word2vec

Simão Gonçalves

Nova Search Reading Group

May 6th 2020

Content

Papers in this presentation

1. Efficient Estimation of Word Representations in Vector Space [Mikolov et al. 2013]
2. Distributed Representations of Words and Phrases and their Compositionality [Mikolov et al. 2013]

Content

- Symbolic representation of words

- Symbolic representation of words
- N-gram models: learn the joint probability of sequence of words.

Motivation

NLP state of the art at the time

- Symbolic representation of words
- N-gram models: learn the joint probability of sequence of words.
  - (+) Simple, well understood math;
  - (+) Scalable;
  - (+) Several NLP applications: spell-checking, information retrieval, machine translation, others;

Motivation

NLP state of the art at the time

- Symbolic representation of words
- N-gram models: learn the joint probability of sequence of words.
  - (+) Simple, well understood math;
  - (+) Scalable;
  - (+) Several NLP applications: spell-checking, information retrieval, machine translation, others;
  - (-) Sparcity: requires large data to capture longer dependencies ($n > 3$);
  - (-) Non local dependency (understanding pronouns and PoS);
  - (-) Markov Assumption too simplistic for more complex tasks;

source: http://ivan-titov.org/teaching/nlmi-15/lecture-2.pdf

- "You shall know a word by the company it keeps" - Firth 1957

Motivation
Distributional Semantics

- "You shall know a word by the company it keeps" - Firth 1957
- I.e. A word's meaning is given by the words that frequently appear close by.

Motivation

Distributional Semantics

- "You shall know a word by the company it keeps" - Firth 1957
- I.e. A word's meaning is given by the words that frequently appear close by.
- Example:

I owe this passion to my high **school** friend Jason.

After Alex goes to work and Jonathan goes to **school**, Destiny and I do the ch

At the Cambridge **school**, for the first time in my l

enjoyed the companionship of seeing and hearing people of my own age.

Motivation

Distributional Semantics

- "You shall know a word by the company it keeps" - Firth 1957
- I.e. A word's meaning is given by the words that frequently appear close by.
- Example:

I owe this passion to my high **school** friend Jason.

After Alex goes to work and Jonathan goes to **school**, Destiny and I do the ch

At the Cambridge **school**, for the first time in my l

enjoyed the companionship of seeing and hearing people of my own age.

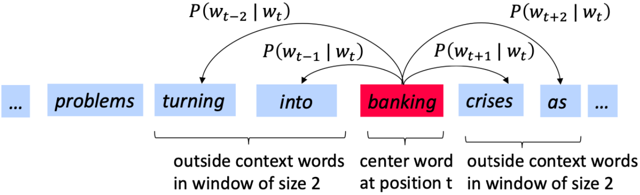
School's meaning is given by all of these contexts in which it appears.

word2vec model

How can we model distributional semantics?

1. Represent words as vectors (so that we can measure similarity!).

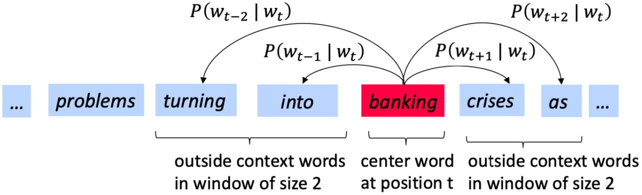
1. Represent words as vectors (so that we can measure similarity!).
2. For each center word, w_c we want to know the probability of the context ("outside") words, w_o.



word2vec model

How can we model distributional semantics?

1. Represent words as vectors (so that we can measure similarity!).
2. For each center word, w_c we want to know the probability of the context ("outside") words, w_o.



- We want our model to output a relatively high probability for these probabilities.

word2vec model

How can we model distributional semantics?

Setup

1. Start with a large corpus of text

word2vec model

How can we model distributional semantics?

Setup

1. Start with a large corpus of text
2. Every word in a fixed vocabulary is represented by a vector (randomly initialized) with a fixed size n.

