CS224n: NLP with Deep Learning

Lecture 2: Word Vectors and Word Senses

Word2vec

Our usual example

- King - Man ~= Idea of Kingship without the Man Part
- +Woman = add Woman Idea to it
- => Get Queen

Plot words on a scatter plot:

for word, (x,y) in zip(words, twodim): plt.text(x+0.05, y+0.05, word)

Word2vec : Vectorization

We have 2 big matrices:

- 1 that represents every outside word's vector U
- 1 which represents every center word's vector V

$$U = \begin{bmatrix} [outside\ word\ vector\ 1] \\ \vdots \\ [outside\ word\ vector\ n] \end{bmatrix}$$

$$V = \begin{bmatrix} [center\ word\ vector\ 1] \\ \vdots \\ [center\ word\ vector\ n] \end{bmatrix}$$

- We multiply U by a center word v_4

$$U \cdot v_4 = \begin{bmatrix} [outside\ word\ vector\ 1] \\ \vdots \\ [outside\ word\ vector\ n] \end{bmatrix} \cdot v_4 = \begin{bmatrix} [similarity\ (u_1, v_4)] \\ \vdots \\ [similarity\ (u_n, v_4)] \end{bmatrix}$$

- We apply softmax to get a vector of probabilities

$$softmax(U \cdot u_4)$$

Remark 1

The outside words that are predicted will always be the same !!

House house [center word] house house if P(house|center word) = max P(context word|center word)

This means that our model will give a reasonably high probability estimate to all words that occur in the context

Remark 2

The words 'and', 'the', 'of', ... will have very high frequency with all the other words

Remark 3

The 2D-projections of the word clouds are very misleading
In very high dimensional space, a word can be close to lots of other words

---

Optimization: Gradient Descent

Objective:

- Minimise cost function $J(\theta)$

Idea:

- For current value of θ, calculate the gradient of $J(\theta)$
- Take a small step in the direction of negative gradient

$$\theta^{new} = \theta^{old} - \alpha \nabla_\theta J(\theta)$$

where α is the learning rate

Stochastic Gradient Decsent

Problem:

- $J(\theta)$ is a function of all windows in the corpus!!

Solution:

Stochastic Gradient Descent !

- Repeatedly sample windows
- Update our parameters after going through each window
- The parameters are updated with amazingly noisy gradient, but it doesn't matter too much
- It allows us to go much quicker

Remark 1

Choose mini-batch size of 32, 64, or other powers of 2, as they allow to make the most out of parallelization

Remark 2

- In each window, we have a certain number of words $2m + 1$
- Our parameter vector θ is in $\mathbb{R}^{2dV}$, which is much bigger
- Hence, $\nabla_\theta J(\theta)$ is a very sparce matrix

=> **Idea:**

Only update the word vectors that appear in our window!

Why 2 vectors for each word?

- It is easy to optimize
- We just average them to get a unique word vector in the end

2 Model variants

1. Skip-gram: Predict outside words (position independent) given center word
2. Continuous Bag of Words: predict center word from a Bag of context words

The results we get via these 2 methods are quite similar, as the dot product is symmetric

Negative Sampling

Idea:

Train binary logistic regression for a true pair (center word & word in its context window)
VS
Several noise pairs (center word & a random word) = Negative samples

- Maximize probability that real outside word appears
- Minimize probability that random words appear around center word

We sample these words using the Unigram distribution to determine their probability of being sampled

$$P(w) = U(w)^{3/4}/Z$$

- The 3/4 power helps to make the most frequent words appear more often, and the less frequent words appear more
- Z is a normalization factor

Remark:

The sigmoid function is like a binary case for the softmax function: maps our values to a probability distribution in [0,1]

Count-based methods

We could also count co-occurences of words within a certain window...
Why don't we just do that?

We could create a matrix of co-occurrences

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

2 words would be 'similar' if their co-occurence vectors are similar

Reduced SVD

Singular Value Decomposition

$$X = U * \Sigma * V^T$$

- Σ is a diagonal matrix (can be rectangular if X not square)
  Its diagonal values are in decreasing order
- U, V are square matrices

