

Sequence Modeling

why sequence models?

→ We need a sequence model when your data has an order/time dependency.

Instead of,

Input $\rightarrow$ Output,

we have

Input sequence $\rightarrow$ Output sequence (for a single output)

A standard NN takes a fixed size input & ignores order,

so you need a model that remembers past information & processes things step by step.

Types of sequence problems

1) Sequence $\rightarrow$ Sequence (X & Y both are sequence)
 input: Sound over time (sequence)
 output: words (sequence)
 lengths can be different.

2) Sequence $\rightarrow$ Single output

ex $\rightarrow$

$\left\{ \begin{array}{l} X = \text{sequence} \\ Y = \text{single value} \end{array} \right\}$

Input: "There is nothing to like in this movie"

Output: A rating

3 > Single Input → Sequence Output

ex → Music generation

input: A genre

output: Sequence of notes

input: "Jazz"

output: "Sequence of notes"

4 > Sequence → Sequence (same length)

ex → NER (Named Entity Recognition)

input: "Elon Musk founded SpaceX"

[Person] [Person] [O] [organization]

Each input word → one output label

5 > Sequence → Sequence (special domains)

ex → DNA Analysis

Input:

[ACGTGTAAT ...]

Output: [labels for each part protein/not protein]

6 > Sequence → Sequence (video)

ex → Activity recognition

Input: Frames over time

Output: "running", "jumping" etc

Notation:-

ex → Named identity recognition

→ here what you need to do is to label whether a character is a name or not.

X: Harry Potter & Hermione Granger invented a new Spell

$x^{(1)}$ $x^{(2)}$ $x^{(3)}$ $x^{(4)}$ $x^{(5)}$ $x^{(6)}$ $x^{(7)}$ $x^{(8)}$ $x^{(9)}$

Y: | | 0 | | | 0 0 0

$y^{(1)}$ $y^{(2)}$ $y^{(3)}$ $y^{(4)}$ $y^{(5)}$ $y^{(6)}$ $y^{(7)}$ $y^{(8)}$ $y^{(9)}$

~~So, here what you can do is label all the example~~

$T_x^{(i)}$ = "Total no of words in the input sequence of the i^{th} example" = 9

$T_y^{(i)}$ = "Total no of words in the output sequence of the i^{th} example" = 9

$x^{(i)}_{(t)}$ = "The t^{th} word in the sequence of the i^{th} example"

here $x^{(i)}_{(t)} = \text{Hermione}$

$y^{(i)}_{(t)}$ = "The t^{th} label in the sequence of the i^{th} example"

Representing words.

X: Harry Potter & Hermione Granger invented a new spell

$x^{(1)}$ $x^{(2)}$ — — — — — $x^{(9)}$

→ To represent the word in a sentence you come up with a vocabulary or dictionary

it's just a bunch of the words that you will have in your input sequence.

Vocabulary

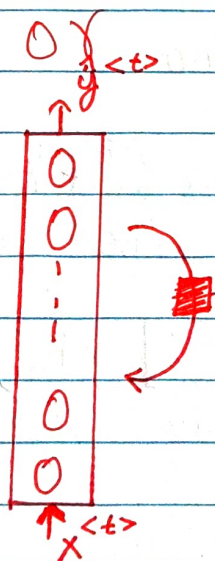
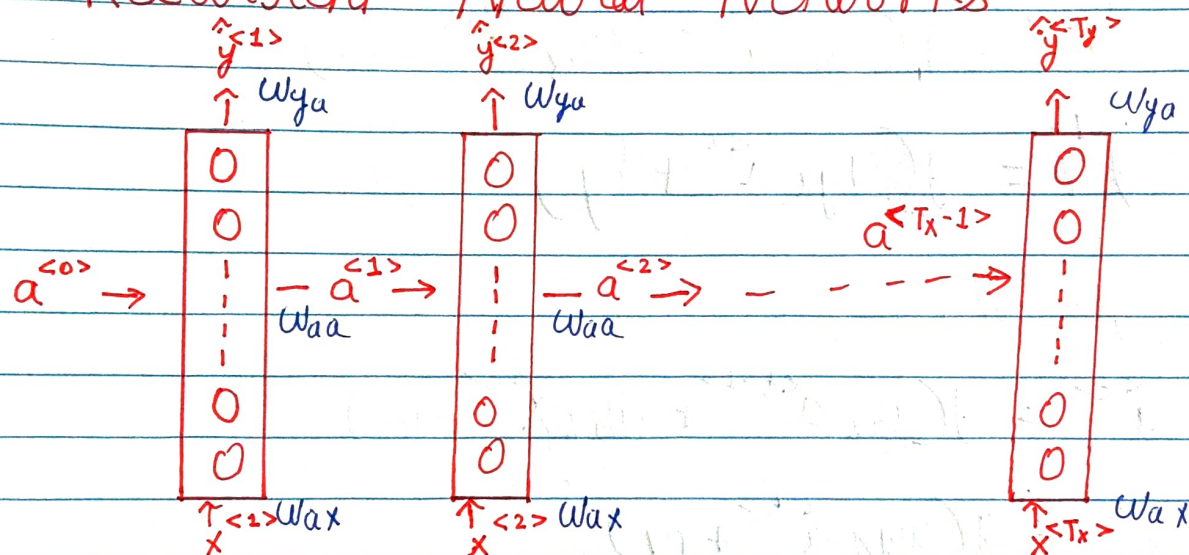
a	1
common	2
...	...
zulu	10000

So now you can ~~use~~ use one-hot encoding on all the ~~char~~ words of your input sequence & get the inputs for NN

Why not standard network for sequence modeling?

- Input & output can be different lengths in different examples.
- Doesn't share features learned across different position of text (No past data is learned by the neural Net)

Recurrent Neural Networks



Both of the diagrams are of Recurrent Neural Nets, the one above is just the unrolled version of the one below.

→ The recurrent Neural Net gets fed the input or the current element of the sequence plus the information of the past sequence as activations.

$a^{<0>} = \vec{0}$, Here the first activation will be the vector's of all zeros

So, $g_1() \rightarrow \tanh$ or relu for activations $a^{<1>}$
 $g_2() \rightarrow \text{sigmoid}$ or softmax

$$a^{<1>} = g_1(W_{aa} a^{<0>} + W_{ax} x^{<1>} + b_a)$$

$$\hat{y}^{<1>} = g_2(W_{ya} a^{<1>} + b_y)$$

in general,

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

$$\hat{y}^t = g(W_{ya} a^{<t>} + b_y)$$

Simplified RNN notation

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

this can be written as :-

$$g(W_a [a^{<t-1>}, x^{<t>}] + b_a)$$

So, here what we do is,

we stack the matrices W_{aa} & W_{ax} to make a new matrix " W_a ".

$$[W_{aa} \parallel W_{ax}] = W_a$$

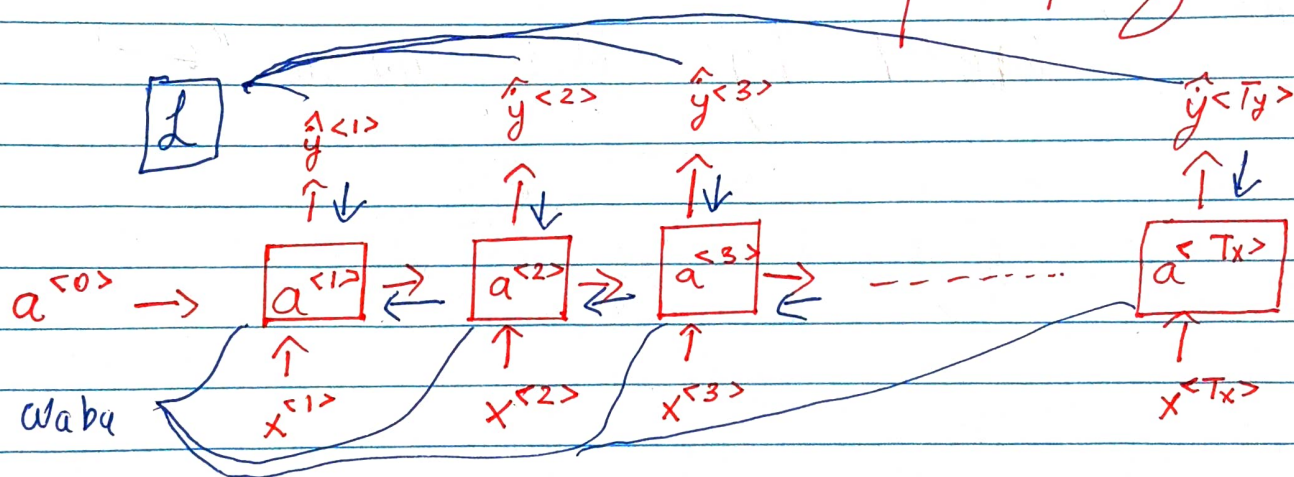
assume if W_{aa} 100×100 & W_{ax} $100 \times 100,000$

then , W_a $(100, 10|100)$

& , $[a^{<t-1>} \ x^{<t>}] \Rightarrow$ will be a vector stacked by both the vectors

$$\begin{matrix} \uparrow 100 \\ \downarrow 10|100 \end{matrix} \begin{bmatrix} a^{<t-1>} \\ x^{<t>} \end{bmatrix} \begin{matrix} \uparrow 100 \\ \downarrow 100,000 \end{matrix}$$

Forward & Backward propagation



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

$$Loss_{\text{cost}} = L(\hat{y}, y) = \sum_{t=1}^{T_y} L^{<t>}(\hat{y}^{<t>}, y^{<t>})$$

$\rightarrow$ Here in forward propagation we only use a fixed set of activation for the prediction in every element of the sequence. which is (W_a, b_a) & generate $\hat{y}$. denoted by " $\rightarrow$ "

in backward pass here we calculate the loss on each element of the example,

by using binary cross entropy & then sum all the losses by elements to get loss on that one training example,

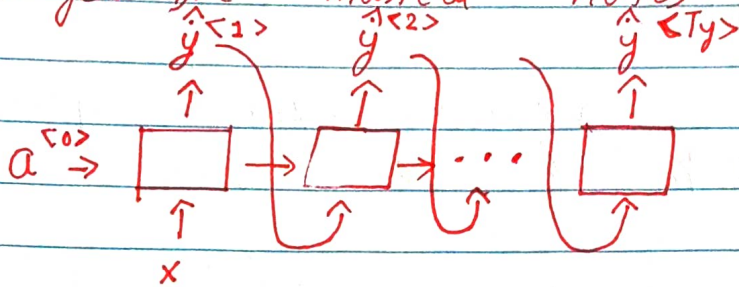
& then we do backward propagation & we go backward in the sequence & update parameters,

this is why it's also called backward propagation through time, denoted by the arrow " $\leftarrow$ ".

Different types of RNN's

→ One to many

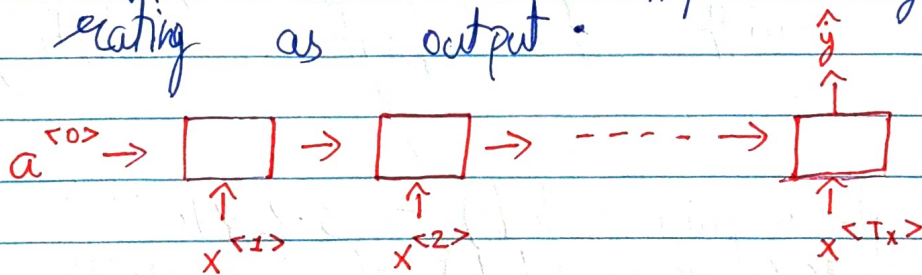
ex → music generation, you give the input as genre & it gives you the musical notes.



