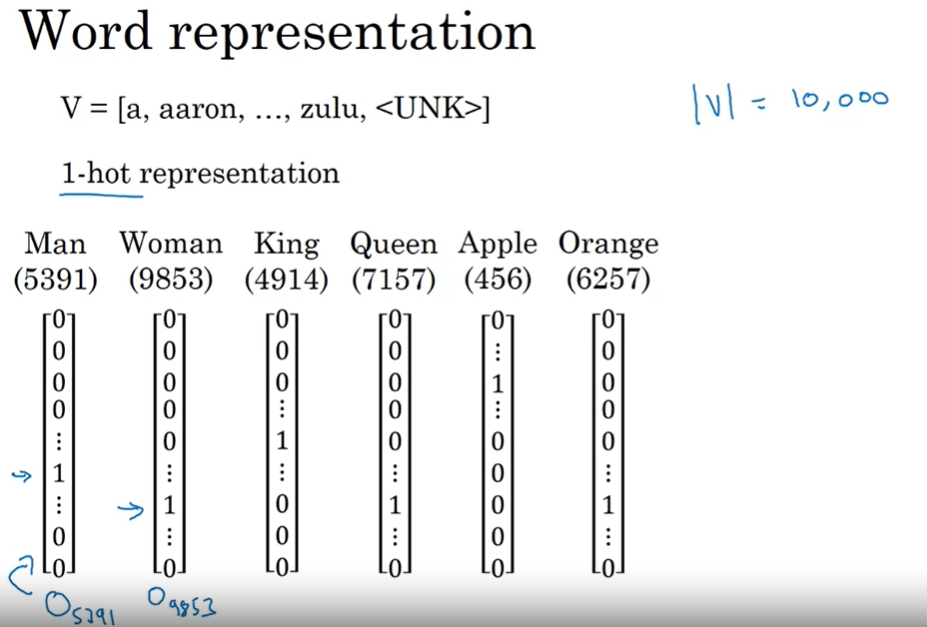
## Word représentation

### One-Hot vector :

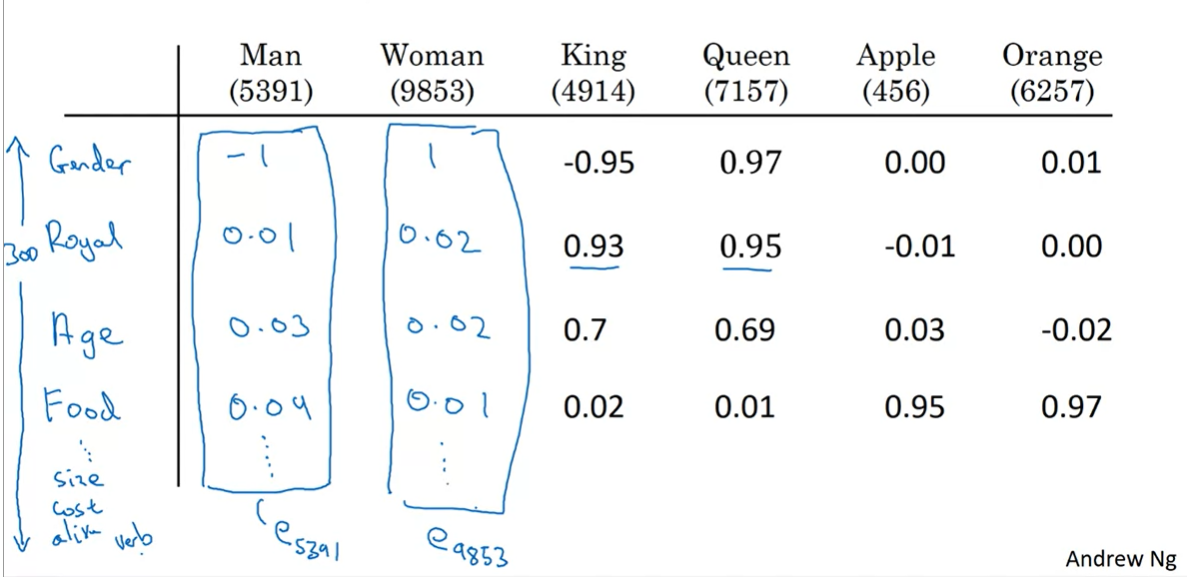


* This is the kind of the representation I used to see in the first week , where each word is represented by a vector having a length equal to the vocabulary size , every vector element is a zero except for the index of the word
  + O5791 is a notation to represent the vector of the word “man” which is indexed by 5391 in our vocabulary

