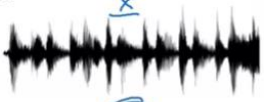
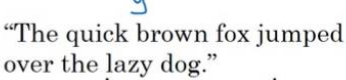






I.	Some of the usage of Sequence models .....	2
II.	A Small demo about the sequence models input: name entity recognition .....	3
III.	Recurrent Neural networks RNN model.....	3
IV.	Language models and Sequence generation.....	7
V.	Vanishing Gradient problem .....	8
VI.	Global recurrent Unit ( GRU ) is the solution : .....	9
VII.	Long Short Term Memory ( LSTM ) .....	13
VIII.	Bidirectional RNN ( BRNN ) :.....	14
IX.	Deep RNN .....	15
X.	What I've learned from the assignments : .....	16
XI.	Word representation.....	17
XII.	Transfer learning and word embeddings: .....	20
XIII.	An interesting similarity between word embeddings and face encoding (face recognition): 21	
XIV.	Word embeddings properties: .....	21
XV.	Embedding Matrix: the mathematic representation of our vocabulary by word embeddings 23	
XVI.	Learning Word embeddings .....	24
XVII.	Sentiment Analysis: an Application using the word embedding techniques.....	30
XVIII.	Debiasing word embeddings: .....	33
XIX.	Sequence to sequence Architecture (many to many RNN ) :.....	34
XX.	Machine translation as building a conditional language model: .....	35
XXI.	Evaluating the machine translations using the BLEU Score: .....	41
XXII.	Attention Model: .....	44
XXIII.	Transformers Intuition : .....	46
XXIV.	Explaining self-attention mechanism: .....	47
XXV.	Multi-Head attention process .....	48
XXVI.	Transformers Network .....	49
XXVII.	Additional details and conclusion: .....	55

# I. Some of the usage of Sequence models

## Examples of sequence data

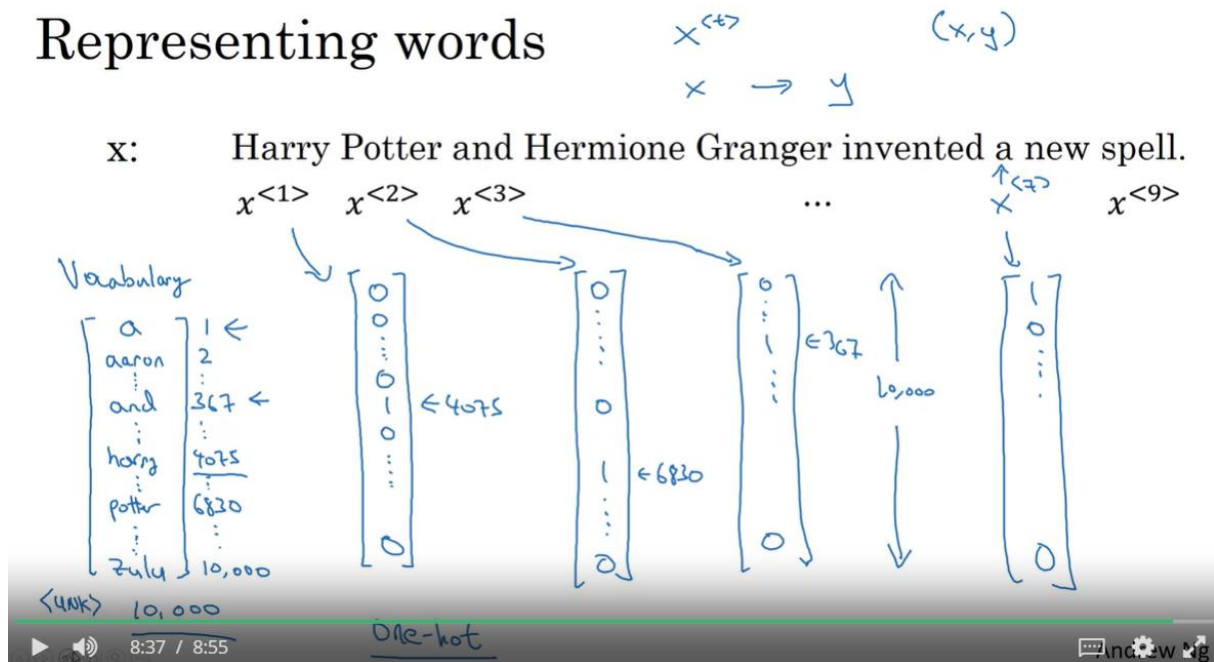
Speech recognition		→	
Music generation		→	
Sentiment classification	"There is nothing to like in this movie."	→	
DNA sequence analysis	AGCCCCTGTGAGGAACTAG	→	AGCCCCTGTGAGGAACTAG
Machine translation	Voulez-vous chanter avec moi?	→	Do you want to sing with me?
Video activity recognition		→	Running
Name entity recognition	Yesterday, Harry Potter met Hermione Granger.	→	Yesterday, <b>Harry Potter</b> met <b>Hermione Granger</b> .

We see that some of the examples has a sequence as an input (X) and a sequence as an output ( Y ) like speech recognition , DNA sequence analysis , Machine translation

And in another cases like Sentiment classification , Name and Video activity recognition we found the sequence data only in the input

And finally , there is examples like Music generation where we found the sequence data in t-he output

## II. A Small demo about the sequence models input: name entity recognition

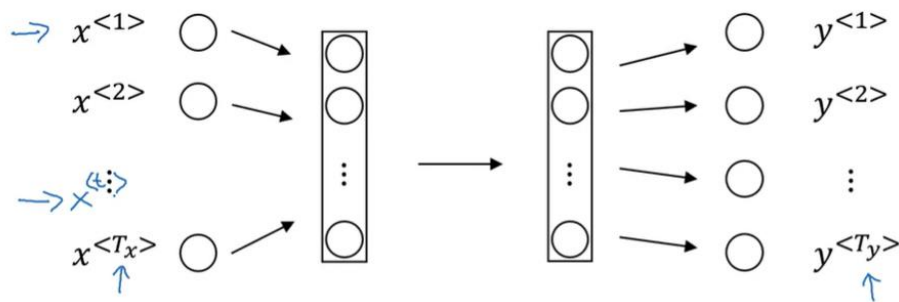


- For name entity recognition we have before everything our Vocabulary vector which contains all the words to recognize ( in this case our vocabulary vector contains 10K words , each word is identified by its position in the vector , the word “and” for example is identified by 367
  - We add in the last index : a special word <unk> to identify the unknown words by the index 10001
- Our dataset will be surely a sentences , each sentence X contain words labeled by  $x^{<t>}$  where t represents its position in the sentence
  - The  $X^{<t>}$  will be represents by a **one-hot vector** with same length as our vocabulary vector ( 10K ) , that means it's in form of vector of zeros and only a one ( 0,0,0,...,1,0,0,...,0 ) . in our case the '1' will be in the index of the specified word ( so  $x^{<3>}$  is 'and' so the 1 will be in position [367] )

## III. Recurrent Neural networks RNN model

Instead of the the classical neural networks , we need a new architecture to handle the sequence data due to some of its weaknesses

## Why not a standard network?



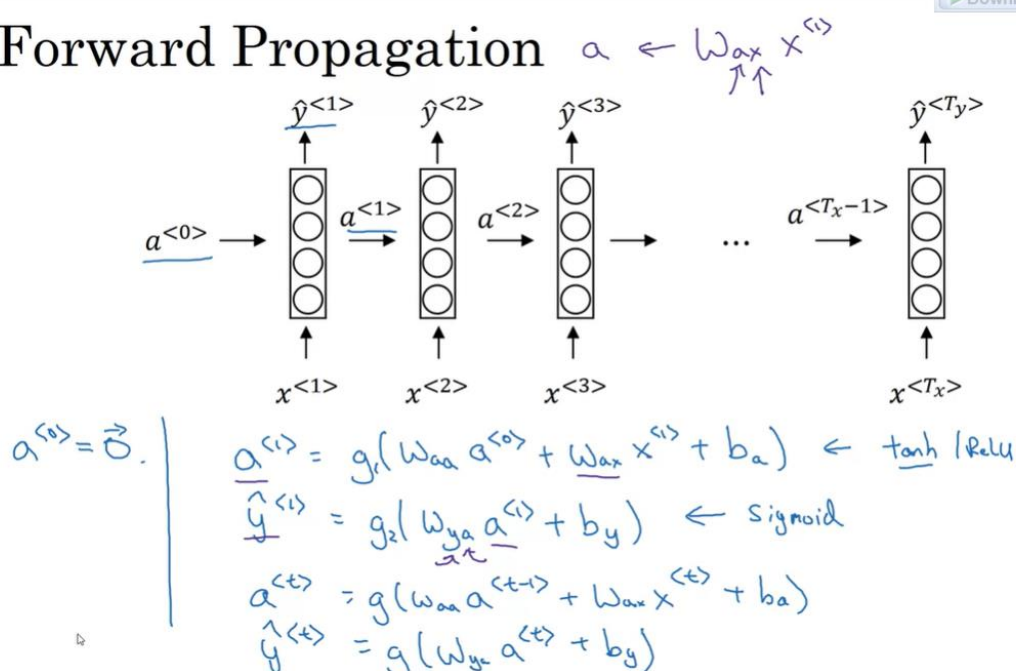
### Problems:

- Inputs, outputs can be different lengths in different examples.
- Doesn't share features learned across different positions of text.

- There isn't any specified length of the input layer and for the output layer also , like we cannot specify the length of the input and/or the output of the Machine translation
- If we regroup all our words in a one vector ( by summing the one-hot vector of each word ) , We lose an important feature in the sequence data which is the order and the position of the text

## A. The New architecture: RNN

### Forward Propagation

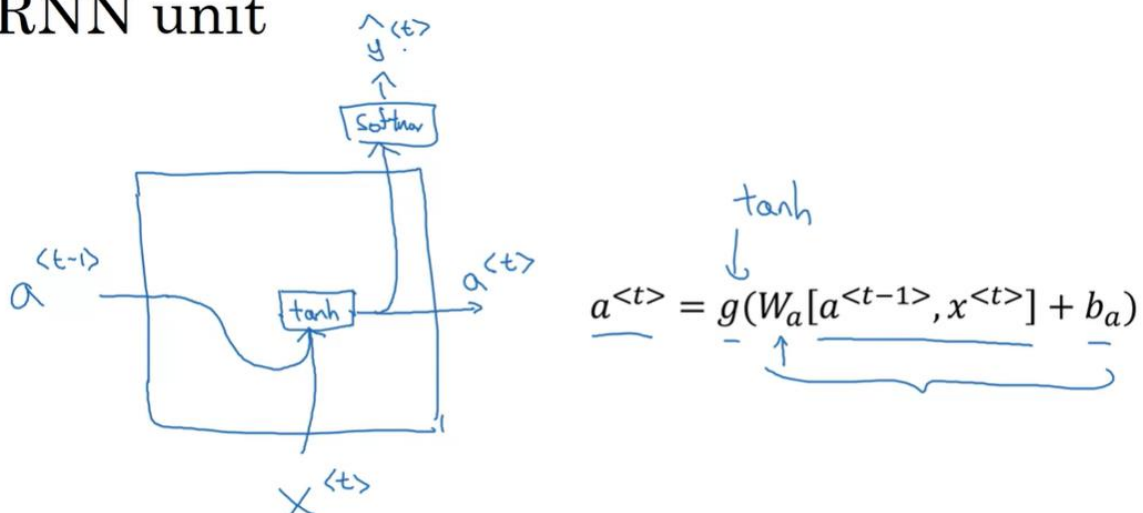


- As the architecture demonstrates and its mathematic formula showed : for every prediction  $y_{<t>}$  related to  $x_{<t>}$  , we are going to use an additional parameter :  $a_{<t>}$  , this parameter uses  $x_{<t-1>}$  to be calculated , so every  $x_{<t>}$  will use the words of the position  $[1,t-1]$  to calculate its prediction which is a thing we didn't see on the classical neural networks , and this is called forward propagation
- The  $g()$  function represents the optimizer function
  - We often use 'tanh' or sometimes 'relu' as the optimizer function to calculate the  $a_{<t>}$
  - We often use "sigmoid" optimizer to calculate the  $y_{<t>}$  ( or "SoftMax" if we found ourselves with categorical-classification problem )
  - In this situation we will get a value between 0 and 1 which represents the probability of a word to be a Named entity , so it's a binary classification => sigmoid is the idea classifier
- $W_{aa}$  ,  $W_{ax}$  ,  $W_{ay}$  ,  $b_y$  and  $b_a$  are hyper parameters , the  $W$ s are matrixes and the  $B$ s are biases
- The  $a_{<t>}$  are called "activation functions"

## B. A RNN UNIT :

This is a general representation of each unit inside the RNN

### RNN unit



## C. The RNNs weaknesses:

The current model isn't perfect to do its work with sequence data , and here is a simple example to demonstrate one of its biggest weakness related to Named Entity Recognition:

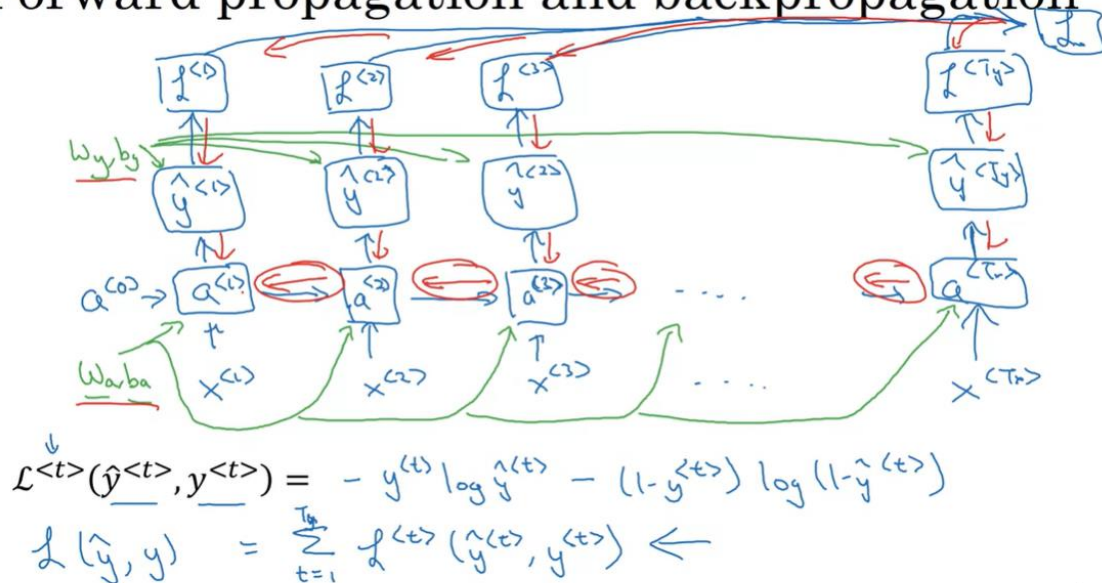
- Given these two sentences:
  - He said "**Teddy** Roosevelt was a great president"
  - He said "**Teddy** bears are on sale"
    - We cannot decide whether "Teddy" is a named entity or not without inspecting the coming words ( $x_{<t+1>}$  ,  $x_{<t+2>}$  , .... )

So, sometimes looking to the precedent entities isn't sufficient to do a correct prediction but we might need the next entities

Solution: Backward propagation with Bidirectional RNN (BRNN)

## D.Forward vs Backward propagation :

### Forward propagation and backpropagation

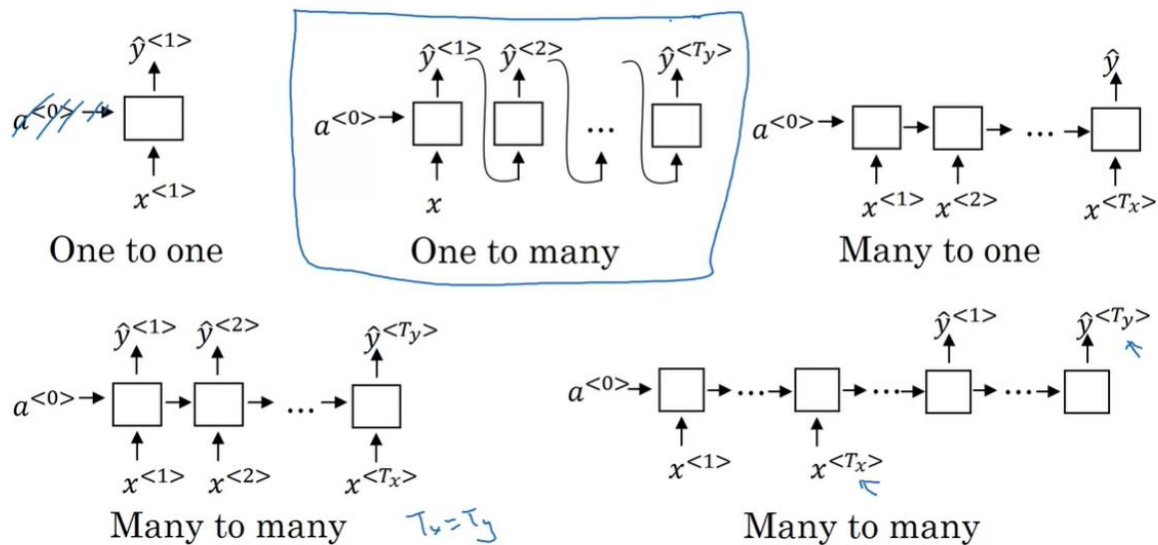


Andrew Ng

As I already wrote, The backward propagation use the future data ( $x^{<t+1>}$ ,  $x^{<t+2>}$ , ...), and actually it needs an additional parameter which is the loss function ( the difference between the real value and the predicted one ), and in this slide it's the Cross-Entropy one , We will calculate the loss for each  $y^{<t>}$   $\mathcal{L}(\hat{y}^{<t>}, y^{<t>})$ , and then we will sum them to get the global loss  $\mathcal{L}(\hat{y}, y)$