One - to - ~~one~~ many

→ Many - to one

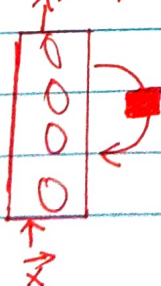
ex → Sentiment classification, you give a movie review as an input & get a star rating as output.



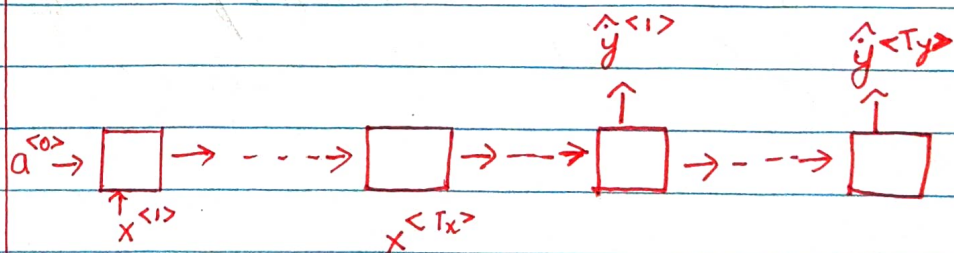
→ Many to many

ex → the one we are

doing right now
"name entity recognition"



ex → Machine translation



→ Here, you enter the words in sequence & get the translation in a sequence

Language model & Sequence generation.

Here is an example,

→ speech recognition,

- i → The apple & pair salad
- ii → The apple & pear salad.

As so above 2 sentences example you can see that both of the sentences sound the same.

So, how does the model decides which one to pick.

The language model computes the probability of both sentence occurring & then chooses the one with the high probability of occurrence

$$\begin{aligned} P(\text{The apple \& pair salad}) &= 3.2 \times 10^{-13} \quad \text{-(i)} \\ P(\text{The apple \& pear salad}) &= 5.7 \times 10^{-10} \quad \text{-(ii)} \end{aligned}$$

Here the probability of $ii > i$ so we pick (ii)

Building a language model

Training set:- large corpus of english text

Let's say you pick a sentence -:

→ Cat averages 15 hours of sleep a day. $\langle \text{EOS} \rangle$
 $y^{(1)}$ $y^{(2)}$ $y^{(3)}$ $y^{(4)}$

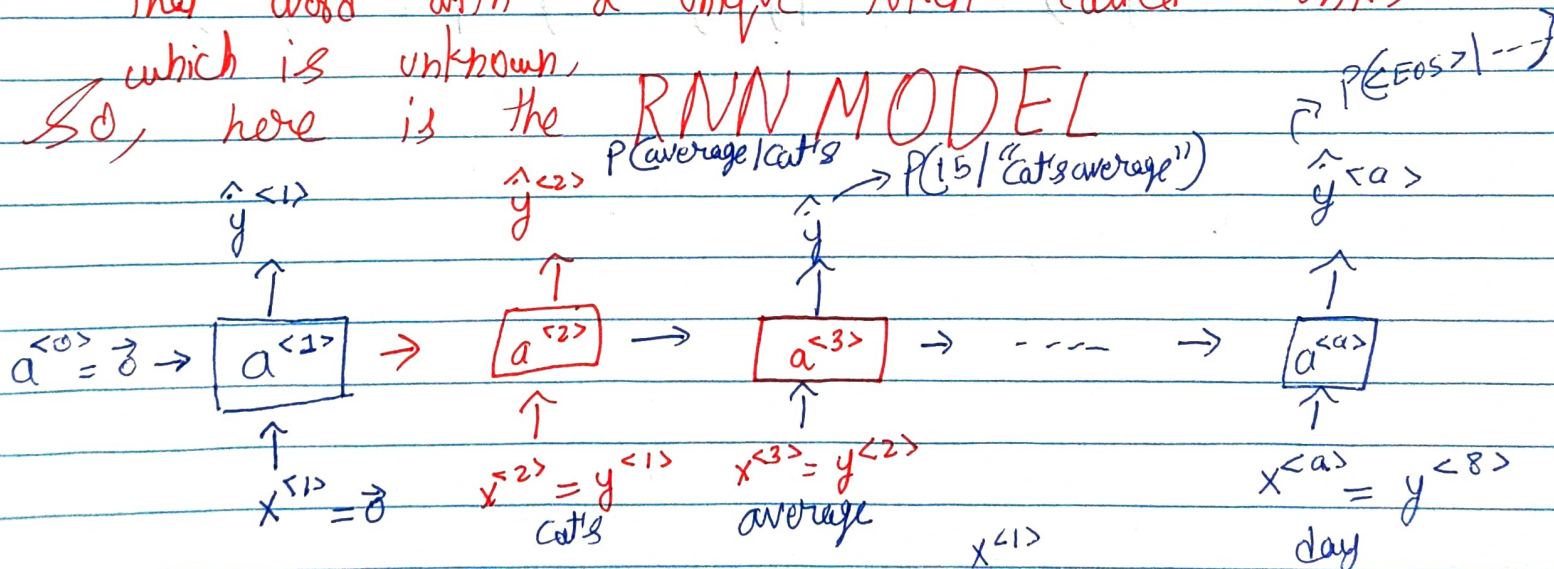
→ here $\langle \text{EOS} \rangle$ means end of sentence
a special token to denote the string has ended

→ so, what if some of the words in your training set are not in your vocabulary or are not in the token,

that word with a unique token called " $\langle \text{unk} \rangle$ " which is unknown, you can replace

So, here is the

RNN MODEL



→ So, you let give the first $a^{(1)}$ as zero vector, for the neural net to come up with some probability for some word,

then in $x^{(2)}$ we feed it the first correct

word & then try to get the probability of the next word,

after that we feed it the activation & the first input & then get the probability of the correct word given these 2 last words,

we do it till the last token

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

$$\frac{1}{2} L = \sum_t L^{<t>}(\hat{y}^{<t>}, y^{<t>})$$

So the given a new sentence,

$\hookrightarrow (y^{<1>}, y^{<2>}, y^{<3>})$

to figure out the probability of the sentence would be,

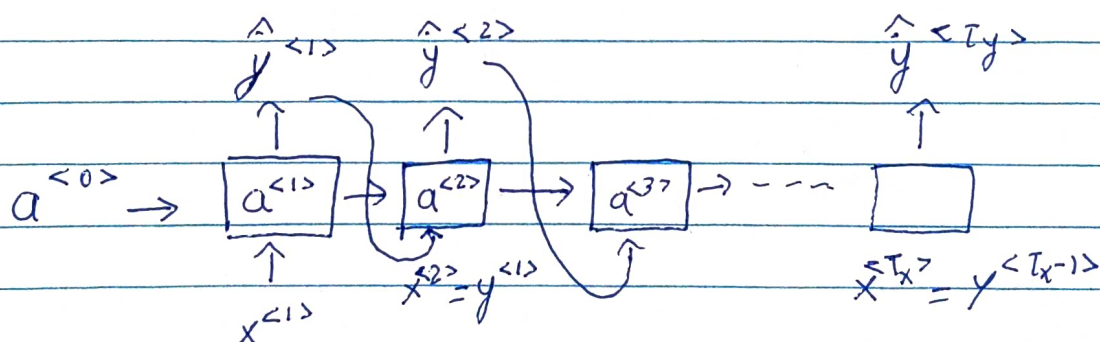
$$\Rightarrow P(y^{<1>}) * P(y^{<2>} | y^{<1>}) * P(y^{<3>} | y^{<1>}, y^{<2>})$$

Sampling Novel sequences

So, to generate the novel sequences from an RNN language model, so you can input any random first input or just vectors of zero

the model will give you some probability & then you can sample from that to get a word after getting that word you can give the model as input that sampled word & keep repeating that till you get the token "<EOS>" or have a ~~little~~ limit defined for the number of characters a model can generate.

Sampling :-



Character level language model

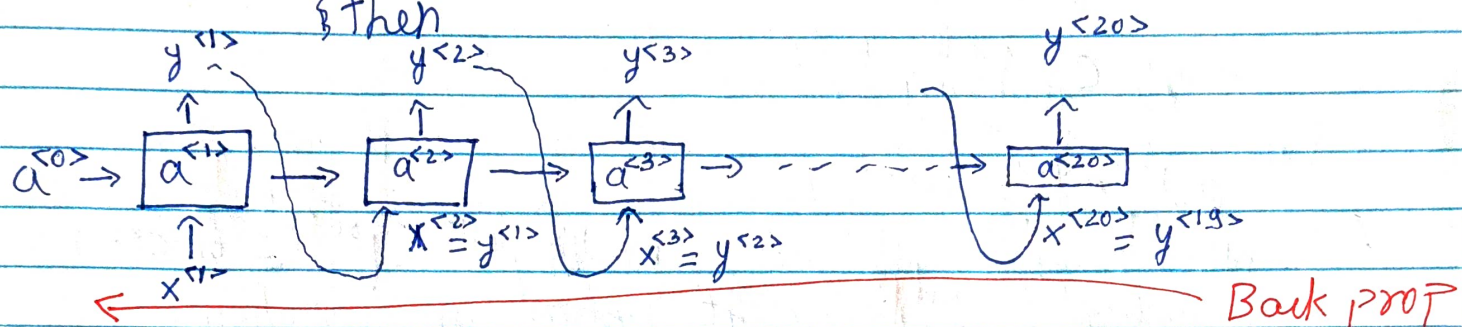
Vocabulary = [a, b, c, ..., z, \, ., i, ..., A, B, ...]

In a character level language model instead of predicting the next word you will be predicting the next character.

but character level language models are not as good as word level language model in capturing long range dependencies between how earlier part of sentence affect the later part of the sentence

Vanishing Gradients With RNNs

↳ So, When an RNN takes a long sequence of words, let's say 20, then



So, it's basically like a Neural Net 20 layers deep, & when you perform gradient descent, going back 20 layers if the gradient is a little small, let's say around 0.9, then multiplying it 20 times will result in a very very small number then the gradient vanishes,

So RNNs can learn patterns & word dependencies from the words closer to each other, it doesn't have an actual memory, it's just repeated transformation.

Simplified GRU (Gated Recurrent unit)

→ Let's say you have an RNN, but that RNN has a memory cell, and that memory cell is equal to the activation

Say, C = memory cell

$$C^{<t>} = a^{<t>}$$

Now, we will be calculating $\tilde{C}^{<t>}$ (C tilde t) to replace $C^{<t>}$ with it after an iteration.
help us

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

after calculating $\tilde{C}^{<t>}$, Now you will calculate " Γ " the symbol is ("gamma" u), " u " stands for update
"Capital gamma" it will be the factor by which we update the memory.

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

In the equation above,

you will use sigmoid activation to get the probability between (0-1) that will give you the factor by which you multiply the previous memory.

