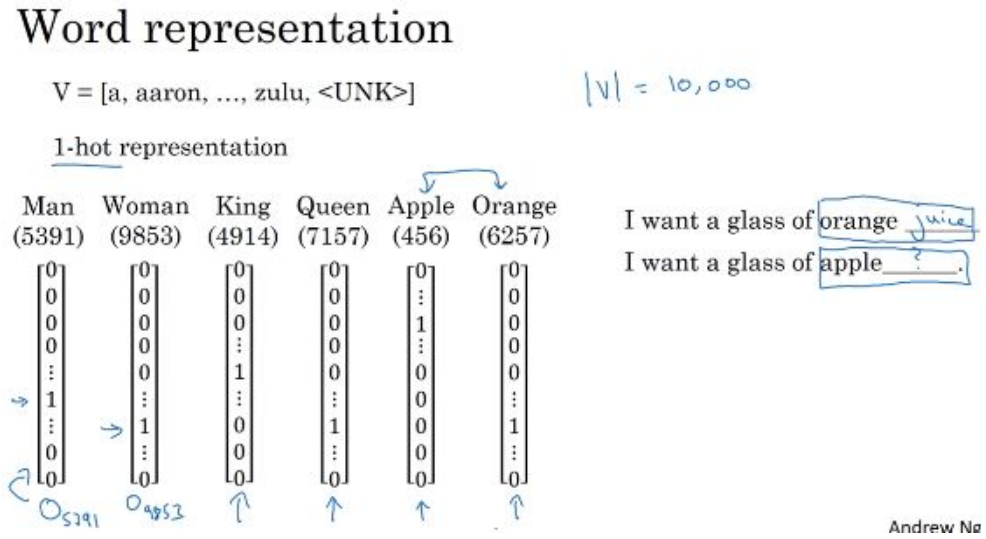
Andrew Ng's Recurrent Neural Networks

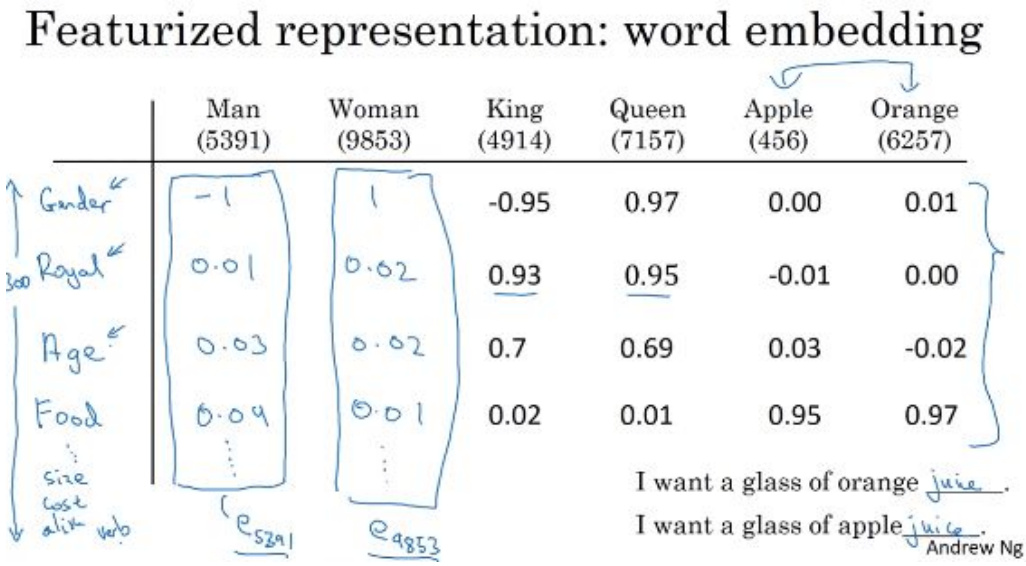
Week 2: Natural Language Processing & Word Embeddings

Word Representation

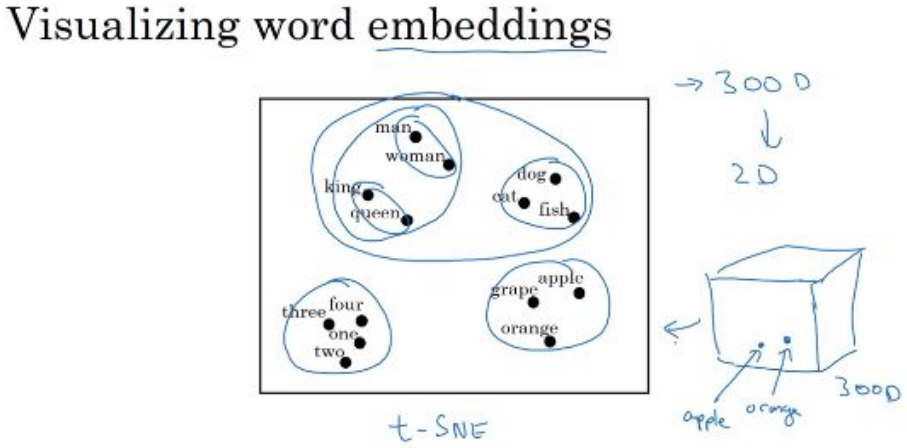
- We start this week by looking at word representation. Last week we had a vector of about 10,000 words and we represented each word as a 1-hot vector using its position in the dictionary. For example, man is in position 5,391 so they 1-hot vector for this words will have all 0's except for a 1 in position 5,391. We do the same process for other words. There is a problem with using this 1-hot representation: it treats each word as a thing and there is no way to generalize across words. An example would be the two sentences below. The algorithm cannot know that orange and apple have a relationship because the product of any two one-hot vectors is zero. So we need to find a better representation.



- Our solution is to use a featurized representation where we take the words in our vocab and create a feature vector that is filled out for every word in the dictionary. The column on the left shows some of the features: gender, royal, age, food, size, etc. Each word will then have a 300-feature vector as shown below (known as word embedding). This will allow our algorithm to know that say, apple and orange have some sort of relationship and will end up helping us to associate the word 'juice' for both words. Why? Notice how the representations of apple and orange, their vectors, have similar values for many features so the algorithm will learn they are similar/related.



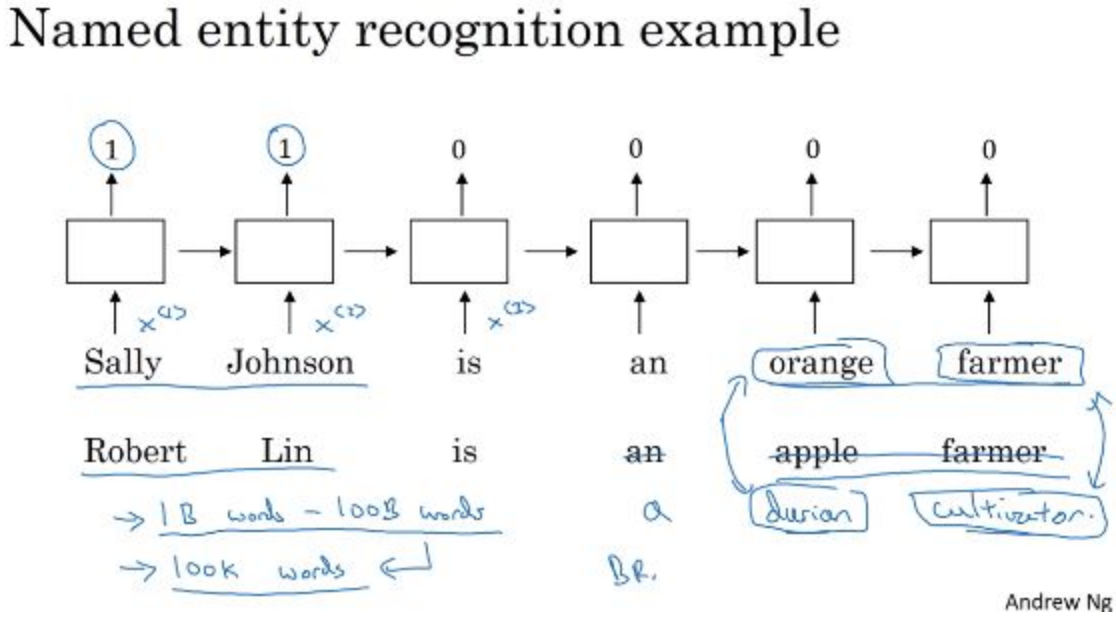
- One popular thing to do is to take the 300-dimensional embedding for each word and plot them in a 2d space as the one shown below. We can see how similar words are grouped together: numbers, fruits, animals, humans, etc. Word embeddings is one of the most important ideas in NLP. In the next session we will see how to use them.