## E. RNN types :

### Summary of RNN types



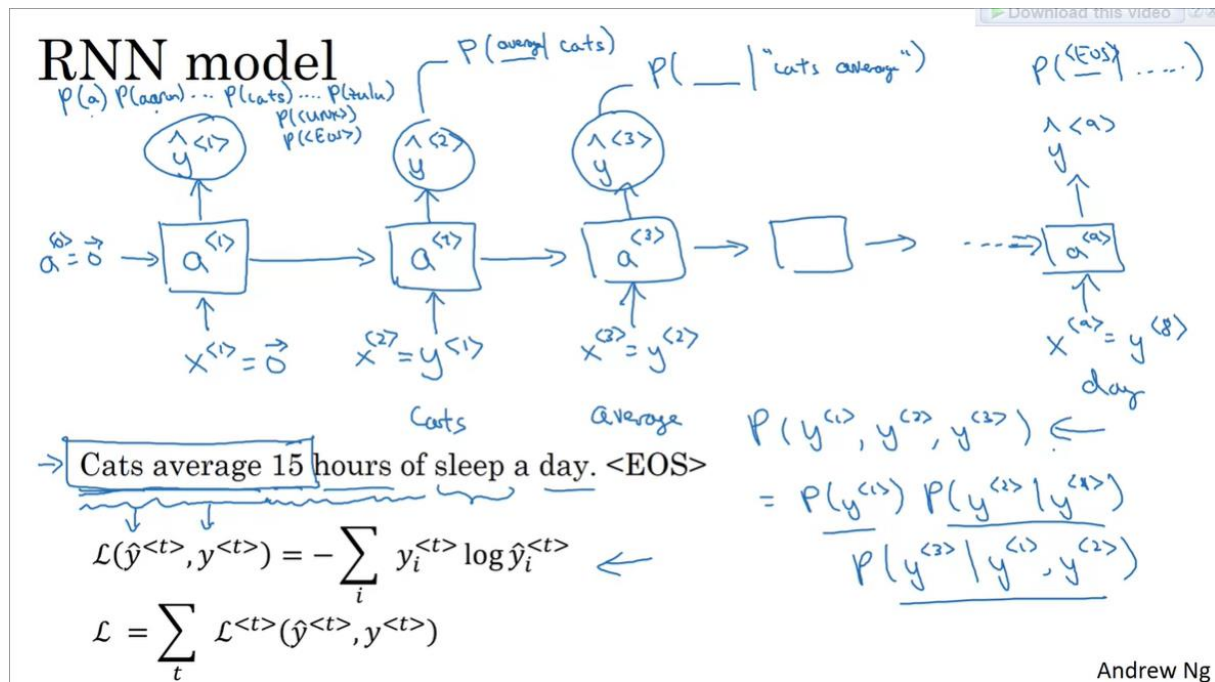
Andrew Ng

- **One to one:** this model is the classic Neural network model (for each input we got a single output)
- **Many to Many (Left Side) :** this is the one we represented previously for Named entity recognition problem , for each  $x^{<t>}$  we got an output  $y^{<t>}$
- **Many to Many (Right Side) :** We will get this model if the length of the output is not equal to the input one ( Example : Machine Translation ) , So we read all the inputs  $x^{<t>}$  one by one ( Encoding step ) before receiving the outputs  $y^{<t>}$  which represents the translated text ( Decoding step )
- **Many to One:** The sentiment analysis takes a text ( sequence of  $x^{<t>}$  ) and the output is a single output which represents the corresponding number of stars to the comment ( between 1 and 5 )
- **One to many:** The music generation takes as an input a number representing the Music to generate genre and the output is a set of  $y^{<t>}$  which represents the generated music notes

## IV. Language models and Sequence generation

In this section, I'm going to explain how the process of generating a sentence with a sense using the RNN (one to many particularly)





- First of all, We have to feed our model by a training set which is in form of Text from newspaper, articles, books, ...etc
- The sentence generation starts by generating the first word. The model is going to look to the training text and using Softmax optimizer; it gives us the word of the highest probability to be in the start of the sentence, which is "Cats" in this case
- For the second word generation we are going to give our model the first generated word "Cats" as an input ( $x^{<2>} = y^{<1>}$ ) and it gives us a word  $y^{<2>}$  with the highest probability to be beside "Cats":  $y^{<2>}$  with  $\text{Max}(P(y^{<2>} / y^{<1>} = \text{"Cats"}) = \text{"average"}$
- For the third word we are going to have the same thing by taking in consideration the first two generated words:  $y^{<3>}$  with  $\text{Max}(P(y^{<3>} / y^{<1>} = \text{"Cats"} \text{ AND } y^{<2>} = \text{"average"}) = \text{"15"}$ , and so on!

## V. Vanishing Gradient problem

### F. Definitions

- **Vanishing gradient definition:** the gradients of the loss gradient-based function approach zero (like Sigmoid), making the network hard to train (learning rate tends to 0)
- **Exploding gradient definition:** it's the opposite of vanishing gradient: Exploding gradients are a problem when large error gradients accumulate and result in very large updates to neural network model weights during training. Gradients are used during training to update the network weights, but when the typically this process works best when these updates are small and controlled.



## G. How RNN are risked to have Vanishing gradient problem?

Let's take as an example of a generated sequence:

- The cat, which ate fish, cheese and milk was full
- The cats, which ate fish, cheese and milk were full

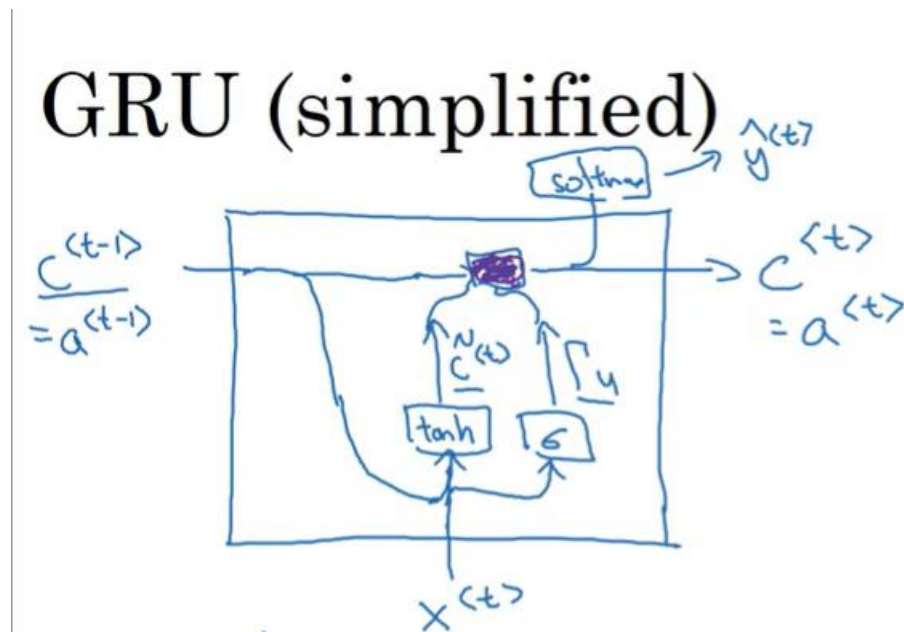
As we see, the  $y_{<2>}$  ( "cat"/"cats") is strongly dependent to  $y_{<9>}$  ( "was"/"were" )

And we will get a wrong sentence if we combine cat with were or cats with was .

The problem appeared well in this particular sentence because there is a big gap between the dependent words: there are not neighbors, and for the RNN: the  $y_{<t>}$  is strongly dependent to its neighbors ( like  $y_{<t-1>}$  ,  $y_{<t-2>}$  ,  $y_{<t+1>}$  ,  $y_{<t+2>}$  ..) but in this case there is a dependence between  $y_{<2>}$  and  $y_{<9>}$  ,

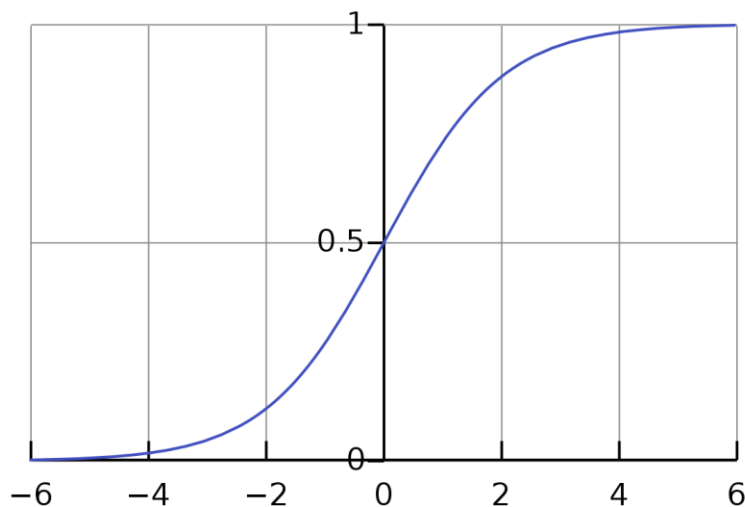
## VI. Global recurrent Unit ( GRU ) is the solution :

### H. GRU Architecture ( simplified version ) :

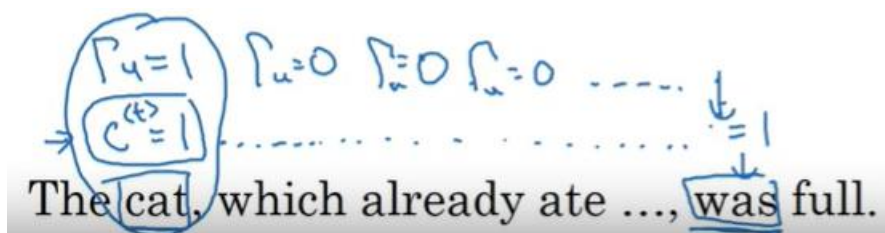


- $C_{<t-1>}$  ,  $C_{<t>}$  : represents the cell memory , it's the parameter that we use to memorize that we have as a subject "cat" and not "cats" ( saved information in C : 1 if plural and 0 if it's singular ) , C could be also a vector instead of a single value in order to store more important information instead of only singular/plural
  - For GRU : the memory cell  $C_{<t>}$  is equal to the activation function  $a_{<t>}$
- $c_{\sim <t>}$  : It's the candidature value for  $c_{<t>}$  if we gonna erase the memory cell  $c_{<t-1>}$

- $\Gamma_u$ : called “Update gate”, it’s value is between 0 and 1 ( due to Sigmoid function ) , actually is either too close to 1 or too close to 0 ( thanks to sigmoid function )



The role of the Update gate is to decide whether we update the current value of the memory cell  $c_{<t>}$  or change it , here is an example while inspecting our sentence :

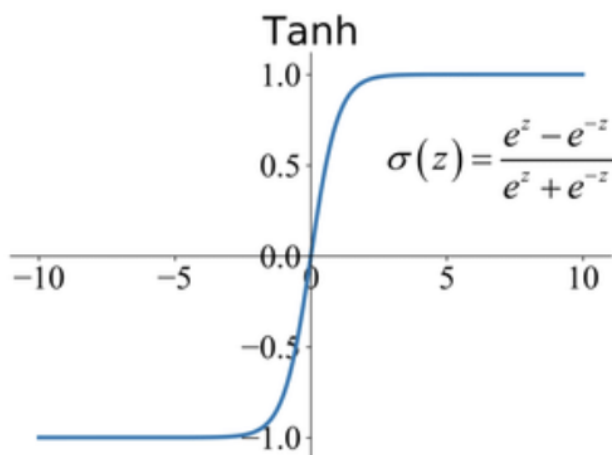


- The  $\Gamma_u$  was 1 when meeting the word “cat” , and she stored value 1 in  $c_{<t>}$  ( 1 for singular ) , and then  $\Gamma_u$  continues of being 0 in order to keep the stored value of  $c_{<2>}$  in  $c_{<3>}$  ,  $c_{<4>}$  ..... until we reach the word “was”

## I. The mathematic formulas :

$$\begin{aligned}
 &C = \text{memory cell} \\
 &\rightarrow \underline{C}^{(t)} = \underline{a}^{(t)} \\
 &\rightarrow \tilde{C}^{(t)} = \tanh(W_c [C^{(t-1)}, x^{(t)}] + b_c) \\
 &\rightarrow \Gamma_u = \sigma(W_u [C^{(t-1)}, x^{(t)}] + b_u) \\
 &\quad \quad \quad \uparrow \text{"update"} \\
 &\left\{ \underline{C}^{(t)} = \underbrace{\Gamma_u}_{=1} * \underbrace{\tilde{C}^{(t)}} + \underbrace{(1-\Gamma_u)}_1 * \underline{C}^{(t-1)} \right.
 \end{aligned}$$

- Quick interpretation about the  $C^{(t)}$  formula:
  - If  $\Gamma_u$  is close to 0 :**
    - $C^{(t)} = C^{(t-1)}$ , and this means that we are saving the old information for the coming iterations
  - If  $\Gamma_u$  is close to 1 :**
    - $C^{(t)} = \tilde{C}^{(t)}$ , and this represents a new information we are going to store ( it uses tanh function which has a values between -1 and 1 )



## J. Full GRU formula :

### Full GRU

$$\tilde{c}^{<t>} = \tanh(W_c[\underbrace{c^{<t-1>} *}_{\text{LSTM}} x^{<t>}] + b_c)$$

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

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

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

The cat, which ate already, was full.

Andrew Ng

It's almost identical to the simplified one except of adding a new parameter which is  $\Gamma_r$

- $\Gamma_r$ : called "relevance gate", it tells how much  $c^{<t-1>}$  is relevant to  $c^{<t>}$ , it's value is between 0 and 1 due to the Sigmoid function
- This new parameter is useful to avoid more and more the vanishing gradient issue

## VII. Long Short Term Memory ( LSTM )

### GRU and LSTM

#### GRU

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

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

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

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

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

#### LSTM

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

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

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

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

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

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

$$\Gamma_o * \tanh c^{<t>}$$

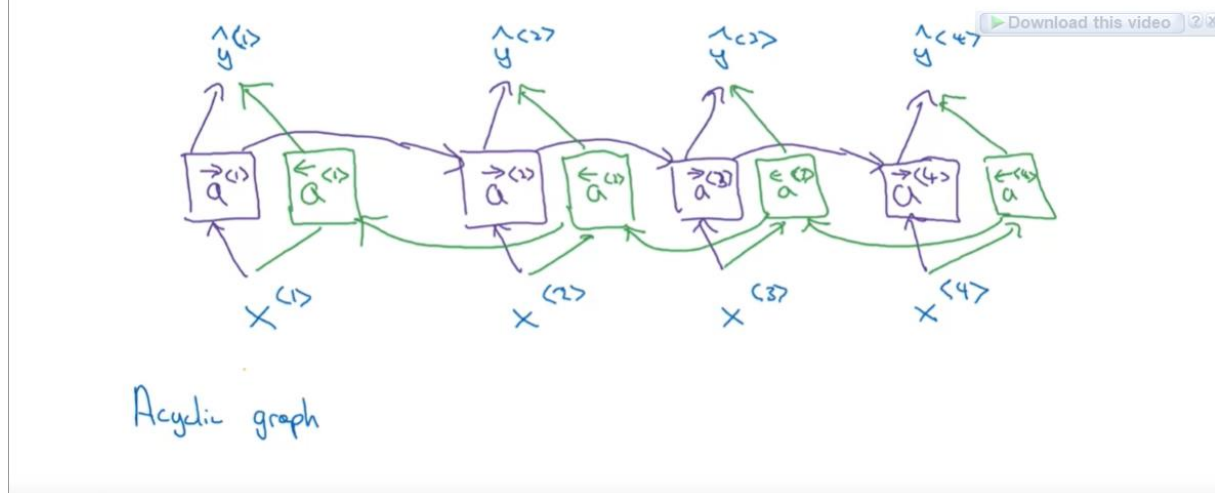
[Hochreiter & Schmidhuber 1997. Long short-term memory] ←

Andrew Ng

- As we see, LSTM and GRU are many common things except of additional changes in LSTM :
  - $a^{<t>}$  is no longer equal to  $c^{<t>}$
  - We have now three gates  $\Gamma$  all of them are sigmoid function ( their values between 0 and 1 )
    - $\Gamma_u$  : "update gate"** : it decides whether we are going to affect our cell memory  $c^{<t>}$  by the new condidtaure value  $c^{<t>}$  or not
    - $\Gamma_f$  : "forget gate"** : it decides whether we are going to keep the stored value in  $c^{<t-1>}$  or not
    - $\Gamma_o$  : "output gate"** : used to calculate  $a^{<t>}$  using  $c^{<t>}$  , it's called output because it decides whate we gonna send as a prediction
  - Thanks to the update and the forget gate we can combine in the actual memory cell  $c^{<t>}$  : the old memory  $c^{<t-1>}$  and an updated info  $c^{<t>}$
  - GRU is simpler and it takes less time in the computation while the LSTM is more powerful but it takes more time to do its computation

## VIII. Bidirectional RNN ( BRNN ) :

### Bidirectional RNN (BRNN)



- Given these two sentences:
  - He said "**Teddy** Roosevelt was a great president"
  - He said "**Teddy** bears are on sale"
    - We cannot decide whether "Teddy" is a named entity or not without inspecting the coming words ( $x_{t+1}$ ,  $x_{t+2}$ , ....)
- To resolve this kind of issues we need to combine the forward propagation (reading the precedent text : From left to right) with a backward propagation (reading the future text : From right to left)
- Beside our four activation functions  $a_{<1>-}$ , ...,  $a_{<4>-}$ , we are going to need four more activation functions for the backward propagation, we notate them by  $a_{<1>+}$ , ...,  $a_{<4>+}$
- The chronological order of the computation of the activation functions in BRNN is :  $a_{<1>-}$ , ...,  $a_{<4>-}$ ,  $a_{<4>+}$ , ...,  $a_{<1>+}$  : we compute the forward activation function from right to left then the backward propagation functions from left to right
- After this computation we can calculate the  $\hat{y}_{<t>}$  the predicted output using  $a_{<t>-}$  and  $a_{<t>+}$
- This technique can be applied to the classical RNN, GRU and LSTM
- For NLP problems: the most common solution is LSTN architecture with BRNN technique

## K. Disadvantages of RNN:

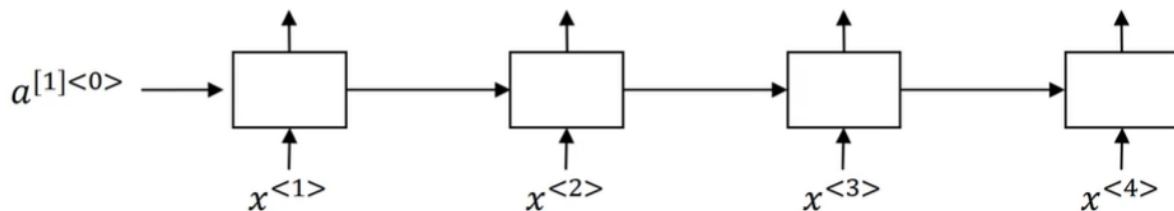
Since we need the whole sequence to do our prediction (Past and future) : we cannot do this kind of architecture to handle a real-time problems ( because there is no end of the sequence ) , and for speech recognition system : We cannot do an instant translation : But we must wait till the user ends recording its vocal , and then we can start our speech recognition process



