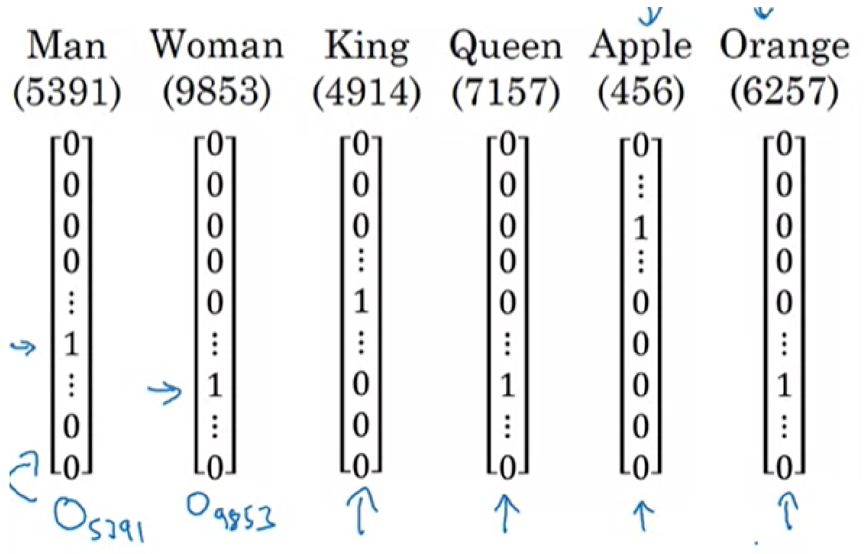
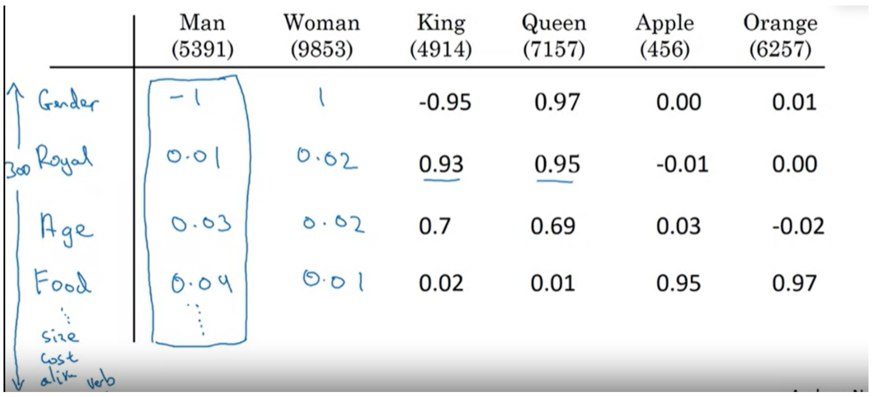
dl.ai_Word_embeddings

1)Using Word embeddings



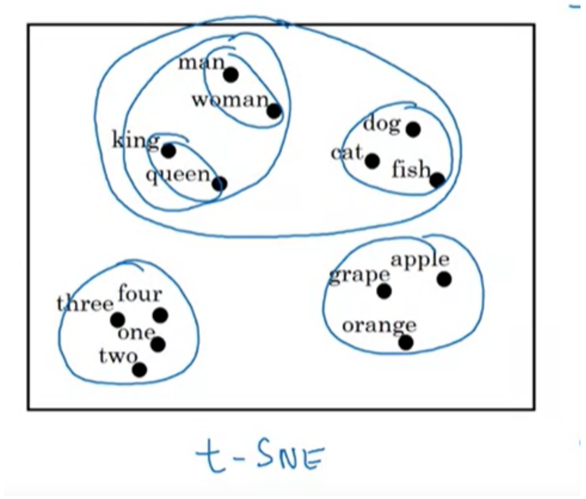
We earlier used the one hot encoded vector of a dictionary containing 10000 words to represent each word in the sequence. Lets say we have trained our model to locate the named word in the sentence “Barath sells marijuana” but during training our model encounters the sentence “Barath sells heroin”. According to one-hot encoded vector heroin has the same relation to marijuana as marijuana has with candy. They all have zero similarity as dot product of two one-hot encoded vectors is zero.