Setup

1. Start with a large corpus of text
2. Every word in a fixed vocabulary is represented by a vector (randomly initialized) with a fixed size n.
3. Go through each position t in the text, which has a "center" word w_c and "context" ("outside") words, w_o.

Setup

1. Start with a large corpus of text
2. Every word in a fixed vocabulary is represented by a vector (randomly initialized) with a fixed size n.
3. Go through each position t in the text, which has a "center" word w_c and "context" ("outside") words, w_o.
4. Use the similarity of the word vectors w_c and w_o to calculate $p(w_o|w_c)$.

Setup

1. Start with a large corpus of text
2. Every word in a fixed vocabulary is represented by a vector (randomly initialized) with a fixed size n.
3. Go through each position t in the text, which has a "center" word w_c and "context" ("outside") words, w_o.
4. Use the similarity of the word vectors w_c and w_o to calculate $p(w_o|w_c)$.
5. Adjust the parameters of the word vectors to maximize this probability.

word2vec model

Objective function

For each position t in the text, predict the context words of t, given t.
Let's define the likelihood:

For each position t in the text, predict the context words of t, given t.
Let's define the likelihood:

$$L(\theta) = \prod_{t=1}^{T} \prod_{\substack{-m \leq j \leq m \\ j \neq 0}} P(w_{t+j} | w_t)$$

For each position t in the text, predict the context words of t, given t.
Let's define the likelihood:

$$L(\theta) = \prod_{t=1}^{T} \prod_{\substack{-m \leq j \leq m \\ j \neq 0}} P(w_{t+j} | w_t)$$

From here we can create the cost function:

For each position t in the text, predict the context words of t, given t.
Let's define the likelihood:

$$L(\theta) = \prod_{t=1}^{T} \prod_{\substack{-m \leq j \leq m \\ j \neq 0}} P(w_{t+j}|w_t)$$

From here we can create the cost function:

$$J(\theta) = -\frac{1}{T} \sum_{t=1}^{T} \sum_{\substack{-m \leq j \leq m \\ j \neq 0}} log(P(w_{t+j}|w_t))$$

word2vec model

How do we compute $P(w_o|w_c)$?

Use two vectors for each word:

Use two vectors for each word:

- v_w when w is a center word

Use two vectors for each word:

- v_w when w is a center word
- u_w when w is a context word

Use two vectors for each word:

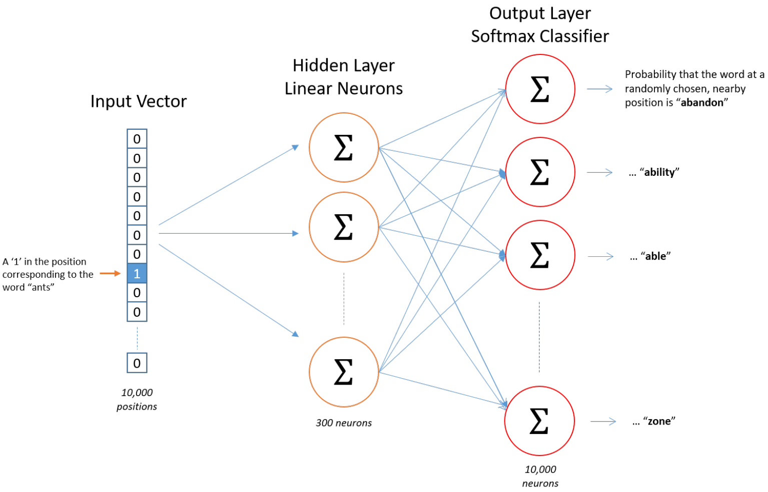
- v_w when w is a center word
- u_w when w is a context word

Then, apply softmax:

$$P(u_o|v_c) = \frac{exp(u_o \cdot v_c)}{\sum_{i=1}^{T} exp(u_i \cdot v_c)} \tag{1}$$

word2vec

How does this model look like



word2vec

How do we update the weights?