→ here, W_u & b_u are separate weights & biases
weights for update weights biases for update

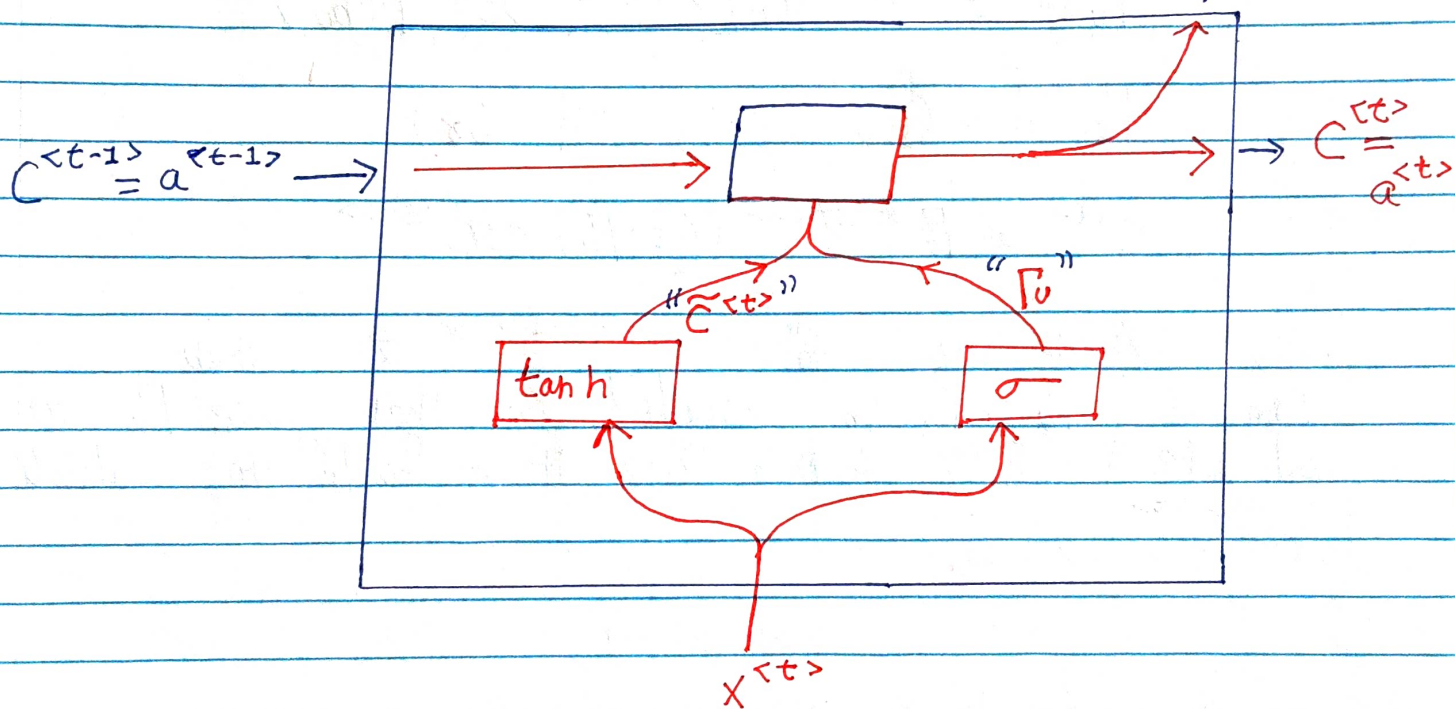
made for the just to get the factor by which you multiply the memory,

Now the new memory cell will be,

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

$$C^{<t>} \text{ or } = \Gamma_u \odot \tilde{C}^{<t>} + (1 - \Gamma_u) \odot C^{<t-1>}$$

the multiplication above are element wise softmax



Now if the sequence is, $\rightarrow$

$\rightarrow$ The cat, which already ate, -- was full
 $\Gamma_u = 1, \Gamma_f = 0 \quad \Gamma_u = 0, \Gamma_f = 0$

$C^{<t>} = 1$

Then the neural net remembers the important thing

Full GRU

→ So in full GRU we introduce " Γ_r " which is to calculate relevance, & that number multiplies with $C^{<t-1>}$ to calculate the relevance candidate that memory cell that will help us replace $C^{<t>}$, it tells you the relevance of $C^{<t-1>}$

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

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

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

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

So, first we take $C^{<t-1>}$ & $\vec{x}$ then compute Γ_u & Γ_r in parallel & then use Γ_r to calculate $\tilde{C}^{<t>}$ & then use it & Γ_u to calculate $C^{<t>}$

LSTM

Long Short Term Memory

So, it's just RNN with more variables,
in LSTM,

you will have Γ_f & Γ_o extra
 Γ_f = forget gate, Γ_o = output gate.

So

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

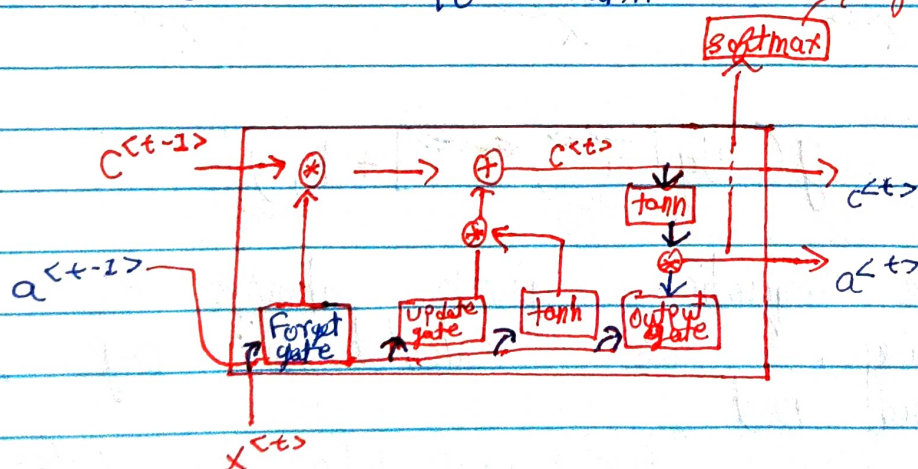
$$\Gamma_u = \sigma(W_u[a^{<t-1>}, n^{<t>}] + b_u)$$

$$\Gamma_f = \sigma(W_f[a^{<t-1>}, n^{<t>}] + b_f)$$

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

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

$$a^{<t>} = \Gamma_o * \tanh C^{<t>}$$



Bidirectional RNN

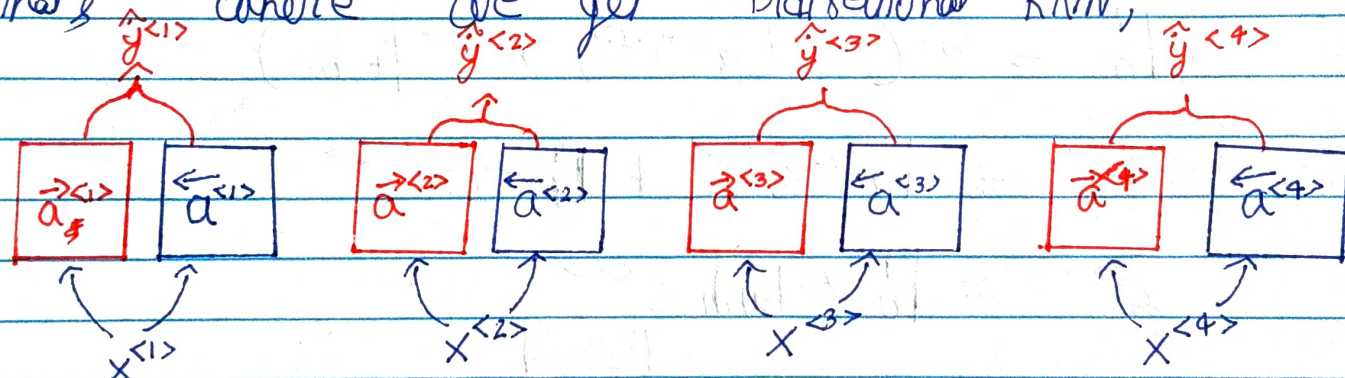
→ He said, "Teddy, bears are on sale!"

→ He said, "Teddy" Roosevelt was a great president!"

→ So, judging ~~the~~ by the first 3 words in both the sentence you can't tell which one is talking about President or which one is talking about the stuff to y,

So, you will need full context of the sentence all at once, to tell that &

that's where we get bidirectional RNN,



Here,

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

→ In the equation given above & the architecture, we calculate $\hat{y}$ by 2 separate neural net,

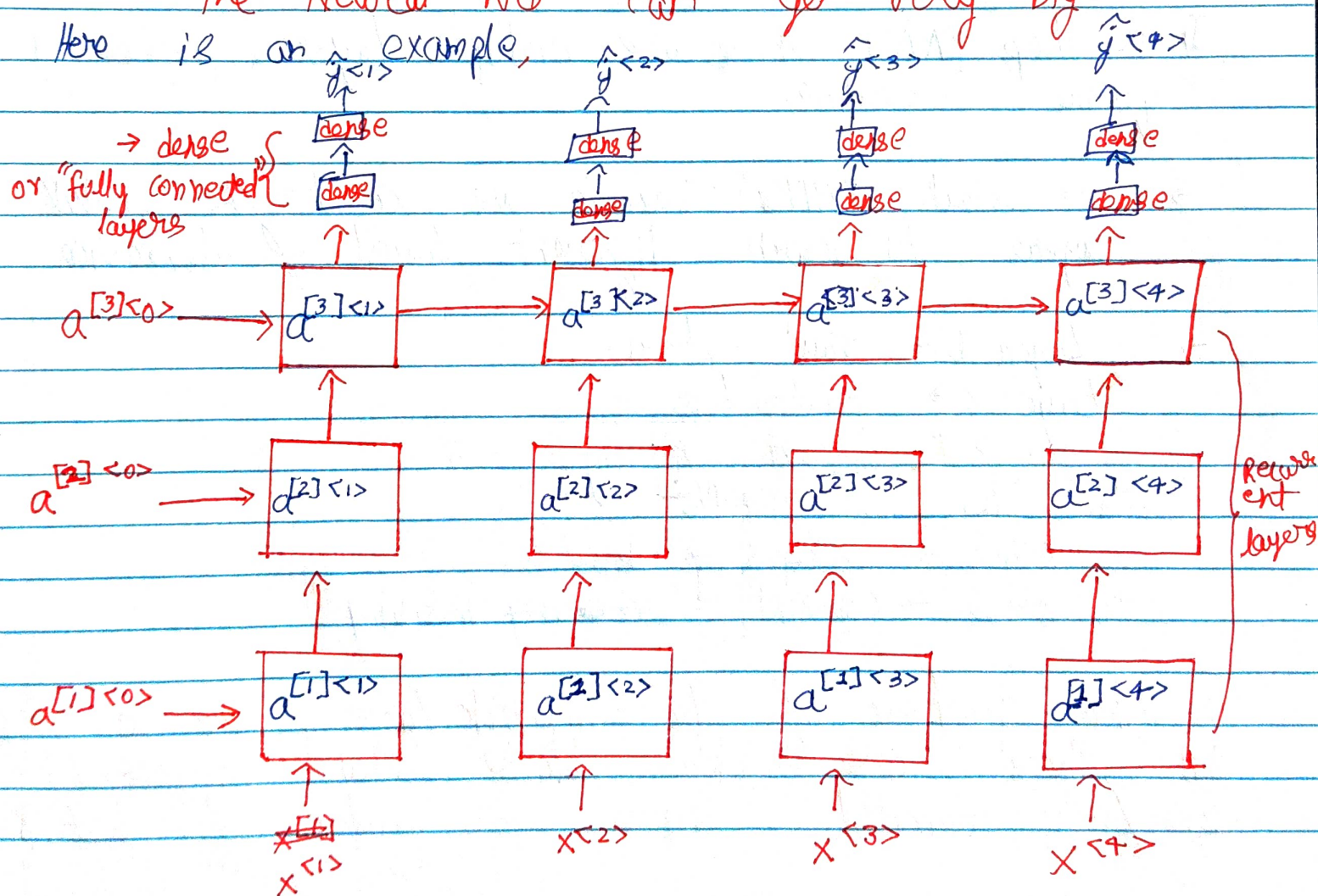
Or we take the help of 2 separate neural net for $\hat{y}$, the first

one goes from left to right & the second one goes from right to left, both of them produce 2 different activations $\vec{a}^{(t)}$ & $\overleftarrow{a}^{(t)}$, then we put these activations in a softmax unit to get an output,

Deep RNN

So, deep RNN has multiple layers of RNN we don't take a lot of layers because the Neural Net can get very big.