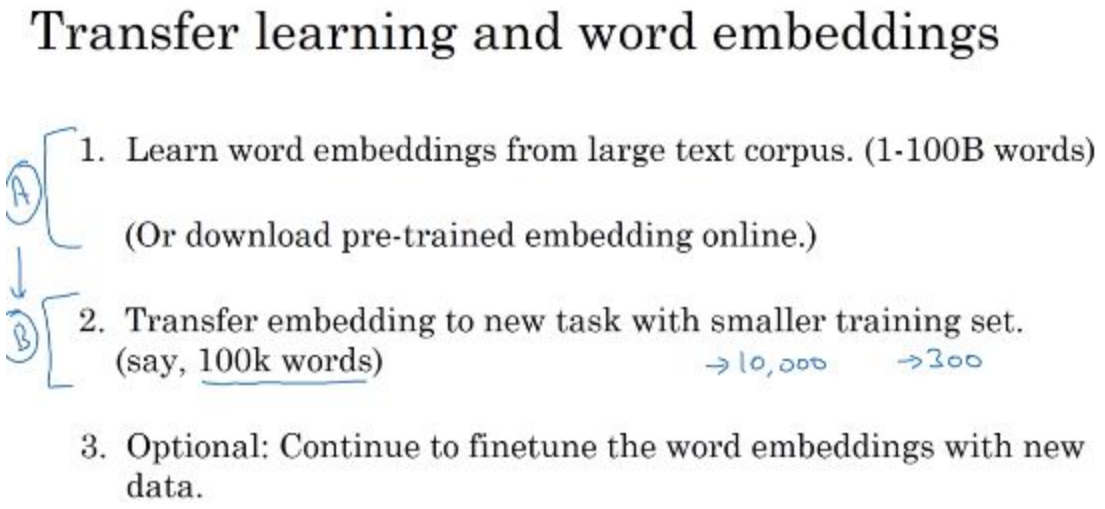
Using word embeddings

- In this session we are going to see how we can use word embeddings. Let's start with an example: below we have a sentence where we need to recognize that 'Sally Johnson' is a name and one way to do that is to look at the rest of the sentence and recognize that 'orange farmer' refers to 'Sally Johnson'. Word embeddings allow us to do thi. If we take another sentence, like the 2nd one, we hopefully know that orange and apple are related and therefore 'apple farmer' refers to 'Robert Lin'. In this day and age it is common to be able to retrieve huge datasets off the internet (1B to 100B words) and use it to learn things like 'durain is a fruit and is related to orange and apple'. We can then take this learnings and apply it to a task where we originally had a much smaller dataset (say, 100K words). Finally, just a reminder for name

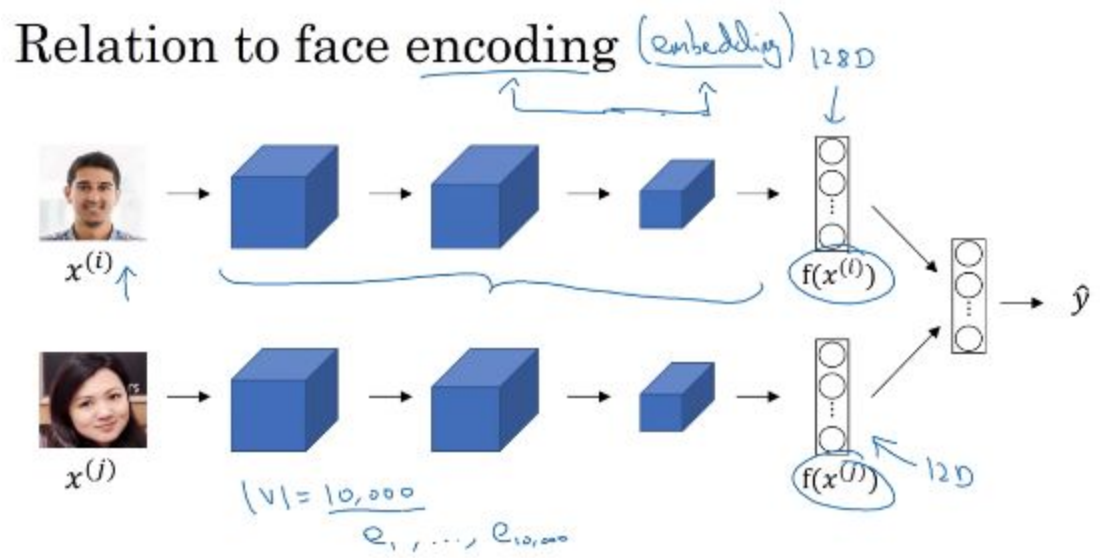
recognition we need to use a bidirectional RNN and not a unidirectional one like the one we drew below.



- The process to do transfer learning is as follows: we learn embeddings from the large dataset we downloaded off the internet (or we just download the pre-trained embeddings). Then we transfer the embedding to the new task that has a smaller data set. Finally, optionally, we continue to finetune the word embeddings with data from the current task. The last step only happens in practice if the task has a large dataset. However, as we can see, transfer learning is most useful when we have a ton of data on task A and we transfer the knowledge to a task B that has a small amount of data.



- Finally, there is a relationship between face encoding and embedding. The deep learning community uses the terms encoding and embedding interchangeably. In the siamese network below the vectors $f(x^{(i)})$ and $f(x^{(j)})$ are also known as embeddings.

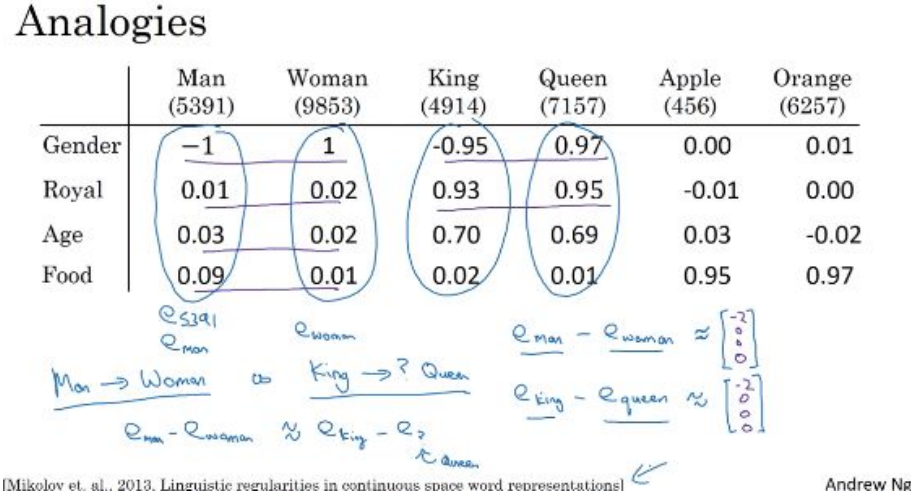


Taigman et. al., 2014. DeepFace: Closing the gap to human level performance]

