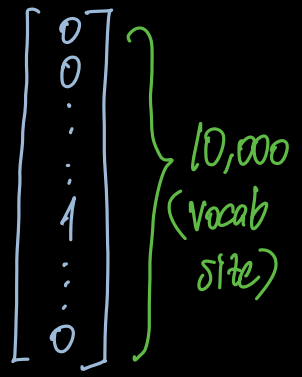
Word Embeddings

Representing words as one-hot vectors acc. to their order in vocab. dictionary is not useful. In one-hot representation, algo does not know e.g. "apple" and "orange" are close words. Instead, word embeddings are used where each word is represented by ~300 dimensional vector with features e.g. gender, age, size ...

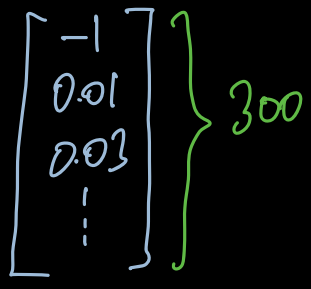
One-hot of "Man" (5391)



O_{5391}

	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

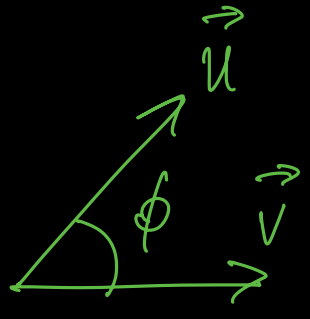
embedding of "Man"

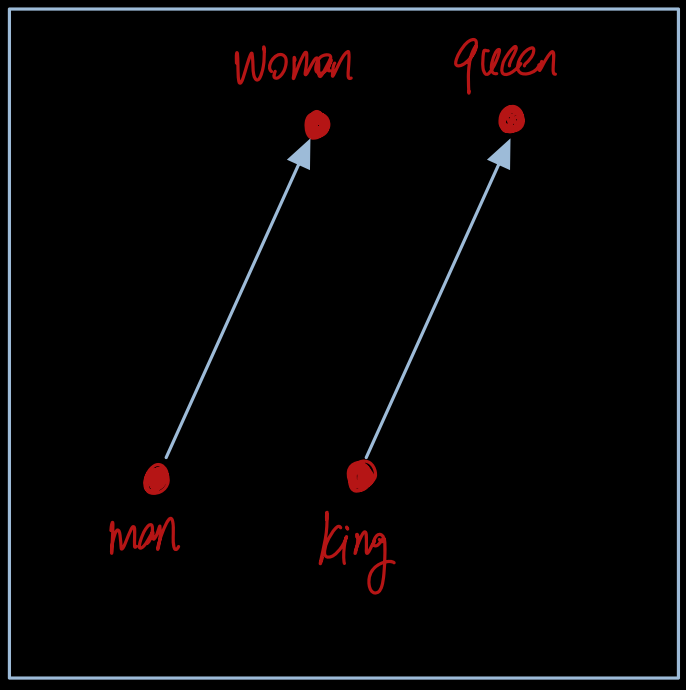


e_{5391}

In this 300D space, vectors of similar words are close to each other. Similarity is defined as the angle or cosine of two vectors:

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



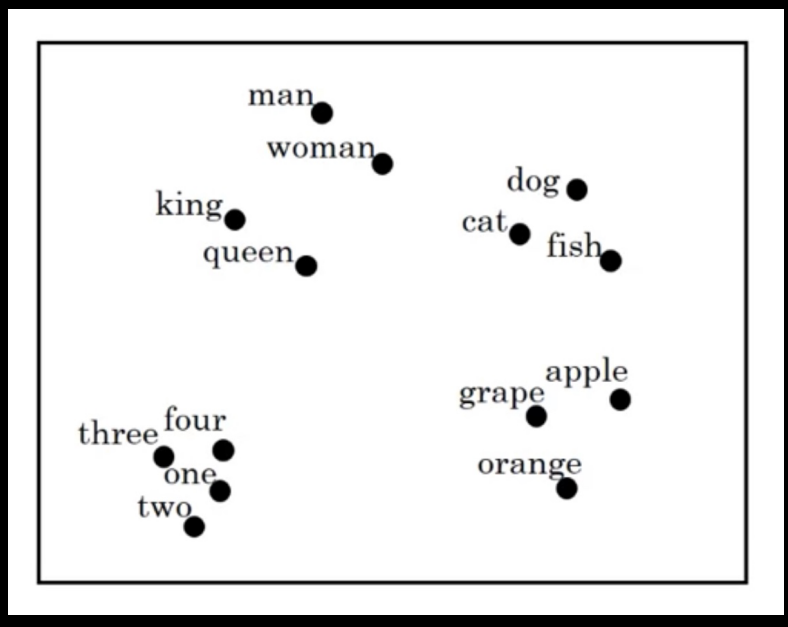


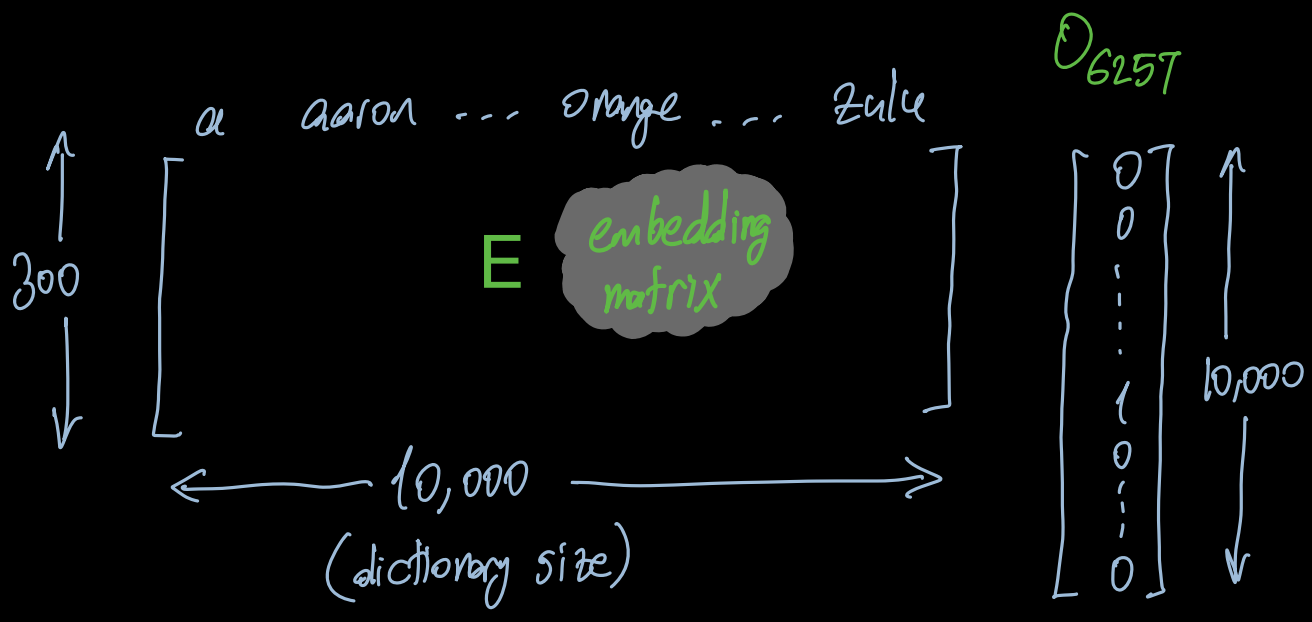
300D space

man is to woman as king is to queen:

$$e_{man} - e_{woman} \approx e_{king} - e_{queen}$$

t-SNE is a popular algo that visualizes 300D word embeddings in 2D. t-SNE is very complex and non-linear algo that is only good for basic visualization purposes e.g. t-SNE representation of words cannot be used for a similarity metric!





$$E \cdot O_{6257} = e_{6257}$$

Learning embedding matrix requires big data sets, thus pretrained embedding matrices are used in NLP tasks (maybe by slightly modifying them based on the needs of the project i.e. further training pre-trained embedding matrix to be used for a medical project) The most commonly used word embeddings in NLP are Word2Vec and GloVe.

