Distributed Representation of Words and Phrases and their Compositionality

T Mikolov, I Sutskever, K Chen, G Corrado, J Dean

Presented by Pranav Shetty, Anish Khazane

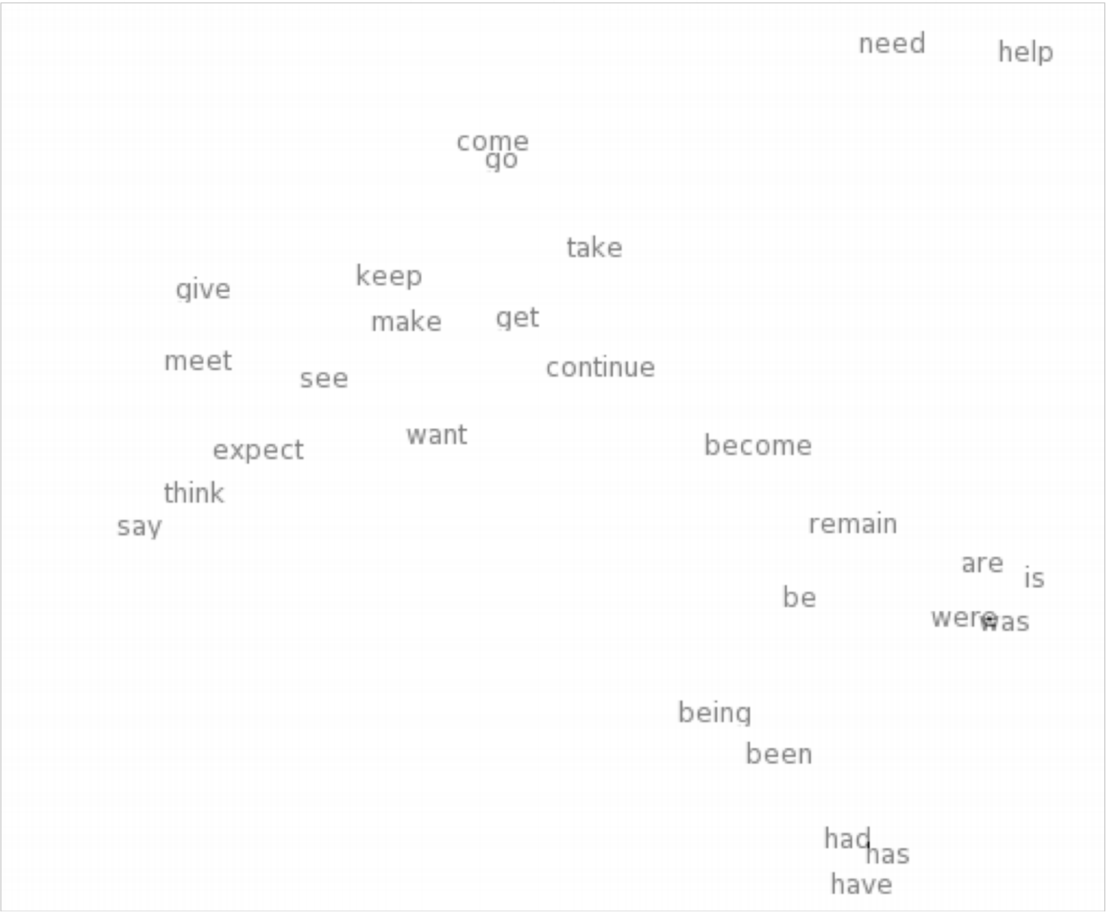
Overview

1. What are Word Vectors? Previous Approaches?
2. Introduce Skip-Gram model
3. Discuss Skip-Gram Shortcomings
4. Introduce Paper Improvements on Skip-Gram
    a. Negative Sampling
    b. Subsampling of Frequent Words
5. Discuss Results from Paper
6. Drawbacks of Word2Vec/Skip-Gram
8. Summary of Discussion

Word Vectors, Previous Approaches

Word meaning as a neural word vector – visualization

$$\textit{expect} = \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_{n-1} \\ v_n \end{bmatrix}$$



*Adapted from CS224d slides at Stanford

Representing words as discrete symbols

In traditional NLP, we regard words as discrete symbols:
hotel, conference, motel – a localist representation

Means one 1, the rest 0s

Words can be represented by one-hot vectors:

motel = [0 0 0 0 0 0 0 0 0 0 1 0 0 0 0]
hotel = [0 0 0 0 0 0 0 1 0 0 0 0 0 0 0]

Vector dimension = number of words in vocabulary (e.g., 500,000)

*Adapted from CS224d slides at Stanford

Problem with words as discrete symbols

motel = [0 0 0 0 0 0 0 0 0 0 1 0 0 0 0]
hotel = [0 0 0 0 0 0 0 1 0 0 0 0 0 0 0]

These two vectors are orthogonal.