- Cost function from before:

$$J(\theta) = -\frac{1}{T}\sum_{t=1}^{T}\sum_{\substack{-m\leq j\leq m \\ j\neq 0}} log(P(w_{t+j}|w_t))$$

- Cost function from before:

$$J(\theta) = -\frac{1}{T}\sum_{t=1}^{T}\sum_{\substack{-m\leq j\leq m \\ j\neq 0}} log(P(w_{t+j}|w_t))$$

- minimize $J_{v_c} = -logP(u_{c-m}, ..., u_{c-1}, u_{c+1}, ..., u_{c+m}|v_c)$

- Cost function from before:

$$J(\theta) = -\frac{1}{T}\sum_{t=1}^{T}\sum_{\substack{-m\leq j\leq m\\ j\neq 0}} log(P(w_{t+j}|w_t))$$

- minimize $J_{v_c} = -logP(u_{c-m}, ..., u_{c-1}, u_{c+1}, ..., u_{c+m}|v_c)$
- Naive bayes assumption:

$$J = -log\prod_{j=-m,j\neq 0}^{2m} P(u_{c-m+j}|v_c)$$

- Cost function from before:

$$J(\theta) = -\frac{1}{T}\sum_{t=1}^{T}\sum_{\substack{-m\leq j\leq m\\ j\neq 0}} log(P(w_{t+j}|w_t))$$

- minimize $J_{v_c} = -logP(u_{c-m}, ..., u_{c-1}, u_{c+1}, ..., u_{c+m}|v_c)$
- Naive bayes assumption:

$$J = -log\prod_{j=-m, j\neq 0}^{2m} P(u_{c-m+j}|v_c)$$

- Objective function for a single input

$$J = -log \prod_{j=-m, j\neq 0}^{2m} P(u_{c-m+j}|v_c)$$

- Objective function for a single input

$$
\begin{aligned}
J &= -log \prod_{j=-m, j\neq 0}^{2m} P(u_{c-m+j}|v_c) \\
&= -\sum_{j=-m, j\neq 0}^{2m} log P(u_{c-m+j}|v_c)
\end{aligned}
$$

word2vec model

How do we update the weights?

- Objective function for a single input

$$
\begin{aligned}
J &= -log \prod_{j=-m, j\neq 0}^{2m} P(u_{c-m+j}|v_c) \\
&= -\sum_{j=-m, j\neq 0}^{2m} logP(u_{c-m+j}|v_c) \\
&= -\sum_{j=-m, j\neq 0}^{2m} log \frac{exp(u_o \cdot v_c)}{\sum_{i=1}^{T} exp(u_i \cdot v_c)}
\end{aligned} \tag{2}
$$

- Cross entropy:

$$J = -\sum_{j=-m, j\neq 0}^{2m} logP(u_{c-m+j} | v_c)$$

- Cross entropy:

$$J = -\sum_{j=-m,j\neq 0}^{2m} logP(u_{c-m+j}|v_c) \\ = \sum_{j=-m,j\neq 0}^{2m} H(\hat{y}, y_{c-m+j}) \quad (3)$$

word2vec

How do we update the weights?

$$\frac{\partial}{\partial v_c} logP(u_0 | v_c)$$

$$\frac{\partial}{\partial v_c} log P(u_0|v_c) = \frac{\partial}{\partial v_c} log \frac{exp(u_0 v_c)}{\sum_{i=1}^{T} exp(u_i v_c)}$$
$$= ...$$

$$
\begin{aligned}
\frac{\partial}{\partial v_c} logP(u_0|v_c) &= \frac{\partial}{\partial v_c} log \frac{exp(u_0 v_c)}{\sum_{i=1}^{T} exp(u_i v_c)} \\
&= ... \\
&= u_0 - \sum_{x=1}^{V} \frac{exp(u_x^T v_c)}{\sum_{w=1}^{V} exp(u_w^T v_c)} \cdot u_x
\end{aligned}
$$