Here is an example,



Here,

$a^{[1]} \leftarrow t$, $[1] \rightarrow \text{layer}$

~~t~~ $\rightarrow$

$t \rightarrow$ word in the sequence.

ex, $\rightarrow$

$$a^{[2,3]} = g[Wa^{[2]} [a^{[2]} \leftarrow 2, a^{[1]} \leftarrow 3] + b_a^{[2]}]$$

$\rightarrow$ The representation or the activations move both from, ~~for~~ bottom to top & left to right

In a normal RNN, there is one layers processing sequence overtime,

In a deep RNN, there are multiple layers processing sequence overtime,

$\rightarrow$ we stack RNN's because we want the RNN layers to learn different level of abstraction

$\rightarrow$ Layer 1 $\rightarrow$ raw patterns

Layer 2 $\rightarrow$ relationships

Layer 3 $\rightarrow$ higher meaning

Same idea as CNNs:-

$\rightarrow$ Edges $\rightarrow$ Shapes $\rightarrow$ objects

$\rightarrow$ tokens $\rightarrow$ Context $\rightarrow$ ~~meaning~~ meaning

In the picture of the architecture, you see that there are blue blocks that represent dense/fully connected layers so, why is that?

→ it turns out,

RNN layers extract features overtime
& dense layers are decision makers

→ The RNN doesn't directly output probabilities,
it outputs a representation (embedding/state)

then, dense layers convert representation into
logits, that logits get passed to softmax
unit which convert logits into
probabilities.

Week-2 Natural language processing & word Embeddings

Word Embeddings

→ Earlier we used one-hot encoding vectors but those encoding doesn't hold any meaning they are just randomly initialized, the distance between those vectors are same, so there is no learning going on there, so here comes embeddings in the embeddings, a word is represented as an "n" dimensional feature vector.

let's say for example, the words king, queen & men, women have approximately same vector distance

so,

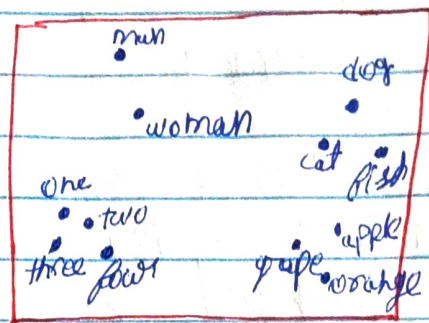
$$\text{king} - \text{men} + \text{women} = \text{queen}$$

& , if you say, I want a glass of orange juice
I want a glass of apple ?
then, the model will most likely say "Juice"
because the words, the model thinks

orange & apple are related
& if it encounters this same sequence, then it is suppose to output the word "Juice"

Visualizing word embeddings.

→ The algorithm that we use to visualize the embeddings are "t-SNE",



it reduces the dimension's of the word embedding to two dimension vector that we can use to visualize.

the model couples the word that it thinks are the same somewhat similar to each other.

Using word embeddings

ex → named entity recognition example

- 1.→ Sally Johnson is an Orange farmer
- 2.→ Robert Lin is a durian cultivator

So, even though in the example, in the second statement, the model has never seen the words "durian" & "cultivator" together,

~~but~~ so the prob should be zero but the model knows,

& Durian $\approx$ fruit
& Cultivator $\approx$ farmer,

can generalize pretty well, & hence it

Transfer learning & word embeddings

1. → Learn word embeddings from large text corpus.
(1-100B words)
(or download pre-trained embedding online).
 - word relationships
 - grammar relations
 - different word dependency
2. → Transfer embedding to new task with smaller training set.
(say, 100k words)
3. → Optional: Continue to fine-tune the ~~word~~ word embeddings with the new data.

Relation to face encoding.

So, in the convolutional Neural Net we studied about the face encoding, which is similar to convolutional Neural Nets face encoding, word embeddings in the Recurrent Neural Nets. encoding & embeddings are basically a same thing.

Properties of Word embeddings

Analogs

lets say, the word encoding for these words are -:

	Man	woman	king	queen
gender	-1	1	-0.95	0.97
Royal	0.01	0.02	0.93	0.95
Age	0.03	0.02	0.70	0.69
food	0.09	0.01	0.02	0.01

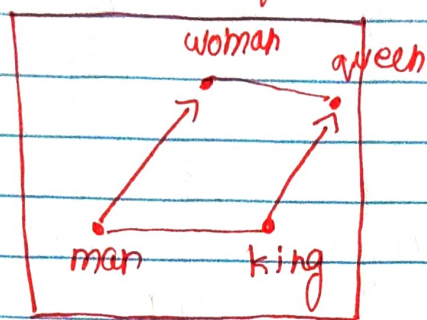
So, one ^{element} dimension of a dimension represents a specific characteristic about that word.

So, for example, $e_m - e_{woman} \approx \begin{bmatrix} -2 \\ 0 \\ 0 \\ 0 \end{bmatrix}$

here, e_{woman} means
embedding for woman

$$e_{king} - e_{queen} \approx \begin{bmatrix} -2 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Analogies using word vectors :-



↳ let's say a
300 dimension embedding

Now, so

$$e_{\text{man}} - e_{\text{woman}} \approx e_{\text{king}} - e_w$$

" e_w means embedding word here"

So let's say,

→ find the word w : $\arg \max_w \text{sim}(e_w, e_{\text{king}} - e_{\text{man}} + e_{\text{woman}})$

in the statement above,

we are trying to
find a word that maximizes ~~the~~ the
similarity between the vectors

$$(e_w, e_{\text{king}} - e_{\text{man}} + e_{\text{woman}}) \text{ or } (e_{\text{man}} - e_{\text{woman}}, e_{\text{king}} - e_w)$$

To find the similarity we will use

Cosine Similarity

→ $\text{sim}(e_w, e_{\text{king}} - e_{\text{man}} + e_{\text{woman}})$

So,

$$\text{sim}(u, v) = \frac{u^T v}{\|u\|_2 \|v\|_2}$$

so, it will give you the output in between ~~$(-1, 1)$~~ $[1, -1]$, so

1 → Same direction (very similar)

0 → Unrelated

-1 → Opposite meaning

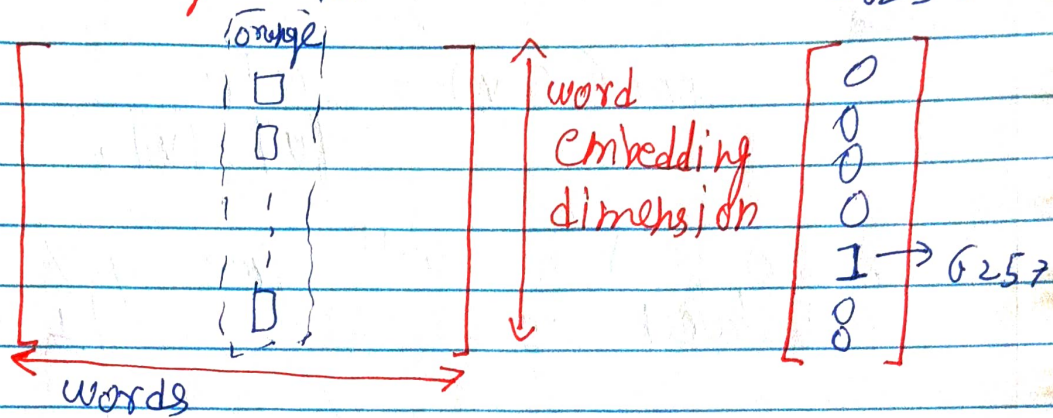
This is why we use cosine similarity to find what word embeddings are related.

Ex →

Tokyo :: Mumbai } similar in some way.
Tall :: Taller }

Embedding Matrix

embedding matrix is just a matrix that contains all the word embeddings. ^{one hot encoding} $\rightarrow$ 0_{6257}



if we multiply, the one hot encoding of the word orange with the embedding matrix then it gives the embedding of the word orange.

$$\begin{matrix} E & \cdot & O_{6257} & = & \begin{bmatrix} \square \\ \vdots \\ \square \end{bmatrix} & = & e_{6257} \\ \begin{pmatrix} 300, \\ 10k \end{pmatrix} & & \begin{pmatrix} 6257 \\ 300, 1 \end{pmatrix} & & \begin{matrix} 300, 1 \end{matrix} \end{matrix}$$

Learning Word Embeddings

→ We can learn word meanings just by training a model to predict words.

Training

→ you take a neural net initialize the embeddings randomly.

→ Now ~~get~~ get a sequence
"I want a glass of ---"

→ & ~~model~~ take the embeddings from the embedding matrix for these words feed it to the NN by stacking them

→ The Neural Net will give you an output

→ based on that example compute loss & do back prop.

→ The embeddings & weights, biases gets updated
over time :-

Similar words :- get similar vectors
different words :- get different vectors

Different ways to Learn embeddings.

1 > Standard (Language model)

Input : Last ~~4~~ words → Predict next word
"glass of an orange" → "juice"

2> Surrounding Context (bidirectional)

Input :- Words before Middle word words after
Predict

Example :-

a glass of "orange" to go along

3> Single previous word.

ex → "orange" → "Juice"
Simpler, still works surprisingly well.

4> Skip-gram idea (Very important)

Input: one word → Predict nearby words

ex → Input: "orange"

Output: "Juice", "fruit", "Sweet"

Word2Vec (skip-gram)

→ This algorithm is a skip-gram algorithm which works by taking a word in a sequence as input then, it has to predict the nearby word like :-

→ A bottle glass of juice.
 So, we will give the input as glass & ask what should be the output aroundy nearby word ± 10 ,

→ we give it a single word input & expect a single word output.

Ex →

→ Context word → "orange"

→ A target word nearby → "Juice"

→ Using Softmax to predict the probability for the ~~sequence~~ each of the word in the embedding given the input word. for ex

$$P(t/c) = \frac{e^{\theta_t^T c}}{\sum_{j=1}^{10000} e^{\theta_j^T c}}$$

here e , 10k is the size of the vocabulary.

