Introduction to Word Vectors

Learning Word Vectors Using `word2vec` and Reasoning With Them

Dario Gjorgjevski[1]
gjorgjevski.dario@students.finki.ukim.mk

[1] Faculty of Computer Science and Engineering
Ss. Cyril and Methodius University in Skopje

January 29, 2017

Contents

1. Motivation
   - Language Models
   - Word Vectors
2. Learning Word Vectors
3. Reasoning With Word Vectors

What is a language model?

Definition (Language model)

A *language model* is a probability distribution over word sequences. More formally, given a word sequence $w_1, \dots, w_m$, the language model assigns a probability

$$p(w_1, \dots, w_m)$$

to the whole sequence.

Until recently, the most widely used language model was the *n-gram model*, which assumes an nth order Markov property.

n-gram model

Concept

In an n-gram model, the probability $p(w_1, \ldots, w_m)$ is approximated as

$$\begin{aligned} p(w_1, \ldots, w_m) &= \prod_{i=1}^{m} p(w_i \mid w_1, \ldots, w_{i-1}) \\ &\approx \prod_{i=1}^{m} p(w_i \mid w_{i-(n-1)}, \ldots, w_{i-1}). \end{aligned}$$

- The conditional probabilities can be calculated by counting; however,
- *Smoothing* is required for unseen n-grams.

n-gram model

Issues

- As the n-gram model is trained on larger and larger texts, the vocabulary size increases as Kn^{β} with $10 \leq K \leq 100$ and $0.4 \leq \beta \leq 0.6$ (*Heaps' law*).
- As the vocabulary size increases, the number of possible sequences of words increases exponentially $\implies$ data sparsity problems.
- Ultimately, we are not sure how a word is to be even represented, and the n-gram model does not provide us with any clues toward that direction.

Localist representations

These representations are also called *one-hot*. They dedicate one unit to each word.

- Easy to understand.
- Easy to code by hand.
  - Have been used as inputs to machine learning algorithms, especially neural networks.
- Easy to learn.
  - It is what mixture models (e.g. hidden Markov models) do.
- Easy to associate with other representations or responses.

However, they are terribly inefficient when the data has componential structure—and language does.

Localist representations

Issues

In vector space terms, these are vectors with one 1 and lots of 0s:

$$\begin{bmatrix} \ldots & 0 & 0 & 1 & 0 & 0 & \ldots \end{bmatrix}.$$

- These vectors are *extremely* sparse.
- They do not encode encode similarities whatsoever, i.e.

  $$\forall i \neq j.\, \mathbf{w}_i \wedge \mathbf{w}_j = \mathbf{0}$$

  even if $\mathbf{w}_i$ and $\mathbf{w}_j$ are inherently similar.

Distributional representations

A lot of information can be obtained by representing a word by means of its neighbors.

> *You shall know a word by the company it keeps.*
>
> — *J. R. Firth*

How do we learn distributional representations?

- Directly, using co-occurrence matrices; or
- As a side-effect by training neural networks to predict either the word given its context, or the context given a word (*continuous space language models*).

Contents

1. Motivation
2. Learning Word Vectors
   - Learning Context by Counting
   - Learning Context by Prediction
   - Making Learning by Prediction Practical
3. Reasoning With Word Vectors

Co-occurrence matrices

Definition (Co-occurrence matrix)

Let $\{w_1, w_2, \dots, w_V\}$ be the vocabulary. A *co-occurrence matrix* is a $V \times V$ matrix which counts how many times a pair of words (w_i, w_j) appear together in some context.

- Very straightforward solution.
- Can learn either:
    - General topics by considering full documents as contexts; or
    - Syntactic and semantic information by using a window context.

The context is usually taken to be a window of size k.

Co-occurrence matrices

Issues

- Dimension increases quadratically with $V \implies$ huge storage requirements, sparsity issues in classifiers.
- Models are less robust because of sparsity issues.
- Low-dimensional vectors can be obtained by SVD. Unfortunately, computing the SVD of an $m \times n$ matrix requires $\mathcal{O}(m^2 n)$ operations:
  - Computationally infeasible for millions of words.
  - Hard to incorporate new data.

Where do we go from here? We do not count anymore; instead, we directly learn low-dimensional dense vectors by neural network prediction.

What to predict?

Two models have received most attention in literature:

- *Continuous bag-of-words* (CBOW) model, which predicts target words (e.g. “mat”) from source context words (e.g. “the cat sits on the”); and
- *Skip-gram* model, which predicts source context words from the target words.

CBOW smoothes over a lot of the distributional information by treating an entire context as a single observation, while skip-gram treats every context–target pair as a new observation.

