>} = g(W_y a^{<t>} + b_y)$

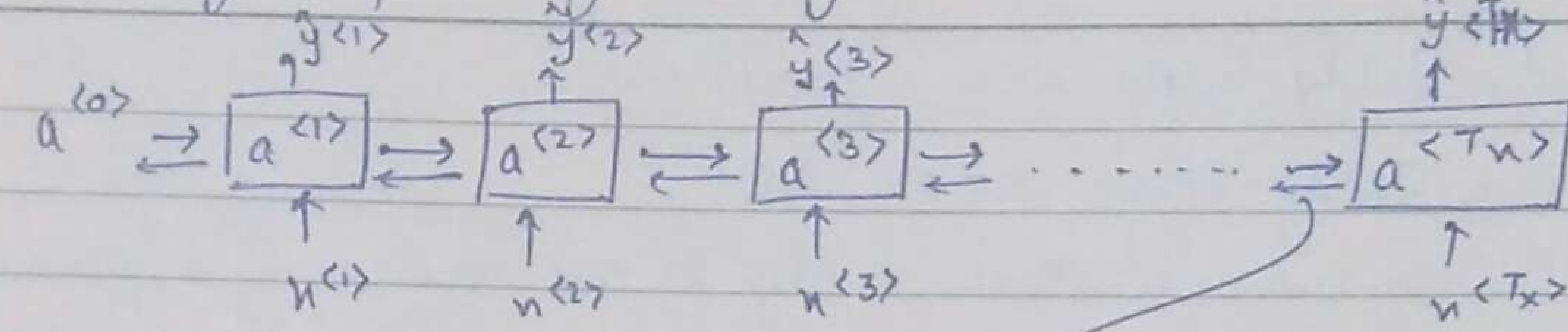
$$L^{<t>}(\hat{y}^{<t>}, y^{<t>}) = -y^{<t>} \log \hat{y}^{<t>} - (1 - y^{<t>}) \log(1 - \hat{y}^{<t>})$$

loss associated with single word of a single sentence.

$$\boxed{L(\hat{y}, y) = \sum_{t=1}^{T_x} L^{<t>}(\hat{y}^{<t>}, y^{<t>})}$$ {loss for entire sequence}

In the back propoga^n procedure, the significant calcula^n is the one that goes from right to left

(i)



$\{T_x = T_y\}$ (Many to many)

called 'Back propogation through time'.

This back propoga^n just like in earlier case, helps update parameters through gradient descent.

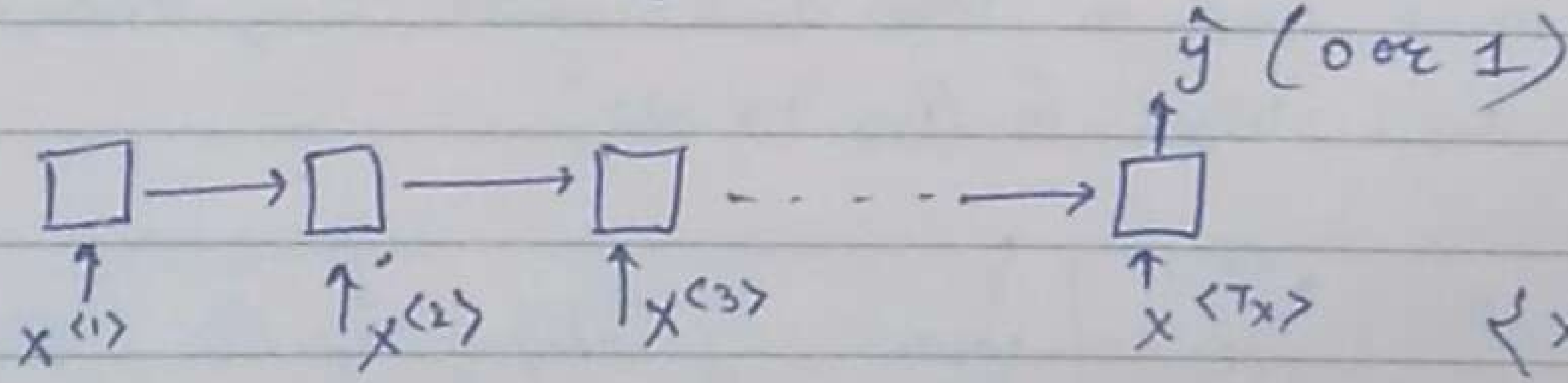
→ What if $T_x \neq T_y$ i.e no. of i/p ≠ no. of o/p?

e.g. sentiment classifica^n.

i/p is a sequence of words but o/p is 0/1 i.e either +ve or −ve review

for this we use many to 1 architecture.

(ii)

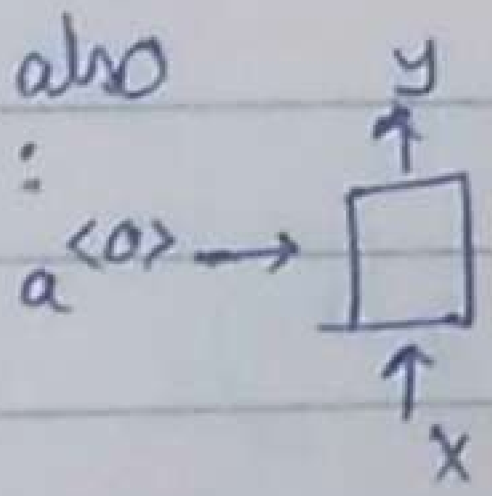


$\{x^{<1>} \ldots x^{<T_x>}$: i/p review about the movie$\}$.

Many to 1

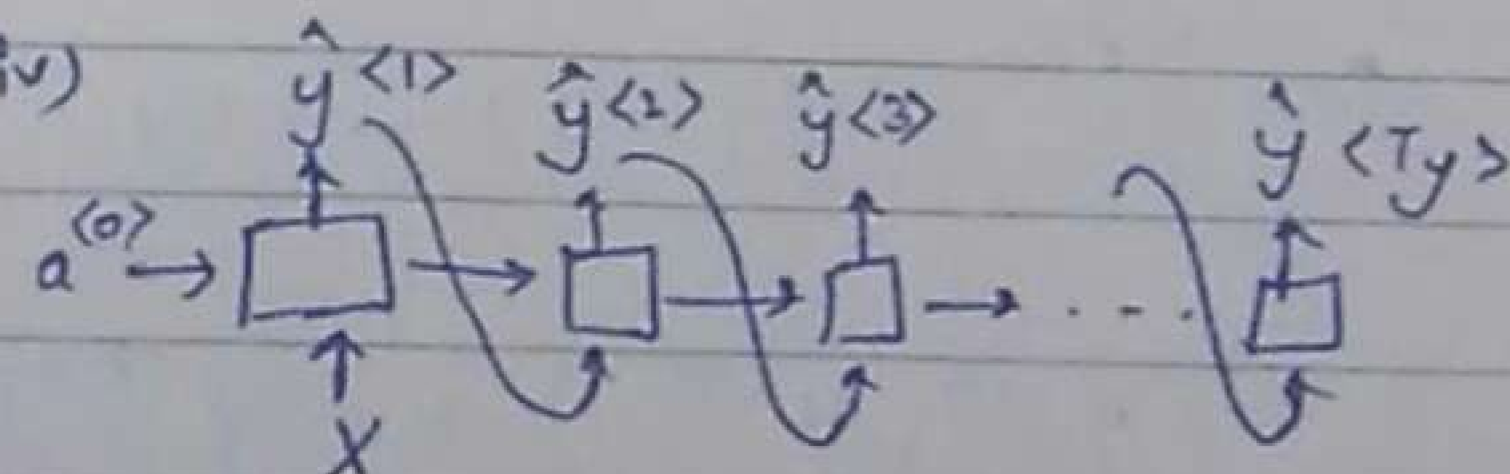
(iii)

There also has:



One to one architecture.

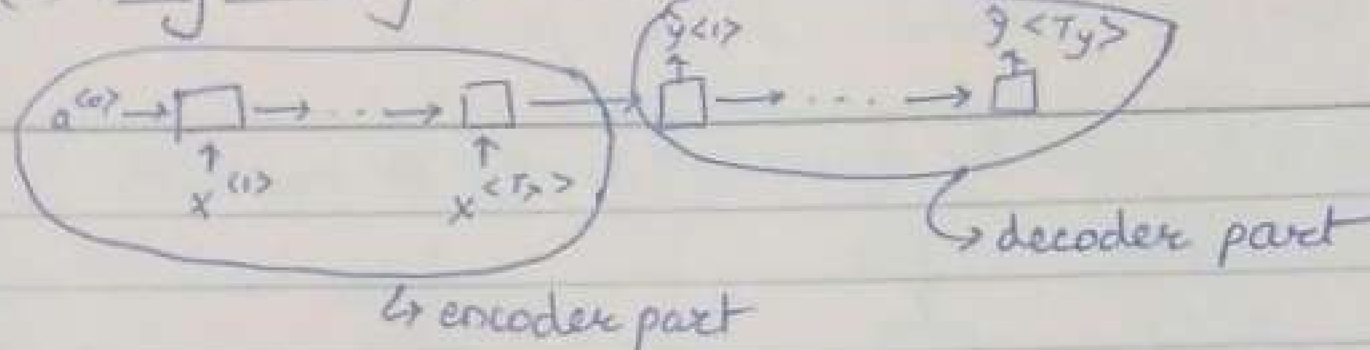
(iv)



One to many architecture

e.g. music genera^n or sequence genera^n.

(v) Many to many architecture with $T_x \neq T_y$.



e.g. French sentence translated to English. i/p and o/p may have diff length of sequences.

⇒ Building a language model using RNN: (e.g. to generate shakespeare like texting).

speech recognition:

$P(\text{The apple and pair salad}) = 3.2 \times 10^{-13}$

$P(\text{The apple and peer salad}) = 5.7 \times 10^{-10}$

{ language model learns that which of the two sentences has higher probability of being spoken depending on its learning. this is used both in speech recognition & machine/language translation }

→ How to build a language model?

Training set: large corpus of text in whatever language we want to build a model.

corpus → chunk / large body

Step1: Tokenize.

e.g. if i/p is Cats average 15 hrs of sleep a day.

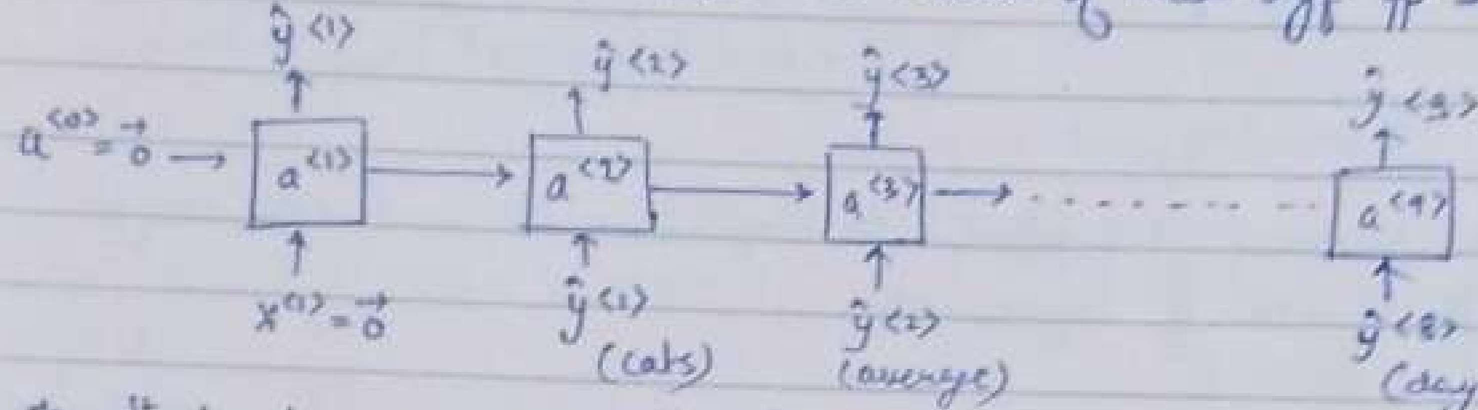
$y^{<1>}$ $y^{<2>}$ $y^{<3>}$ . . . . . . $y^{<8>}$ <EOS>

$y^{<1>}$ → one-hot vectors to represent them in dictionary.

<EOS> → additional token to tell end of sentence, if needed.

If some word appears in the training set which is not ~~in~~ in the dictionary, then token UNK (unknown) is used for it.

Step 2: build RNN model to model the chance of these diff. i/p sequences.



The 1st step has a softmax trying to know the 1st word depending on probabilities of the 1st word being any one from the dictionary of 10,000 words. Hence, this softmax o/p is 10,000 dimensional vector. whichever word gets the highest probability is considered to be the 1st word of sentence.