Model -:

Vocab size = 10,000

Context "c" ("orange") $\rightarrow$ target "t" ("juice")
6257 4839

Now,

$$O_c \rightarrow E_E \rightarrow C_c \xrightarrow{\text{softmax}} \hat{y}$$

$O_c \Rightarrow$ one hot of context

$E_E \Rightarrow$ Embedding matrix

$C_c = E \cdot O_c =$ Embedding ^{vectors} of that specific word.

softmax: $P(t/c) = \frac{e^{\theta_t^T C_c}}{\sum_{j=1}^{10,000} e^{\theta_j^T C_c}}$

(Note: Red arrows point from 'target' to θ_t and from 'context' to C_c)

$\theta_t^T =$ Parameter associated with output t . & taking its transpose

$\Rightarrow$ 10k is the no of words in the vocab

$$Loss = L(\hat{y}, y) = - \sum_{i=1}^{10k} y_i \cdot \log \hat{y}_i$$

categorical cross entropy

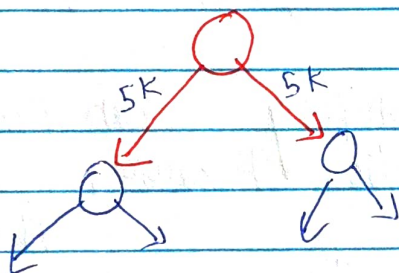
~~$y = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}$~~ $\rightarrow$ 4839 (4839, 2)

Problems with softmax classification

$$P(t/c) = \frac{e^{\theta_t^T e_c}}{\sum_{j=1}^{\text{look}} e^{\theta_j^T e_c}}$$

→ it's compute heavy, in the denominator you can see that you will have to do sum over

→ So here insted of full softmax we will use, Hierarchical softmax



→ In the Hierarchical softmax insted of calculating the probability for every word. we, ask the root node to tell us, ^{what is whether} the prob of the word ^{being} is in the left ~~or~~ subtree or right subtree, & repeat this process till you get to the leaf node where your word is. so the probability becomes

$$P(t/c) = \prod_{\text{nodes on path}} P(\text{left/right decision})$$

Basically every ~~node~~ node is doing a logistic regression.

So, the time complexity drops from

$$\left. \begin{array}{l} O(v) \\ \text{to} \\ O(\log v) \end{array} \right\} \begin{array}{l} v, \text{ here is the vocab} \\ \text{size} \end{array}$$

You place the common words like 'the', 'of', 'or' above in the tree not too deep because these words appear too often & other words like 'durian' can be buried deeper in the ~~network~~ tree.

How to sample the context c ?

→ So instead of sampling randomly we use a different heuristic because,

A sentence has a lot of 'is', 'a', 'the', 'are' & other common words & hence the words that are rarely seen like "durian" will be dominated by those simple common words,

that's why we avoid random sampling & go for some heuristic. & avoid model from learning useless ~~learning~~ information.

Negative Sampling

→ In negative sampling we take context to target pairs,

But there is a twist,

Instead of taking only positive pairs & doing softmax

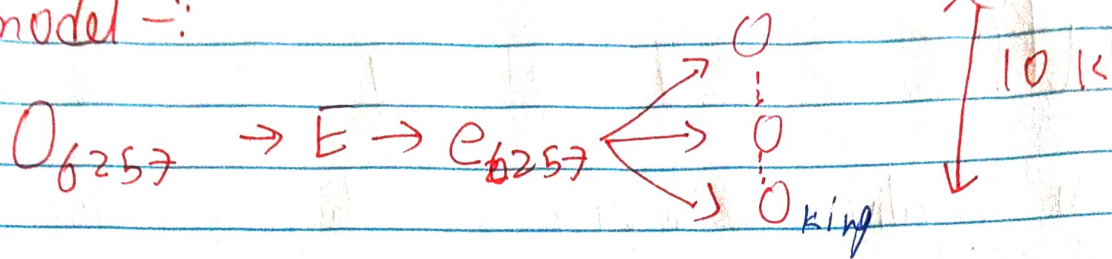
we take one context word & map it out with a ~~lot~~ of positive ^{target} word & "k" number of words (target) that are chosen at random & are labeled negative

$\left\{ \begin{array}{l} k = 5 - 20 \text{ for smaller dataset.} \\ k = 2 - 5 \text{ for larger dataset.} \end{array} \right\}$

for example,

Context	word	target
orange	Juice	1
Orange	king	0
Orange	book	0
Orange	the	0
Orange	of	0

The model -:



in the model above every output node computes a probability between $\{0-1\}$ whether the context is related to the word or not, ("basically logistic regression") but instead of doing it for 10k examples we only do it for the ~~we~~ "k" number words that we chose,

& hence instead of computing for all the words we only compute output for those 5 words, & update embeddings & parameters of the neural net, And that's how it's done.

Selecting -ve examples

You can choose the word by the sampling from the corpus using the formula.

$$P(w_i) = \frac{f(w_i)^{3/4}}{\sum_{j=1}^{10000} f(w_j)^{3/4}}$$

so here what we are doing is

- count word frequencies
- multiply them with 0.7500
- apply power 3/4
- normalize it
- Now sample from it "The distribution"

in the equation on left,

$F(w_i)$ $\rightarrow$ frequency of the word

& in the denominator we are summing over all the examples ~~#~~ frequencies, basically to normalize it.

$\rightarrow$ we don't do random sampling because the occurring words would be too out of the place like context $\rightarrow$ orange

word $\rightarrow$ Thermodynamics, Nucleus, liver, Mozart
"sampled"

$\rightarrow$ & we don't go for sampling purely on frequency basis because then the words would be full of ~~typical~~ those common words like: of, the, and, or

$\rightarrow$ so the sweet spot is $\boxed{F(w_i)^{3/4}}$ & the formula on left

GloVe word vectors

- Another method to compute word vectors
- it's not used much
- it stands for Global vectors for word representation

⇒ C, t

X_{ij} = # times i appears in the context of j
↓ ↓ ↓
 C t " C " " t "

→ you ~~go~~ Go through your corpus of text & count up how many time target " t " occurs in the context " C "

So, you might can also get,

$$X_{ij} = X_{ji}$$

Now,

" X_{ij} " is a count how ~~often~~ often do words i & j appear with each other.

Model,

In the model we have to minimize,

$$\sum_{i=1}^{10,000} \sum_{j=1}^{10,000} F(X_{ij}) (\theta_i^T e_j + b_i + b_j - \log X_{ij})^2$$

Here

θ_i^T = Parameter associated with output

$e_j \Rightarrow$ embedding of the given word.

$b_i \Rightarrow$ bias associated with context $'i'$

$b_j \Rightarrow$ bias associated with target $'j'$
 $\rightarrow$ given symbol

$F(x_{ij}) = 0 : x_{ij} = 0$
{ because log of 0 is not defined

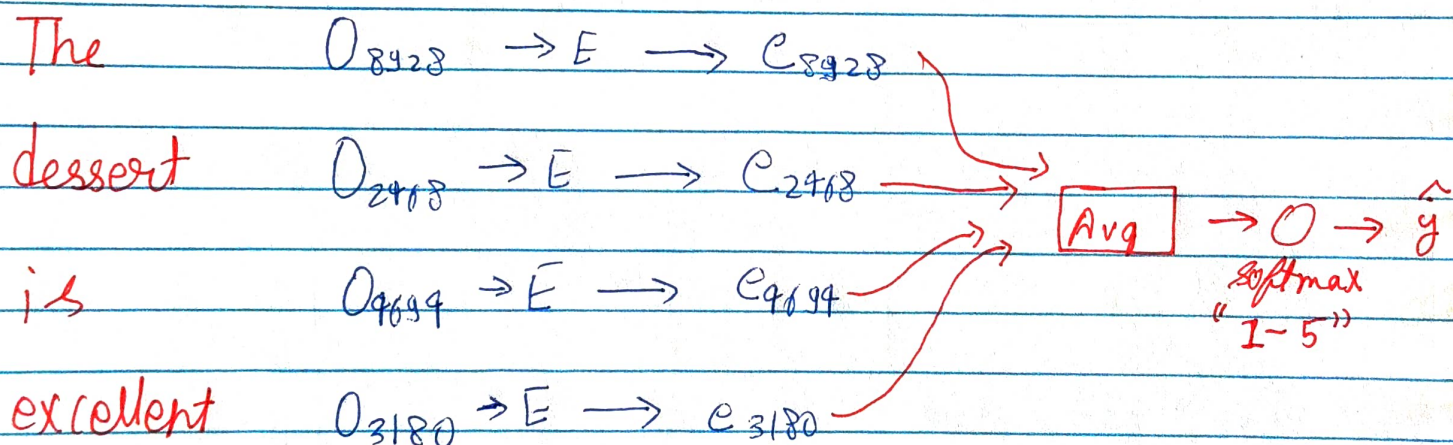
Sentiment Classification

Input $\rightarrow$ A string of characters
Output $\rightarrow$ A rating

ex $\rightarrow$ Input $\rightarrow$ The desert is excellent
Output $\rightarrow$ 3.5 stars

Simple sentiment classification model

The desert is excellent
8928 2468 4694 3180



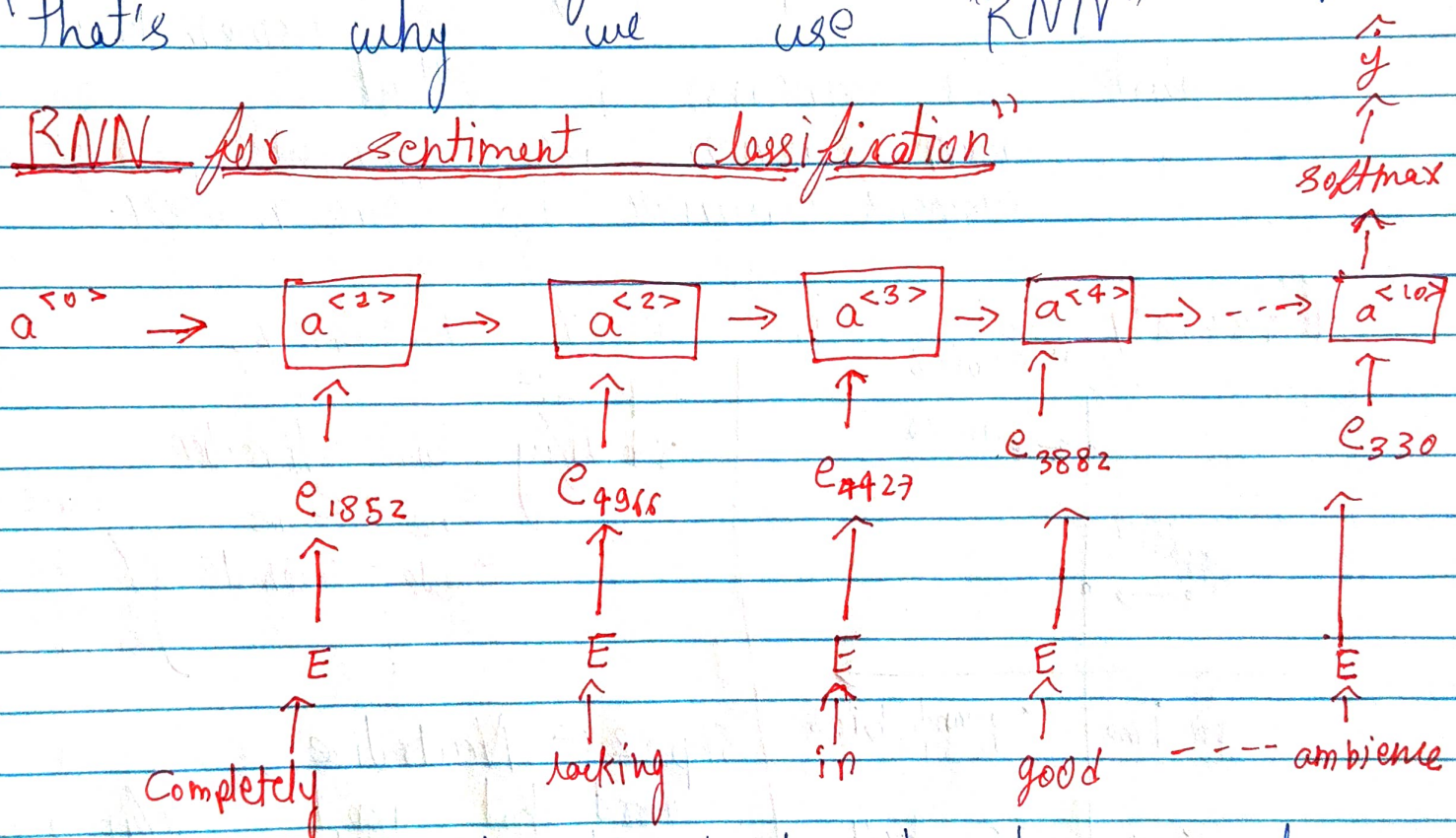
$\rightarrow$ So, in a simple sentiment analysis we take the "n" dimensional embeddings of each word in the sequence & then average them out, feed the embedding to softmax & it gives you an output.

