Natural Language Processing & Word Embeddings

CSE 4237 - Soft Computing

Mir Tafseer Nayeem
Faculty Member, CSE AUST
tafseer.nayeem@gmail.com

Introduction

- Using word vector representations and embedding layers you can train recurrent neural networks with outstanding performances in a wide variety of applications like:-
    - **Sentiment Analysis**
    - **Named Entity Recognition**
    - **Machine Translation**

- Word embeddings is a way of representing words, to give the model a possibility to automatically understand analogies like:-
    - **a man is related to a woman**
    - **a king is related to a queen**

With word embedding you can train models with relatively small labeled data.

Content Credit: Andrew Ng

Word representation

Word Representation: So far we're representing words using a vocabulary of words, and each input is a one hot vector of the size of the vocabulary.

$$V = [\text{a}, \text{aaron}, \ldots, \text{zulu}, \text{<UNK>}]$$

Size of the Vocabulary |V| = 10,000

- Understanding the context of words is important. Detecting the similarity between the words we have seen in previous examples: **‘time’ and ‘age’, or 'stupidity' and 'foolishness'.**

Content Credit: Andrew Ng

Word representation

$V = [\text{a, aaron, …, zulu, <UNK>}]$

Size of the Vocabulary |V| = 10,000

1-hot representation

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

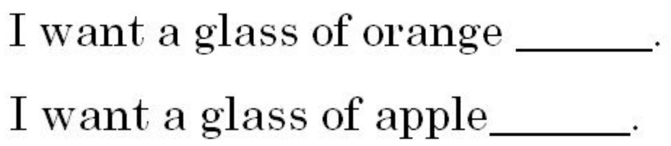
O (5391) **O (4914)** **O (456)**

Problems:-

- It treats each words individually.
- There isn't any relationship between the words, given that the product between any two vector is zero and not the similarity of the two words.
- It doesn't allow an algorithm to generalize across words.

Word representation

Let’s say, we have a language model which can predict the next word.


I want a glass of orange ______.

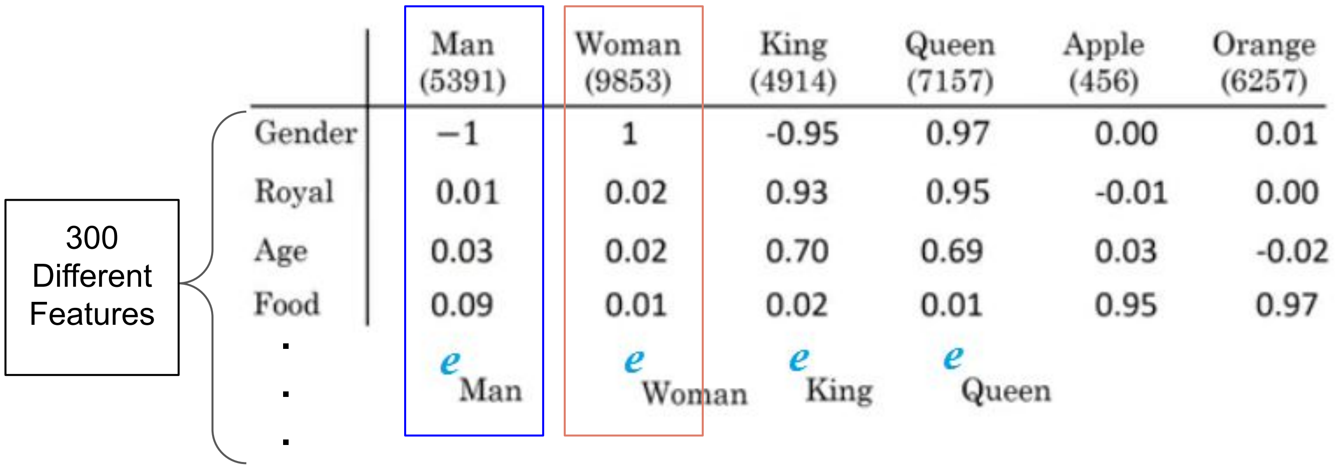
I want a glass of apple______.

juice

- The model should predict the next word as **juice**, given the previous word as **Apple**.
- In case of the second example,given the previous word as **apple**, the model won't easily predict **juice** here if it wasn't trained on it. **So the two examples aren't related although orange and apple are similar.**
- **Inner product between any one-hot encoding vector is zero**. Also, the distances between them are the same.

Featurized representation: word embedding

- Instead of a one-hot presentation, can we learn a featurized representation with each of these words: man, woman, king, queen, apple, and orange?

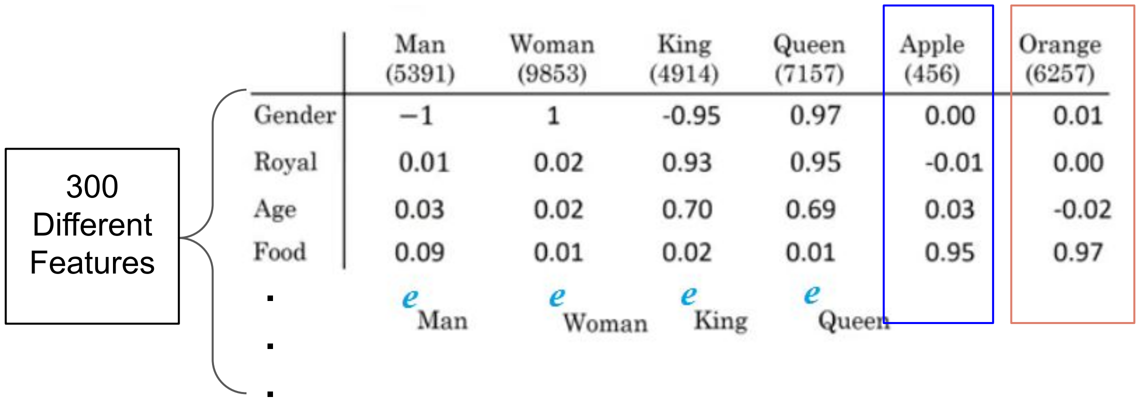

300 Different Features

	Man (5391)	Woman (9853)	King (4914)	Queen (7157)	Apple (456)	Orange (6257)
Gender	−1	1	-0.95	0.97	0.00	0.01
Royal	0.01	0.02	0.93	0.95	-0.01	0.00
Age	0.03	0.02	0.70	0.69	0.03	-0.02
Food	0.09	0.01	0.02	0.01	0.95	0.97
⋮	e_{Man}	e_{Woman}	e_{King}	e_{Queen}		

- Each word will have a, for example, **300 features** with a type of **float point number**.
- Each word column will be a 300-dimensional vector which will be the representation.