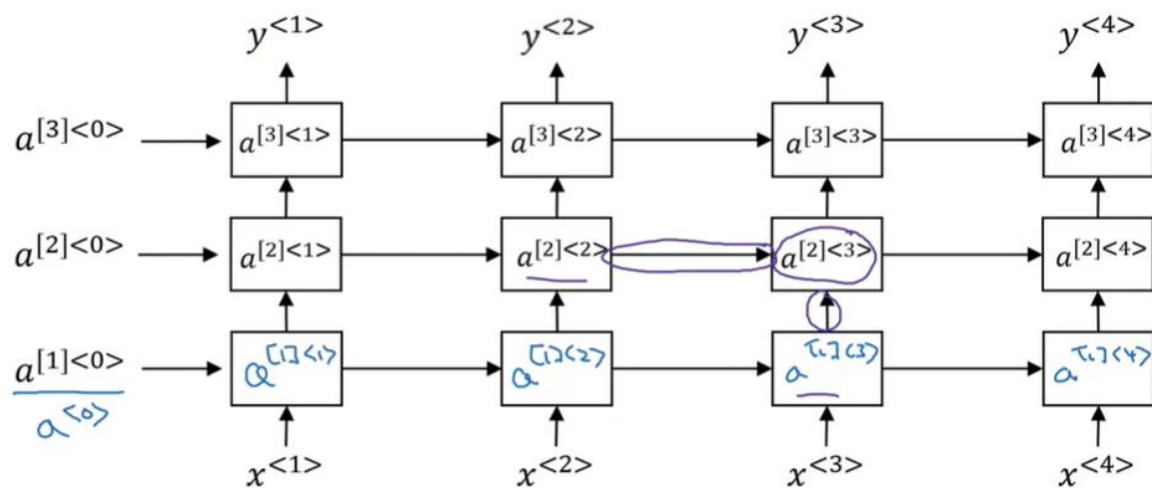
## IX. Deep RNN

It consists of stacking up more than a one layer of RNN ( as we already saw so far )

- RNN/LSTM/GRU with one layer architecture:



- DRNN architecture for classical RNN/LSTM/GRU :

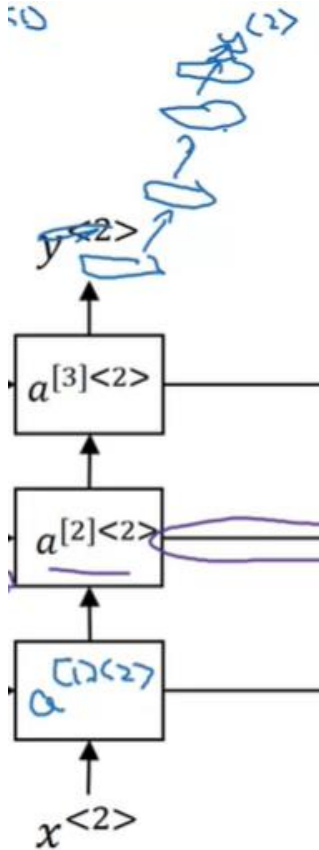


- The output of a layer will be fed to the next layer
- Each layer has its own activation functions the layer one have  $a[1]<0>$  , ...,  $a[1]<4>$  and so on
- The activation functions of a layer will be calculated using activation function from the bottom layer , for  $a[2]<3>$  for example :  $a[1]<3>$  is Involved beside  $a[2]<2>$  , and here is its formula :

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

- The predicted output  $y<t>$  is the output of the last layer

- We can instead of using directly the output of the last layer : We use it as an input to our Deep neural network ( which is not recurrent ) like it showed in this image for  $y^{<2>}$  :



- Generally , a three layers are enough to implement a deep RNN ( and not more ; because RNN takes too much time to do its computation )

## X. What I've learned from the assignments :

- Usually we do the training by batches , and that means while fitting our model we pass more than one training example ( one example = 1 sequence like sentence ) simultaneously to gain time
  - The size of the mini batches are  $(n_x, m, T_x)$  :
    - **$N_x$**  : is the vocabulary size ( one hot vector )
    - **$m$**  : is the size of the batch ( number of trainings per batch )
    - **$T_x$**  : is the size of the longest sequence
  - $x^{<t>}$  in this case is equal to  $X[:, :, t]$
  - The dimension of the prediction is  $y^{\wedge}$  is  $(n_y, m, T_y)$  because we are going to predict a one-hot vector for each timestep in the sequence
- the activation  $a^{<t>}$  is called "hidden state"

- `np.dot()` or `@` are used to do a real mathematical matrix multiplication while `*` is used to do a scalar multiplication ( `a[i][j]*b[i][j]` )
- For NLP process , In the character-level sequence generation ( we generate character per character and not word per word ) we should replace each character by a unique index to replace it , because the RNN accepts only numerical values , and for each training example ( word) we end `'\n'` as a significance about the end of the generated word . the `'\n'` should be represented too in our dictionary `char->index` so when our model generates it , we will know it's the end of the generated sequence
- While Vanishing gradient is a common problem in the RNNs , we must take ion consideration also the Exploding gradient issue by using the gradient clipping technique
  - Exploding gradients make the training process more difficult, because the updates may be so large that they "overshoot" the optimal values during back propagation.
  - The gradient is represented by the parameters : `"dWaa"`, `"dWax"`, `"dWya"`, `"db"`, `"dbx"`
  - We define a max and a min value for the gradient array values `[-max , max ]` , if a value is inferior than `-max` we replace it by `-max` and if a value surpass `max` we replace it by `max` , and by this process we assure a controlled values under the gradient
  - We use the function `np.clip()` to achieve that
- For the sampling ( a sequence generation process after the training ) : We said that we will take `y<t>` with the highest probability , the occurred problem is we will fell always into the same sequence ( if `y<0>` with `Max(P(y<0>))` is "cats" , then all our generated sequences will start with "cats" and we will get the same `y<1>`,`y<2>` , ...,`y<ty>` always and this is not really interesting )
  - To make the results more interesting, use `np.random.choice` to select a next letter that is likely, but not always the same.
  - his means that you will pick the index (`idx`) according to the distribution: if the output of the softmax function is `y= [ 0.1 , 0, 0.7 , 0.2]` then `P(index=0)=0.1,P(index=1)=0.0,P(index=2)=0.7,P(index=3)=0.2` .
  - We use the function `np.random.choice()` to achieve that

## XI. Word representation

## L. One-Hot vector :

$V = [a, aaron, \dots, zulu, <UNK>]$

$|V| = 10,000$

1-hot representation

Man (5391)	Woman (9853)	King (4914)	Queen (7157)	Apple (456)	Orange (6257)
$\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \\ 0 \end{bmatrix}$	$\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \end{bmatrix}$	$\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \\ 0 \end{bmatrix}$	$\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \end{bmatrix}$	$\begin{bmatrix} 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$	$\begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \end{bmatrix}$

Handwritten notes: Blue arrows point to the vectors for 'Man' and 'Woman'. Below the 'Man' vector is a handwritten circle containing '5391'. Below the 'Woman' vector is a handwritten circle containing '9853'.

- This is the kind of the representation I used to see in the first week , where each word is represented by a vector having a length equal to the vocabulary size , every vector element is a zero except for the index of the word
  - o  $O_{5791}$  is a notation to represent the vector of the word “man” which is indexed by 5391 in our vocabulary

### ➤ Disadvantage of this representation :

- There isn't any information inside this representation that tells the nature of the relation between each pair of words which can be so important and useful for the NLP tasks .
  - o For Sequence generation for example :
    - If we have in the training set “I want a glass of **orange juice**”:
      - Our sequence generator will not be able to generate: “I want a glass of **apple juice** “
      - Our model knows that there is a relation of neighborhood between “orange” and “juice” but he doesn't know that “orange” and “apple” are similar to each other by being “Fruit” but for example “orange” and “queen” are not similar

## M. Word embedding : Featurized representaion :

	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	<u>0.93</u>	<u>0.95</u>	-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
...	...	...				
size						
cost						
alix verb						

Handwritten notes on the left: ↑ Gender, 300 Royal, Age, Food, size, cost, alix verb. Below the Man and Woman columns are handwritten labels  $e_{5391}$  and  $e_{9853}$  respectively.

Andrew Ng

- In this representation , each word will be represented by a set of features with values between -1 and 1 ( for gender for example : -1 means male and 1 means female ) , so know the length of the vector depends of number of features to take in consideration instead of our vocabulary size
- Example of features: gender, age , food , size , cost , ...
- This representation is so important to detect the similarities between the words
  - o We can see that King and Queen are both royal words for example
  - o Woman and queen are both female gender words
- And by looking globally : we can see the similarities in global view not by just a single feature
  - o Apple and orange are **globally** similar words ( they have identical values in their vectors ) so now , our model can generate for us apple juice and orange juice :

Queen (7157)	Apple (456)	Orange (6257)
0.97	0.00	0.01
<u>0.95</u>	-0.01	0.00
0.69	0.03	-0.02
0.01	0.95	0.97

Handwritten blue arrows point from the top of the Apple and Orange columns to the word "juice" in the sentences below. A large blue bracket on the right groups the four rows of the table.

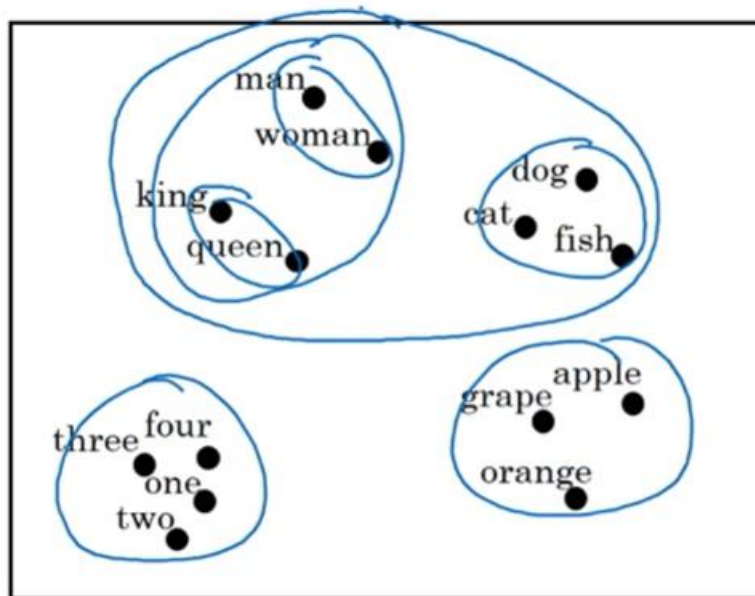
I want a glass of orange juice.

I want a glass of apple juice.

### ➤ Visualizing Embedding words :

We cannot represent directly the word embeddings representation in a plan because it's dimension dependent to the number of features ( which can be : 100 ; 200 , 300 , .. )

We will use an algorithm called “T-SNE” to have a 2D presentation of the embedding words.



t-SNE

## XII. Transfer learning and word embeddings:

- Often, our training set (100K words for example ) is too short to handle all the language vocabulary which is necessary to detect the similarity between the words for the later processing , example “dorian” is a fruit localized in Singapore , It's more likely that we will not find this word in our training set
- So before doing word embeddings in our training set, We must start learning word embeddings from a very large text corpus (1B-100B words ) before doing it to our training set
- As an alternative solution, we can use a pre-trained model and then transfer the learning into our tiny training set ( as I did in Nazim Workshop )
- And later : we can optionally finetune our training set with a new data and redo the training



### XIII. An interesting similarity between word embeddings and face encoding (face recognition):

If we have in our training set , a images about the face of the same person , the encoded data ( the array ) will contain almost same information ( same values ) due to representing the same face or a similar one , Like the presentation of the similar words in word embedding

### XIV. Word embeddings properties:

“A “man” to “woman” as “king” is to “x” “?

The word embeddings presentation has the ability to answer this question and he will tell us it's “queen” !, this property is called **“word analogies”** .

#### N. How's that !

Well , let's do quick reminder about how the world are represented in the word embeddings :

#### Analogy

	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	0.70	0.69	0.03	-0.02
Food	0.09	0.01	0.02	0.01	0.95	0.97

$e_{\text{man}} - e_{\text{woman}} \approx e_{\text{king}} - e_{\text{?}}$ 
 $e_{\text{man}} - e_{\text{woman}} \approx \begin{bmatrix} -2 \\ 0 \\ 0 \\ 0 \end{bmatrix}$ 
 $e_{\text{king}} - e_{\text{queen}} \approx \begin{bmatrix} -2 \\ 0 \\ 0 \\ 0 \end{bmatrix}$

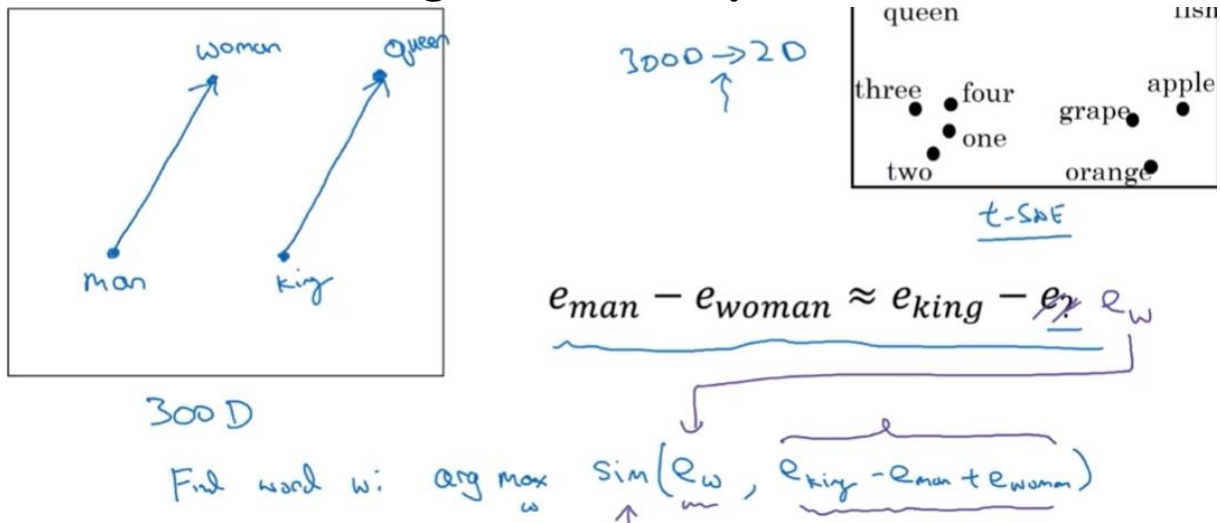
Man  $\rightarrow$  Woman  $\approx$  King  $\rightarrow$  ? Queen  
 $e_{\text{man}} - e_{\text{woman}} \approx e_{\text{king}} - e_{\text{queen}}$

[Mikolov et. al., 2013, Linguistic regularities in continuous space word representations] Andrew Ng

As the slide illustrates , We calculate the vector representing the gap between “man” and “woman”  $e_{\text{man}} - e_{\text{woman}}$  .

And we calculate the vector between  $e_{\text{queen}}$  with all the words in the vocabulary in order to get the word “x” which has a distance with “queen” equals the distance between “man” and “woman”

## O. Calculating the similarity :



In other way , we will calculate the similarity between the vector representing  $e_{king} - e_{man} + e_{woman}$  with all the words until we got the most similar vector

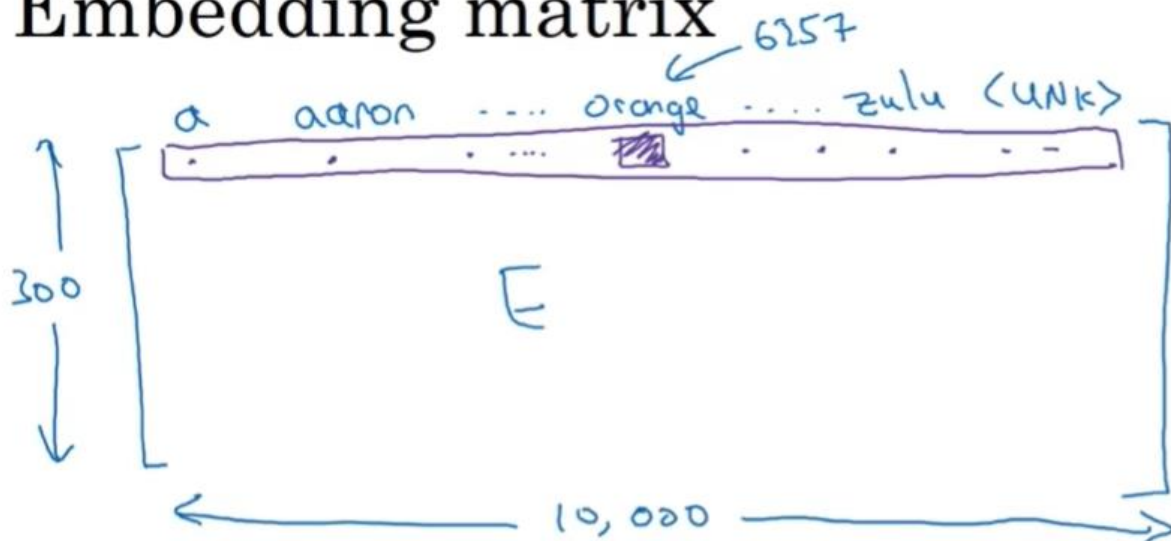
- There are many similarity functions to use for example: cosin similarity

