

Notes:

04 February 2024 07:48

Lecture Outline:

Lecture Plan

Lecture 2: Word Vectors, Word Senses, and Neural Network Classifiers

1. Course organization (2 mins)

2. Finish looking at word vectors and word2vec (10 mins)

3. Optimization basics (8 mins)

4. Can we capture the essence of word meaning more effectively by counting? (8m)

5. The GloVe model of word vectors (8 min)

6. Evaluating word vectors (12 mins)

7. Word senses (6 mins)

8. Review of classification and how neural nets differ (8 mins)

9. Introducing neural networks (14 mins)

Key Goal: To be able to read word embeddings papers by the end of class

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.

2b. Word2vec algorithm family: More details

Why two vectors? → Easier optimization. Average both at the end

• But can implement the algorithm with just one vector per word ... and it helps

Two model variants:

1. Skip-grams (SG)

Predict context ("outside") words (position independent) given center word

2. Continuous Bag of Words (CBOW)

Predict center word from (bag of) context words

We presented: Skip-gram model

Additional efficiency in training:

1. Negative sampling

So far: Focus on naive softmax (simpler, but expensive, training method)

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.

The skip-gram model with negative sampling (HW2)

• The normalization term is computationally expensive

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

• Hence, in standard word2vec and HW2 you implement the skip-gram model with negative sampling

• Main idea: train binary logistic regressions for a true pair (center word and a word in its context window) versus several noise pairs (the center word paired with a random word)

- 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)

The skip-gram model with negative sampling (HW2)

• From paper: "Distributed Representations of Words and Phrases and their Compositionality" (Mikolov et al. 2013)

• Overall objective function (they maximize):
$$J(\theta) = \frac{1}{T} \sum_{t=1}^T J_t(\theta)$$
$$J_t(\theta) = \log \sigma(u_o^T v_c) + \sum_{i=1}^k \mathbb{E}_{j \sim P(w)} [\log \sigma(-u_j^T v_c)]$$

• The logistic/sigmoid function: $\sigma(x) = \frac{1}{1+e^{-x}}$ (we'll become good friends soon)

• We maximize the probability of two words co-occurring in first log and minimize probability of noise words

$v_c \rightarrow$ center word
 $u_o \rightarrow$ window (outside word) } True pair
 $u_j \rightarrow$ Negative sampled word from Corpus

\therefore 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.

prob. $\frac{V(w)^{3/4}}{V(w)}$ $V(w) \rightarrow$ unigram prob.

$$f(z) = \frac{1}{z} \quad z \rightarrow \text{total corpus.}$$

why $3/4 \rightarrow$ helps to sample commonly occurring & 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 options to build co-occurrence matrix:
 - Window-based or full document based

Example: Window based co-occurrence matrix

- Window length 1 (more common: 5–10)
- Symmetric (irrelevant whether left or right context)
- Example corpus:
 - I like deep learning
 - I like NLP
 - I enjoy flying

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 \rightarrow models are less robust.

- How to make them more useful? \rightarrow dimensionality reduction.
 - Using SVD:

Classic Method: Dimensionality Reduction on X (HW1)

Singular Value Decomposition of co-occurrence matrix X

Factorizes X into $U\Sigma V^T$, where U and V are orthonormal

$$\begin{bmatrix} * & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \end{bmatrix} = \begin{bmatrix} * & * & * \\ * & * & * \\ * & * & * \end{bmatrix} \begin{bmatrix} \bullet & & \\ & \bullet & \\ & & \bullet \end{bmatrix} \begin{bmatrix} * & * & * & * & * \\ * & * & * & * & * \\ * & * & * & * & * \end{bmatrix}$$

Retain only k singular values, in order to generalize.
 \hat{X} is the best rank k approximation to X , in terms of least squares.
 Classic linear algebra result. Expensive to compute for large matrices.

- 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)

5. Towards GloVe: Count based vs. direct prediction

Count based	Direct prediction
<ul style="list-style-type: none"> LSA, HAL (Lund & Burgess), COALS, Hellinger-PCA (Rohde et al, Lebel & Collobert) 	<ul style="list-style-type: none"> Skip-gram/CBoW (Mikolov et al), NNLM, HLBL, RNN (Bengio et al, Collobert & Weston; Huang et al; Mnih & Hinton)
<ul style="list-style-type: none"> Fast training Efficient usage of statistics Primarily used to capture word similarity Disproportionate importance given to large counts 	<ul style="list-style-type: none"> Scales with corpus size Inefficient usage of statistics Generate improved performance on other tasks Can capture complex patterns beyond word similarity

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

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

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

	$x = \text{solid}$	$x = \text{gas}$	$x = \text{water}$	$x = \text{random}$
$P(x \text{ice})$	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

\rightarrow To do this, hypothesis was to make dot products of two vectors similar to its conditional log prob.

Encoding meaning in vector differences

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

$w_i \cdot w_j \approx \log P(i|j) \rightarrow$ it dot product of

w_i & $w_j \approx \log (\text{prob. } (i \text{ given } j))$

& if $w_j = w_a - w_b$ then. diff bet'n two vectors \rightarrow similarity or dissimilarity

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

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

Combining the best of both worlds

GloVe [Pennington, Socher, and Manning, EMNLP 2014]

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

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

- Fast training
- Scalable to huge corpora
- Good performance even with small corpus and small vectors

Stanford

$w_i \cdot w_j = \log P(i|j) \rightarrow$
 GloVe: vector similarity (i.e) dot product \simeq prob. of co-occurrence.
 $f(x_{i,j}) \rightarrow f(\text{small_val}) \simeq \text{large num.}$
 $\therefore f(x_{i,j}) \rightarrow$ helps to balance the rarely & commonly occurring words

Evaluation of Word Vectors

6. How to evaluate word vectors?

- Related to general evaluation in NLP: Intrinsic vs. extrinsic
- Intrinsic:
 - Evaluation on a specific/intermediate subtask
 - Fast to compute
 - Helps to understand that system
 - Not clear if really helpful unless correlation to real task is established
- Extrinsic:
 - Evaluation on a real task
 - Can take a long time to compute accuracy
 - Unclear if the subsystem is the problem or its interaction or other subsystems
 - If replacing exactly one subsystem with another improves accuracy \rightarrow Winning!

Stanford

Intrinsic Evaluation:

Intrinsic word vector evaluation

- Word Vector Analogies

$a:b :: c:?$
 man:woman :: king:?

$$d = \arg \max_i \frac{(x_b - x_a + x_c)^T x_i}{\|x_b - x_a + x_c\|}$$
- Evaluate word vectors by how well their cosine distance after addition captures intuitive semantic and syntactic analogy questions
- Discarding the input words from the search!
- Problem: What if the information is there but not linear?

Stanford

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