Andrew Ng

Properties of word embeddings

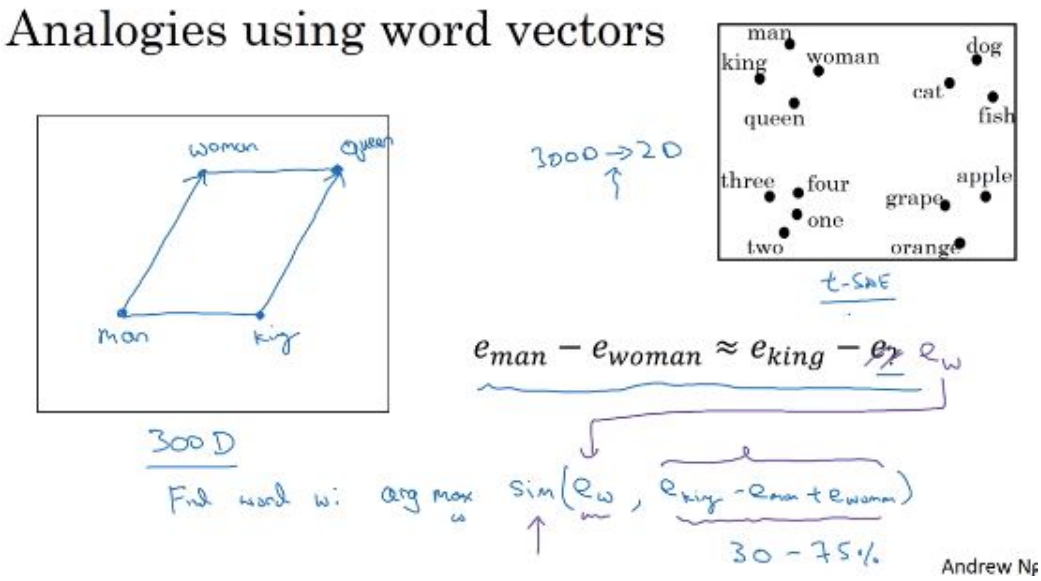
- Word embeddings can help with analogy reasoning. Let's see how it is done. Below you will see the word embeddings for a few words. Given the questions Man-> Woman as King -> ?? could we answer it using the word embeddings? Turns out we can: What we do is we subtract Man's word embedding (ϱ_{man}) from Woman's word embedding (ϱ_{woman}) and end up with a vector [-2, 0, 0, 0]. We then take King's word embedding (ϱ_{king}) and subtract the remaining embeddings from it. When we do King's word embedding - Queen's word embedding (ϱ_{queen}) we end up with vector [-2, 0, 0, 0] and that is our answer.



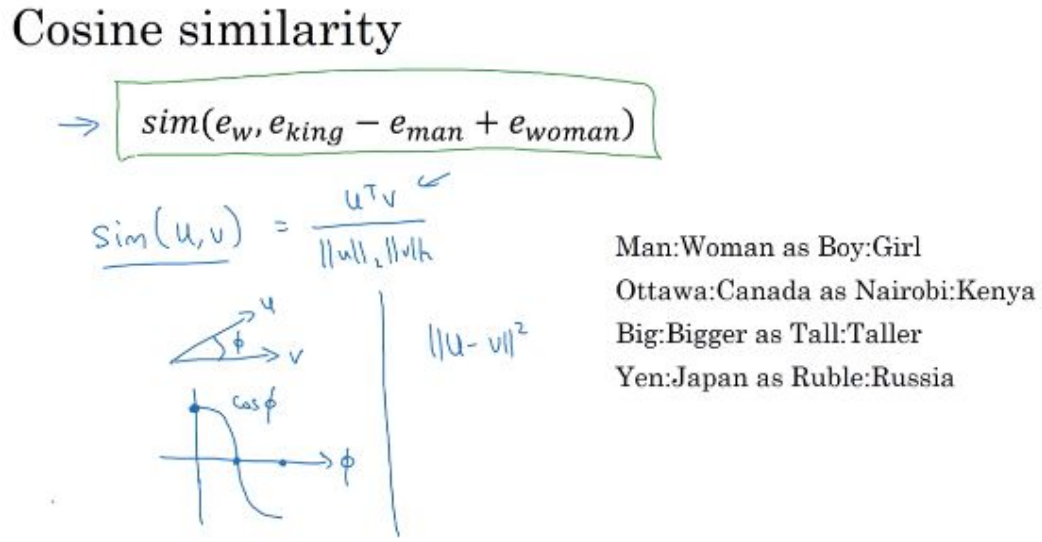
[Mikolov et. al., 2013. Linguistic regularities in continuous space word representations]

Andrew Ng

- Let's formalize this in the following manner: In a plot words are depicted as shown below on the left. Given the equation we had before, $e_{man} - e_{woman} = e_{king} - e_{??}$. We are going to substitute $e_{??}$ with e_w. We are looking for the most similar word so our similarity equation can be written as $(e_w, e_{king} - e_{man} + e_{woman})$.

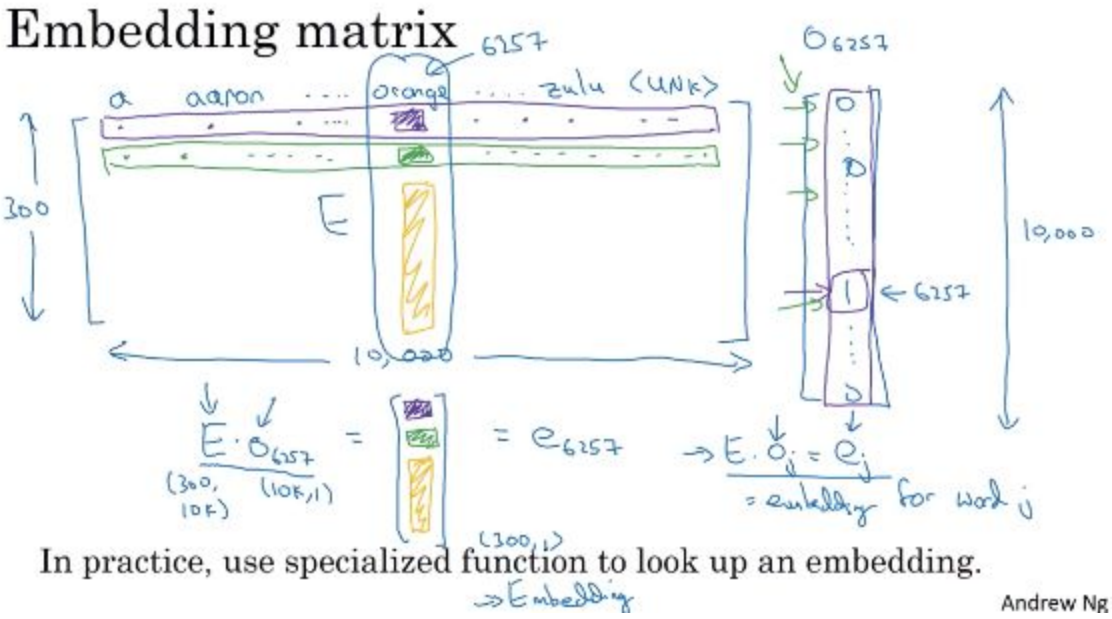


- The most used similarity function is cosine similarity which is expressed as $sim(u, v) = u^T * v/(||u||_L ||v||_L)$. We can also use euclidean distance $(||u - v||^2)$ as a similarity function although strictly speaking it is a measure of dissimilarity instead of a measure of similarity. We still see cosine similarity used more often. Below are also some other examples of the analogies that word embeddings can resolve.