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

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

## XV. Embedding Matrix: the mathematic representation of our vocabulary by word embeddings

### Embedding matrix

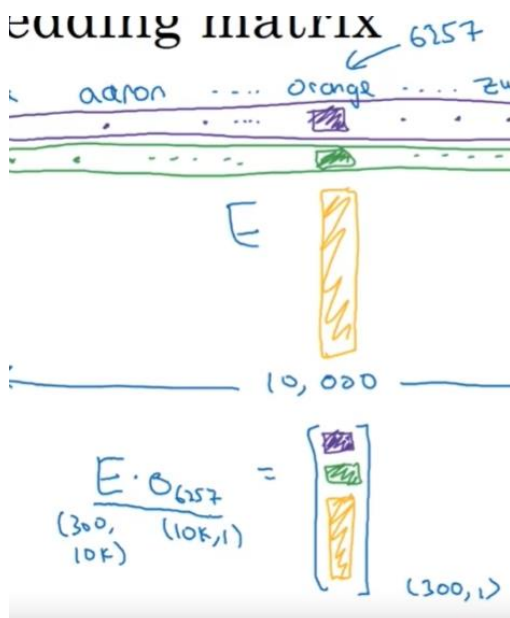


- The Embedding matrix "E" : has number of features\*vocabulary size as a dimension.
- Each word in our vocabulary will be represented by a vector of features : for each feature corresponds a value between -1 and 1

## P. How to get mathematically a features vector of a specific word :

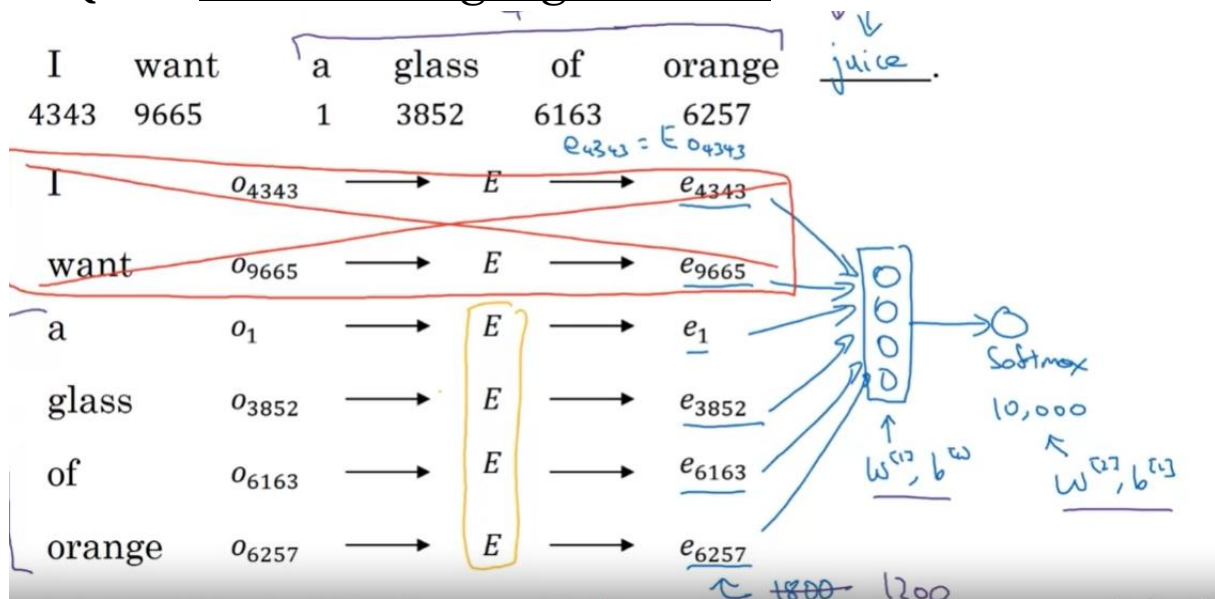
We multiply the Embedding matrix ( 300x10000) by the corresponding one-hot vector of the specific word (10000x1) to get its word embedding representation vector (300x1)

So  $e_{6257} = E \times O_{6257}$ :



## XVI. Learning Word embeddings

### Q. Natural Language Model :



In order to predict a coming word for a previous sequence of words , we should give to our model , **the context** which is for example the four previous words in order to fix the number of inputs which are their features vectors extracted from the features Matrix E , We apply the SoftMax function and the output is the word which has a biggest probability from SoftMax

➤ Another possible context :

4 words on left & right

a glass of orange ? to go along with

Last 1 word

orange ?

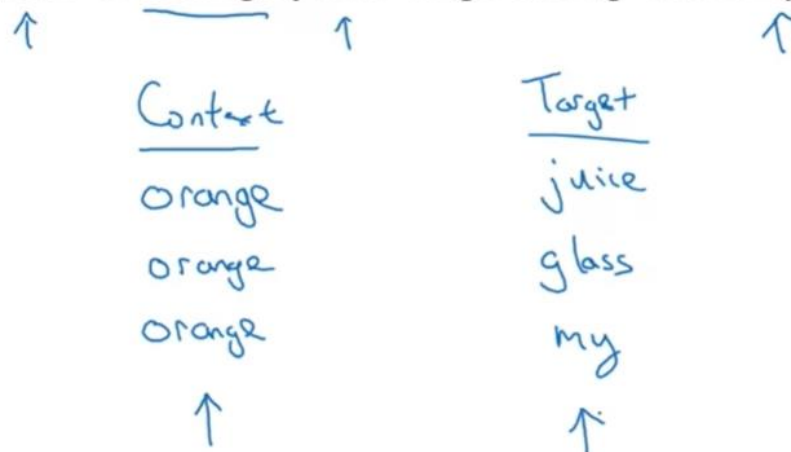
Nearby 1 word

glass ?

- 4 words on left and right : to predict a word in the middle
- Last 1 word: in order to reduce the calculations amount
- Nearby 1 word: the nearby word isn't necessary a direct previous word , but a word which is close to the target ( we often eliminate the no-significant words like "of" , "the" , "a" )

R. Skip-grams algorithm with Word2Vec :

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



In this algorithm, rather than having the context be always the last four words or the last end words immediately before the target word, we randomly pick a word to be the **context** word ( it's embedding vector  $e_x$  ); and then we pick a random word from the close range to be the **target** ( its embedding vector  $e_y$  ).

For example: and like the image showed from the given sentence we randomly took “orange” as the context and “juice” as its target and so on.

And by doing that: we are having a prepared dataset to do a supervised learning, for each data (context) : we have its label ( target ) . so, we can solve the problem “given a context: predict me the target”

➤ How to sample the context “c”:

We will try to pick a rare word instead of a word which appears frequently , So we gona pick “orange” , “glass” , ... instead of “the” , “a” , “of” .....

➤ The probability function used in softmax for word embeddings :

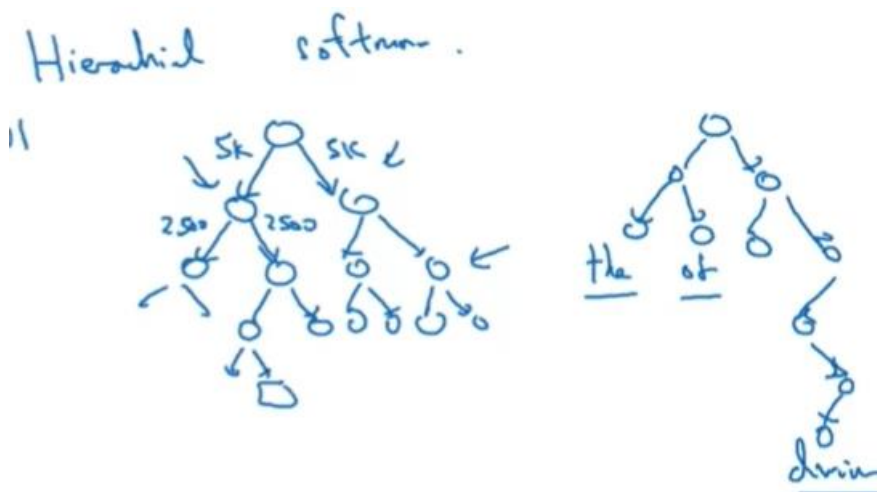
$$\underline{p(t|c)} = \frac{e^{\theta_t^T e_c}}{\sum_{j=1}^{10,000} e^{\theta_j^T e_c}}$$

- “t” represents the target and “c” is the context so we calculate “the probability of t given c as a context”
- $\theta^t$  is a parameter associated to the softmax unit

➤ the disadvantages of this probability function  $p(t/c)$  :

the problem is in the denominator where its calculation is expensive in term of computation , and we ran replace it by a **Hierarchal SoftMax**

➤ Hierarchal SoftMax :



During the calculation of the denominator, we aren't going to pass by all the words in our vocabulary but it tells you is the target word in the first 5,000 words in the

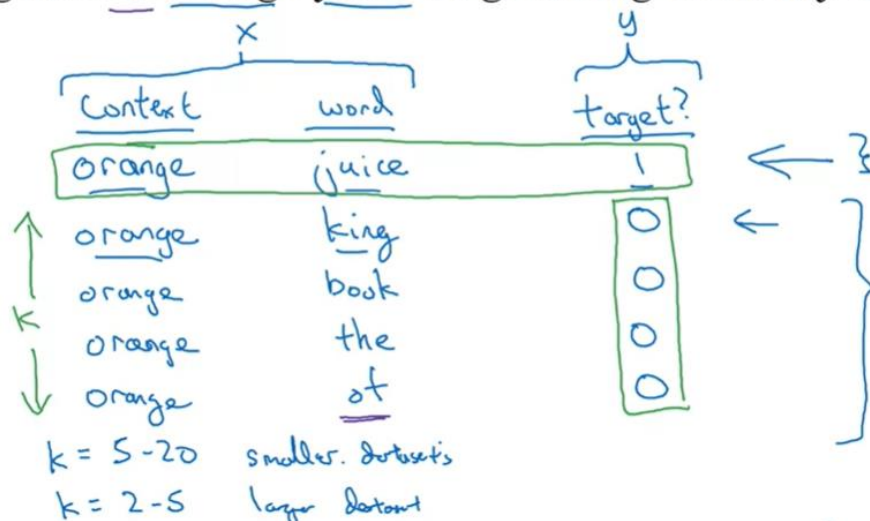


vocabulary? Or is in the second 5,000 words in the vocabulary? And so on until we got our target ( the computation complexity is  $O(\log(x))$  in place of  $O(x)$  ), but it's still a high cost computation

## S. Negative sampling: an optimized technique to do the word embeddings :

instead of dealing with a multi-classification problem and being obliged to do SoftMax calculations , we are going to transform our learning problem to a binary-classification problem :

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



- We are going to take a context and target from our corpus text as we did in skip-grams algorithm: we pick a random word from the text and we consider it as the "context" word in our target , and the "target" word ( the label ) is taken from the same text in the neighborhood of the context word , and let's say it's "orange" and "juice" from the sentence : "I want a glass of orange juice to go along with my cereal"
- The pair ( "orange" , "juice" ) will be a row in our training data with "1" as a label meaning that the pair ( "orange" , "juice" ) is defining really a pair of ( context , target )
- After that , we are going to do the negative sampling by picking  $K$  ( = 4 In this example ) words from our vocabulary and considering them as a false pair ( context , target ) with the word "orange" as a context and the picked word as the target by sitting the label "0"
  - o  $K$  is equal to  $[5-20]$  for a small dataset and  $[2-5]$  for a large dataset
- We repeat the same step with different words extracted from our training text corpus

### ➤ Sigmoid instead of Softmax :

Since the label is now either 1 or 0 , we are now into a binary-classification problem of this generated dataset and we are going to calculate the probability of the label being 1 giving the pair (t , c ) as a (target , context )

$$P(y=1 | c, t) = \sigma(\Theta_t^T e_c)$$

- $\Theta_t^T$  is the parameter associated to the target t , and  $e_c$  is the feature vector of the context c

And by doing that we are having 10000 binary classification problem instead of a 10000 SoftMax calculations , this 10000 binary classifications are quite cheap to compute , and for every word in the 10K , our learning iteration contains K+1 outputs , K false sampling and a single correct one

### ➤ How do we select the K false samples for each word?

- We can just pick randomly a word from our vocabulary with a uniform probability distribution ( every word has 1/10K to be appeared ) but this is a non-representative of the language vocabulary , because the words haven't the same probability to be appeared in the text
- Or we can associate to each word a probability normalized ( sum of probability = 1 ) equals to its empirical frequency of appearing In the text , But this is not really beneficial because we will get often words like "the" , "of" , "on" .....
- The best formula according to the mathematicians is to use the empirical frequency power  $3/4$ :

$$P(w_i) = \frac{f(w_i)^{3/4}}{\sum_{j=1}^{10,000} f(w_j)^{3/4}}$$

- $f(w_i)$  represents the number of apparitions of the word " $w_i$ " in our learning text

## T.GloVe ( Global vectors ) : an algorithm which takes in consideration the global statistics and not only the local ones :

For False Sampling , and Word2Vec we saw that these two algorithms concentrates about the neighborhood between the context and the target for each text corpus , one by one .

In Glove we are going to add an additional parameter named  $X_{ij}$  which represents the co-occurrence time of “j” appears in the context of “i” , and  $X_{ij}=X_{ji}$  .

And we will gonna have 2D matrix like this :

	the	cat	sat	on	mat
the	0	1	0	1	1
cat	1	0	1	0	0
sat	0	1	0	1	0
on	1	0	1	0	0
mat	1	0	0	0	0

And the goal is minimizing:

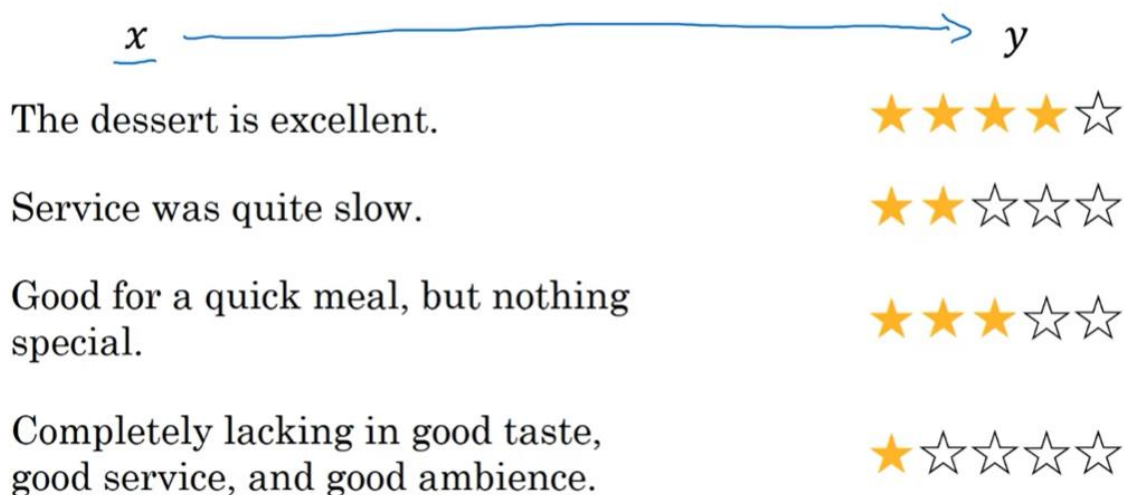
$$\text{minimize } \sum_{i=1}^{10,000} \sum_{j=1}^{10,000} f(X_{ij}) (\theta_i^T e_j + b_i + b'_j - \log X_{ij})^2$$

- The role of  $f(X_{ij})$  :
  - If  $X_{ij}=0$  then  $f(X_{ij}) = 0$
  - It will reduce the very large value of the formula (which concerns the words which appears a lot like “a” , “the’ , “of”
  - It will increase slightly the very little value of the formula (which concerns the words which appears rarely like “Dorian” the fruit )

I cannot interpret the formula because I didn't understand it xD

## XVII. Sentiment Analysis: an Application using the word embedding techniques

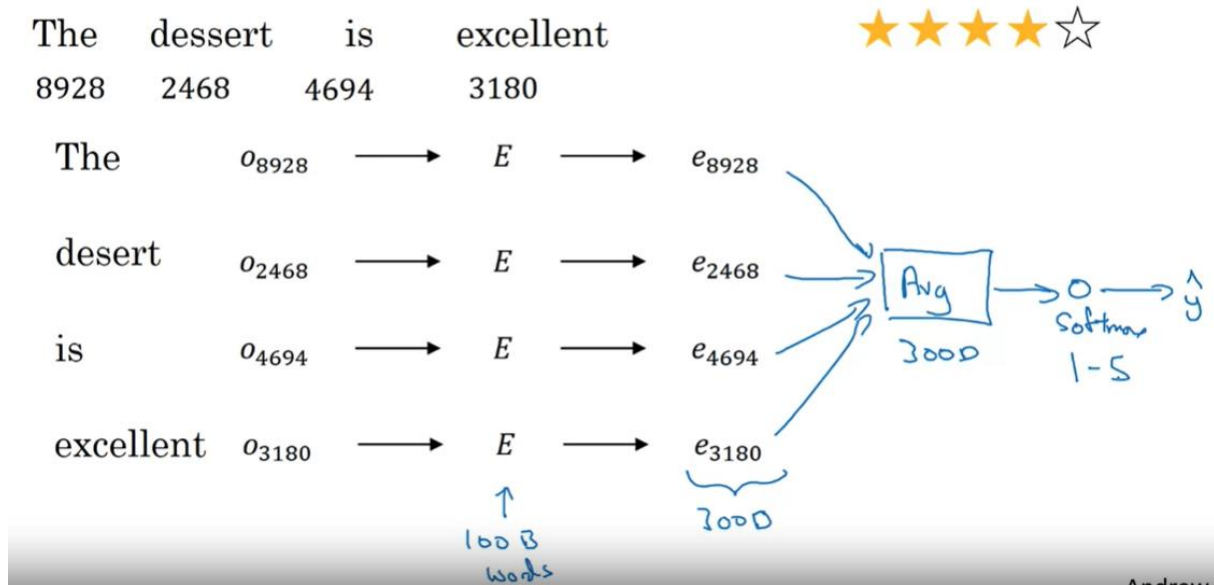
### U.The presentation of the problem :



Given X: a sequence of words : predict the number of stars “y” it corresponds ( from 1 to 5 )

- For sentiment analysis tasks: it's hard to find a large labeled dataset to do your training on , but thanks to the Embedding Matrix ( E ) that we got from a pre-trained model or even by training our model in a very large text corpus : it will cover the problem of having a little dataset for training about our sentiment classifications

