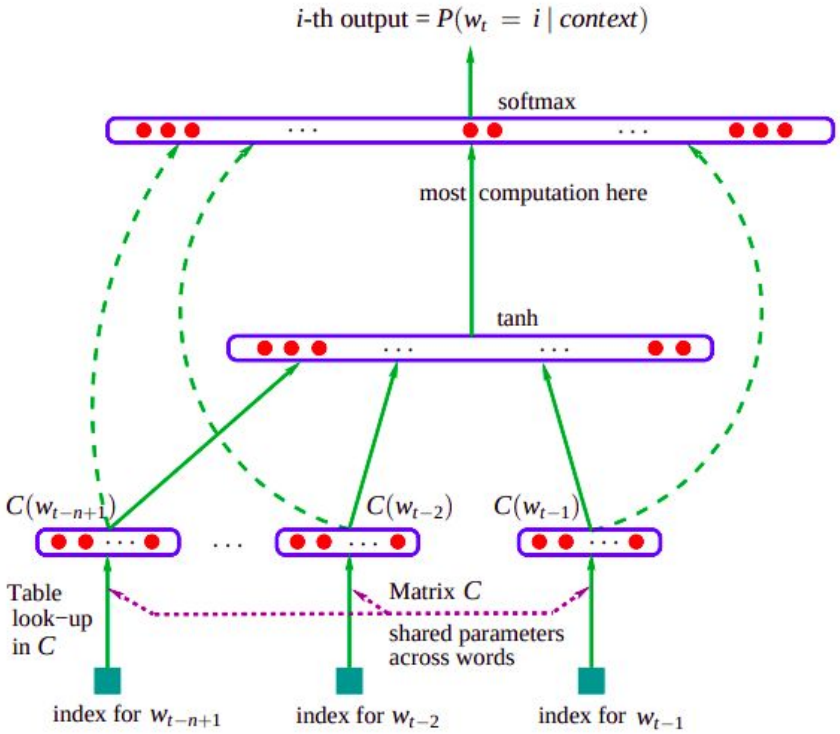
Word2Vec

Introduction to NLP, Tutorial - 3 [Part II]

“You shall know a word by the company it keeps”
~ J. R. Firth, 1957

- We want word vectors to be representative of the meaning of the word it represents. But how do we define the meaning in itself?
- Context plays an important part - a word itself may have multiple meanings, but context grounds the meaning to resolve any ambiguity.
- The idea of word2vec lies in making use of the context to capture the meaning of a word in the form of a trained “word vector” or a “word embedding” of *k* dimensions
- The embedding would in effect be representative of an aggregate of all the context the word appears in.

Previous work: Neural Network Language Model



In this architecture, a language model is trained by:

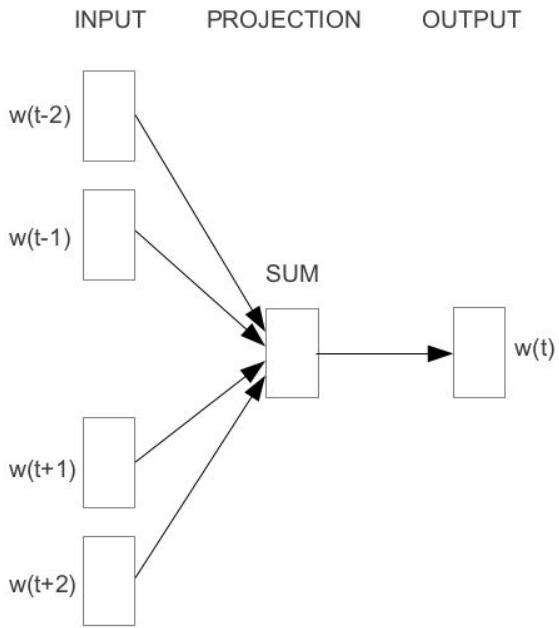
1. Projecting previous N words on an N × D dimensional space
2. Using a hidden layer of dimensionality H to softmax over vocabulary size V

Computational complexity:

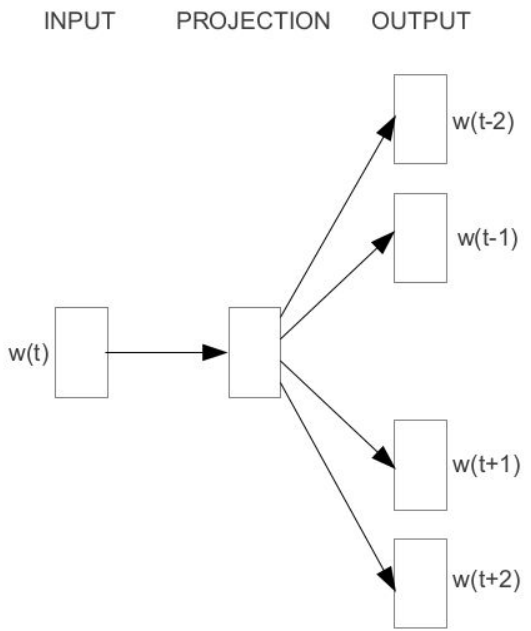
Q = N × D + **N × D × H** + H × V

The non-linear hidden layer adds a lot of complexity, although making the modeling powerful. Can we make the model simpler?

Word2vec: Continuous Bag of Words (CBOW) & Skip Gram (SG)



CBOW



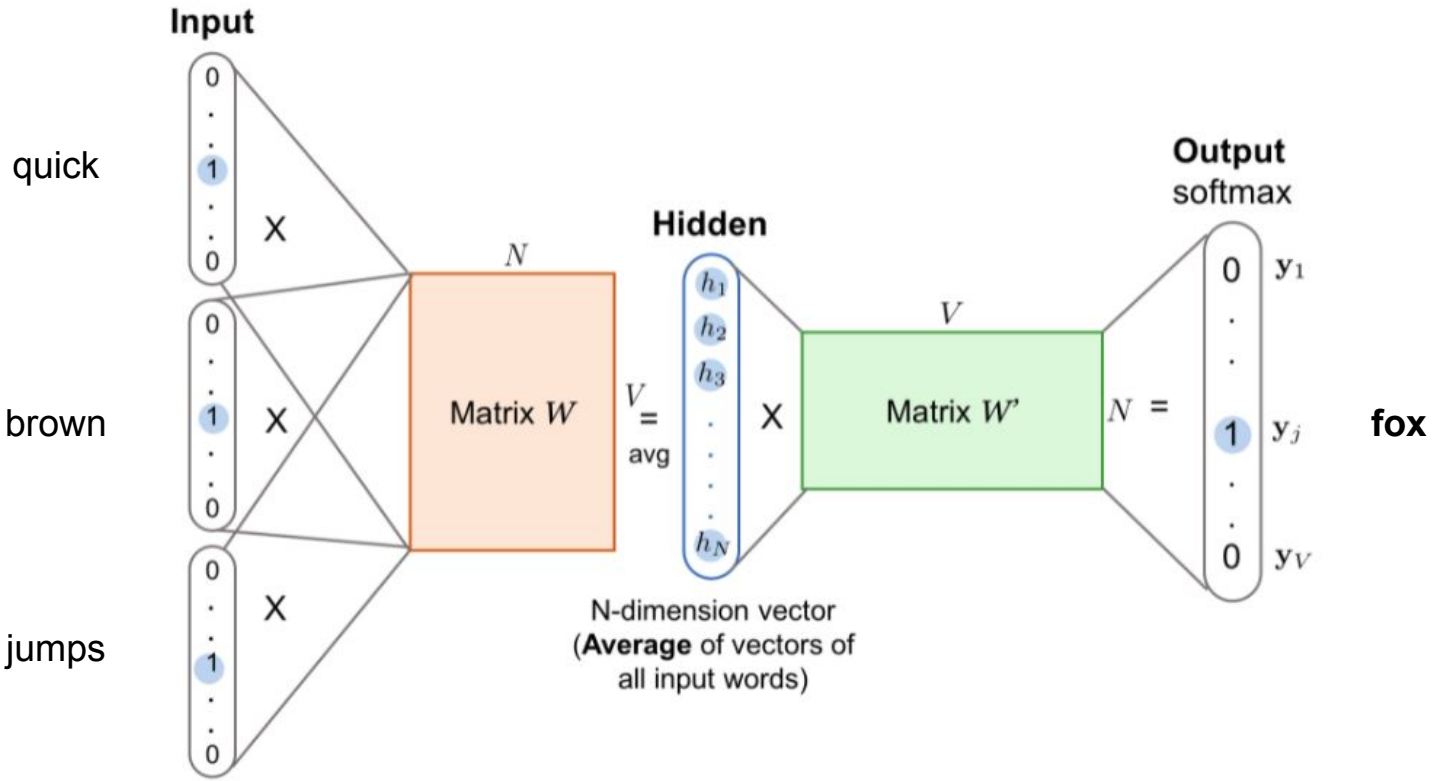
Skip-gram

We don't really need a language model.

Instead, try two ways that bring in the context of the word:

1. Using context to predict the word (CBOW)
2. Predict the context from the word (SG)

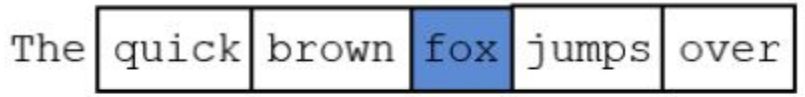
Continuous Bag of Words (CBOW)



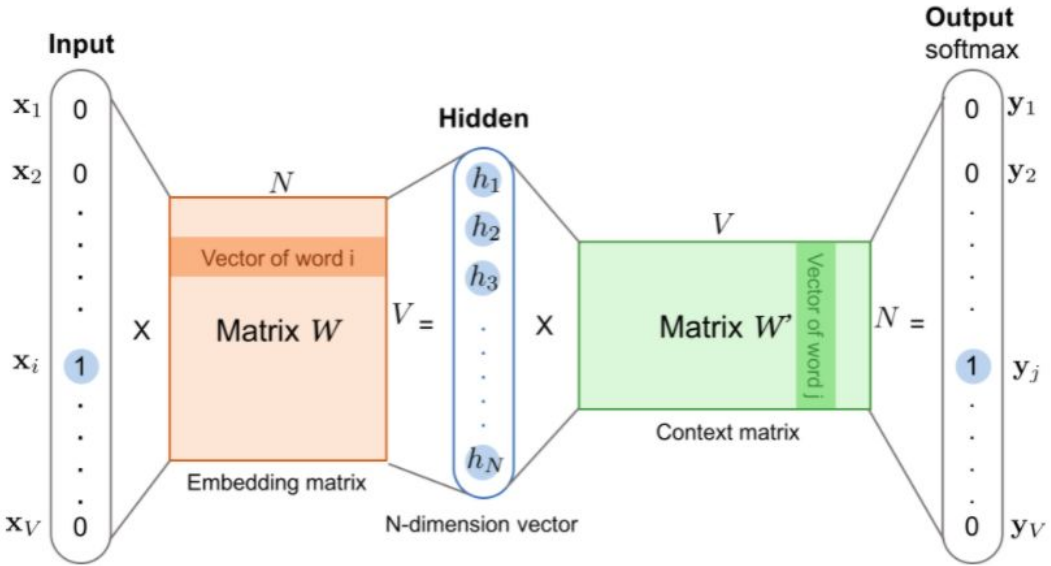
CBOW: Continued

- Training objective:
  Correctly classify the central word over the vocabulary, given the context
- Why is this still a bag-of-words model?
  We average the embeddings of surrounding words, not taking in their order
- However, note that we use a continuous distributed representation of the context (basically the averaged embeddings) to predict the centre word
- This distributed representation is more informed than mere co-occurrence or frequency-based information
- Training complexity of the vanilla architecture:
  $N \times D + D \times V$
  (V can be reduced to $\log_2(V)$ by a hierarchical softmax representation.)

Skip Gram (SG)



(fox, quick)
(fox, brown)
(fox, jumps)
(fox, over)



SG: Continued

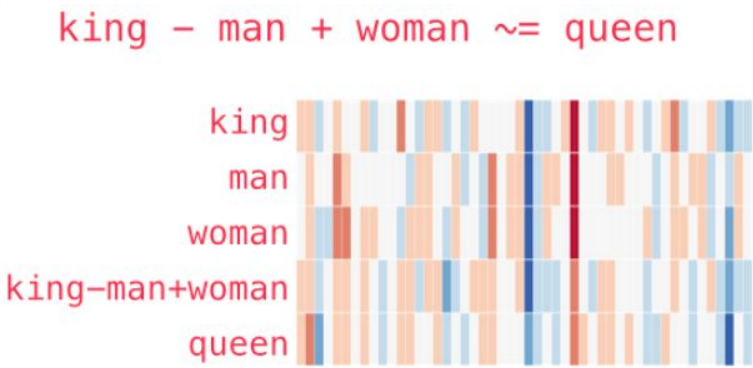
- Training objective:
  Maximize the classification of the word in context, given the central word.
- Increasing the context to predict around the word has a marginal increase in the quality of word vectors obtained, with the added cost of computation
- Since the farther words contribute lesser to the meaning of a word than the ones closer to it, it is appropriate to fix a suitable, small, window size for the context.
- Training complexity of the vanilla architecture:
  $2 \times W \times (D + D \times V)$
  (V can be reduced to $\log_2(V)$ by a hierarchical softmax representation.)

Effectiveness of word2vec representation

The quality of embeddings was evaluated by testing on the efficacy in determining syntactic and semantic relationships between the words. Essentially, the corresponding vectors were added/subtracted to find the word with closest vector to the resultant, which should correspond to the one obvious from the analogy.

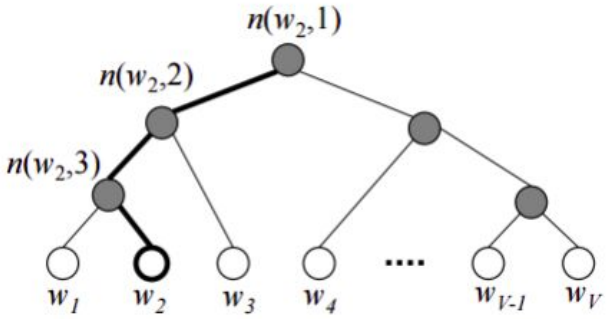
E.g.: Paris - France + Italy = Rome

Relationship	Example 1	Example 2	Example 3
France - Paris	Italy: Rome	Japan: Tokyo	Florida: Tallahassee
big - bigger	small: larger	cold: colder	quick: quicker
Miami - Florida	Baltimore: Maryland	Dallas: Texas	Kona: Hawaii
Einstein - scientist	Messi: midfielder	Mozart: violinist	Picasso: painter
Sarkozy - France	Berlusconi: Italy	Merkel: Germany	Koizumi: Japan
copper - Cu	zinc: Zn	gold: Au	uranium: plutonium
Berlusconi - Silvio	Sarkozy: Nicolas	Putin: Medvedev	Obama: Barack
Microsoft - Windows	Google: Android	IBM: Linux	Apple: iPhone
Microsoft - Ballmer	Google: Yahoo	IBM: McNealy	Apple: Jobs
Japan - sushi	Germany: bratwurst	France: tapas	USA: pizza



Improvements to vanilla word2vec

1. Phrase vectors instead of word vectors for idiomatic phrases like “Boston Globe” that are not compositions of the constituent words
2. Subsampling frequent words to counter the imbalance between rare and frequent words, and decrease training time $P(w_i) = 1 - \sqrt{\frac{t}{f(w_i)}}$
3. Hierarchical softmax



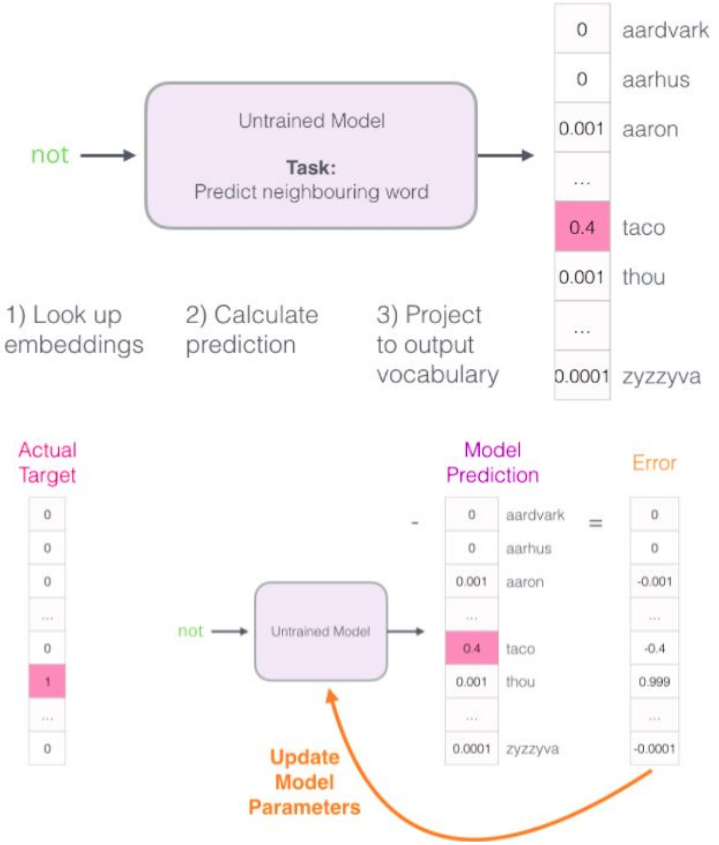
4. **Negative Sampling**

Issues in word2vec that Negative Sampling can handle

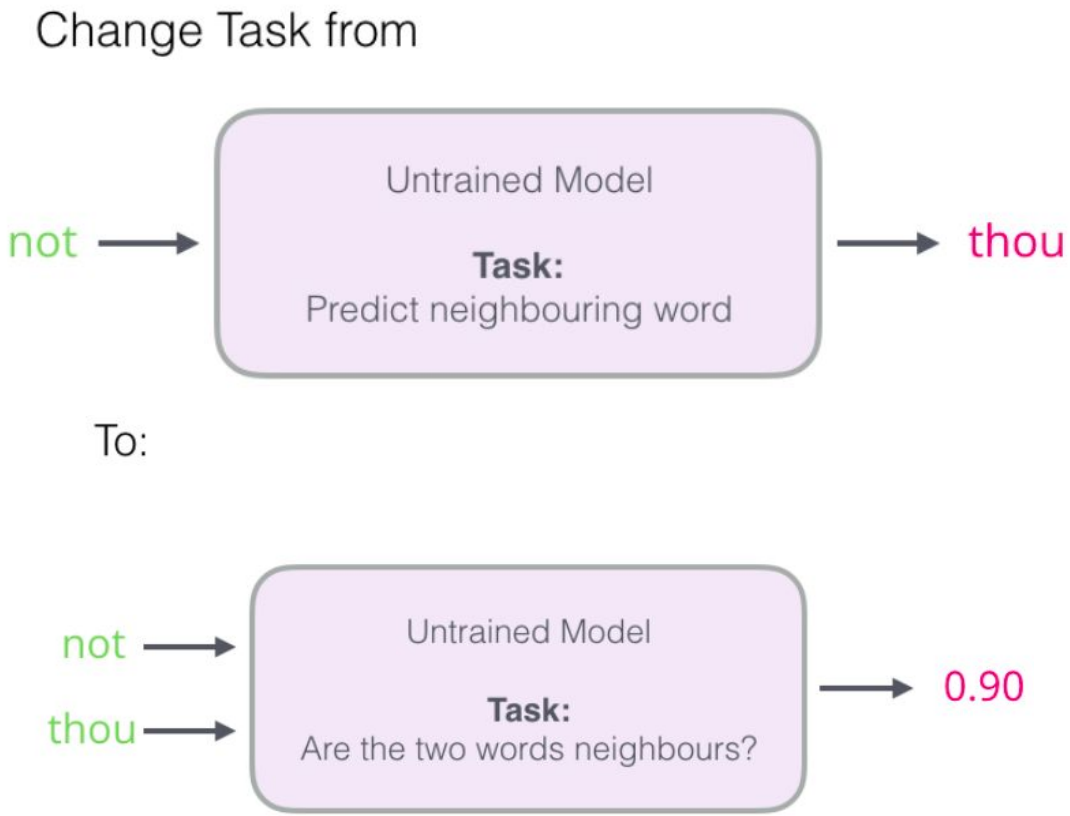
Since we output the probability distribution over the entire vocab, naturally we have:

1. Larger training time
2. Computation of gradients over all the words
3. Larger memory consumption in storing the gradients computed

Negative Sampling seeks to provide a good approximation to the process by using only a sample of negative words instead of all of the remaining vocab.



Negative Sampling: Intuition with Skip-Gram



Negative Sampling: Objective

- Vanilla training objective for word2vec:

$$p(w_O|w_I) = \frac{\exp\left({v'_{w_O}}^\top v_{w_I}\right)}{\sum_{w=1}^{W} \exp\left({v'_w}^\top v_{w_I}\right)}$$

- Inspired by Noise Contrastive Estimation (NCE), we minimize the following objective in training:

$$\log \sigma({v'_{w_O}}^\top v_{w_I}) + \sum_{i=1}^{k} \mathbb{E}_{w_i \sim P_n(w)} \left[\log \sigma(-{v'_{w_i}}^\top v_{w_I})\right]$$

- Here, $P_n(w)$ is the noise distribution drawn from the vocabulary sampled from the unigram distribution raised to its $3/4^{th}$ power.

Thank you!