but for examples like -:

⇒ "completely lacking in good taste, good service, & good ambience."

Here you can see that there are multiple "good" words in the sentence, so the model might think, it's a ~~good~~ ~~rev~~ positive review, but it's actually lacking the positional context of words in the sequence that's why we use "RNN"

"RNN for sentiment classification"



→ You should understand it by yourselves brother +5 is too late for you to not get it, lmao come on, seriously if you can't then, either you are doing like a CEO thing or something has gone horribly wrong.

Debiasing word embeddings

for example :-

word embeddings can reflect gender, ethnicity, age, sexual orientation, & other biases

for example :-

Man: Computer-programmer as Women:
Homemaker

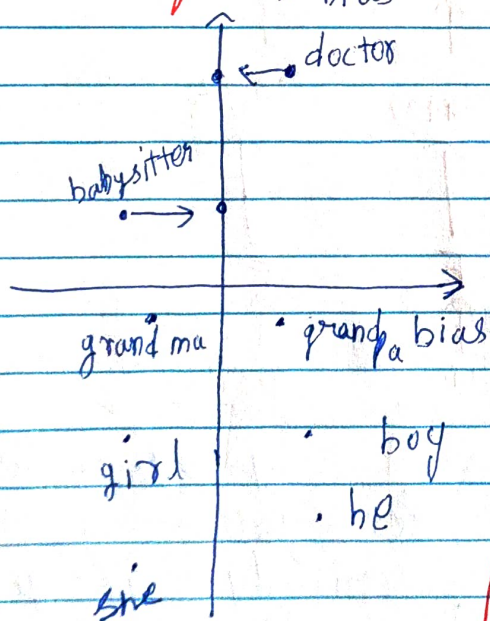
this means

~~Man~~ Computer-programmer - Man + Women = Homemaker

what this means is that

man's career is computer programmer
women's career is home-maker.

Addressing bias in word embeddings



step 1:-

Identify bias direction

$e_{he} - e_{she}$
 $e_{male} - e_{female}$
...
take average of the resultant vectors from subtraction

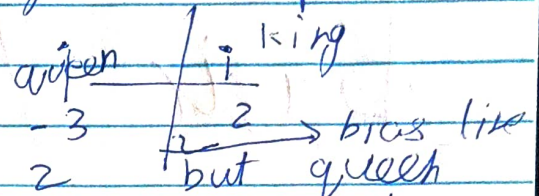
step 2:- Neutralize, for every word that isn't definitional, project to get rid of bias. like the "PCA"

The non-bias line should be in the $\perp$ direction of the bias line.

3> Equalize pairs.

→ So, equalizing pairs needs to be done to reduce the asymmetry,

for example



notice that king is at 2 but queen is at -3 so that's not good you want both of your vector points to be at equal distance from bias point,

so we choose a bias point's that is equal from the bias.

Week - 3

Sequence models & attention Mechanism to sequence

Basic Models

sequence to sequence model
Example (1) :-
french sentence $\rightarrow$ Jane visite l'Afrique en
septembre
 $x^{<1>}$ $x^{<2>}$ $x^{<3>}$ $x^{<4>}$ $x^{<5>}$

English translation $\rightarrow$ Jane is visiting africa
in september.
 $y^{<1>}$ $y^{<2>}$ $y^{<3>}$ $y^{<4>}$ $y^{<5>}$ $y^{<6>}$

So, the model architecture will look like,

there are 2 networks :-

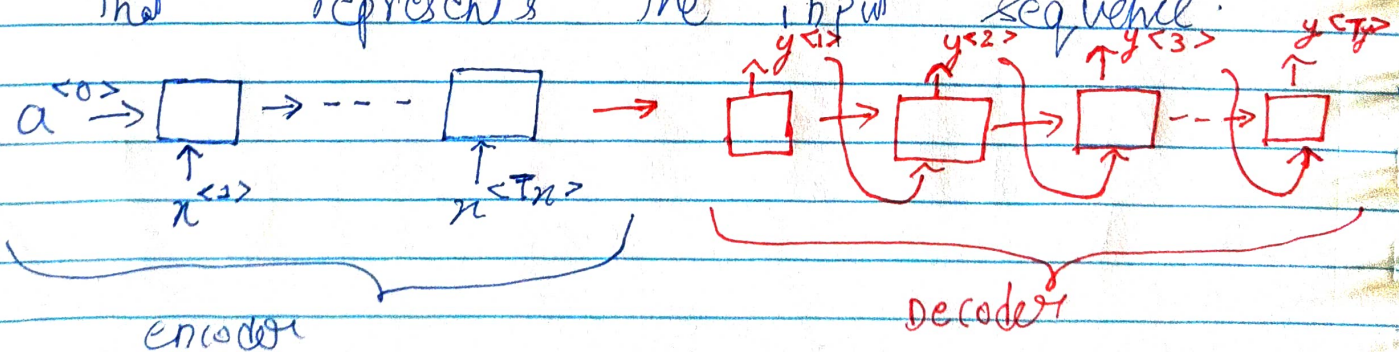
(1)

(2)

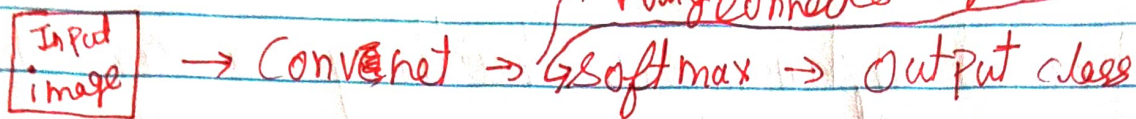
encoder network

decoder network

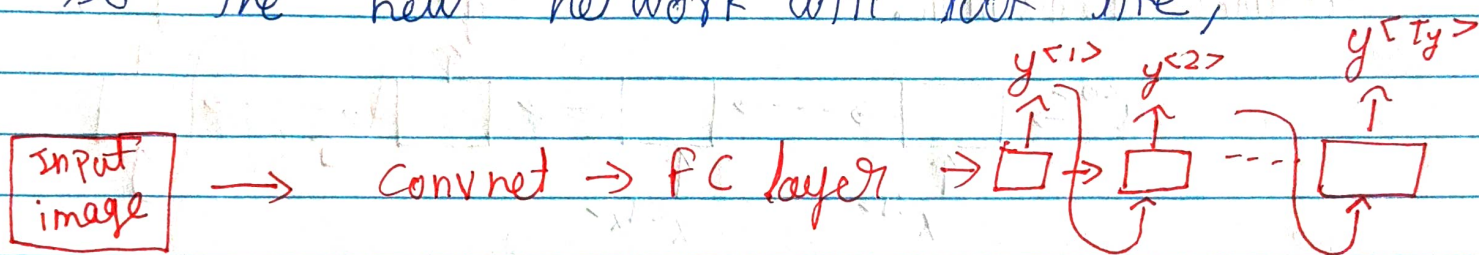
$\rightarrow$ The encoder network will take the sequence of input & output a vector that represents the input sequence.



example (2) → Image captioning



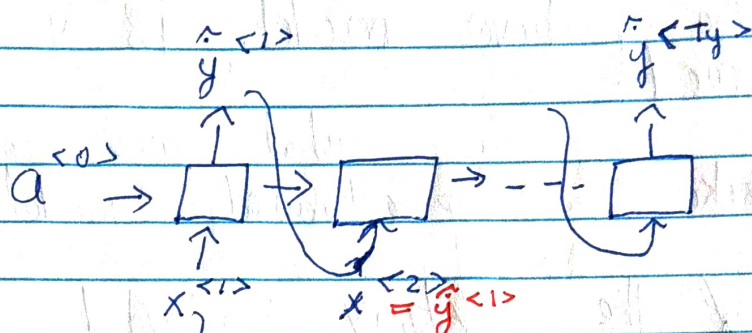
The above model is an image classification model, so what we will do is that take a pretrained model like this, get rid of the softmax & instead insert an RNN that will give you token outputs. So the new network will look like,



Picking the most likely sentence.

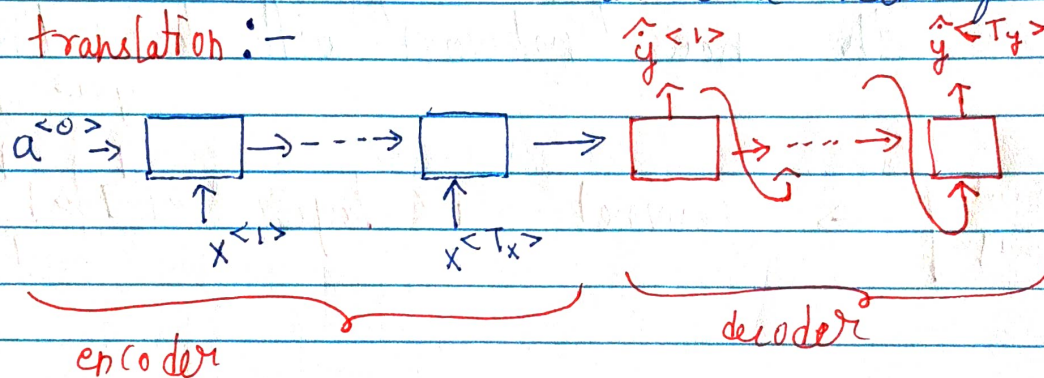
→ Machine translation as building a conditional language model.

Language model:-



it can be a vector full of zeros

Machine translation:-



→ Notice that the Machine translation model's decoder looks exactly like the language model,

it's just that we have added encoder as well

So, how we are calculating,

we will try to maximize the probability of the English translation given the original French translation

$$P(y^{<1>}, \dots, y^{<T_y>} | x^{<1>}, \dots, x^{<T_x>})$$

finding the most likely translation that will maximize the probability.

$$P(y^{<1>}, \dots, y^{<T_y>} | n)$$

french input

So, the given sentence in french is.

