Understanding Natural Language with Word Vectors (and Python)

@MarcoBonzanini

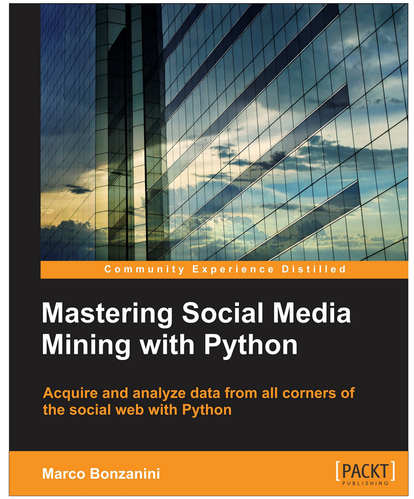
PyData Bristol 1st Meetup

March 2018

Nice to meet you

PyData London

April 27-29



WORD EMBEDDINGS?

Word Embeddings

=

Word Vectors

=

Distributed Representations

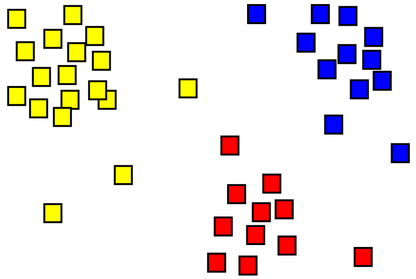
Why should you care?

Why should you care?

Data representation is crucial

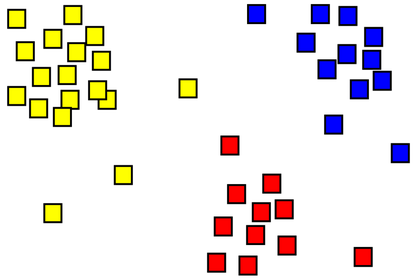
Applications

Applications



Classification

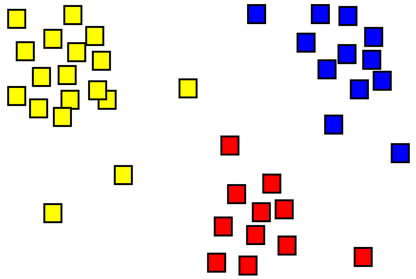
Applications



Classification

Recommender Systems

Applications

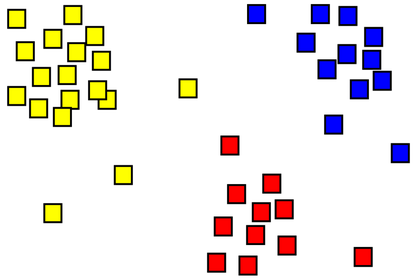


Classification

Recommender Systems

Search Engines

Applications

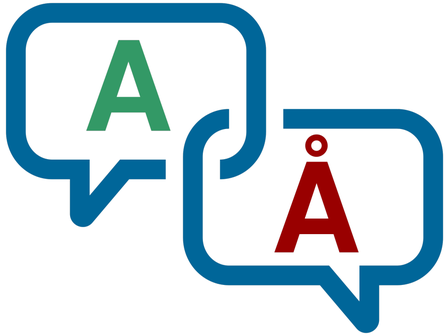


Classification

Recommender Systems

Search Engines

Machine Translation



One-hot Encoding

One-hot Encoding

```
Rome   = [1, 0, 0, 0, 0, 0, …, 0]

Paris  = [0, 1, 0, 0, 0, 0, …, 0]

Italy  = [0, 0, 1, 0, 0, 0, …, 0]

France = [0, 0, 0, 1, 0, 0, …, 0]
```

One-hot Encoding


```
Rome   = [1, 0, 0, 0, 0, 0, …, 0]

Paris  = [0, 1, 0, 0, 0, 0, …, 0]

Italy  = [0, 0, 1, 0, 0, 0, …, 0]

France = [0, 0, 0, 1, 0, 0, …, 0]
```

One-hot Encoding


```
Rome   = [1, 0, 0, 0, 0, 0, …, 0]

Paris  = [0, 1, 0, 0, 0, 0, …, 0]

Italy  = [0, 0, 1, 0, 0, 0, …, 0]

France = [0, 0, 0, 1, 0, 0, …, 0]
```

Word Embeddings

Word Embeddings

```
Rome   = [0.91, 0.83, 0.17, …, 0.41]

Paris  = [0.92, 0.82, 0.17, …, 0.98]

Italy  = [0.32, 0.77, 0.67, …, 0.42]

France = [0.33, 0.78, 0.66, …, 0.97]
```

Word Embeddings

n. dimensions << vocabulary size

```
Rome   = [0.91, 0.83, 0.17, …, 0.41]

Paris  = [0.92, 0.82, 0.17, …, 0.98]

Italy  = [0.32, 0.77, 0.67, …, 0.42]

France = [0.33, 0.78, 0.66, …, 0.97]
```

Word Embeddings

```
Rome   = [0.91, 0.83, 0.17, …, 0.41]
Paris  = [0.92, 0.82, 0.17, …, 0.98]
Italy  = [0.32, 0.77, 0.67, …, 0.42]
France = [0.33, 0.78, 0.66, …, 0.97]
```