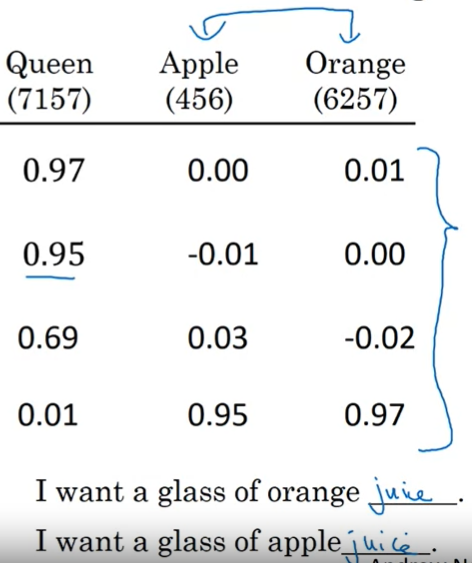
#### Disadvantage of this representation :

* 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 .
  + 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

### Word embedding : Featurized representaion :



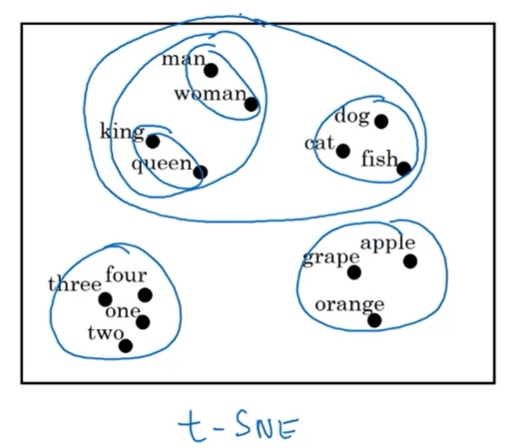
* In this representation , each word will be represented by a set of features with values between -1 and 1 ( for gender for example : -1 means male and 1 means female ) , so know the length of the vector depends of number of features to take in consideration instead of our vocabulary size
* Example of features: gender, age , food , size , cost , …
* This representation is so important to detect the similarities between the words
  + We can see that King and Queen are both royal words for example
  + 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
  + 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 :



#### 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.



## 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

## 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

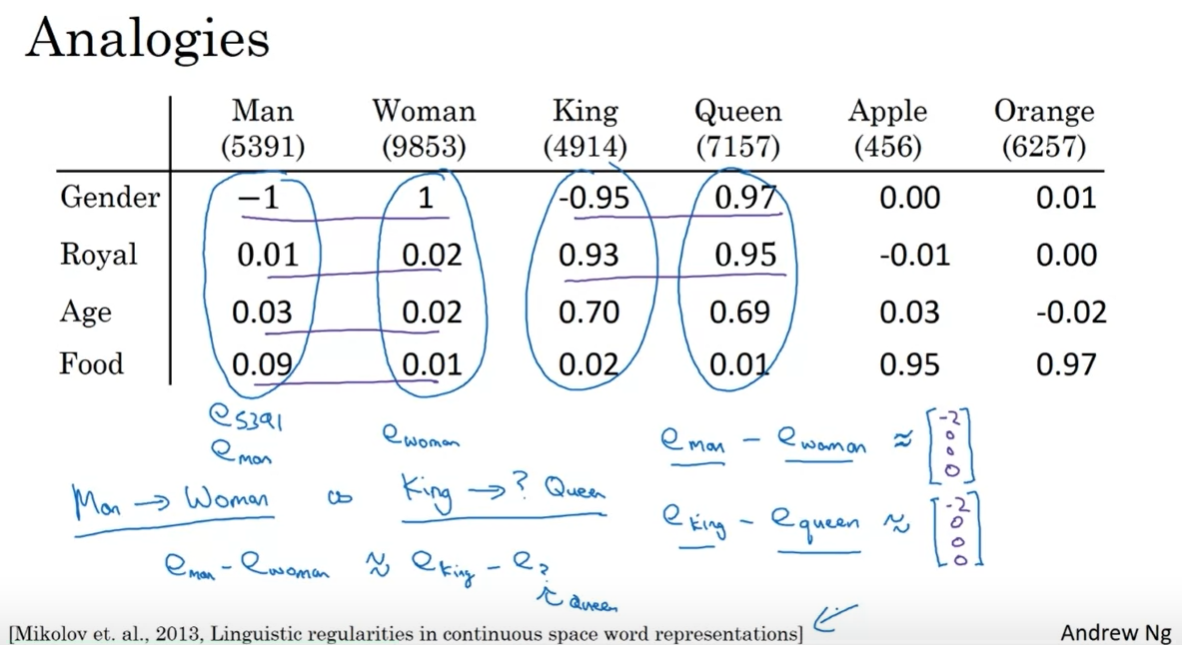
## Word embeddings properties:

“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 propery is called **“word analogies”** .

### How’s that !

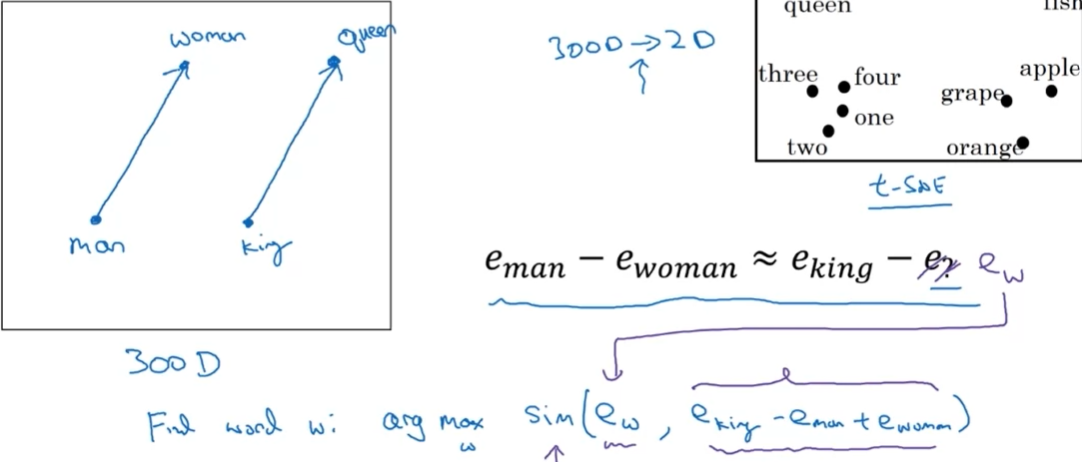
Well , let’s do quick reminder about how the world are represented in the word embeddings :



As the slide illustrates , We calculate the vector representing the gap between “man” and “woman” eman-ewoman .

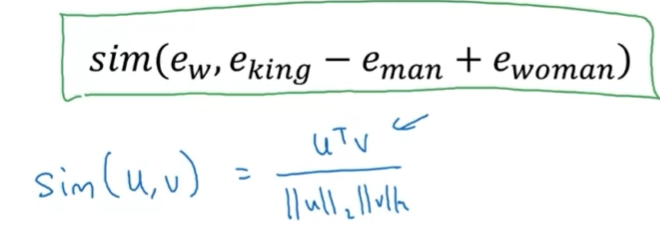
And we calculate the vector between equeen with all the words in the vocabulary in order to get the word “x” which has a distance with “queen” equals the distance between “man” and “woman”