There is no natural notion of **similarity** for one-hot vectors!

Instead: learn to encode similarity in the vectors themselves

$$\text{obama} = \underbrace{\underset{|V|}{[\overset{w_1}{0} \ldots \ldots \overset{\text{obama}}{1} \ldots \ldots \overset{w_{|V|}}{0}]}}_{\text{“one-hot” vector}} \qquad \text{obama} = \underbrace{\underset{m \ll |V|}{[\overset{\text{feature}_1}{0.12} \ldots - \overset{\text{feature}_m}{0.25}]}}_{\text{feature vector}}$$

*Adapted from CS224d slides at Stanford, DaSciM slides

We also need an approach that *scales*

- Dimensionality reduction and neural network-based approaches (LSA, Bengio's NNLM) use large hidden-layers to compute embeddings
- Vocab Sizes for popular datasets
  - 20K Standard Speech
  - 400K/6B Wikipedia Corpus
  - 2.2M/840B common crawl corpus
  - 13M Google Corpus
- Using multiple hidden layers, matrices, etc to compute representations is **computationally expensive.** We need something more efficient.

Skip-Gram Overview

Representing words by their context

- <u>Distributional semantics</u>: **A word's meaning is given by the words that frequently appear close-by**
  - *"You shall know a word by the company it keeps"* (J. R. Firth 1957: 11)
  - One of the most successful ideas of modern statistical NLP!
- When a word *w* appears in a text, its **context** is the set of words that appear nearby (within a fixed-size window).
- Use the many contexts of *w* to build up a representation of *w*

…government debt problems turning into	*banking*	*crises as happened in 2009…*
…saying that Europe needs unified	*banking*	*regulation to replace the hodgepodge…*
…India has just given its	*banking*	*system a shot in the arm…*

These context words will represent ***banking***

*Adapted from CS224d slides at Stanford

Skip-Gram

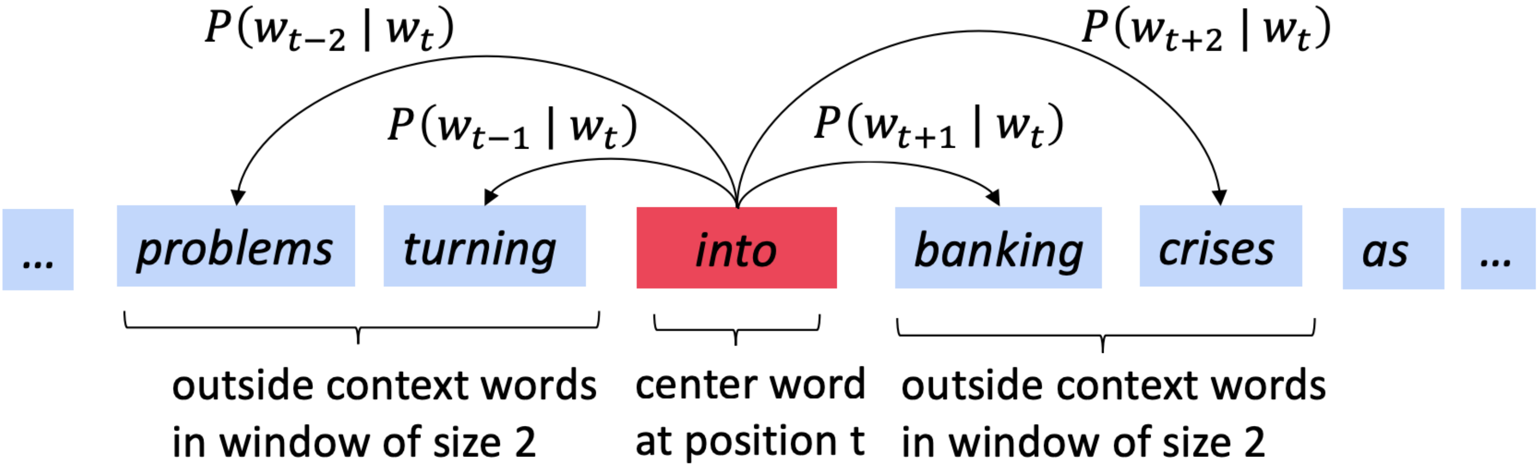
- *"Efficient Estimation of Word Representations in Vector Space" (Mikolov, et al.)*
  - Presents two frameworks for learning word vectors
    - Skip-Gram
    - CBOW (Continuous Bag of Words)
- Today's paper focuses on Skip-Gram:
  - We have a large corpus of text
  - Every word in a fixed vocabulary is represented by a vector
  - Go through each position *t* in the text, which has a center word *c* and context ("outside") words *o*
  - Use the similarity of the word vectors for *c* and *o* to calculate the probability of *o* given *c* (or vice versa)
  - Keep adjusting the word vectors to maximize this probability