What if instead of 10000 dimension one hot encoded vectors we use 300 dimension feature vector were each word can have a value between -1 and +1.



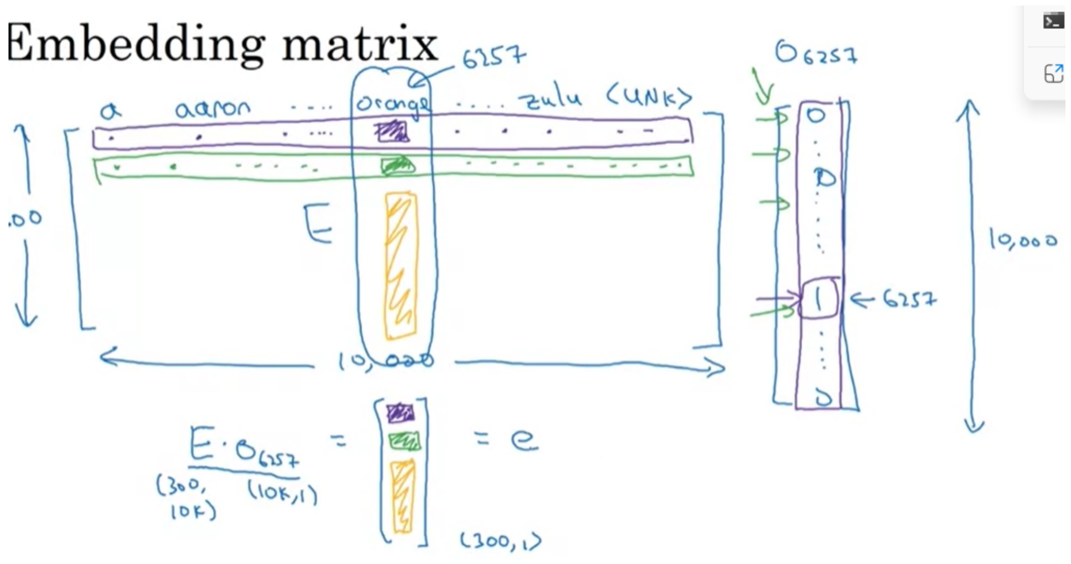
	Man (5391)	Woman (9853)	King (4914)	Queen (7157)	Apple (456)	Orange (6257)
Gender	-1	1	-0.95	0.97	0.00	0.01
300 Royal	0.01	0.02	0.93	0.95	-0.01	0.00
Age	0.03	0.02	0.7	0.69	0.03	-0.02
Food	0.04	0.01	0.02	0.01	0.95	0.97
⋮	⋮					
size						
cost						
alive						
verb						

We can further take this 300 dimensional vector and embed this into a 2dimensional space using t-SNE algorithm. We can see that the similar words are closer.

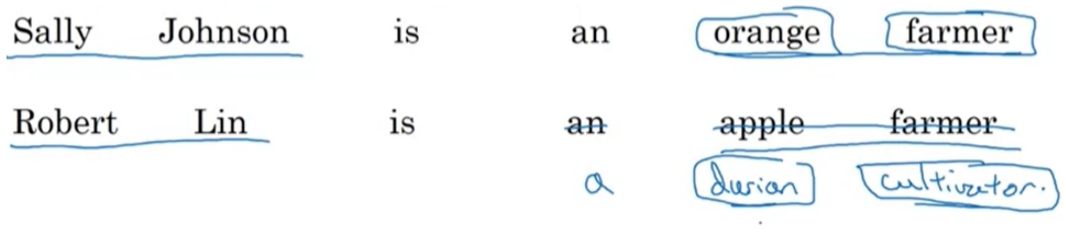


a) Embedding matrix and Transfer Learning through it

Lets say we have a sentence like below and we have to identify the named words in it we first pass all the words indivisually into a embedding matrix(multiply) and get the word embeddings out of it.



