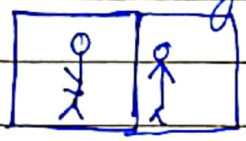
SEQUENCE MODELS

few examples :

1) Machine translation

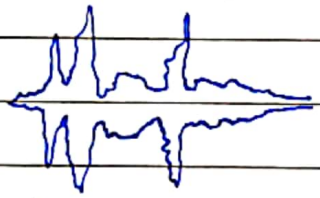
Voulez-vous chanter avec moi? → Do you want to sing with me.

2) Video activity recognition



→ running.

3) Speech recognition



→ the quick brown fox jumped over the lazy dog.

NOTATIONS

Motivating example :

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

y :	1	1	0	1	1	0		
	0	0	0					

NOTE: This is a named entity recognition problem

x: $x^{<1>}$, $x^{<2>}$, ... $x^{<t>}$ ... $x^{<9>}$

y: $y^{<1>}$, $y^{<2>}$, ... $y^{<t>}$ ..... $y^{<9>}$

T_x : length of input sequence = 9

T_y : length of output sequence = 9.

So,

1) $x^{(i)<t>}$: means, the "t" element in the training sample "i"

2) $T_x^{(i)}$: means the length of input sequence in " " "i".

similiarly for $y^{(i)<t>}$ & $T_y^{(i)}$

let's now talk about how we would represent individual words in a sentence.

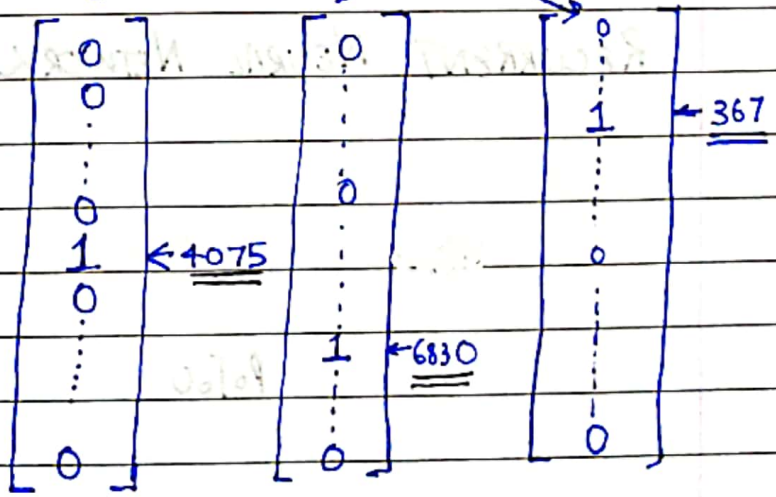
first we ned to make a vocabulary, making a list of words that yow will use in your representation.

x: harry potter and

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

Vocabulary

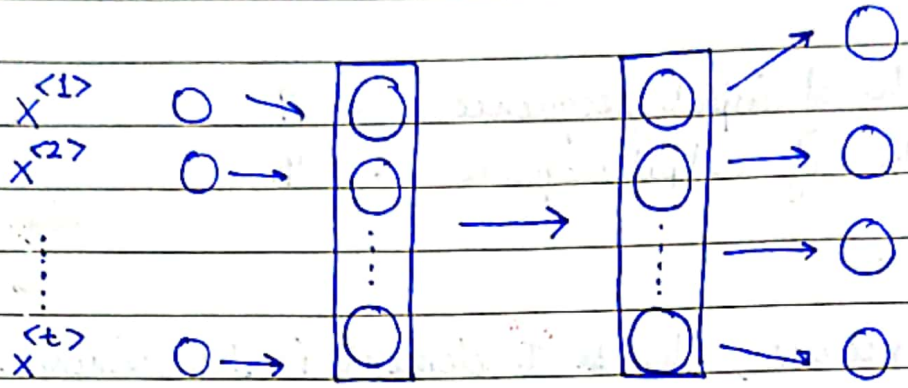
a	1
aron	2
⋮	⋮
and	367
⋮	⋮
harry	4075
⋮	⋮
potter	6830
⋮	⋮
Zulu	10000



THIS IS KNOWN AS OH ENCODINGS

" basically you have made vectors "

Why not a standard Network ?



PROBLEMS

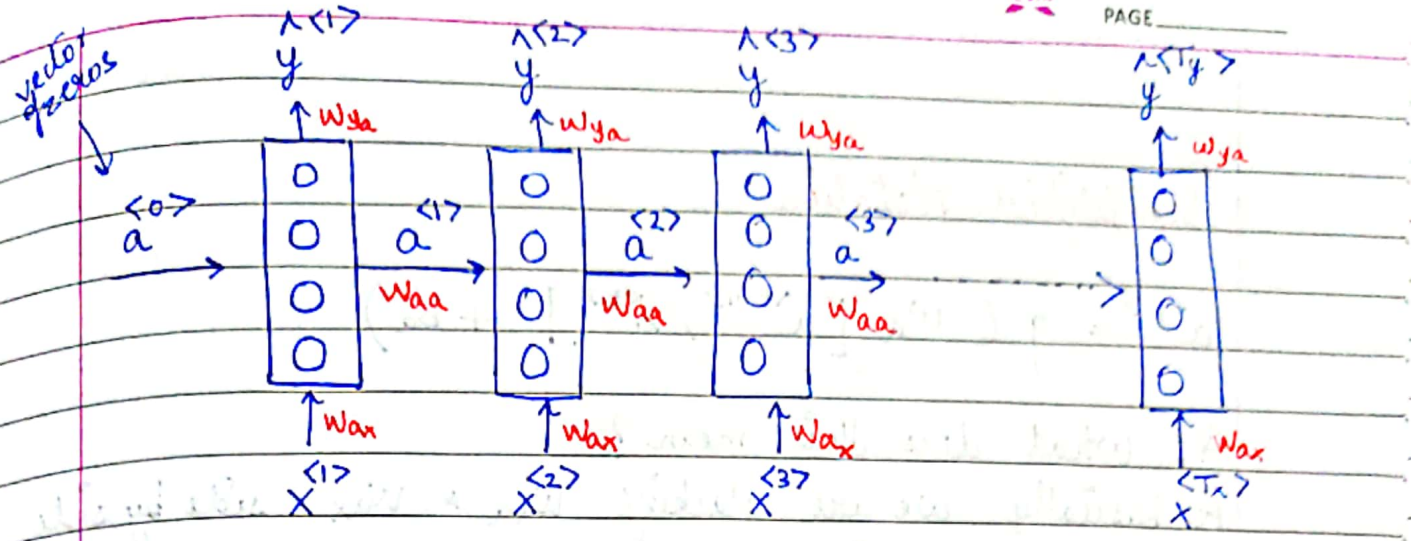
→ In different training samples, the inputs & outputs can be of different length, in other words. $T_x^{(i)}$ & $T_y^{(i)}$ varies with i.

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

→ ONE are too compulative

RECURRENT NEURAL NETWORK

PoTo0



So in this recurrent neural network, what this means is that when making the prediction for $\hat{y}^{<3>}$, it gets the information not only from $x^{<3>}$ but also the info. from $x^{<1>}$ & $x^{<2>}$ because the information on $x^{<1>}$ can pass through this way to help to prediction with $\hat{y}^{<3>}$

DISADVANTAGE

* RNN only uses the information that is earlier in the sequence to make a prediction. So, LEFT → RIGHT

* prediction on $\hat{y}^{<3>}$ does not depend on $\hat{y}^{<4>}$

FORWARD PROP

$$a^{<1>} = g_1(W_{aa}\, a^{<0>} + W_{ax}\, x^{<1>} + b_a)$$ tanh / relu

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

generalizing.

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

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

Simplified Notation

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

Q. What does this mean?

A. basically we are stacking W_{aa} & W_{ax} side by side to make "W_a"

If a was 100 dimensional
& x was 10,000 dimensional

W_{aa} $(100, 100)$ — W_{ax} $(100, 10{,}000)$ — DIMENSION

"by stacking"

W_a $(100, 10100)$

also,

$$\left[a^{<t-1>}, x^{<t>}\right] = \begin{bmatrix} a^{<t-1>} \\ \cdots \\ x^{<t>} \end{bmatrix}$$

100 (for $a^{<t-1>}$), 10000 (for $x^{<t>}$), 10,100 (total)