### Calculating the similarity :

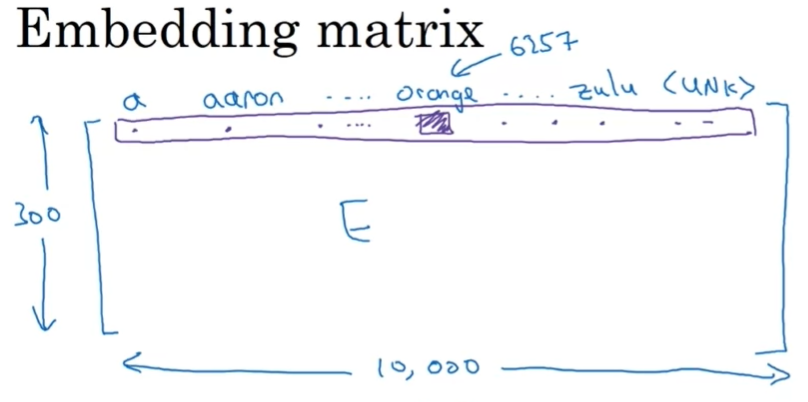


In other way , we will calculate the similarity between the vector representing eking-eman+ewoman with all the words until we got the most similar vector

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



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

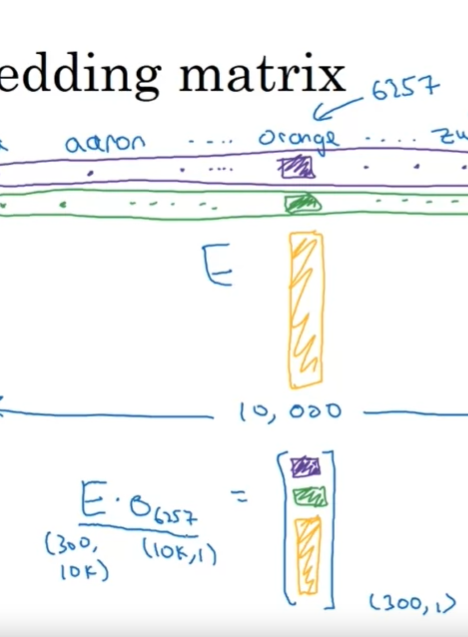


* The Embedding matrix “E” : has number of features\*vocabulary size as a dimension.
* Each word in our vocabulary will be represented by a vector of features : for each feature corresponds a value between -1 and 1

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

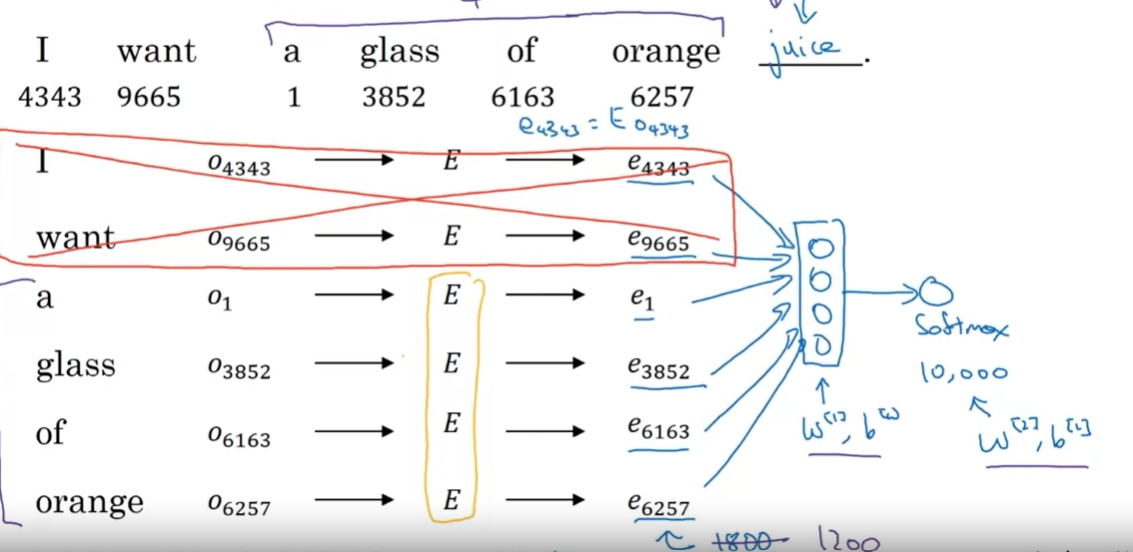
We multiply the Embedding matrix ( 300x10000) by the corresponding one-hot vector of the specific word (10000x1) to get its word embedding representation vector (300x1)