Embedding Matrix

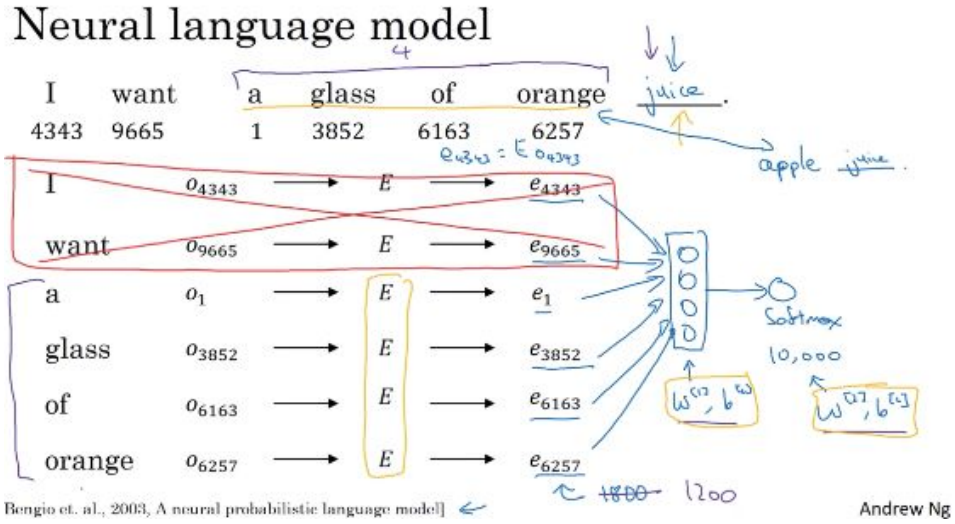
- How do we learn a word embedding? We actually learn an embedding matrix. Supposed our dictionary has 10,000 words. We then have a 300x10,000 matrix and the columns of the matrix are the embeddings we learn for words in our vocabulary. Assume orange is word 6257 so the one-hot vector for orange is 0_{6257} and it has zeros everywhere except in position 6257 where it has a 1. If we call the embedding matrix E we can then multiply E (dimensions 300,10,000) * $0_{6257}(dimensions\ 10,000x1) = \varrho_{6257}(dimensions\ 300,1)$ and end up for the embedding for word 6257. Generalizing, for word j $E * 0_j = \varrho_j$, which is the embedding for the word j. In practice it's not very efficient to do this what matrix multiplication so we use a specialized function. For example, in keras we use an embedding layer.



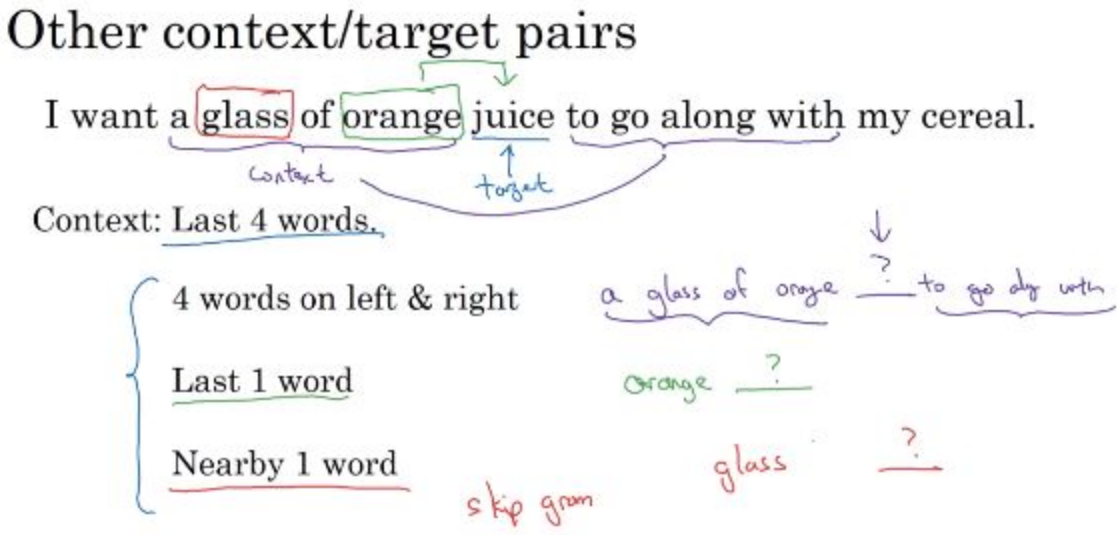
Learning Word Embeddings

- Let's now look at how we can learn word embeddings. We start with building a language model using a neural network as shown below. We have 6 words in the sentence below and our neural network will try to predict the last word in the sentence. We start by taking the one-hot vector of each of the words in the sentence and multiplying it by the E matrix and we end up with the ϱ vectors for each word. Each of these vectors is of length 300. We then feed these vectors into a neural network layer and finally to a softmax activation which will give us the prediction for the last word in the sentence. The neural network layer has parameters $w^{[1]} and\ b^{[1]}$ and the softmax activation has parameters $w^{[2]} and\ b^{[2]}$. The input for the neural network layer is all the ϱ vectors so that's 1,800 (300 * 6 words). In practice, we can use a hyperparameter to only look back at say 4 words instead of the 6 we have. If

we do that our input to the neural network is only 4 vectors, each of length 300, for a total of 1,200 length input.



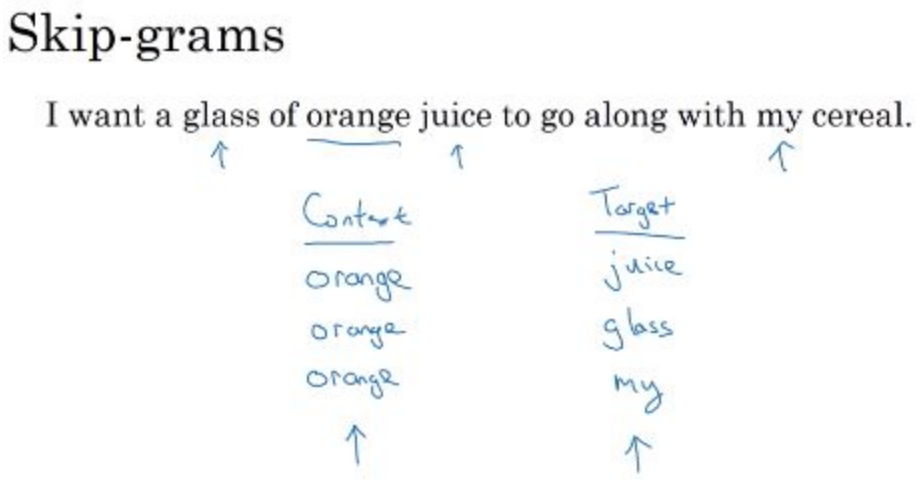
- Let's formalize this a bit more. The act of choosing which words to feed the neural network is known as context. In the example above we used a context of the last w words but we can use other contexts. Some examples are:
  - 4 words on the left and right
  - Last 1 word
  - Nearby 1 word (also known as skip gram)