Content Credit: Andrew Ng

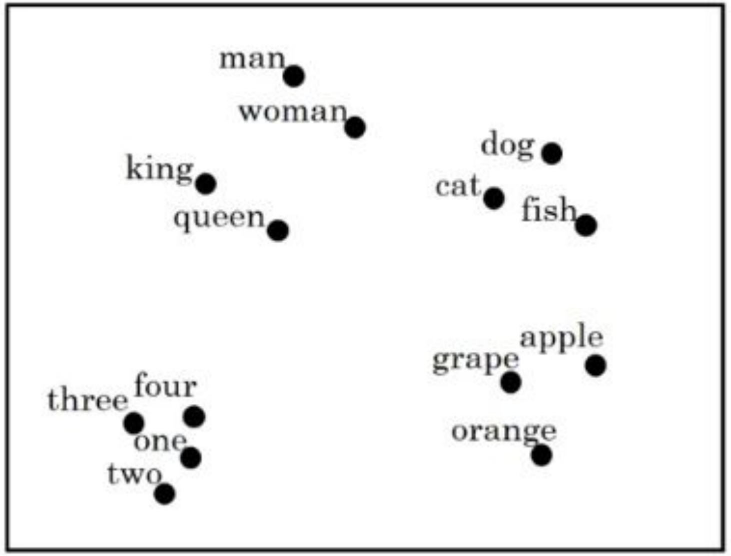
Featurized representation: word embedding


300 Different Features

	Man (5391)	Woman (9853)	King (4914)	Queen (7157)	Apple (456)	Orange (6257)
Gender	−1	1	-0.95	0.97	0.00	0.01
Royal	0.01	0.02	0.93	0.95	-0.01	0.00
Age	0.03	0.02	0.70	0.69	0.03	-0.02
Food	0.09	0.01	0.02	0.01	0.95	0.97
⋮	e_{Man}	e_{Woman}	e_{King}	e_{Queen}		

- Now, if we return to the examples we described again:
  - "I want a glass of **orange** ______"
  - “I want a glass of **apple** ______”
- Orange and apple **now share a lot of similar features** which makes it easier for an algorithm to generalize between them.
- We call this representation **Word embeddings**. Which is a **high dimensional feature vector** that gives a better representation than **One Hot Vector**.

Visualizing word embeddings



To visualize word embeddings we use a **t-SNE** algorithm to reduce the features to **2 dimensions** which makes it easy to visualize

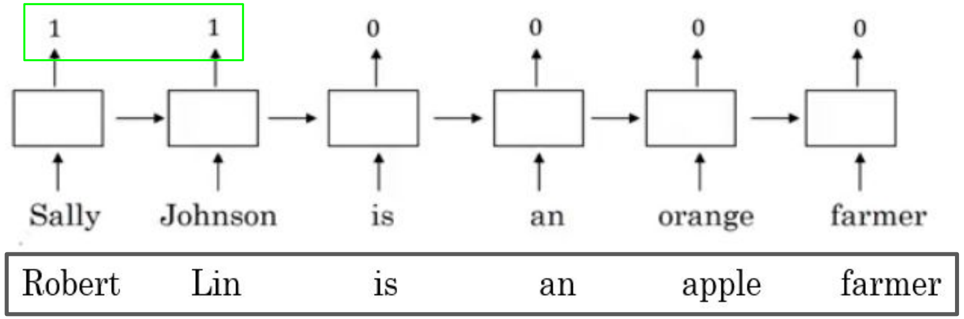
tSNE Algo (300 D) → 2D

- We are able to learn a given vector representation **(dimension of the vector << size of the vocabulary)**
- Take this high dimensional data and **embed it on 2D space**, we see similar words are closer together.

Using word embeddings

- We can take the feature representation we have extracted from each word and apply it in the **Named entity recognition** problem.

Sally Johnson is a person's name.



Labeled

Unlabeled

- As apple and orange have similar representations, the learning algorithm should label **Robert Lin** as **person's name**.
- **Much less common cases:-** *"Karim Rahman is a durian cultivator"* . the network should learn the name **even if it hasn't seen the word *durian* before (during training)**. That's the power of word representations.

Content Credit: Andrew Ng

Transfer learning and word embeddings

The algorithms that are used to learn word embeddings can examine **billions of words of unlabeled text** - for example, **100 billion words** and learn the representation from them which is available for free in the internet.