Now instead of passing O_i as the input for each unit of RNN(x<t>) we pass e_i. So our pipeline consists of two part the embedding matrix and the RNN. The embedding that we use is completely independent of the RNN,LSTM or GRU we are gonna use.


Sally Johnson is an orange farmer
Robert Lin is an apple farmer
a durian cultivator

The model was trained on the first sentence which had the words orange farmer it was able to generalize to apple farmer because of the word embeddings(10000 words) but we still cant generalize to durian cultivator because it was not present in our vocabulary. We can solve this by using a larger pretrained word embedding,

The larger embedding has been trained on a task with 1 billion to 100 billion words by using this on a task whose training set contains only 10000 words we are able generalize to words outside the training set. This is a form of transfer learning and we can finetune tune the pretrained embeddings to our new data if we want.

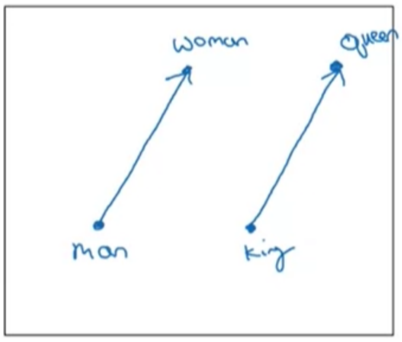
b) Properties of Word embeddings

In the embedding matrix we can observe that opposite words tens to have a difference of -2 in one feature and 0 in other features.

$$\underline{e_{man}} - \underline{e_{woman}} \approx \begin{bmatrix} -2 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

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

This property can be used to find analogous words if man → woman what is king→?



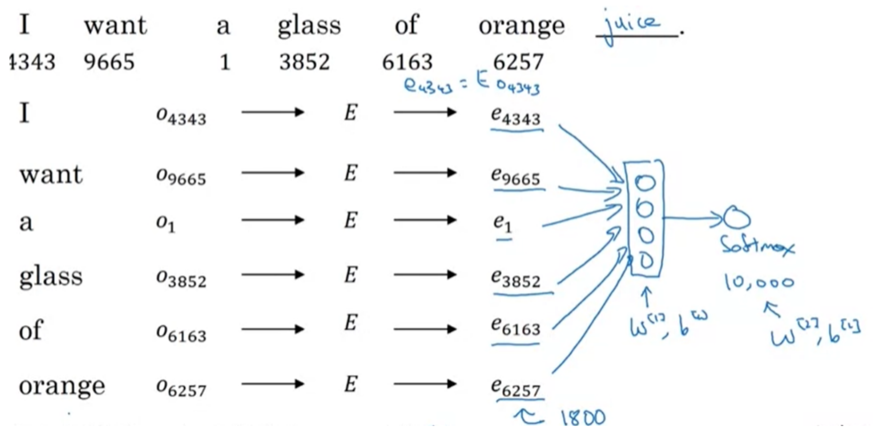
We simply need to find the vector which is most similar to e_king - e_man + e_woman. We may have not have an exact matching word always .The parallel mapping given in the left is for 300 dimensional mapping not 2d mapping. We define similarity by cosine similarity