*Adapted from CS224d slides at Stanford

We want to minimize the objective function:

$$J(\theta) = -\frac{1}{T}\sum_{t=1}^{T}\sum_{\substack{-m\le j\le m\\ j\neq 0}} \log P\left(w_{t+j} \mid w_t; \theta\right)$$

<u>Question:</u> How to calculate $P\left(w_{t+j} \mid w_t; \theta\right)$?

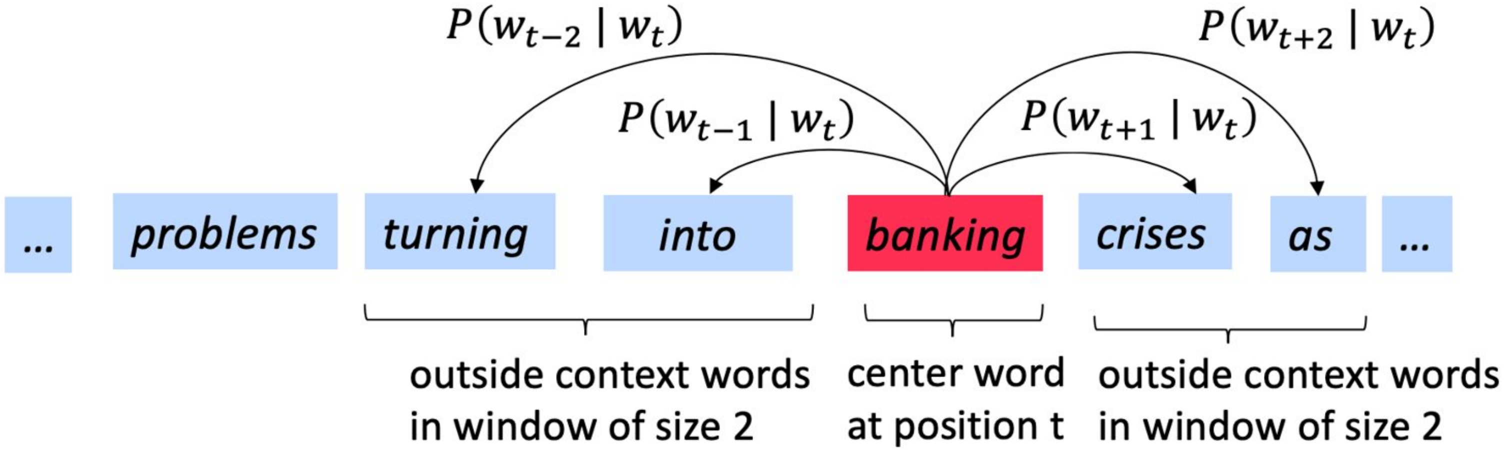


*Adapted from CS224d slides at Stanford

We want to minimize the objective function:

$$J(\theta) = -\frac{1}{T}\sum_{t=1}^{T}\sum_{\substack{-m\le j\le m\\ j\ne 0}} \log P\left(w_{t+j} \mid w_t; \theta\right)$$

<u>Question:</u> How to calculate $P\left(w_{t+j} \mid w_t; \theta\right)$?



*Adapted from CS224d slides at Stanford

Exponentiation makes anything positive

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

Dot product compares similarity of *o* and *c*.