Hence,

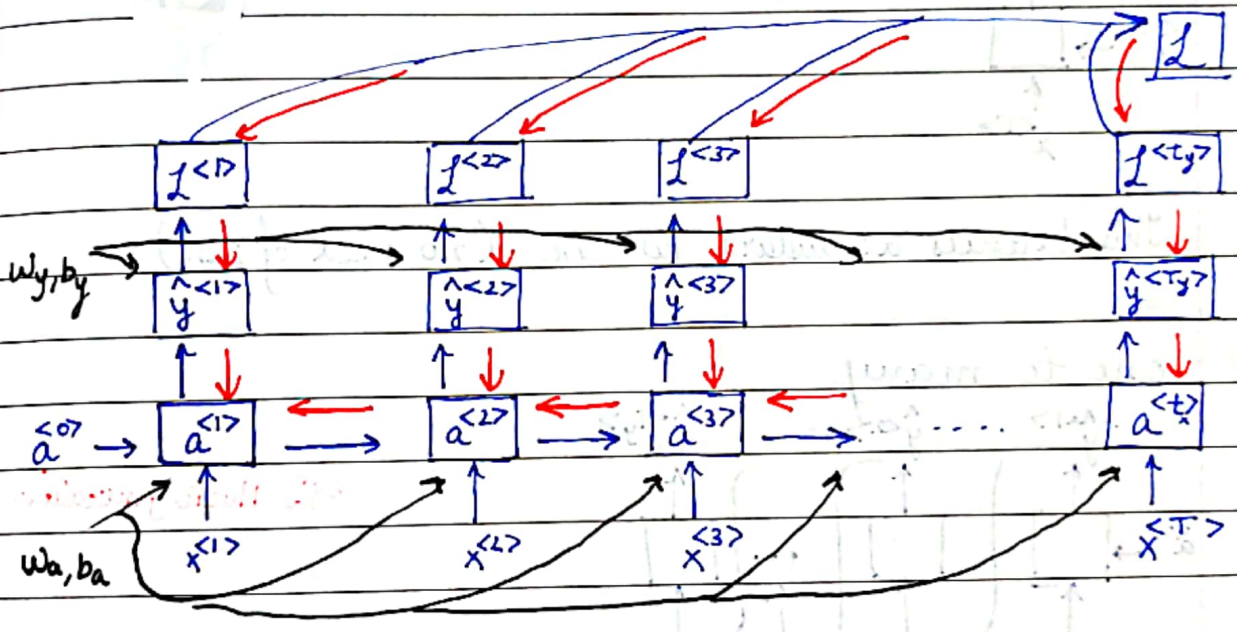
$$\begin{bmatrix} W_{aa} \vdots W_{ax} \end{bmatrix} \begin{bmatrix} a^{<t-1>} \\ x^{<t>} \end{bmatrix} = W_{aa} a^{<t-1>} + W_{ax} x^{<t>}$$

BACK PROP

activations are obtained moving left to right in our network.

so parameters are updated moving right to left.

Drawing a simplified diagram:-



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

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

↳ Overall loss (considering each word)

DATE
PAGE

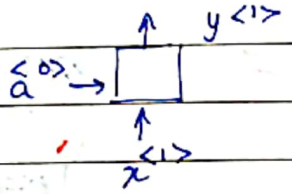
DIFFERENT TYPES OF RNN

so far we have only seen networks which have $t_x = t_y$ but this won't always be the case.

such as one sentence in one language won't be the same in different language.

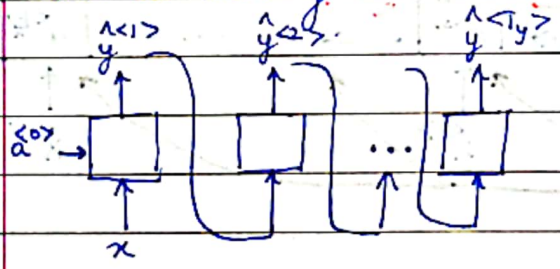
MACHINE TRANSLATION

1) One to one



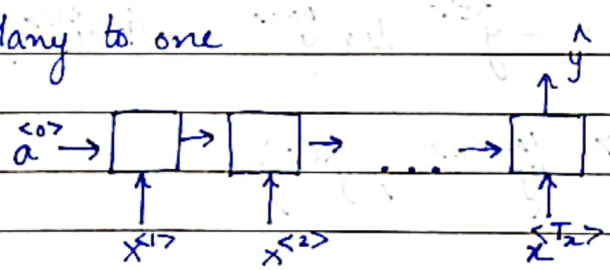
This becomes a standard network (no need of rnn)

2) one to many



eg: Music generation

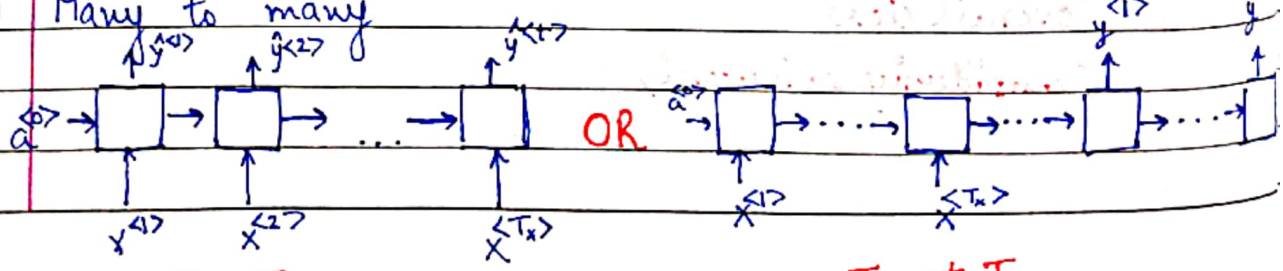
3) Many to one



eg: sentiment classification

x = text
y = rating

4) Many to many



$T_x = T_y$

$T_x \neq T_y$

LANGUAGE MODEL AND SEQUENCE GENERATION

speech recognition system.

language model does is, given any sentence, its job is tell you what is the probability of sentence.

How to proceed with building a language model :-

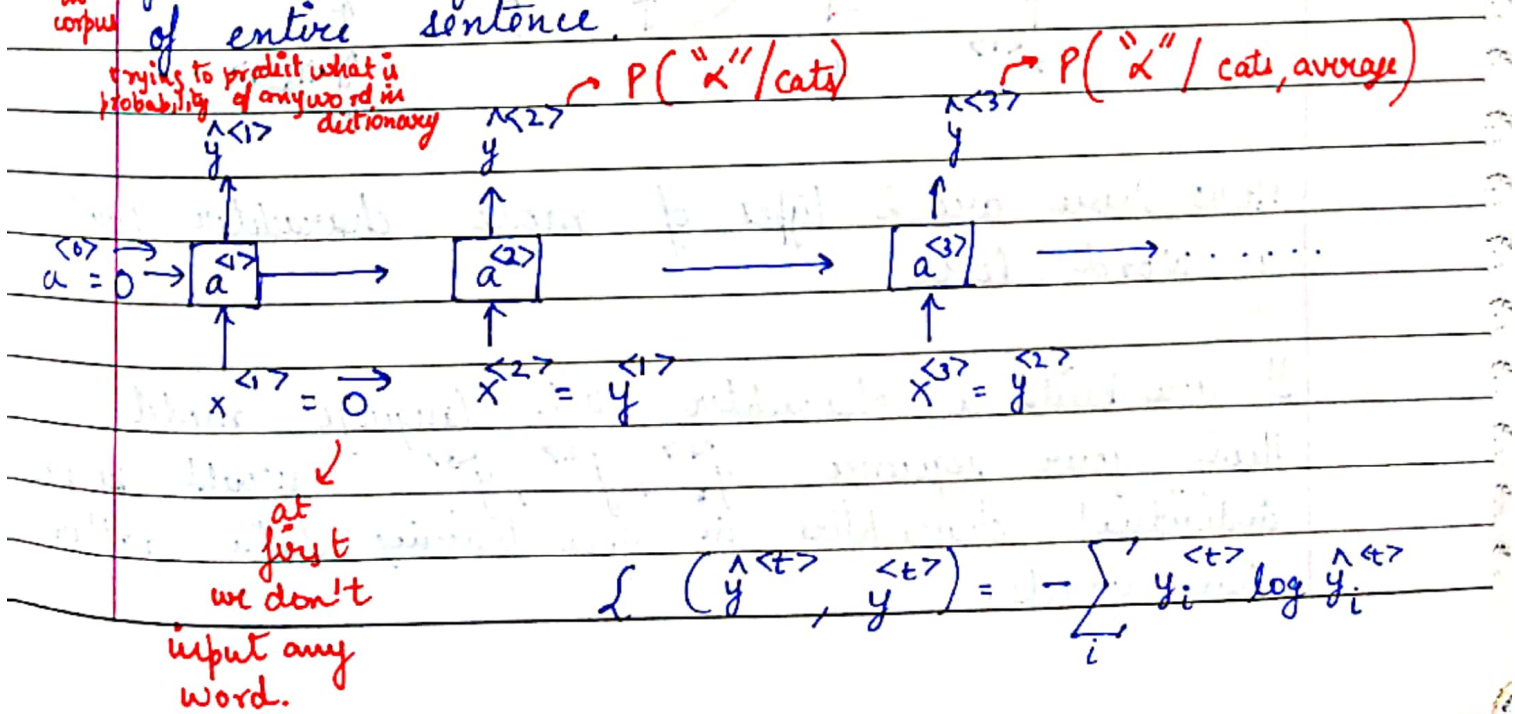
1) first take a large corpus of words. eg 10,000 words.

2) tokenize the given sentence and use O.H.E acc. to the corpus of words.

3) use this example and model to understand.

Assume the sentence is "Cats average 15 hours of sllleep a day."

then we need to predict the exact sentence. each word will have some probability of occurence in accordance to the words occured before. we need to find prob. of each word and then find prob. of entire sentence.



So to find probability of entire sentence.

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

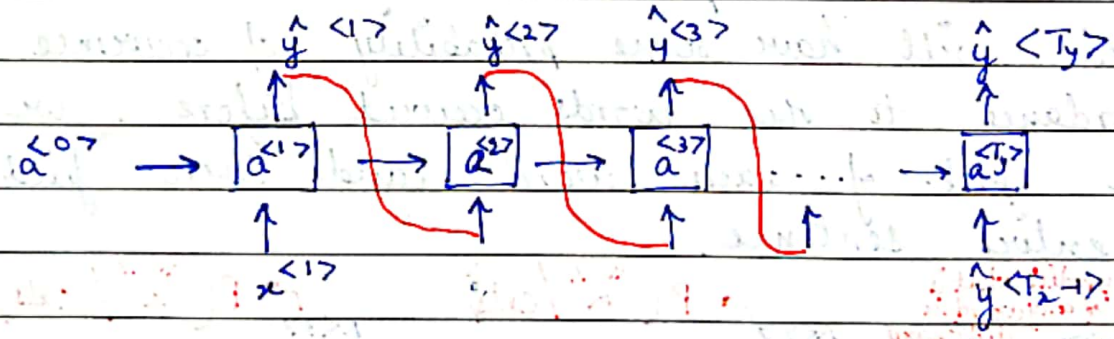
$$= P(y^{<1>}) \times P(y^{<2>} / y^{<1>}) \times P(y^{<3>} / y^{<1>} y^{<2>})$$

SAMPLING NOVEL SEQUENCES

In the earlier network, after finding probability of each word for $\hat{y}^{<1>}$, use np.random to select a word at random.