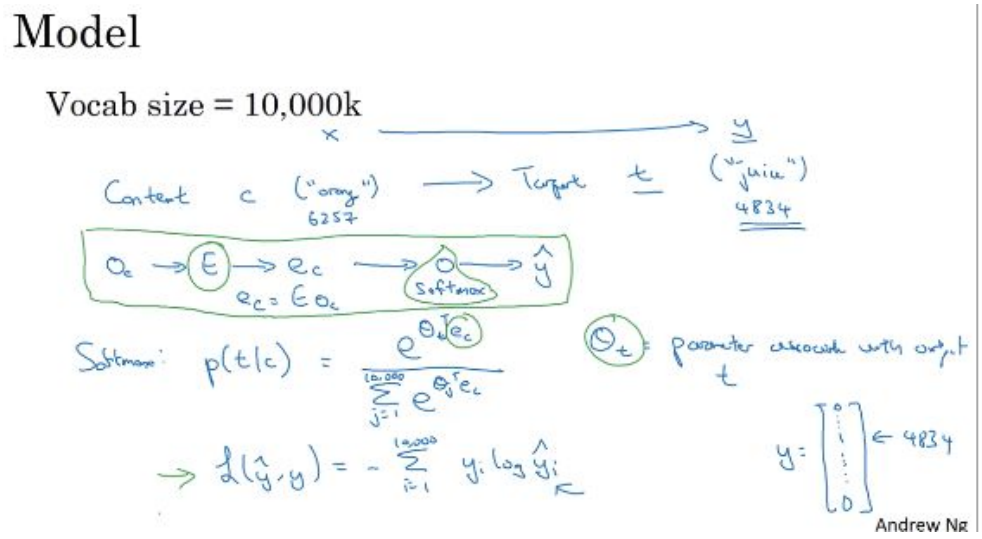
Researchers found out that if you want to build a language model it is effective to use the last few words as a context but for learning word embeddings we can use different contexts (like the ones mentioned above) and they will give us very good results.

Word2Vec

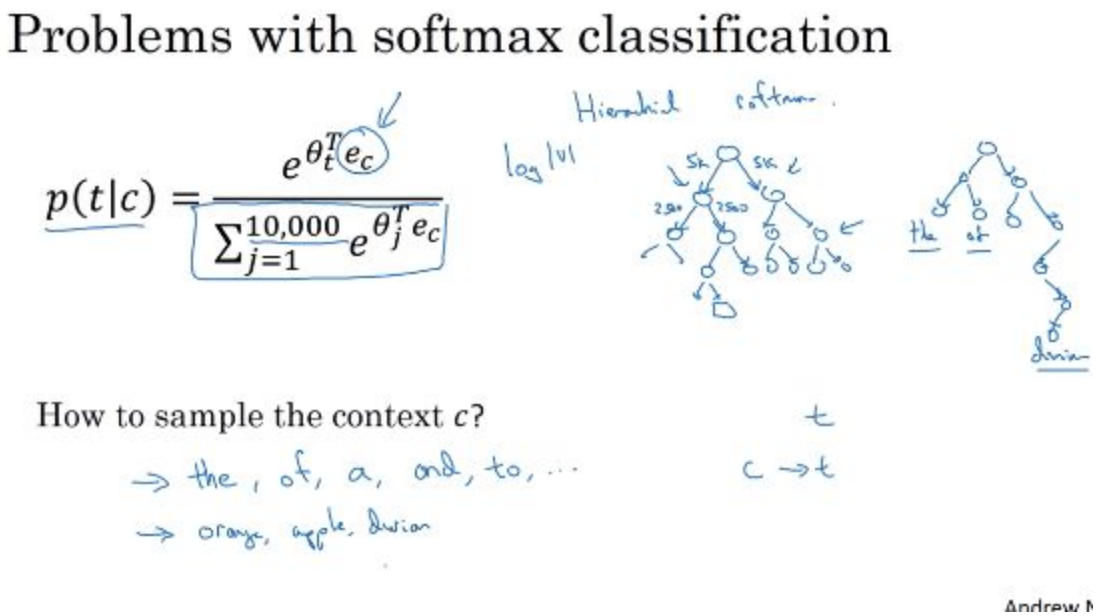
- In this session we are going to go over the Word2Vec algorithm which is a simpler and more efficient way of learning word embeddings. Suppose we have the sentence below and what we are going to do is randomly pick the context and the target word. It is called a skip gram because once we pick the context, the target word is picked by a selection of +/- number of words, meaning we are skipping a few words. Below are some examples of each.



- Now let's go over the details of the model. Given our usual 10,000 word vocab what we are trying to do is take context c ('orange', position 6257) and map it to a target t ('juice', position 4834). The process we do is similar to what we have done before: we take the one-hot vector 0_c, multiply it by the matrix E and we get e_c. We feed that to a softmax activation which will input our prediction. It is expressed as follows: $0_c \rightarrow E - e_c \rightarrow softmax\ activation \rightarrow yhat$. The softmax equation is $p(t|c) = e^{\theta_t^T e_c} / \sum_{i=1}^{10,000} e^{\theta_j^T e_c}$ and the loss function is shown below.

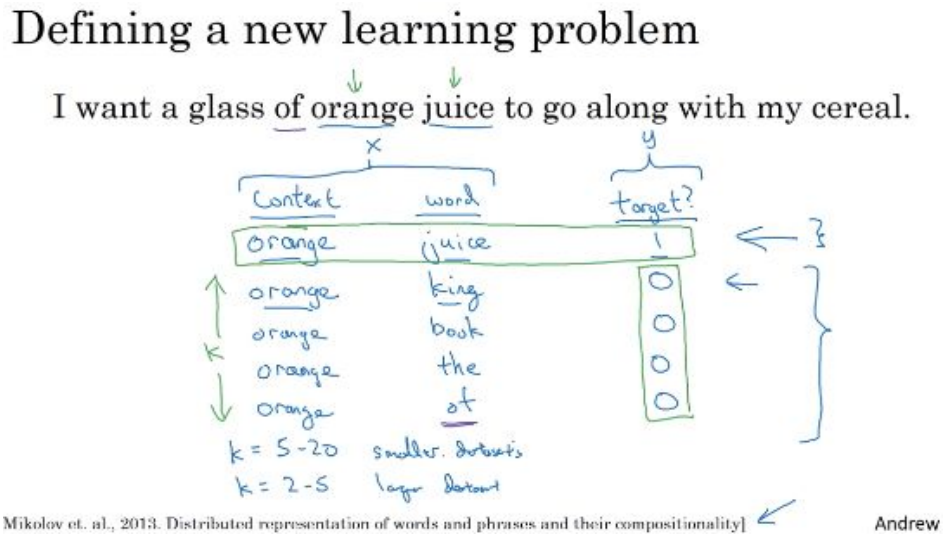


- There are some problems with using a softmax activation for this kind of model. One of the problems is that it is computationally expensive to calculate probability p because we have to sum over the whole vocabulary. Some alternatives to this is to use hierarchical softmax where we use a binary tree to detect where the target word is and we end up with a leaf that is a binding classifier. In practice, the binary tree used is not perfectly balanced. Last thing to go over is to decide how to sample the context c. We have just seen how to choose target t but how about context c? You will see some words that appear more frequently such as the, of, a, an, and, to, etc and some other words that do not appear very frequently such as orange, juice and durian. We don't want our training set to be dominated by the frequent words so we use different heuristics to help us balance out common and not-so-common words.