$u^T v = u.v = \sum_{i=1}^{n} u_i v_i$

Larger dot product = larger probability

Normalize over entire vocabulary to give probability distribution

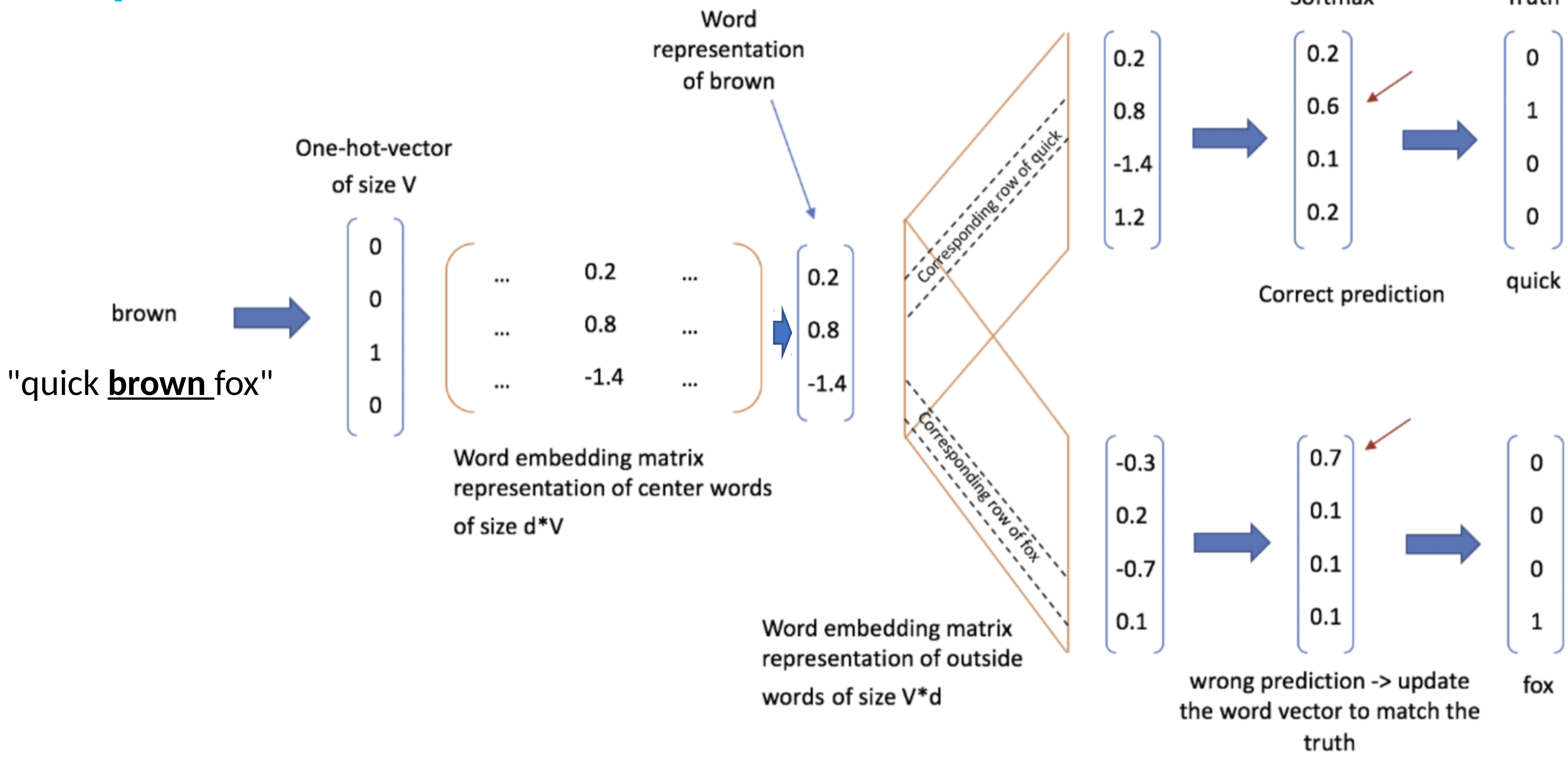
- This is an example of the **softmax function** $\mathbb{R}^n \rightarrow \mathbb{R}^n$

$$\text{softmax}(x_i) = \frac{\exp(x_i)}{\sum_{j=1}^{n} \exp(x_j)} = p_i$$

- The softmax function maps arbitrary values x_i to a probability distribution p_i
  - "max" because amplifies probability of largest x_i
  - "soft" because still assigns some probability to smaller x_i
  - Frequently used in Deep Learning