## V. Algorithm 1 : Simple sentiment classification model without the RNN architecture :



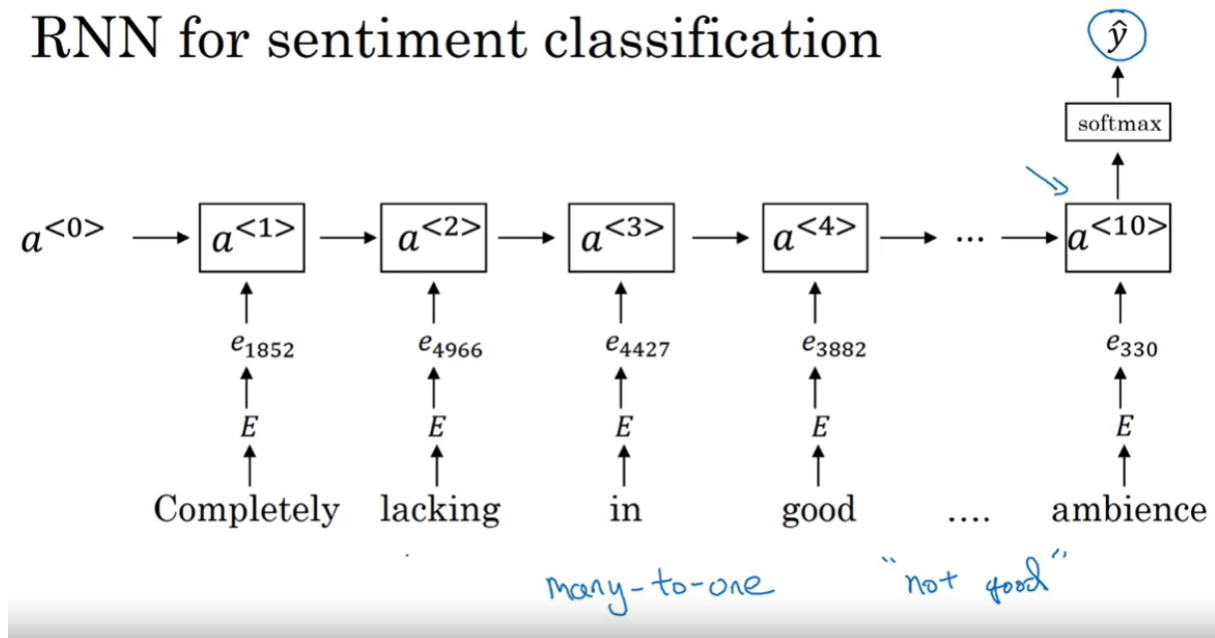
- For each word in our sentence , we will extract it's word embedding representation using the  $E$  ( which has over 100B words ) , and then we sum the word's vectors , we take their average and pass it to softmax unit to do a multi-classification problem ( 1 Star , 2 Star , ...or 5 Stars )

### ➤ Disadvantages of This algorithm:

- It doesn't take In consideration the position of the word inside the sentence or its neighborhood which would make our model go into a false predictions for sentences like "Completely lacking in good taste, good service and good ambience" , because "good" appears a lot but In a negative sense ( lack of goodness ) that our algorithm will not detect due to doing only the average of each word representation without taking in consideration their context

## W. Algorithm 2 : RNN for sentiment classifications using the word embeddings presentation

### RNN for sentiment classification



- It's exactly the "Many-To-One" RNN model that we talked previously about
- The only ( big) difference is that we are going to pass in the input layer the embedding word representation  $e_w$  instead of the one-hot vector  $O_w$
- And now, our model can know that "not good" relates to a bad rating

#### ➤ How Word embedding can make our model stronger than the same if we use one-hot vector instead

Since our training set is little and cannot handle all the vocabulary in the language, We can for example have in our model good words like "good", "nice", "delicious" and bad ones like "horrible", "bad", "disgusting", but a word like "magnificent" didn't appear in our labeled training set . So If we try to do sentiment classification of a sentence containing the word "magnificent" : our model will just consider it as a <unk> token for unknown word . But thanks to the word embedding representation in the big embedding matrix  $E$  which contains 100B words, the model will know that "magnificent" is a similar word to "good", "nice" and the other positive words

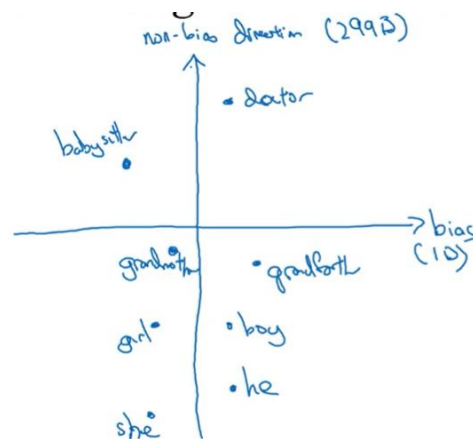


## XVIII. Debiasing word embeddings:

“A man is to Doctor like woman is to Nurse”,

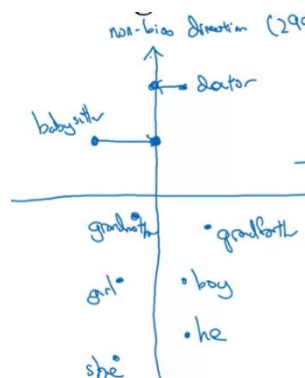
“Babysitting is a work for the grandmother more than to the grandfather”

We can say here that the word embeddings can reflect gender , ethnicity , gender , age , sexual orientation and other biases caused by the text used to do the training



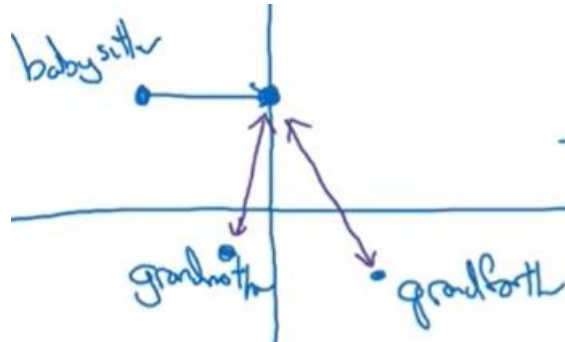
In our society, we believe that there isn't any distinguish between the people to get opportunities or judging them by their race , their gender or their age ...

So neutral words like “Doctor” and “babysitter” which doesn't represent any specific gender in their definitions , we should biases them to put them in the middle , so our model will not take any decision base on the sex , the race or the ethnicity generally of the human



- Of course, we cannot do that to all the words : words like “grandfather” and “grandmother” or “she” and “he” represents the gender in their definitions so we cannot biases them to the middle

- One last step to do is "equalizing **the pairs**", this means that we will make the same distance between the words representing the genders ( like "he" , "she" , "boy" , "girl" ) with the axe which represents the gender like so :



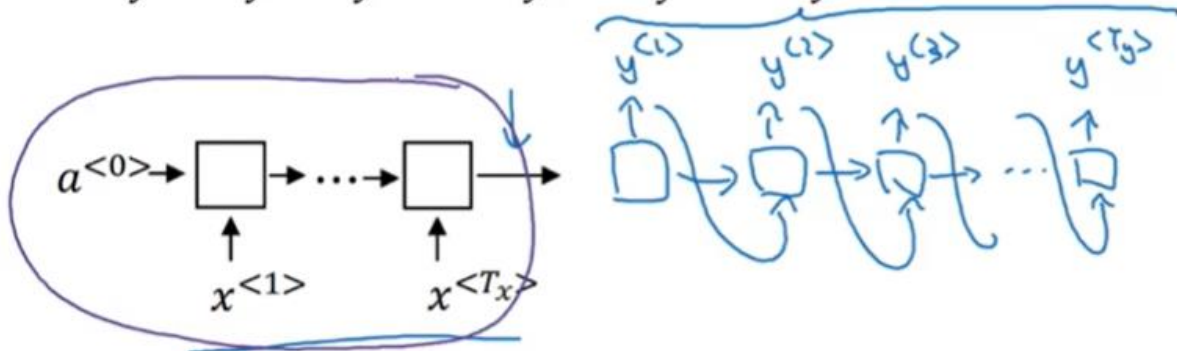
We see that even removing the bias of the word "babysitter", "grandmother" still closer to it more than the "grandfather" does . so "grandfather" and "grandmother" words should be symmetric the axe who represents the gender

## XIX. Sequence to sequence Architecture (many to many RNN) :

$x^{<1>} \quad x^{<2>} \quad x^{<3>} \quad x^{<4>} \quad x^{<5>}$   
 Jane visite l'Afrique en septembre

→ Jane is visiting Africa in September.

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

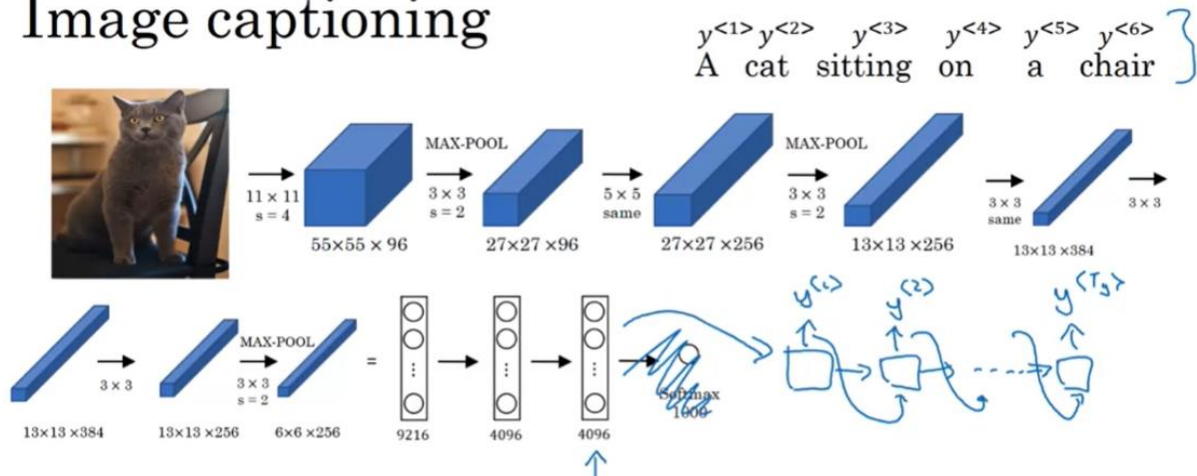


- As we already saw, the many to many expects a sequence of  $T_x$  size and the output is sequence of  $T_y$  size , where  $X$  represents the encoding part and  $y$  the decoding part
- We pass the whole input sequence (  $x^{<1>} , \dots, x^{<T_x>}$  ) before starting to calculating the sequence representing the output  $Y$

- The machine translation ( the image above ) is a good illustration of sequence to sequence model

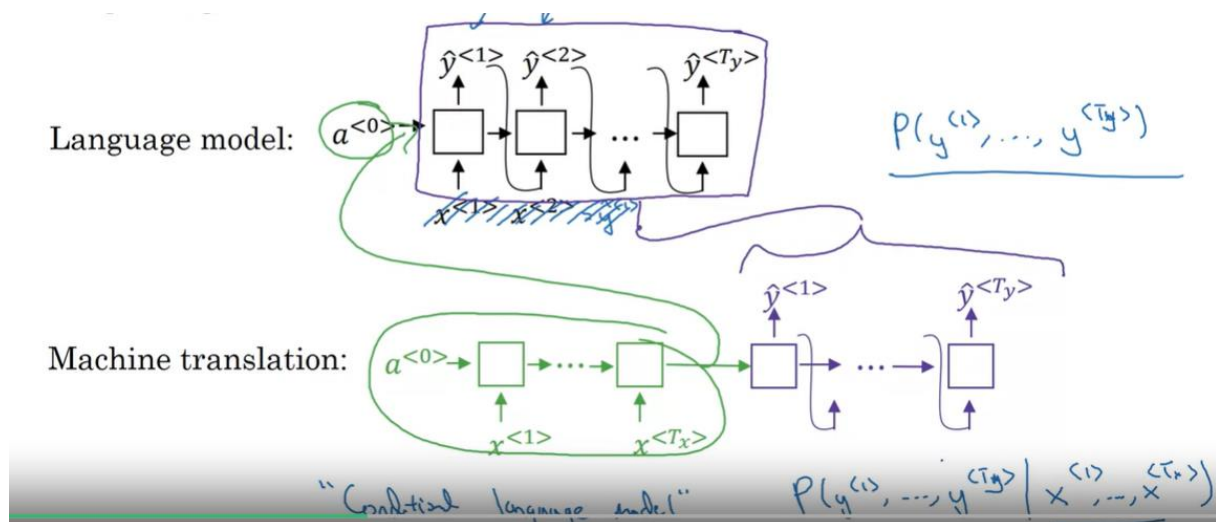
## X. Image captioning is also an interesting illustration:

### Image captioning



- The task here is “give me a suitable caption about the image” which is “a cat sitting on a chair” in this example
- Instead of doing a softmax classifier to predict what’s inside the pic as we used to do in computer visions, we inject the output vector from the last layer (the 4096 dimension vector ) to the decoding part of our RNN model .

## XX. Machine translation as building a conditional language model:



For one-to-many architecture like language modeling : the first hidden state  $a^{<0>}$  is just a zeros-vector , but this is different for sequence to sequence architectures where instead of having a zero-vector  $a^{<0>}$  we are having a set of  $x^{<t>}$  that's gonna update the  $a^{<0>}$  before getting passed to the decoding part of our RNN model . So we can consider the sequence  $X$  as a condition to consider where generating the output  $Y$  ( which is a zero vector for one-to-many architectures like language modeling )

- So we are going to generate the most suitable set of  $y^{<1>}, \dots, y^{<t>}$  given  $x^{<1>}, \dots, x^{<T_x>}$  as an input by maximizing the probability of  $P(y^{<1>}, \dots, y^{<T_y>} | x^{<1>}, \dots, x^{<T_x>})$

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

## Y. Greedy search as a potential solution :

- The greedy search algorithms consists of : starting to generate  $y^{<1>}$  by finding the best one which responds to  $\text{Max}(P(y^{<1>} | X))$  and then going to generate  $y^{<2>}$  by finding the best one which responds to  $\text{Max}(P(y^{<1>}, y^{<2>} | X))$  and so on

### ➤ Greedy search isn't a good solution:

- Suppose we wanna translate the sentence "Jean visite l'Afrique en Septembre" to English:

↓ ↓  
Jane is visiting Africa in September.

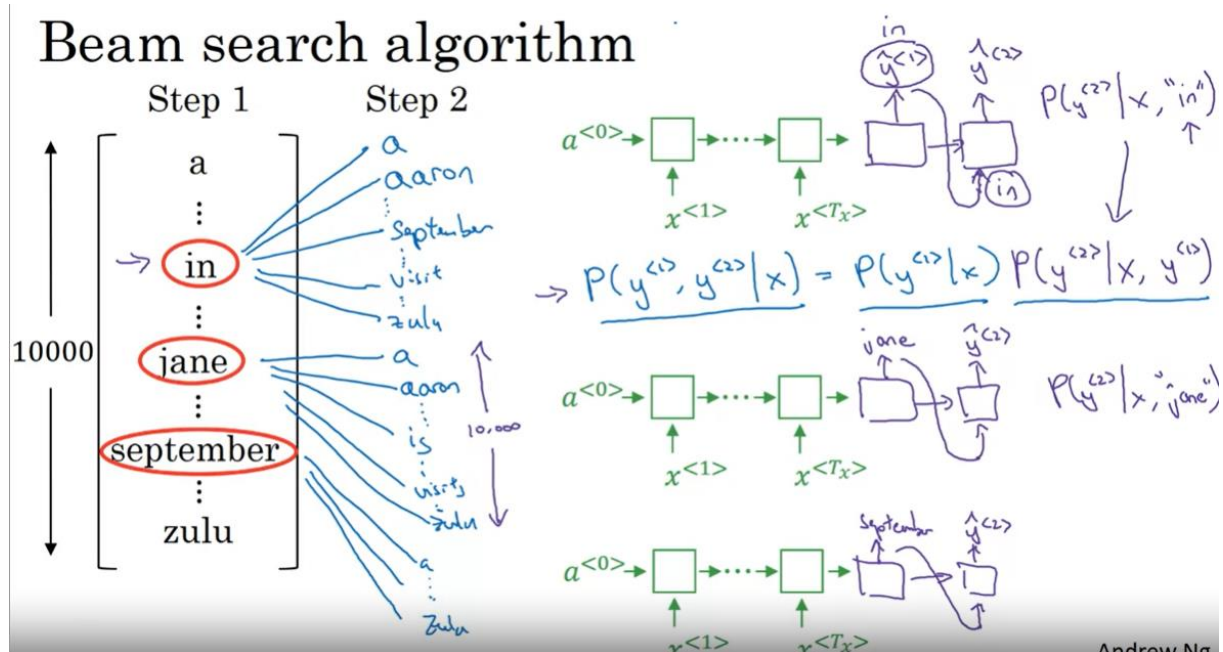
↓ ↓ ↓  
Jane is going to be visiting Africa in September.

