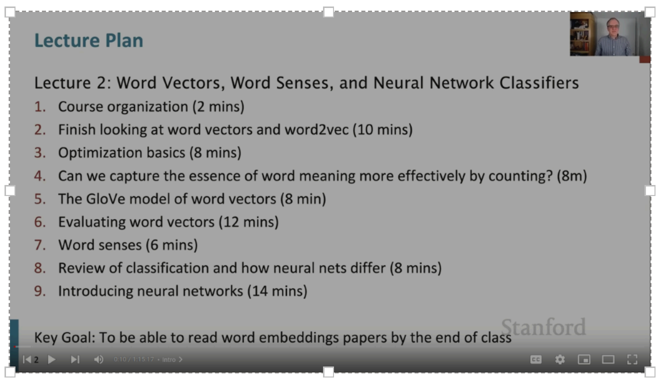
Notes:

04 February 2024 07:48

Lecture Outline:



Review of word2vec:

- Word2vec learns to group words by similarities just by predicting the words around a word (center word).

- Word2vec is a BAG OF WORDS MODEL:
  - Word2Vec does not distinguish word based on where they appear in the window? I.e. It is not considered how far/near a window word is to the center word?

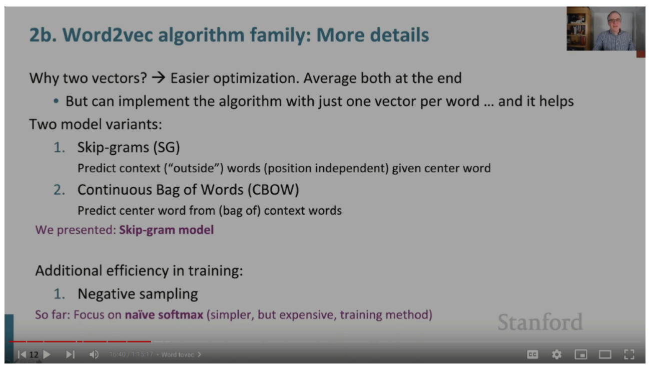
- How about finding a model:
  - That assign high probabilities to all the words that are always used in context all over the corpus? (irrespective of context window of an algorithm)

Word2Vec algorithm family:

2 types of word2vec algorithms:

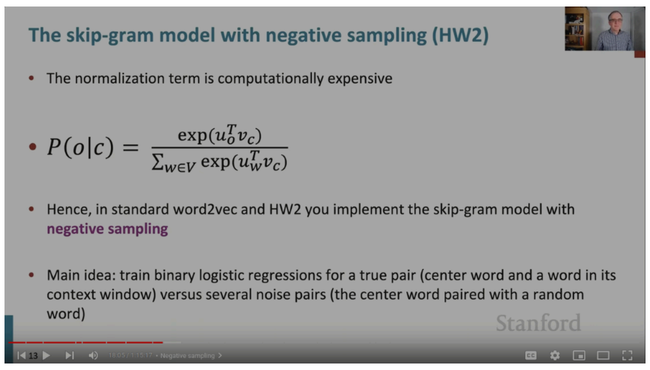
- Skip-gram: We have seen skip-gram (SG) version of word2vec: I.e. predicting window_words given a center_word.

- Continuous-Bag-of-Words: sort of inverse of SG. Predicts center word given window_words.

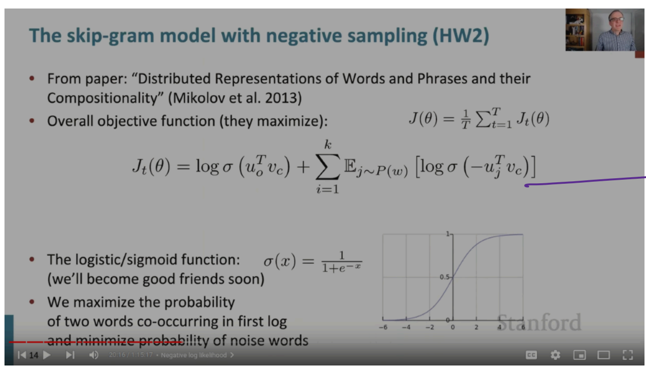


Negative Sampling Training Strategy:

- Training of Word2vec with cross entropy is expensive as the denominator has to be calculated over all the words in the CORPUS.



- Negative Sampling:
  - Train the model with a true pair (center word and true window word) and a negative pair (center word and random non-window word from corpus)



v_c → center word
u_o → window (outside word) } True Pair
u_j → Negative sampled word from Corpus

∴ maximize $\text{Log}(\text{sigmoid}(u_o^T \cdot v_c))$ &

minimize $\text{Log}(\text{sigmoid}(u_j^T \cdot v_c))$

as sigmoid is symmetric function, to minimize, we take negative of

  - Sampling words from Unigram distribution:
    - e.g. in a billion word corpus, a word occurs 100 times then 100/1 billion would be the unigram probability of that word.