Word2Vec Embeddings

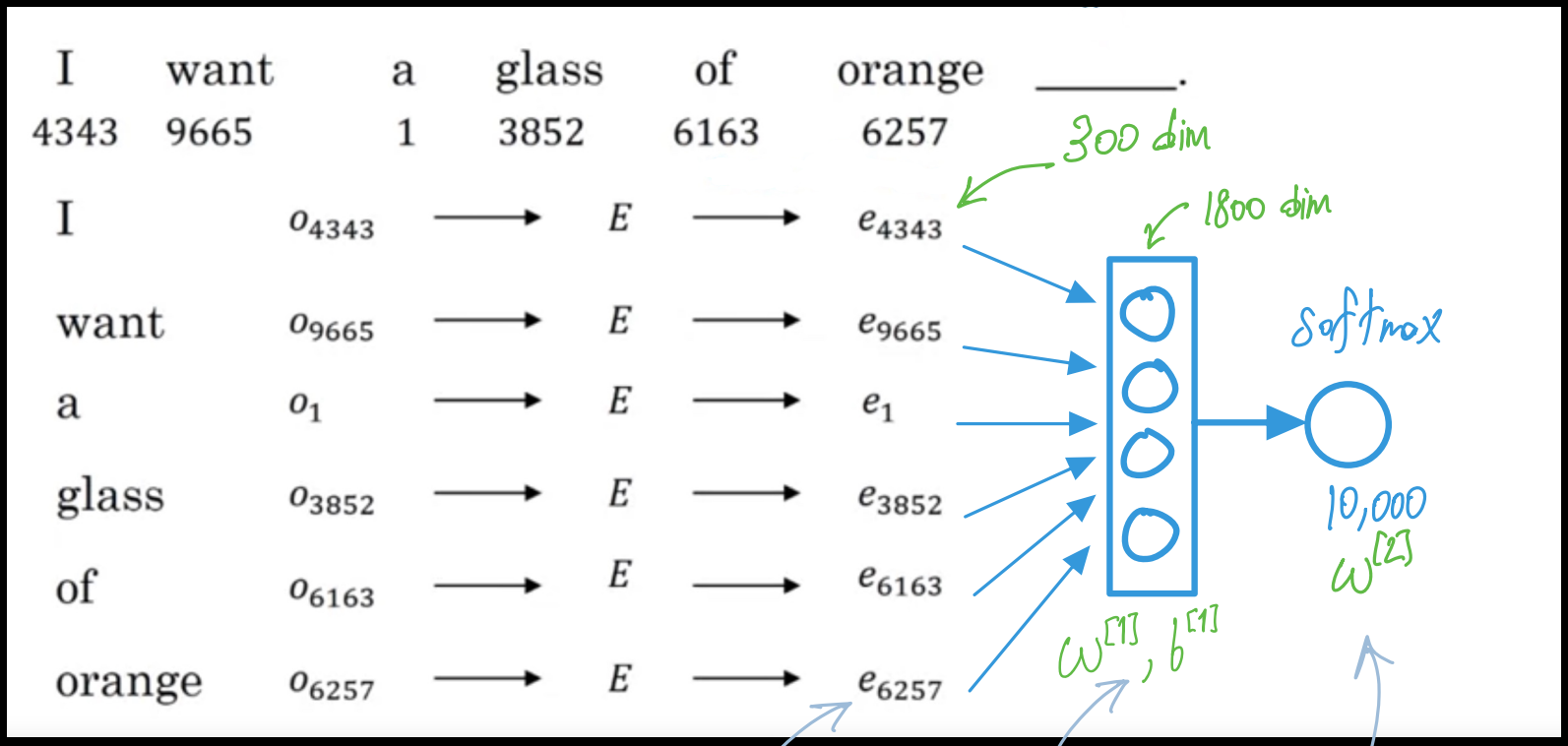
There are two major approach to learn embedding matrix using Word2Vec algo:

1) Continuous bag of words (CBOW): given a context, shallow network tries to predict the most likely target word

2) Skip-grams: the other way around: uses the target word to predict its context.