Then replace $y^{<1>}, y^{<2>} \ldots y^{<t>}$ with $\hat{y}^{<1>}, \hat{y}^{<2>}, \ldots \hat{y}^{<t>}$

So the model becomes.



Now there are 2 types of mode character level & word level

If you build a character level language model then your sequence $y^{<1>}, y^{<2>}, y^{<3>}$ would be the individual characters in your training data rather than words.

DATE
PAGE

Charachter level pros :-

1) don't have to worry about unknown word tokens.

cons :-

1) we end up with much longer sequences, so CLH are not ~~god~~ good at capturing long range dependencies between how the earlier part of sentence affects the later part of sentence.
2) more computationally expensive

VANISHING GRADIENT PROBLEMS

It turns out that the basic RNN we've seen so far is not very good at capturing very long term dependencies.

consider a very deep neural network, then the gradient from this output y would have a very hard time propogating back to affect the weights of these earlier layers, to affect the computations of the earlier layers

It can be quite difficult because of the same vanishing gradients problem for the output of the errors associated with the later time step to affect the computations that are earlier.

In practice, what this means is it might be difficult to get a neural network to realize it needs to memorize.

So basically did you see a singular noun or a plural noun so that later on in the sequence it can generate either was or were, depending on whether it was singular or plural.

Because of this problem, the basic RNN model has many local influences, meaning that the output $\hat{y}^{<3>}$ and a value here is mainly influenced by inputs that are somewhat close.

AVOIDING VANISHING GRADIENTS

We have 2 models GRU & LSTM to avoid vanishing gradients.

GRU

As we read sentence from left to right, the GRU unit is going to have a new variable called C, which stands for cell, for memory cell.

What the memory cell do is it will provide a bit of memory.

Remember, for example, whether cat was singular or plural, so that when it gets much further into the sentence it can still work on the consideration whether the subject of the sentence was singular or plural.

The important idea of the GRU, it will be that we'll have a gate.

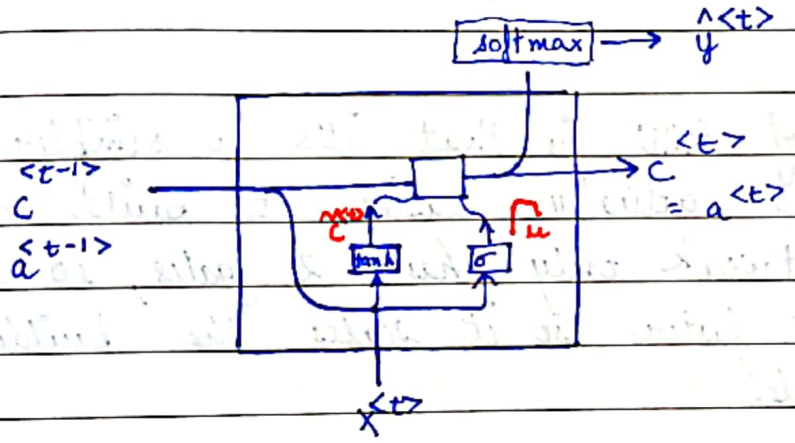
developing intuition of how GRU's work, think of Gamma_u this gate value as being always 0 or 1.

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

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

↳ "update" value of $c^{<t>}$

next the key part of GRU is the above eqn, which is that we have come up with a candidate where we're thinking of updating C using c tilde and then the gate will decide whether or not we actually update it.



GRU

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

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

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

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

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

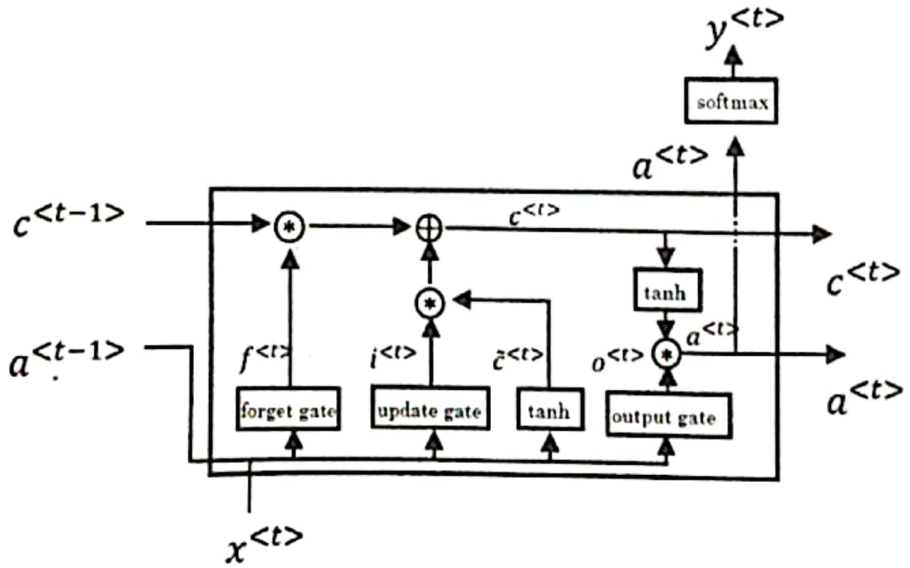
$$\hat{y}^{<t>} = g\left(a^{<t>}\right)$$

LSTM in pictures

$$\tilde{c}^{<t>} = \tanh(W_c[a^{<t-1>}, x^{<t>}] + b_c)$$
$$\Gamma_u = \sigma(W_u[a^{<t-1>}, x^{<t>}] + b_u)$$
$$\Gamma_f = \sigma(W_f[a^{<t-1>}, x^{<t>}] + b_f)$$
$$\Gamma_o = \sigma(W_o[a^{<t-1>}, x^{<t>}] + b_o)$$
$$c^{<t>} = \Gamma_u * \tilde{c}^{<t>} + \Gamma_f * c^{<t-1>}$$
$$a^{<t>} = \Gamma_o * \tanh c^{<t>}$$

Advantage of GRU is that it's a simpler model And so it's actually easier to build a much bigger network only has 2 gates, so computation runs a bit faster so it scales the building, somewhat bigger models.

But the LSTM is more powerfull & more flexible since there's 3 gates instead of 2



DATE
PAGE

WORD EMBEDDINGS

earlier we have studied ONE. One weakness of this representation is that it treats each word as a thing unto itself, and it doesn't allow an algorithm to easily generalize the cross words.

To explain this consider an example :-

If a learning algorithm has learned that "I want a glass of orange juice" is a likely sentence, if it sees "I want a glass of apple ____" It might still not learn juice here. As far as it knows the relationship b/w apple and orange is not any closer as the relation b/w any of the other words man, woman, king, queen, orange.

So, it's not easy for the learning algorithm to generalize from knowing that orange juice is a popular thing to recognizing that apple juice might also be a popular thing. This happens because the product b/w any 2 different one-hot vector is zero. you couldn't distance b/w any pair of these vectors is also the same. So it just doesn't know that apple & orange are much more similar than king & orange or queen & orange.

featurized representation or word embedding can be used.

← WORDS →

FEATURES	Man (5391)	Woman (9853)	King (4914)	Queen (7157)	Apple (456)	Orange (6257)
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	0.09	0.01	0.02	0.01	0.95	0.97
⋮						
size						
cost						

a column in this matrix represents a word and its vector. This is feature representation a vector's dimension would depend on the feature space.

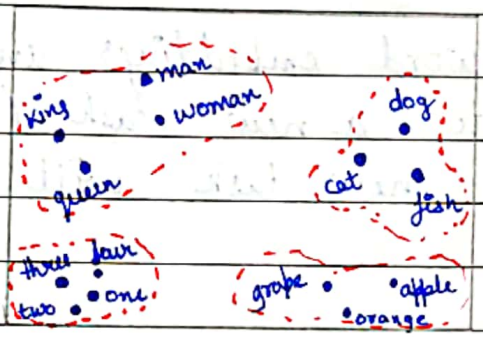
This featurized representation we learn, will allow an algorithm to quickly figure out that apple and orange are more similar than say, king and orange or queen and orange.

One thing you can do is take this 300 dimensional data and embed it say, in a 2-D space so that you can visualise them.

One comman algorithm for doing this is the t-SNE algorithm

This gives you a sense that word embeddings algorithms like this can learn similar features. By visualising the given 2-D space, the similar or correlated words are mapped somewhat together.

300D
↓ t-SNE
2D



USING WORD EMBEDDINGS

You can use these word embeddings and plug them into NLP applications

Taking our example of named entity recognition example.

previously we had used one hot encodings to represent these words, $x^{<1>}, x^{<2>} \ldots x^{<t>}$ but replacing them with these word embeddings. Now knowing that orange and apple are very similar will make it easier for your learning algorithm to generalize to figure out that Robert Lin is also a human, is also's a person's name

∴ This makes the learning model more capable as it is able to find some correlation b/w different words.