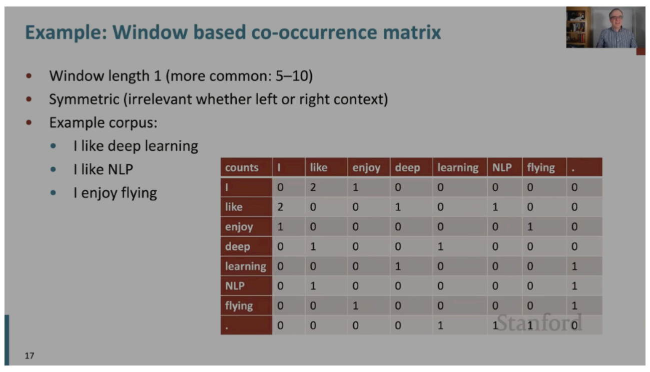
$P(w)$ $U(w)^{3/4}$ — $U(w)$ → unigram prob.

$P(w_j) = \frac{\quad}{z}$ $\quad z \rightarrow$ total corpus.

why 3/4 → helps to sample commonly occuring & rare words evenly.

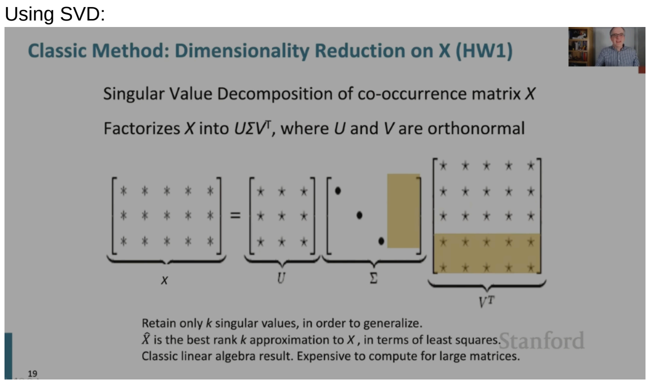
Co-Occurrence matrix:

- Idea: If we want to find out word occurring together in the corpus, why not build a matrix that counts the (unique) words that occur together?
- Intuition: from the matrix, embeddings of the words
    - I – will have like and enjoy words nearby
    - Deep – will have learning associated with it
- Two option to build co-occurrence matrix:
    - Window-based or full document based



counts	I	like	enjoy	deep	learning	NLP	flying	.
I	0	2	1	0	0	0	0	0
like	2	0	0	1	0	1	0	0
enjoy	1	0	0	0	0	0	1	0
deep	0	1	0	0	1	0	0	0
learning	0	0	0	1	0	0	0	1
NLP	0	1	0	0	0	0	0	1
flying	0	0	1	0	0	0	0	1
.	0	0	0	0	1	1	1	0

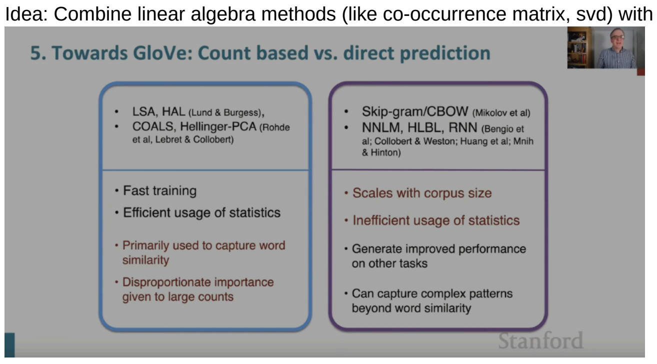
- Cons:
    - Dimensions are proportional to num_words in corpus
    - Mostly sparse -> models are less robust.
- How to make them more useful? -> dimensionality reduction.
    - Using SVD:



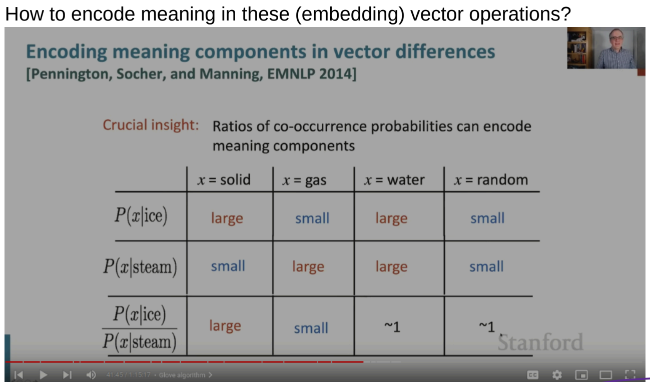
- Using SVD on raw co-occurrence matrix does not help much
    - Some hacks are used to improve upon this situation e.g. the most prominent being the Kohls model.
    - Ref of Kohls model: An Improved Model of Semantic Similarity Based on Lexical Co-Occurrence, 2005

The GloVe algorithm

- Idea: Combine linear algebra methods (like co-occurrence matrix, svd) with learning based methods (Skip-gram/CBoW)



- How to encode meaning in these (embedding) vector operations?



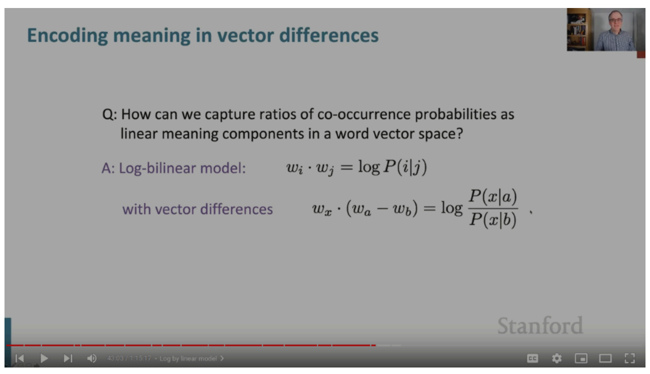
	x = solid	x = gas	x = water	x = random
$P(x\|\text{ice})$	large	small	large	small
$P(x\|\text{steam})$	small	large	large	small
$\frac{P(x\|\text{ice})}{P(x\|\text{steam})}$	large	small	~1	~1

Linear meaning component = King − Man + woman = Queen

Bilinear function:

$f(x, y)$ is linear in x when y fixed & vice versa

To do this, hypothesis was to make dot product of two vectors similar to it's conditional log prob.

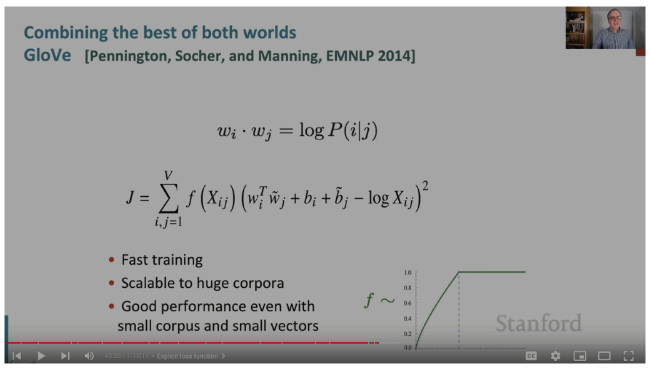


$w_i \cdot w_j \simeq \log(P(i|j))$ → it dot product of w_i & w_j $\simeq$ Log (Prob. (i given j))

& if $w_j = w_a - w_b$ then. diff bet^n two vectors → similarity or → dissimilarity

$w_x \cdot w_j = \log P(i|j)$

$w_x \cdot (w_a - w_b) = \log \left(\frac{P(x|a)}{P(x|b)} \right)$



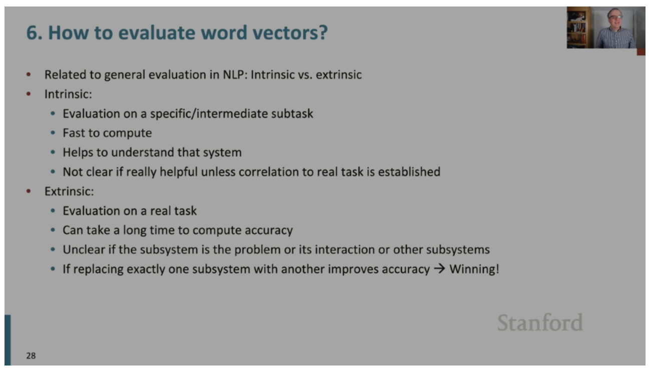
$w_i . w_j = \log P(i|j)$

Glove: vector similarity (i.e) dot product $\simeq$ prob. of co-occurrence.

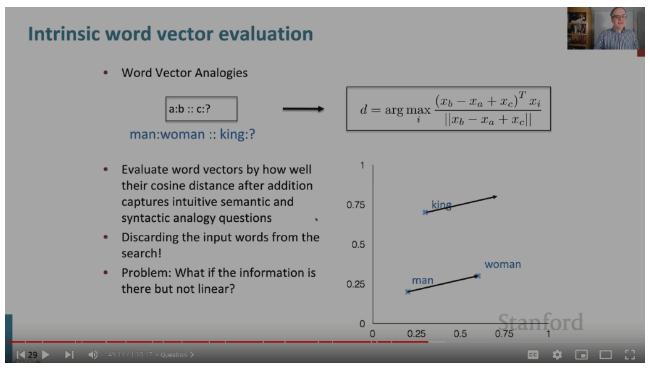
$f(x_{i,j}) \rightarrow f(\text{small\_val}) \simeq$ large num.

$\therefore f(x_{i,j}) \rightarrow$ helps to balance the rarely & commonly occuring words

Evaluation of Word Vectors



Intrinsic Evaluation:



Extrinsic Evaluation:

- Using the word embeddings for a downstream task e.g. Named-entity-recognition and then evaluating the performance of a model on the benchmark of that task