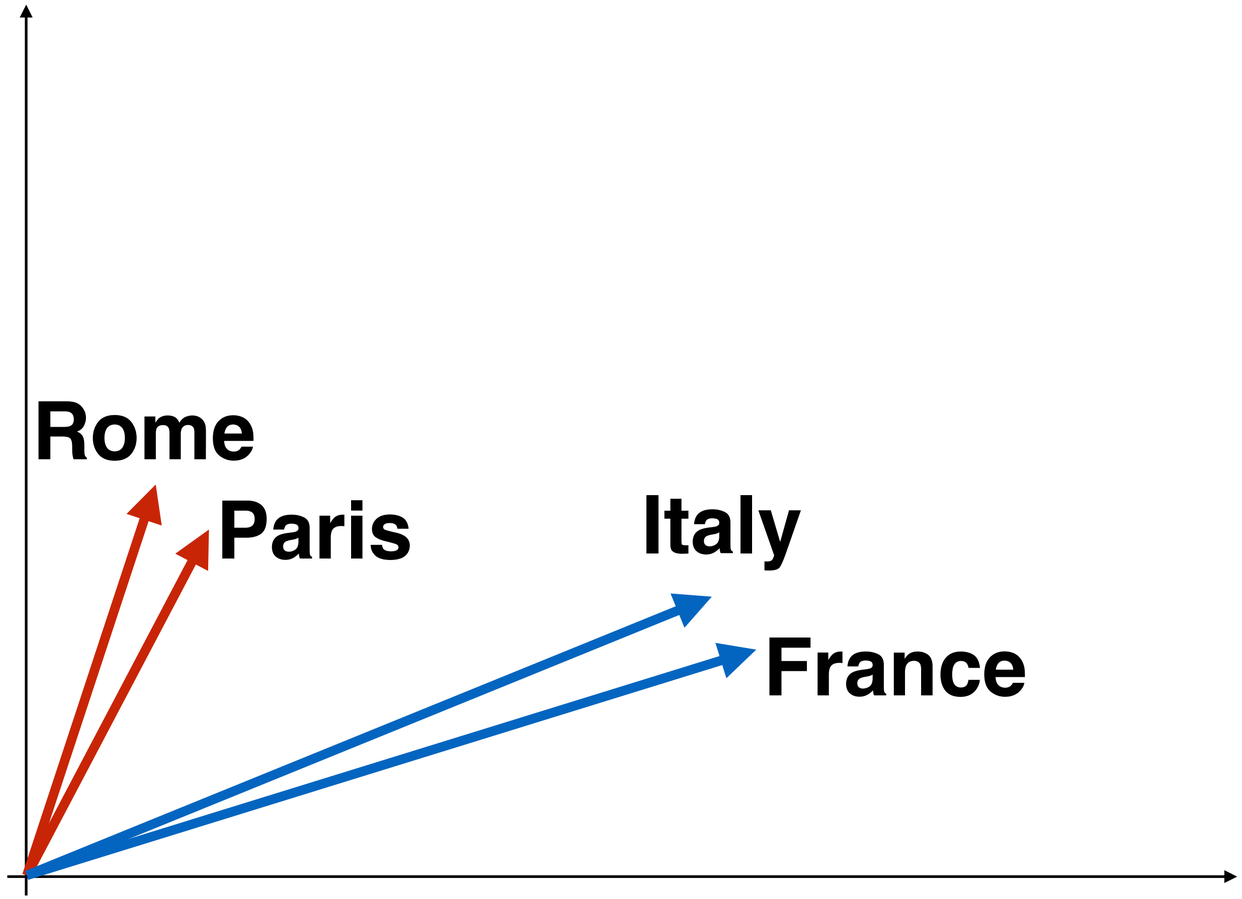
Word Embeddings

```
Rome   = [0.91, 0.83, 0.17, …, 0.41]

Paris  = [0.92, 0.82, 0.17, …, 0.98]

Italy  = [0.32, 0.77, 0.67, …, 0.42]

France = [0.33, 0.78, 0.66, …, 0.97]
```

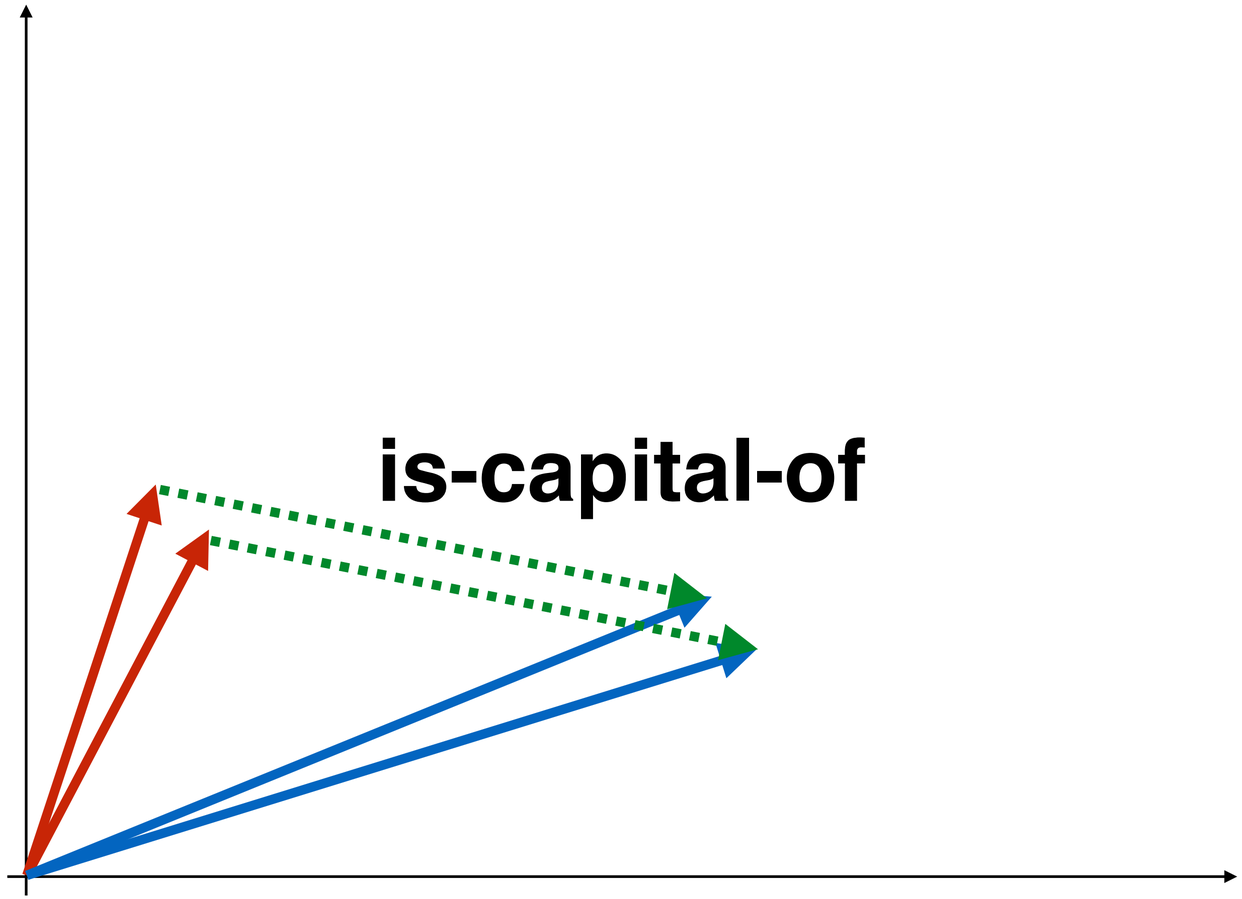
Word Embeddings

```
Rome   = [0.91, 0.83, 0.17, …, 0.41]
Paris  = [0.92, 0.82, 0.17, …, 0.98]
Italy  = [0.32, 0.77, 0.67, …, 0.42]
France = [0.33, 0.78, 0.66, …, 0.97]
```

