Natural Language Processing
Word Vectors

Felipe Bravo-Marquez

October 7, 2019

Word Vectors

- A major component in neural networks for language is the use of an embedding layer.
- A mapping of discrete symbols to continuous vectors.
- When embedding words, they transform from being isolated distinct symbols into mathematical objects that can be operated on.
- Distance between vectors can be equated to distance between words.
- This makes easier to generalize the behavior from one word to another.

Distributional Vectors

- **Distributional Hypothesis** [Harris, 1954]: words occurring in the same **contexts** tend to have similar meanings.
- Or equivalently: “a word is characterized by the **company** it keeps”.
- **Distributional representations**: words are represented by **high-dimensional vectors** based on the context’s where they occur.

Word-context Matrices

- Distributional vectors are built from word-context matrices M.
- Each cell (i, j) is a co-occurrence based association value between a **target word** w_i and a **context** c_j calculated from a corpus of documents.
- Contexts are commonly defined as windows of words surrounding w_i.
- The window length k is a parameter (between 1 and 8 words on both the left and the right sides of w_i).
- If the Vocabulary of the target words and context words is the same, M has dimensionality $|\mathcal{V}| \times |\mathcal{V}|$.
- Whereas shorter windows are likely to capture **syntactic information** (e.g, POS), longer windows are more likely to capture topical similarity [Goldberg, 2016, Jurafsky and Martin, 2008].

Distributional Vectors with context windows of size 1

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

[0] Example taken from:
`http://cs224d.stanford.edu/lectures/CS224d-Lecture2.pdf`

Word-context Matrices

The associations between words and contexts can be calculated using different approaches:

1. Co-occurrence counts.
2. Positive point-wise mutual information (PPMI).
3. The significance values of a paired t-test.

The most common of those according to [Jurafsky and Martin, 2008] is PPMI. Distributional methods are also referred to as count-based methods.

PPMI

- PMI calculates the log of the probability of word-context pairs occurring together over the probability of them being independent.

$$\text{PMI}(w, c) = \log_2 \left(\frac{P(w, c)}{P(w)P(c)} \right) = \log_2 \left(\frac{\text{count}(w, c) \times |D|}{\text{count}(w) \times \text{count}(c)} \right) \tag{1}$$

- Negative PMI values suggest that the pair co-occurs less often than chance.
- These estimates are unreliable unless the counts are calculated from very large corpora [Jurafsky and Martin, 2008].
- PPMI corrects this problem by replacing negative values by zero:

$$\text{PPMI}(w, c) = \max(0, \text{PMI}(w, c)) \tag{2}$$

Distributed Vectors or Word embeddings

- Count-based distributional vectors increase in size with vocabulary i.e., can have a very high dimensionality.
- Explicitly storing the co-occurrence matrix can be memory-intensive.
- Some classification models don't scale well to high-dimensional data.
- The neural network community prefers using **distributed representations**[1] or **word embeddings**.
- Word **embeddings** are low-dimensional continuous dense word vectors trained from document corpora using **neural networks**.
- The dimensions are not directly interpretable i.e., represent latent features of the word, "hopefully capturing useful syntactic and semantic properties" [Turian et al., 2010].
- They have become a crucial component of neural network architectures for NLP.

[1] Idea: The meaning of the word is "distributed" over a combination of dimensions.

Distributed Vectors or Word embeddings (2)

- There are two main approaches for obtaining word embeddings:
  1. Embedding layers: using an embedding layer in a task-specific neural network architecture trained from labeled examples (e.g., sentiment analysis).
  2. Pre-trained word embeddings: creating an auxiliary predictive task from unlabeled copora (e.g., predict the following word) in which word embeddings will naturally arise from the neural-network architecture.
- These approaches can also be combined: one can initialize an embedding layer of a task-specific neural network with pre-trained word embeddings obtained with the second approach.

Distributed Vectors or Word embeddings (2)

- Most popular models based on the second approach are skip-gram [Mikolov et al., 2013], continuos bag-of-words [Mikolov et al., 2013], and Glove [Pennington et al., 2014].
- Word embeddings have shown to be more powerful than distributional approaches in many NLP tasks [Baroni et al., 2014].
- In [Amir et al., 2015], they were used as **features** in a regression model for determining the association between Twitter words and **positive sentiment**.