*Adapted from CS224d slides at Stanford

Skip-Gram Architecture



*Adapted from CS224d slides at Stanford

Problems with basic Skip-Gram

- Softmax objective function is computationally very intensive

$$p(w_O|w_I) = \frac{\exp\left({v'_{w_O}}^\top v_{w_I}\right)}{\sum_{w=1}^{W} \exp\left({v'_w}^\top v_{w_I}\right)} \longrightarrow O(W),\ W \sim 10^5 \text{-} 10^7$$

- Idiomatic phrases not well represented i.e meaning of 'Air Canada' cannot be distinguished 'Air' and 'Canada'

'Air Canada' ≠ ('Air' , 'Canada')

- Very frequent words provide less information than rare words. Thus vector representations of very frequent words do not change significantly on repeated training

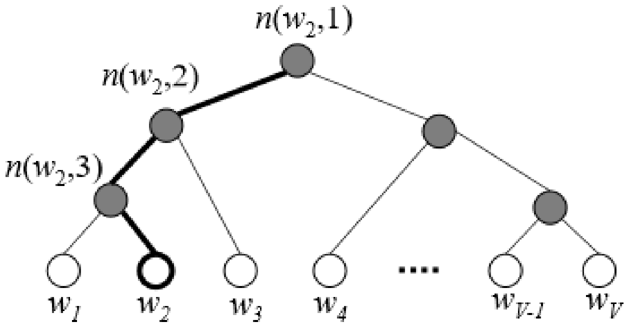
	Doc A	Doc B	Doc C	Doc D
"NBA"	10	0	0	4
"Trump"	0	15	0	0
"Loss"	0	0	8	0
"the"	20	30	0	40

*Adapted from class notes

Improvements on Skip-Gram

Hierarchical Softmax

- Computation of Loss function done by traversing a binary tree



$$p(n, right) = \sigma(-{v'_n}^T h) = \sigma(-{v'_n}^T v_{w_I})$$

$$p(n, left) = \sigma({v'_n}^T h) = \sigma({v'_n}^T v_{w_I})$$

$$p(w|w_I) = \prod_{j=1}^{L(w)-1} \sigma(\langle n(w, j+1) = ch(n(w,j)) \rangle \, {v'_n}^T v_{w_I})$$

- Average length of path from root to leaf ~ log (W)
- Each node in the tree has an associated vector representation v_n'

https://towardsdatascience.com/hierarchical-softmax-and-negative-sampling-short-notes-worth-telling-2672010dbe08

Negative Sampling

- Intuitively, we try to construct a loss function which maximizes the probability of a word and context being in the corpus if it indeed is and maximize the probability if it is not

$$= \underset{\theta}{\operatorname{argmax}} \sum_{(w,c)\in D} \log P(D=1|w,c,\theta) + \sum_{(w,c)\in \tilde{D}} \log(1 - P(D=1|w,c,\theta))$$

$$= \underset{\theta}{\operatorname{argmax}} \sum_{(w,c)\in D} \log \frac{1}{1+\exp(-u_w^T v_c)} + \sum_{(w,c)\in \tilde{D}} \log(1 - \frac{1}{1+\exp(-u_w^T v_c)})$$

$$= \underset{\theta}{\operatorname{argmax}} \sum_{(w,c)\in D} \log \frac{1}{1+\exp(-u_w^T v_c)} + \sum_{(w,c)\in \tilde{D}} \log(\frac{1}{1+\exp(u_w^T v_c)})$$

- When simplified, this works out to:

$$\log \sigma({v'_{w_O}}^\top v_{w_I}) + \sum_{i=1}^{k} \mathbb{E}_{w_i \sim P_n(w)} \left[\log \sigma(-{v'_{w_i}}^\top v_{w_I}) \right]$$

- The k negative words are sampled from a probability distribution $P_n(w)$

*Adapted from CS224d notes, Stanford

Phrases

- Unigram and bigram counts computed using the below metric:

$$\text{score}(w_i, w_j) = \frac{\text{count}(w_i w_j) - \delta}{\text{count}(w_i) \times \text{count}(w_j)}$$

- Here delta is a discounting coefficient which prevents too many phrases consisting of very infrequent words to be formed
- Bigrams with a score above a certain threshold are used as phrases