Word Embeddings

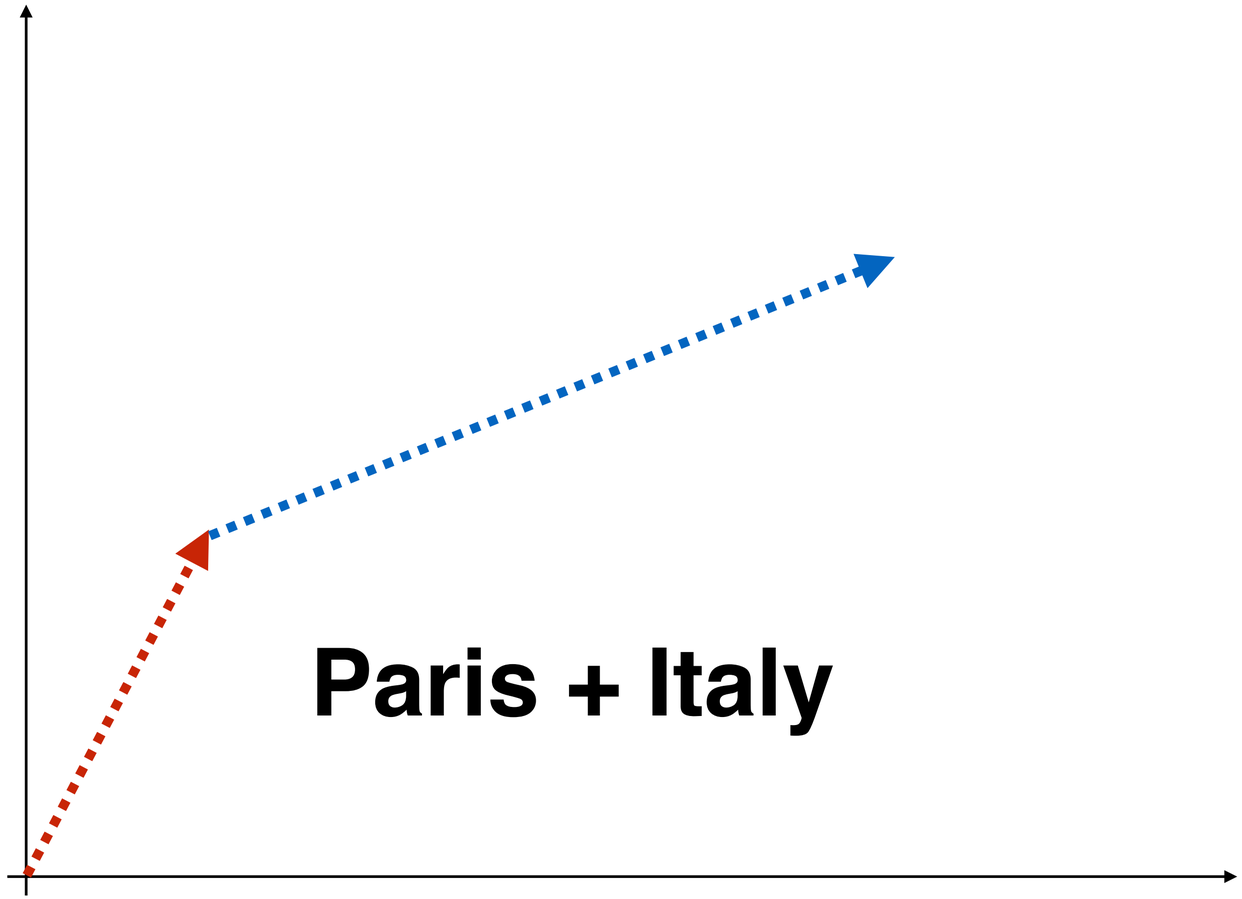


Word Embeddings

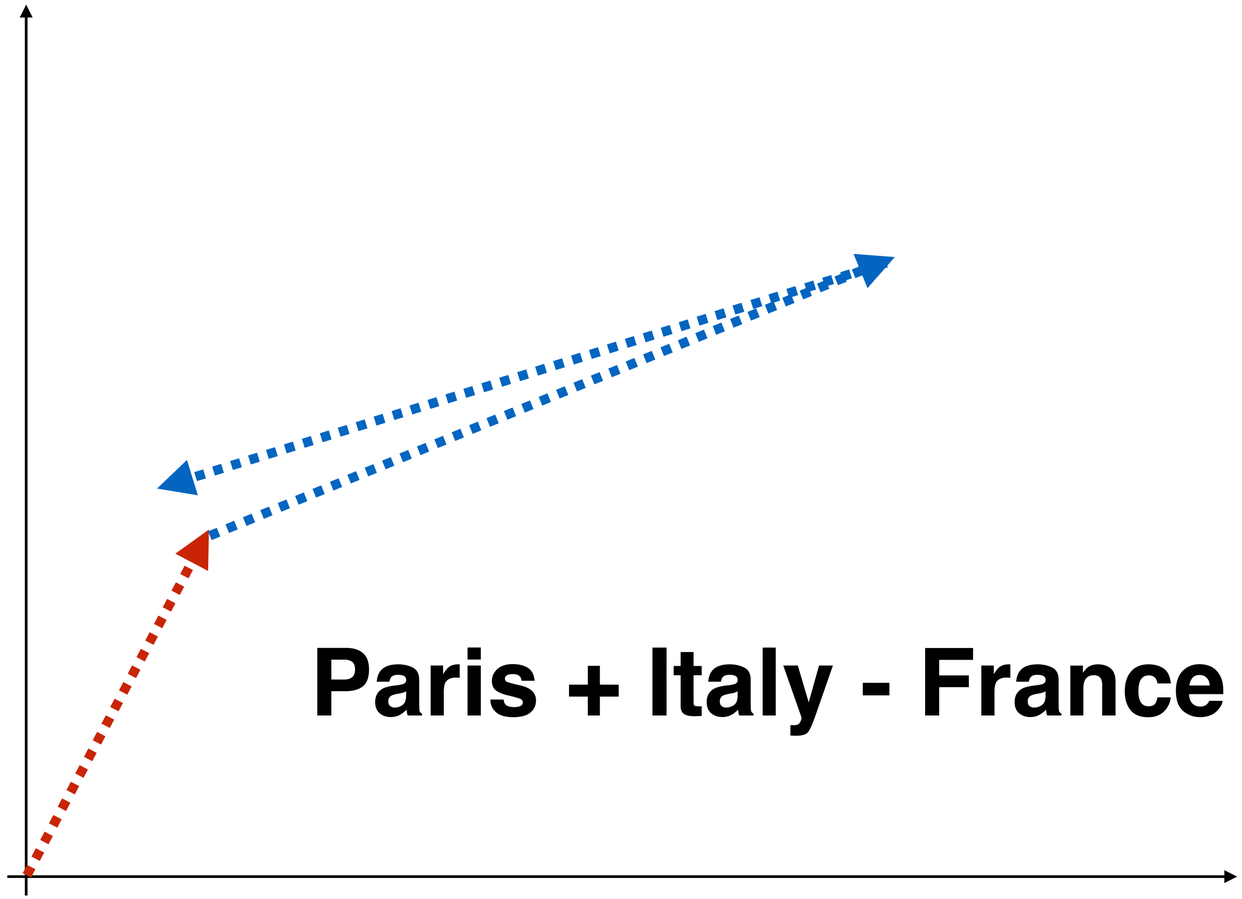


Word Embeddings

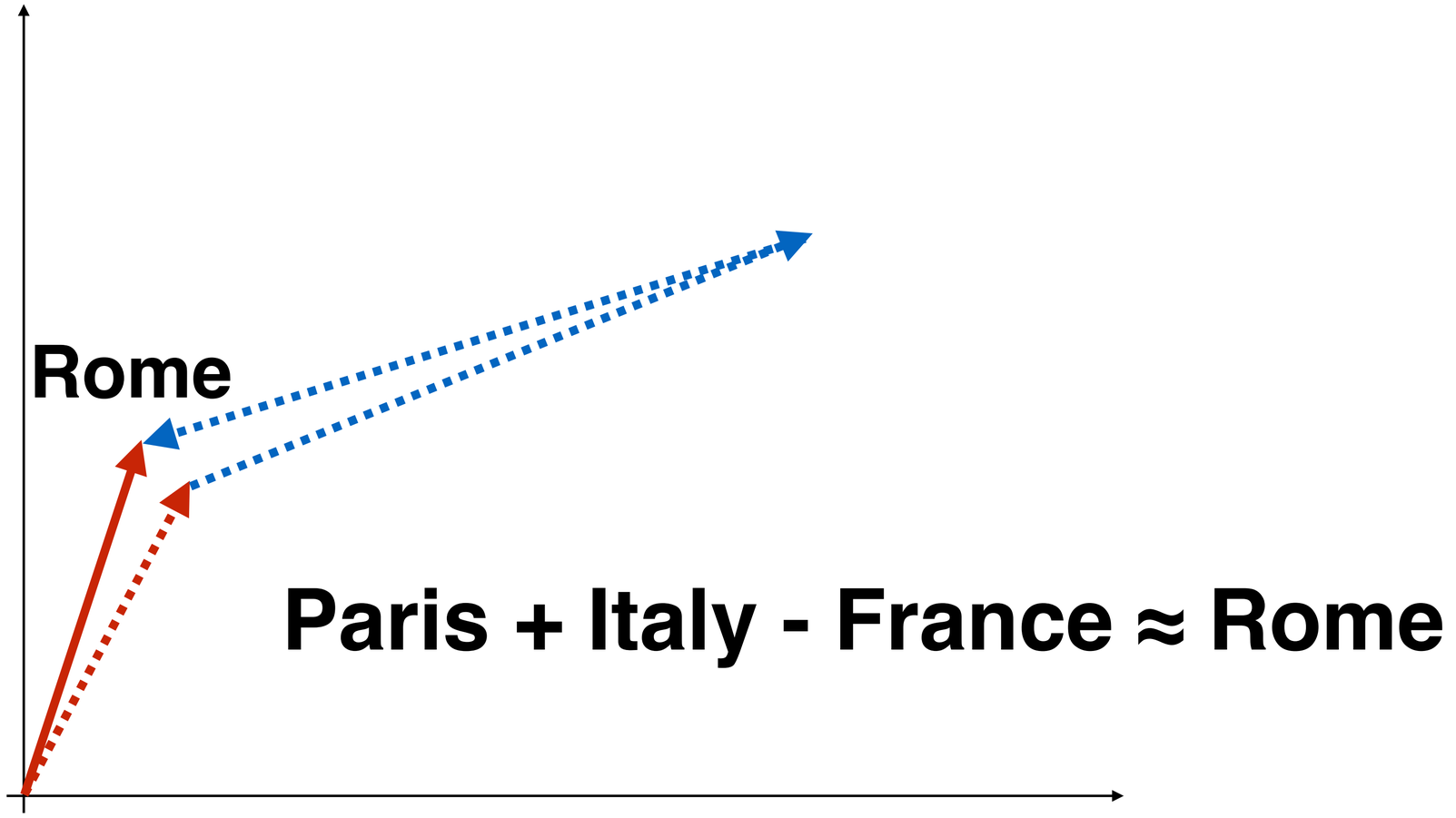
Word Embeddings



Word Embeddings



Word Embeddings



FROM LANGUAGE TO VECTORS?

Distributional Hypothesis

“You shall know a word by the company it keeps.”

–J.R. Firth, 1957

“Words that occur in similar context tend to have similar meaning.”

–Z. Harris, 1954

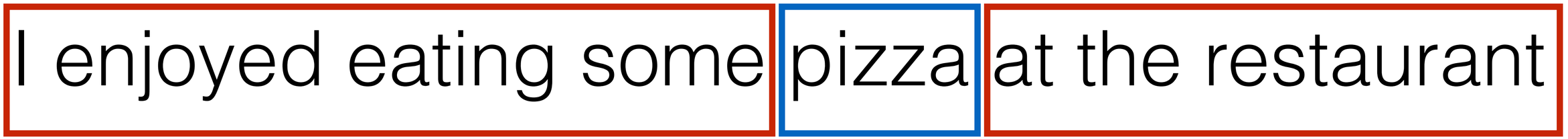
Context ≈ Meaning

I enjoyed eating some pizza at the restaurant

Word

I enjoyed eating some pizza at the restaurant

Word



The company it keeps

I enjoyed eating some pizza at the restaurant

I enjoyed eating some Welsh cake at the restaurant

I enjoyed eating some pizza at the restaurant

I enjoyed eating some Welsh cake at the restaurant

Same Context

I enjoyed eating some pizza at the restaurant

I enjoyed eating some Welsh cake at the restaurant

Same Context

?