Word2Vec

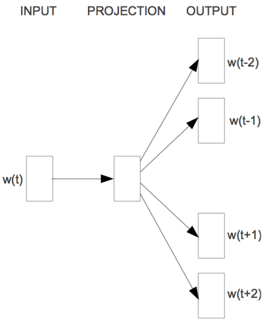
- Word2Vec is a software package that implements two neural network architectures for training word embeddings: Continuos Bag of Words (CBOW) and Skip-gram.
- It implements two optimization models: Negative Sampling and Hierarchical Softmax.
- These models are shallow neural networks that are trained to predict the contexts of words.

Skip-gram Model

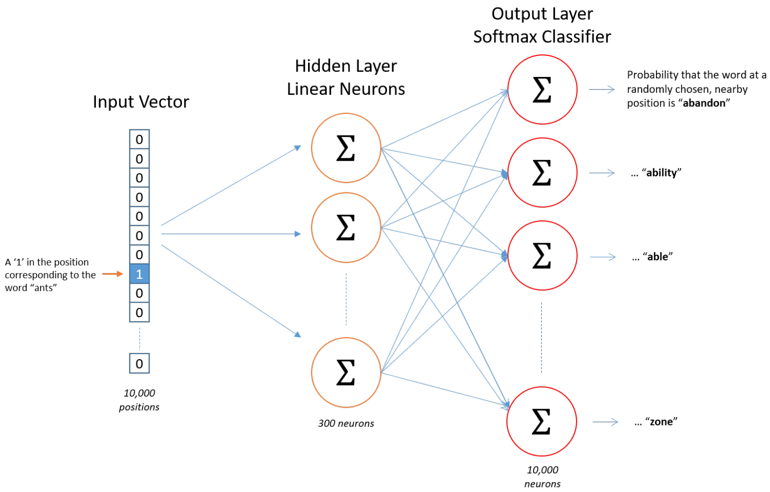
- A neural network with one “projection” or “hidden” layer is trained for predicting the words surrounding a center word, within a window of size k that is shifted along the input corpus.
- The center and surrounding k words correspond to the input and output layers of the network.
- Words are initially represented by 1-hot vectors: vectors of the size of the vocabulary ($|V|$) with zero values in all entries except for the corresponding word index that receives a value of 1.

Skip-gram Model

- The output layer combines the k 1-hot vectors of the surrounding words.
- The hidden layer has a dimensionality d, which determines the size of the embeddings (normally $d \ll |V|$).



Skip-gram Model



[1] Picture taken from: `http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/`

Parametrization of the Skip-gram model

- We are given an input corpus formed by a sequence of words $w_1, w_2, w_3, ..., w_T$ and a window size k.
- We denote target or (center) words by letter w and surrounding context words by letter c.
- The objective of the Skip-gram model is to maximize the average log probability of the context words given the target words:

$$\frac{1}{T}\sum_{t=1}^{T}\sum_{c \in c_{1:k}} \log p(c|w_t)$$

- The conditional probability of a context word c given a center word w is modeled with a softmax (C is the set of all context words, which is usually the same as the vocabulary):

$$p(c|w) = \frac{e^{\vec{c}\cdot\vec{w}}}{\sum_{c' \in C} e^{\vec{c'}\cdot\vec{w}}}$$

- Model's parameters θ: $\vec{c}$ and $\vec{w}$ (vector representations of contexts and target words).

Parametrization of the Skip-gram model

- Let D be the set of correct word-context pairs (i.e., word pairs that are observed in the Corpus).
- The optimization goal is to maximize the conditional likelihood of the contexts c:

$$\arg\max_{\vec{c},\vec{w}} \sum_{(w,c)\in D} \log p(c|w) = \sum_{(w,c)\in D} (\log e^{\vec{c}\cdot\vec{w}} - \log \sum_{c'\in C} e^{\vec{c}'\cdot\vec{w}}) \quad (3)$$

- Assumption: maximizing this function will result in good embeddings $\vec{w}$ i.e., similar words will have similar vectors.
- The term $p(c|w)$ is computationally expensive because of the summation $\sum_{c'\in C} e^{\vec{c}'\cdot\vec{w}}$ over all the contexts c'.
- Fix: replace the softmax with a hierarchical softmax (the vocabulary is represented with a Huffman binary tree).
- Huffman trees assign short binary codes to frequent words, reducing the number of output units to be evaluated.