$$\frac{u^T v}{\|u\|_2 \|v\|_2}$$

2) Learning word embedding

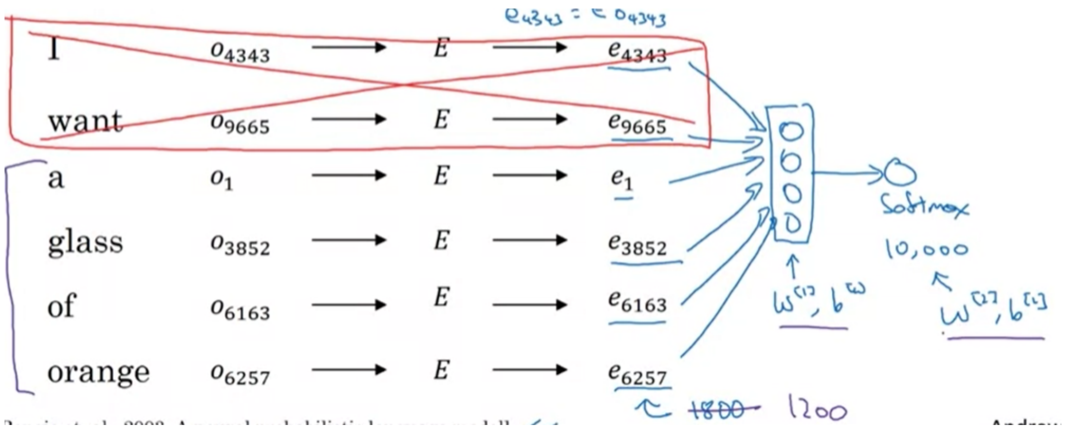
a)Simple NLM with softmax

We have seen how to use word embeddings but how do we find the embedding matrix?. We can learn the word embeddings through a natural language model. Lets say we have to predict the next word in a sequence by simply applying a softmax on a concatenation of all the words before.



I want a glass of orange juice.
4343 9665 1 3852 6163 6257
$e_{4343} = E\,o_{4343}$
I o_{4343} → E → e_{4343}
want o_{9665} → E → e_{9665}
a o_1 → E → e_1
glass o_{3852} → E → e_{3852}
of o_{6163} → E → e_{6163}
orange o_{6257} → E → e_{6257}
Softmax
10,000
$W^{[1]}, b^{[1]}$
$W^{[2]}, b^{[2]}$
1800

We would first multiply each one hot encoded word to a embedding matrix filled with parameters(it is not fixed) we will pass the embeddings into a sigmoid and get a 10000 dimensional vector which represents the target. We may just use the last 4 words always to have a fixed size vector before softmax.



$e_{4343} = E\,o_{4343}$
I o_{4343} → E → e_{4343}
want o_{9665} → E → e_{9665}
a o_1 → E → e_1
glass o_{3852} → E → e_{3852}
of o_{6163} → E → e_{6163}
orange o_{6257} → E → e_{6257}
Softmax
10,000
$W^{[1]}, b^{[1]}$
$W^{[2]}, b^{[2]}$
~~1800~~ 1200

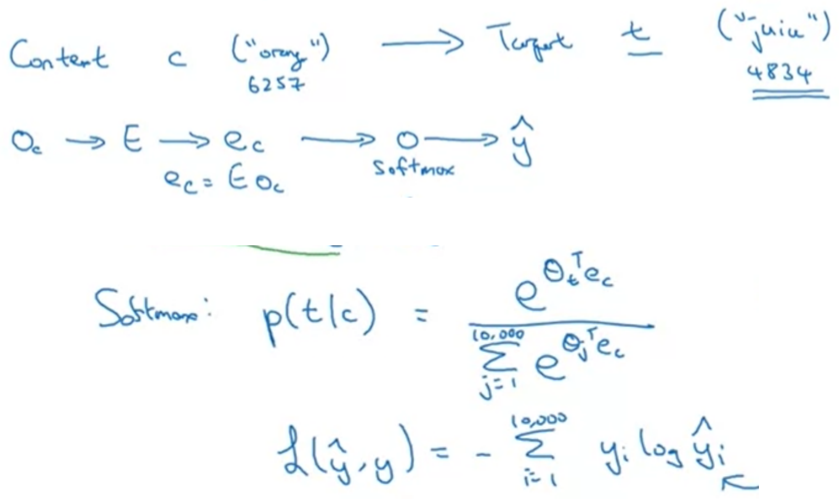
The parameters in the embedding matrix are learned by gradient descent during back propagation. Because the algorithm has to fit 10000 different words into a 300 dimension vector it will automatically end up group similar things together.

Our task here is not to improve our prediction of the next word but to improve the word embeddings. We try various problem tasks to learn this embeddings.