In 2nd step, o/p is again predicted by softmax with probability of 2nd word being any one from dictionary given 1st words is cats

$P(\_\_\_ \mid \text{cats}) = \hat{y}^{<2>}$

$\hat{y}^{<3>} = P(\_\_\_ \mid \text{"cats average"})$

$\hat{y}^{<9>} = P(\_\_\_ \mid \text{"cats average 15 hrs of sleep a day"})$.

Note In this model $x^{<t>} = y^{<t-1>}$

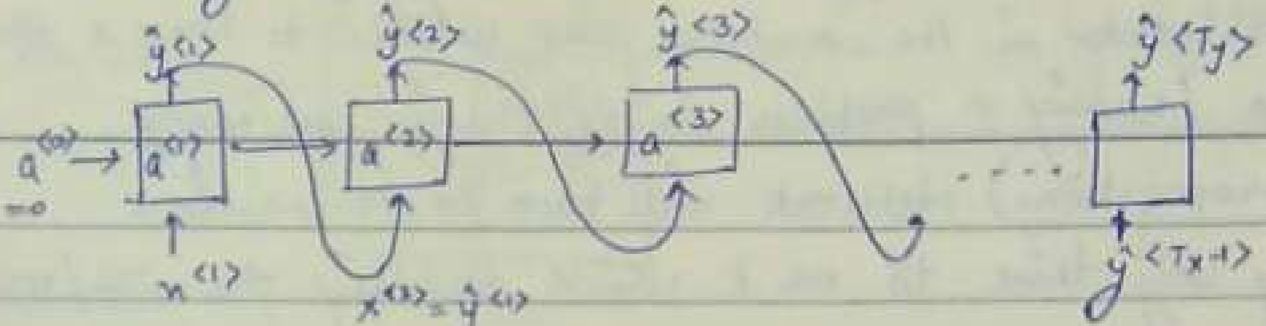
This RNN learns to predict one word at a time given from left to right.

$$L(\hat{y}^{<t>}, y^{<t>}) = -\sum_i y_i^{<t>} \log \hat{y}_i^{<t>}$$

$$\boxed{L = \sum_t L^{<t>}(\hat{y}^{<t>}, y^{<t>})}$$

The diff. time stamps in RNN are nothing but layer.

⇒ Sampling novel sequences:



when $\hat{y}^{<T_y>}$ = EOS, we know end of sentence is reached and we can stop. This can also be done by setting a word limit.

$\hat{y}^{<1>}$ i.e the probabilities help sample the first words. Then the 2nd timestamp helps sample the second words based on $\hat{y}^{<1>}$ i.e the 1st word sampled, and so on.

Also, to avoid UNK from getting sampled, any samples containing it are rejected and resampling is done from remaining vocabulary.
This all will generate a random sentence.

Note Depending on applica^n character level RNN can also be built, with $x^{<t>}$, $y^{<t>}$ as characters and not words. Vocabulary also consists of characters (alphabets, digits, space, punctua^n) in this case.

Pros: no worry about UNK tokens, any word can be generated.
Cons: much longer sequences in character level model and not able to capture long range dependencies (i.e how the earlier parts of sentence also affect the later part of sentence) like the word level model does.
Also, they are computationally expensive to train.

→ Vanishing gradients with RNN:

eg of long range dependencies : The cat which already ate ...., was full.
The cats " " ......, were full.

The RNNs we have seen so far suffer from vanishing gradient decent and as a result later words in the sentence are unable to have a effect on the earlier words (having a problem in upgrading so many weights during backpropogation.) Network will have to remember cat/cats for a very long time to use it while deciding for was/were

Hence, in basic RNN value of $\hat{y}^{<t>}$ is affected mainly by $\hat{y}$ which are close to this $\hat{y}^{<t>}$, hence not capturing long term dependencies.

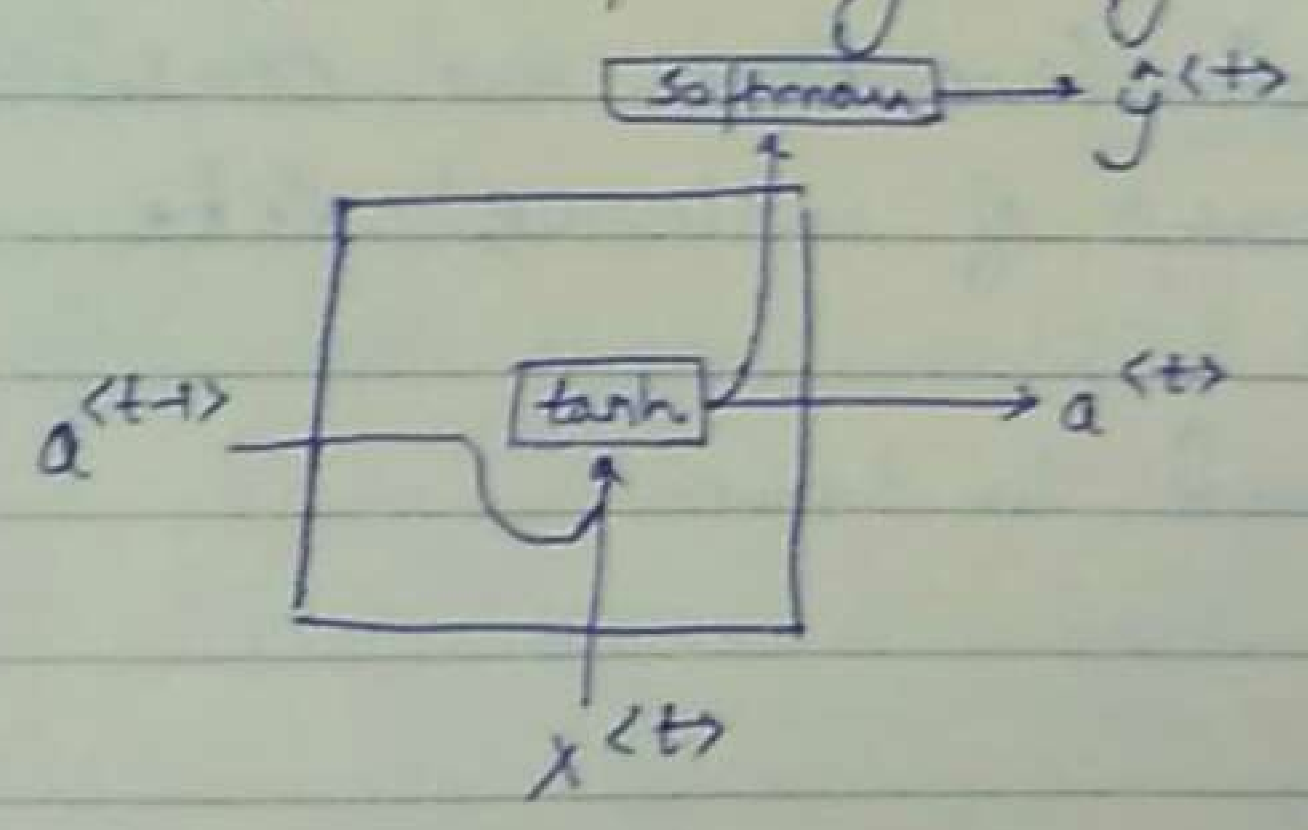
It is easy to spot exploding gradients than vanishing gradients as the parameters just blow up resulting in lot of NaNs (Not a number).

Note solution to exploding gradient decent : Gradient clipping
↓
i.e clipping the gradient vectors whose dimensions have grown beyond a threshold.

* Difficult to solve vanishing gradient decent problem.

⇒ $\boxed{\text{Gated Recurrent Unit (GRU)}}$ {solu^n to vanishing gradient decent}

It helps in capturing long distance dependencies.

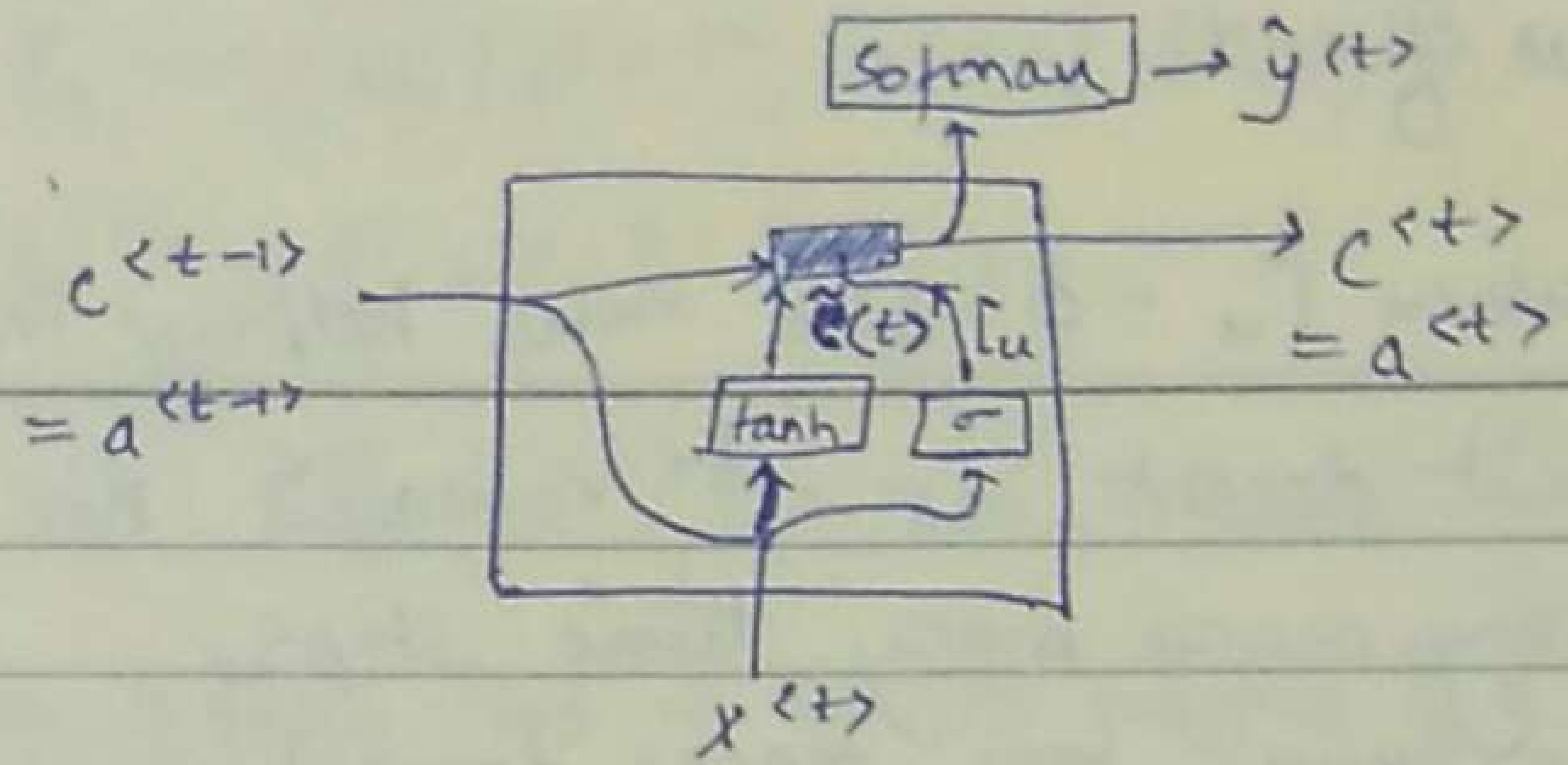


Basic RNN

Note GRU has a c = memory cell to remember whether cat was singular or plural.

for GRU ; $c^{<t>} = a^{<t>}$

memory cell value at timeslot 't' = activa^n output at timeslot 't'.



Γ_u : gate value
↳ 'updated'

GRU picture

At every time step, memory cell value is overwritten by $\tilde{c}^{<t>}$

$$\tilde{c}^{<t>} = \tanh(w_c [\underbrace{c^{<t-1>}}_{c^{<t-1>}}, x^{<t>}] + b_c)$$

$$\Gamma_u = \sigma(W_u [c^{<t-1>}, x^{<t>}] + b_u)$$

{ This is either 0 or 1 most of the time. }

(gate)
u → update
(value b/w 0 and 1)

Supposing we set $c^{<t>}$ as 1 when cat is singular.

e.g. $c^{<t>} = 1$ . . . . $c^{<t'>} = 1$

The cat, which already ate . . . . . ., was full.

$c^{<t>}$ remains to be 1 in $c^{<t'>}$ as well representing that singular is there implying 'chose was'.

Job of Γ_u is to decide when to update these values of $c^{<t>}$. When we came across cat (the subject of the sentence), it was considered the right time to update $c^{<t>}$ and then maybe when it was done being used, no need to memorize it any longer.