Skip-gram with Negative Sampling

- Negative-sampling (NS) is presented as a more efficient model for calculating skip-gram embeddings.
- However, it optimizes a different objective function [Goldberg and Levy, 2014].
- NS maximizes the probability that a word-context pair (w, c) came from the set of correct word-context pairs D using a sigmoid function:

$$P(D = 1|w, c_i) = \frac{1}{1 + e^{-\vec{w} \cdot \vec{c_i}}}$$

- Assumption: the contexts words c_i are independent from each other:

$$P(D = 1|w, c_{1:k}) = \prod_{i=1}^{k} P(D = 1|w, c_i) = \prod_{i=1}^{k} \frac{1}{1 + e^{-\vec{w} \cdot \vec{c_i}}}$$

- This leads to the following target function (log-likelihood):

$$\underset{\vec{c}, \vec{w}}{\arg\max} \quad \log P(D = 1|w, c_{1:k}) = \sum_{i=1}^{k} \log \frac{1}{1 + e^{-\vec{w} \cdot \vec{c_i}}} \tag{4}$$

Skip-gram with Negative Sampling (2)

- This objective has a trivial solution if we set $\vec{w}$,$\vec{c}$ such that $p(D = 1|w, c) = 1$ for every pair (w, c) from D.
- This is achieved by setting $\vec{w} = \vec{c}$ and $\vec{w} \cdot \vec{c} = K$ for all $\vec{w}, \vec{c}$, where K is a large number.
- We need a mechanism that prevents all the vectors from having the same value, by disallowing some (w, c) combinations.
- One way to do so, is to present the model with some (w, c) pairs for which $p(D = 1|w, c)$ must be low, i.e. pairs which are not in the data.
- This is achieved sampling negative samples from $\tilde{D}$.

Skip-gram with Negative Sampling (3)

- Sample m words for each word-context pair $(w, c) \in D$.
- Add each sampled word w_i together with the original context c as a negative example to $\tilde{D}$.
- Final objective function:

$$\arg\max_{\vec{c},\vec{w}} \sum_{(w,c)\in D} \log P(D = 1|w, c_{1:k}) + \sum_{(w,c)\in \tilde{D}} \log P(D = 0|w, c_{1:k}) \tag{5}$$

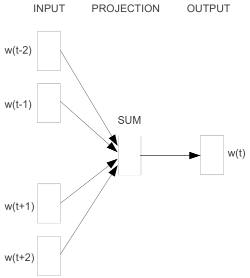
- The negative words are sampled from smoothed version of the corpus frequencies:

$$\frac{\#(w)^{0.75}}{\sum_{w'} \#(w')^{0.75}}$$

- This gives more relative weight to less frequent words.

Continuos Bag of Words: CBOW

- Similar to the skip-gram model but now the center word is predicted from the surrounding context.



CBOW

GloVe

- GloVe (from global vectors) is another popular method for training word embeddings [Pennington et al., 2014].
- It constructs an explicit word-context matrix, and trains the word and context vectors $\vec{w}$ and $\vec{c}$ attempting to satisfy:

$$\vec{w} \cdot \vec{c} + b_{[w]} + b_{[c]} = \log \#(w, c) \quad \forall (w, c) \in D \tag{6}$$

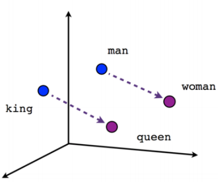
- where $b_{[w]}$ and $b_{[c]}$ are word-specific and context-specific trained biases.