$P(\text{Jane is going} | x) > P(\text{Jane is visiting} | x)$

- The greedy search will favorize “Jane is going to be visiting Africa in September” , although “Jane is visiting Africa in September” is more suitable and précised and this is due to having  $P(\text{“Jane is going”} | X) > P(\text{“Jane is visiting”} | X)$  because “going” is a very popular word in English but it took us to wrong path

## Z. Beam search is the ideal search algorithm for sequence-to-sequence models :

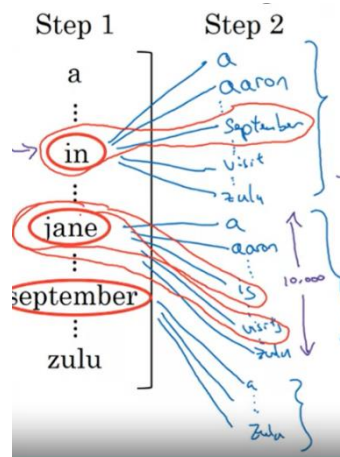
### Beam search algorithm



- Beam search is a heuristic algorithm , it means that he will not give assure us the best solution , but it guarantees us a good enough solution
- For the step 1: instead of picking a single word  $y^{(1)}$  that maximizes  $P(y^{(1)} | X)$  , we are gonna pick  $B=3$  words that have the most big probabilities ( If  $B=1$  then

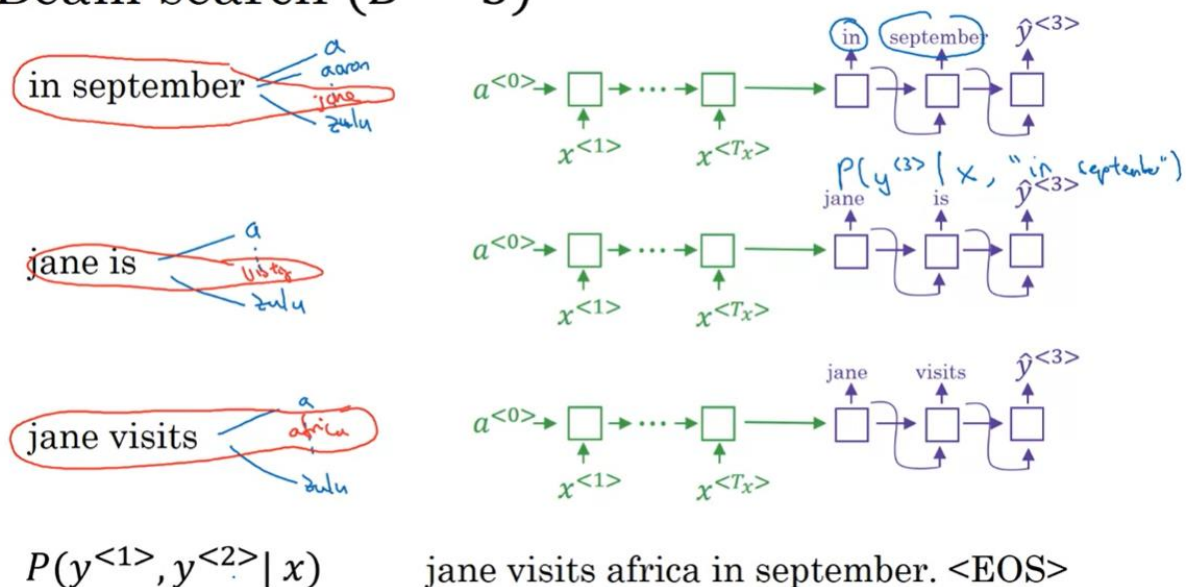
we are doing the greedy search approach ) , we found that the max 3 probabilities goes to "In", "Jane", "September"

- For the step 2: we instantiate 3 copies of our RNN model , the first one will be fed by  $y_{<1>} = \text{"In"}$  , the second one by  $y_{<1>} = \text{"Jane"}$ , and the third one by  $y_{<1>} = \text{"September"}$  . And then for each  $y_{<1>}$  of these three words : we calculate the probabilities (  $3 \times 10000$  probabilities ) of  $P(y_{<2>} | X, y_{<1>})$  and we will pass to the 3<sup>rd</sup> step with the  $y_{<1>}, y_{<2>}$  which has the 3 best probabilities



- We found that the best probabilities goes to  $y_{<1>}, y_{<2>} = \text{"In September"}$  , "Jane is" and "Jane visits"

## Beam search ( $B = 3$ )



- And we continue with same approach ( calculating  $B \times \text{vocab\_size}$  probabilities ) until we get <EOS> a token which defines the end of Sentence



➤ Refining the Beam Search algorithm using Length normalization :

## Length normalization

$$\begin{aligned}
 & \arg \max_y \prod_{t=1}^{T_y} P(y^{<t>} | x, y^{<1>}, \dots, y^{<t-1>}) \\
 & \quad \text{log} \quad \arg \max_y \sum_{t=1}^{T_y} \log P(y^{<t>} | x, y^{<1>}, \dots, y^{<t-1>}) \leftarrow \\
 & \quad \frac{1}{T_y^\alpha} \sum_{t=1}^{T_y} \log P(y^{<t>} | x, y^{<1>}, \dots, y^{<t-1>}) \quad \alpha = 0.7 \quad \begin{matrix} \alpha = 1 \\ \alpha = 0 \end{matrix}
 \end{aligned}$$

Handwritten notes and diagrams:

- Top right:  $P(y^{<1>} \dots y^{<T_y>} | x) = \frac{P(y^{<1>} | x) P(y^{<2>} | x, y^{<1>}) \dots P(y^{<T_y>} | x, y^{<1>}, \dots, y^{<T_y-1>})}{P(y^{<T_y>} | x, y^{<1>}, \dots, y^{<T_y-1>})}$
- Middle right: A diagram showing a circle with a vertical line and a horizontal line intersecting at the center. To the right of the circle, the text  $\log P(y|x) \leftarrow P(y|x) \leftarrow$  is written.

- Explaining the problem with Beam search calculations : By calculating  $P(y^{<1>}, y^{<2>}, \dots, y^{<t>} | X)$  we are doing a multiplication of  $P(y^{<1>} | X) * P(y^{<2>} | X, y^{<1>}) * \dots * P(y^{<t>} | X, y^{<1>}, y^{<2>}, \dots, y^{<t-1>})$  ( look to the first equation ) : and since  $0 < P(x) < 1 \dots$  Doing several multiplications will make the result goes to a very tiny values so close to 0 that the computer may find hard to store their values .... So We will apply the  $\log()$  that will transform the multiplication of values between 0 and 1 into a sum of negative values ( $< \log(1) = 0$ ) and we will take the max value ( the most closer one to zero ) ( look to the second equation )
- We can refine the calculations more by doing the normalization, by dividing the sum of logs by the length of the output  $T_y$  to get the average ( look to the third equation ) or replacing  $T_y$  by  $T_y^{\alpha}$  with alpha between 0 and 1 ( alpha =0 => no normalization and alpha =1 => full normalization )

➤ Choosing the right Beam Width B :

- Choosing a little B ( 10 or less ) will make the calculations faster but will give us a not very good solution
- Choosing a big B will slow the calculations because there is many possibilities to compute for each iterations but It will give us a more precise solution in the end



- Error analysis : Our model didn't predict well , is it because the RNN architecture or because the Beam Search with a wrong B :

## Error analysis on beam search

Human: Jane visits Africa in September. ( $y^*$ )

$$P(y^*|x)$$

Algorithm: Jane visited Africa last September. ( $\hat{y}$ )

$$P(\hat{y}|x)$$

Case 1:  $P(y^*|x) > P(\hat{y}|x) \leftarrow \arg \max_y P(y|x)$

Beam search chose  $\hat{y}$ . But  $y^*$  attains higher  $P(y|x)$ .

Conclusion: Beam search is at fault.

Case 2:  $P(y^*|x) \leq P(\hat{y}|x) \leftarrow$

$y^*$  is a better translation than  $\hat{y}$ . But RNN predicted  $P(y^*|x) < P(\hat{y}|x)$ .

Conclusion: RNN model is at fault.

- The role of our RNN is to calculate and define  $P(Y|X)$  and the role of the beam search algorithm to find the  $Y$  that gives the max ( or a very high ) value of  $P(X|Y)$
- If we note the perfect translation of our input "Jane visite l'Afrique en Septembre" by  $y^*$  and a not good translation by  $\hat{y}$  we will fall in one of two unique cases :
  - o Case 1 : our RNN model gives us  $P(y^*|X) > P(\hat{y}|X)$  but we got at the end as output  $\hat{y}$  :
    - We notice that RNN did his job correctly by favorizing and maximizing  $P(y^*|X)$  over the wrong translations like  $\hat{y}$  so we conclude that **the fault is in the beam search algorithm**
  - o Case 2 : our RNN model gives us  $P(y^*|X) < P(\hat{y}|X)$ 
    - We notice that the RNN favorizes the wrong translation  $\hat{y}$  over the perfect one  $y^*$  , so it's not the Beam Search algorithm fault to gives us as an output  $\hat{y}$  so we conclude that **the fault is in our RNN architecture**

➤ The error analysis Process :

Human	Algorithm	$P(y^* x)$	$P(\hat{y} x)$	At fault?
Jane visits Africa in September.	Jane visited Africa last September.	$2 \times 10^{-10}$	$1 \times 10^{-10}$	B
...	...	—	—	R
...	...	—	—	B
				R
				R
				...

Figures out what fraction of errors are “due to” beam search vs. RNN model

- We iterate through our training set and particularly for the inputs where we got a wrong translations , we calculate and compare the values of  $P(y^*|X)$  and  $p(\hat{y}|X)$  and conclude for each sequence wither it's Beam search algorithm B fault or the RNN architecture fault R , we compare the two proportions and we decide in which part we gonna focus our ameliorations to have a better prediction later on

## XXI. Evaluating the machine translations using the BLEU Score:

French: Le chat est sur le tapis.

Reference 1: The cat is on the mat. ←

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

MT output: the the the the the the the.

Let's say we have in the training set a French sentence "le chat est sur le tapis" and its corresponding human perfect translations "The cat is on the mat" and "There is a cat on the mat", if our model output is "the the the the the the the" ... How can we know mathematically it's a wrong output with a very low precision close to 0 ?

The solution is : **BLEU ( Bilingual evaluation understudy ) Score**

## AA. Understanding Bleu score : calculating unigrams bleu score ( P1 ) :

French: Le chat est sur le tapis.

→ Reference 1: The cat is on the mat. ←  
2 occurrences

→ Reference 2: There is a cat on the mat. ←

→ MT output: the the the the the the the.

For each distinct word  $w$  in the MT output ( which is only "the" in this case) , we calculate the ratio  $\text{CountClip}(w)/\text{Count}(w)$  and we do the sum

- Count : is the number of occurrences of the word in the Mt output ( = 7 in this case )
- Count Clip : is the maximum number of appearance of the word in the Human translations : in this example  $\text{Count Clip}(\text{"the"}) = \text{Max}(\text{Ref1Count}(\text{"the"}), \text{Ref2Count}(\text{"the"})) = \text{Max}(2,1) = 2$

The final precision is 2/7 which is quite far from the perfect one ( 1) so we conclude that we have a bad translation

## A. Understanding Bleu score : calculating bigrams bleu score ( P2 ) :

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. ←

	Count	Count <sub>clip</sub>
the cat	2	1
cat the	1	0
cat on	1	1
on the	1	1
the mat	1	1

For the bigrams , Instead of picking by a single word , we pick all the possible combinations of 2 of the closest words and we count their Count and their CountClip

- For w="The cat" for example , its count is 2 in the MT output and its count clip is  $\text{Max}(1,1)=1$  ( because "the cat" exists in both human references )

We get in the end precision equals to  $4/6$  for "The car the cat on the mat" which is bigger than the ones we got with " the the ...the" and it makes a sense !

## B. The final Formula of Bleu Score :

$p_n$  = Bleu score on n-grams only

$p_1, p_2, p_3, p_4$

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

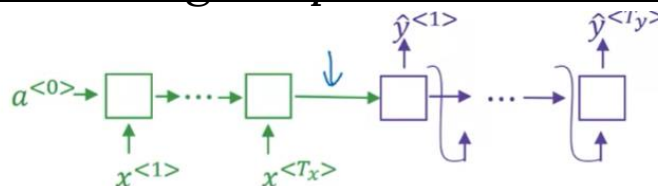
$BP$  = brevity penalty

$$BP = \begin{cases} 1 & \text{if } \underline{MT\_output\_length} > \underline{reference\_output\_length} \\ \exp(1 - MT\_output\_length / reference\_output\_length) & \text{otherwise} \end{cases}$$

- We will Sum  $P_1, P_2, P_3$  and  $P_4$  and we apply the exponential function to the sum
- We multiply the result by a function called BP : Brevity Penalty
  - o The Brevity Penalty function role is reduce the score for the short MT translations since it's easy to get a high score for the short MT outputs even if it's a wrong translation

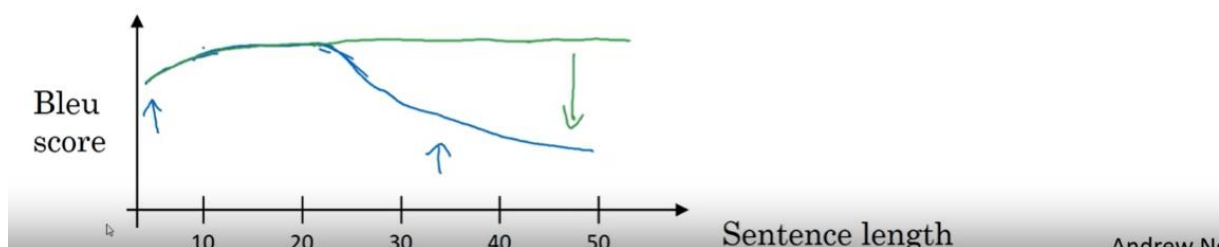
## XXII. Attention Model:

### C. Understanding the problem with long sequences:



Jane s'est rendue en Afrique en septembre dernier, a apprécié la culture et a rencontré beaucoup de gens merveilleux; elle est revenue en parlant comment son voyage était merveilleux, et elle me tente d'y aller aussi.

Jane went to Africa last September, and enjoyed the culture and met many wonderful people; she came back raving about how wonderful her trip was, and is tempting me to go too.

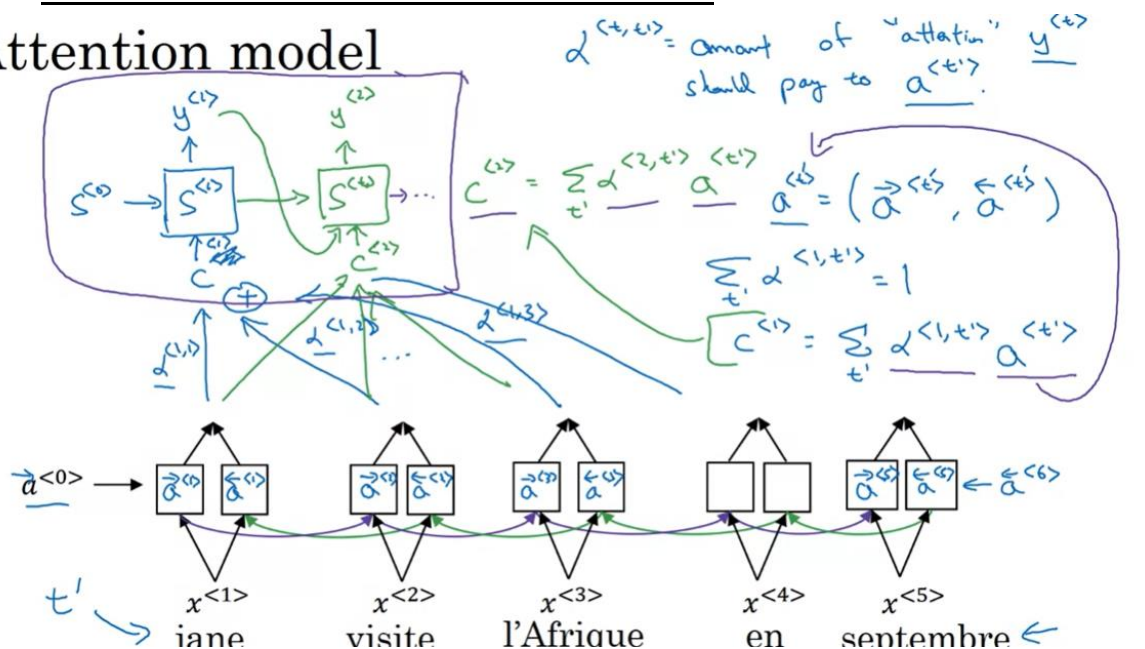


If we want for example translate this three lines long French sentence , it would be so hard for the human brain to memorize the whole sequence right before doing the translation , The human translator will just try to do the translations part by part , for

each part there is just some words that he have to keep in his mind ( which are the context ) for the given part , And we want to do the same approach with our RNN model because it's proven that it has a lower bleu score for the sentences with a big length like it's shown In the graph

## D.Attention Model Architecture:

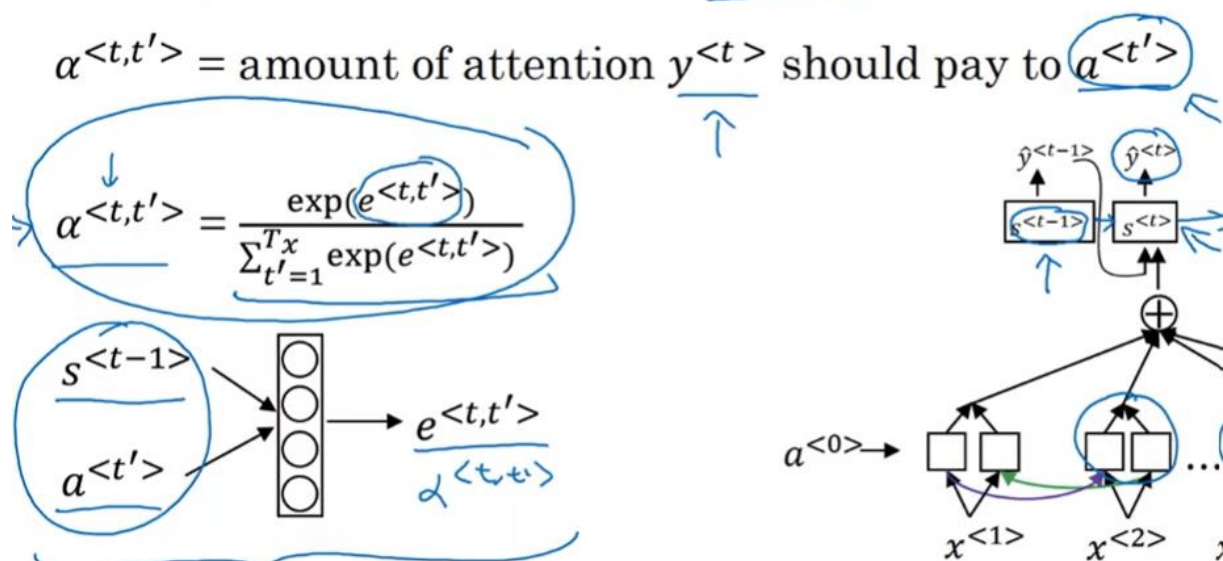
### Attention model



- We are going to have 2 RNN models:
  - o The first one (the bottom of the picture):
    - Its role is to calculate the attention weights  $\alpha(i, j)$  as outputs
      - $\alpha(i, j)$ : how much you should give attention to the  $j^{\text{th}}$  input word (X) while generating the  $i^{\text{th}}$  output word (Y)
    - it's a BRNN because we have to consider the whole sequence and not just the previous words while calculating the attention weights
    - we give a notation  $a^{<t>}$  to represent the forward and backward hidden states ( $a^{<t>->}$ ,  $a^{<t>-<-}$ )
  - o The second one (the top of the picture):
    - It's the one who will generate the output (The Machine translation)
    - It has its own hidden states, we notate him by  $S^{<t>}$  to distinguish it from the first model's hidden state
    - It takes as an input (its  $x^{<t>}$ ) the different attention weights related to  $\alpha(t, j)$  ( $j$  from 1 to  $T_x$ ), they all represent the context  $c^{<t>}$  (which is  $x^{<t>}$  for RNN) for the output  $y^{<t>}$  and the hidden state  $S^{<t-1>}$



