Word Embeddings for NLP in Python

Marco Bonzanini

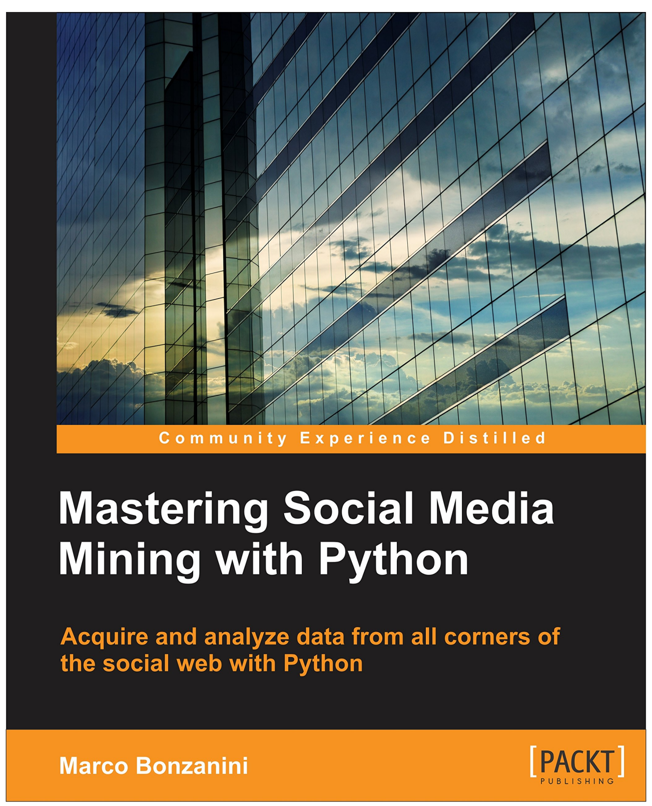
London Python Meet-up
September 2017

Nice to meet you

PyData London

NUMFOCUS
OPEN CODE = BETTER SCIENCE

PYCON UK



WORD EMBEDDINGS?

Word Embeddings

=

Word Vectors

=

Distributed Representations

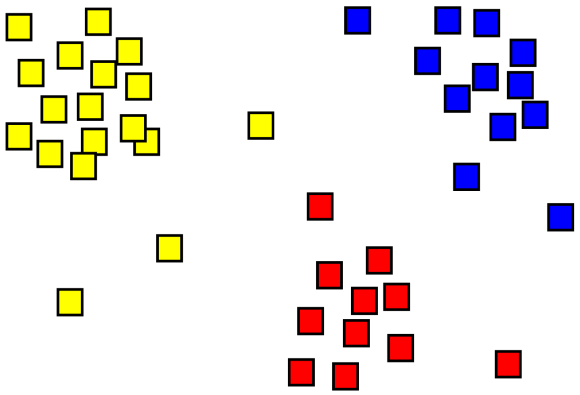
Why should you care?

Why should you care?