→ element wise multiplica^n

So, $c^{<t>} = \Gamma_u * \tilde{c}^{<t>} + (1-\Gamma_u) * c^{<t-1>}$

If $\Gamma_u = 1 \Rightarrow$ set $c^{<t>}$ to a new value $\tilde{c}^{<t>}$

* ⇒ Γ_u was 0 everywhere b/w cat and was, but got = 1 at 'was' indicating to update $c^{<t>}$ at 'was'. $\Gamma_u = 0$ indicated not to

update and not to forget the value of $c^{<t>}$.

from formula $c^{<t>} = c^{<t-1>}$ when $\Gamma_u = 0$ and thus helping in the problem of vanishing gradient decent as $c^{<t>}$ remains the same as $c^{<t-1>}$ even after many many many time steps, allowing the NN to go on even very long range dependencies.

$c^{<t>}$ can be a n-dimensional vector, so, $\tilde{c}^{<t>}$ is also of same dimension. And so will be Γ_u telling which of the n-dimensional bits in $\tilde{c}^{<t>}$ to be updated. 1 bit might be to remember that cat is singular and other might be remembering that food is being talked about.

<u>Full GRU</u>: (commonly used version)

$$\tilde{c}^{<t>} = \tanh(W_c[\Gamma_r * c^{<t-1>}, x^{<t>}] + b_c)$$

tells the relevance of $c^{<t-1>}$ in computing $c^{<t>}$.

$$\Gamma_r = \sigma(W_r[c^{<t-1>}, x^{<t>}] + b_r)$$

(relevance gate)

→ $\boxed{\text{LSTM (long short Term Memory)}}$

- same advantages as that of GRU but even more powerful & general version of GRU.
- In LSTM $a^{<t>} \neq c^{<t>}$

eqn of LSTM:

$$\tilde{c}^{<t>} = \tanh(W_c[a^{<t-1>}, x^{<t>}] + b_c)$$

$$\Gamma_u = \sigma(W_u[a^{<t-1>}, x^{<t>}] + b_u) \quad \{\text{update gate}\}$$

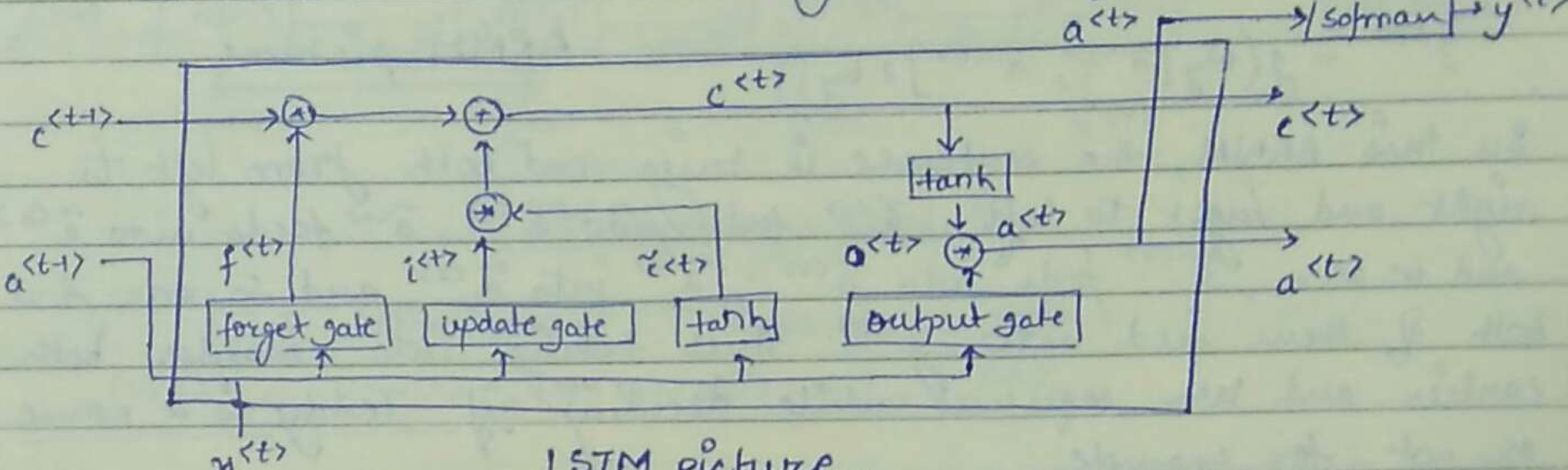
$$\Gamma_f = \sigma(W_f[a^{<t-1>}, x^{<t>} + b_f) \quad \{\text{forget gate}\}.$$

$\Gamma_o = \sigma(W_o[a^{<t-1>}, x^{<t>}] + b_o)$ {output gate}

$c^{<t>} = \Gamma_u * \tilde{c}^{<t>} + \Gamma_f * c^{<t-1>}$

$a^{<t>} = \Gamma_o * (c^{<t>}) = \Gamma_o * \tanh c^{<t>}$

So, 3 gates used in LSTM instead of 1 as in case of GRUs. Hence, its a bit more complicated and places the gates at different places.



LSTM picture.

Note Peephole connection: means that the gate values may depend not just on $a^{<t-1>}$ and $x^{<t>}$ but also on the previous memory cell value i.e $c^{<t-1>}$.

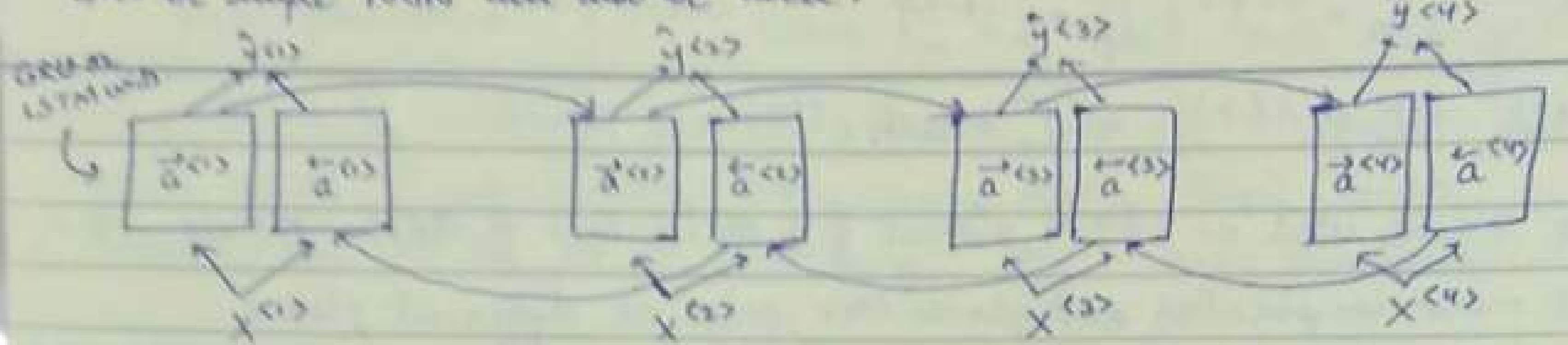
* GRUs came later than LSTMs as a simplified version of the much complex LSTMs.

⇒ $\boxed{\text{Bidirectional RNN}}$

GRUS were much simpler than LSTMs and had less computational cost due to "nce" of 2 gates but LSTMs are more powerful & effective bcoz of "nce" of 3 gates. Due to simplicity GRU is easier to be implemented in big deep networks.

- BRNN allows to take info both from earlier and later sequences, thus helping us with that 'Teddy Bear' example taken earlier.

Bidirectional RNN with a LSTM appears to be commonly used, but GRU or simple RNN can also be used.



$$\hat{y}^{<t>} = g(W_y[\vec{a}^{<t>}, \overleftarrow{a}^{<t>}] + b_y)$$

BRNN picture

In this BRNN, the sentence is being read both from left to right and right to left. $\vec{a}^{<1>}$ feeds into $\vec{a}^{<2>}$, $\vec{a}^{<2>}$ feeds into $\vec{a}^{<3>}$ and so on. $\overleftarrow{a}^{<4>}$ feeds into $\overleftarrow{a}^{<3>}$, $\overleftarrow{a}^{<3>}$ into $\overleftarrow{a}^{<2>}$ and so on. And both of them feed into $y^{<t>}$ thus taking into consider both earlier and later sequence while deciding if Teddy is a name or not for example.

Hence, it uses info potentially from the entire sentence.

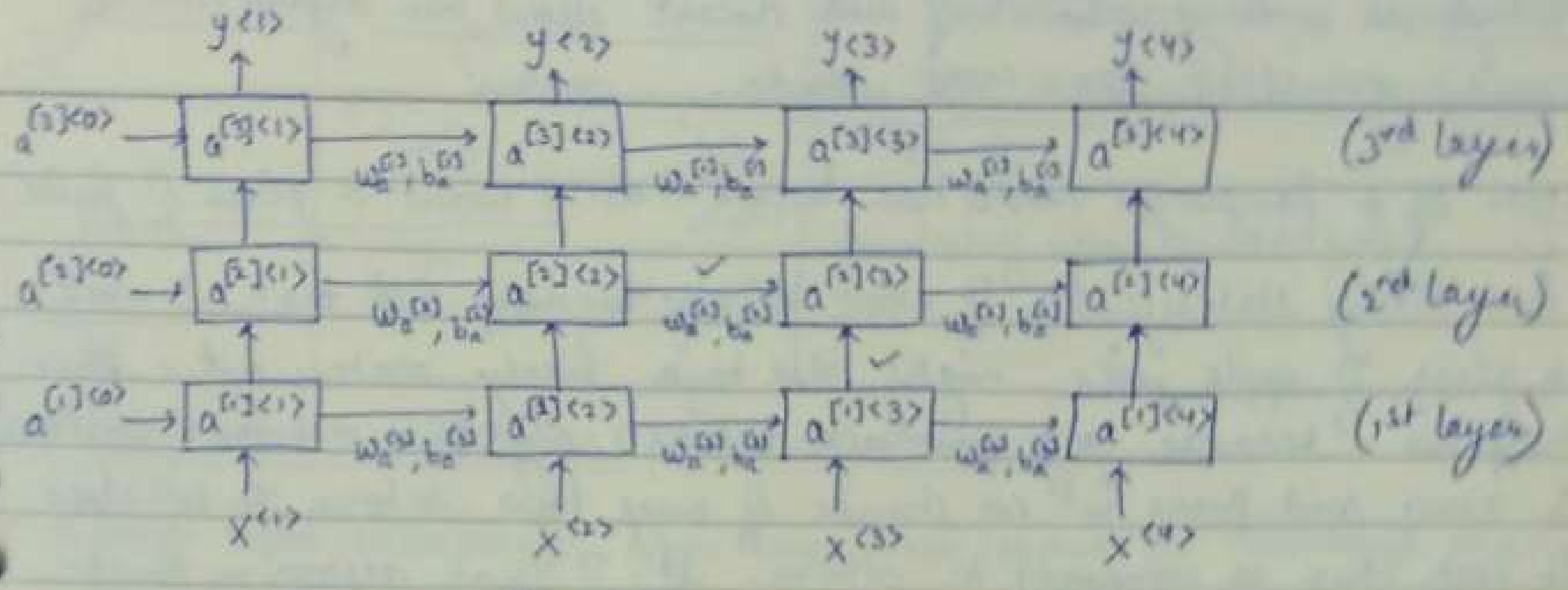
Disadv: entire sequence of data is needed before making any decision

e.g. If a speech recognition system is made using BRNN, then we will have to wait for the person to utter the complete sentence before any processing can be done to reach a conclusion, hence it won't be possible to do real time speech recognition.

Adv: when we have entire sequences available like in NLP problem, BRNN is quite effective.

To get better results we can stack multiple layers of RNN, and get deep RNN versions.

⇒ Deep RNN



3 hidden layer RNN

$a^{[l]<t>}$: t^{th} activation of l^{th} layer of deep RNN.

$$a^{[2]<3>} = g(W_a^{[2]}[a^{[2]<2>}, a^{[1]<3>}] + b_a^{[2]})$$

for RNNs, having 3 layers is quite deep bcoz of the temporal dimension, that already makes the network quite big. More layers can be stacked up to predict $\hat{y}$ but without horizontal connections.

Deep RNNs are computationally very expensive, hence not much hidden layer used as are used in case of CNNs.
Bidirectional RNN can also be converted to deep RNN.

⇒ Word embeddings:
- a way of representing words.