=

WORD2VEC

word2vec (2013)

Efficient Estimation of Word Representations in Vector Space

Tomas Mikolov
Google Inc., Mountain View, CA
tmikolov@google.com

Kai Chen
Google Inc., Mountain View, CA
kaichen@google.com

Greg Corrado
Google Inc., Mountain View, CA
gcorrado@google.com

Jeffrey Dean
Google Inc., Mountain View, CA
jeff@google.com

Abstract

We propose two novel model architectures for computing continuous vector representations of words from very large data sets. The quality of these representations is measured in a word similarity task, and the results are compared to the previously best performing techniques based on different types of neural networks. We observe large improvements in accuracy at much lower computational cost, i.e. it takes less than a day to learn high quality word vectors from a 1.6 billion words data set. Furthermore, we show that these vectors provide state-of-the-art performance on our test set for measuring syntactic and semantic word similarities.

Distributed Representations of Words and Phrases and their Compositionality

Tomas Mikolov
Google Inc.
Mountain View
mikolov@google.com

Ilya Sutskever
Google Inc.
Mountain View
ilyasu@google.com

Kai Chen
Google Inc.
Mountain View
kai@google.com

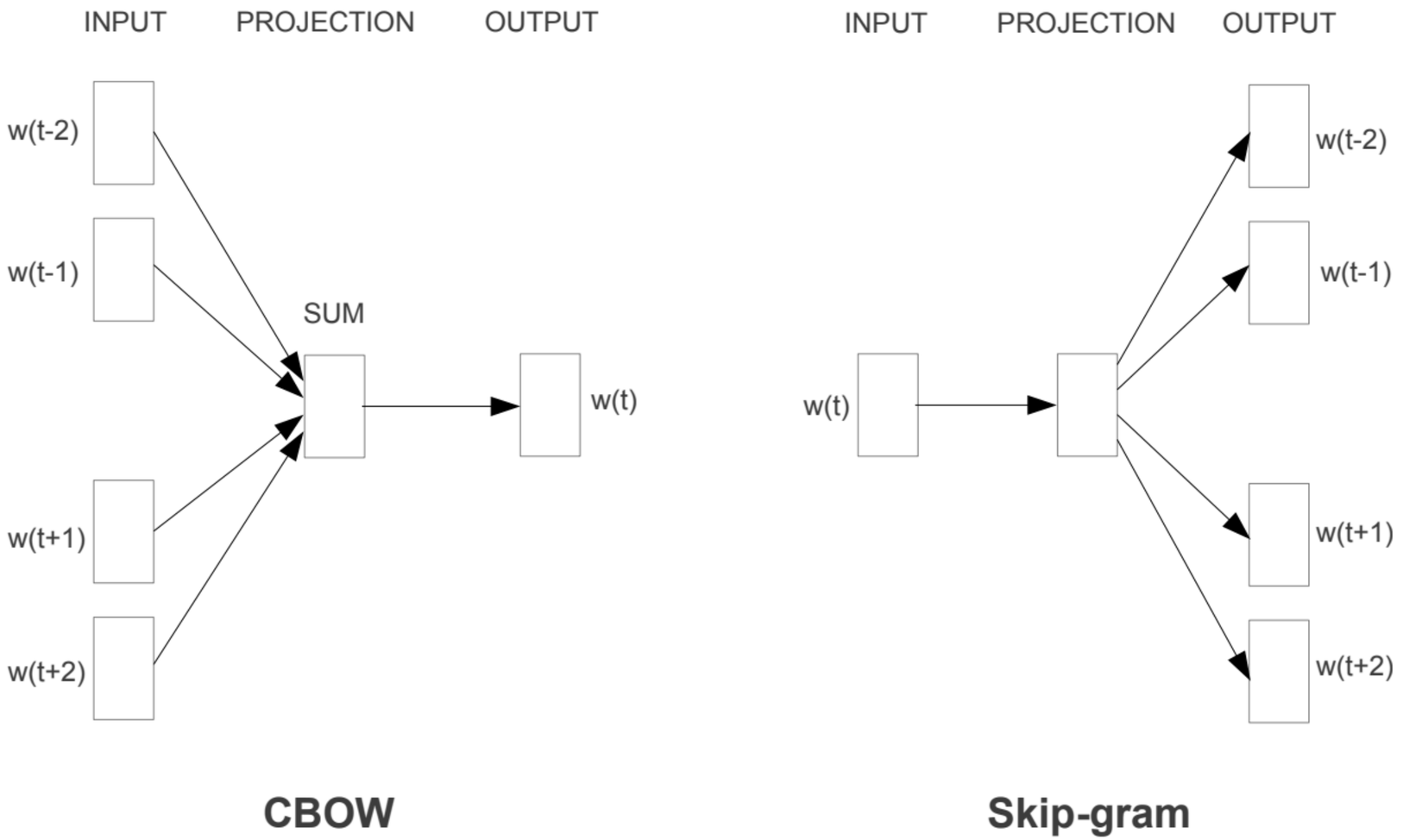
Greg Corrado
Google Inc.
Mountain View
gcorrado@google.com

Jeffrey Dean
Google Inc.
Mountain View
jeff@google.com

Abstract

The recently introduced continuous Skip-gram model is an efficient method for learning high-quality distributed vector representations that capture a large number of precise syntactic and semantic word relationships. In this paper we present several extensions that improve both the quality of the vectors and the training speed. By subsampling of the frequent words we obtain significant speedup and also learn more regular word representations. We also describe a simple alternative to the hierarchical softmax called negative sampling.

word2vec Architecture



Mikolov et al. (2013) Efficient Estimation of Word Representations in Vector Space

Vector Calculation

Vector Calculation

Goal: learn vec(word)

Vector Calculation

Goal: learn vec(word)

1. Choose objective function

Vector Calculation

Goal: learn vec(word)

1. Choose objective function
2. Init: random vectors

Vector Calculation

Goal: learn vec(word)

1. Choose objective function
2. Init: random vectors
3. Run stochastic gradient descent

Vector Calculation

Goal: learn vec(word)

1. Choose objective function
2. Init: random vectors
3. Run stochastic gradient descent

Vector Calculation

Goal: learn vec(word)