Hence we can try other type of contexts rather than just last 4 word before like 4 word each to left and right,1 word each to left and right or just a single random word.(skip gram)

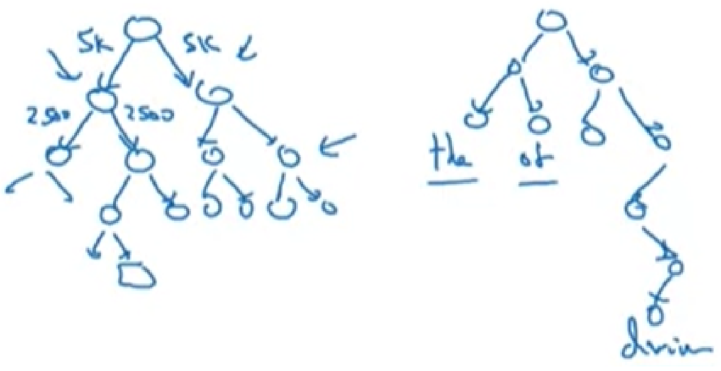
b)Word2vec-Skip gram

Word2vec is one of the alternative algorithms it is also called the skip-gram model because it is taking as input one word like orange and then trying to predict some word(target) skipping a few words from the left or the right side. To predict what comes little bit before or little bit after the context words. We sample a context word, look around a window of say, +-10 words, and pick a target word.



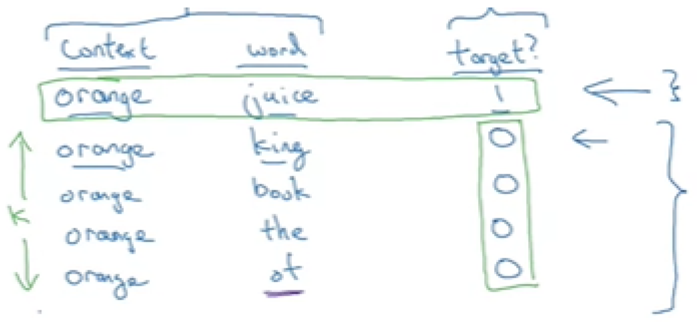
Content c ("orange") 6257 → Target t ("juice") 4834
$O_c \rightarrow E \rightarrow e_c \rightarrow O \rightarrow \hat{y}$
$e_c = E O_c$
softmax
Softmax: $p(t|c) = \frac{e^{\theta_t^T e_c}}{\sum_{j=1}^{10,000} e^{\theta_j^T e_c}}$
$\mathcal{L}(\hat{y}, y) = -\sum_{i=1}^{10,000} y_i \log \hat{y}_i$

The issue with skip gram is to find soft-max we have to sum over all 10000 possible outputs hence increasing computation cost. We can solve this by using heuristics.

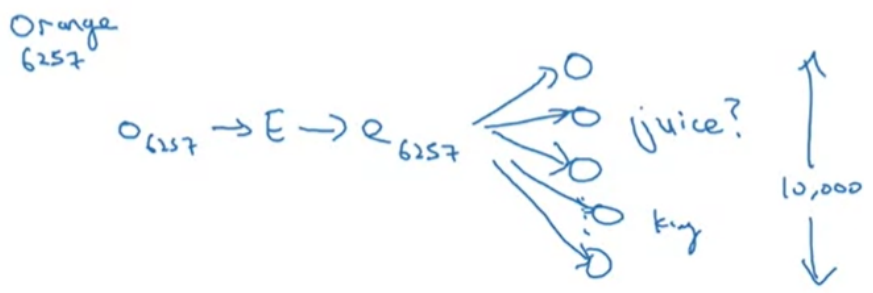


Another way to solve this is by using Negative sampling. We wont be trying to predict a target given a context word instead the problem task is, given a pair of words like orange and juice, we're going to predict, is this a context target pair?

We generate a positive pair exactly how we generated it in the previous example, sample context word, look around a window of say, +-10 words, and pick a target word like orange and juice. Then to generate the negative examples, we are going to take the same context word and then just pick a word at random from the dictionary like king. We take k negative samples. We have a label indicating whether the pair is true.