1. Learn word embeddings from large text corpus (1 - 100B Words),
    a. Or **download / re-use** pre-trained embeddings from online.
2. Transfer embedding to new task with the **smaller labeled training set (say, 100k words)**.
3. **Optional:** continue to **fine-tune (adjust)** the word embeddings with new data for our task.
    a. You bother doing this if your smaller training set **(from step 2) is big enough**.
    b. **If data in the step 2 is small then it's better not to fine-tune.**

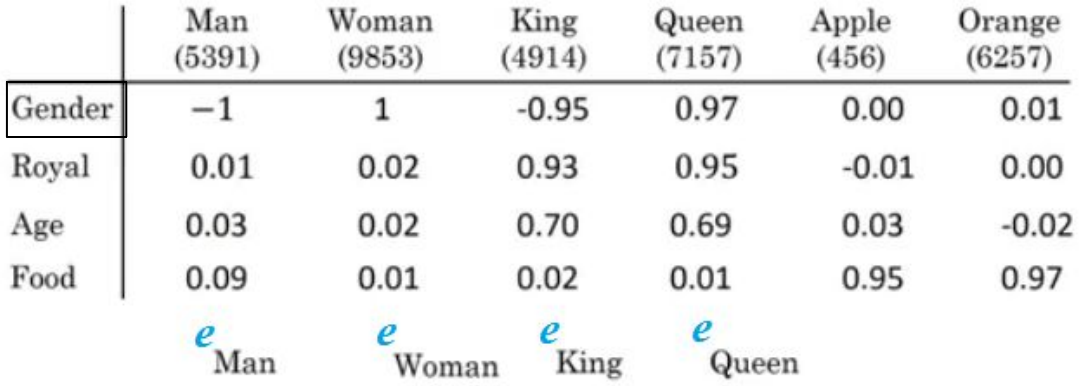
Other Advantages of Word Embedding

- Word embeddings tend to make the biggest difference when the task you're trying to carry out has a **relatively smaller training set**.
- Also, other advantages of using word embeddings is that it reduces the size of the input! **10,000 one hot (sparse) compared to 300 features vector (dense).**

Content Credit: Andrew Ng

Properties of word embeddings

- Word embeddings can also help with **analogy reasoning** which might help convey a sense of what these word embeddings can do to NLP applications.

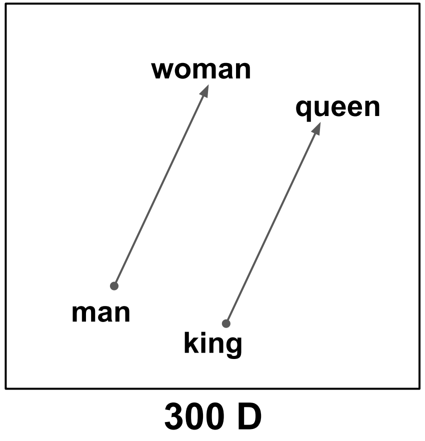


	Man (5391)	Woman (9853)	King (4914)	Queen (7157)	Apple (456)	Orange (6257)
Gender	−1	1	-0.95	0.97	0.00	0.01
Royal	0.01	0.02	0.93	0.95	-0.01	0.00
Age	0.03	0.02	0.70	0.69	0.03	-0.02
Food	0.09	0.01	0.02	0.01	0.95	0.97
	e_{Man}	e_{Woman}	e_{King}	e_{Queen}		

- **Man ==> Woman**
- **King ==> ??**

- Subtract **e(Man)** from **e(Woman)** equal the vector `[-2 0 0 0]`
- Similarly, **e(King)** - **e(Queen)** = `[-2 0 0 0]`

Analogies using word vectors



- So we can reformulate the problem to find:
  - **e(man) - e(woman) ≈ e(king) - e(w) ??**
- It can be represented mathematically by:-

$$\text{argmax}_w \; sim\left(e_w, e_{king} - e_{man} + e_{woman}\right)$$

- It turns out that **e(queen)** is the best solution here that gets the the similar vector.

Similarity functions

There are two commonly used similarity functions, such as:-

1. **Euclidean distance**

$$\boxed{\sqrt{\sum_{i=1}^{k}\left(u_i - v_i\right)^2}}$$

2. **Cosine similarity**

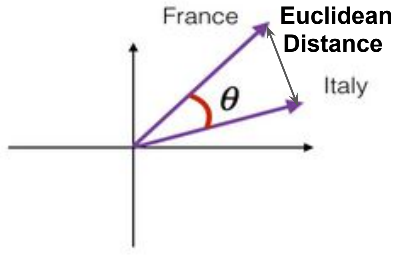
$$\boxed{\text{CosineSimilarity}(\text{u, v}) = \frac{u \bullet v}{||u||_2 . ||v||_2} = cos(\theta)}$$

Cosine similarity

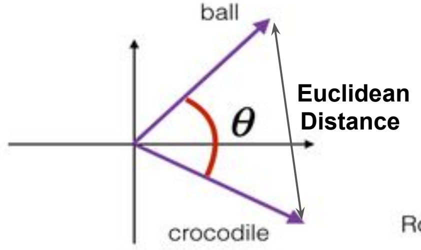
$$\text{CosineSimilarity(u, v)} = \frac{u \bullet v}{||u||_2 . ||v||_2} = cos(\theta) \quad = \frac{u \bullet v}{\sqrt{u \bullet u}\sqrt{v \bullet v}}$$

where **(u.v)** is the dot product (**or inner product**) of two vectors, denominator is the **L2 norm (or length)** of the vector **u** and **v**, and **ϴ** is the angle between **u** and **v**.

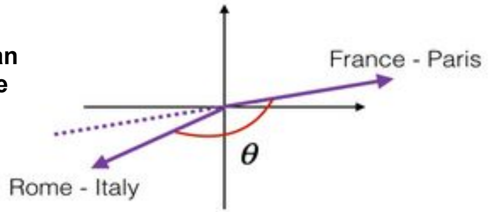
This similarity depends on the angle between **u** and **v. If u** and **v** are very similar, their **cosine similarity will be close to 1**; if they are dissimilar, the cosine similarity will have a smaller value.