## E. Formulas to calculate $\alpha_{<t,t'>}$

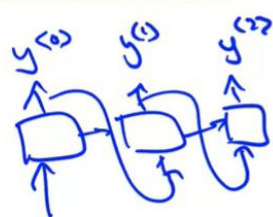


## XXIII. Transformers Intuition :

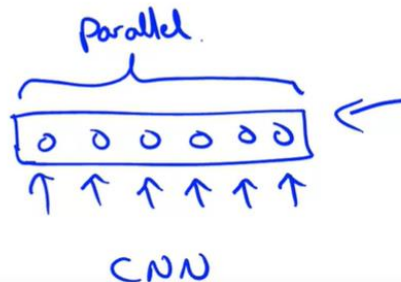
### • Attention + CNN

#### • Self-Attention

#### • Multi-Head Attention



$A^{(1)} A^{(2)} A^{(3)} A^{(4)} A^{(5)}$

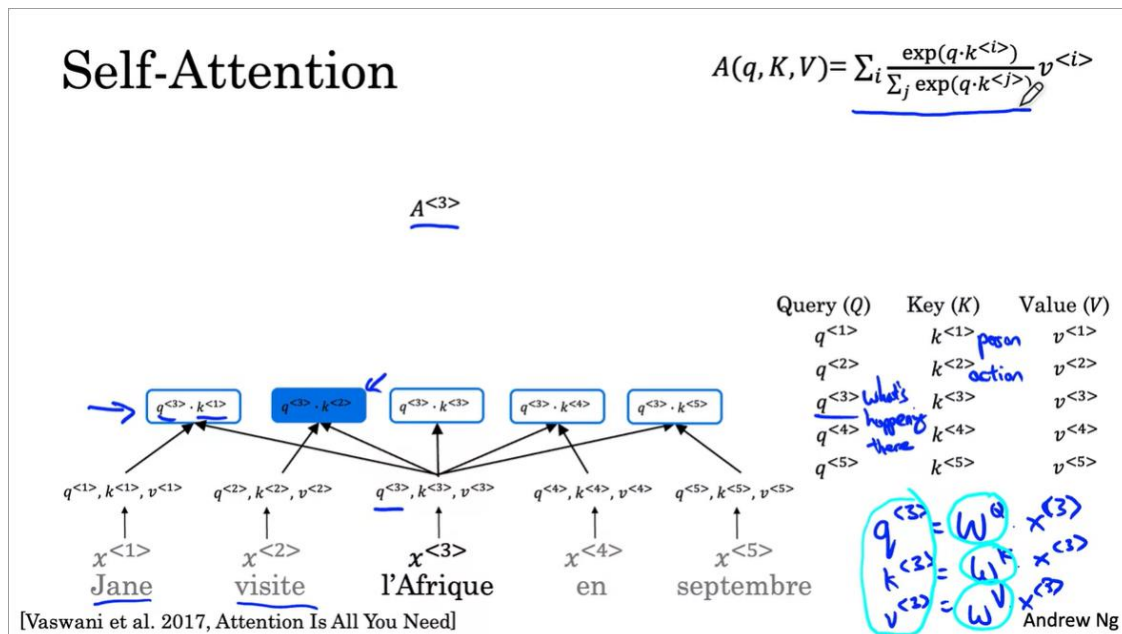


- Transformers are a combination of CNN architecture and the Attention-mechanism I saw in the previous week
  - o CNN:
    - Instead of reading the sequence : word by word , We will do a parallel treatments in the whole sequence , so It's not a RNN anymore ... and this will make the training phase quicker
  - o Attention:
    - For each word in the sequence: we will calculate the attention weights relative to the other words



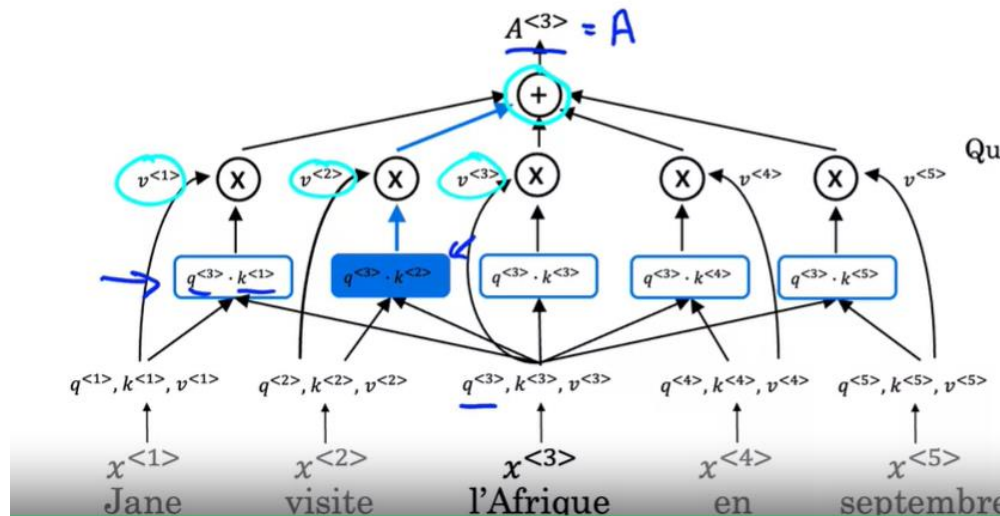
- There are two kinds of attentions: Self Attention and Multi-Head Attention

## XXIV. Explaining self-attention mechanism:



- The goal of the self-attention mechanism is rather than using the word embedding of a certain word for the embedding matrix (like “Afrique” for example) which is totally independent from the sentence context (talking about Africa in economy is different from talking about it in tourism, geography, History ....etc.) : We are going to build “our” embedding vector for the words in our sequence which is dependent of the sentence’s context **it’s the A<t>**
- Each word  $x^{<t>}$  going to have its corresponding  $q^{<t>}$ ,  $k^{<t>}$  and  $v^{<t>}$  :
  - Q<t>**: the query, represents the most relatable question to the word  $x^{<t>}$  depending to the sentence context: for  $x^{<3>} = \text{“Africa”}$  we have the corresponding vector of the question  $q^{<3>}$  : “What’s happening there?”, We got it by multiplying  $x^{<3>}$  with the matrix parameters  $W_q$
  - K<t>**: the key, it represents the nature of the corresponding word  $x^{<t>}$ , In the example  $x^{<1>} = \text{“Jane”}$  has  $k^{<1>} = \text{“person”}$  and  $x^{<2>} = \text{“visite”}$  has  $k^{<2>} = \text{“action”}$ , We got it by multiplying  $x^{<3>}$  with the matrix parameters  $W_k$
  - V<t>**: the Value, I don’t have a confirmed information but It’s most likely the original word embedding of the word  $x^{<t>}$ , We got it by multiplying  $x^{<3>}$  with the matrix parameters  $W_v$

## F. The $A_{<t>}$ computation process :



- For each word  $x_{<t>}$  :
  - o For  $i$  from 1 to  $T_x$ :
    - we are going to calculate the attention weights with the other words by doing the multiplication  $\langle q_{<t>}, k_{<i>} \rangle$
    - Multiply  $\langle q_{<t>}, k_{<i>} \rangle$  by  $v_{<i>}$
  - o Sum the partial Sums to get the final  $A_{<t>}$
- The image shows how to compute  $A$  for  $t=3$ , for  $q_{<3>} = \text{"What's happening there"}$ 
  - o We got the max value in  $\langle q_{<3>}, k_{<2>} \rangle$  ( background blue ) and that means that the  $x_{<2>} = \text{"Visite"}$  is probably the answer of the question "What's happening there" which is related to "Africa" so we will get to know that the "Africa" word is present here in the context of "being visited"

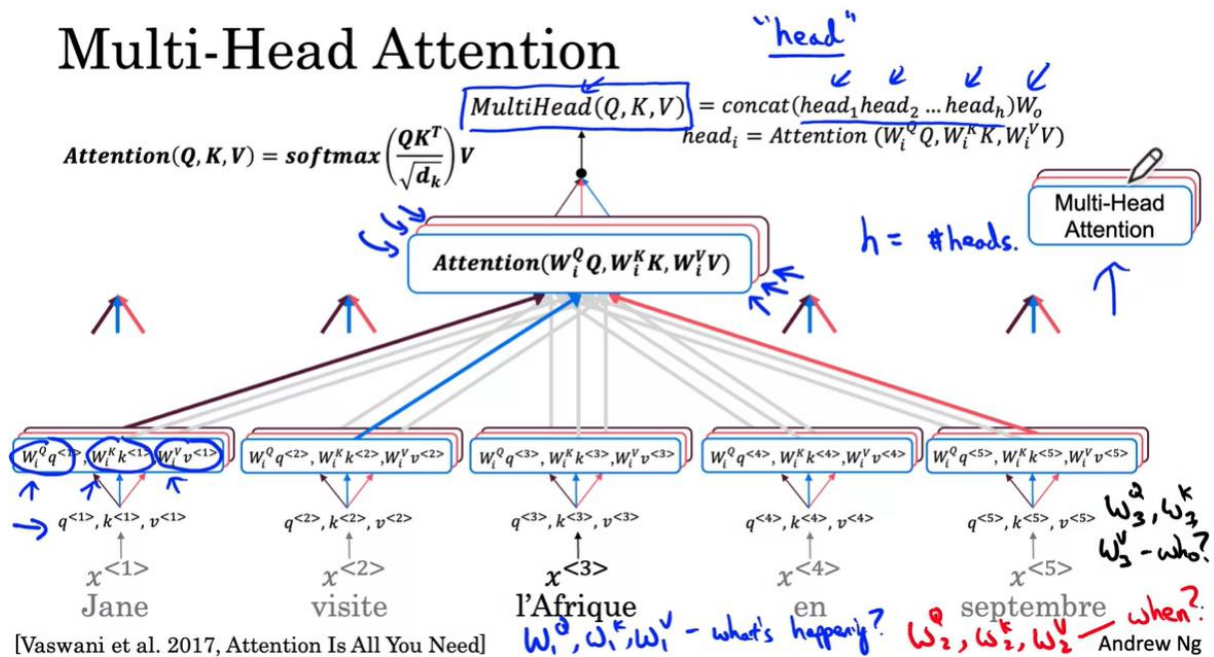
## G. The final formula :

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

- We added  $\sqrt{d_k}$  as a denominator to avoid the values explosion
- The Softmax is important in the  $A_{<1>}, \dots, A_{<5>}$  in order to guarantee that the sum of the vectors is 1

## XXV. Multi-Head attention process

## Multi-Head Attention



- In the Self-attention computation for the word  $x_{<3>}$  = "Africa", It was all about the question  $q_{<3>}$  = "What's happening there"
- For the multi-Head attention process, we are going to repeat the self-attention process  $h$  times but with a different question  $q_{2<3>}$
- To get another questions  $q_{1..h<3>}$ : We are going to use Different Matrix parameters  $W_{1..h}^Q, W_{1..h}^K$
- As illustrated in the picture ( by red lines ), the  $q_{2<3>}$  = "When ?" and the biggest  $\langle q_{2<3>}, k_{2<i>}\rangle$  goes to the word  $I$  = "September"
- And for the third question we have  $q_{3<3>}$  = "Who ?", and the biggest value goes to "Jane"
- And as we see, we already gather many interesting information about "Africa" in our sentence
- Each question self-attention is called "Head" and that's why this mechanism is called "Multi-Head" attention

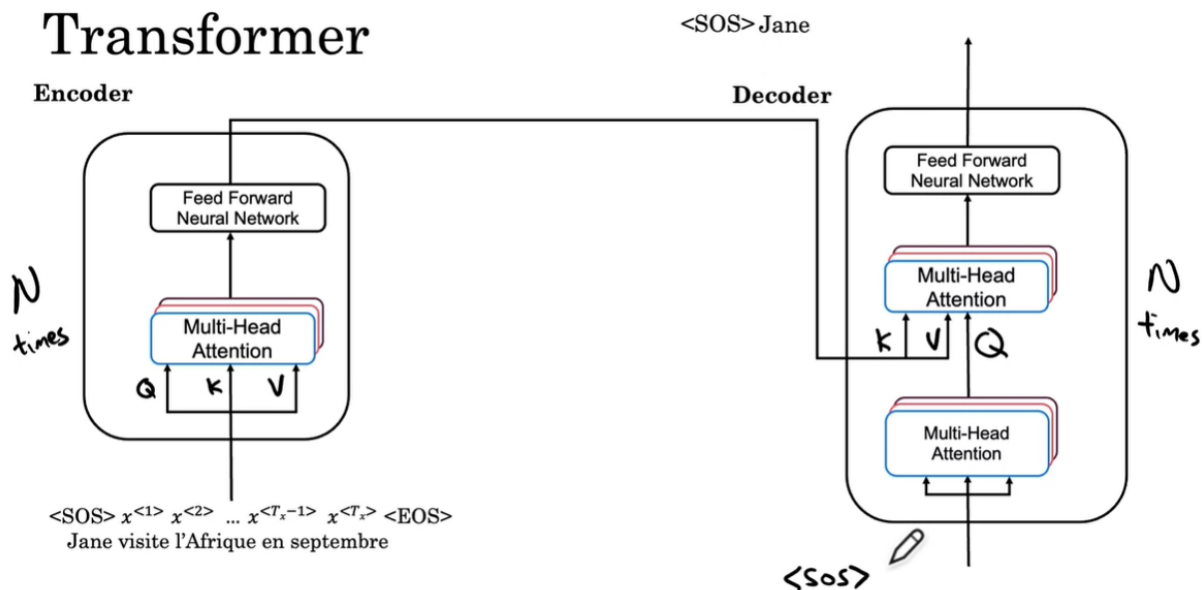
### H. The final formula:

$$MultiHead(Q, K, V) = \text{concat}(\text{head}_1, \text{head}_2, \dots, \text{head}_h)W_o$$

$\text{head}_i = \text{Attention}(W_i^Q Q, W_i^K K, W_i^V V)$

- The multi-Head vector is going to be calculated finally by concatenating the values that we got for each iteration ( each question ) multiplied by the parameter vector  $W_o$

## XXVI. Transformers Network

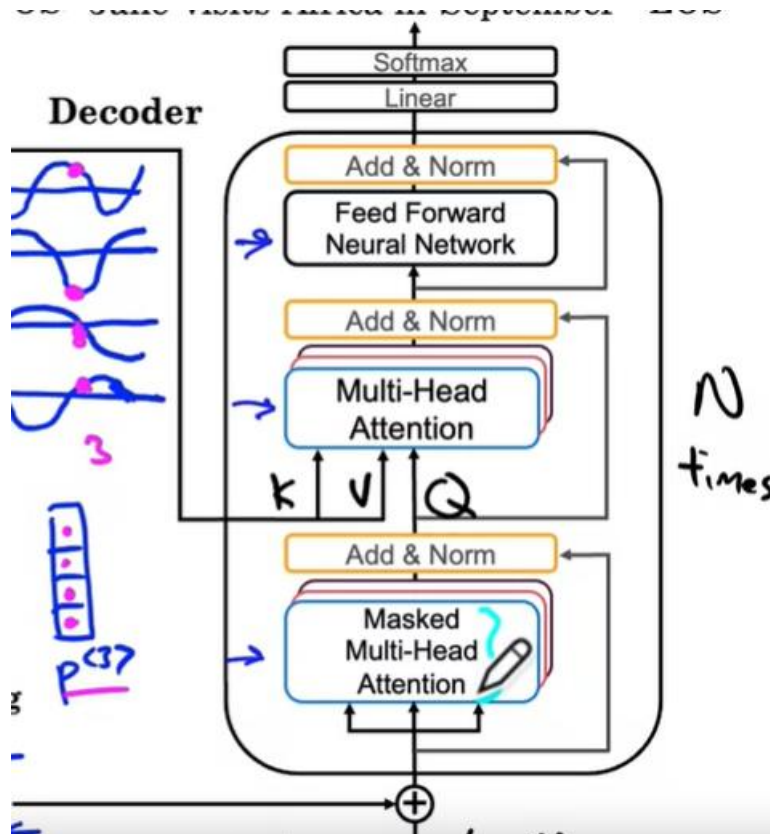


- Transformers architecture is like sequence-to-sequence models : composed of two parts : Encoder and Decoder part
  - o Encoder part:
    - The Multi-Head Attention mechanism we saw is just the first step of the Encoder section: it's fed by the matrices parameters: Q, K, V
    - The output of the Multi-Head Attention will be the input layer of a Feed Forward neural network
    - This process will be looped N times ( we will loop through the Encoder block N times before going to the Decoder part )
  - o Decoder part:
    - The job of the decoder part is to output the English translation
    - The role of the first Multi-Head Attention layer in the decoder to be fed by the already generated word , For the first time it will be only the  $\langle \text{SoS} \rangle$  token which indicates the “start of the sequence”
    - The already generated sequence will be used to calculate Q,K and V of this first layer of Multi-Head Attention , its role is to generate the appropriate Q matrix for the next word to generate  $y^{(t+1)}$  in the second layer of Multi-Head Attention
    - The K and V matrix for the second layer of multi-Head attention will be generated using the output of the Encoder part
    - And then the output of this second layer will be the input layer of a Feed forward neural network
    - We loop through this block N times before generating the generated word after “ $\langle \text{SoS} \rangle$ ” which is “Jane” in this case
    - To generate the next word “Visits” , we will repeat the Decoder process with “ $\langle \text{SOS} \rangle \text{Jane}$ ” as input to the first layer of Multi-Head Attention layer instead of “ $\langle \text{SOS} \rangle$ ” only , until we generate the “ $\langle \text{EOS} \rangle$ ” token which stands for “end of sequence” ( its input will be “ $\langle \text{SOS} \rangle \text{Jane visits Africa in September}$ ” and the output will be “ $\langle \text{SOS} \rangle \text{Jane visits Africa in September } \underline{\langle \text{EOS} \rangle}$ ” )