Instead of modelling a softmax with 10000 outputs we would simply be modelling 10000 or even lesser(that is k)different sigmoid binary classification.



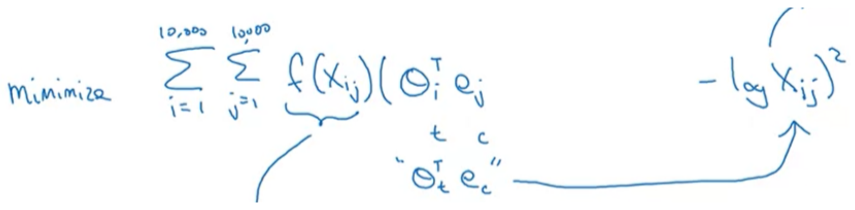
We would find e_orange just like previous case but we wont pass it through an softmax but instead pass it through k sigmoid operations with different possible target words.

How do we select the negative examples we do it by randomly sampling a word with probability based on the frequency with which it appears generally in the english language.

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

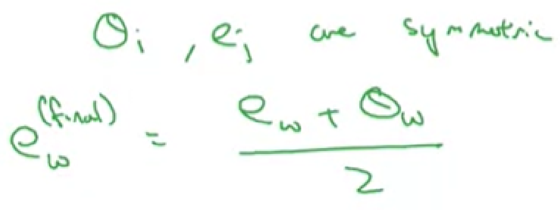
c) GloVe embeddings

GloVe word vectors use a different problem task to learn embeddings. Our problem task is to predict how related two words are by measuring the number of times they appear in the context of each other. Xij represents the actual number of times the context j appears for the target i if our prediction is closer to Xij our loss must be lesser.



We added a weighting term f(Xij) to ensure that the when Xij is zero the loss doesn’t become undefined.

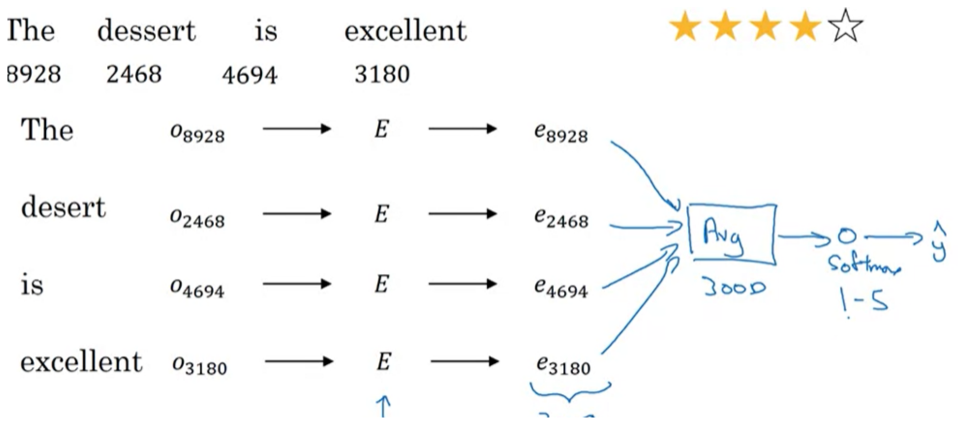
θ_i, e_j are symmetrical for this task so we finally take average of these two values to get the embedding matrix.



3) Other stuff

Why RNNS instead of soft-max?