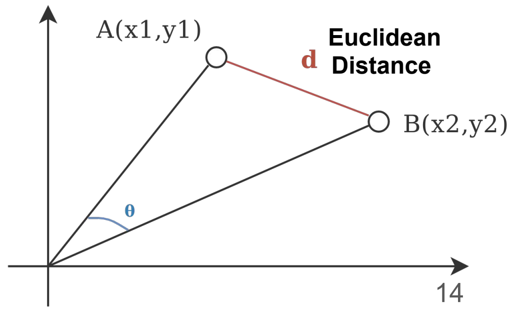
France and Italy are quite similar
θ is close to 0°
$\cos(\theta) \approx 1$



ball and crocodile are not similar
θ is close to 90°
$\cos(\theta) \approx 0$

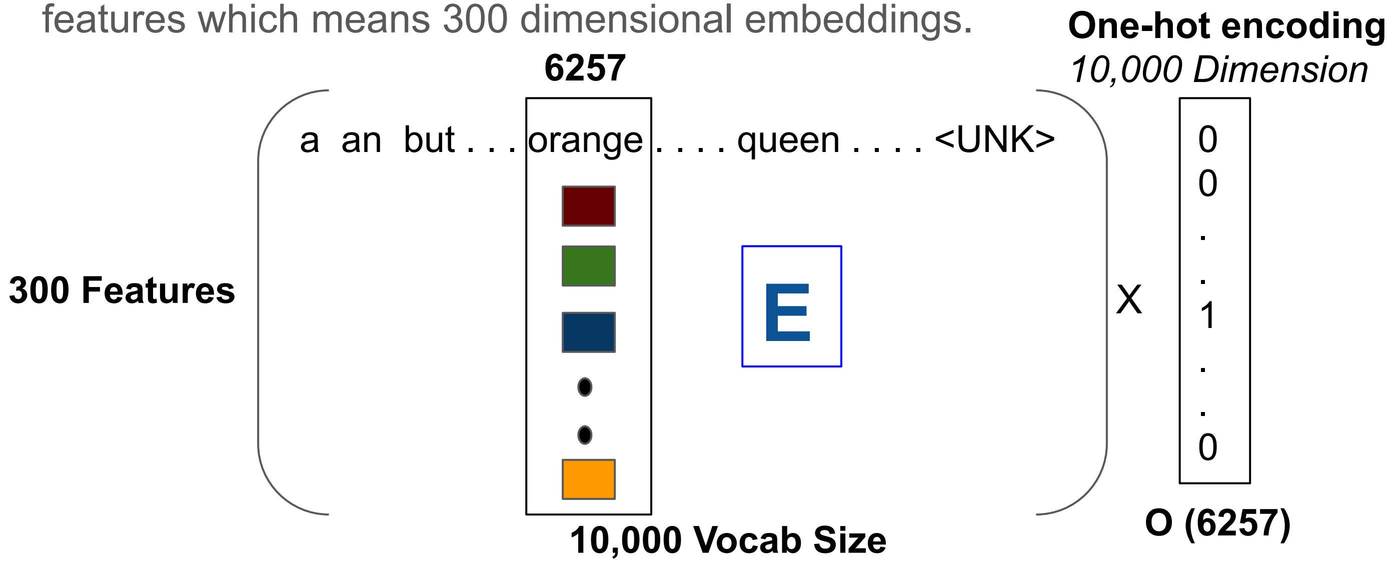


the two vectors are similar but opposite
the first one encodes (city - country)
while the second one encodes (country - city)
θ is close to 180°
$\cos(\theta) \approx -1$

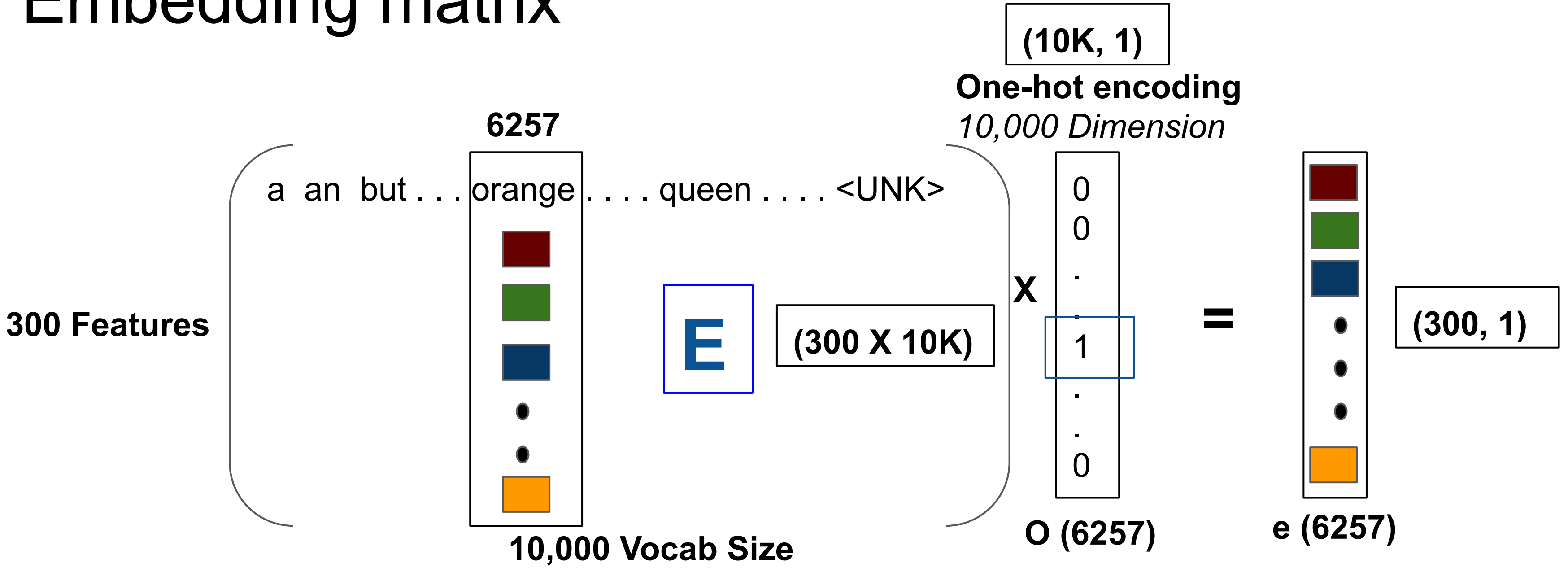


Embedding matrix

- When we implement an algorithm to learn a **word embedding**, what we end up learning is a **embedding matrix**.
- Suppose we are using **10,000 words** as our vocabulary (*including token like <UNK>*)
- The algorithm should create a matrix E of the size **300 x 10,000** if we are extracting 300 features which means 300 dimensional embeddings.