→ Jane visite l'Afrique en septembre.

If you sample at random you might get.

→ "Jane is visiting Africa in September" which is a good translation,

but you might get bad translations also because you are sampling at random.

ex.

→ In September, Jane will visit Africa.

So, when you are using this model for machine translation.

you are trying to sample at random from this distribution,

instead you are looking for a english sentence (y) that maximizes that conditional probability,

$$\arg \max_{y^{<1>}, \dots, y^{<T_y>}} P(y^{<1>}, \dots, y^{<T_y>} | n)$$

The most common way of getting this sequence is beam search, but you may ask a question

why not greedy search?

→ so in greedy search we just pick the largest probability of " $\hat{y}^{<i>}$ " given the input sequence, but that doesn't work.

We are trying to maximize the joint probability of the whole sequence not a single word, so,

for example

→ Jane is visiting Africa in september

→ Jane is going to be visiting Africa in september.

while picking the 3rd word insted of picking visiting it can pick going because going is more common & has the maximum probability.

the total no of combination of words in an english sentence is exponentially large.

so, if you have 10,000 words then the no of possible permutations will be $(10,000)^{10}$.

which is too much

to check

Beam Search Algo

Step 1:- Calculate the probability of every token in the vocabulary given the input vector from decoder.

Step 2:- Here the beam search comes into play it has a parameter called "B", which is "beam width" which is a hyperparameter & usually set manually.

So if the beam width is 3 then the algorithm will pick top 3 words with the most probability.

Step 3:- In this step the algorithm will use the first, then second & then third token to then generate the Probability,

$$P(\hat{y}_1 | x, y_1)$$

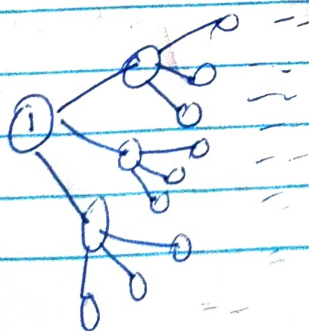
$$P(\hat{y}_2 | x, y_2)$$

$$P(\hat{y}_3 | x, y_3)$$

So, we will

calculate probability & continue from step 1 again till we get the "end" token.

visual diagram



till you get the end of statement token

Refinements to beam Search

→ length normalization

So, we are trying to maximize

$$\rightarrow \arg \max_y \prod_{t=1}^{T_y} P(y^{<t>} | x, y^{<1>}, \dots, y^{<t-1>})$$

→ But the probabilities are very small numbers
so when you multiply small numbers by
they become even smaller, & your computer
can't store it, that's why we take log of
the values.

So,

$$\arg \max_y \prod_{t=1}^{T_y} P(y^{<t>} | x, y^{<1>}, \dots, y^{<t-1>}) = P(y^{<1>} \dots y^{<T_y>} | x)$$

$$= P(y^{<1>} | x) * P(y^{<2>} | x, y^{<1>}) * \dots$$

$$\dots P(y^{<T_y>} | x, y^{<1>}, \dots, y^{<T_y-1>})$$

so, after log it will look like,

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

→ Here is the part where we normalize the length, so to reduce the amount of log we normalize it, because longer sequences get punished unfairly

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

the alpha " α " on the superscript is normalization parameters, it's a hyperparameter. $[0-1]$, we don't want to fully normalize, so we use α for partial normalization, it's usually taken " $\alpha=0.7$ "

Error analysis in Beam Search

→ While doing error analysis you can get two type of errors, error in beam search or error in RNN.

Search

Ex → (y*) Human translation :- Jane visit's Africa in september
(ŷ) Algorithm :- Jane visited Africa last september

Case 1 → $P(y^*|x) > P(\hat{y}|x)$

Beam search chose $\hat{y}$, but y^* attains higher $P(y|n)$.

Conclusion: Beam search is at fault.

Case 2 → $P(y^*|x) \leq P(\hat{y}|x)$

y^* is better translation than $\hat{y}$. But RNN Predicted $P(y^*|n) \leq P(\hat{y}|n)$

Conclusion: RNN is at fault

So, now you take multiple examples & on each example count who is at fault, whichever has the biggest number is at fault.

Bleu Score

stands for $\therefore$ bilingual evaluation understudy

→ It's used to evaluate machine translation system because given a sentence in french there can be multiple english translations which are correct

ex → french : Le chat est sur le tapis

Reference 1: There is a cat on the mat

Reference 2: The cat is on the mat

output : the the the the the the the

Precision : $\frac{7}{7}$ } the words that are in output are they in the reference's

Modified precision:

$\frac{2}{7}$
Count ("the")
Count ("the")
Total count of "the" in the output

so, here we only take the value of the as many times as it is in the Reference

So, we have only been looking at in the isolation meaning one word at a time, but now let's look at 2 of those at a time "bigrams"

Blue score on bigrams

example: Reference 1:- The cat is on the mat

Reference 2:- There is a cat on the mat.

MT output :- The cat the cat on the mat

Possible bigrams are :-

	Count	Count _{clip}
the cat	2	1
cat the	1	0
cat on	1	1
on the	1	1
the mat	1	1

"Counts in the output"

"Counts in the reference's"

$$\text{Precision} = \frac{\text{Sum of the Count}_{\text{clip}}}{\text{"Total no. of bigram appeared"}} \Rightarrow \frac{4}{6} \Rightarrow \boxed{\frac{2}{3}}$$

Blue Score on unigrams

Example :- ~~Reference 1: The cat is on the mat~~

generalized form

(Precision unigram) $P_1 = \frac{\sum_{\text{unigrams} \in \hat{y}} \text{Count}_{\text{clip}}(\text{unigram})}{\sum_{\text{unigram} \in \hat{y}} \text{Count}(\text{unigram})}$

Precision "n-gram"

$$P_n = \frac{\sum_{n\text{-gram} \in \hat{y}} \text{Count}_{\text{clip}}(n\text{-gram})}{\sum_{n\text{-gram} \in \hat{y}} \text{Count}(n\text{-gram})}$$

Combining all of it we get

P_n = Bleu score on n-grams only

So, taking side of P_1, P_2, P_3, P_4 only here for the example.

Combined Bleu score:- $BP \exp\left(\frac{1}{4} \sum_{n=1}^4 P_n\right)$

BP = Brevity penalty,

it is a penalty that applies on the model if it tries to shorten the output, because shorter output's can have high precision

The, $BP = \begin{cases} 1 & \text{if } \text{MT-output length} > \text{reference output length} \\ \exp(1 - \text{reference output length} / \text{MT-output length}) & \text{otherwise} \end{cases}$

→ The Bleu score is a good single value evaluation metric to test a machine translation model

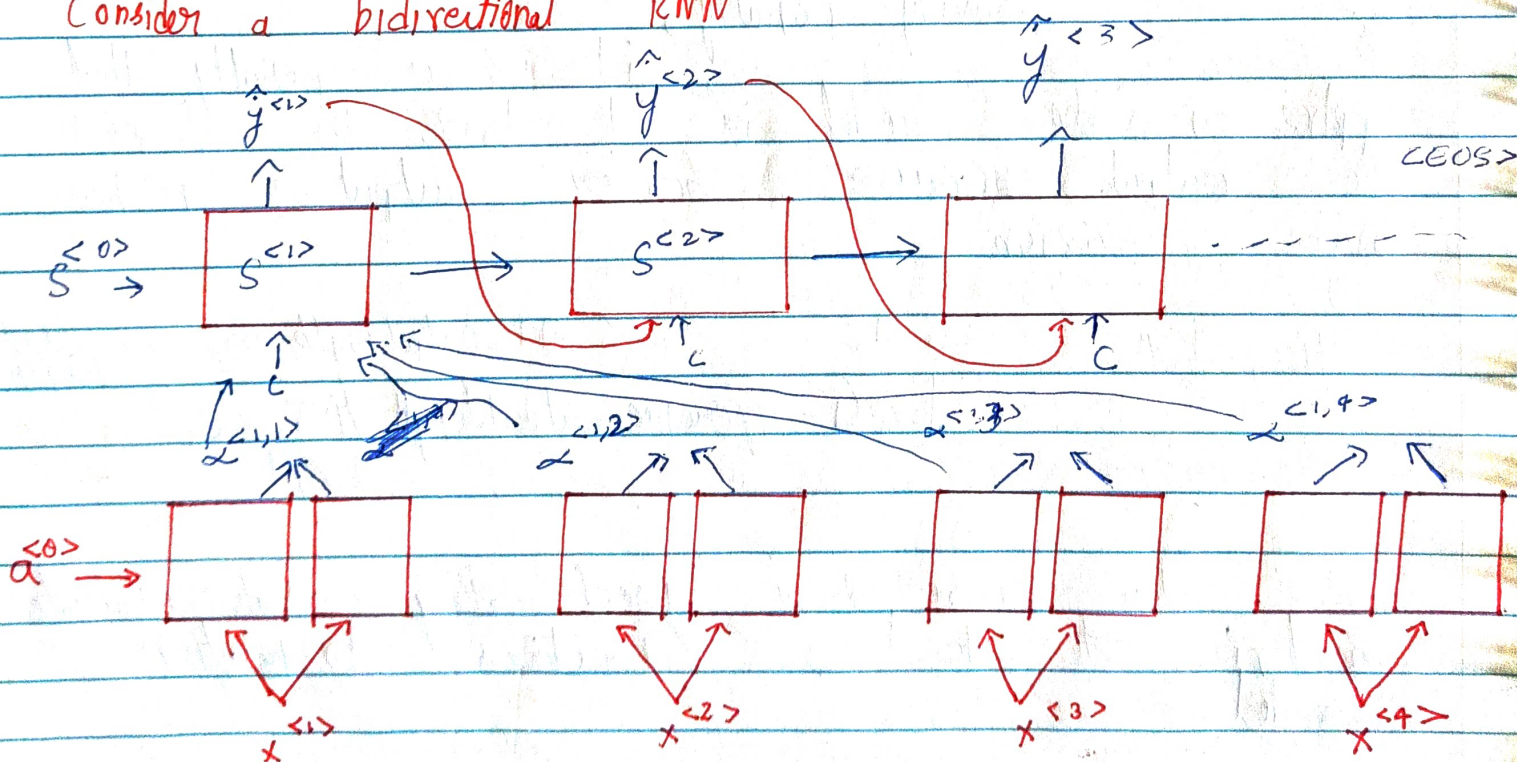
Attention Model intuition

The problem of long sequences

→ so, given a long sequence an encoder encodes the sequence, then it goes into decoder, but that's not how the human translation work,

humans don't memorize ~~store~~ every word & then start to translate it, they look at some parts of the ~~entire~~ sentence, translate it then look at other parts of the sentence translate it, so, here to mimic that behaviour we introduce attention mechanism

Consider a bidirectional RNN



→ In the attention model we introduce another parameter " α ", α is the attention weight which tells the model how much should you focus on a given activation, while generating the output, here $s^{<0>}$ refers to the hidden states of the decoder RNN just like $a^{<0>}$ in the encoder RNN.

& at last it stops when the decoder RNN outputs $\langle \text{EOS} \rangle$ "End of statement" token or you can say delimiter.