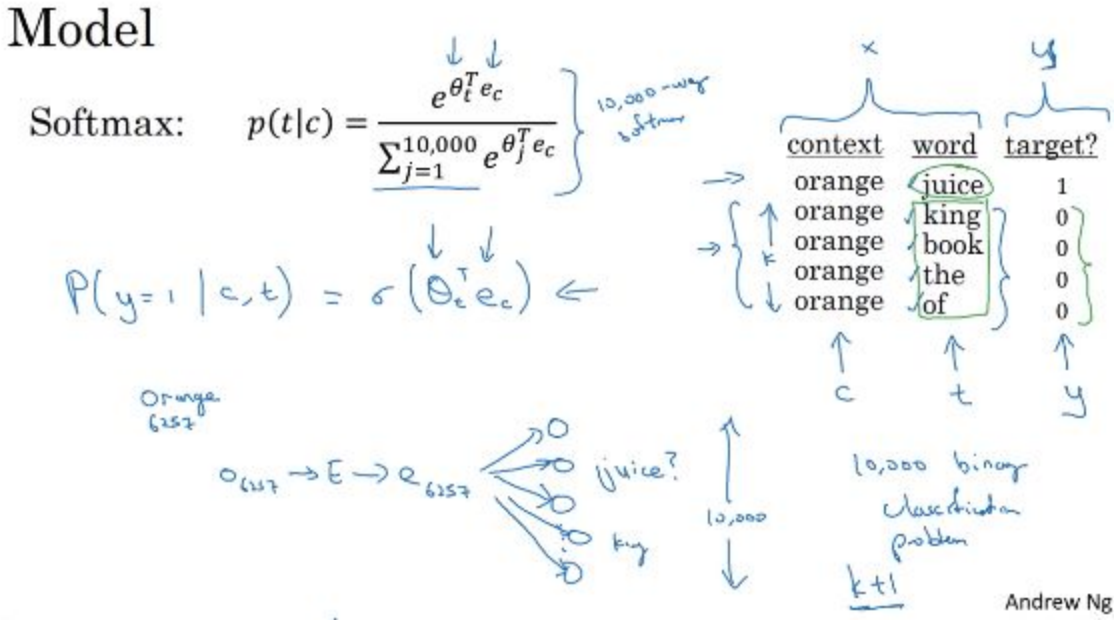


Negative Sampling

- In this session we see another way to create a supervised learning problem. Assuming we have a sentence like the one below, the first thing we do is we are going to create our training set. To do that our first sample is the context word and the actual target word (orange & juice in the example below) and we assign it a label of 1. We then take the same context word and using + or - x number of words chosen randomly we create more elements of our training set (orange book, orange king, orange of, etc.) and assign them a label of 0. Our word pairs are then X and our 1,0 labels are y and now we have a supervised learning problem. How many word pairs do we need to create? The guideline is 5-20 for small datasets and 2-5 for big datasets.

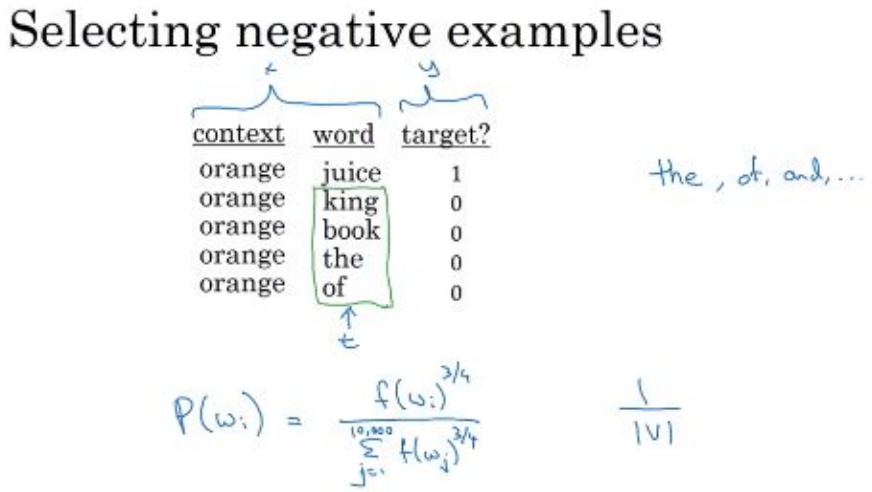


- Now we can build the model. Below we have the computationally expensive softmax model we learned in the previous session. Our new model is basically a bunch of logistic regressions. It is expressed as $P(t \mid c) = \sigma(\theta_t^T * \varrho_c)$. So we take the context word, orange in this case, and use its one-hot vector, O_{6256} in this case, multiply it by the E matrix and we end up with ϱ_{6257} which is a 10,000-length vector where each node is a binary classification problem, namely if the word in that node is the target word. So we now have a 10,000 binary classification problem which is much cheaper, computational speaking, than a 10,000 length softmax activation. This model is called negative sampling because we deliberately created a number of negative examples



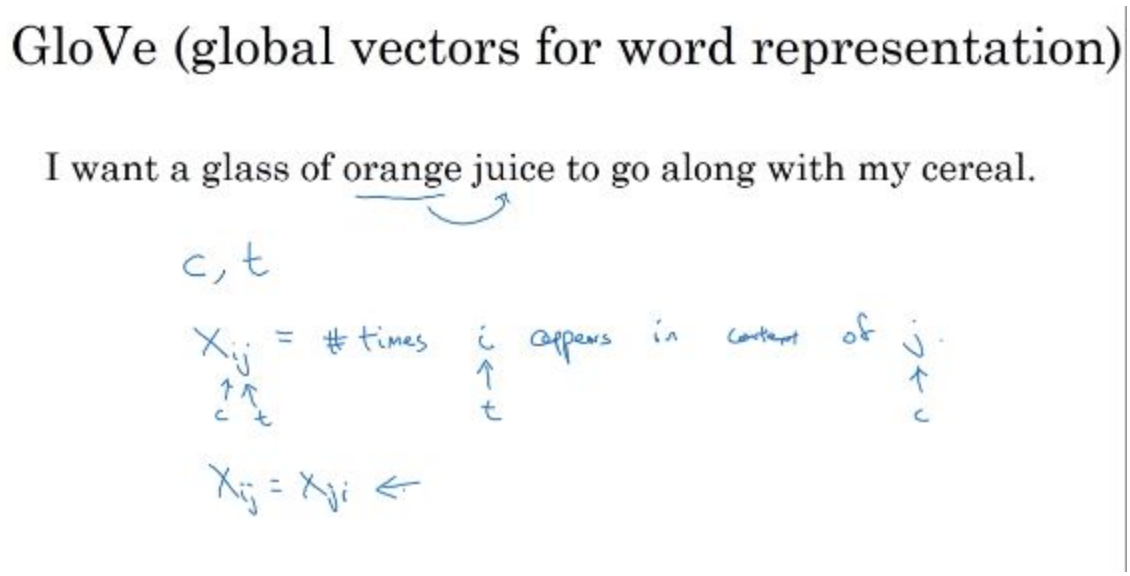
- One more detail: how do we choose the number of negative examples? The recommended formula is shown below. We can use the empirical frequency of

words or we can use 1/|V| which means we are sampling using the uniform distribution but both choices are not representative so we end up with the formula below.