Vocabulary = [a, aaron, ..., zulu, <UNK>]

every word is then representd as one hot encoding with 1 at the posn where it exists in the vocabulary. Represented by O_{zulu} or O_{aaron}

In this one-hot representation, weakness is that it treats each word as a thing unto itself and doesn't allow an algorithm to easily generalise the cross words.

e.g. If a language model has learnt that I want a glass of juice is a likely statement, it won't be able to make out that I want a glass of apple juice might also be a likely statement. This happens because any product between any two one-hot vectors is zero and there is no chance of any two vectors to be close to each other as compared to others. So, it can't recognize that apple and orange are more similar than any other word like king.

Hence, a need for featurised representaⁿ rather than one-hot vector for each of the word

so as to learn a set of features and values for each of them.

e.g.

Features	Man	Woman	King	Queen	Apple	Orange
Gender	-1	1	-0.95	0.97	0.00	0.01
Royal	0.01	0.02	0.93	0.95	-0.01	0.00
Age	0.03	0.02	0.7	0.69	0.03	-0.02
Food	0.04	0.01	0.02	0.01	0.95	0.97
Noun	. . .	. . .	. . .	. . .	. . .	. . .

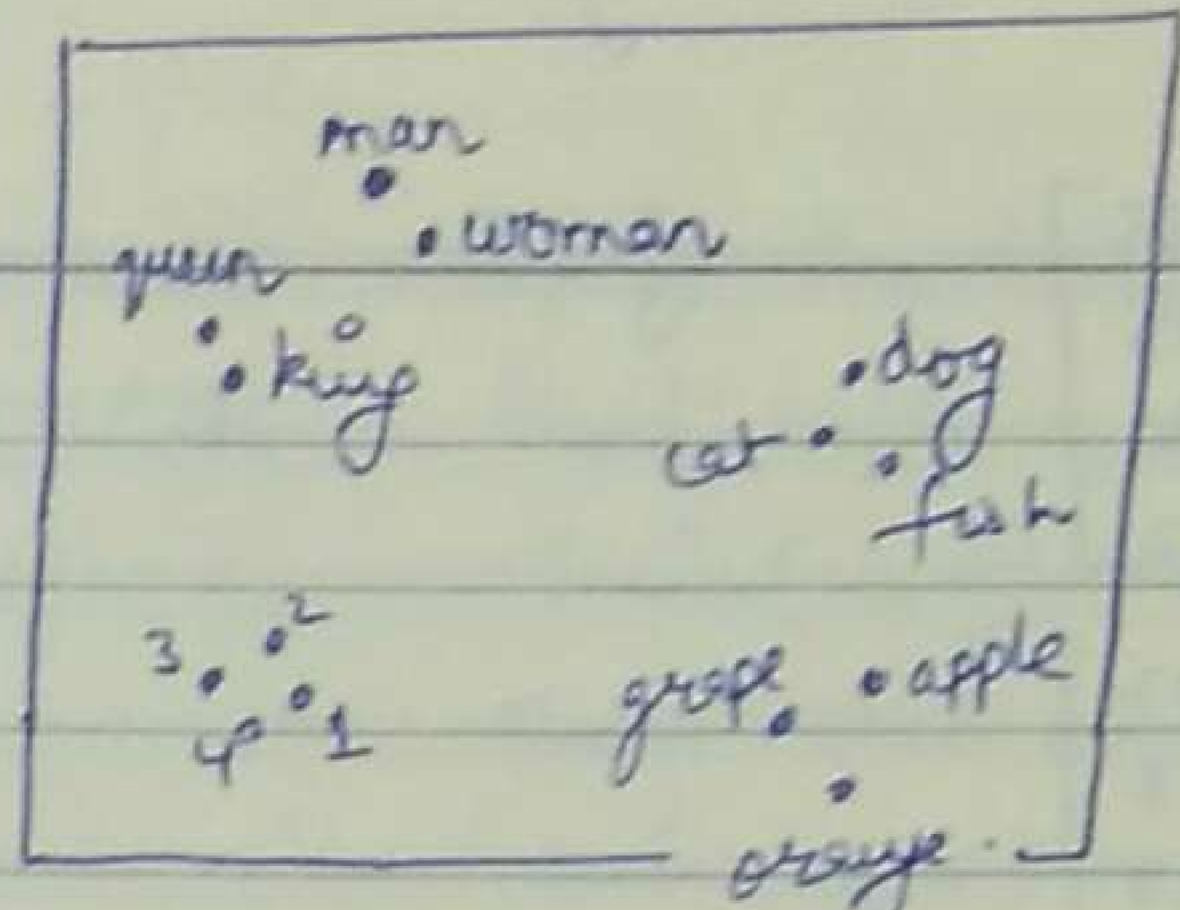
{some words in the vocabulary}

n

So, now each word is a n-dimensional vector and now its easier to notice the relationship b/w words like king and queen, man & woman, orange & apple, etc. So, it generalizes better across different words. Though it is difficult to learn what exactly the word says or represent, but easy to tell which words are more related and which are less.

t-SNE algorithm to convert these n-dimensional vectors to 2D space to visualize them better. It a very complicated & non-linear mapping.

e.g.



{concept of word embeddings}.

→ Using word embeddings for transfer learning:

1) Learn word embeddings from large text corpus or download pre trained word embeddings.
2) Transfer embedding to new task, with smaller training set.

Note One hot vectors are fast than the featurized vectors but give less info.

3) Continue to fine tune the word embeddings with new data, if wanted [just a case of choice)

Word embeddings tend to make the biggest difference when the task, to be carried out has a relatively smaller training set.

Applica^n: Named entity recogni^n, text summarize^n, co-reference resolution, parsing.

Note Transfer learning from A to B is useful when A has lot of data than B.

Word embeddings ~ Picture encoding (as seen in Siamese network)

Only difference is, in word embedding vocabulary is limited but pictures can be any number in case of picture encoding.

- Word embeddings help in knowing analogies e.g. If man : woman :: king : ? .

To solve this we use the word embedding vectors as made on previous pages.

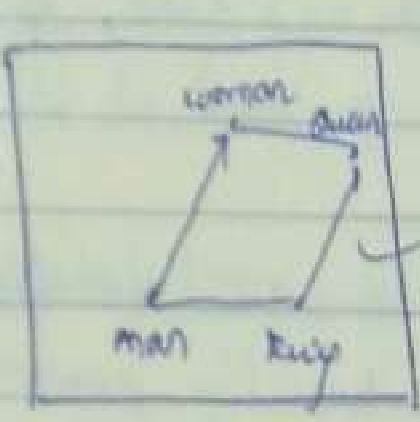
$$e_{man} = \begin{bmatrix} -1 \\ 0.01 \\ 0.03 \\ 0.09 \end{bmatrix} \qquad e_{king} = \begin{bmatrix} -0.95 \\ 0.93 \\ 0.70 \\ 0.02 \end{bmatrix}$$

$$e_{woman} \begin{bmatrix} 1 \\ 0.02 \\ 0.02 \\ 0.01 \end{bmatrix} \qquad e_{queen} = \begin{bmatrix} 0.97 \\ 0.95 \\ 0.69 \\ 0.01 \end{bmatrix}$$

interesting property of these vectors:

$$e_{man} - e_{woman} \approx \begin{bmatrix} -2 \\ 0 \\ 0 \\ 0 \end{bmatrix} \approx e_{king} - e_{queen} \Rightarrow$$ main diff b/w the two is gender.

used to calculate the analogy man : woman :: king : ?



this vector is basically the one representing gender and is approx. equal.

<u>llgm analogy</u>

$$\boxed{\text{find word } w \text{ such that: } \arg\max_{w} \; sim(e_w, e_{king} - e_{man} + e_{woman})}$$

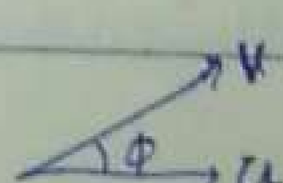
i.e finding a word w that maximises its similarity with $e_{king} - e_{man} + e_{woman}$.

Many $\parallel^{gm}$ analogy relationships will be broken by t-SNE algorithm. One must not count on it to find out e_w.

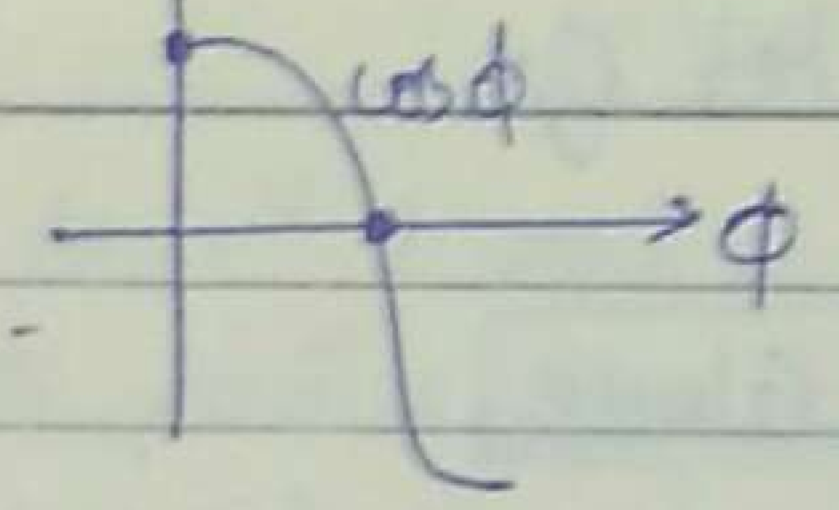
<u>Cosine similarity</u>:

$$\boxed{\cos\phi = sim(u,v) = \frac{u^T v}{\|u\|\,\|v\|}}$$

inner product b/w u and v.



If u and v are similar, their inner product will be a large value.



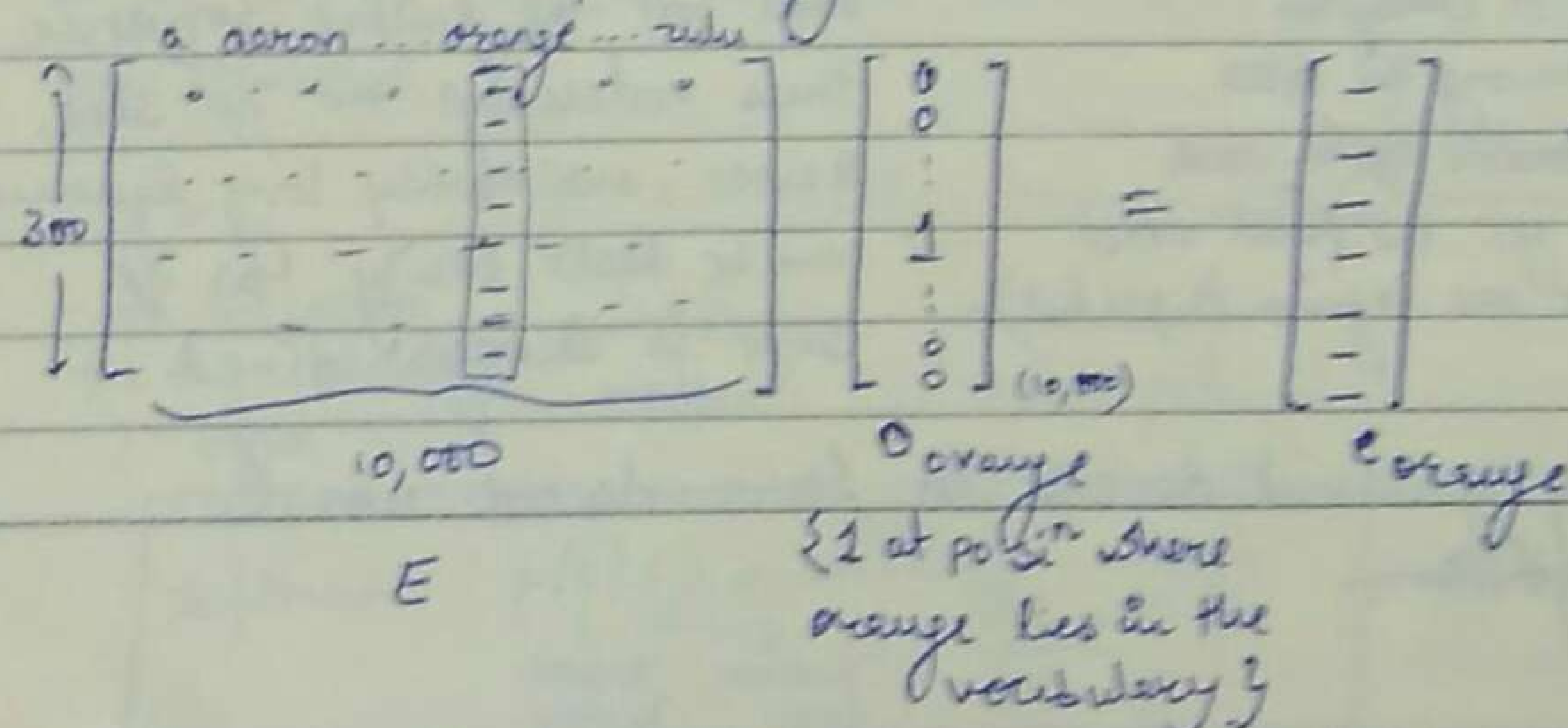
So, if the two words have $\phi = 0$ b/w them that means they are similar and so $sim(u,v) = \cos\phi = 1$. If they are anti then $sim(u,v) = \cos\phi = -1$ and if $\phi = 90°$, $sim(u,v) = \cos\phi = 0$.