$$
\begin{aligned}
\frac{\partial}{\partial v_c} logP(u_0|v_c) &= \frac{\partial}{\partial v_c} log \frac{exp(u_0 v_c)}{\sum_{i=1}^{T} exp(u_i v_c)} \\
&= ... \\
&= u_0 - \sum_{x=1}^{V} \frac{exp(u_x^T v_c)}{\sum_{w=1}^{V} exp(u_w^T v_c)} \cdot u_x \\
&= u_0 - \sum_{x=1}^{V} p(u_x|v_c).u_x
\end{aligned}
$$

$$\begin{aligned}
\frac{\partial}{\partial v_c} logP(u_0|v_c) &= \frac{\partial}{\partial v_c} log \frac{exp(u_0 v_c)}{\sum_{i=1}^{T} exp(u_i v_c)} \\
&= \ldots \\
&= u_0 - \sum_{x=1}^{V} \frac{exp(u_x^T v_c)}{\sum_{w=1}^{V} exp(u_w^T v_c)} \cdot u_x \\
&= u_0 - \sum_{x=1}^{V} p(u_x|v_c).u_x \\
&= u_0 - E[u_x]
\end{aligned} \tag{4}$$

1. Let V be the size of the vocabulary, $M^{(V)} \in \mathbb{R}^{n \times V}$ our embedded matrix of center words and $M^{(U)} \in \mathbb{R}^{n \times V}$ our embedded matrix of context words.

1. Let V be the size of the vocabulary, $M^{(V)} \in \mathbb{R}^{n \times V}$ our embedded matrix of center words and $M^{(U)} \in \mathbb{R}^{n \times V}$ our embedded matrix of context words.
2. Generate the one-hot input vector for the center word. $x \in \mathbb{R}^{V}$

1. Let V be the size of the vocabulary, $M^{(V)} \in \mathbb{R}^{n \times V}$ our embedded matrix of center words and $M^{(U)} \in \mathbb{R}^{n \times V}$ our embedded matrix of context words.
2. Generate the one-hot input vector for the center word. $x \in \mathbb{R}^{V}$
3. Get the embedded word vector for the center word: $v_c = M^{(V)} \cdot x \in \mathbb{R}^{n}$

1. Let V be the size of the vocabulary, $M^{(V)} \in \mathbb{R}^{n \times V}$ our embedded matrix of center words and $M^{(U)} \in \mathbb{R}^{n \times V}$ our embedded matrix of context words.
2. Generate the one-hot input vector for the center word. $x \in \mathbb{R}^V$
3. Get the embedded word vector for the center word: $v_c = M^{(V)} \cdot x \in \mathbb{R}^n$
4. Generate the score vector $z = U^{(V)} v_c$

1. Let V be the size of the vocabulary, $M^{(V)} \in \mathbb{R}^{n \times V}$ our embedded matrix of center words and $M^{(U)} \in \mathbb{R}^{n \times V}$ our embedded matrix of context words.
2. Generate the one-hot input vector for the center word. $x \in \mathbb{R}^{V}$
3. Get the embedded word vector for the center word: $v_c = M^{(V)} \cdot x \in \mathbb{R}^{n}$
4. Generate the score vector $z = U^{(V)} v_c$
5. Turn the score vector into probabilities: $\hat{y} = softmax(z)$

1. Let V be the size of the vocabulary, $M^{(V)} \in \mathbb{R}^{n \times V}$ our embedded matrix of center words and $M^{(U)} \in \mathbb{R}^{n \times V}$ our embedded matrix of context words.
2. Generate the one-hot input vector for the center word. $x \in \mathbb{R}^V$
3. Get the embedded word vector for the center word: $v_c = M^{(V)} \cdot x \in \mathbb{R}^n$
4. Generate the score vector $z = U^{(V)} v_c$
5. Turn the score vector into probabilities: $\hat{y} = softmax(z)$
6. Update $M^{(V)}$ and $M^{(U)}$:

1. Let V be the size of the vocabulary, $M^{(V)} \in \mathbb{R}^{n \times V}$ our embedded matrix of center words and $M^{(U)} \in \mathbb{R}^{n \times V}$ our embedded matrix of context words.
2. Generate the one-hot input vector for the center word. $x \in \mathbb{R}^V$
3. Get the embedded word vector for the center word: $v_c = M^{(V)} \cdot x \in \mathbb{R}^n$
4. Generate the score vector $z = U^{(V)} v_c$
5. Turn the score vector into probabilities: $\hat{y} = softmax(z)$
6. Update $M^{(V)}$ and $M^{(U)}$:
   - $M^{(U)}_{new} = M^{(U)}_{old} - \alpha \nabla_U J$

1. Let V be the size of the vocabulary, $M^{(V)} \in \mathbb{R}^{n \times V}$ our embedded matrix of center words and $M^{(U)} \in \mathbb{R}^{n \times V}$ our embedded matrix of context words.
2. Generate the one-hot input vector for the center word. $x \in \mathbb{R}^V$
3. Get the embedded word vector for the center word: $v_c = M^{(V)} \cdot x \in \mathbb{R}^n$
4. Generate the score vector $z = U^{(V)} v_c$
5. Turn the score vector into probabilities: $\hat{y} = softmax(z)$
6. Update $M^{(V)}$ and $M^{(U)}$:
   - $M^{(U)}_{new} = M^{(U)}_{old} - \alpha \nabla_U J$
   - $M^{(V)}_{new} = M^{(V)}_{old} - \alpha \nabla_V J$

Table 8: *Examples of the word pair relationships, using the best word vectors from Table 4 (Skip-gram model trained on 783M words with 300 dimensionality).*

Relationship	Example 1	Example 2	Example 3
France - Paris	Italy: Rome	Japan: Tokyo	Florida: Tallahassee
big - bigger	small: larger	cold: colder	quick: quicker
Miami - Florida	Baltimore: Maryland	Dallas: Texas	Kona: Hawaii
Einstein - scientist	Messi: midfielder	Mozart: violinist	Picasso: painter
Sarkozy - France	Berlusconi: Italy	Merkel: Germany	Koizumi: Japan
copper - Cu	zinc: Zn	gold: Au	uranium: plutonium
Berlusconi - Silvio	Sarkozy: Nicolas	Putin: Medvedev	Obama: Barack
Microsoft - Windows	Google: Android	IBM: Linux	Apple: iPhone
Microsoft - Ballmer	Google: Yahoo	IBM: McNealy	Apple: Jobs
Japan - sushi	Germany: bratwurst	France: tapas	USA: pizza

source: https://arxiv.org/pdf/1301.3781.pdf

Czech + currency	Vietnam + capital	German + airlines	Russian + river	French + actress
koruna	Hanoi	airline Lufthansa	Moscow	Juliette Binoche
Check 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

Table 5: Vector compositionality using element-wise addition. Four closest tokens to the sum of two vectors are shown, using the best Skip-gram model.

source: https://arxiv.org/pdf/1301.3781.pdf

word2vec - Applications

- Dependency parsers

word2vec - Applications

- Dependency parsers
- Named Entity recognition

word2vec - Applications

- Dependency parsers
- Named Entity recognition
- Sentiment analysis
- Predict the person who wrote the text

word2vec - Applications

- Dependency parsers
- Named Entity recognition
- Sentiment analysis
- Predict the person who wrote the text
- Information retrieval: Extract features.

Problems with word2vec

Let's look at our previous model:

Problems with word2vec

Let's look at our previous model:

- Vector embeddings: 300 components

Problems with word2vec

Let's look at our previous model:

- Vector embeddings: 300 components
- Vocabulary size: 10.000 components

Problems with word2vec

Let's look at our previous model:

- Vector embeddings: 300 components
- Vocabulary size: 10.000 components
- One hidden layer matrix with size: 300×10.000

Problems with word2vec

Let's look at our previous model:

- Vector embeddings: 300 components
- Vocabulary size: 10.000 components
- One hidden layer matrix with size: 300×10.000
- And an output layer with the same matrix size

Let's look at our previous model:

- Vector embeddings: 300 components
- Vocabulary size: 10.000 components
- One hidden layer matrix with size: 300×10.000
- And an output layer with the same matrix size
- Total parameters to tune: 6 million!