Continuous bag-of-words model

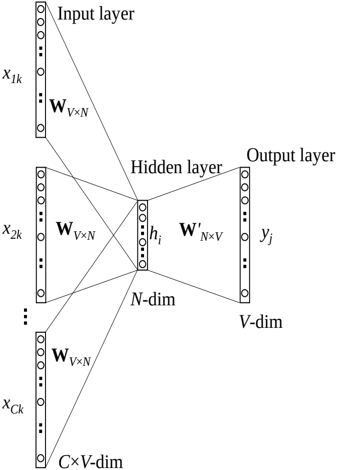

Figure: CBOW model with C words in the context.

Continuous bag-of-words model

Mathematical definition

- The input consists of one-hot encoded words, $\mathbf{x}_1, \dots, \mathbf{x}_C$.
- The input → hidden and hidden → output weights are represented by $\mathbf{W}$ and $\mathbf{W}'$ respectively. The rows of $\mathbf{W}$ (the $\mathbf{v}_w$'s) and the columns of $\mathbf{W}'$ (the $\mathbf{v}'_w$'s), give the N-dimensional input and output representations of a word w.
- The hidden layer is *linear* and computes an average

$$\mathbf{h} := \frac{\left(\sum_{c=1}^{C} \mathbf{x}_c\right) \mathbf{W}}{C} = \frac{\sum_{c=1}^{C} \mathbf{v}_{w_c^{(\mathrm{in})}}}{C}.$$

- The output layer is a softmax panel outputting a multinomial distribution over the vocabulary.

Continuous bag-of-words model

Update equations

As usual with softmax, the cross-entropy loss function is used,

$$\begin{aligned} E &:= -\log p(w^{(\text{out})} \mid w_1^{(\text{in})}, \ldots, w_C^{(\text{in})}) \\ &= -\mathbf{v}'^{\top}_{w^{(\text{out})}}\mathbf{h} + \log \sum_{v=1}^{V} \exp(\mathbf{v}'^{\top}_{w_v}\mathbf{h}). \end{aligned} \tag{1}$$

By taking the corresponding derivatives of (1), we arrive at the update equations:

$$\begin{aligned} \mathbf{v}'_{w_v} &:= \mathbf{v}'_{w_v} - \eta e_v \mathbf{h} \qquad (\forall 1 \leq v \leq V) \\ \mathbf{v}_{w_c^{(\text{in})}} &:= \mathbf{v}_{w_c^{(\text{in})}} - \frac{\eta}{C}\frac{\partial E}{\partial \mathbf{h}} \quad (\forall 1 \leq c \leq C). \end{aligned} \tag{2}$$

Continuous bag-of-words model

Understanding the update equations

The derivatives in (2) can be computed using backpropagation:

$$e_v = \text{prediction error for word } v,$$
$$\frac{\partial E}{\partial h_i} = \sum_{v=1}^{V} e_v w'_{iv}.$$

Intuitively,

- If we overestimate (resp. underestimate) the probability for word w_v, we subtract from (resp. add to) $\mathbf{v}'_{w^{(\text{out})}}$ a portion of the input $\implies$ $\mathbf{v}'_{w^{(\text{out})}}$ becomes closer to the averaged input vectors.
- Since $\partial E / \partial \mathbf{h}$ is the sum of all output vectors weighted by their prediction errors, (2) can be seen as distributing a portion of every output vector to each of the C input vectors.

Skip-gram model

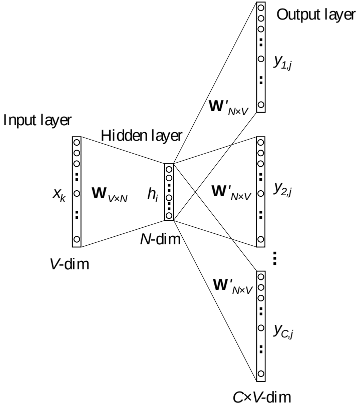

Figure: Skip-gram model for predicting C context words.

Skip-gram model

Mathematical definition

- The input consists of a one-hot encoded word, $\mathbf{x}$, with $x_k = 1$ and $x_i = 0$ for all $i \neq k$.
- The weights encode the same information as in the CBOW model.
- The hidden layer simply copies the row of the matrix $\mathbf{W}$ associated with the input word $w^{(\text{in})}$:

$$\mathbf{h} = \mathbf{W}_{(k,\cdot)} =: \mathbf{v}_{w^{(\text{in})}}.$$

- The output layer consists of C softmax panels, and outputs C multinomial distributions over the vocabulary.