So steps are as follows :-

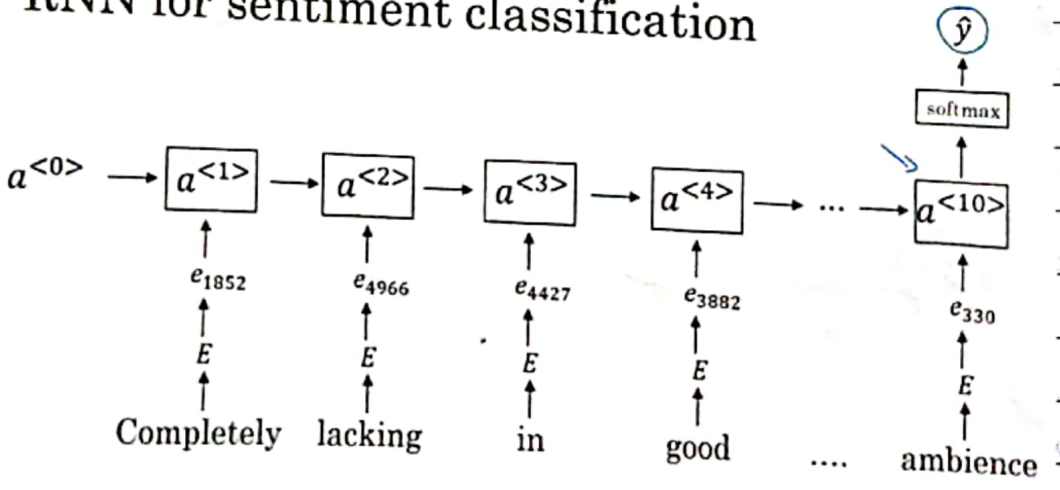
1) Learn word embeddings from large text corpus. (or download pre-trained embeddings online)

2) Take these word embeddings and transfer the embedding to a new task (as data). An example of new task will be "named entity example"

So now instead of using a 10,000 dimensional one-hot vector, you can now instead ... use ... a 300 dimensional dense vector.

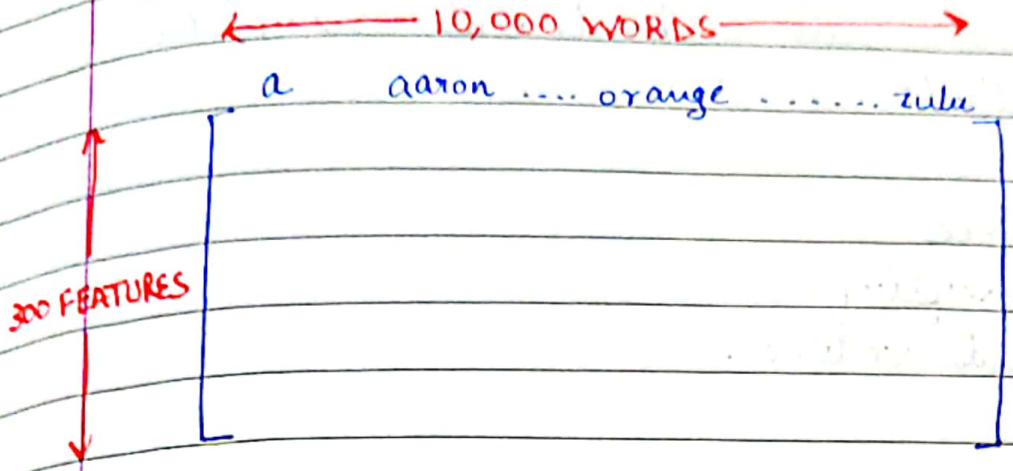
So word embeddings tend to make the biggest difference when the task you are trying to carry out has a relatively smaller training set.

RNN for sentiment classification



DATE
PAGE

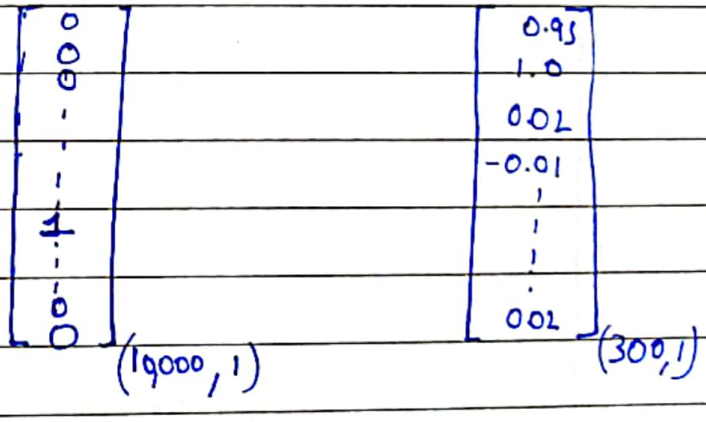
Embedding matrix notation



In order to find the feature vector of any word let's say orange, then :-

DIMENSION

E $(300, 10{,}000)$ $*$ $O.H.E_{orange}$ $(10{,}000, 1)$ $=$ feature vector $(300, 1)$ (e_{orange})



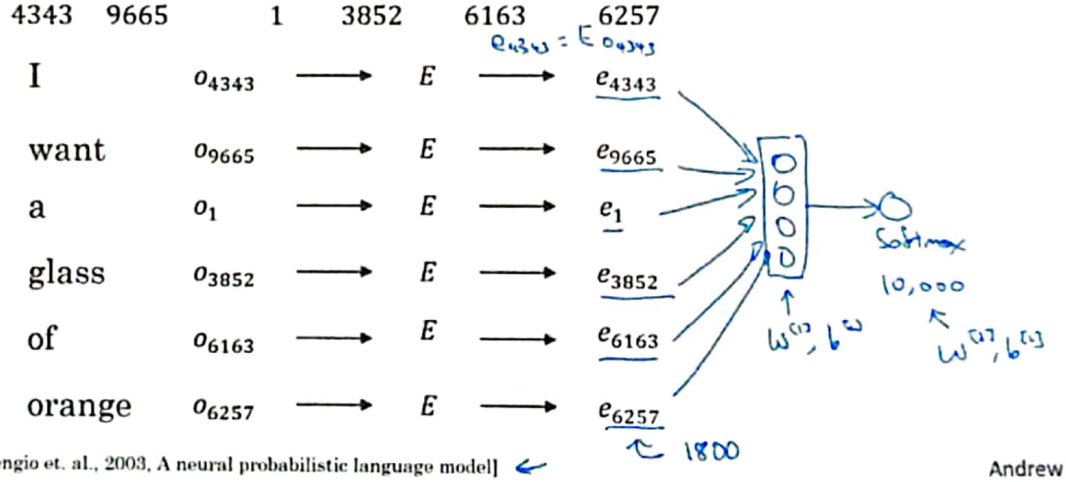
Word embeddings can be learnt using the following methods.

1) word to Vec
2) negative sampling
3) GloVe Word vectors.

Building a neural language model is the small way to learn a set of embeddings.

Neural language model

I	want	a	glass	of	orange	juice.
4343	9665	1	3852	6163	6257	



[Bengio et. al., 2003, A neural probabilistic language model]

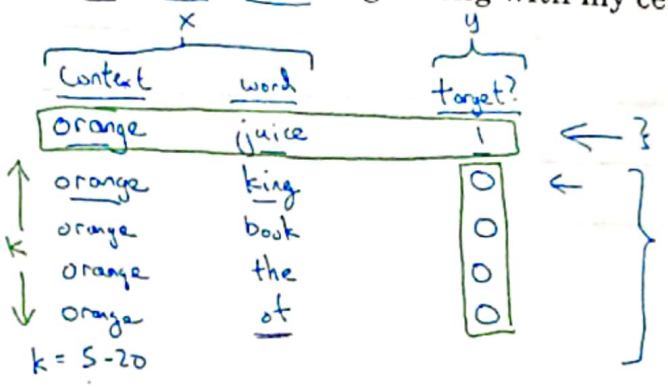
Andrew Ng

NEGATIVE SAMPLING

We are going to create a supervised learning problem where the learning algo inputs x, inputs this pair of words, and it has to predict the target label to predict the output label y.

Defining a new learning problem

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



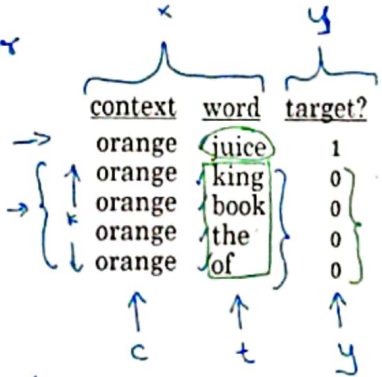
[Mikolov et. al., 2013. Distributed representation of words and phrases and their compositionality]

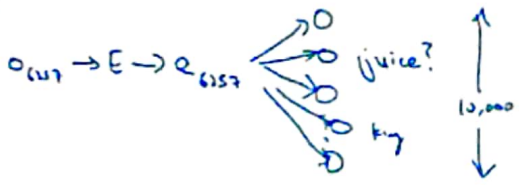
Andrew Ng

Model

Softmax: $p(t|c) = \frac{e^{\theta_t^T e_c}}{\sum_{j=1}^{10,000} e^{\theta_j^T e_c}}$ (10,000-way softmax)

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





Andrew Ng

APPLICATIONS THAT USE WORD EMBEDDINGS

Sentiment classification problem

x ⟶ y

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 ambience.	★☆☆☆☆

10,000 → 100,000 words

Andrew Ng

Consider a review "Completely lacking in good taste, good service, and good ambience" the word good apears a lot.
So if you use an algorithm like this that ignores word order and just sums or averges all of the embeddings for the different words, then you end up having a lot of the representation of good in your final feature vector and your classifier will probably think this is a good review even though this actually very harsh.