So **e6257 = E x O6257 :**



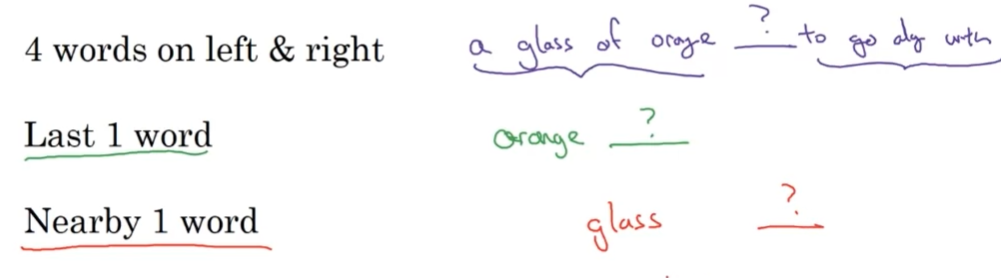
## Learning Word embeddings

### Natural Language Model :



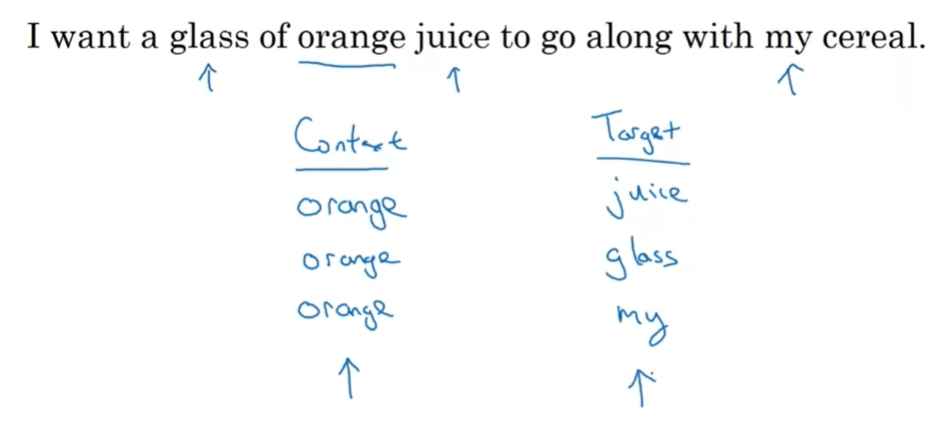
In order to predict a coming word for a previous sequence of words , we should give to our model , **the context** which is for example the four previous words in order to fix the number of inputs which are their features vectors extracted from the features Matrix E , We apply the SoftMax function and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

#### Another possible context :



* 4 words on left 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” )

### Skip-grams algorithm with Word2Vec :



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 ex ); and then we pick a random word from the close range to be the **target (** its embedding vector ey )**.**

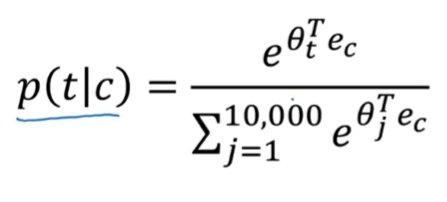
For example: and like the image showed from the given sentence we randomly took “orange” as the context and “juice” as its target and so on.

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 :

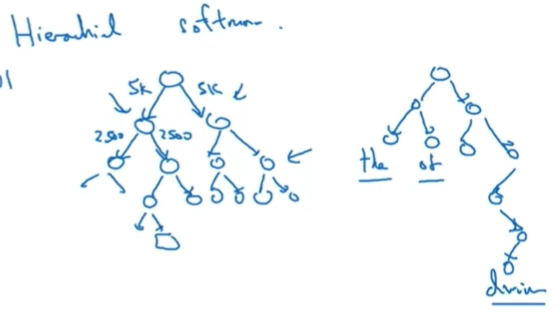


* “t” represents the target and “c” is the context so we calculate “the probability of t given c as a context”
* θt  is a parameter associated to the softmax unit