Embedding matrix



- To find the embeddings of the word *'orange'* which is at the **6257th** position, we multiply the above embedding matrix with the **one-hot vector** of orange:

E x O(6257) = e (6257)

- The shape of **E** is **(300, 10k)**, and **O** is **(10k, 1)**. The embedding vector **e(6257)** will be of the shape **(300, 1)**.

Embedding matrix

Generally, $E \cdot O_j = e_j$

- **E** = Embedding Matrix.
- **O_j** = One-hot vector of word j in the vocabulary.
- **e_j** = embedding for word j in the vocabulary.

- We initialize the matrix **E (300 x 10K)** randomly and use **gradient descent** to learn the **parameters of the matrix**.
- In practice, it is **not efficient to use vector multiplication** to extract the word embedding, but rather a specialized function to **look up the embeddings**.
- Because, **O** vector is a high dimensional **one-hot vector** and most of these elements will be zero.

Learning word embeddings

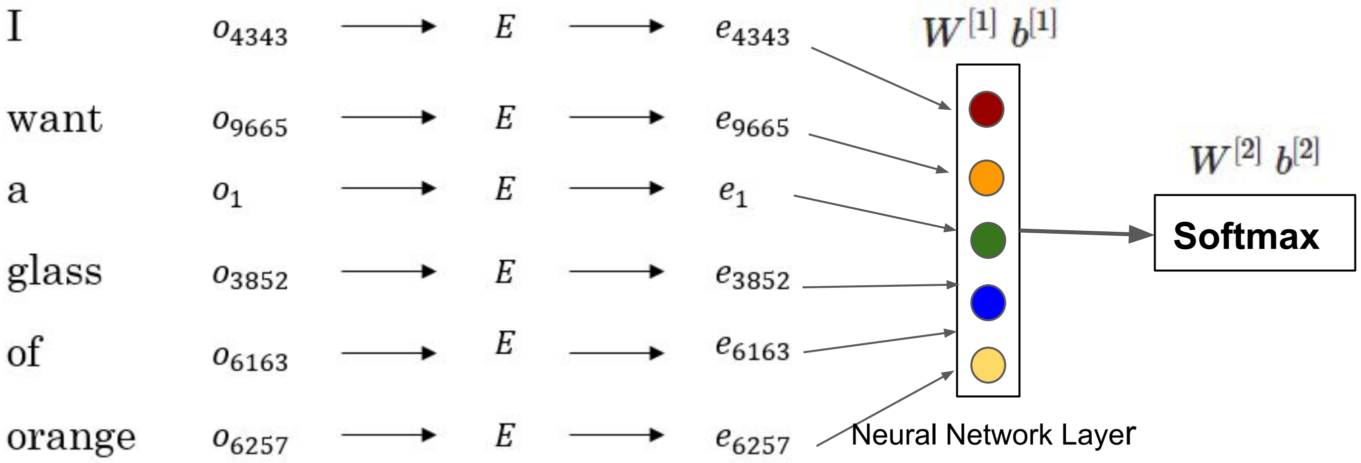
People started off with relatively **complex algorithms**. And then over time,researchers discovered they can use simpler algorithms and still get very good results especially for a large dataset.

I	want	a	glass	of	orange	$\overset{?}{\underline{\qquad\quad}}$.
4343	9665	1	3852	6163	6257	

Consider we are building a **language model** using a neural network. The input to the model is **"I want a glass of orange"** and we want the model to predict the **next word** in a sequence.

Learning word embeddings

$$E \cdot O_j = e_j$$



- We first embed the input word, and feed them into a **hidden layer** and then use a **softmax layer** to predict the probabilities over **10,000 words**.
- NN layer has parameters **W1** and **b1** while softmax layer has parameters **W2** and **b2.**
- Input dimension is **(300*6, 1)** if the window size is **6 (six previous words)**. Take individual embedding vectors of 300 dimensions and **stacking them together.**

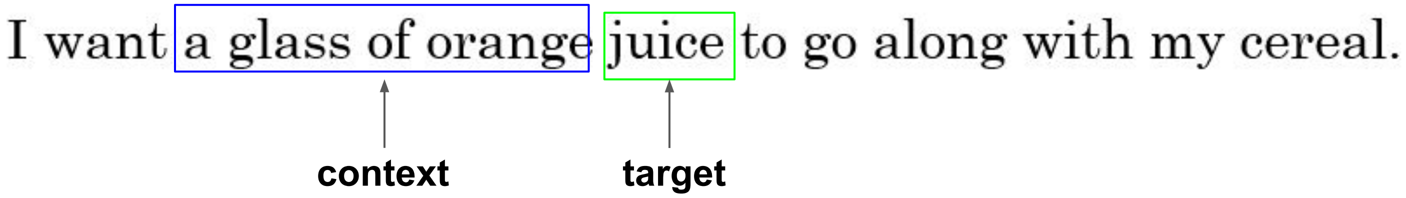
Learning word embeddings

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

Hyperparameter

- We can **reduce the number of input words**, to decrease the input dimensions.
- We want our model to use maximum **previous 4 words (fixed)** only to make prediction. In this case, the **input will be 1200 dimensional**.
- The input can also be referred as context and there can be various ways to select the context.
- The parameters for this model are:
    - Embedding matrix (**E**) [use the same E for all the words]
    - W[1], b[1]
    - W[2], b[2]
- Use gradient descent to perform backpropagation to maximize the likelihood to predict the next word given the context (previous words).