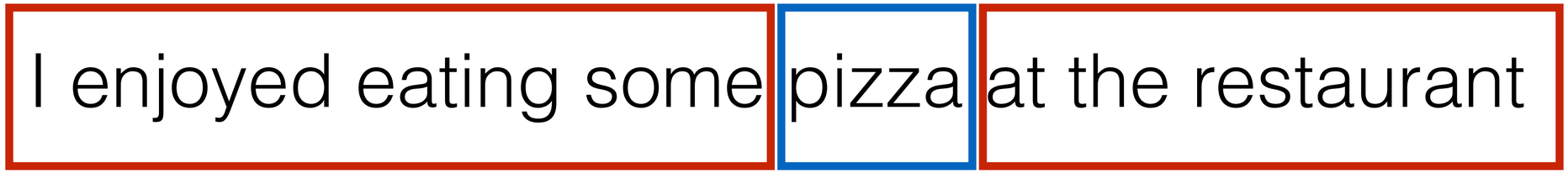
1. Choose objective function
2. Init: random vectors
3. Run stochastic gradient descent

Objective Function

Objective Function

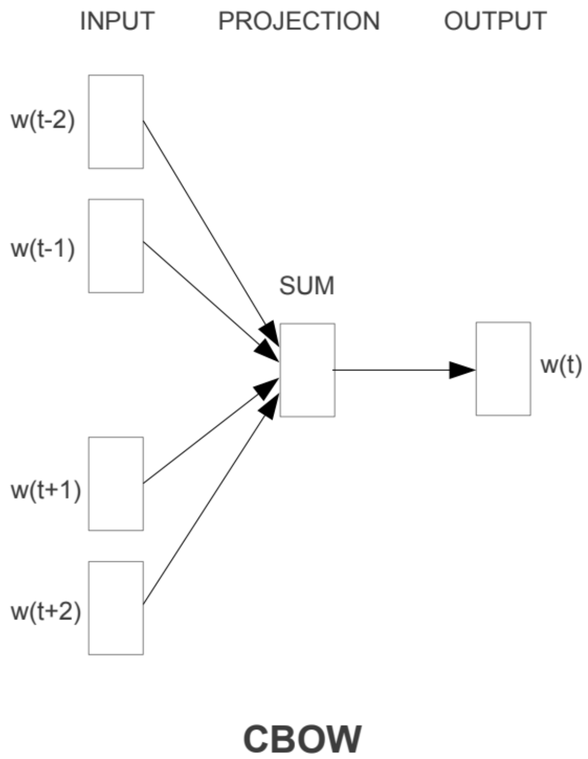
I enjoyed eating some pizza at the restaurant

Objective Function



Objective Function

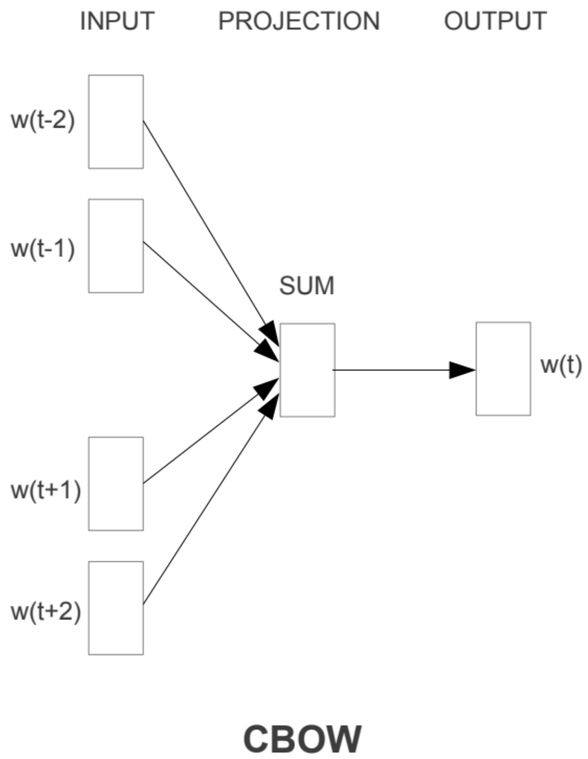
I enjoyed eating some	pizza	at the restaurant



CBOW

Objective Function

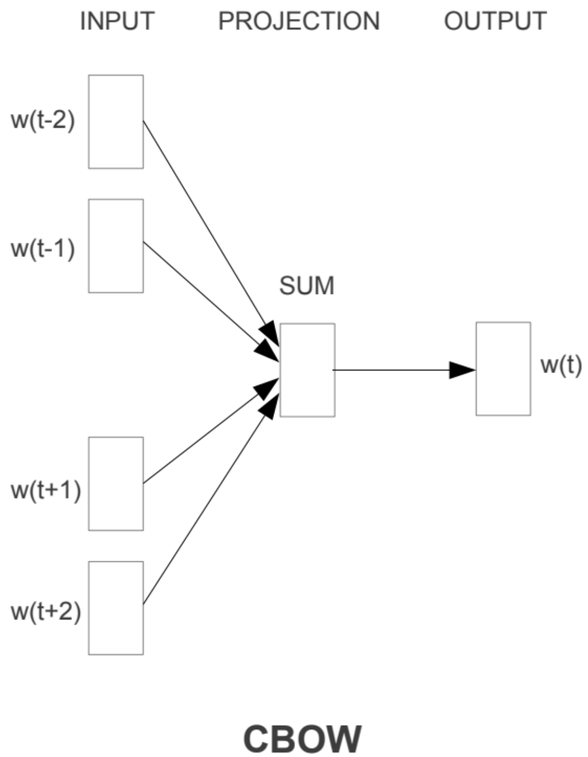
I enjoyed eating some	pizza	at the restaurant



maximise
the likelihood of a **word**
given its **context**

Objective Function

I enjoyed eating some	pizza	at the restaurant

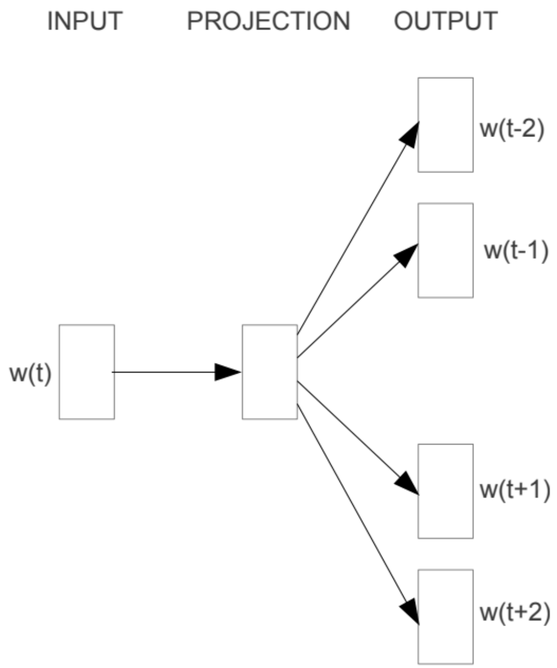


maximise
the likelihood of a word
given its context

e.g. P(**pizza** | **restaurant**)