Continuous bag of words method:

In the ex. sentence below, we are trying to predict the last word "juice" given the previous words in the sentence.



300 dim embedding of each word is calculated using E

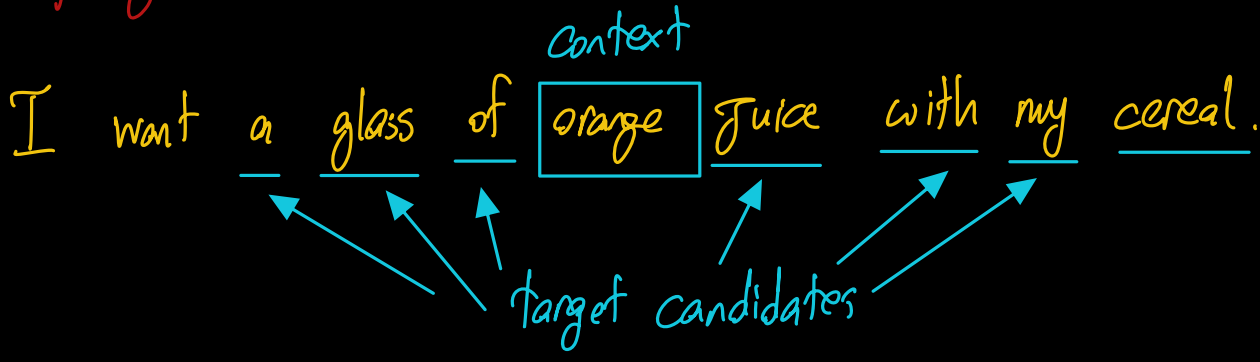
Six embedding vectors concatenated

Outputs prob. of each word in the corpus

- The embedding matrix E, params of the shallow network $W^{[1]}$, $b^{[1]}$ and params of the softmax $W^{[2]}$ are trainable params
- Optionally, only the last 4 words before the target word could be chosen as the context (instead of all the words before)

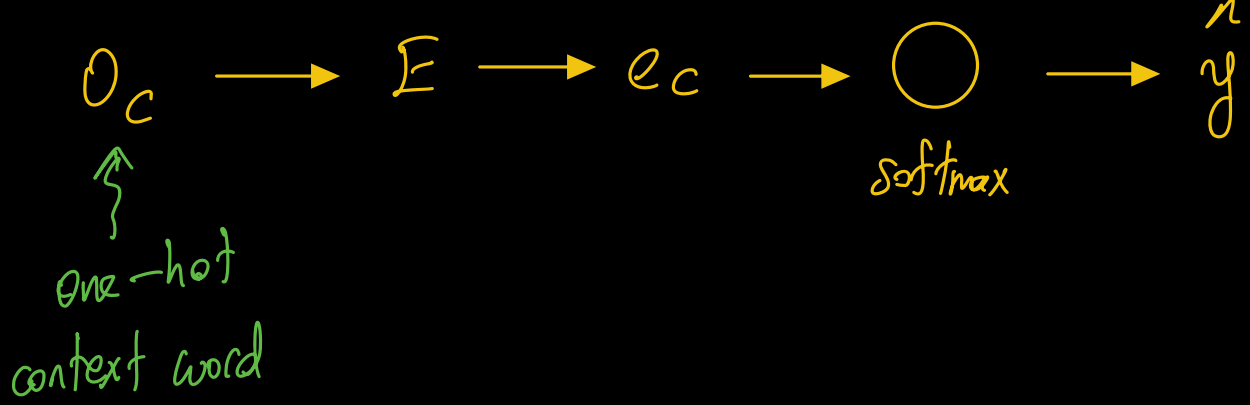
Or even the last 1 word would be also ok. Or 2 words before and 2 words after...