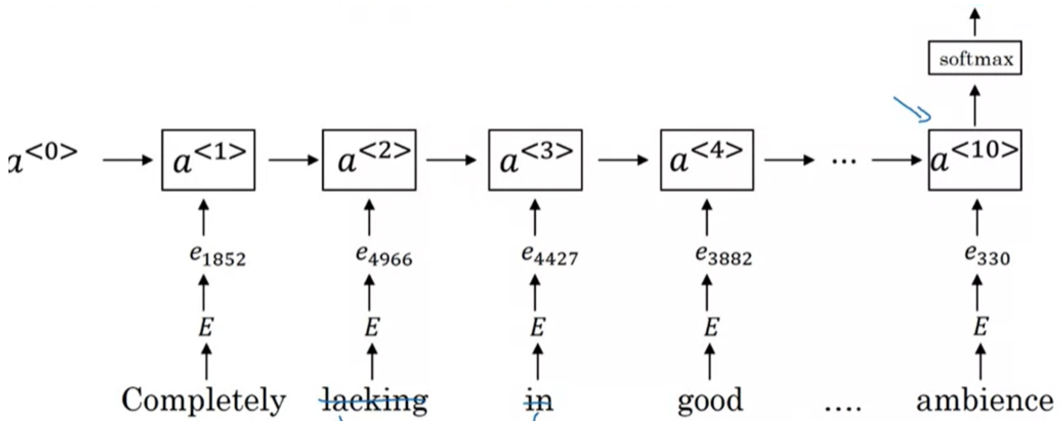
Simply using a average or sum of embeddings can’t be used for a task like sentiment classification.



But if positive words occur more than negative words in the sentence the classifier will then give a positive result even if the sentence on its whole was negative.

> "Completely lacking in good taste, good service, and good ambience."

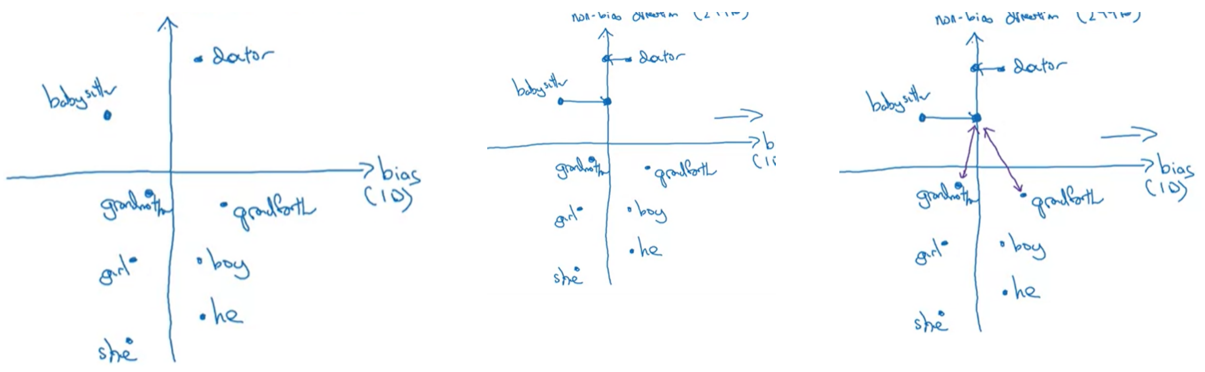
We can hence use a simple many to one RNN with vector embedding



Debiasing Embeddings

We saw that vector embeddings allow us to find analogies between words but sometimes these analogies might reveal intrinsic biases in the embeddings. Example man→ computer programmer as woman→ homemaker

We would need to debias vector embeddings in order to remove such biases by a three step process.



1)Identify the bias direction using heuristics . For example we can find the bias direction of gender by simply subtracting definitional words e_he - e_she, e_male - e_female etc and then take average. In practical cases we use a method called SVU singular value decomposition as the bias direction we have , might not correspond to a feature(axis) in the 300 dimensional embedding. It is similar to principal component analysis. The 299dimensional axis excluding the 1d bias axis is said to be our non bias axis.

2) We neutralize non definitional words by projecting them onto the non bias axis. That is whatever component they had along bias direction is made zero. Since the non bias axis (which contains 299 features) is perpendicular no other information is lost

3) finally we equalize the pairs of definitional words so that they are equidistant from the axis. Lets say we for the gender axis we have he and she, the word he and she must be equidistant from the non bias axis. But this may not be the case as the non bias axis was decided by us. So we take a word like doctor which lies on the non bias axis after step 2 and move the word he and she such that they are equidistant from doctor.

We could identify definitional words by using a linear classifier. We identify their pairs using heuristics.