Distributed Representations of Words and Phrases and their Compositionality

Distributed Representations of Words and Phrases and their Compositionality

- Sub sampling of frequent words

Distributed Representations of Words and Phrases and their Compositionality

- Sub sampling of frequent words
- Negative Sampling

Sub Sampling of frequent words

- Stopwords don't add meaning as context words

Sub Sampling of frequent words

- Stopwords don't add meaning as context words
- We have more samples of stopwords than we need

Idea:

Sub Sampling of frequent words

- Stopwords don't add meaning as context words
- We have more samples of stopwords than we need

Idea:

- Implement (sub)sampling rate

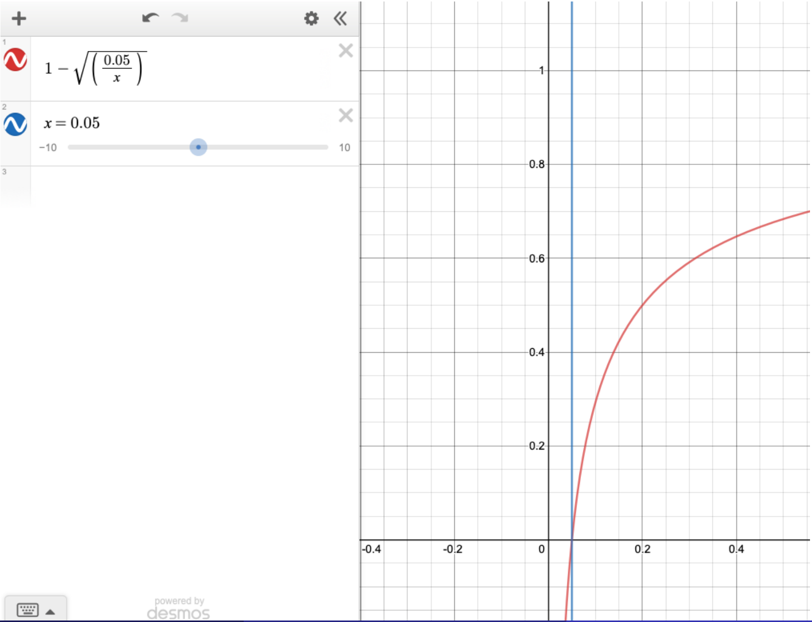
Sub Sampling of frequent words

- Stopwords don't add meaning as context words
- We have more samples of stopwords than we need

Idea:

- Implement (sub)sampling rate
- $P(w_i) = 1 - \sqrt{\frac{t}{f(w_i)}}$

Sub Sampling of frequent words



Negative Sampling

Negative Sampling

- Remember: Each training input will tweak 3 million parameters!

Idea:

- Select small n of "negative" words (e.g. 5).

Negative Sampling

- Remember: Each training input will tweak 3 million parameters!

Idea:

- Select small n of "negative" words (e.g. 5).
- Update their weights (as well as the positive word)

Negative Sampling

- Remember: Each training input will tweak 3 million parameters!

Idea:

- Select small n of "negative" words (e.g. 5).
- Update their weights (as well as the positive word)
- Total of 6 output neurons to update + 1 hidden neuron.
- That's 1800 weights = 0.06% of the 3M weights.

Negative Sampling

How to choose negative samples?

$$P(w_i) = \frac{f(w_i)}{\sum_{j=0}^{V} f(w_j)} \tag{5}$$

Negative Sampling

How to choose negative samples?

$$P(w_i) = \frac{f(w_i)}{\sum_{j=0}^{V} f(w_j)} \tag{5}$$

$$P(w_i) = \frac{f(w_i)^{3/4}}{\sum_{j=0}^{V} f(w_j)^{3/4}} \tag{6}$$

The end.

Thank you

References

- CS 224n:"NLP with Deep Learning" http://cs224n.stanford.edu
- "word2vec tutorial" http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/