GloVe (2)

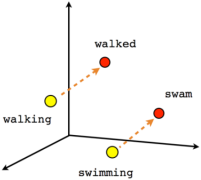
- In terms of matrix factorization, if we fix $b_{[w]} = \log \#(w)$ and $b_{[c]} = \log \#(c)$ we'll get an objective that is very similar to factorizing the word-context PMI matrix, shifted by $\log(|D|)$.
- In GloVe the bias parameters are learned and not fixed, giving it another degree of freedom.
- The optimization objective is weighted least-squares loss, assigning more weight to the correct reconstruction of frequent items.
- When using the same word and context vocabularies, the model suggests representing each word as the sum of its corresponding word and context embedding vectors.

Word Analogies

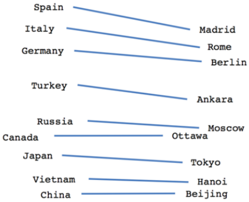
- Word embeddings can capture certain semantic relationships, e.g. male-female, verb tense and country-capital relationships between words.
- For example, the following relationship is found for word embeddings trained using Word2Vec: $\vec{w}_{king} - \vec{w}_{man} + \vec{w}_{woman} \approx \vec{w}_{queen}$.



Male-Female



Verb tense



Country-Capital

[2]Source: `https://www.tensorflow.org/tutorials/word2vec`

Correspondence between Distributed and Distributional Models

- Both the distributional “count-based” methods and the distributed “neural” ones are based on the distributional hypothesis.
- The both attempt to capture the similarity between words based on the similarity between the contexts in which they occur.
- Levy and Goldebrg showed in [Levy and Goldberg, 2014] that Skip-gram negative sampling (SGNS) is implicitly factorizing a word-context matrix, whose cells are the pointwise mutual information (PMI) of the respective word and context pairs, shifted by a global constant.
- This ties the neural methods and the traditional “count-based” suggesting that in a deep sense the two algorithmic families are equivalent.

FastText

- FastText embeedings extend the skipgram model to take into account the internal structure of words while learning word representations [Bojanowski et al., 2016].
- A vector representation is associated with each character *n*-gram.
- Words are represented as the sum of these representations.
- Taking the word *where* and $n = 3$, it will be represented by the character *n*-grams: $<$wh, whe, her, ere, re$>$, and the special sequence $<$where$>$.
- Note that the sequence $<her>$, corresponding to the word “her” is different from the tri-gram “her” form the word “here”.
- FastText is useful for morphologically rich languages. For example, the words “amazing” and “amazingly” share information in FastText through their shared *n*-grams, whereas in Word2Vec these two words are completely unrelated.

FastText (2)

- Let $\mathcal{G}_w$ be the set of n-grams appearing in w.
- FastText associates a vector $\vec{g}$ with each n-gram in $\mathcal{G}_w$.
- In FastText the probability that a word-context pair (w, c) came from the input corpus D is calculated as follows:

$$P(D|w,c) = \frac{1}{1 + e^{-s(w,c)}}$$

where,

$$s(w,c) = \sum_{g \in G_w} \vec{g} \cdot \vec{c}.$$

- The negative sampling algorithm can be calculated in the same form as in the skip-gram model with this formulation.

Sentiment-Specific Phrase Embeddings

- Problem of word embeddings: antonyms can be used in similar contexts e.g., my car is nice vs my car is ugly.
- In [Tang et al., 2014] **sentiment-specific** word embeddings are proposed by combining the skip-gram model with emoticon-annotated tweets :) :(.
- These embeddings are used for **training** a word-level polarity classifier.
- The model integrates sentiment information into the continuous representation of phrases by developing a tailored neural architecture.
- Input: $\{w_i, s_j, pol_j\}$, where w_i is a phrase (or word), s_j the sentence, and pol_j the sentence's polarity.