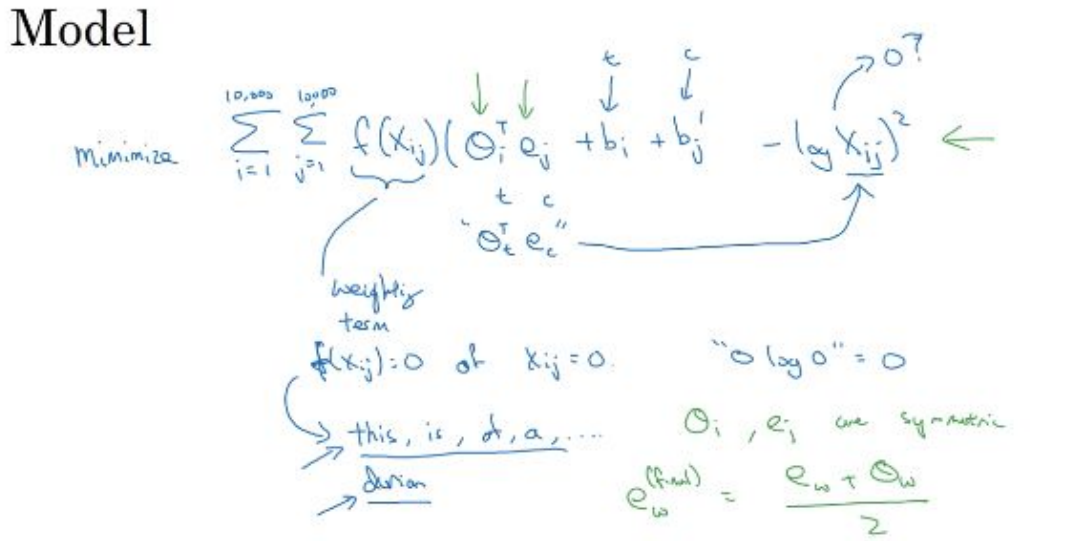
GloVe Word Vectors

- We are going to look at GloVe word vectors in this session. GloVe vectors have many fans because of its simplicity. Let's see how it works. GloVe stands for Global Vectors for word representation. If we have our context and target words, x_{ij} stands for the number of times j (t) appears in the context of i ©. In other words, it is a count of how many times the 2 words appear together (together mean +- 10 words apart).

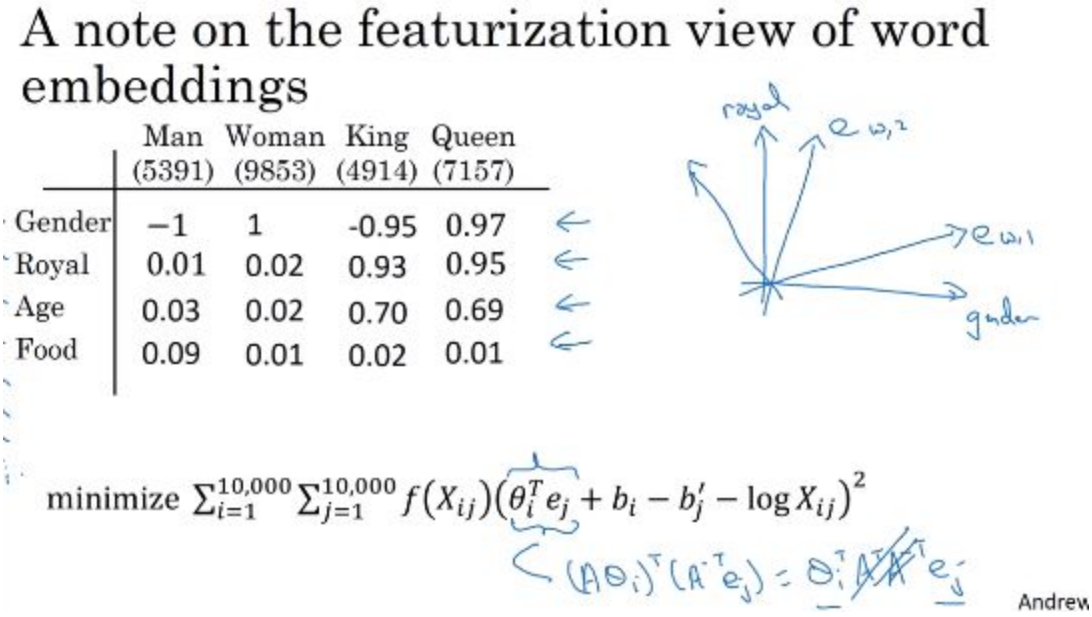


- Our model is then going to try to do the following: minimize the formula below where x_{ij} is the number of times i and j appear together so there will be times where it could be 0. If it is 0 then we add a weighting term f(x) and we assume that 0 * log0 is 0 and not infinite. This weighting term balances the value of the words in our dictionary. It

doesn't give too much weight to the words that appear infrequently (say, durian) and doesn't give too little weight to the words that appear all the time (say, the, a, an, etc.



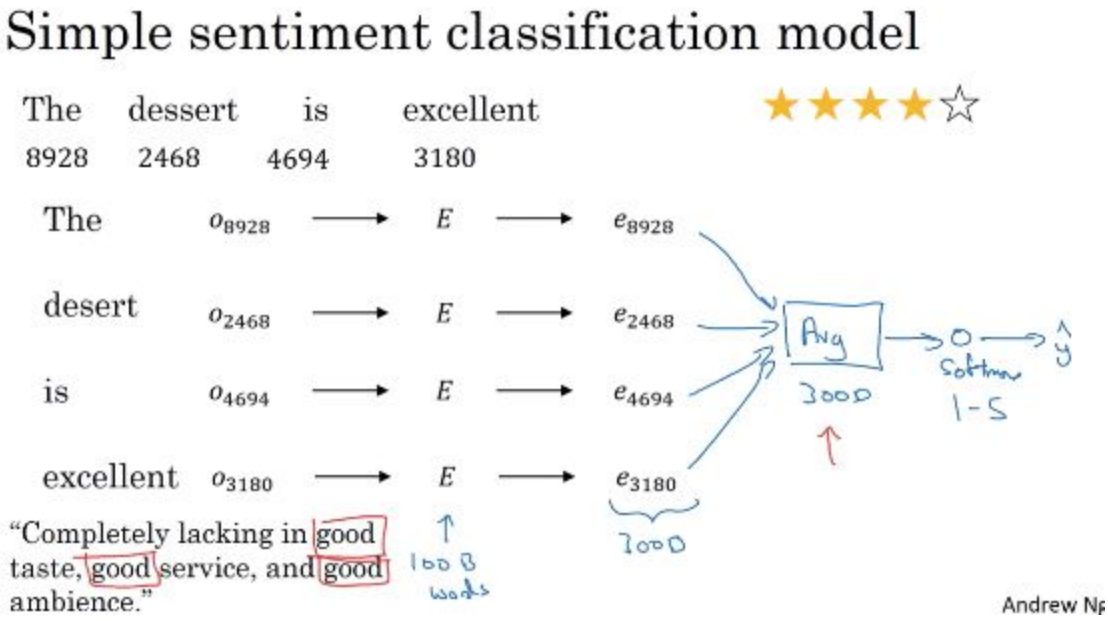
- A final note is that we cannot guarantee the word embeddings we learn through say, GloVe will be human interpretable using our initial word embedding motivation (shown below). That's the case because the learning algorithm might choose different axis to be the ones to represent features and they will be hard to understand.