Subsampling of frequent words

- The i^{th} word is discarded with probability $P(w_i)$ as given below:

$$P(w_i) = 1 - \sqrt{\frac{t}{f(w_i)}}$$

Model and dataset

Model/dataset parameter	Value
Word Vector Dimension	300
Training Dataset	Large Newspaper corpus
Vocabulary size	~692 k tokens
Minimum number of occurrences to create word vector	5 occurrences
Context size	5 words

Analogies with Word Vectors

- Paris : France : Rome : ____ ?

```
vector("France") – vector("Paris") = answer_vector – vector("Rome")
```

Semantic Analogy

Input	Result Produced
dancing : danced : : decreasing	decreased
dancing : danced : : describing	described
dancing : danced : : enhancing	enhanced
dancing : danced : : falling	fell
dancing : danced : : feeding	fed
dancing : danced : : flying	flew
dancing : danced : : generating	generated
dancing : danced : : going	went
dancing : danced : : hiding	hid
dancing : danced : : hitting	hit

Syntactic Analogy

*Table adapted from CS224d notes, Stanford

Results

Method	Time [min]	Syntactic [%]	Semantic [%]	Total accuracy [%]
NEG-5	38	63	54	59
NEG-15	97	63	58	**61**
HS-Huffman	41	53	40	47
The following results use 10^{-5} subsampling				
NEG-5	14	61	58	60
NEG-15	36	61	61	**61**
HS-Huffman	21	52	59	55

- Using a context window of 15 gave much better performance than a context window of 5
- Negative sampling has better accuracy than hierarchical softmax
- With sub sampling, accuracy remains the same or better but computational time drops

*Table from "Distributed Representations of Words and Phrases and their Compositionality"

Phrase Analogy Results

Method	Dimensionality	No subsampling [%]	10^{-5} subsampling [%]
NEG-5	300	24	27
NEG-15	300	27	42
HS-Huffman	300	19	**47**

- Larger Context Window -> Better performance
- Hierarchical softmax changes from worst to best upon down sampling

Additive Compositionality

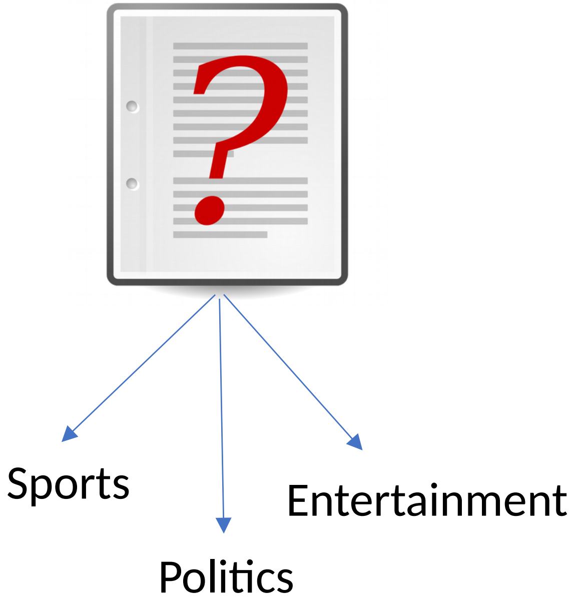
- Word Vectors represent a distribution of the contexts in which they appear
- Addition of 2 word vectors may be semantically close to a meaningful word vector

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

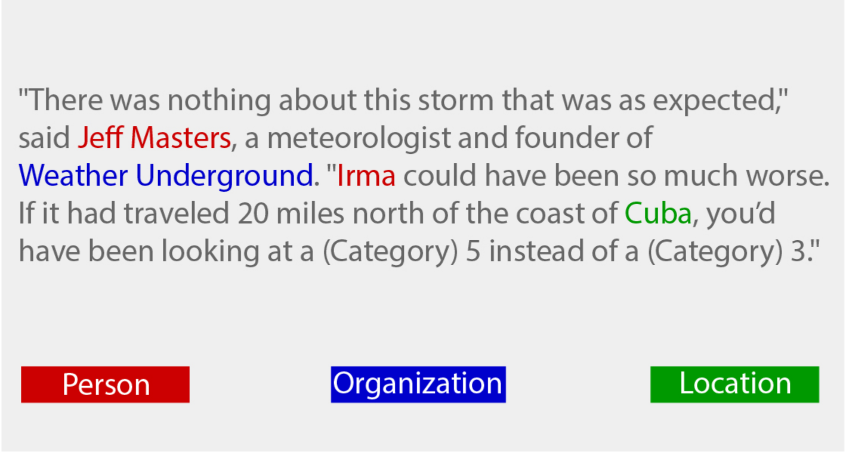
*Tables from "Distributed Representations of Words and Phrases and their Compositionality"