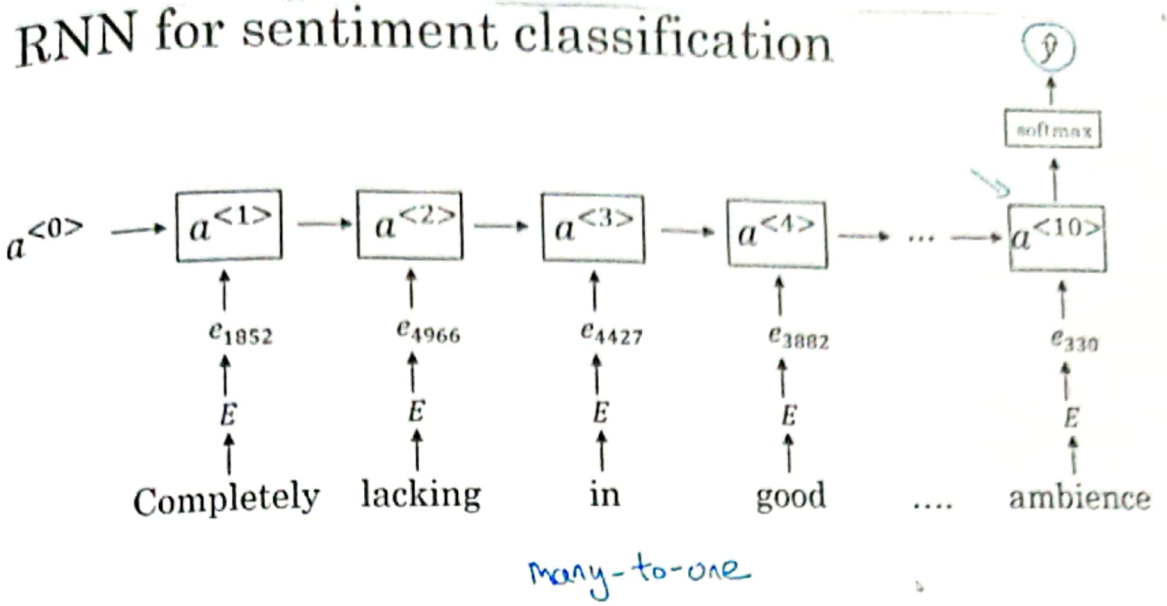
for such tasks we should rather use a more sophisticated model "RNN"

Simple sentiment classification model

The	dessert	is	excellent	
8928	2468	4694	3180	★★★★☆

The o_{8928} ⟶ E ⟶ e_{8928}

RNN for sentiment classification



many-to-one

Andrew Ng

The job of the RNN is to then compute the representation at the last time step that allows you to predict y-hat.

With an algorithm like this, it will be much better at taking word sequence into account & realise that "things are lacking in good taste" is a negative review and "not good" a negative review unlike the previous algorithm which just sums everything together into a by-word vector mush & doesn't realise that "not good" has a very different meaning than the words "good" or "lacking in good taste" and so on.

BASIC MODELS

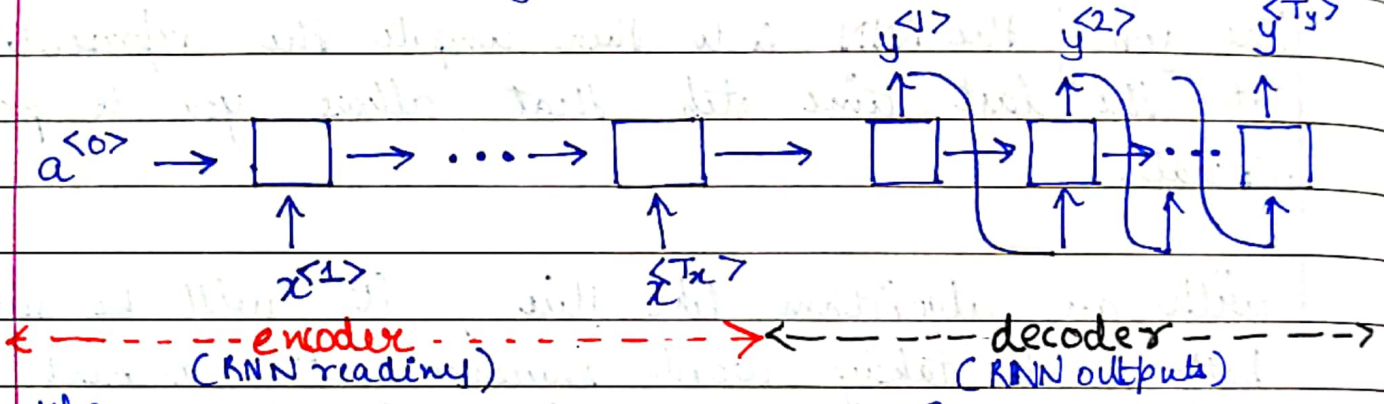
SEQUENCE TO SEQUENCE MODEL

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

Jane visite l'Afrique en septembre

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

Jane is visiting Africa in September.

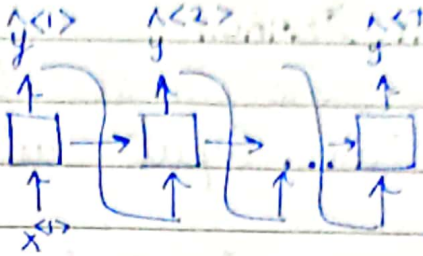


This model simply uses an encoding network whose job it is to find an encoding of the input french sentence and then using a decoding network to then generate the corresponding english translation.

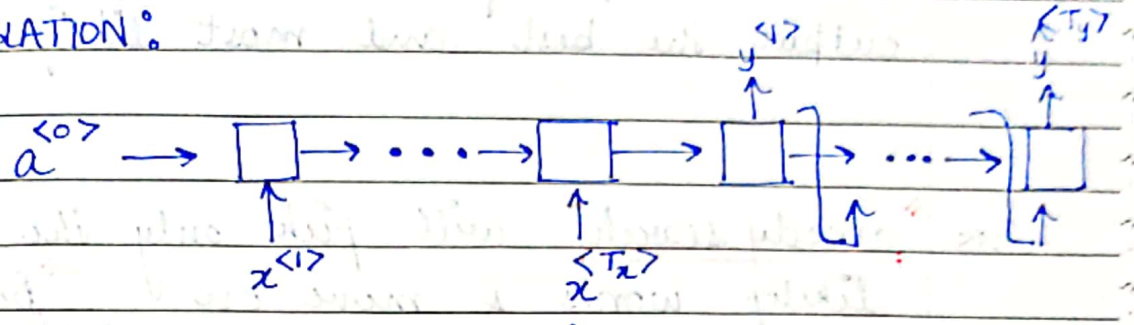
Keep in mind you want the most likely translation or you don't want to randomly choose in caption

Language model & machine translation model.

LANGUAGE MODEL :



MACHINE TRANSLATION :



The decoder of the machine translation model is similar to the language model

so basically in machine translation model, instead of starting along with the vector of all zeros, it has an encoder network that figures out some representation for the input sentence, so it takes this input sentence and starts off the decoder network with representation of input sentence.

This is a conditional language model, instead of modelling the probability of any sentence, it is now modeling the probability of say, the output english translation, conditioned on some input french sentence

FINALLY, implementing the model will tell you what is the probability of different english translations given that french input.

DATE
PAGE

BEAM SEARCH

Picking the most likely sentence or translation of an input using beam search.

* given an input french sentence, you want to output the best and most likely English translation.

* "greedy search" will pick only the one most likely words & move on, "Beam search" instead can consider multiple alternatives.

* Beem search parameter b, beam width if = 3 then rather than considering one possibility it will consider top 3 at the time.

1ST step
run the input french sentence through this encoder network, then this first step will then decode the network, softmax output overall 10,000 possibilities

top 3 possibilities of the 10,000 possibilities will then be chosen & kept in memory.

2nd step
Having picked in, June & september, beam search will now for each of these 3 choices considered what should be the 2nd word. So maybe after "in", the most probable 2nd word is "a".

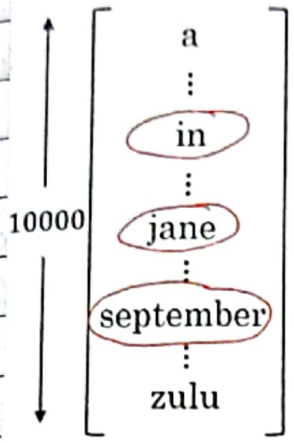
NOTE :

If B·W = 1, then Beam search essentially becomes greedy search.