Sentiment Classification

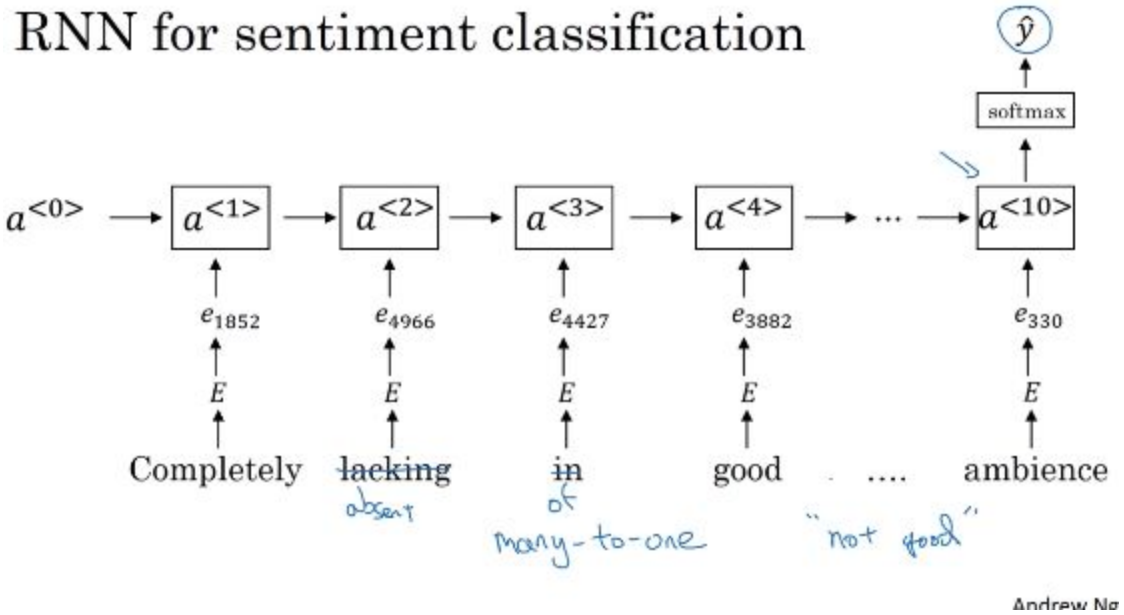
- Sentiment classification is one of the building blocks of NLP and used in many applications. It is the act of taking an input x (say a piece of text) and assign it an output y (say, a rating). Below are some examples. One of the challenges of this type of problem is we might not have a big training set, common sizes are 10,000 words to 100,000 words. Word embeddings can help us in this regard.

- Here's an example of what we mean. Let's take the example below. What we do is take the one-hot vector for each word, multiply it by the embedded matrix E and we end up with the embedding vector ϱ for each word. It will have a 300 dimension. We then average all the embedding vectors and we end up with an average vector of length 300. We then feed that to a softmax classifier that will output *yhat*. Notice that this model works for long texts since we always end up with an average of the embedded vectors.



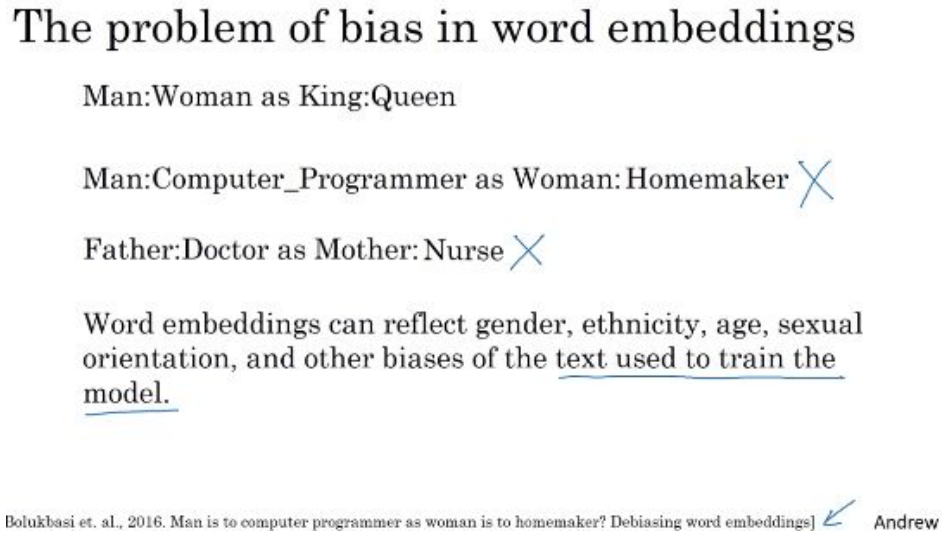
One problem with this model is the text shown in the bottom left corner above. It has the word ‘good’ many times so chances are our averaged embedding vector will have it and our classifier will think it is a good review when in reality it is a very bad review.

- To correct that we can use a better mode: a Recurrent Neural Network for sentiment classification. What we do in this case is similar as before: we compute the embedding vector for each word and feed it to a RNN whose last step would be to output yhat. This model works better than the previous model because our word embeddings can be trained on a much larger dataset and therefore will generalize better with some words that might not have been present in your training set.



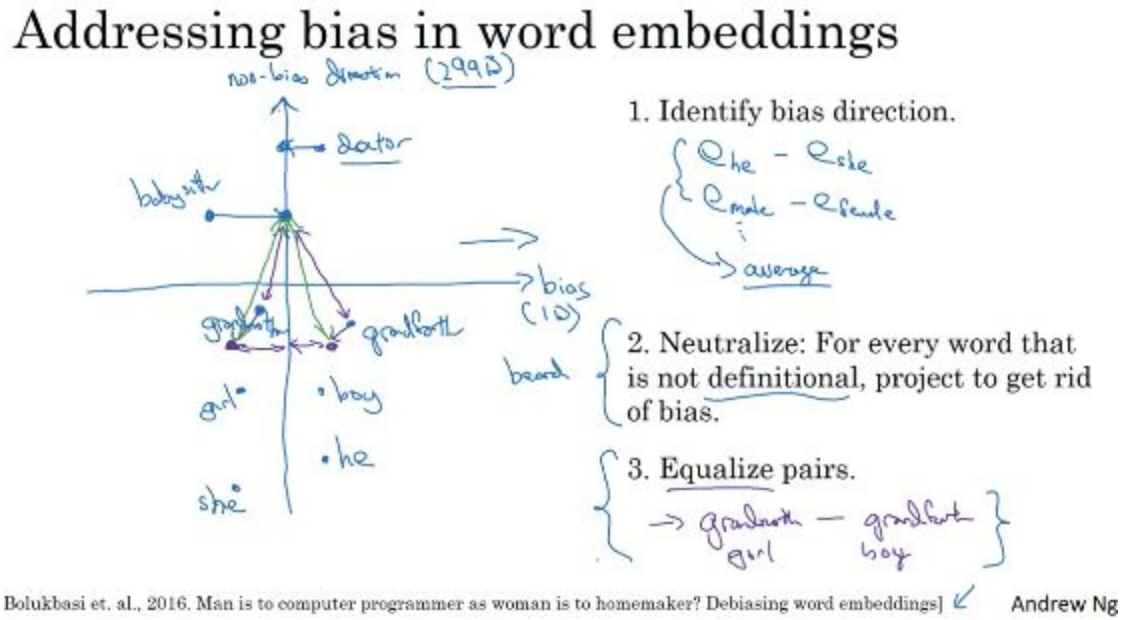
Debiasing Word Embeddings

- Word embeddings can reflect the biases of the text used to train the model.



We have a couple of examples above: a learning algorithm had as an output clearly biased responses (woman as a homemaker or nurse). This is a very important issue because learning algorithms are asked to weigh in in very important decisions.

- How do we get rid of the bias? We start by identifying the bias direction. We plot word embeddings as shown below and the first thing we need to do is identify the bias direction. How do we do that: we take the average of the difference between the embedding vectors of a few words. In the example below we are using gender bias but it can also be applied to other types of biases (i.e.ethnicity). So we have the average, then what? The second step is to neutralize and get rid of the bias for every word that is not definitional. Definitional words are words that have the gender 'built in' (i.e. grandfather, niece, nephew). We neutralize the bias by moving the non definitional words to the middle of the bias axis as shown below. Finally, we want to equalize word pairs because, as shown below with babysitter and grandmother and grandfather), babysitter is already biased to be female (the distance from babysitter to grandmother is smaller than the distance from babysitter to grandfather). The distance should be the same so we make it the same using linear algebra.



Finally, a classifier can be trained to tell us which words are definitional and which ones are not. The not definitional words we pass it through the neutralization steps to get rid of the bias. Most words in the English are not definitional and therefore, they should be neutralized.