Also, $\boxed{\|u-v\|^2}$ is measure of dissimilarity b/w u and v unlike sim(u, v) which is a measure of similarity. sim(u, v) is used more often. 'u' & 'v' are nothing but word embeddings.
All kind of analogies can be learnt by just running a word embedding algorithm on the large ~~text~~ corpus.

⇒ <u>Embedding matrix</u> : (E)

$E = (300, 10{,}000)$ matrix
- 300 → no. of features
- 10,000 → vocabulary size

$E \cdot O_{orange} = e_{orange}$ ⟹ In general $\boxed{E \cdot O_j = e_j}$ = embedding for word j in vocabulary.
- E → embedding matrix
- O_{orange} → one-hot vector
- e_{orange} → embedding vector for orange.



<u>Note</u> But multiplying one-hot vector as a matrix is not practised bcoz most of this matrix is 0. So, a specialised fn is used to just look up to a column in E.

In Keras, the embedding layer simply pulls out the column corresponding to the word (e.g. orange) from matrix E instead of doing matrix multiplication with the one-hot-vector O_j.

⇒ Learning word embeddings: $\boxed{\text{Word2Vec and Glove}}$

Building a neural language model is the small way to learn a set of embeddings. Here we will build a NN to predict the next word in the sentence

I want a glass of orange ______ .
4343 9665 1 3852 6163 6257 → posi^n of these words in the vocabulary.

I $\quad O_{4343} \longrightarrow E \longrightarrow e_{4343}$ → each of this is a 300 dimensional embedding vector.

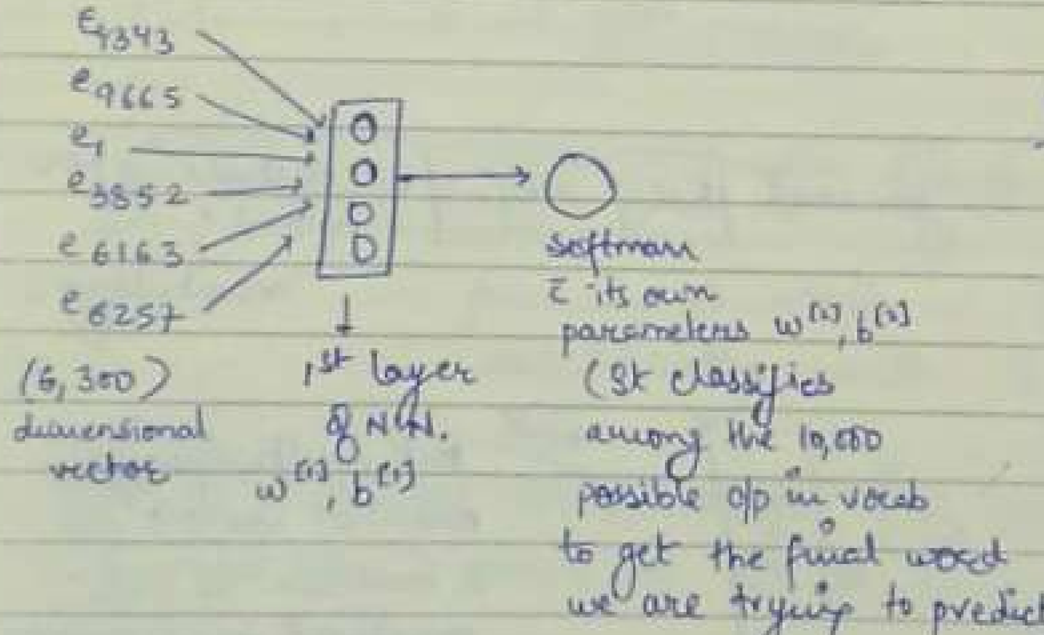
want $\quad O_{9665} \longrightarrow E \longrightarrow e_{9665}$

a $\quad O_{1} \longrightarrow E \longrightarrow e_{1}$

glass $\quad O_{3852} \longrightarrow E \longrightarrow e_{3852}$

of $\quad O_{6163} \longrightarrow E \longrightarrow e_{6163}$

orange $\quad O_{6257} \longrightarrow E \longrightarrow e_{6257}$



Note: A history hyperparameter can be used. It says to predict juice I just need the last 4 words of the sequence and not the entire sequence, thus reducing the i/p size. Hence, arbitrarily long sequences can be dealt easily as i/p size is always fixed.

Perform backprop to apply gradient decent to learn decent word embeddings in this algorithm.

e.g. I want $\underbrace{\text{a glass of orange}}_{\text{content}}$ juice $\underbrace{\text{to go along}}_{\text{content}}$ with my cereal.

↑ target word

So, 1st 4 word and last 4 words are fed into the NN to learn to predict the target word juice.
1 word can also be used while feeding content into NN, that will help train the model in predicting nearby words.
All this helps us in learning good word embeddings.

→ $\boxed{\text{Word to Vec Algo}}$ (simple & more efficient to learn embeddings)

- Skip gram model: Here, unlike the previous algo, the content is not taken to be the immediate words lying before or after the target word but any random word can be chosen as the content for the target word. e.g.

I want a glass of orange juice to go along with my cereal.

Content	Target	
orange	Juice	{word chosen within a 1 word window}
Orange	glass	{ " " " 2 " " }
Orange	my	{ " " " 5 " " }

So, the aim of this supervised learning problem is set to predict a target word for the content within a given window size around the content word.

$$\underbrace{O_c}_{\substack{\text{one-hot vector}\\\text{for the content}\\\text{word.}}} \times \underbrace{E}_{\substack{\text{embedding}\\\text{matrix}}} \longrightarrow \underbrace{e_c}_{\substack{\text{embedding}\\\text{vector for}\\\text{i/p content c.}}} \longrightarrow \overset{\text{softmax}}{O} \longrightarrow \hat{y}$$

$$\boxed{\text{Softmax: } p(\underbrace{t}_{\text{target word}} | \underbrace{c}_{\text{content word}}) = \frac{e^{\theta_t^T e_c}}{\sum_{j=1}^{10k} e^{\theta_j^T e_c}}}$$

θ_t: parameter associated with output t.

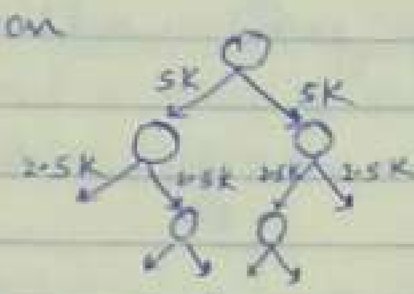
$$L(\hat{y}, y) = -\sum_{i=1}^{10K} y_i \log \hat{y}_i \qquad \hat{y}, y = \text{one hot vector}$$

Skip gram model called so bcoz it takes as i/p one word like orange and then tries to predict some words skipping a few words from the left or the right

Disadv: computational speed bcoz of $\sum_{j=1}^{10K} e^{\theta_j^T e_c}$ term in denominator.

Soln: Heirarchial softmax classifier

(It is generally designed such that more common words are at the top & less common are buried much deeper in the tree)



This solves the problem of summing over entire vocab size in the denominator.

This is classifier to classify if the word is in the 1st 5000 words or the 2nd 5K words and so on i.e binary classification.

Note: we remove frequently occurring words like 'of', 'a', 'the', 'and', 'to', etc bcoz otherwise much effort will be wasted in updating their e_c, so as to focus on the less common but important words.

(under skip-gram):

⇒ $\boxed{\text{Negative Sampling}}$: (modified algo similar to skip gram but more efficient)

Note: CBOW: Continuous Backward model (It considers the surrounding words of the context words to predict the target & has its own set of adv. and disadv.)

Learning problem: Given a pair of words like orange and juice, we need to predict is this a context-target pair?

In this e.g. orange juice is a +ve example.

but orange and king is a -ve example.

So

content (c)	word (t)	target (y)
orange	juice	1
orange	king	0
orange	book	0
orange	the	0
orange	of	0

x = content, word; y = target; k = number of rows labelled 0; training set

to create such pairs, for +ve, target word is taken from a window of some size around the content word while for -ve example, target word is taken at random from the dictionary.

After taking 1 +ve e.g. take k random words from the dictionary for the same content word and label them 0. It's ok if by chance that random word appears to be in the window around the content word. Now this table created will act as X and Y for the supervised learning algorithm. Output is Y i.e 1 or 0 given a pair of words, i.e the algorithm predicts if it thinks the two i/p words were obtained by sampling two words close to each other or not.

k is 2-5 for larger datasets and 5-20 for smaller datasets.

Softmax: $\boxed{p(t|c) = \frac{e^{\theta_t^T e_c}}{\sum_{j=1}^{10k} e^{\theta_j^T e_c}}}$

→ e_c: embedding vector for each possible content word

→ θ: parameter θ for each possible target word.

$$\boxed{P(y=1 \mid c,t) = \sigma(\theta_t^T e_c)}$$

NN:

$O_{orange} * E \rightarrow e_{orange}$ → 10k logistic regression classification problems.

but we will train the models for only k+1 of them.

Hence, this way instead of having one giant 10,000 way Softmax, which is very expensive to compute, we've instead turned it into 10K binary classifica^n problems each of which is quite cheap to compute and at every itera^n we'll train only 5 i.e K+1 of them (i.e k -ve and 1 +ve example) hence further reducing the computa^n cost.

Hence, is called Negative sampling. But how to choose the -ve examples?

→ We can sample the words for a given context word depending on the empirical frequency i.e how often the different words appear in the vocabulary, or use $\frac{1}{\text{Vocab size}}$, sample the -ve examples uniformly at random but that is also very non-representative of the distribu^n of English words.

So, the middle path of the above two solutions : $$\boxed{P(w_i) = \frac{f(w_i)^{3/4}}{\sum_{j=1}^{10K} f(w_j)^{3/4}}}$$

$f(w_i)$ is observed frequency of the word w_i in English language or in the training corpus.

Note,
Hence, this approach is somewhere b/w the extreme of taking uniform distribu^n and the other extreme of just taking whatever was the observed distribution in the training set (bcoz this approach sampled the, of, and, etc which are actually not important words).

⇒ $\boxed{\text{GloVe word vectors}}$ (even simpler embedding algorithm)

Not used as much as word to vec or skip gram but is quite simple.

GloVe : Global vectors

x_{ij} = no. of times i appears in content of j = x_{ji} {acc. to definition}

target (i), context (j)

- Count that captures how often do words i and j appear $\bar{c}$ each other, or close to each other.

if content and target are being selected as +nt within 10 or so word limit of each other
But $x_{ij} \neq x_{ji}$ if target lies before content always or something like that.

GloVe model:

minimize $\sum_{i=1}^{10k} \sum_{j=1}^{10k} f(x_{ij}) \left(\theta_i^T e_j + b_i + b_j' - \log x_{ij} \right)^2$ —— (1)

$f(x_{ij})$: weighting term

(it does not give too much weight to stopwords and gives significant weight to the less frequent but imp words)

$\theta^T e_j \equiv (A\theta_i)^T (A^{-T} e_j)$

$f(x_{ij}) = 0$ if $x_{ij} = 0$ {as log 0 is undefined}

$e_w^{(final)} = \frac{e_w + \theta_w}{2}$ {as θ_i and e_j are symmetric}

In (1), such kind of algo, it can't be guaranteed that the axis used to represent the features will be well-aligned with what might be easily humanly interpretable axis.

→ $\boxed{\text{Sentiment classifica}^{n}}$ (Applica^{n} of these algorithms)

e.g.

x (i/p text)	y (sentiment)
The dessert is excellent.	★★★★☆
Service was quite slow.	★★☆☆☆
Good for a quick meal, but nothing special.	★★★☆☆
Completely lacking in good taste, good service, and good ambiance.	★☆☆☆☆

training dataset to figure out the sentiment of a review.