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

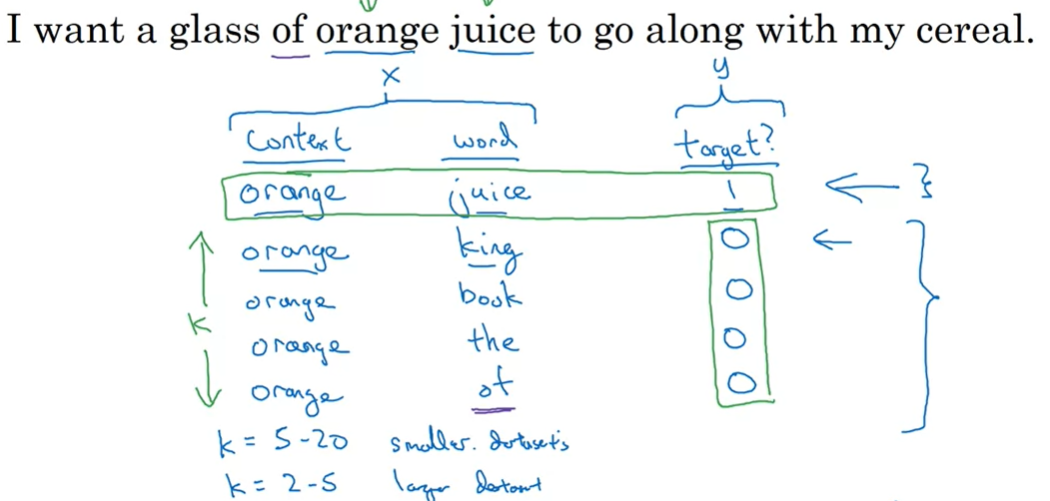
the problem is in the denominator where its calculation is expensive in term of computation , and we ran replace it by a **Hierarchal SoftMax**

#### 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

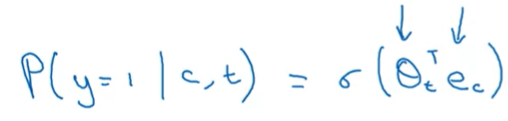
### Negative sampling: an optimized technique to do the word embeddings :

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 :

* 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” world ( the label ) is taken from the same text in the neighborhood of the context world , 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”
  + 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 )

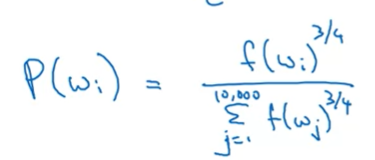


* ΘTt is the parameter associated to the target t , and ec is the feature vector of the context c

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 top 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 ¾:



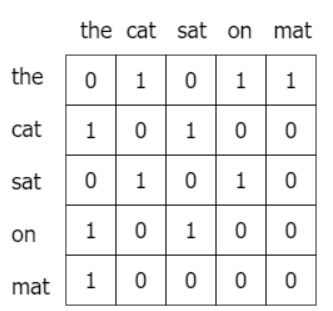
* + - * f(wi) represents the number of apparitions of the word “wi” in our learning text

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

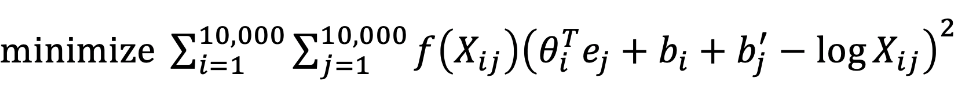
For False Sampling , and Word2Vec we saw that these two algorithms concentrates about the neighborhood between the context and the target for each text corpus , one by one .

In Glove we are going to add an additional parameter named Xij which represents the co-occurrence time of “j” appears in the context of “I” , and Xij=Xji .

And we will gonna have 2D matrix like this :



And the goal is minimizing:

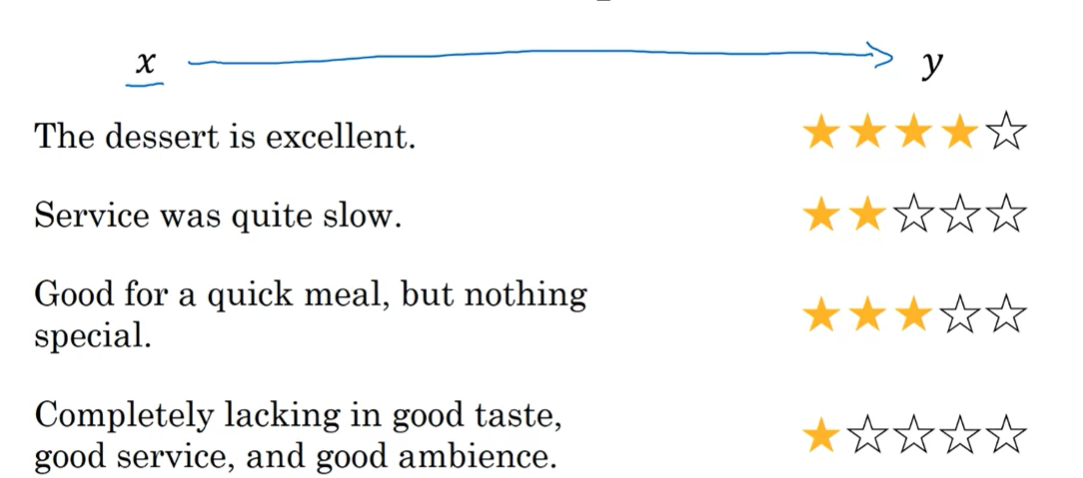


* The role of f(Xij ) :
  + If Xij=0 then f(Xij) = 0
  + It will reduce the very large value of the formula (which concerns the words which appears a lot like “a” , “the’, “of”
  + It will increase slightly the very little value of the formula (which concerns the words which appears rarely like “Dorian”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

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

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

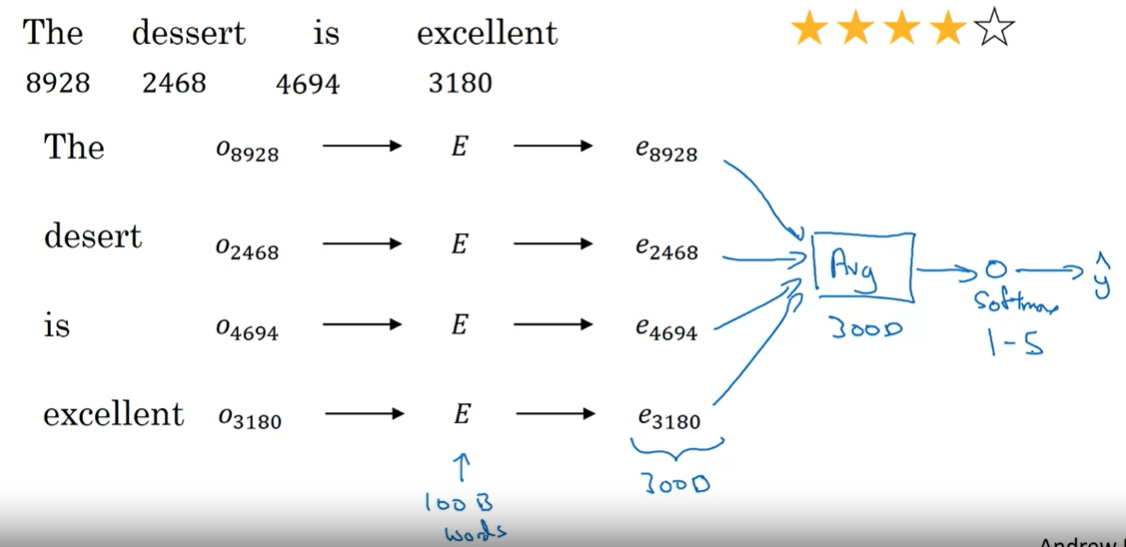
### The presentation of the problem :



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

* For sentiment analysistasks: 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

