Embeddings

&

Word2vec

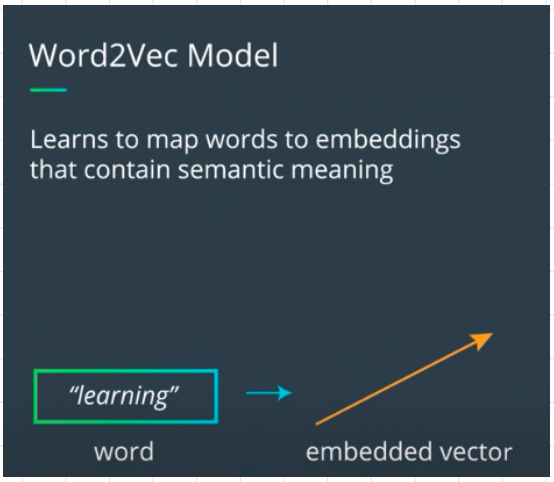
Word Embeddings 詞嵌入

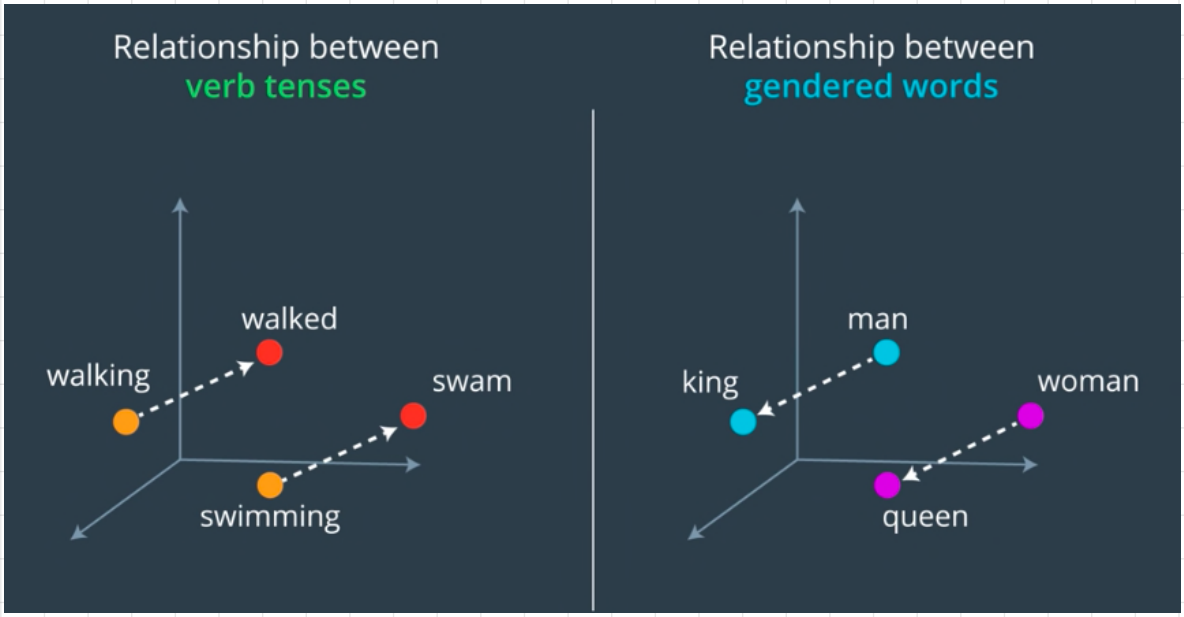
Models that map a set of words/phrases/vocabs to vectors of numerical values.

↳ Embeddings

Use: Reduce dimensionality of text data

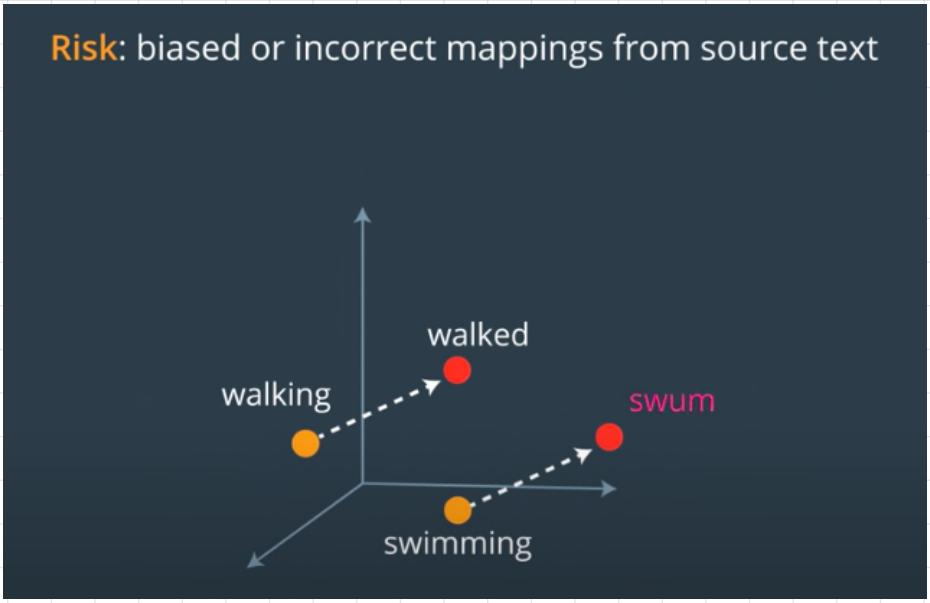
Our Focus →





Example:

- Verb tenses, gendered words, etc.
- Embeddings ≡ vectors that mathematically represent relationships bt. words in vocab



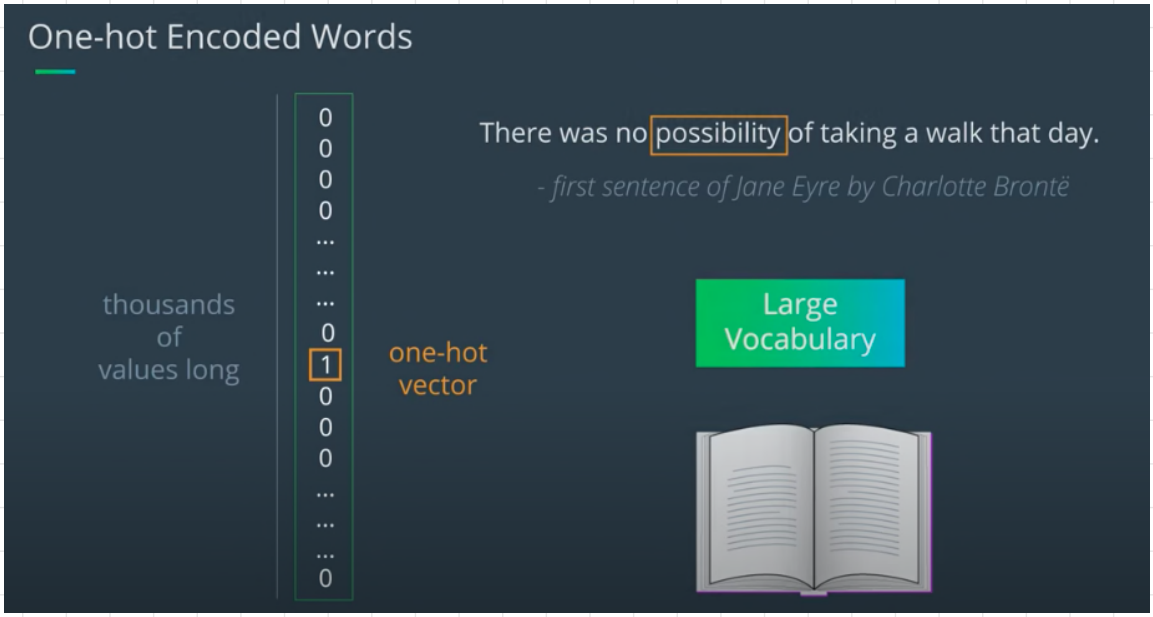
Bias: Incorrect info can be replicated!

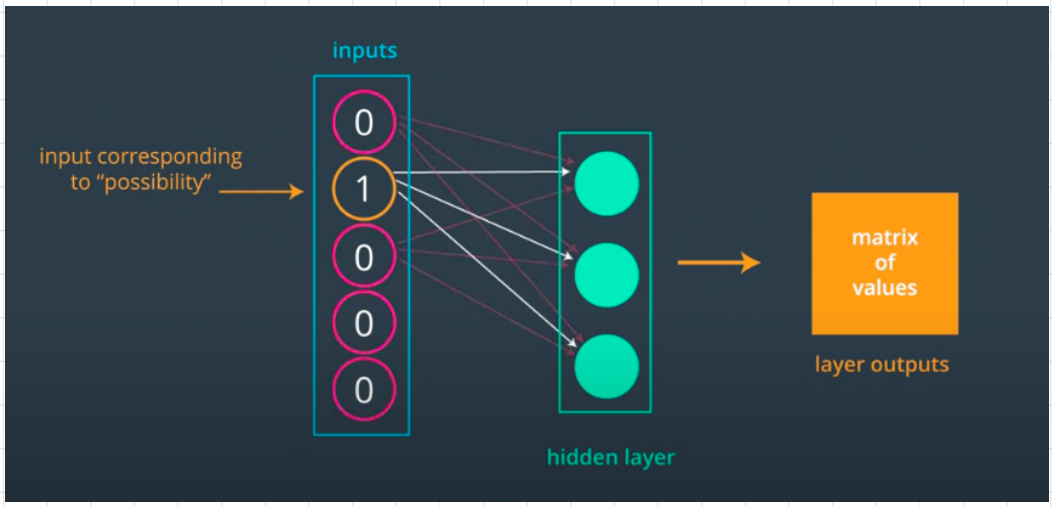
Embeddings Weight Matrix & Lookup Table.

Overview:

Embeddings represent text data as lower-dim vectors to allow NN to better learn from data

For RNN →

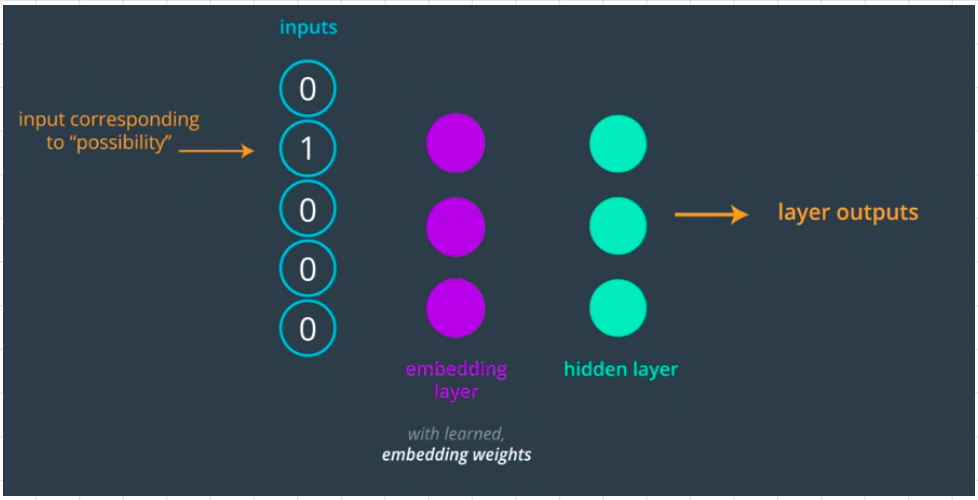


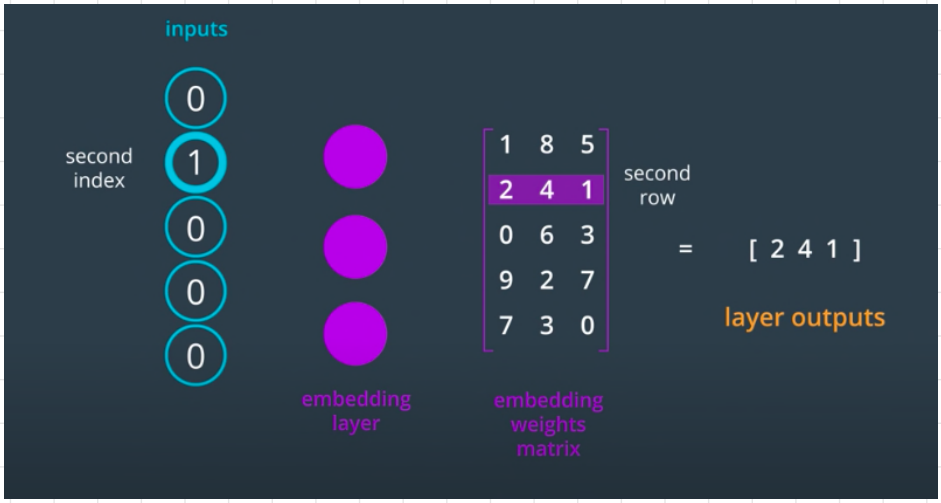


} Most are "0". Waste computation

Embedding layer

A hidden layer have trained wts make up the embedding wt. matrix



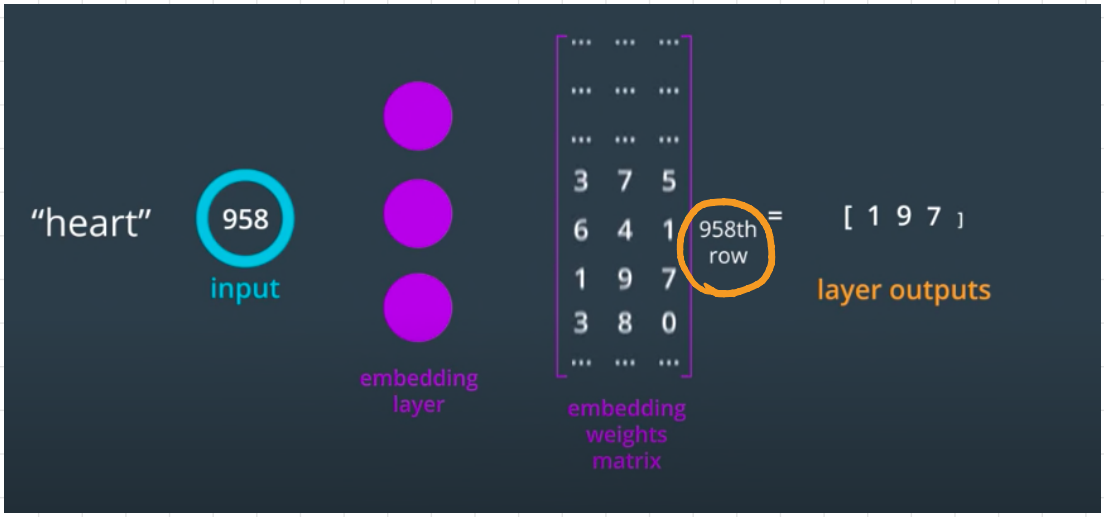


Embedding lookup

Instead of matrix multiplication, lookup the weight matrix & find the corresponding row.

E.g.

If heart is the 958th input.



# of hidden units → Embedding dim.

Massive # of words —(one-hot encode)→ Very inefficient

↓

MM results in mostly 0-valued hidden outputs

$$[0 \quad 0 \quad 0 \quad 1 \quad 0] \times \begin{bmatrix} 8 & 2 & 1 & 9 \\ 6 & 5 & 4 & 0 \\ 7 & 1 & 6 & 2 \\ 1 & 3 & 5 & 8 \\ 0 & 4 & 9 & 1 \end{bmatrix} = [1 \quad 3 \quad 5 \quad 8]$$

Solution: use number as input instead of one-hot vectors.

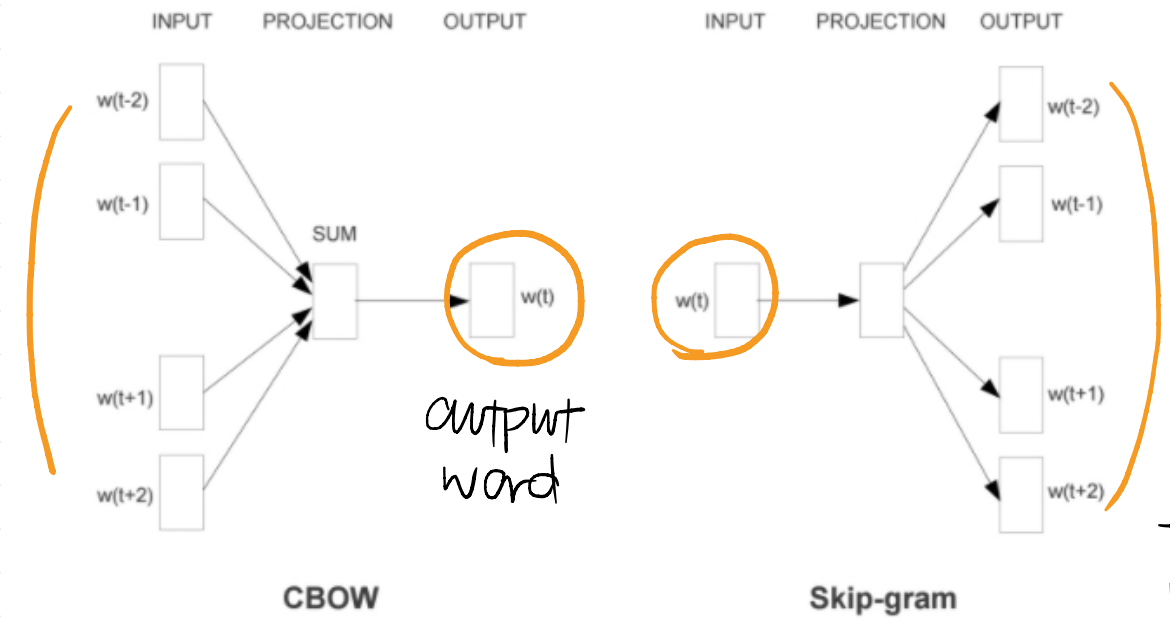
The Word2Vec algorithm finds much more efficient representations by finding vectors that represent the words. These vectors also contain semantic information about the words.

I often drink coffee in the mornings.
When I'm thirsty, I drink water.
I drink tea, before I go to sleep.

] Similar context

∴ coffee, water, tea have vectors near each other

Two architectures for implementing Word2Vec:



input context

output word

Have to specify window size. E.g.: 2 words that come before & after a word of interest.

Context doesn't involve the word of interest

Preprocessing

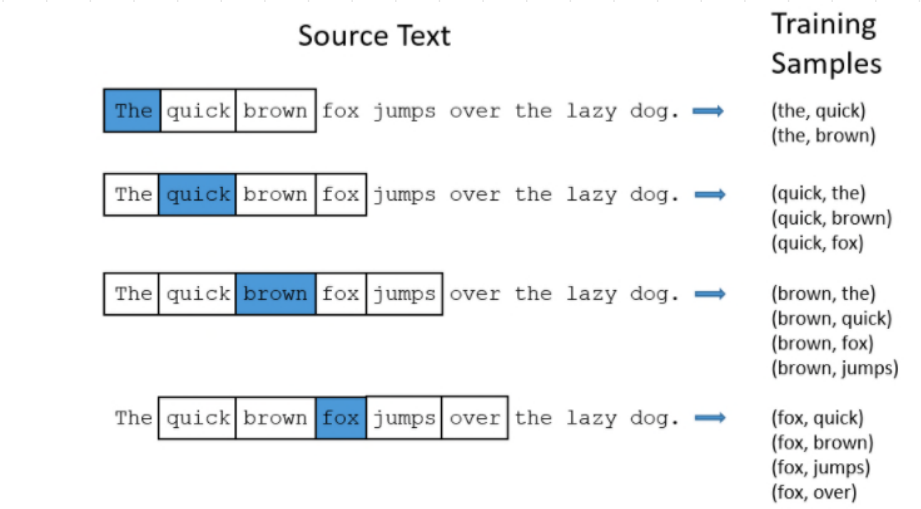
- Create a look up table for each word, giving them an index
- **Subsampling**, get rid of word w/ highest appearing frequency.

$$P(w_i) = 1 - \sqrt{\frac{t}{f(w_i)}}$$

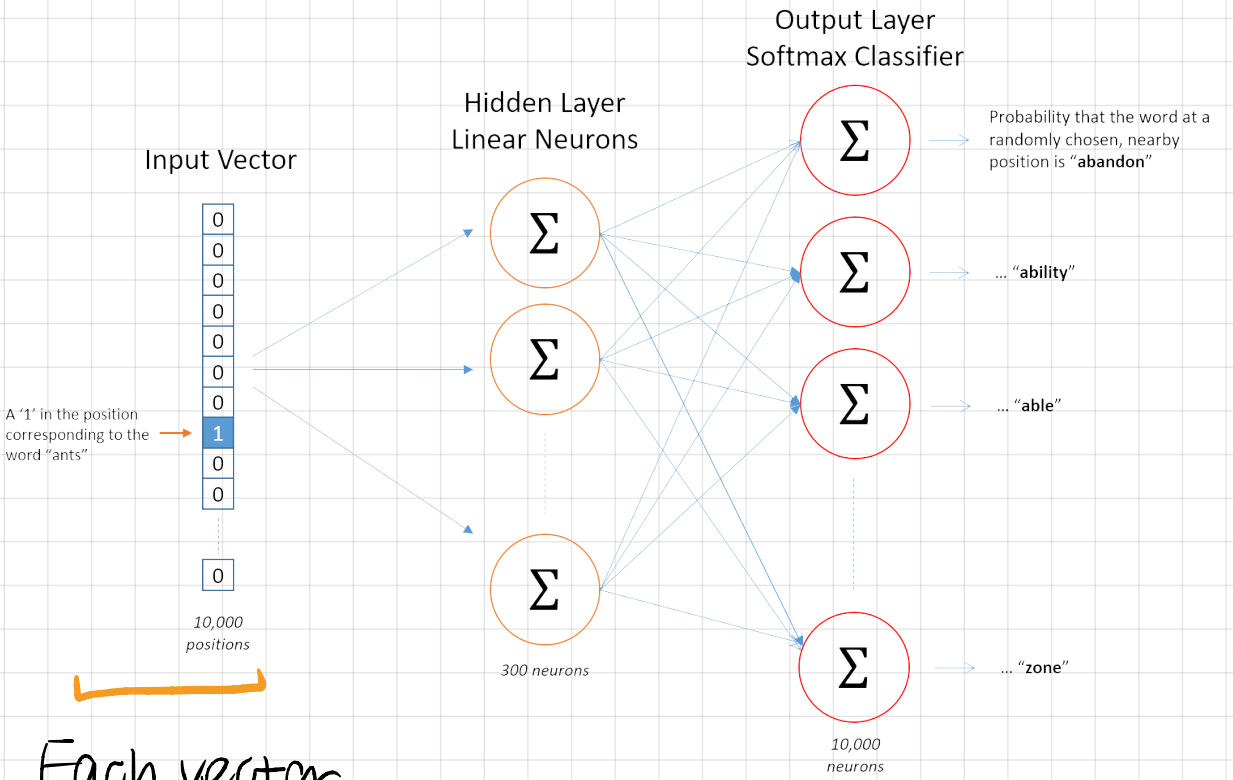
For Skip Gram Model:

Check out Chris McCormick.

Skip-Gram Model, McCormick



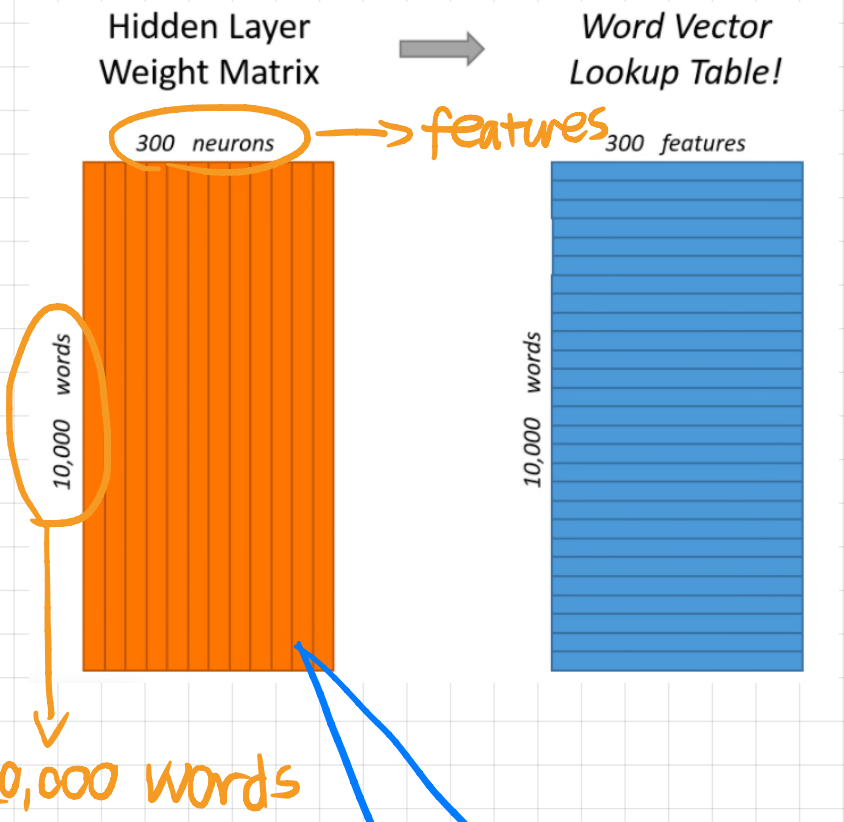
- Input a word, define a window size & let the network train on pairings bt. input & nearby words.
- Learn on statistics from the # of times each pairing shows up.



output a prob. distribution for each word, shows relation w/ input word

Each vector has 10,000 components because we have 10,000 words

Hidden Layer



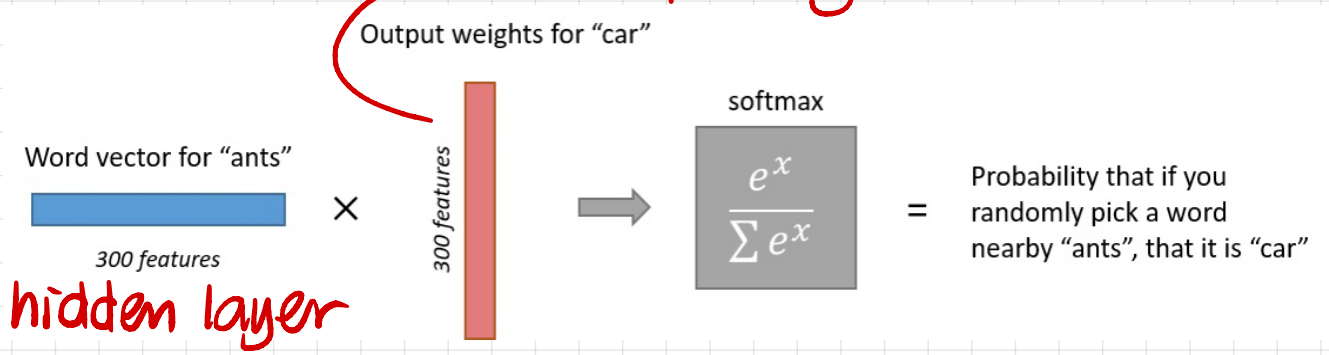
Each row is a word vector in weight matrix

10,000 words

$$[0 \quad 0 \quad 0 \quad 1 \quad 0] \times \begin{bmatrix} 8 & 2 & 1 & 9 \\ 6 & 5 & 4 & 0 \\ 7 & 1 & 6 & 2 \\ 1 & 3 & 5 & 8 \\ 0 & 4 & 9 & 1 \end{bmatrix} = [1 \quad 3 \quad 5 \quad 8]$$

Basically just a lookup table.

Output Layer



Intuition

- if two words have similar context, their **word vectors** will be similar too.

Negative Sampling.

Problems:

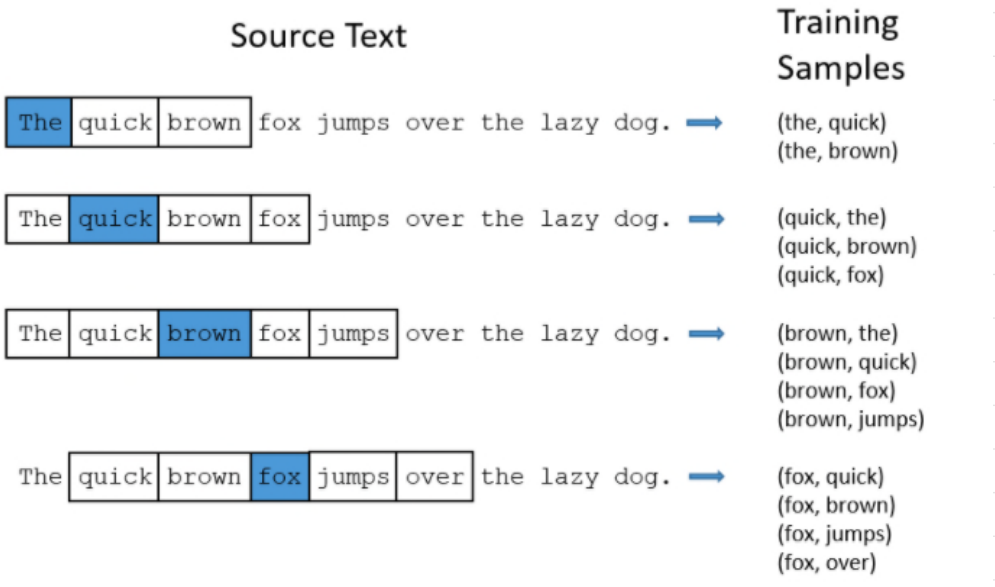
- Too many weights:

300 components × 10,000 words = 3mil. parameters

- Gradient Descent will be slow

Therefore:

- Subsampling freq. words to reduce # training examples
- Each training sample to be modified to update only a small % of the model's wts.



Here, "the" doesn't really give much context.

Each word has a prob. of getting deleted from text, depends on its frequency.

If we have a window size of 10, and we remove a specific instance of "the" from our text:

1. As we train on the remaining words, "the" will not appear in any of their context windows.
2. We'll have 10 fewer training samples where "the" is the input word.

Note how these two effects help address the two problems stated above.

Examples

- $P(w_i) = 1.0$ (100% chance of being kept) when $z(w_i) <= 0.0026$.
  - This means that only words which represent more than 0.26% of the total words will be subsampled.
- $P(w_i) = 0.5$ (50% chance of being kept) when $z(w_i) = 0.00746$.
- $P(w_i) = 0.033$ (3.3% chance of being kept) when $z(w_i) = 1.0$.
  - That is, if the corpus consisted entirely of word w_i, which of course is ridiculous.

Before:

- Each training sample adjust all the wts.

Now:

- Negative Sampling modifies a small percentage of wts

How this works?

- Say for every input word, you look at 5 "negative words" & 1 "positive word"

then, thats only 1800 output neurons, instead of 3M. → Word pair

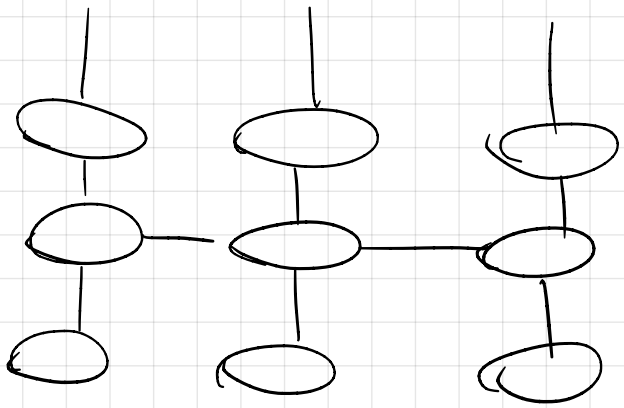
- In hidden layer, only wts for input are updated.

Selecting Negative Samples

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

Selected by Unigram Distribution

↑freq. ↑likely to be neg. samples.



Linear

LSTM 2layers

Embed