Skip-gram method:



randomly choose one target word within say 3 words

Context	Target
orange	juice
orange	glass
orange	of
⋮	⋮



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

θ_t is the softmax param to be trained — has the same dim. as e_c

prob of a nearby target word t given context word c

one-hot vector of i'th word target in corpus

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

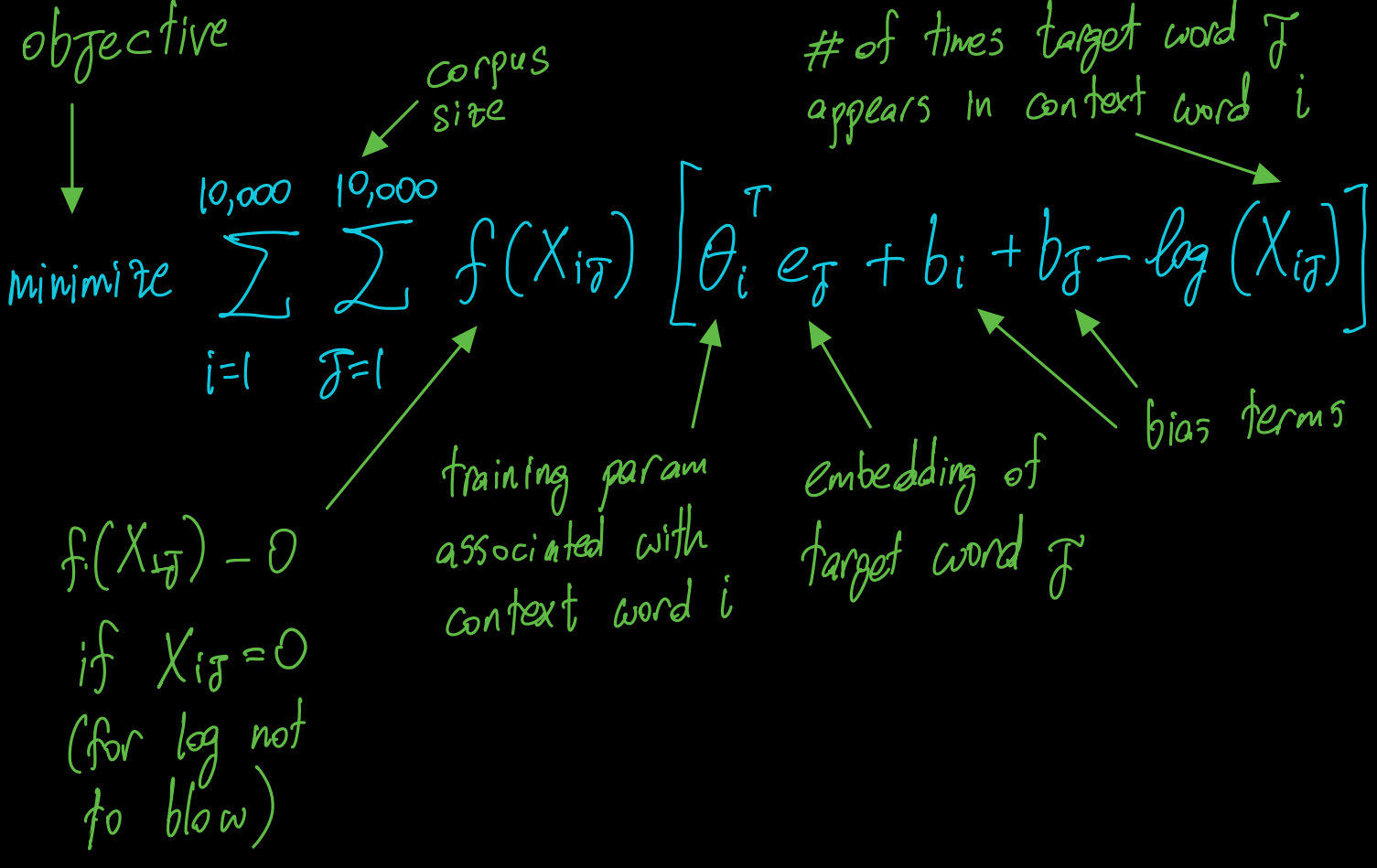
Embedding matrix E and softmax params θ_t are trained and useful embedding matrix E is obtained from there.