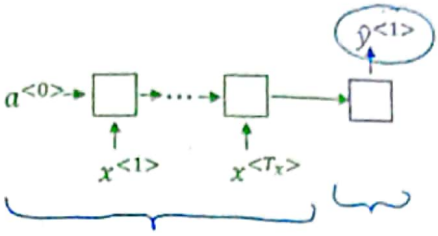
Beam search algorithm

B = 3 (beam width)

Step 1



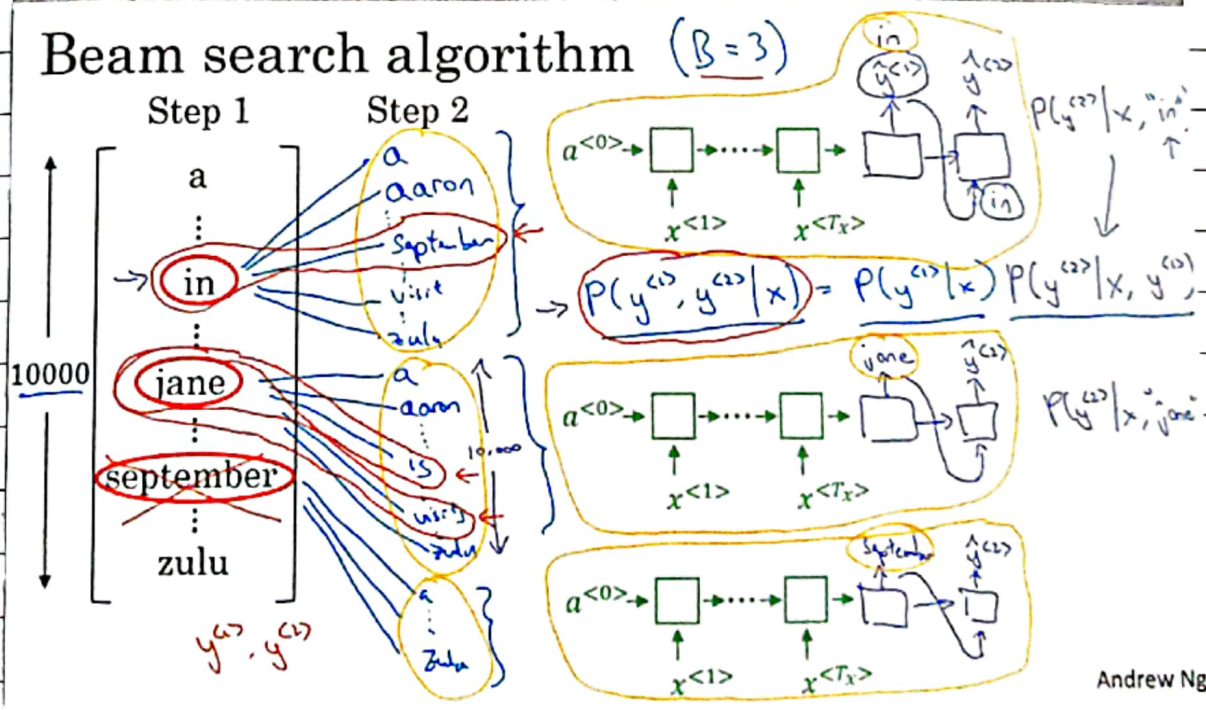
$\rightarrow P(y^{<1>} \mid x)$



Andrew Ng

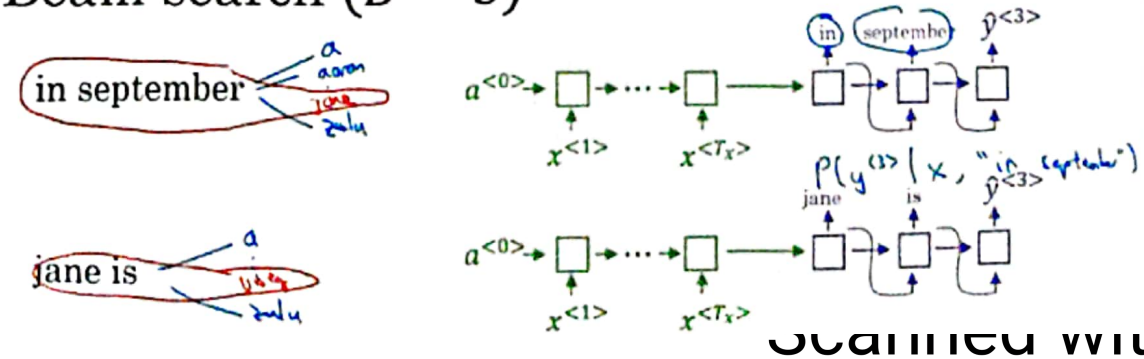
Beam search algorithm (B = 3)

Step 1 Step 2



Andrew Ng

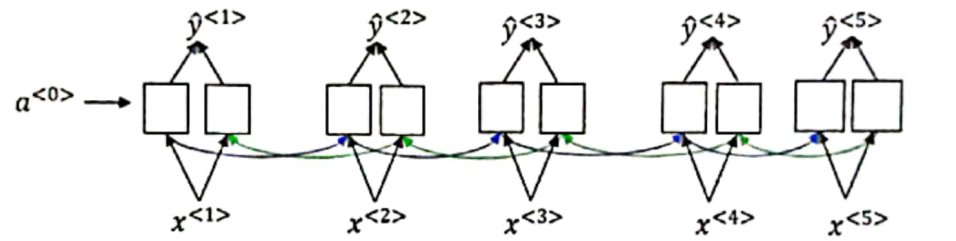
Beam search (B = 3)



BI DIRECTION MODELS

Basically in BI-DIRECTION RNN's one can take information from bothe earlier & later in the sequence.

In a sequene, in order to predict sometimes it is important to gain information from later words in the sequence.

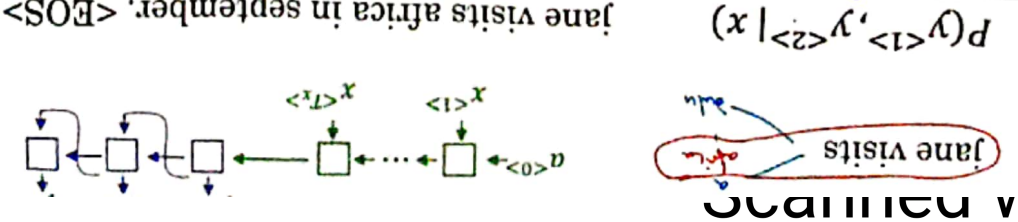


[Bahdanau et. al., 2014. Neural machine translation by jointly learning to align and translate] Andrew Ng

ATTENTION MODEL

modification to the encoder-decoder network is the attention model.

given a whole paragraph in french, then the previous models or the encoder-decoder model will first memorize the entire input and then translate which is not feasible for many tasks.



encoder - decoder works quite well for short sentences but not for long sentences.

while translating the first word, what part of the input french sentence should you be looking at?
you should be looking at words in the vicinity but not way at the end of sentence

weights
$\alpha^{<t,t'>}$ that tells it, when you are trying to generate the "T" english word, how much should you be paying attention to T' (french words)

This allows on every time step to look only maybe within a local window of the french sentence to pay attention to, when generating a spicific english word.

this alpha parameter tells us how much the context would depend on the features we're getting or the activations we're getting from the different time step.

the way we define the context is actually a weighted sum of the features from the different time steps weighted by these attention weights

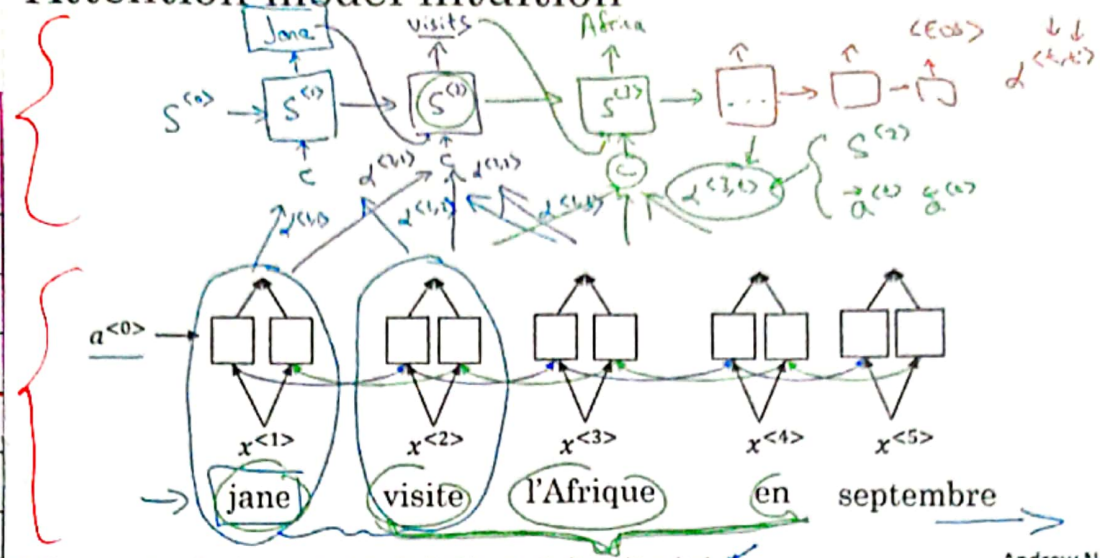
SO BASICALLY
When you are generating the t of output words, how much you should be paying attention to the ~~t-prime~~th t' input word

RNN

+

BI-DIREC RNN

Attention model intuition



[Bahdanau et. al., 2014. Neural machine translation by jointly learning to align and translate]

Andrew Ng

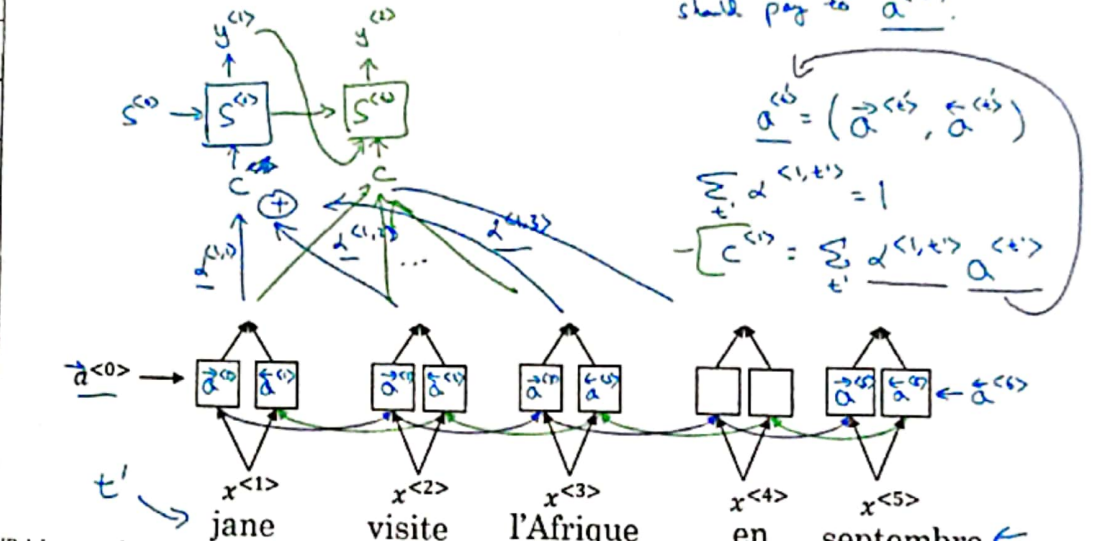
Attention model

$\alpha^{\langle t,t' \rangle}$ = amount of "attention" $y^{\langle t \rangle}$ should pay to $a^{\langle t' \rangle}$