Skip-gram model

Update equations

Again, we use the cross-entropy loss function. Note that care must be taken as we are now dealing with C multinomial distributions rather than just one.

$$\begin{aligned} E &:= -\log p(w_1^{(\mathrm{out})}, \dots, w_C^{(\mathrm{out})} \mid w^{(\mathrm{in})}) \\ &= -\sum_{c=1}^{C} \mathbf{v'}^{\top}_{w_c^{(\mathrm{out})}} \mathbf{h} + C \log \sum_{v=1}^{V} \exp(\mathbf{v'}^{\top}_{w_v} \mathbf{h}). \end{aligned}$$

The equations are identical as in the CBOW model, except the prediction errors must be summed across all C softmax panels.

$$e_v = \sum_{c=1}^{C} (\text{prediction error of the } c\text{th softmax panel for word } v).$$

Issues with the CBOW and skip-gram models

- Both models are very similar in their nature and have a same intuitive interpretation.
- Due to the smoothing, CBOW is recommended for small datasets; while skip-gram is able to learn better representations give sufficient data (“there is no better regulizer than a large dataset”).
- Unfortunately, both are very hard to train.

We tried to avoid the vast size of the vocabulary as hard as possible, however, (2) performs $\mathcal{O}(V)$ weight updates for both the CBOW and skip-gram models. To ameliorate this and make learning practical, we will look at

- hierarchical softmax; and
- negative sampling.

Hierarchical softmax

The hierarchical softmax uses a binary tree to represent the vocabulary. There are V leaves, one for each word; and $V-1$ inner units.

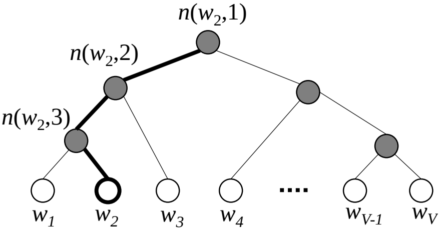

Figure: Example hierarchical softmax with path to w_2.

Hierarchical softmax

Details

- There is no output vector representation—each of the $V-1$ inner units has an output vector $\mathbf{v}_{n(w,j)}$.
- The probability of a word being the output word is

$$\Pr(w = w^{(\text{out})}) = \prod_{j=1}^{L(w)-1} \sigma(\llbracket n(w,j+1) = \text{left}(n(w,j)) \rrbracket \mathbf{v'}_{n(w,j)}^{\top} \mathbf{h}),$$

where $L(w)$ is the length of the path to word w, $\text{left}(n(w,j))$ is the left child of the inner unit $n(w,j)$, σ is the sigmoid function, and

$$\llbracket p \rrbracket = \begin{cases} 1 & \text{if } p \text{ is true} \\ -1 & \text{otherwise.} \end{cases}$$

Hierarchical softmax

Update equations

By backtracing the path from the leaves to the root, one obtains the update equations for the inner nodes

$$\mathbf{v}'_{n(w,j)} := \mathbf{v}'_{n(w,j)} - \eta \left(\sigma(\mathbf{v}'^{\top}_{n(w,j)}\mathbf{h}) - t_{n(w,j)} \right) \mathbf{h} \quad (\forall 1 \leq j \leq L(w) - 1),$$

where $t_{n(w,j)}$ denotes the "ground truth" at level j ("left" or "right"). The derivative of the error with respect to the hidden layer can be found as

$$\frac{\partial E}{\partial \mathbf{h}} = \sum_{j=1}^{L(w)-1} \left(\sigma(\mathbf{v}'^{\top}_{n(w,j)}\mathbf{h}) - t_j \right) \mathbf{v}'_{n(w,j)}.$$

Hierarchical softmax

Bottom line

- Hierarchical softmax can be applied to both the CBOW and skip-gram models.
- For CBOW, one can plug in the equations directly. For skip-gram, we need to iterate over the entire context we're predicting.
- Computational complexity is reduced from $\mathcal{O}(V)$ to $\mathcal{O}(\log V)$.
- Number of parameters remains roughly the same.

Can we do even better? Turns out, if some heuristic assumptions are satisfied, we can.

Negative sampling

The idea is very straightforward: instead of updating all output vectors, update only a sample of them.

Assumption (Negative sampling)

In order to learn the context, it is enough to update only the output word ("ground truth") along with a few words as negative samples.

The negative samples are drawn from a noise distribution $p^{(\text{neg})}(w)$ which is determined empirically. `word2vec` uses the unigram distribution raised to the power of 3/4.