Objective Function

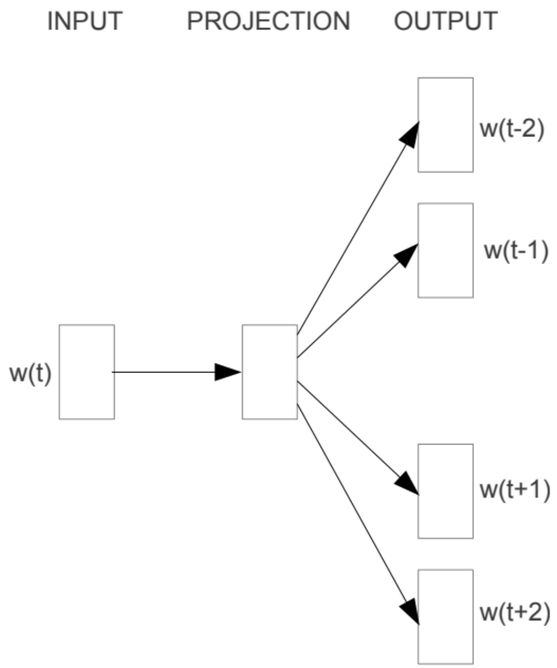
I enjoyed eating some	pizza	at the restaurant



Skip-gram

Objective Function

I enjoyed eating some	pizza	at the restaurant

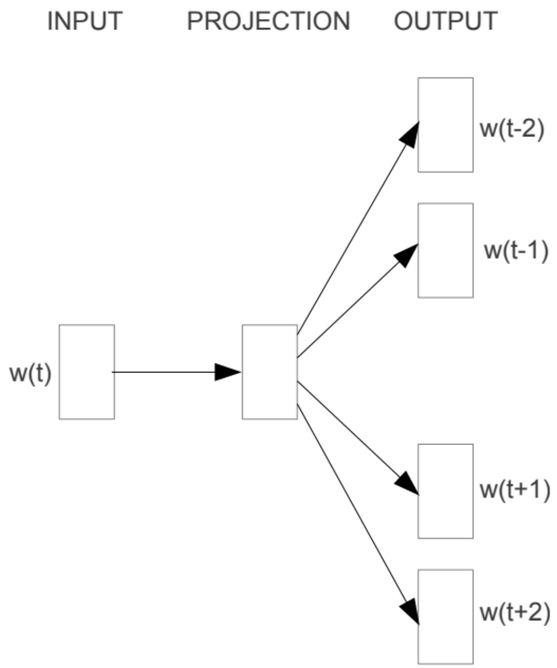


Skip-gram

maximise
the likelihood of the **context**
given the **focus word**

Objective Function

I enjoyed eating some **pizza** at the restaurant



Skip-gram

maximise
the likelihood of the context
given the focus word

e.g. P(**restaurant** | **pizza**)

WORD2VEC IN PYTHON

gensim – Topic Modelling in Python

Gensim is a Python library for *topic modelling*, *document indexing* and *similarity retrieval* with large corpora. Target audience is the *natural language processing* (NLP) and *information retrieval* (IR) community.

Features

- All algorithms are **memory-independent** w.r.t. the corpus size (can process input larger than RAM, streamed, out-of-core),
- **Intuitive interfaces**
  - easy to plug in your own input corpus/datastream (trivial streaming API)
  - easy to extend with other Vector Space algorithms (trivial transformation API)
- Efficient multicore implementations of popular algorithms, such as online **Latent Semantic Analysis (LSA/LSI/SVD)**, **Latent Dirichlet Allocation (LDA)**, **Random Projections (RP)**, **Hierarchical Dirichlet Process (HDP)** or **word2vec deep learning**.
- **Distributed computing**: can run *Latent Semantic Analysis* and *Latent Dirichlet Allocation* on a cluster of computers.
- Extensive documentation and Jupyter Notebook tutorials.

If this feature list left you scratching your head, you can first read more about the Vector Space Model and unsupervised document analysis on Wikipedia.

Support

gensim – Topic Modelling in Python

Gensim is a Python library for *topic modelling*, *document indexing* and *similarity retrieval* with large corpora. Target audience is the *natural language processing* (NLP) and *information retrieval* (IR) community.

Features

- All algorithms are **memory-independent** w.r.t. the corpus size (can process input larger than RAM, streamed, out-of-core),

pip install gensim

- Efficient multicore implementations of popular algorithms, such as online **Latent Semantic Analysis (LSA/LSI/SVD)**, **Latent Dirichlet Allocation (LDA)**, **Random Projections (RP)**, **Hierarchical Dirichlet Process (HDP)** or **word2vec deep learning**.
- **Distributed computing**: can run *Latent Semantic Analysis* and *Latent Dirichlet Allocation* on a cluster of computers.
- Extensive documentation and Jupyter Notebook tutorials.

If this feature list left you scratching your head, you can first read more about the Vector Space Model and unsupervised document analysis on Wikipedia.