## I. Further Details about the transformers

### Architecture :

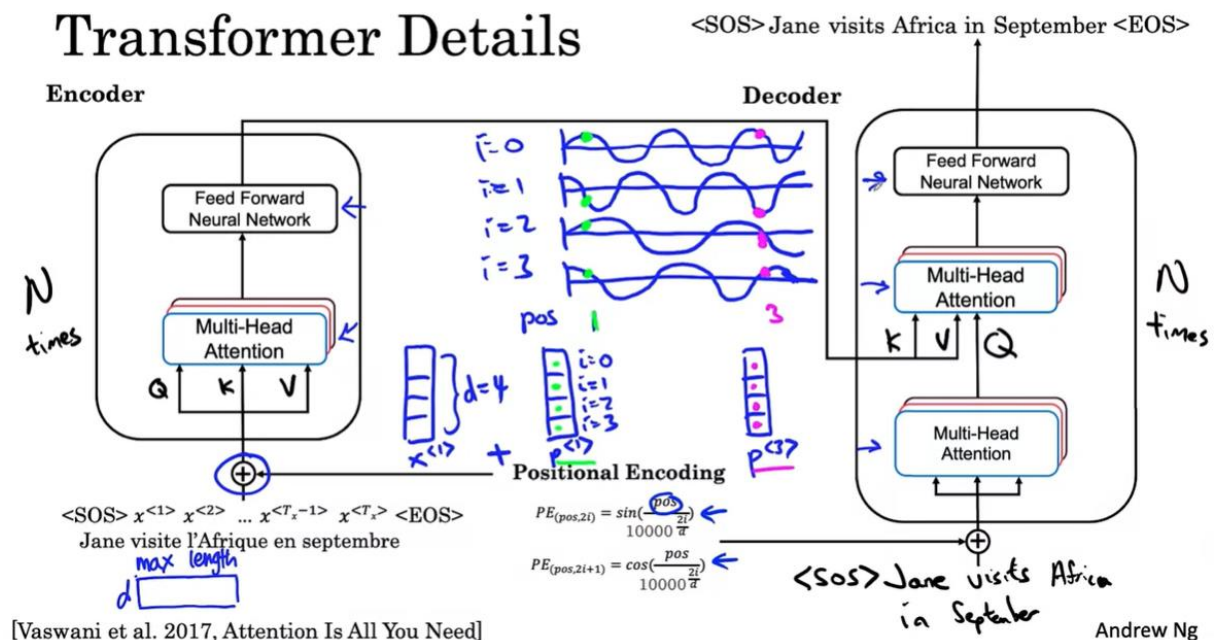
- The Transformer output (Decoder output ) :



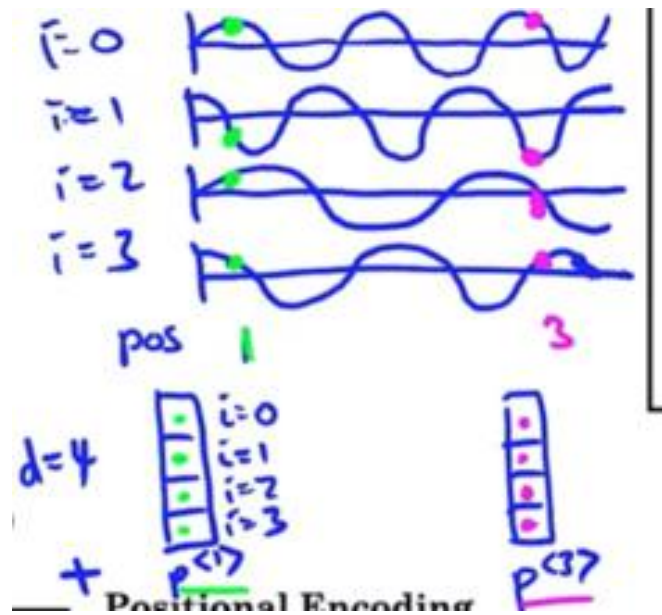
- Since the problem is to find the most suitable word ( translation ) , we are in multi classification problem so We Are going to use SoftMax Activation Layer fed to a Linear Layer ( equivalent to Keras Dense Layer )



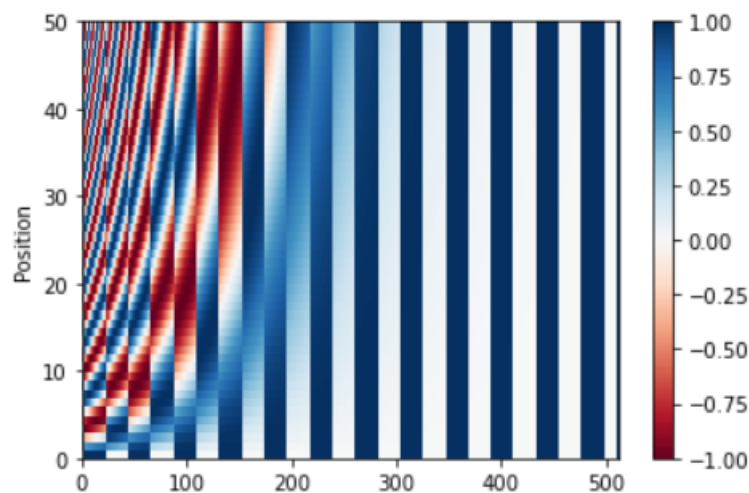
➤ Adding Positional Encoding part to the input:



- In Order to make our architecture better, we are going to add an additional vector to the input which defines the words positions (because the positions of the word is so important while doing the machine translations )
- Suppose that the word embedding vector  $X^{(t)}$  has 4 values so its dimension is  $d=4$
- The positional embedded vector for each word will have the same dimension as the word embedding vectors have (  $d=4$  )  $P^{(t)}$
- In the  $PE(pos, 2i)$  and  $PE(pos, 2i+1)$ ,  $pos$  is the numerical position of the word in the input sequence so  $pos(\text{"Jane"}) = 1$ , and  $i$  is the cell position under the positional embedding, the cells which has a pair index  $(2i)$  will use Sinus function to calculate its position encoding value while the odd index  $(2i+1)$  will use the Cosine Function
- Notice that  $PE(pos, 0)$  and  $PE(pos, 1)$  have the same angle to calculate its cosine/sine,  $PE(pos, 2)$  and  $PE(pos, 3)$  has similar angle too. The main difference between  $2i$  and  $2i+1$  is that the first uses the Angle sine while the second uses the Angle cosine
- The role of Cos and Sin functions is to guarantee unique positional vectors among the words and this is thanks to "pos" value in the nominator of PE formula (notice the differences of values between  $P^{(1)}$  and  $P^{(3)}$  in the image below, even there is some similarity in some indexes but they never can have the identical vectors)
- The input of the Encoder and Decoder part will be the sum of the words embeddings and their positional Encoding instead of just the word embeddings

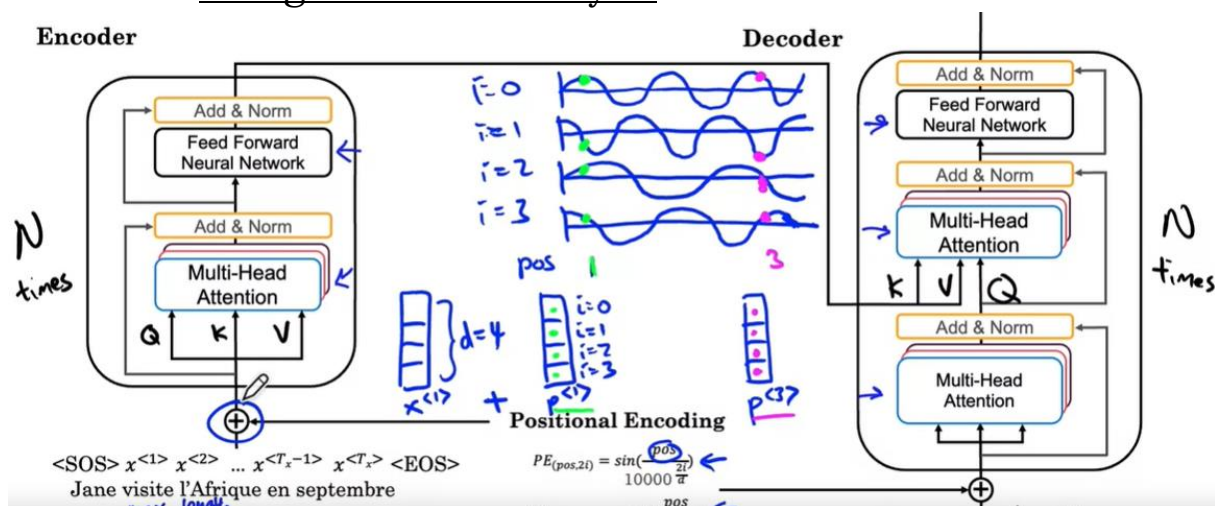


- And for the positional Encoding vectors , for each word we have a one , so we have  $T \times (\text{max Length})$  vectors , each one has number of elements equals to the word embedding vector dimension (  $d$  ) so the final dimension is  $T \times d$  ,
- The picture below illustrates how each position has its own unique positional encoding vector:



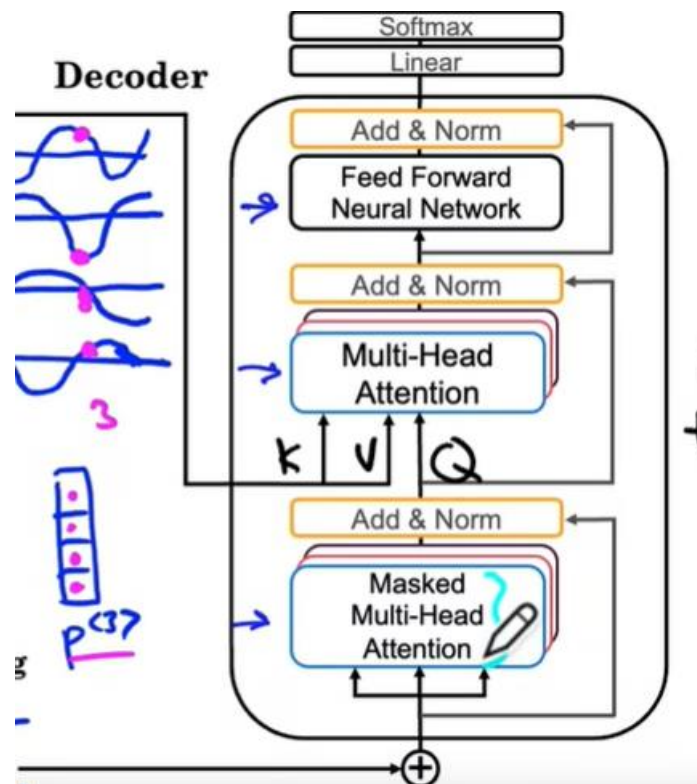


➤ Using Add & Norm layer:



- Batch Normalization layer is a layer used between each two layers of a very deep neural network to ensure that the input weights of each batch are standardized ( mean =0 and std = 0 ) , its role is to accelerate the learning during the training phase by avoiding the problem of coordinating updates across many layers due to having a numerical difference between their weights
- This Add & Norm layer in the Transformers has exactly the same role ( accelerate the training by the normalization process )

➤ Masked Multi-head Attention layer in the decoder :



- This layer will be useful only in the training phase
- Its job is to repeatedly pretends that the network had perfectly translated the first few words and hides the remaining words to see if given a perfect first part of the translation, whether the neural network can predict the next word in the sequence accurately.
- Example: we give to the decoder as an input “Jane visits Africa” which is the perfect translation of the first part, and we let them without our guidance continue the translation
- The kind of masking is called “Look-ahead mask”

## XXVII. Additional details and conclusion:

### J. NLP with Transformers in brief words:

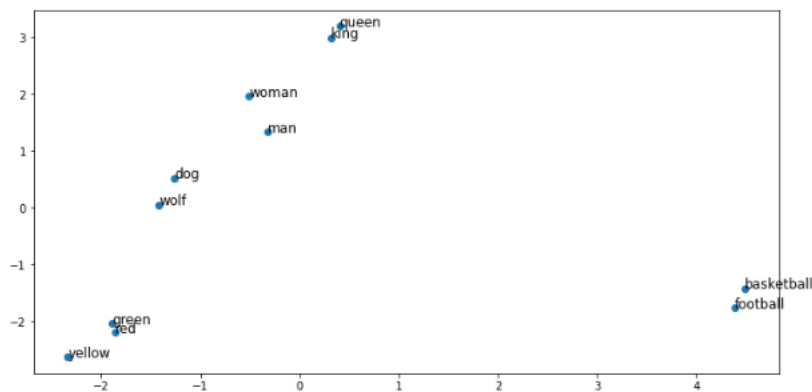
- It is a standard practice in natural language processing tasks to convert sentences into tokens before feeding texts into a language model. Each token is then converted into a numerical vector of fixed length called an embedding, which captures the meaning of the words. In the Transformer architecture, a **positional encoding** vector is added to the embedding to pass positional information throughout the model.

## K. The Positional Encoding vector has a big impact in the word representations:

```
texts = ['king queen man woman dog wolf football basketball red green yellow',  
        'man queen yellow basketball green dog woman football king red wolf']
```

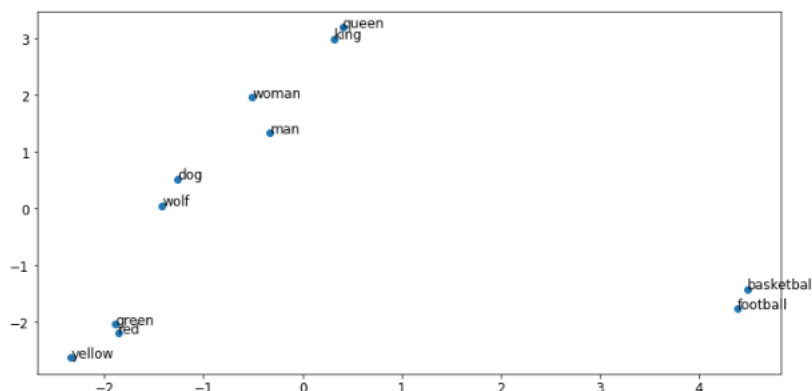
- To show the impact of the positional encoding, we did a PCA representation of the words embedding vectors of these two sentences
- These two sentences haven't any sense , the first one put the words whose had a similar semantic : close to each others ( king & queen ) , ( man & woman ) ,...etc while the second sentence used the same words in the first sentence but in a random order
- By displaying the representation of word\_embeddings :

```
In [19]: plot_words(embedding, sequences, 0)
```

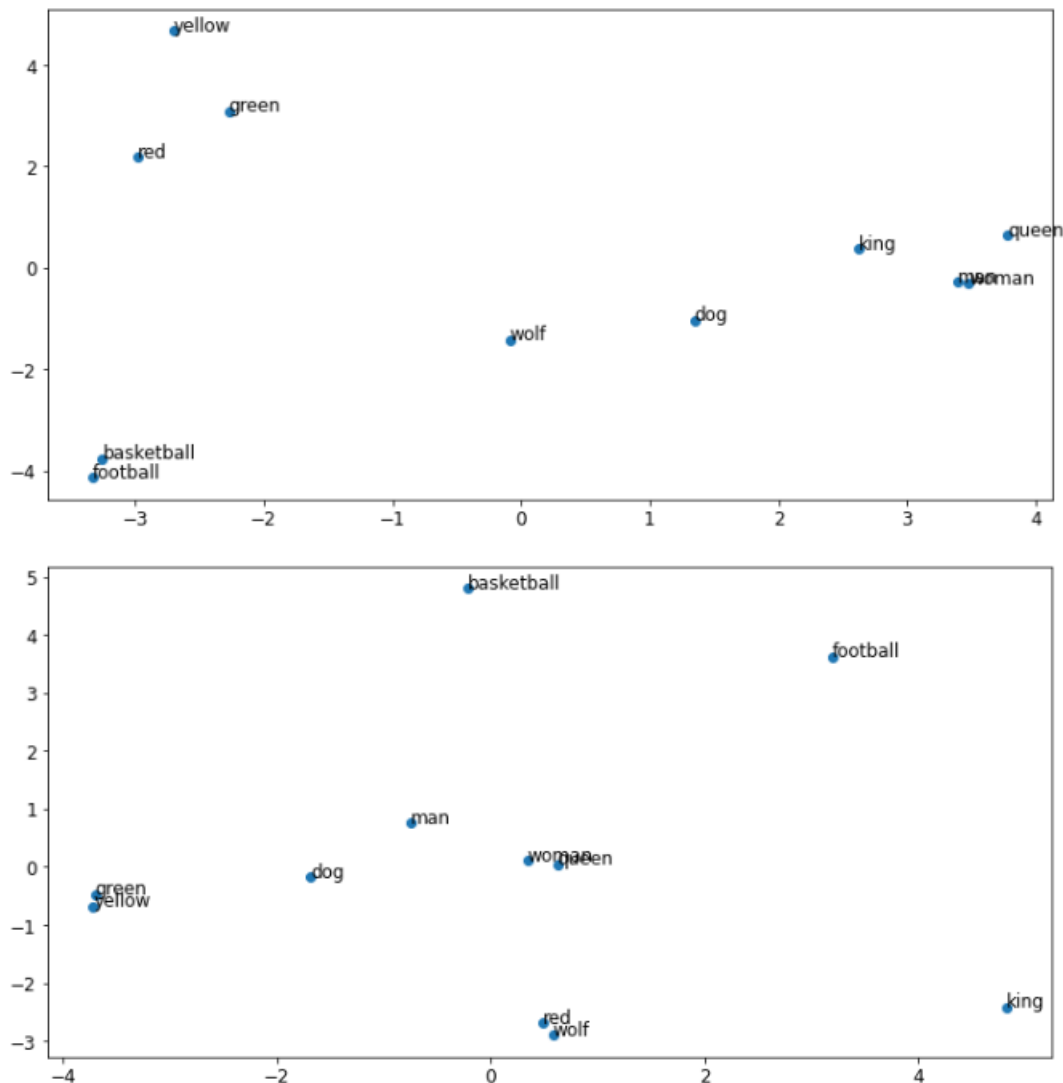


Plot the word of embeddings of the second sentence. Recall that the second sentence contains the same words as the first sentence, just in a different order. You can see that the order of the words does not affect the vector representations.

```
In [20]: plot_words(embedding, sequences, 1)
```



- We notice that we have an identical word representations in both sequences which means that the word embeddings don't get affected by the word positions and the words who has a close semantic , appeared near each others ( dog -wolf , green-red-yellow ,...)
- We display now the word representations of word\_embedding + positional encoding vector:



- Firstly, we notice there is a huge difference between the two charts that we saw before
- In the first chart , we can still see the related words appeared kinda near to each other ( basketball-football , men-women , ....) , and this is because they appeared in adjacent positions in the first sentence
- But in the second chart which corresponds to the second sentence , We notice that there is a huge mess happened , we see red & wolf close to each other , dog & man too
- And this example showed to us how much is important the words positions for the machine translation and their big impacts for the word representations