Negative sampling

Loss function and error derivatives

For further heuristics, negative sampling does not use a loss function that produces a well-defined posterior multinomial distribution.

$$E = -\log\sigma(\mathbf{v'}^{\top}_{w^{(\text{out})}}\mathbf{h}) - \sum_{w^{(\text{neg})}\in\mathcal{W}^{(\text{neg})}} \log\sigma(-\mathbf{v'}^{\top}_{w^{(\text{neg})}}\mathbf{h}),$$

where $\mathcal{W}^{(\text{neg})}$ is a set of negative samples. This results in updates very similar to the hierarchical softmax. Letting $\mathcal{W} := \{w^{(\text{out})}\} \cup \mathcal{W}^{(\text{neg})}$,

$$\mathbf{v}'_{w_j} := \mathbf{v}'_{w_j} - \eta\left(\sigma(\mathbf{v'}^{\top}_{w_j}\mathbf{h}) - t_j\right)\mathbf{h} \quad (\forall\, w_j \in \mathcal{W})$$

$$\frac{\partial E}{\partial \mathbf{h}} = \sum_{w_j\in\mathcal{W}} \frac{\partial E}{\partial \mathbf{v'}^{\top}_{w_j}\mathbf{h}} \frac{\partial \mathbf{v'}^{\top}_{w_j}\mathbf{h}}{\partial \mathbf{h}} = \sum_{w_j\in\mathcal{W}} \left(\sigma(\mathbf{v'}^{\top}_{w_j}\mathbf{h}) - t_j\right)\mathbf{v}'_{w_j},$$

where $t_j := \mathbb{1}\{w_j = w^{(\text{out})}\}$.

Further improvements

Subsampling

In order to counter the imbalance between rare and frequent words, `word2vec` uses subsampling. Namely, each word in the training set is discarded with probability

$$1 - \sqrt{\frac{d}{f(w_j)}},$$

where $f(w_j)$ is the frequency of the word w_j and d is the discard threshold, typically around 10^{-5}.

Contents

1 Motivation

2 Learning Word Vectors

3 Reasoning With Word Vectors

Country–capital city relationships

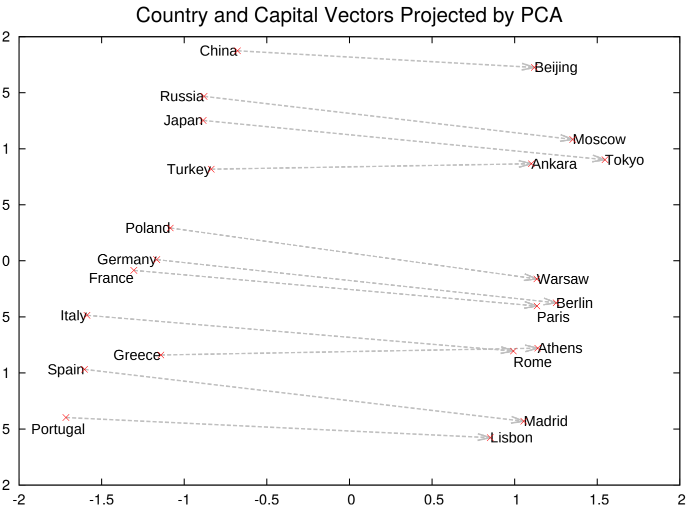

Figure: Country–capital city relationships projected with PCA.

Phrase representations

The aforementioned models are not restricted only to words—they can use phrases and other constructs just as well.

Table: Phrase analogy with 10^{-5} subsampling.

Phrase	NEG	HS
Vasco de Gamma	Lingsugur	Italian explorer
Lake Baikal	Great Rift Valley	Aral Sea
Alan Bean	Rebbeca Naomi	moonwalker
Ionian Sea	Ruegen	Ionian Islands
chess master	chess grandmaster	Garry Kasparov

Additive compositionality

Interestingly, element-wise addition yields meaningful combinations, too. This property can be explained by looking at the training objective: the vectors are related logarithmically to the probabilities computed by the output layer, so the sum of two word vectors is related to the product of the two context distributions.

Table: Additive compositionality showing the 4 closest vectors.

Czech + currency	Vietnam + capital	German + airlines	Russian + river	French + actress
Koruna	Hanoi	airline Lufthansa	Moscow	Juliette Binoche
Czech crown	Ho Chi Minh City	carrier Lufthansa	Volga River	Vanessa Paradis
Polish zolty	Viet Nam	flag carrier Lufthansa	upriver	Charlotte Gainsbourg
CTK	Vietnamese	Lufthansa	Russia	Cecile De