GloVe Embeddings

- Word2Vec learns embeddings by relating target words to their context. However, it ignores whether some context words appear more often than others. A frequent co-occurance of words creates more training examples, but that information is ignored / not considered by Word2Vec.

- In contrast, GloVe stresses that the frequency of co-occurance is vital information and should not be wasted.
- GloVe embeddings can be interpreted as a summary of the training corpus with low dimensionality based on co-occurances.

GloVe objective



Sentiment Classification — an application of word embeddings:

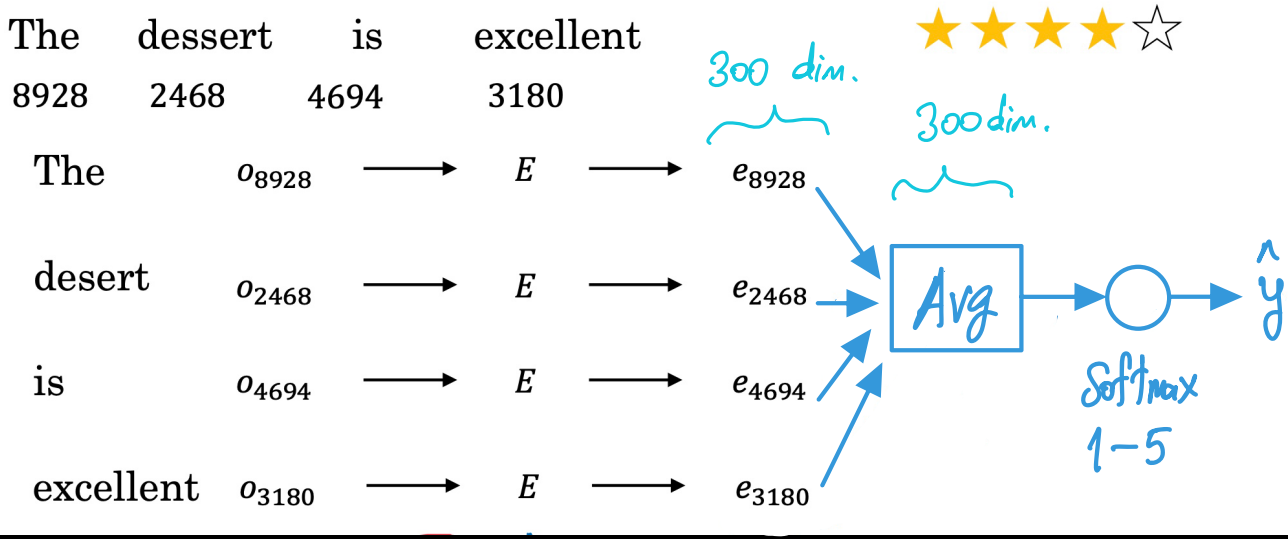
x	y
The dessert is excellent.	★★★★☆
Service was quite slow.	★★☆☆☆
Good for a quick meal, but nothing special.	★★★☆☆
Completely lacking in good taste, good service, and good ambience.	★☆☆☆☆