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

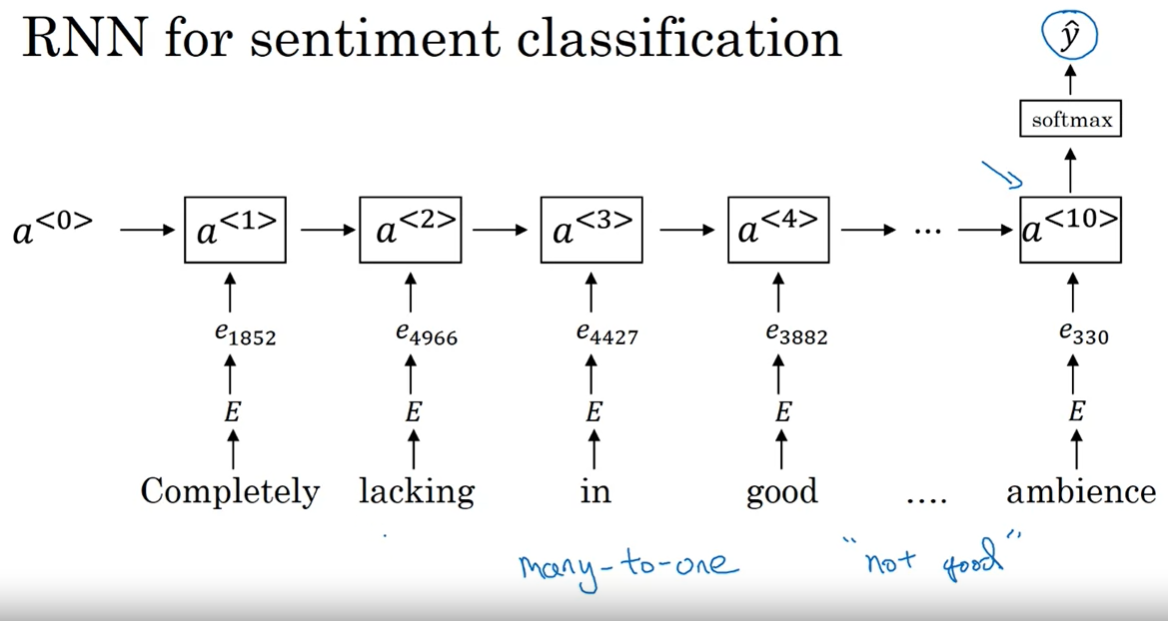


* For each word in our sentence , we will extract it’s word embedding representation using the E ( which has over 100B words ) , and then we sum the word’s vectors , we take their average and pass it to softmax unit to do a multi-classification problem ( 1 Star , 2 Star , …or 5 Stars )

#### 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

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



* 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 ew instead of the one-hot vector Ow
* 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

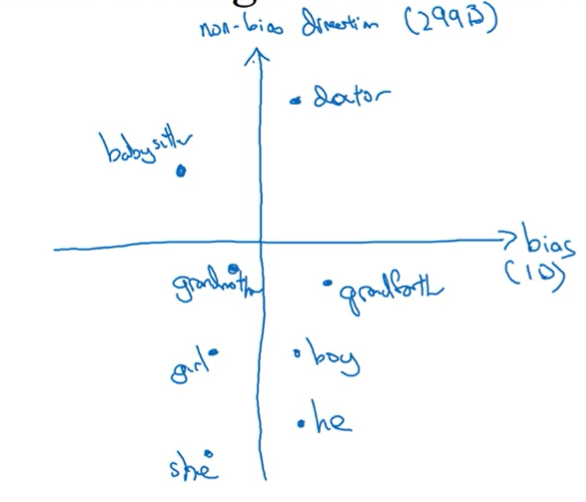
Since our training set is little and cannot handle all the vocabulary in the language, We can for example have in our model good words like “good”, “nice” , “delicious” and bad ones like “horrible” , “bad” , “disgusting” , but a word like “ magnificent” didn’t appear in our labeled training set . So 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

## Debiasing word embeddings:

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

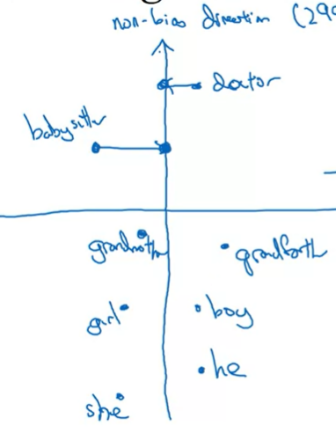
“Babysitting is a work for the grandmother more than to the grandfather”

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



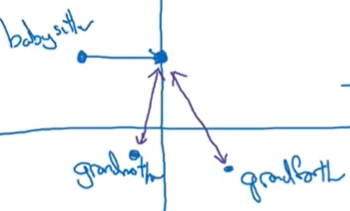
In our society, we believe that there isn’t any distinguish between the people to get opportunities or judging them by their race , their gender or their age …

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