→ labelled data is not always available for training. But with word embeddings, good sentiment classifica^{n} model can be built just by using only modest-size label training set.

e.g. The dessert is excellent.

y ★★★★☆

The $o_{8928} \rightarrow E \rightarrow e_{8928}$

dessert $o_{2468} \rightarrow E \rightarrow e_{2468}$

is $o_{4694} \rightarrow E \rightarrow e_{4694}$

excellent $o_{3150} \rightarrow E \rightarrow e_{3150}$

$\rightarrow \boxed{\text{Sum or Avg.}} \rightarrow \underset{\text{Softmax}}{O} \rightarrow \hat{y}$

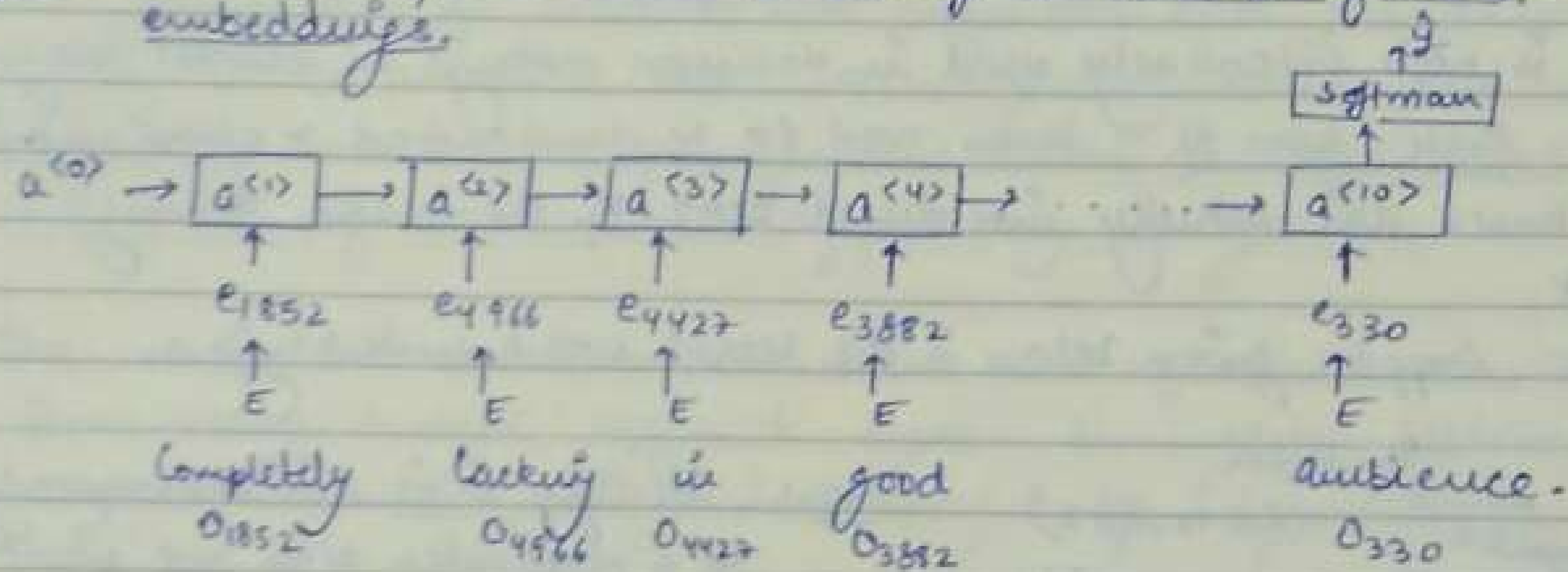
E: already learnt from a large training corpus, different from this problem's labelled training set, thus helps in using external info also.

Softmax: to calculate the probability of getting the different ratings from 1 to 5, hence giving o/p $\hat{y}$.

So, this algo works for reviews of any word length as it averages up all the embedding vectors of the words. But this algo ignored word order, and simply took the average.

Disadv: In the review "Completely lacking in good taste, good service and good ambience" is a negative review but bcoz 'good' appears too many times, this algorithms will probably classify this review as good even though it is actually harsh. This was bcoz word order was not taken into account.

Sol^n: Use RNN for sentiment classifica^n instead of word embeddings.



Many to one RNN architecture

So, considering the order of sequence, this algo will help classify this review as -ve.

Training this algo gives a pretty decent sentiment classifica^n algo and also the use of E (learnt from a large corpus of text from somewhere else) helps the model take decision even on such words which are not +nt in the labelled training dataset.

⇒ Debiasing word embeddings

- means removing gender, ethnicity, sexual orientation bias, and not the technical bias used in ML.

eg Man : Woman :: King : Queen

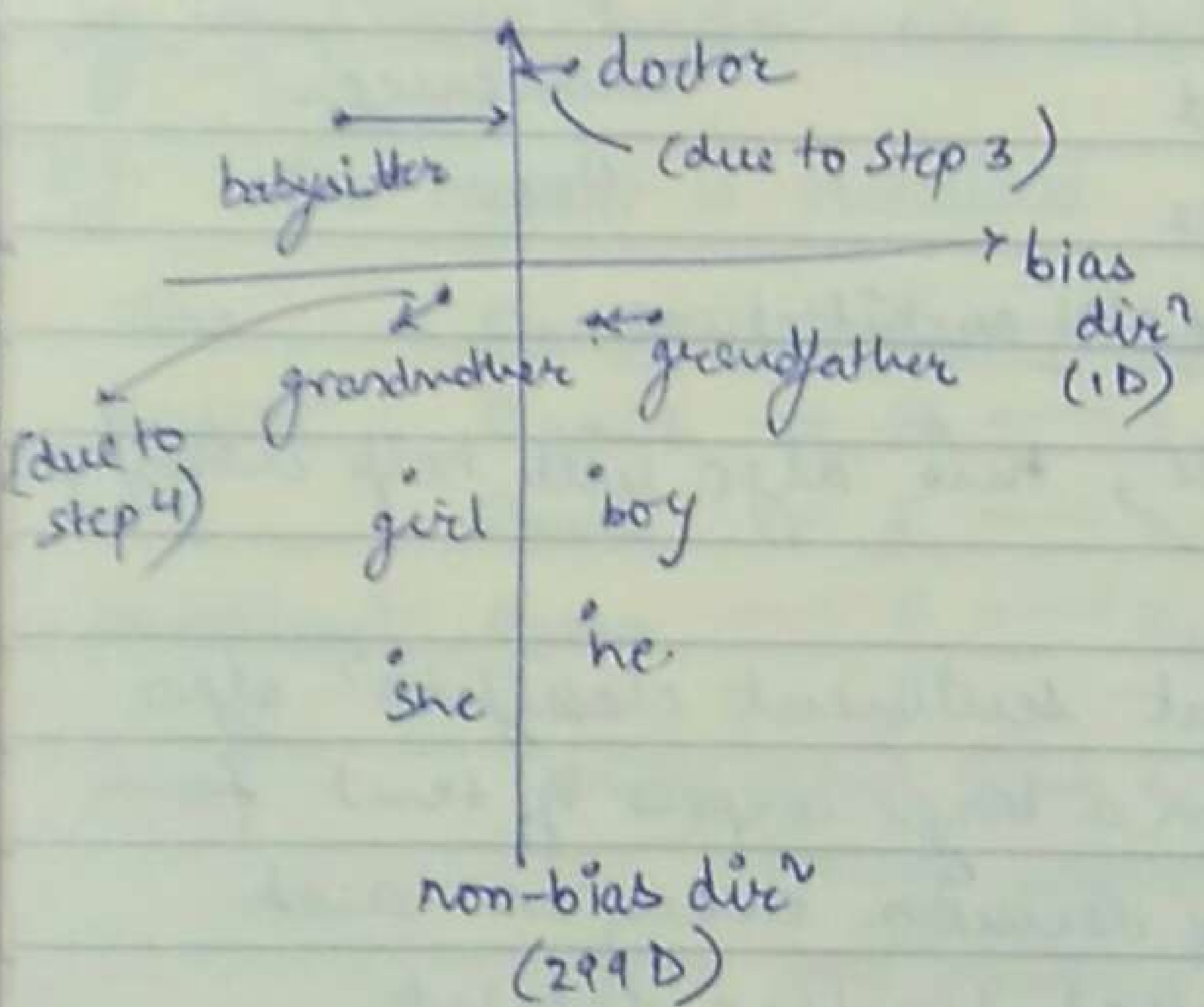
but $\boxed{\text{Man : Computer programmer :: Woman : ?}}$

It was seen that on learning from word embedding the result came out as 'homemaker' and that enforces a unhealthy gender stereotype and it's wrong. An unbiased system would have outputted 'computer programmer' as answer.

Also, ~~Do~~Father : Doctor :: Mother : Nurse were observed.

As ML is now extensively used in decision making in almost every field, these types of biases need to be diminished & eliminated. These biases are actually introduced bcoz of the biased training dataset.

Soln: Suppose given below is the learnt word embedding.



Step 1: Identify bias dirn corresponding to a particular bias we want to reduce or eliminate. Take for example gender bias.

Step 2: $e_{he} - e_{she}$
$e_{male} - e_{female}$
⋮
} take their avg or use SVD (singular value decomposition)

avg. ⇒ horizontal dirn is the bias direction.

Step 3: Neutralization step - for every word that is not definitional, project to get rid of bias.

(means words which do not represent a gender, e.g. he, she, father, lady, etc)

Note: Neutralization is done to eliminate their component in the bias direction. e.g. of gender neutral words like doctor, nurse, etc.

Step 4: Equalization pairs - make sure that pairs like grandfather & grandmother are both exactly the same similarity or exactly the same distance from words that should be gender neutral, such as doctor or babysitter. This happens using linear algebra that will move grandmother & grandfather to a pair of points that are equidistant from the axis in the middle and its effect is that now the distance b/w babysitter and grandfather is same as b/w babysitter and grandmother unlike in the previous case where grandmothers end up babysitting more than grandfathers. This is done for all such pair of words which differ only in gender.

The words that need to be neutralized and equalized can be learn easily through different algos. In English dictionary, very few words are gender specific by definition.

★ $\boxed{\text{Sequence models \& Attention mechanisms:}}$

↓

from machine translⁿ to speech recogⁿ.

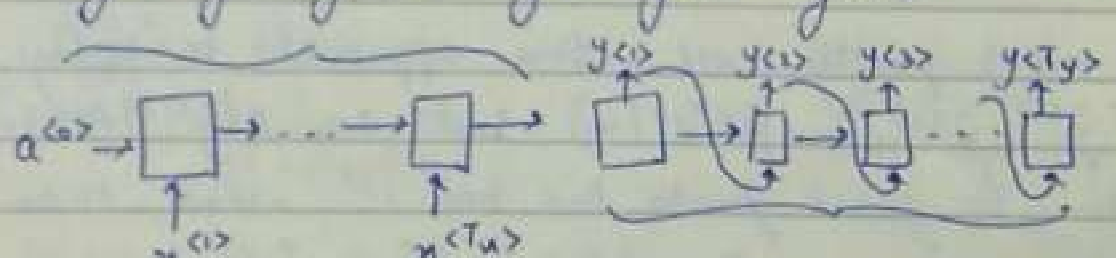
① Sequence to Sequence model -

$x^{<1>}$ $x^{<2>}$ $x^{<3>}$ $x^{<4>}$ $x^{<5>}$

eg Jane visite l'Afrique en septembre.

→ Jane is visiting Africa in September.

$y^{<1>}$ $y^{<2>}$ $y^{<3>}$ $y^{<4>}$ $y^{<5>}$ $y^{<6>}$



'encoder network'

(built as RNN)

'decoder network'

This model works well for given a enough pairs of French & English sentences.

e.g. Image captioning

$$\underbrace{\text{Image as i/p} \xrightarrow{\text{ConvNet}} \text{Image encoded as a feature vector}}_{\text{encoder network}} \underbrace{\xrightarrow{\text{RNN}}}_{\text{decoder network}} \text{caption generated one word at a time.}$$

→ this works well especially if the caption is not too long.

The difference b/w this sequence to sequence model and the earlier text generator model is that now we want the most likely sentence as o/p (like in transla° or captioning) unlike the text generator case of random o/p which were just grammatically correct.

The similarity are there in language model & Machine translation model except that instead of always starting along with the vector of all zeros, machine transla° has an encoded network that figures out some representa° for the input sentence to start off the decoded network. Hence, it is called 'conditional language model'. Now instead of modelling the probability of any sentence, it is now modelling the probability of an English translation as o/p given a French i/p. And the one with max probability is selected as the o/p.
For this Beam Search is used.