Support

Example

Example

```
from gensim.models import Word2Vec
fname = ‘my_dataset.json’
corpus = MyCorpusReader(fname)
model = Word2Vec(corpus)
```

Example

```
from gensim.models import Word2Vec

fname = 'my_dataset.json'

corpus = MyCorpusReader(fname)

model = Word2Vec(corpus)
```

Example

```
model.most_similar('chef')

[('cook', 0.94),
 ('bartender', 0.91),
 ('waitress', 0.89),
 ('restaurant', 0.76),
 ...]
```

Example

```
model.most_similar('chef',
                   negative=['food'])
```

```
[('puppet', 0.93),
 ('devops', 0.92),
 ('ansible', 0.79),
 ('salt', 0.77),
 ...]
```

Pre-trained Vectors

Pre-trained Vectors

```
from gensim.models.keyedvectors \
      import KeyedVectors

fname = ‘GoogleNews-vectors.bin'

model = KeyedVectors.load_word2vec_format(
    fname,
    binary=True
)
```

Pre-trained Vectors

```
model.most_similar(
   positive=['king', ‘woman'],
   negative=[‘man’]
)
```

Pre-trained Vectors

```
model.most_similar(
   positive=['king', ‘woman'],
   negative=[‘man’]
)
```

```
    [('queen', 0.7118),
     ('monarch', 0.6189),
     ('princess', 0.5902),
     ('crown_prince', 0.5499),
     ('prince', 0.5377),
    …]
```

Pre-trained Vectors

```
model.most_similar(
   positive=['Paris', ‘Italy'],
   negative=[‘France’]
)
```

Pre-trained Vectors

```
model.most_similar(
   positive=['Paris', 'Italy'],
   negative=['France']
)
    [('Milan', 0.7222),
     ('Rome', 0.7028),
     ('Palermo_Sicily', 0.5967),
     ('Italian', 0.5911),
     ('Tuscany', 0.5632),
    …]
```

Pre-trained Vectors

```
model.most_similar(
   positive=[‘professor', ‘woman'],
   negative=[‘man’]
)
```

Pre-trained Vectors

```
model.most_similar(
   positive=['professor', 'woman'],
   negative=['man']
)
    [('associate_professor', 0.7771),
     ('assistant_professor', 0.7558),
     ('professor_emeritus', 0.7066),
     ('lecturer', 0.6982),
     ('sociology_professor', 0.6539),
    …]
```

Pre-trained Vectors

```
model.most_similar(
   positive=[‘professor', ‘man'],
   negative=[‘woman’]
)
```

Pre-trained Vectors

```
model.most_similar(
   positive=['professor', 'man'],
   negative=['woman']
)
    [('professor_emeritus', 0.7433),
     ('emeritus_professor', 0.7109),
     ('associate_professor', 0.6817),
     ('Professor', 0.6495),
     ('assistant_professor', 0.6484),
    …]
```

Pre-trained Vectors

```
model.most_similar(
   positive=[‘computer_programmer’, ‘woman'],
   negative=[‘man’]
)
```

Pre-trained Vectors

```
model.most_similar(
   positive=[‘computer_programmer’, ‘woman'],
   negative=[‘man’]
)
    [('homemaker', 0.5627),
     ('housewife', 0.5105),
     ('graphic_designer', 0.5051),
     ('schoolteacher', 0.4979),
     ('businesswoman', 0.4934),
    …]
```

Pre-trained Vectors

Culture is biased

Pre-trained Vectors

Culture is biased

Language is biased

Pre-trained Vectors

Culture is biased

Language is biased

Algorithms are not?

NOT ONLY WORD2VEC

GloVe (2014)

GloVe (2014)

- Global co-occurrence matrix

GloVe (2014)

- Global co-occurrence matrix
- Much bigger memory footprint

GloVe (2014)

- Global co-occurrence matrix
- Much bigger memory footprint
- Downstream tasks: similar performances

GloVe (2014)

- Global co-occurrence matrix
- Much bigger memory footprint
- Downstream tasks: similar performances
- Not in gensim (use spaCy)

doc2vec (2014)

doc2vec (2014)

- From words to documents

doc2vec (2014)

- From words to documents
- (or sentences, paragraphs, categories, …)

doc2vec (2014)

- From words to documents
- (or sentences, paragraphs, categories, …)
- P(word | context, label)

fastText (2016-17)

fastText (2016-17)

- word2vec + morphology (sub-words)

fastText (2016-17)

- word2vec + morphology (sub-words)
- Pre-trained vectors on ~300 languages (Wikipedia)

fastText (2016-17)

- word2vec + morphology (sub-words)
- Pre-trained vectors on ~300 languages (Wikipedia)
- rare words 👍

fastText (2016-17)

- word2vec + morphology (sub-words)
- Pre-trained vectors on ~300 languages (Wikipedia)
- rare words 👍
- out of vocabulary words 👍 (sometimes 👎)

fastText (2016-17)

- word2vec + morphology (sub-words)
- Pre-trained vectors on ~300 languages (Wikipedia)
- rare words 👍
- out of vocabulary words 👍 (sometimes 👎)
- morphologically rich languages 👍

FINAL REMARKS

But we’ve been doing this for X years

But we've been doing this for X years

- Approaches based on co-occurrences are not new
- … but usually outperformed by word embeddings
- … and don't scale as well as word embeddings

Garbage in, garbage out

Garbage in, garbage out

- Pre-trained vectors are useful … until they're not
- The business domain is important
- The pre-processing steps are important
- > 100K words? Maybe train your own model
- > 1M words? Yep, train your own model

Summary

Summary

- Word Embeddings are magic!
- Big victory of unsupervised learning
- Gensim makes your life easy

THANK YOU

@MarcoBonzanini

speakerdeck.com/marcobonzanini

GitHub.com/bonzanini

marcobonzanini.com

Credits & Readings

Credits & Readings

Credits

- Lev Konstantinovskiy (`@teagermylk`)

Readings

- Deep Learning for NLP (R. Socher) `http://cs224d.stanford.edu/`
- “GloVe: global vectors for word representation” by Pennington et al.
- “Distributed Representation of Sentences and Documents” (doc2vec) by Le and Mikolov
- “Enriching Word Vectors with Subword Information” (fastText) by Bojanokwsi et al.

Credits & Readings

Even More Readings

- “Man is to Computer Programmer as Woman is to Homemaker? Debiasing Word Embeddings” by Bolukbasi et al.
- “Quantifying and Reducing Stereotypes in Word Embeddings” by Bolukbasi et al.
- “Equality of Opportunity in Machine Learning” - Google Research Blog

  `https://research.googleblog.com/2016/10/equality-of-opportunity-in-machine.html`

Pics Credits

- Classification: `https://commons.wikimedia.org/wiki/File:Cluster-2.svg`
- Translation: `https://commons.wikimedia.org/wiki/File:Translation_-_A_till_%C3%85-colours.svg`
- Welsh cake: `https://commons.wikimedia.org/wiki/File:Closeup_of_Welsh_cakes,_February_2009.jpg`
- Pizza: `https://commons.wikimedia.org/wiki/File:Eq_it-na_pizza-margherita_sep2005_sml.jpg`