Task is to learn 1-5 stars from a sentence

Simple model

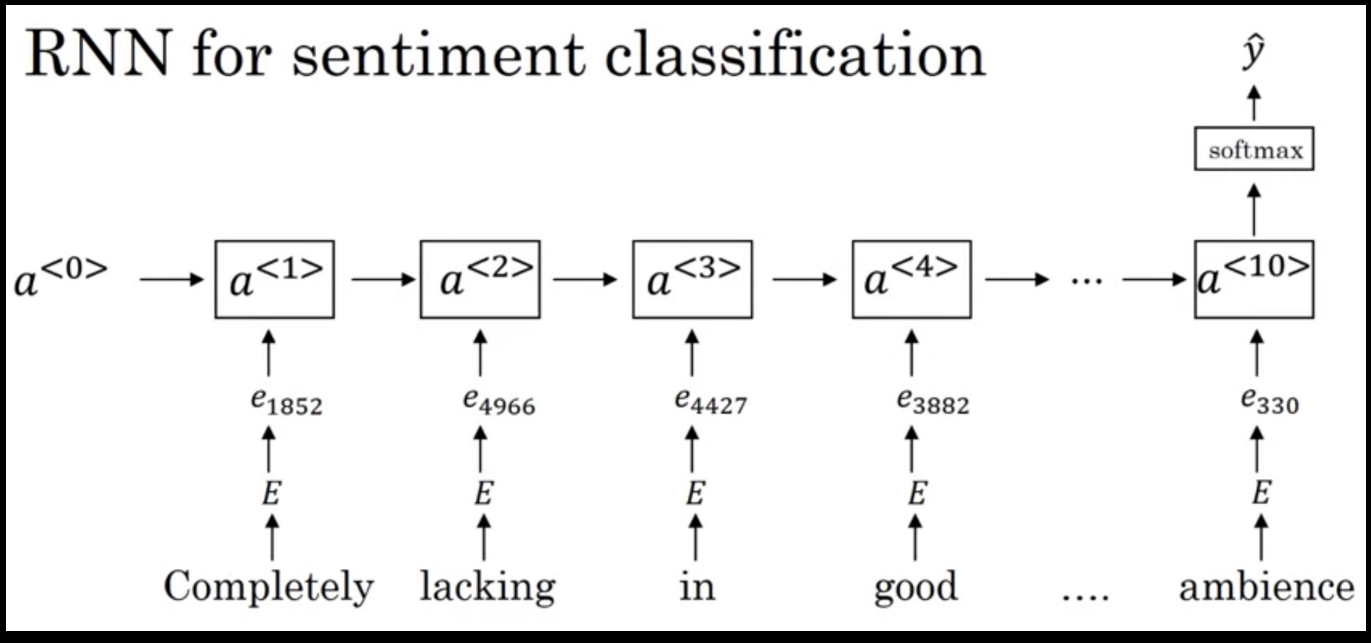
This naive model averages embeddings of all words in a given review sentence. It does not take into account word orders. Thus, it's not able to capture "not good" or "lacks of good taste"... Even so, it has a decent accuracy!

Simple sentiment classification model



more complex model: RNN

Using RNN is better as it can take into account word orderings. This is an many-to-one RNN example.



few NLP implementation tips

What you should remember:

- If you have an NLP task where the training set is small, using word embeddings can help your algorithm significantly.
- Word embeddings allow your model to work on words in the test set that may not even appear in the training set.
- Training sequence models in Keras (and in most other deep learning frameworks) requires a few important details:
  - To use mini-batches, the sequences need to be **padded** so that all the examples in a mini-batch have the **same length**.
  - An `Embedding()` layer can be initialized with pretrained values.
    - These values can be either fixed or trained further on your dataset.
    - If however your labeled dataset is small, it's usually not worth trying to train a large pre-trained set of embeddings.
  - `LSTM()` has a flag called `return_sequences` to decide if you would like to return every hidden states or only the last one.
  - You can use `Dropout()` right after `LSTM()` to regularize your network.