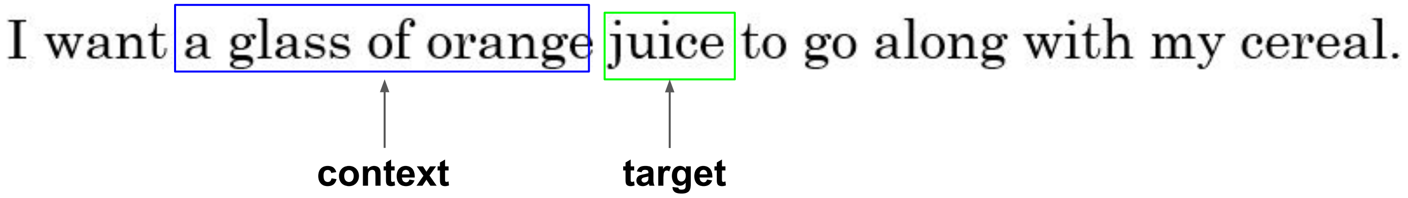
Other context/target pairs



To learn juice, choices of context are:

1. **Last 4 words.**
   a. We use a window of last 4 words (*4 is a hyperparameter*), *"a glass of orange"* and try to predict the next word from it.
2. **4 words on the left and on the right.**
   a. "a glass of orange" and "to go along with"
3. **Last 1 word.**
   a. "orange"

Other context/target pairs



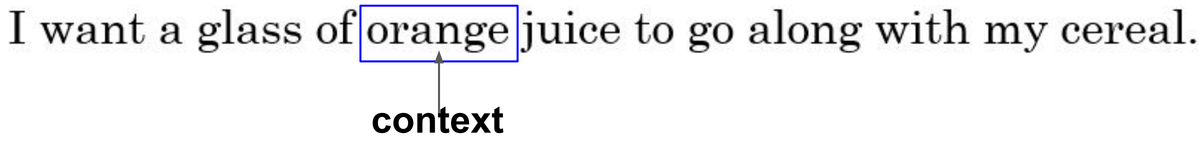
4. **Nearby 1 word.**
   a. "glass" word is near juice.
   b. This is the idea of skip grams model.
   c. The idea is much simpler and works remarkably well.

- If you really want to build a language model, it's natural to use the **last few words** as a context. But if you want to learn a **good word embedding**, then you can use all of these other contexts.
- Language modeling problem is a **machines learning problem** where you input the context (*like the last four words*) and predict some target words which allows you to learn good word embeddings.

Word2Vec : Skip Gram

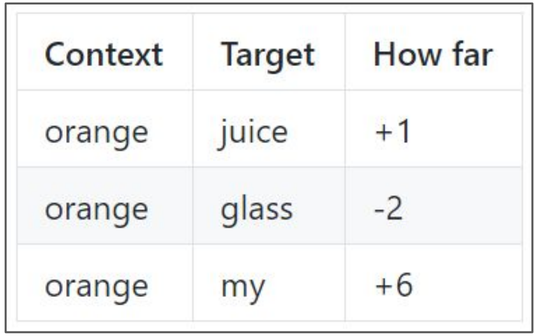
It is a **simple and more efficient way** to learn word embeddings.

Consider we have a sentence in our training set,



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

- We will choose context and target pairs. Randomly pick a context word.
- The target is chosen randomly based on a window with a specific size.
- We have converted the problem into a supervised problem to predict the target word given the context word.
- This is not an easy learning problem because learning within **-10/+10 words** is hard as there could be lot of different words.
- We want to learn this to get our word embeddings model.



Context	Target	How far
orange	juice	+1
orange	glass	-2
orange	my	+6

Word2Vec Model

- Vocabulary size = **10,000 words**
- Let's say that the context word is **c** (*"orange"*) and the target word is **t** (*"juice"*).
- We want to learn a mapping from **c to t**

X → **Y**

context c [6257] (*"orange"*) → target t [4834] (*"juice"*)

$$\begin{tikzcd} O_c \arrow[r] & E \arrow[r] & e_c \arrow[r] & \bigcirc \arrow[r] & \hat{y} \end{tikzcd}$$

Softmax

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

θ_t = parameters associated with the output **t** .

Word2Vec Model

- $$\begin{tikzcd} O_c \arrow[r] & E \arrow[r] & e_c \arrow[r] & \underset{\textbf{Softmax}}{\bigcirc} \arrow[r] & \hat{y} \end{tikzcd}$$

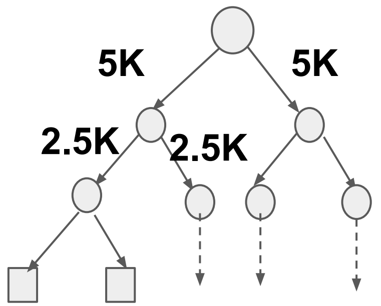
Softmax $$p(t|c) = \frac{e^{\theta_t^T e_c}}{\sum_{j=1}^{10,000} e^{\theta_j^T e_c}}$$

θ_t = parameters associated with the output **t** .

Loss Function $$\mathcal{L}(\hat{y}, y) = -\sum_{i=1}^{10,000} y_i \log \hat{y}_i$$

$$y = \begin{pmatrix} 0 \\ 0 \\ . \\ 1 \\ . \\ 0 \\ 0 \end{pmatrix} \quad \xleftarrow{\textbf{4834}} \quad \xrightarrow{\textbf{4834}} \quad \hat{y} = \begin{pmatrix} 0.01 \\ 0.03 \\ . \\ 0.87 \\ . \\ 0.05 \\ 0.02 \end{pmatrix}$$