'Greedy Search' doesn't work pretty well. This approach basically doesn't focus on $P(\hat{y}^{<1>}, \hat{y}^{<2>}, \ldots \hat{y}^{<T_y>} | x)$ but on $P(\hat{y}^{<1>} | x)$ i.e. best individual words. It first finds the best 1st word, then the best 2nd word and so on rather than finding the probability of best overall sentence. e.g. in Greedy Search:

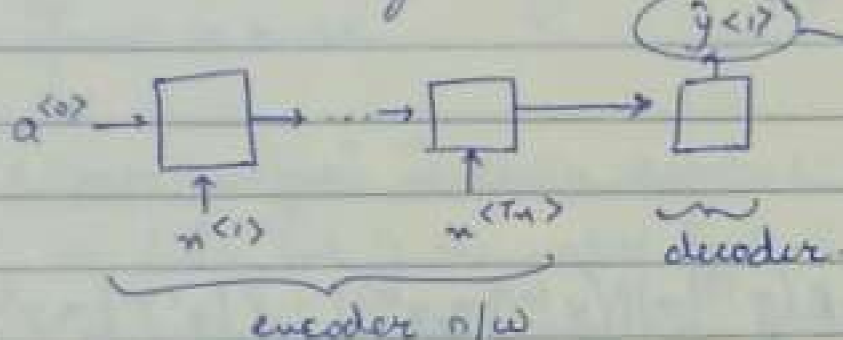
P(Jane is visiting Africa in September) > P(Jane is going to be visiting Africa in September.)

bcoz occurrence of going > visiting in the vocabulary and hence bcoz of the greedy approach, 2nd statement might be selected as the o/p which is good but not as good as the 1st one.

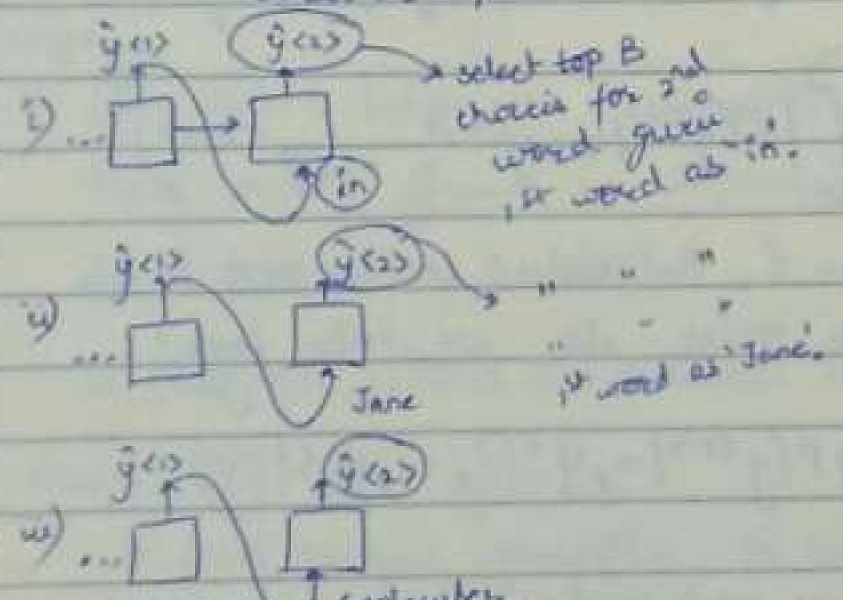
→ <u>Beam Search Algorithm:</u> (approx/heuristic search algo)

B = beam width (no. of guesses per word made by this algo)

It works similar to greedy but instead of finding just the best word, it finds 'B' words from the vocabulary for posi^n 1 in the English translation.



Step1:
Softmax o/p with overall 10,000 (vocab size) possibilities but only B of them which are the most probable are kept in memory.



If the B choices for 1st word were <u>in</u>, <u>Jane</u> and <u>September</u>.

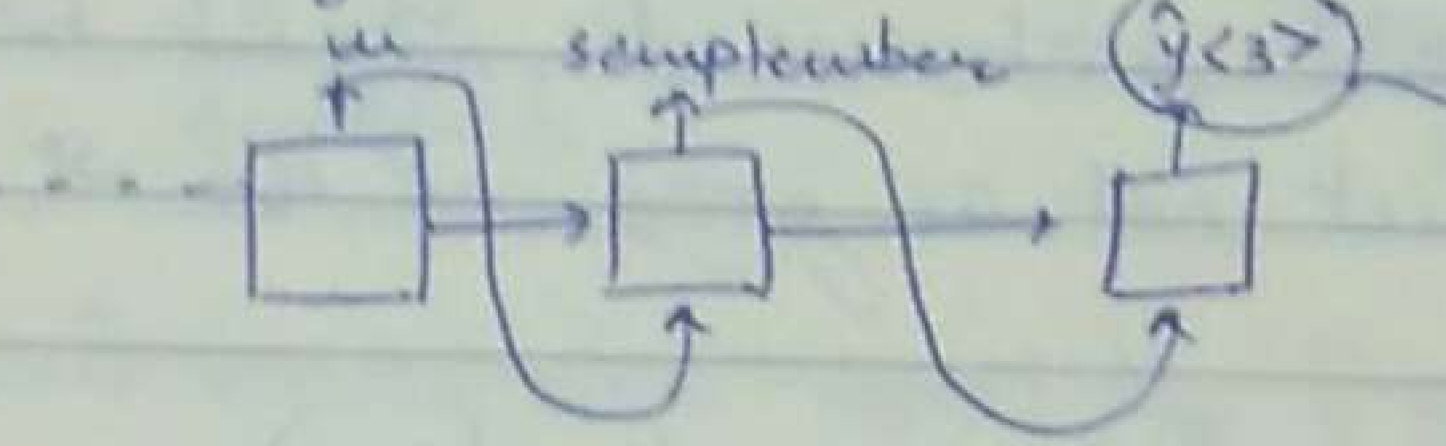
Step2:

For each of the 'B' choices for the 1st word, now the 2nd word is guessed, and this time not only $P(\hat{y}_2)$ is maximised but $P(\hat{y}_1, \hat{y}_2 | x)$ is maximised.

$$\underline{P(\hat{y}_1, \hat{y}_2 | x)} = P(y^{<1>} | x) \, P(y^{<2>} | x, y^{<1>})$$

Now the top 'B' of these possible combina^n of 1st 2 words is stored in memory.

Step 3: If the top B choices for 1st two words were 'jane is', 'in semptember', 'jane visits', then the same procedure as step is applied to predict the best possible 3 choices for the 3rd word for each pair of 1st 2 words.



'B' choices given 1st two words as 'in september'.

$$P(\hat{y}^{<3>} \mid x, \text{"in semptember"})$$

and now similar to step 2 $P(\hat{y}^{<1>}, \hat{y}^{<2>}, \hat{y}^{<3>} / x)$ is maximised. and 'B' choices (top) for the 1st 3 words are stored in memory.

Note Beam search with $B = 1 \Rightarrow$ greedy search algorithm.

- Refinements to beam search:

① Length Normalizaⁿ -

In beam search, $P(y^{<1>}, y^{<2>} \ldots y^{<T>} \mid x) = P(y^{<1>} \mid x) P(y^{<2>} \mid x, y^{<1>}) \cdots P(y^{<T_y>} \mid x, y^{<1>}, \ldots y^{<T_y - 1>})$

$$\text{Beam search} = \arg\max_y \prod_{t=1}^{T_y} P(y^{<t>} \mid x, y^{<1>}, \ldots, y^{<t-1>})$$

bcoz all probabilities are less than 1, the value comes out to be very small and round off errors occur. So, we take log:

$$= \arg\max_y \sum_{y=1}^{T_y} \log P(y^{<t>} \mid x, y^{<1>}, \ldots, y^{<t-1>})$$

so that rounding errors are reduced as now individual probabilities are added and not multiplied.

Drawback is that even this formula works well for only short sentences, so normalizaⁿ is done. As all probabilities are less than 1, their log is -ve and so including more terms (i.e longer sentences) makes it even more -ve. So,

$$\boxed{\frac{1}{T_y^{\alpha}} \sum_{t=1}^{T_y} \log P(y^{<t>} \mid x, y^{<1>}, \ldots, y^{<t-1>})}$$

$\alpha \approx 0.7$ (hyperparameter)

in b/w full and zero normalization.

α can be varied acc to what value gives the best results.

-- How to decide B (beam length)?
Longer B means more possibilities and hence better results but higher computational overhead and memory requirement & hence slow.
There is high gain in results in ↑ing B from 1 to 3 to 10 to 100 but the gains are not that big when B goes from 1K to 3K

Note Unlike BFS, DFS; beam search runs faster but is not guaranteed to find exact max for $\arg\max_y P(y|x)$

- Error analysis in beam search:
to find out if the value of B is the problem or the RNN model used should be redesigned for better performance.

Jane visite l'Afrique en septembre.
Jane visits Africa in September (y^*) - Human translation
Jane visited " last " ($\hat{y}$) - Algorithm translation

To find out where the problem lies, we need to know $P(\hat{y}|x)$ & $P(y^*|x)$ using RNN model.

Case 1: $P(y^*|x) > P(\hat{y}|x)$
⇒ RNN guessed correctly, so beam search is at fault bcoz of which it didn't chose max P o/p & o/p was wrong.

Case 2: $P(y^*|x) < P(\hat{y}|x)$
⇒ RNN guess was wrong and it was not bcoz of beam search that $\hat{y}$ was obtained as o/p.

Going through various such examples, we find out how many times beam search is at fault and how many times RNN is at fault.

Note RNN generates the objective for what beam search is supposed to maximise.

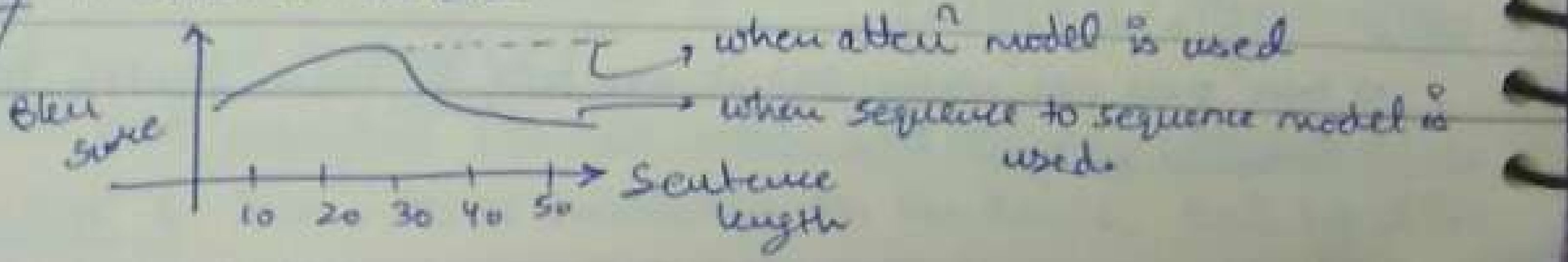
If RNN is found at fault, a deeper layer of analysis can be added and regularization, more training data or modifying N/w architecture can be done. Instead if beam search is found to be at fault, try a greater B.

→ <u>BLEU Score</u> (to help with the problem of having multiple correct o/p or English transla' to a given French sentence.)
↓
bilingual evaluation

It evaluates the o/p of a machine translation system, instead of a human doing so. This is done by looking at each of the words in the o/p and see if it appears in the <u>references</u> given by humans (as test set) This is called <u>precision</u> of the machine transla' output.