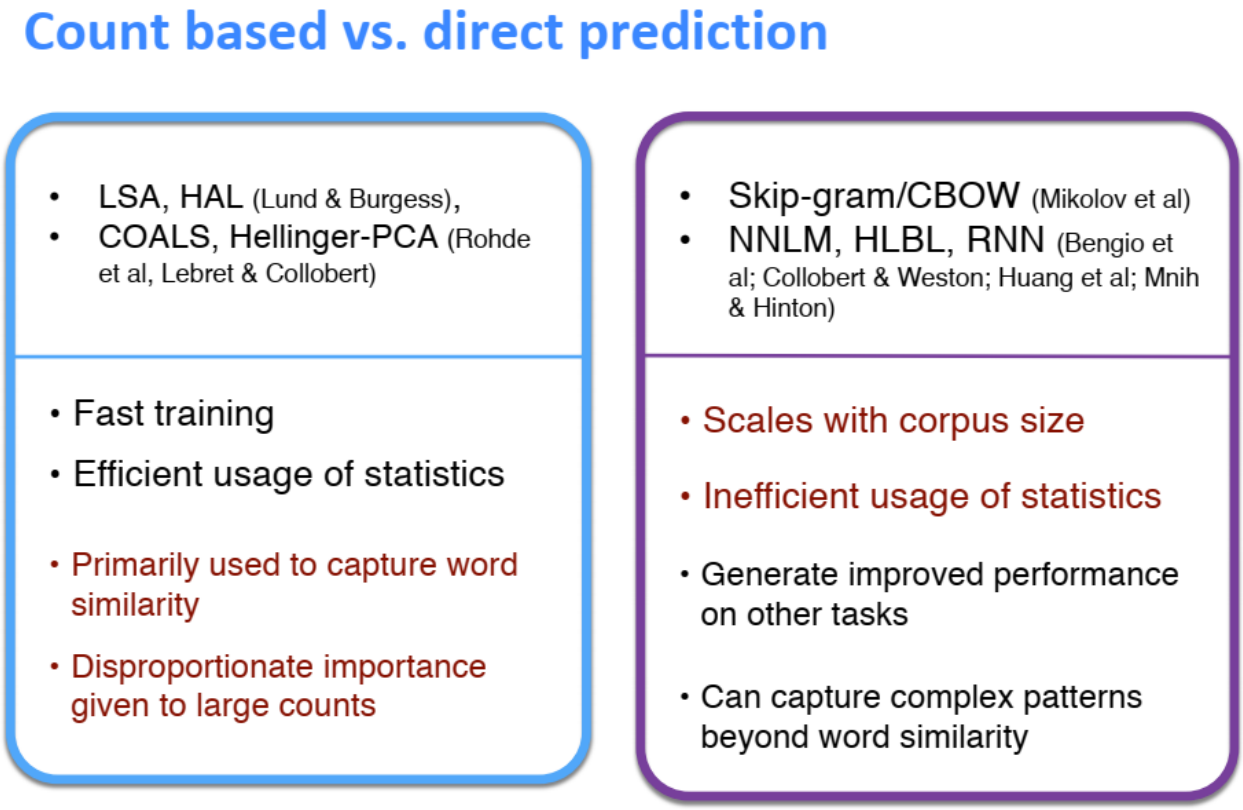
Reduced version:
We remove the dimensions with the lowest diagonal values for Σ
The corresponding column of U, and line of V^T are removed as well
We then have k dimensions left

We the get the best rank k approximation to X, in terms of least squares

Improvement ideas (*Rohde, 2005*)

- Windows that give more importance to closer words:
  Same as in word2vec, we sample closer words more often

Comparison of the 2 methods



GloVe

General Idea

You want to have components of meaning to be linear operations in a vector space !
In particular, ratios of co-occurrence probabilities

We want to have dot-product to be equal to log(Probability)

$$w_i \cdot w_j = logP(i|j)$$

so that:

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

Thus,

- The objective function is the squared difference between the dot-product and the log of the co-occurence probabilities
- It is complexified a bit by adding bias terms for both words

- The f function is used to limit the influence of very common words pairs

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

Evaluating word vectors

Intrinsic VS Extrinsic evaluation

Intrinsic

- Evaluation on a specific / intermediate task
- Fast to compute

Extrinsic

- Evaluation on a real task that humans like to use
  (web search, question answering, phone dialogue system...)
- Can take a long time to compute accuracy
- Unclear where the problem is:
  - The subsystem?
  - Its interaction with other subsystems?

On the Dimensionality of Word Embeddings

There seems to be a blip at 200-300, for the dimension of the embeddings, that seems to optimize performance

The performance stays flat, even when we continue to increase dimensionality up to infinity

Using Wikipedia data seems to work better than a news corpus

Word Senses

Words can have lots of meanings!
(especially common words, and words that have existed for a long time)

First simple idea

- Cluster word windows around words: 1 word vector for each sense

Different senses using linear superposition

Different senses of a word reside in a weighted sum of the vectors of each sense

$$v_{pike} = \alpha_1 v_{pike_1} + \alpha_2 v_{pike_2} + \alpha_3 v_{pike_3}$$

where:

$$\alpha_1 = \frac{f_1}{f_1 + f_2 + f_3}$$
$$f_i = frequency\,(word\,i)$$

Because we have so many dimensions, and as the words vectors for each sense are sparse, the results we get are quite good !!

Thus, we can get back the vectors of the senses using the weighted summed vector !!