Problems with softmax classification



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

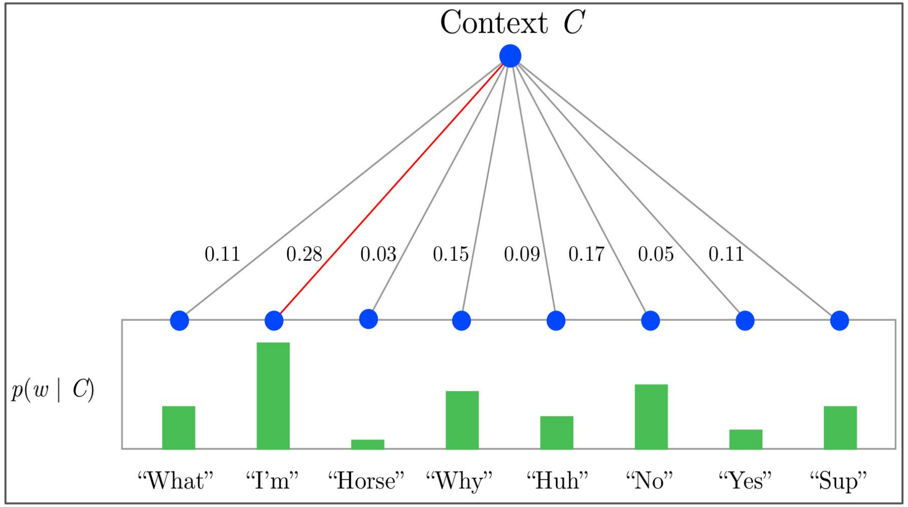
Problems:

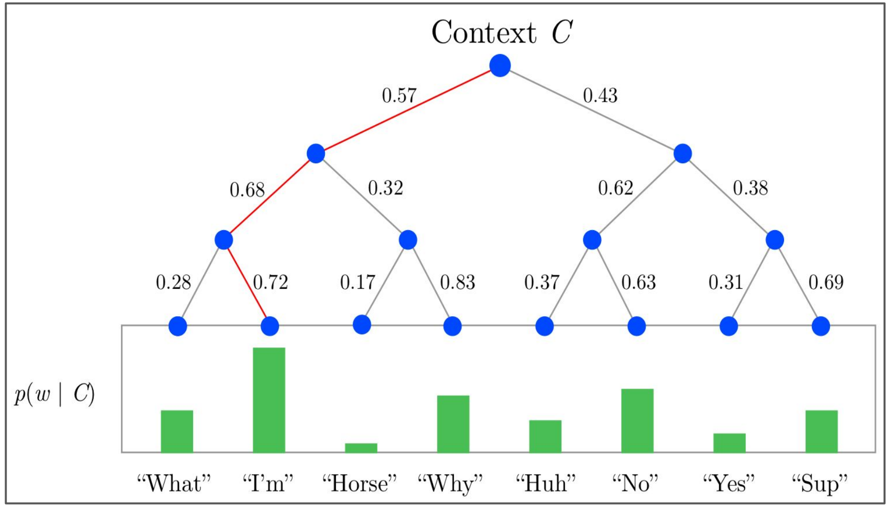
1. Here we are summing **10,000** numbers which corresponds to the number of words in our vocabulary.
2. If this number is larger say **100K or 1 million**, the computation will become **very slow**.

Solution:

- Use **"Hierarchical softmax classifier"** which works as a tree classifier.
- Complexity of Hierarchical softmax classifier is **O(log(n)) instead of O(n)**.

Hierarchical softmax classifier

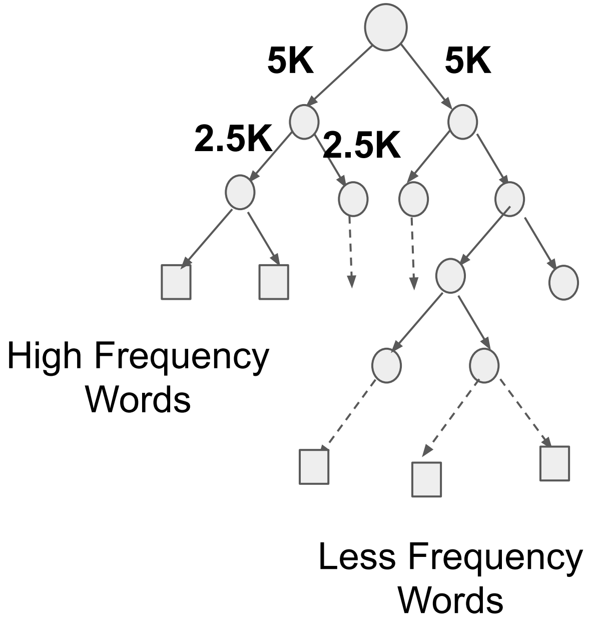




- To evaluate the probability of a given word, take the product of the probabilities of each edge on the path to that node: $P(\text{I'm}|C) = 0.57 * 0.68 * 0.72 = 0.28$
- Now, in the case of a binary tree, this can provide an exponential speedup. In the case of **1 million words**, the computation involves **log(1000000)=20 multiplications!**

Hierarchical softmax classifier

This tree (can be asymmetric), where most commons words tend to be on top and less common words deeper to further reduce the computations.



Many neural language models nowadays use either hierarchical softmax or other softmax approximation techniques. For more reading, check out:

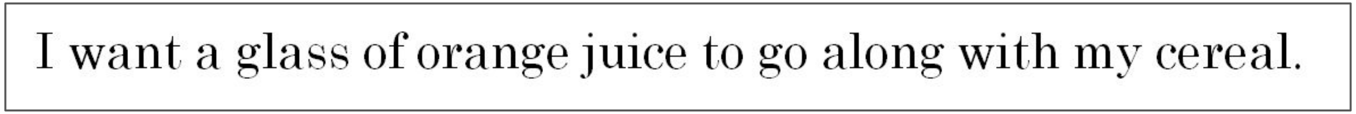
- **Negative sampling**
- Differentiated softmax
- [Adaptive] importance sampling

How to sample the context c?