② <u>Attention model</u> -

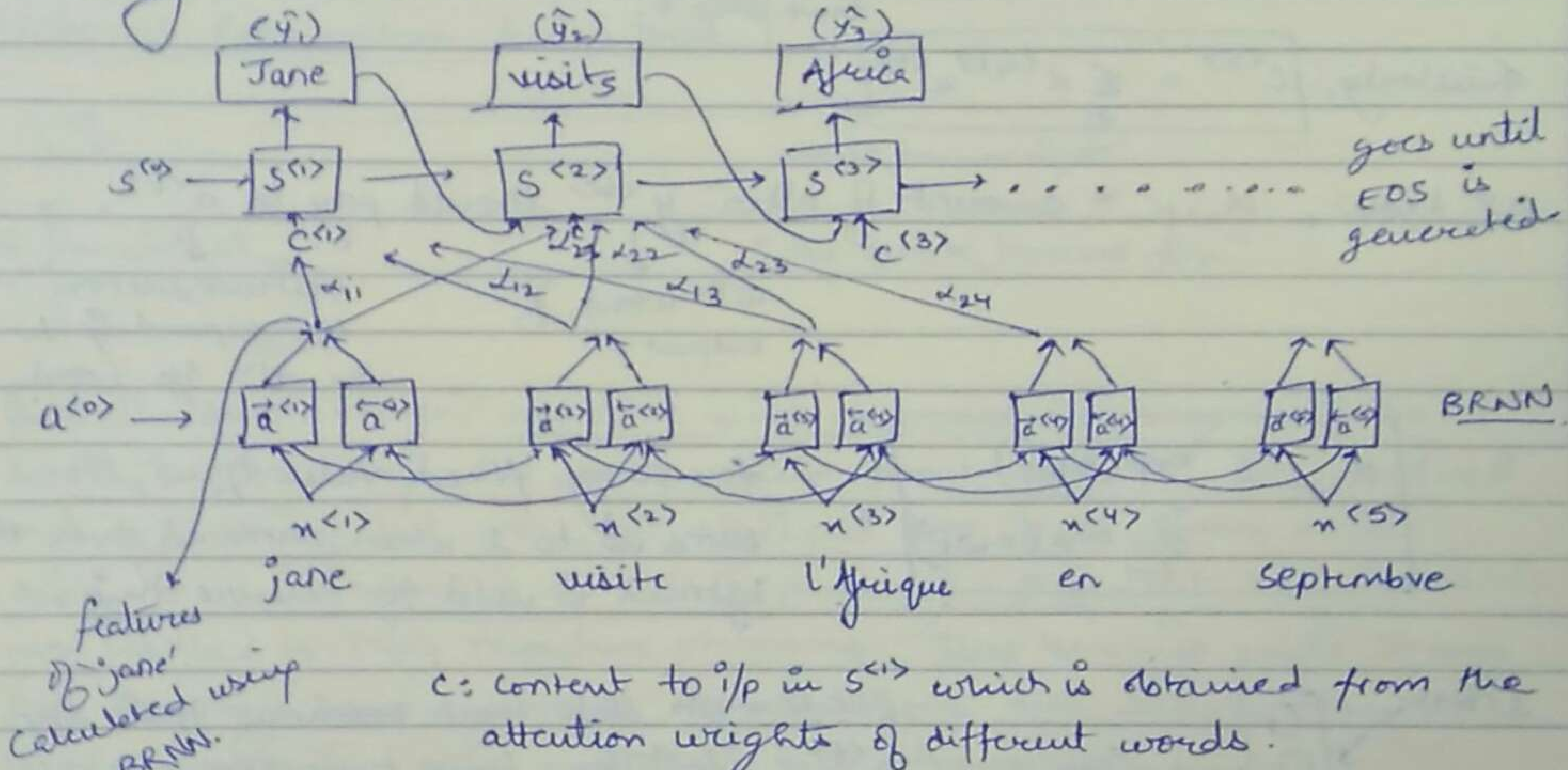
The encoder-decoder mechanism works well for short sentences, as it involves memorizing the whole sentence, encoding it in vector form and then decoding it to give o/p. But in case of long sentences, memorizing the whole sentence at once and then translating it becomes tedious.



for very short sentences, bleu score is less bcoz its difficult to predict the exact words and for very long sentences, bleu score is less bcoz its difficult to memorize them.

Hence, comes Attention models that work like human beings i.e reading the sentence part by part and hence able to translate long sentences.

Attention model actually uses attention weights (α_{ij}) where α_{ij} means how much attention is to be paid to word j while predicting the word i.



c: content to i/p in $s^{<1>}$ which is obtained from the attention weights of different words.

the fwd activa^n $\vec{a}^{<t>}$, the bwd activa^n $\overleftarrow{a}^{<t>}$, $s^{<t-1>}$ contribute in deciding α_{wt} i.e atten^n weight of word t in predicting word.

This allows the model to look only within a local window of the French sentence to predict a particular word of the English translation.

$\underline{a^{<t>}} = (\overrightarrow{a}^{<t>}, \overleftarrow{a}^{<t>})$ = feature vector for time step t.

$$\sum_t \alpha^{<1,t>} = 1$$ (None of the α is -ve)

$$C^{<1>} = \sum_t \underbrace{\alpha^{<1,t>}}_{\text{attention weights}} \underbrace{a^{<t>}}_{\text{feature vector corresponding to i/p word at time step } t.}$$

Similarly, $\boxed{C^{<2>} = \sum_t \alpha^{<2,t>} a^{<t>}}$

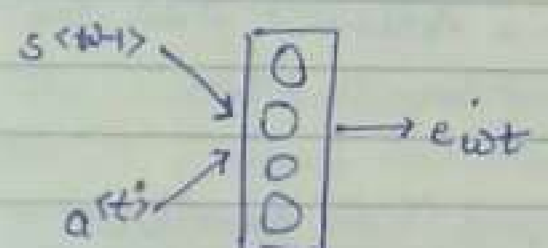
we know, $\alpha_{\omega t}$ = amount of atten^n $y^{<\omega>}$ should pay to $a^{<t>}$.

$y^{<\omega>}$: ω^{th} word of output.

$a^{<t>}$: feature vector corresponding to the t^{th} i/p word.

$$\boxed{\alpha_{\omega t} = \frac{\exp(e_{\omega t})}{\sum_{t=1}^{T_x} \exp(e_{\omega t})}}$$

for every fixed value of ω, it sums up to 1 when summed over t. <u>softmax is used to ensure this.</u>



$s^{<\omega-1>}$: NN state from previous time step

$a^{<t>}$: features from time step t of i/p.

Obviously, $e_{\omega t}$ depends on $s^{<\omega-1>}$ & $a^{<t>}$.

Train this NN to get $e_{\omega t}$ corresponding to diff s and t, using gradient decent. It is found that it works pretty well in deciding the correct atten^n weights.

Note $\boxed{\text{total no. of atten}^n \text{ parameters} = T_x T_y}$

(so, quadratic cost of this algo)

$a^{<t>} = (\vec{a}^{<t>}, \overleftarrow{a}^{<t>})$ = feature vector for time step t.

$\sum_t \alpha^{<1,t>} = 1$ (None of the α is -ve)

$$C^{<1>} = \sum_t \underbrace{\alpha^{<1,t>}}_{\text{attention weights}} \underbrace{a^{<t>}}_{\text{feature vector corresponding to i/p word at time step } t.}$$

Similarly, $\boxed{C^{<2>} = \sum_t \alpha^{<2,t>} a^{<t>}}$

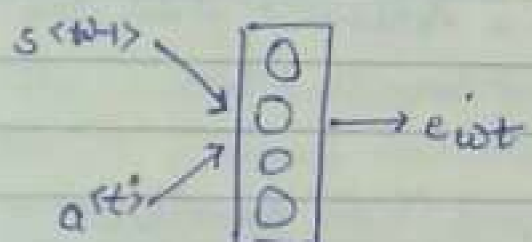
we know, $\alpha_{\omega t}$ = amount of atten^n^ $y^{<\omega>}$ should pay ~~to~~ $a^{<t>}$.

$y^{<\omega>}$ → ω^{th} word of output.

$a^{<t>}$ → feature vector corresponding to the t^{th} i/p word.

$$\boxed{\alpha_{\omega t} = \frac{\exp(e_{\omega t})}{\sum_{t=1}^{T_x} \exp(e_{\omega t})}}$$

for every fixed value of ω, it sums up to 1 when summed over t. softmax is used to ensure this.



$s^{<\omega-1>}$: NN state from previous time step

$a^{<t>}$: features from time step t of i/p.

Obviously, $e_{\omega t}$ depends on $s^{<\omega-1>}$ & $a^{<t>}$.

Train this NN to get $e_{\omega t}$ corresponding to diff s and t, using gradient decent. It is found that it works pretty well in deciding the correct atten^n^ weights.

Note: $\boxed{\text{total no. of atten}^n \text{ parameters} = T_x T_y}$

(So, quadratic cost of this algo)

- Attenⁿ models are also applied in <u>image captioning</u>, using the same kind of architecture as just discussed i.e <u>paying atten^n to only some parts of picture ~~this~~ at a time while predicting</u> the caption.
- It is also used to get <u>normalized dates</u>.

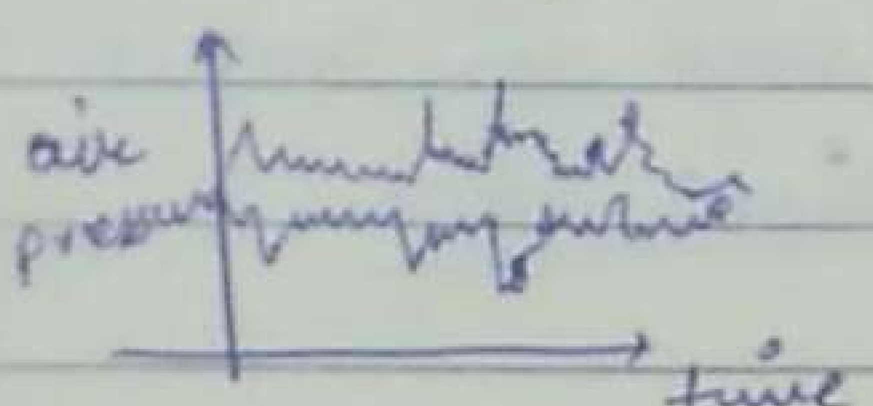
<u>e.g.</u> July 20th 1969 → 1969-07-20

23 April, 1564 → 1564-04-23

⇒ $\boxed{\text{Speech Recognition - Audio Data.}}$

x ⟶ y

audio clip — text transcript



"the quick brown fox".

Initially speech recogniⁿ <u>was done using phonemes</u> i.e recognizing the sound and outputting the corresponding phonemes to it. It involved hand engineering but were not efficient bcoz if a person said quick, it was outputted as kwik (phonem). Different kind of sounds were labelled to their respective phonems. Deep learning made it easy thus removing hand-engineered representations and simply training the n/w with audio clips and their text transcripts

- <u>Attention Models</u> can be used for speech recogniⁿ.
- <u>CTC cost for speech recognition</u> is another method.

↓

connectionist temporal classification

<u>Note</u> In speech recognition the number of output time steps is much lesser than the input time steps.

e.g. If in a 10 sec video, features come at 100 samples per second, then a 10 sec audio will have 1K features on i/p. But the o/p might not have a thousand alphabets / characters. So, what CTC does is it allows the RNN to generate an output like:

(ttt_h_eee___ ␣ ␣ ---qqq---) and it's considered

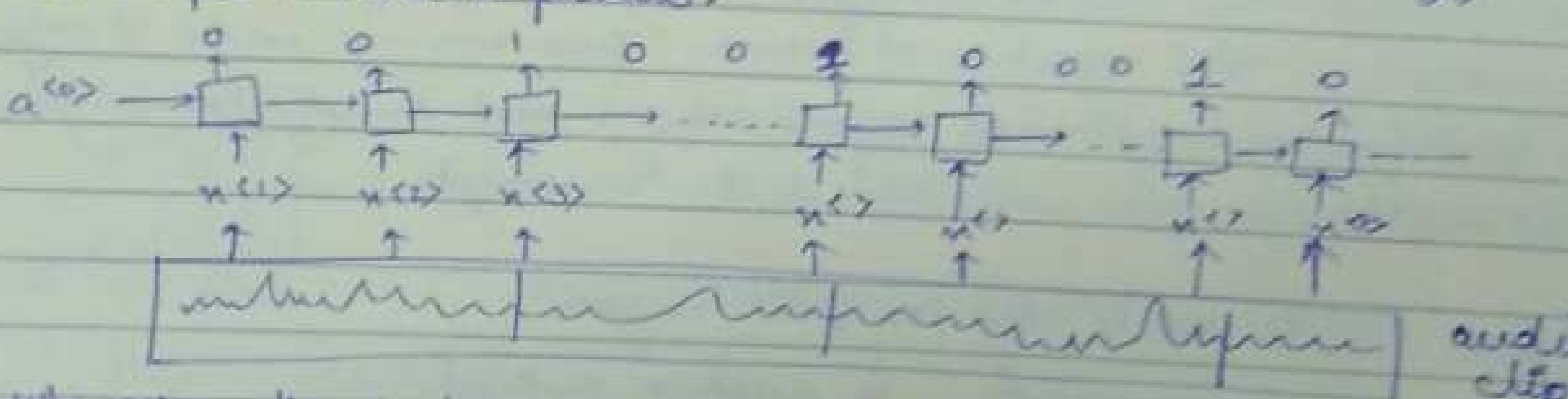
blank
space

the correct output for the audio i/p "the quick". The rule of CTC is to collapse output characters not separated by "blanks". So the sequence is collapsed into "the q" which allow the NN to have 1000 outputs by repeating characters and still end up with a much shorter output text transcript.

CTC was needed bcoz the architecture of RNN used in speech recognition was the one with equal i/p and o/p time steps.

→ Trigger word detection:

'Trigger word' is actually like 'Hey Siri' for Siri, 'Ok Google' for Google, 'Mahuis' for me and likewise. This trigger word helps initiate a process.



whenever the system encounters the points where someone has just finished saying the trigger word, then in the training set target labels are set to be zero for everything before that point and 1 right at that point. Later on if trigger word is said again, then target label is again set to 1 right after the 0's. The only disadvantage is that it creates an imbalance with lot of zeroes and very few 1s. So instead of setting just a single time step equal to 1, we can output 1 for a time duration before reverting back to zero. This evens out the ratio of 1s to 0s slightly to some extent.

DOUBTS: