

Learnings:

Problems of words appearing in high frequency:

There is a problem on words like: the, and etc, due to their higher frequency they have bigger word vectors, resulting in bigger probabilities. This is true and correct but if we want to show their semantic properties better we have to take care of this high frequency effect. One of the crude ways of solving this problem is by locking the frequency's effect.

Stochastic gradient descent:

Usually our corpus has billions of words we cannot evaluate for the whole corpus in every step of the gradient descent. So what we take is a window (importantly a different window in every step) and calculate the gradient within it (remember: for better results do this in mini batches of 32 or 64).

Problem with stochastic gradient descent:

We might not even encounter many of the words when using windows. The windows might not hold enough variety of words and we might end up with word vector matrices with most of the rows zeroes (initial values). We could solve this by using sparse matrix matrix functions to selectively update certain rows of u and v , or we can use hashing.

WordToVec models:

Skip-Grams model:

In this model we predict the context word given the center word.

Continuous bag of words model:

In this model a bag of context words is used to predict the center word.

Negative sampling optimization:

In the calculation of the probability the denominator of the probability function ,

$$\bullet P(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{w \in V} \exp(u_w^T v_c)}$$

The denominator is a dot product iterated over the full corpus. Therefore the calculation takes too much time. We fight this by using binary logistic function on a center word and a context word in its word window and words outside its word window. This is called skip gram model with negative sampling.

Choosing random words for negative sampling:

In negative sampling we choose words randomly, this means words which appear more frequently have a more probability of appearing. This is reduced by using unigram distribution.

- $P(w) = U(w)^{3/4} / Z$,
the unigram distribution $U(w)$ raised to the 3/4 power
(We provide this function in the starter code).
- The power makes less frequent words be sampled more often

Z is a normalization term.

Cost functions:

1. When we maximize:

$$J_t(\theta) = \log \sigma(u_o^T v_c) + \sum_{i=1}^k \mathbb{E}_{j \sim P(w)} [\log \sigma(-u_j^T v_c)]$$

↓
↓

Cases inside context window
Cases outside

Notice the negative sign for cases outside the window.

2. When we minimize:

(basically just a minus b4 it)

$$J_{neg-sample}(o, v_c, U) = -\log(\sigma(\underline{u_o^T v_c})) - \sum_{k=1}^K \log(\sigma(\underline{-u_k^T v_c}))$$

Notice the negative sign for cases outside the window.

Tricks used in multiple occurrences/ hacks to x:

When multiple occurrences of a word is present instead of just counting them we can:

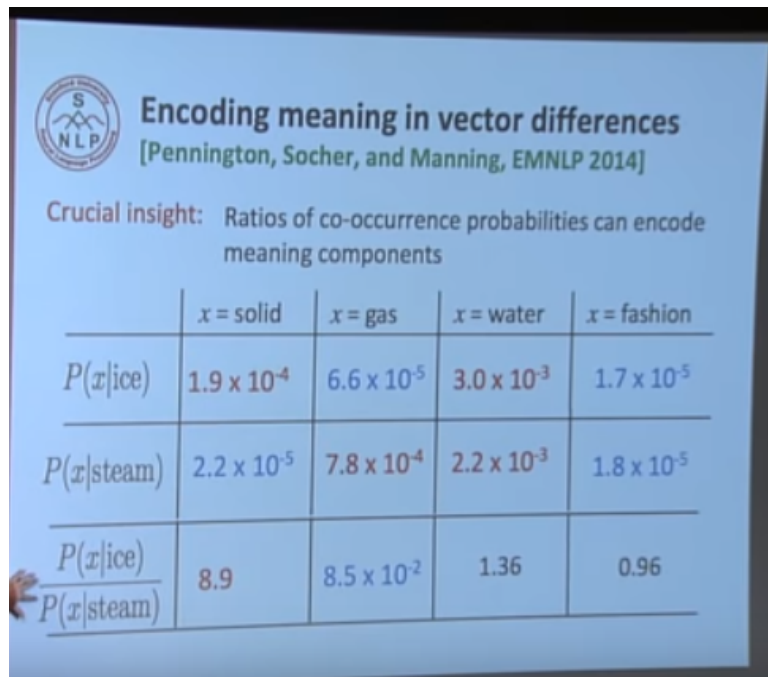
- log scale them
- Do a ceil function
- Count words that occur closer as more

to get better results.

Count based vs. direct prediction:

In count based prediction there is fast training and we use the concepts of statistics better, while on the other hand direct prediction. But the disadvantage of count based prediction is that it is mainly used for finding similarity and not complex relations.

Ratios of co-occurrence probability for meaning:



Encoding meaning in vector differences
[Pennington, Socher, and Manning, EMNLP 2014]

Crucial insight: Ratios of co-occurrence probabilities can encode meaning components

	$x = \text{solid}$	$x = \text{gas}$	$x = \text{water}$	$x = \text{fashion}$
$P(x \text{ice})$	1.9×10^{-4}	6.6×10^{-5}	3.0×10^{-3}	1.7×10^{-5}
$P(x \text{steam})$	2.2×10^{-5}	7.8×10^{-4}	2.2×10^{-3}	1.8×10^{-5}
$\frac{P(x \text{ice})}{P(x \text{steam})}$	8.9	8.5×10^{-2}	1.36	0.96

The ratio of co-occurrence probability gives us a dimension of meaning. A small number represents the opposite meaning of a big number and numbers close to one represent independence with the meaning.

Now how do we make the ratios as linear meaning components ?

Simple! Just take log and ratios become differences.

Make the dot products such that it gives log of probability.

Q: How can we capture ratios of co-occurrence probabilities as linear meaning components in a word vector space?

A: Log-bilinear model: $w_i \cdot w_j = \log P(i|j)$

with vector differences $w_x \cdot (w_a - w_b) = \log \frac{P(x|a)}{P(x|b)}$

The cost function:

$$J = \sum_{i,j=1}^V f(X_{ij}) (w_i^T \tilde{w}_j + b_i + \tilde{b}_j - \log X_{ij})^2$$



Where b is a bias term and the f function has a cap to avoid over representation of high frequency words.

The above given method is called the

GloVe

Method .

Evaluation of word vectors:

Intrinsic:

In intrinsic evaluation a part of the whole or a subtask is evaluated. This method is hence easier to get results with lesser computation but its results might not show concretely the real world effectiveness.

Extrinsic:

In this method, evaluation is done on real world scenarios but it is much harder to acquire results and takes a lot of computation. If the result is bad it still doesn't explicitly mention where the problem is.

Important points:

1. All the major deep learning packages store each word in a row:

The slide is titled "Word2vec parameters and computations". It illustrates the process of calculating word probabilities in a Word2vec model. On the left, there are two 4x4 grids of dots representing matrices U (labeled "outside") and V (labeled "center"). To the right of these is a 4x1 column of dots representing the dot product $U \cdot v_4^T$ (labeled "dot product"). Further right is another 4x1 column of dots representing the softmax output $\text{softmax}(U \cdot v_4^T)$ (labeled "probabilities"). Handwritten blue arrows point from the first and second rows of the V matrix to the first and second rows of the dot product vector, with labels w_1 and w_2 written in red. Below the matrices, the text "Same predictions at each position" is written. At the bottom of the slide, a presenter is visible, and the text "We want a model that gives a reasonably high probability estimate to *all* words that occur in the context (fairly often)" is displayed.

Not the other way around.

2. Why do we use 2 vectors? We use two vectors as after optimization the method with two vectors is faster. We can just average both at the end. If you don't want to you can actually just use one vector during computations.
3. For GloVe vectors the the good window size is 8 words around the center word.
4. The good number of dimensions for a word vector is 300.
5. Asymmetric context (having words only to the left of the center word as context word) are not as good.
6. A thought: Words have different meanings in different parts of the world. Example: mate. So maybe when making word vectors we could make different word vectors for different countries using text from their respective cultures.
7. Or do what the instructor proposed, for words with multiple meanings, make pseudo words and replace them , example jaguar, with jaguar1, jaguar2 etc. Now we get all the representation of words. If we don't do this we just get a superposition (a weighted average) of all the word vectors as the resulting word vector.
Example:

$$v_{\text{pike}} = \alpha_1 v_{\text{pike}_1} + \alpha_2 v_{\text{pike}_2} + \alpha_3 v_{\text{pike}_3}$$

Where $\alpha_1 = \frac{f_1}{f_1 + f_2 + f_3}$, etc., for frequency f

Doubts:

1. Why do we need negative sampling in skip grams model? The main disadvantage is how big the denominator is, but this is already countered by using only small but different samples in the stochastic gradient descent model. Why should we implement negative sampling which takes only a few words into account in which a lot more data of the corpus might not be used. Is the calculation that computationally hard that we need both negative sampling and stochastic gradient descent ?
2. Did not understand what the underlined notation represented.

$$J_t(\theta) = \log \sigma(u_o^T v_c) + \sum_{i=1}^k \mathbb{E}_{j \sim P(w)} [\log \sigma(-u_j^T v_c)]$$