$a^{\langle t' \rangle} = (\overrightarrow{a}^{\langle t' \rangle}, \overleftarrow{a}^{\langle t' \rangle})$

$\sum_{t'} \alpha^{\langle 1,t' \rangle} = 1$

$c^{\langle 1 \rangle} = \sum_{t'} \alpha^{\langle 1,t' \rangle} a^{\langle t' \rangle}$



[Bahdanau et. al., 2014. Neural machine translation by jointly learning to align and translate]

Andrew Ng

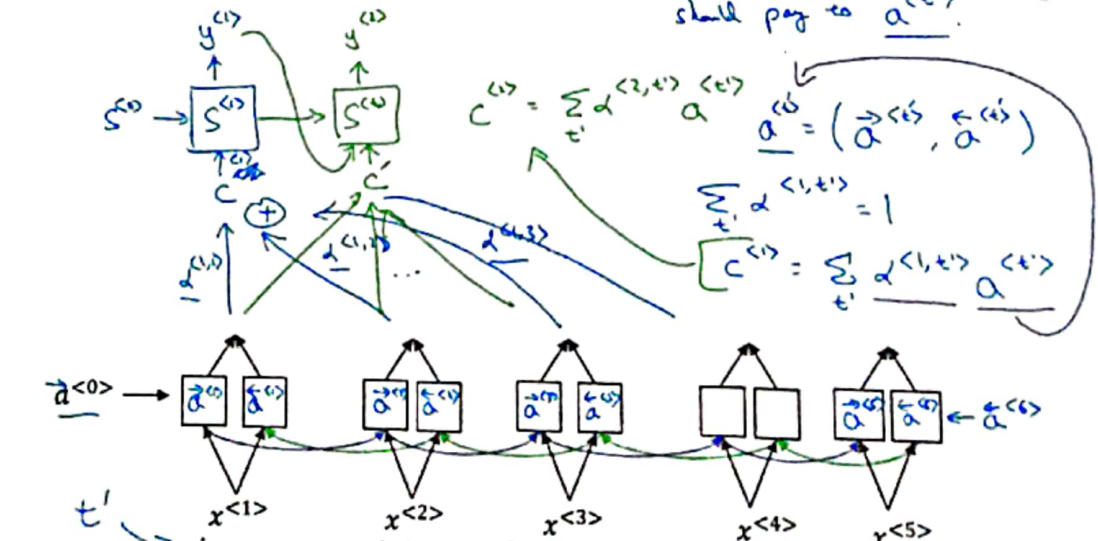
Attention model

$\alpha^{\langle t,t' \rangle}$ = amount of "attention" $y^{\langle t \rangle}$ should pay to $a^{\langle t' \rangle}$

$c^{\langle 1 \rangle} = \sum_{t'} \alpha^{\langle 1,t' \rangle} a^{\langle t' \rangle}$

$a^{\langle t' \rangle} = (\overrightarrow{a}^{\langle t' \rangle}, \overleftarrow{a}^{\langle t' \rangle})$

$\sum_{t'} \alpha^{\langle 1,t' \rangle} = 1$



[Bahdanau et. al., 2014. Neural machine translation by jointly learning to align and translate]

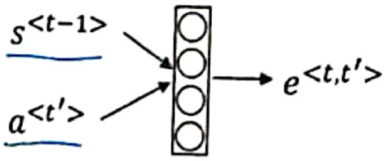
Andrew Ng

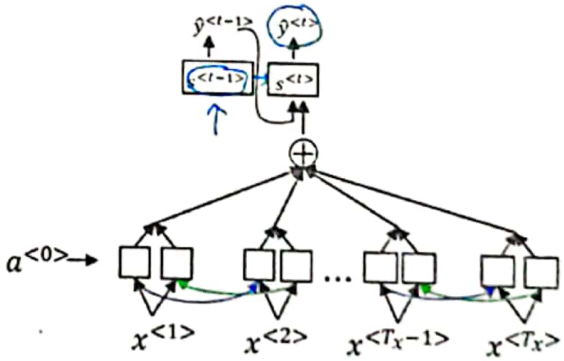
This little neural network does a pretty decent job telling you how much attention $y^{<t>}$ should pay to $a^{<t'>}$ and this formula makes sure that the attention weights sums to one and then as you chug along generating one word at a time, this neural network actually pays attention to the right parts of sentence that learns all this using gradient descent.

Computing attention $\alpha^{<t,t'>}$

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

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





[Bahdanau et. al., 2014. Neural machine translation by jointly learning to align and translate]
[Xu et. al., 2015. Show attention and tell: neural image caption generation with visual attention]

Andrew Ng

TRANSFORMERS

started with RNN and found that it has problem with vanishing gradients.

We then looked into LSTM & GRU as a way to resolve many of those problems where you use gates to control the flow of information. you can ingest an entire sentence all at the same time rather than processing it one word at a time from left to right.

RNN processes one output at a time. But a transformer network has **self attention**, so if you have a sentence of 5 words then it will end up computing representations for those 5 words.

This will be an attention based way of computing representations for all the words in your sentence in parallel.

turns out that these representations, which will be very rich representations can be used for machine translation or other NLP tasks.

SELF ATTENTION

you calculate, where you create attention-based representations for each of the words in your input sentence.

In self attention, for every word, you have 3 values — query, key & value.

The matrices W^Q, W^K, W^V are parameters of this learning algorithm & they allow you to calculate query, key & value vector for each word.

consider the word $x^{<3>}$ afrique, $q^{<3>}$ is a question that you get to ask about $x^{<3>}$

So we then compute the inner product between $q^{<3>}$ & $k^{<1>}$, so basically this will tells us how good of an answer (key) $k^{<1>}$ is for our question (query) $q^{<3>}$

Then we compute product b/w $q^{<3>}$ & $k^{<2>}$ & see good of an answer $k^{<2>}$ was for $q^{<3>}$. this way we do for all words.

goal of this operation is to pull up most information that's needed to help us compute the most useful representation $A^{<3>}$ here.

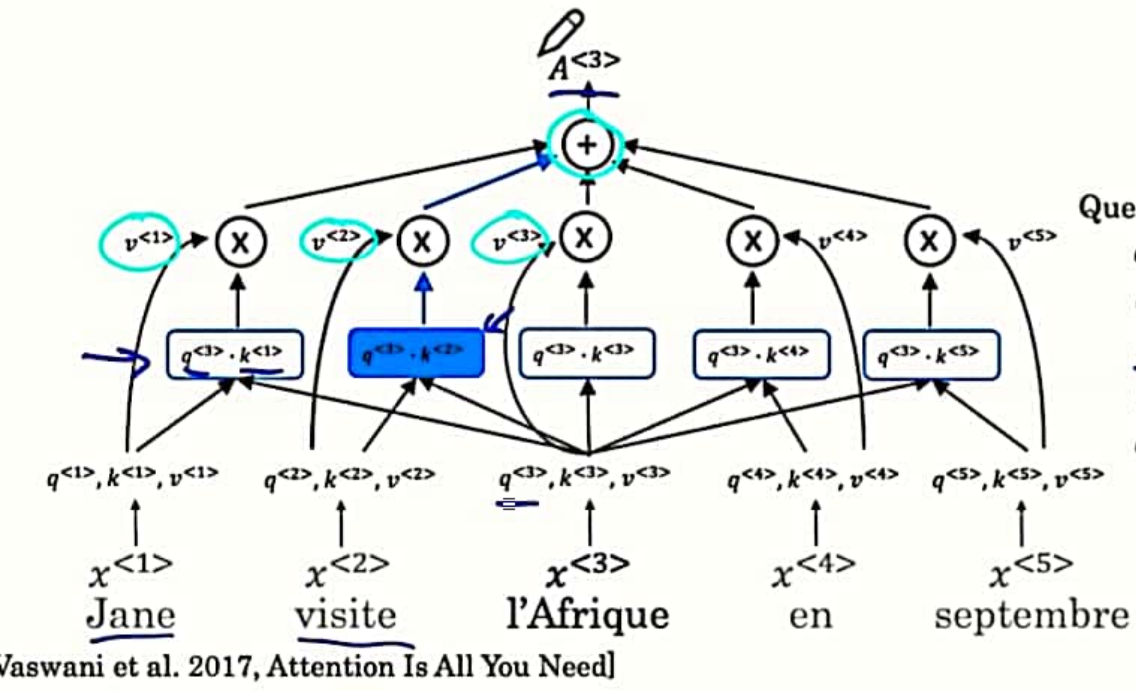
then these values $(q^{<3>} \cdot k^{<i>})$ gets multiplied with the respective $v^{<i>}$.