Applications of Word2Vec

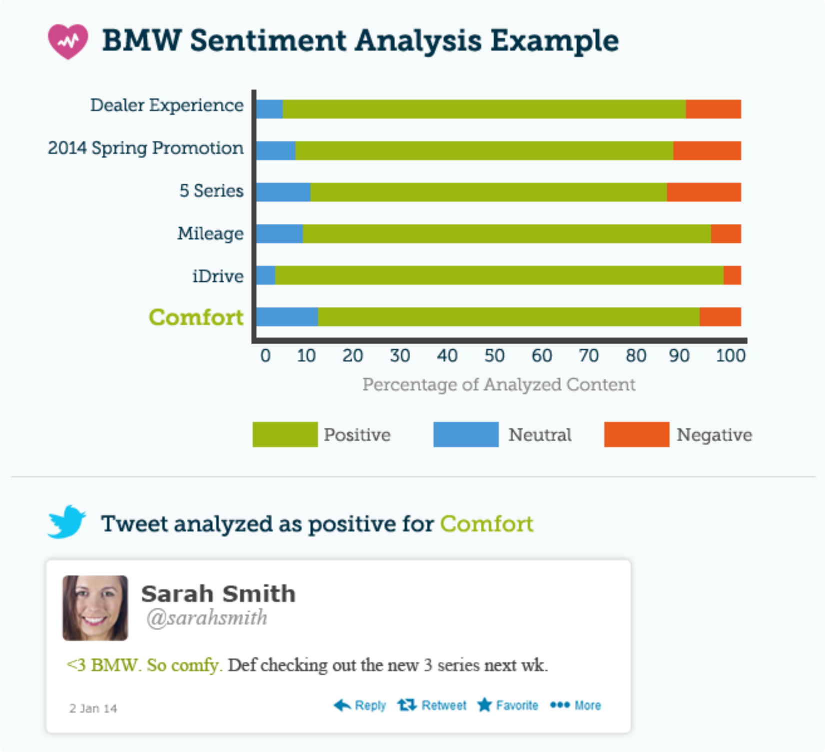
Doc2Vec: Document Classification



Named Entity Recognition



Sentiment analysis



http://cs-wordpress.s3.amazonaws.com/crowdsource-v4/uploads/2013/11/sentiment-analysis-ui.png
http://blog.paralleldots.com/wp-content/uploads/2017/09/Named-entity-recognition-Paralleldots.jpg

Word2Vec : Drawbacks

- Single representation for a word irrespective of the context in which it occurs
  - He stood by the **bank** of the river
  - He went to the **bank** to withdraw money
- No representation for words outside initial training corpus
- Does not make efficient use of statistics of co-occurrence in the corpus – **Glove**
- Morphological features of a word ie *fast*, *faster*, *fastest* not explicitly accounted for and must be learnt from context – Character level embeddings like **Elmo**

Summary of results

- Down sampling and negative sampling allow Skip-gram to be trained on a much larger corpus thus improving the model
- Down sampling leads to better representations for rarer words
- Word vectors can be combined linearly to give meaningful results

References

- CS 224d slides + notes "Deep Learning for NLP" at Stanford
  - Lecture 1: http://web.stanford.edu/class/cs224n/slides/cs224n-2019-lecture01-wordvecs1.pdf
  - Lecture 2: http://web.stanford.edu/class/cs224n/slides/cs224n-2019-lecture02-wordvecs2.pdf
  - Notes 1: https://cs224d.stanford.edu/lecture_notes/notes1.pdf
  - Notes 2: https://cs224d.stanford.edu/lecture_notes/notes2.pdf
- Data Science and Mining Team Slides ~ Word Embeddings
  - http://www.lix.polytechnique.fr/~anti5662/word_embeddings_intro_tixier.pdf
- Distributed Representations of Words and Phrases and their Compositionality (Mikolov et al.)
  - https://papers.nips.cc/paper/5021-distributed-representations-of-words-and-phrases-and-their-compositionality.pdf