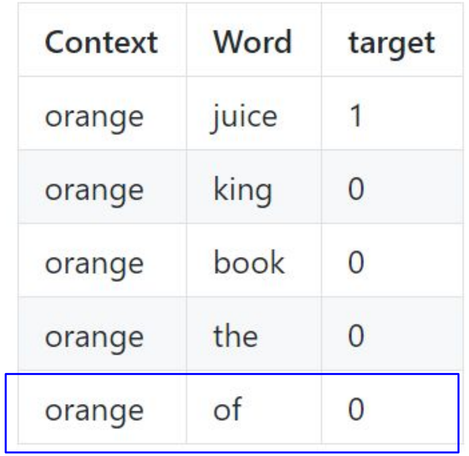
- One way could be to sample the **context word at random**. Then target ***t*** can be sampled within **[-10 , +10]** window.
- The **problem with random sampling** is that the common words like ***is, the, and, to, of*** will appear more frequently whereas the unique words like **orange, apple** might not even appear once in our **c** → **t** mapping pairs.
- We don’t want our training set to be dominated by **extremely frequent words**. Then we will learn only the embeddings of the frequently occuring words.
- In practice, we don't take the **context uniformly random**, instead we try to choose a method which gives more weightage to less frequent words and less weightage to more frequent words.
- word2vec paper includes 2 ideas of learning word embeddings. One is skip-gram model and another is **CBoW (continuous bag-of-words).**

Negative sampling

- One downside of skip gram model was **high computational cost due to softmax**.
- Negative sampling allows you to do something similar to the skip-gram model, but with a much **more efficient learning algorithm**. We will create a different learning problem.


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

- It creates a new supervised learning problem, where given a pair of words say **“orange” and “juice”**, we will predict whether it is a context-target pair?
- We get positive example by using the same skip-grams technique with a fixed window say **[-10 to +10]**
- To generate a negative example, we pick a word **randomly from the vocabulary**. These 0 values represent that it is a negative sample.
- Notice, that we got word **"of"** as a negative example although it appeared in the same sentence.

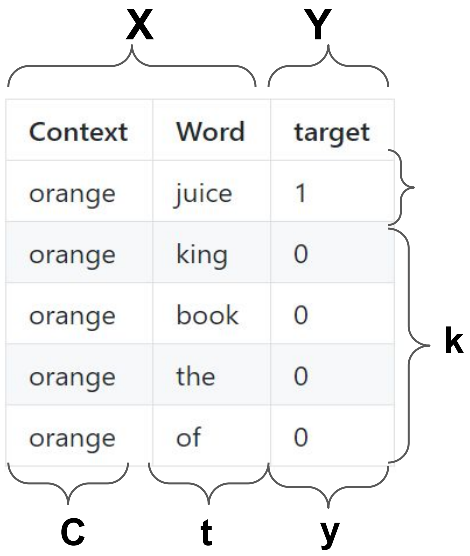


Context	Word	target
orange	juice	1
orange	king	0
orange	book	0
orange	the	0
orange	of	0

Negative sampling

So the steps to generate the samples are:

- Pick a positive context word such as Orange.
- Pick a **k** negative contexts from the dictionary.



Context	Word	target
orange	juice	1
orange	king	0
orange	book	0
orange	the	0
orange	of	0

- k is recommended to be from **5 to 20** in small datasets. For larger ones: **2 to 5**.
- We will have a ratio of **k** negative examples to 1 positive ones in the data we are collecting.

Now let's define the model that will learn this supervised learning problem:

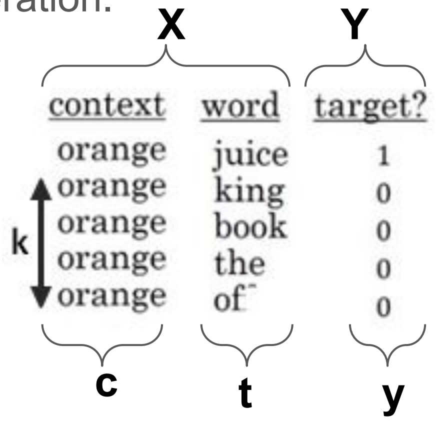
- Let's say that the context word is **c** and the word is **t** and **y** is the target.
- We will apply the simple logistic regression model.

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

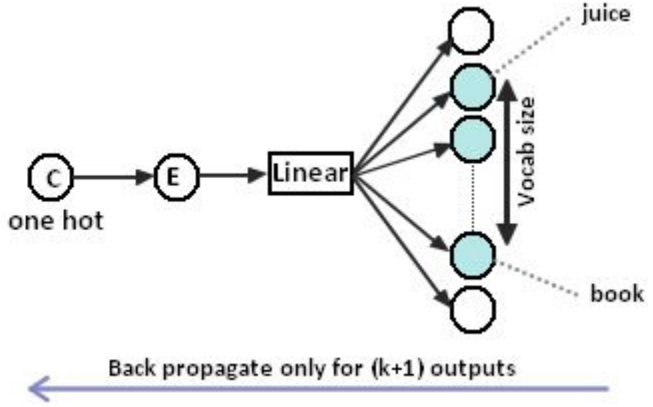
Negative sampling

- To solve the computation problem, we model the task as a **binary logistic regression** problem, so instead of using **10,000 way softmax**, at each step only **k+1** classifiers are modified **(k negatives and 1 positive)**.
- So we are like having **10,000 binary classification problems**, and we only train **k+1** classifier of them in each iteration.

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



context	word	target?
orange	juice	1
orange	king	0
orange	book	0
orange	the	0
orange	of	0



Selecting negative examples

- We can **sample** according to empirical frequencies in words corpus which means according to how often different words appears. But the problem with that is that we will have more frequent words like **the, of, and**...

$$\boxed{P(w_i) = \frac{f(w_i)}{\sum_{j=0}^{n} (f(w_j))}}$$

The best is to sample with this equation (according to authors):

$$\boxed{P(w_i) = \frac{f(w_i)^{3/4}}{\sum_{j=0}^{n} \left(f(w_j)^{3/4}\right)}}$$

$f(\omega_i)$ Frequency of a particular word **i** in the corpus.

END