finally, adding all the values gives us $A^{<3>}$

$$A(q,k,v) = \sum_i \frac{\exp(q \cdot k^{<i>})}{\sum_j \exp(q \cdot k^{<j>})} \times v^{<i>}$$

Self-Attention

$$A(q, K, V) = \sum_i \frac{\exp(q \cdot k^{<i>})}{\sum_j \exp(q \cdot k^{<j>})} v^{<i>}$$



[Vaswani et al. 2017, Attention Is All You Need]

Query (Q)	Key (K)	Value (V)
$q^{<1>}$	$k^{<1>}$	$v^{<1>}$
$q^{<2>}$	$k^{<2>}$	$v^{<2>}$
$q^{<3>}$	$k^{<3>}$	$v^{<3>}$
$q^{<4>}$	$k^{<4>}$	$v^{<4>}$
$q^{<5>}$	$k^{<5>}$	$v^{<5>}$

$q^{<3>} = W^Q \cdot x^{<3>}$

$k^{<3>} = W^K \cdot x^{<3>}$

$v^{<3>} = W^V \cdot x^{<3>}$

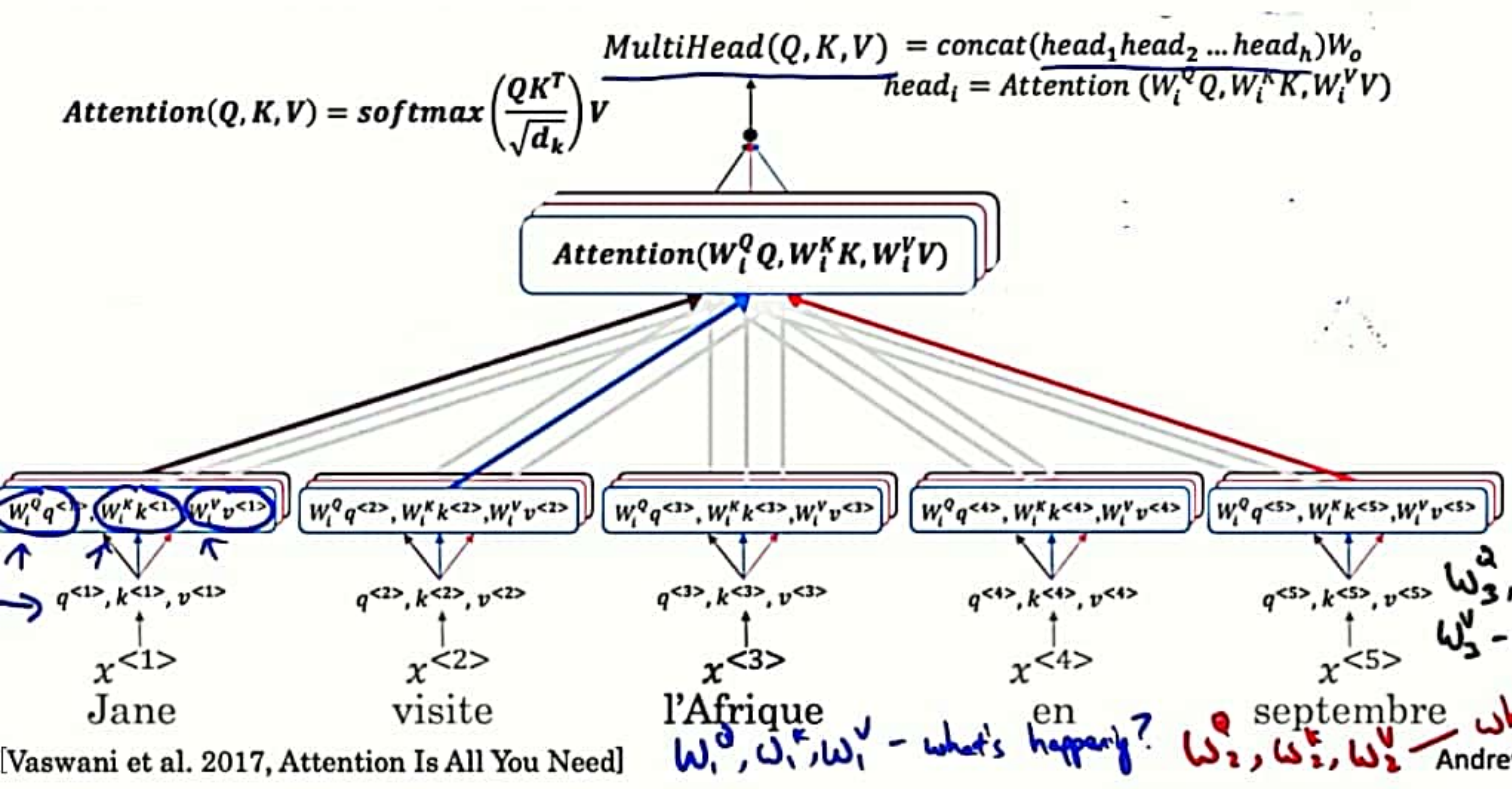
Andrew Ng

Multi-Head Attention

$$Attention(Q,K,V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

$$MultiHead(Q,K,V) = concat(head_1 head_2 \dots head_h)W_o$$

$$head_i = Attention\,(W_i^Q Q, W_i^K K, W_i^V V)$$



[Vaswani et al. 2017, Attention Is All You Need]

Andrew Ng

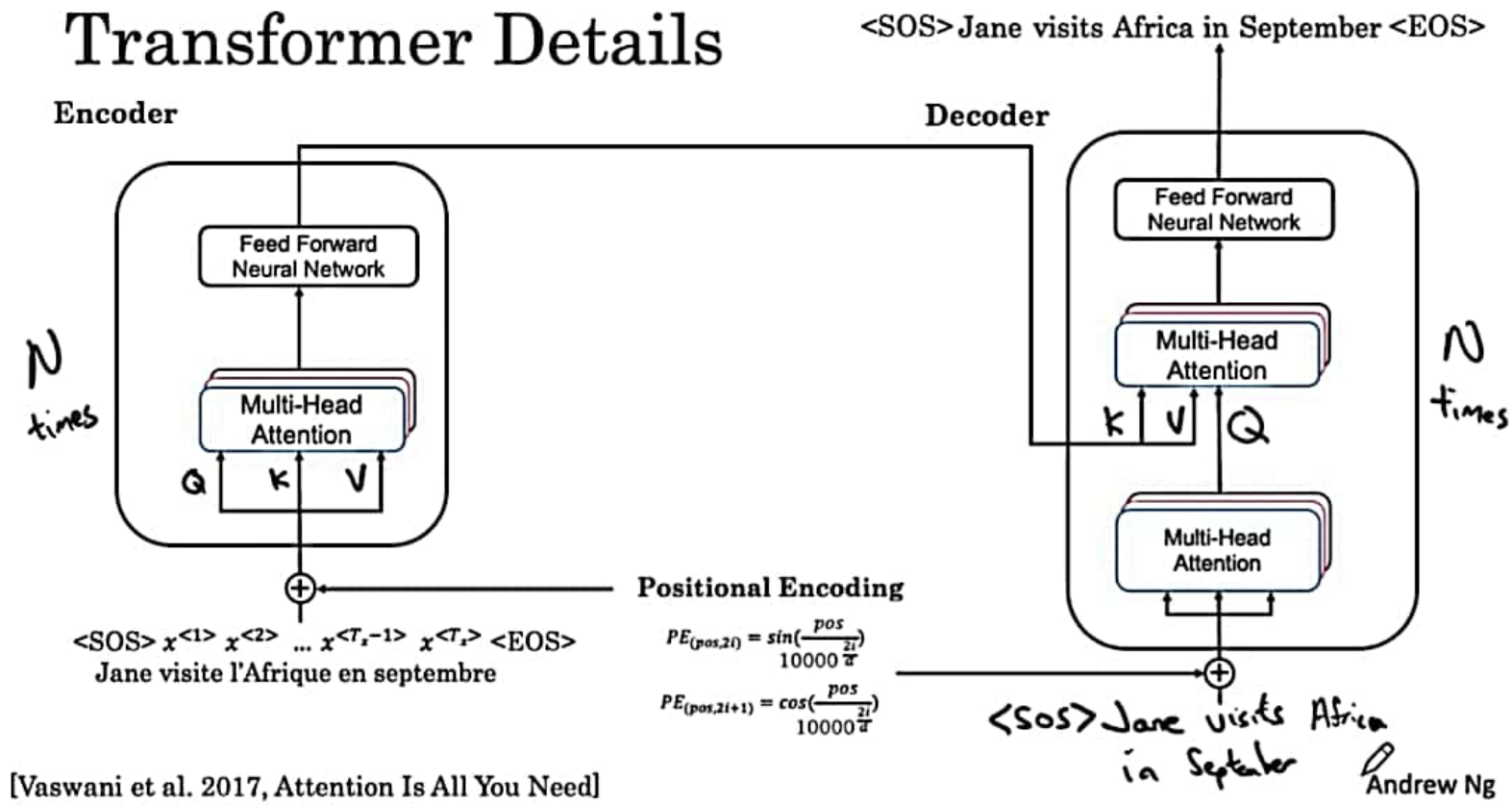
MULTI HEAD ATTENTION

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

In case of multi head attention, now we will have a parameters W^Q, W^K, W^V.
these parameters are multiplied with q, K, V.
This way we are able to ask different set of questions for the same word & find the best possible keys acc. to the question all parallely.

If we will take 3 heads, then basically for each word we are asking 3 set of questions. So for h no. of heads, the concatenation of these h_i values or A values is used to compute the output of the multi head attention.

Transformer Details



[Vaswani et al. 2017, Attention Is All You Need]

Andrew Ng