Sentiment-Specific Phrase Embeddings (2)

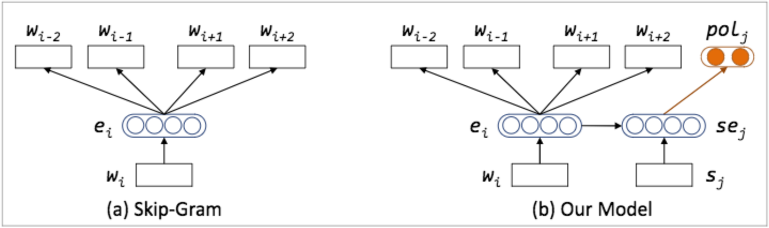
- The training objective uses the embedding of w_i to predict its context words (in the same way as the skip-gram model), and uses the sentence representation se_j to predict pol_j.
- Sentences (se_j) are represented by averaging the word vectors of their words.
- The objective of the sentiment part is to maximize the average of log sentiment probability:

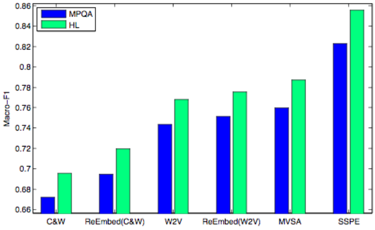
$$f_{sentiment} = \frac{1}{S} \sum_{j=1}^{S} \log p(pol_j | se_j)$$

- The final training objective is to maximize the linear combination of the skip-gram and sentiment objectives:

$$f = \alpha f_{skipgram} + (1 - \alpha) f_{sentiment}$$

Sentiment-Specific Phrase Embeddings





(b) Sentiment classification of lexicons with different embedding learning algrithms.

Gensim

Gensim is an open source Python library for natural language processing that implements many algorithms for training word embeddings.

- https://radimrehurek.com/gensim/
- https://machinelearningmastery.com/develop-word-embeddings-python-gensim/

Questions?

Thanks for your Attention!

References I

Amir, S., Ling, W., Astudillo, R., Martins, B., Silva, M. J., and Trancoso, I. (2015).
Inesc-id: A regression model for large scale twitter sentiment lexicon induction.
In *Proceedings of the 9th International Workshop on Semantic Evaluation (SemEval 2015)*, pages 613–618, Denver, Colorado. Association for Computational Linguistics.

Baroni, M., Dinu, G., and Kruszewski, G. (2014).
Don't count, predict! a systematic comparison of context-counting vs. context-predicting semantic vectors.
In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics*, pages 238–247. Association for Computational Linguistics.

Bojanowski, P., Grave, E., Joulin, A., and Mikolov, T. (2016).
Enriching word vectors with subword information.
arXiv preprint arXiv:1607.04606.

Goldberg, Y. (2016).
A primer on neural network models for natural language processing.
J. Artif. Intell. Res.(JAIR), 57:345–420.

Goldberg, Y. and Levy, O. (2014).
word2vec explained: deriving mikolov et al.'s negative-sampling word-embedding method.
arXiv preprint arXiv:1402.3722.

References II

Harris, Z. (1954).
Distributional structure.
Word, 10(23):146–162.

Jurafsky, D. and Martin, J. H. (2008).
Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition.
Prentice Hall, Upper Saddle River, NJ, USA, 2nd edition.

Levy, O. and Goldberg, Y. (2014).
Neural word embedding as implicit matrix factorization.
In *Advances in neural information processing systems*, pages 2177–2185.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., and Dean, J. (2013).
Distributed representations of words and phrases and their compositionality.
In Burges, C., Bottou, L., Welling, M., Ghahramani, Z., and Weinberger, K., editors, *Advances in Neural Information Processing Systems 26*, pages 3111–3119. Curran Associates, Inc.

Pennington, J., Socher, R., and Manning, C. D. (2014).
Glove: Global vectors for word representation.
In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL*, pages 1532–1543.

References III

Tang, D., Wei, F., Qin, B., Zhou, M., and Liu, T. (2014).
Building large-scale twitter-specific sentiment lexicon : A representation learning approach.
In *COLING 2014, 25th International Conference on Computational Linguistics, Proceedings of the Conference: Technical Papers, August 23-29, 2014, Dublin, Ireland*, pages 172–182.

Turian, J., Ratinov, L., and Bengio, Y. (2010).
Word representations: a simple and general method for semi-supervised learning.
In *Proceedings of the 48th annual meeting of the association for computational linguistics*, pages 384–394. Association for Computational Linguistics.