Attention model : in detail

$t' \rightarrow$ the index of the ~~en~~ input of encoder wordes

$a^{<t'>}$ = activation on the t' -th word of the input.

$$a^{<t>} = (\vec{a}^{<t'>}, \vec{a}^{<t'>})$$

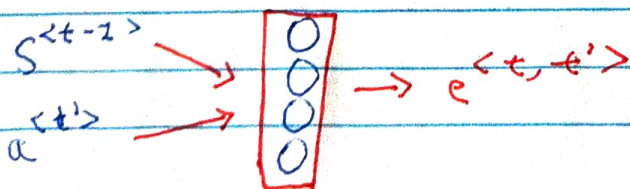
$$\sum_{t'} \alpha^{<t', t'>} = 1 \quad \left\{ \begin{array}{l} \text{The sum of all the weight's} \\ \text{for a context must be} \\ \text{equal to 1} \end{array} \right\}$$

$$c^{<t>} = \sum_{t'} \alpha^{<t', t'>} a^{<t'>}$$

$\Rightarrow$ Context let's say for the first iteration will be the multiplication of activations by their respective weight's.

$\alpha^{<t, t'>}$ = amount of attention $y^{<t>}$ should pay to $a^{<t'>}$

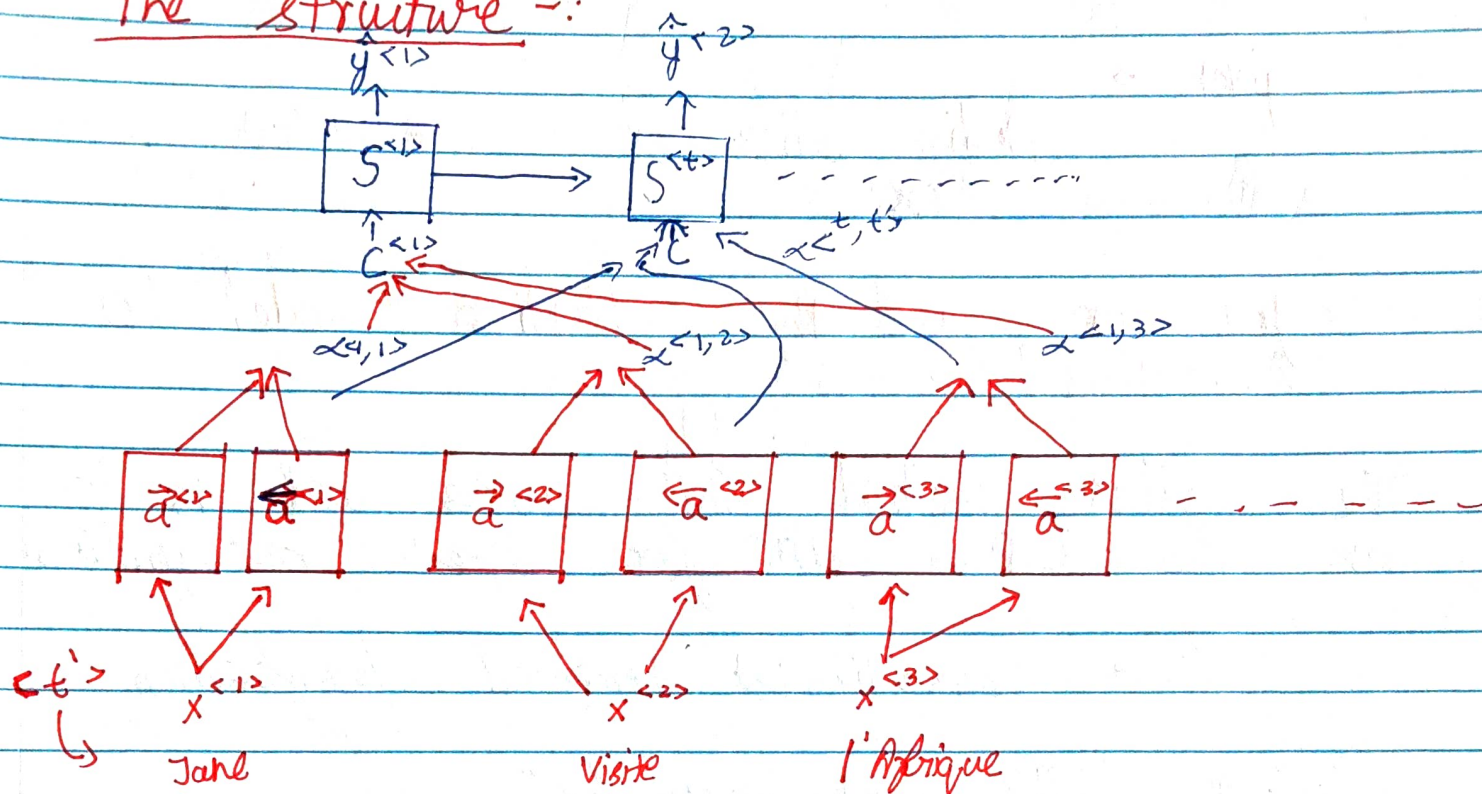
$$\alpha^{<t, t'>} = \frac{\exp(e^{<t, t'>})}{\sum_{t'=1}^T \exp(e^{<t, t'>})}$$



we use a small neural net to output the $e^{<t, t'>}$, & then compute the α using the softmax for input we provide it with the activation

activation of decoder
of the previous ~~output~~ $s^{<t-1>}$ & all the activations

The structure -:



The process -:

So, the words enter in the Bidirectional RNN, GRU or LSTM RNN, GRU or LSTM & we get the activations " $a^{<t>}$ ",

then there is this small Neural network who ^{takes} inputs the activation of all the encoder's & activations & the previous activation of decoder " $s^{<t-1>}$ " & It calculates the weights " α " for the " $a^{<t>}$ " activation which, then we multiply the weights by their activations,

& sum it all which becomes the input for decoder & then standard LSTM Stuff which gives output.

Speech recognition

given $\rightarrow$

Audio clip
"x"

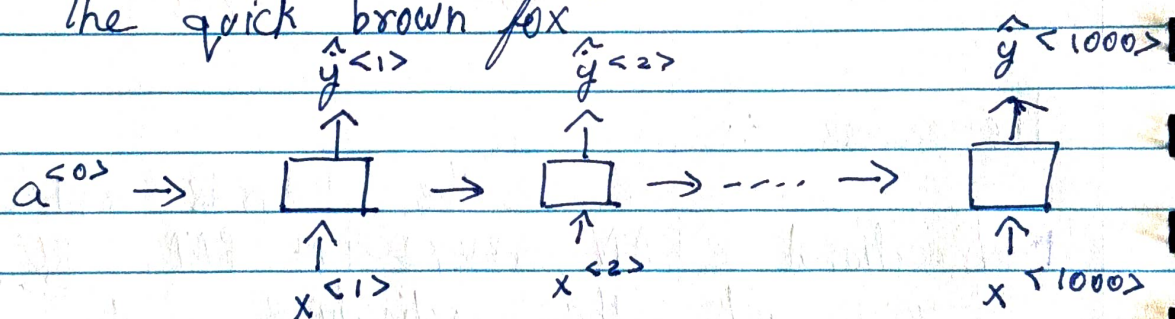
$\longrightarrow$ Transcript
"y"

$\rightarrow$ you can use the attention model to do this too

The other method that works well is to use the CTC cost for speech recognition.

CTC $\rightarrow$ Connectionist temporal classification

ex $\rightarrow$ "The quick brown fox"



here, we have drawn unidirectional RNN but in practice we will have bidirectional

$\rightarrow$ Here the number of time steps here is very large,

for example there are 1000 input & output's, if you have 10 seconds of audio & your features come at 100 hertz so

you will have 100 samples per second.

But your output not have a 1000 characters or Alphabet's,

Now what do you do.

The CTC cost function allows the RNN to generate output like

t t t - - - h h h - - - e e - - - 1 - - - q q q - - -

This gets

↳ blank space

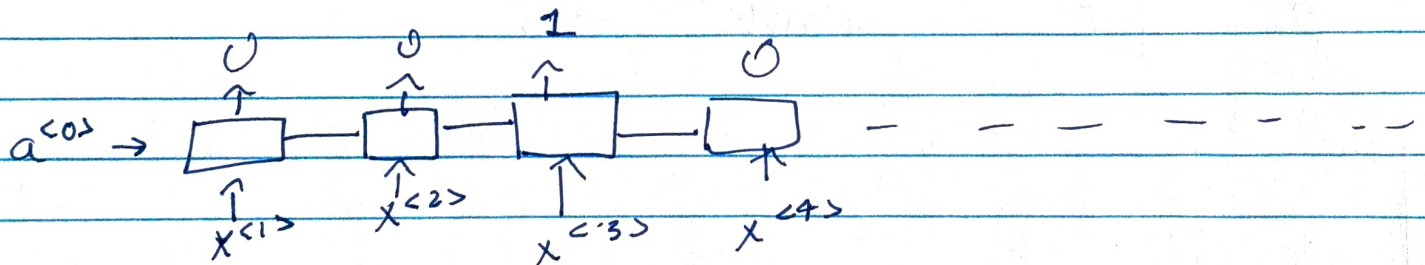
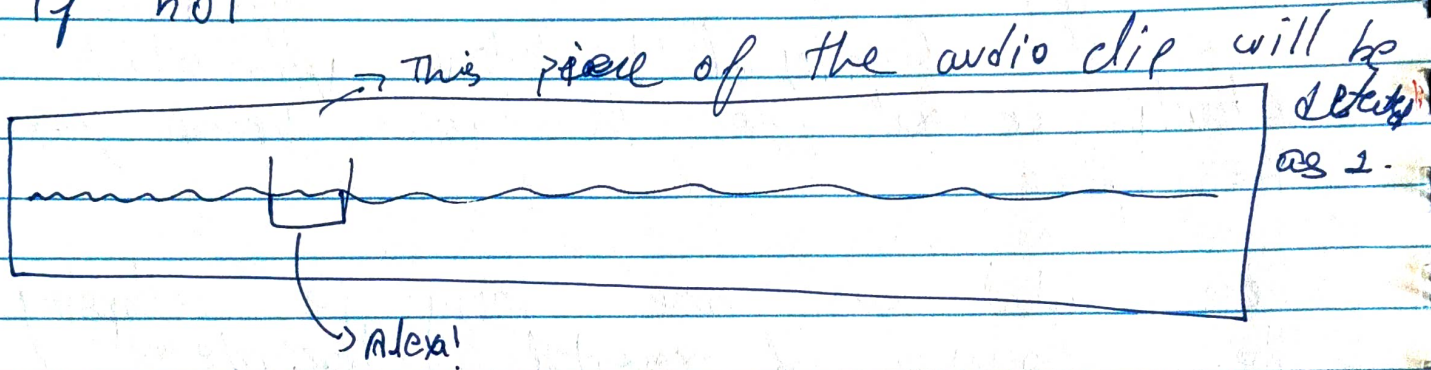
Condensed to "the q;" which is actually correct for the "quick brown fox." the first

here what we are doing is collapsing the sequence of repeated characters to get a complete word

Trigger word detection.

Like "hey google" & "Alexa!"

Take an RNN Feed it a sequence of words or audio & label the output as 1 if trigger word is detected & 0 if not



That's it

Now imma go sleep im AF RN brother i cant take ts anymore.