Data representation is crucial

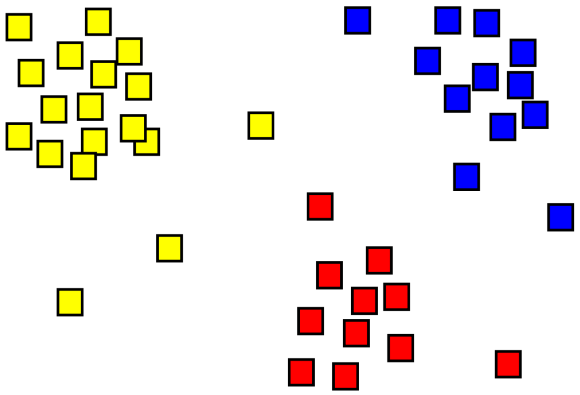
Applications

Applications



Classification

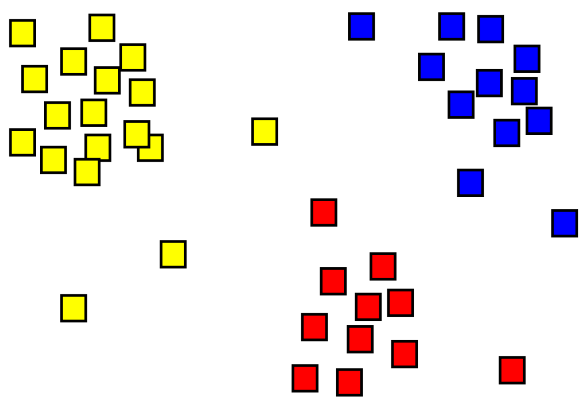
Applications



Classification

Recommender Systems

Applications

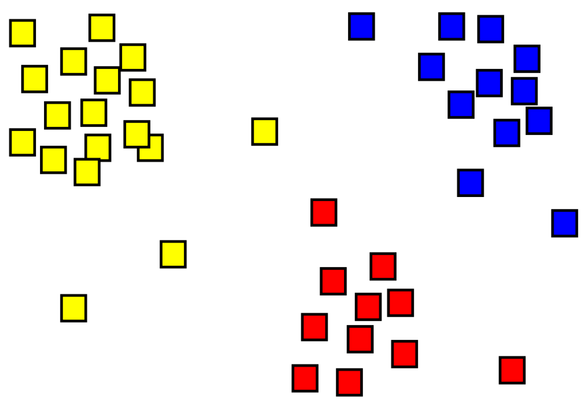


Classification

Recommender Systems

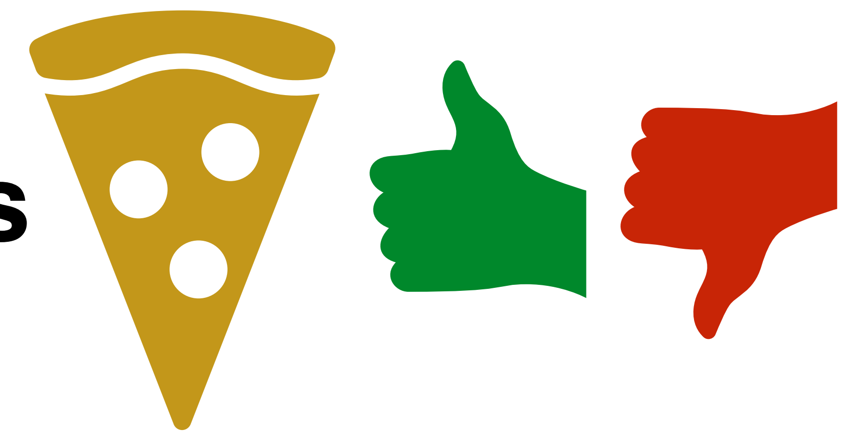
Search Engines

Applications



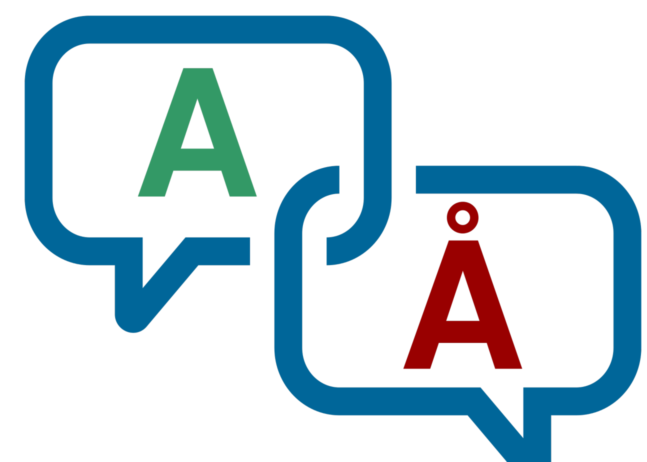
Classification

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

Paris

Rome   word V

```
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

$$\overbrace{\hphantom{[1, 0, 0, 0, 0, 0, \ldots, 0]}}^{\textbf{V = vocabulary size (huge)}}$$

```
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]
```

Bag-of-words

Bag-of-words

```
doc_1 = [32, 14, 1,  0, …,  6]
doc_2 = [ 2, 12, 0, 28, …, 12]
…        …
doc_N = [13,  0, 6,  2, …, 0]
```

Bag-of-words

```
          Rome    Paris                      word V
doc_1 = [32, 14, 1,   0, …,   6]
doc_2 = [ 2, 12, 0, 28, …, 12]

…         …

doc_N = [13,  0, 6,   2, …,  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

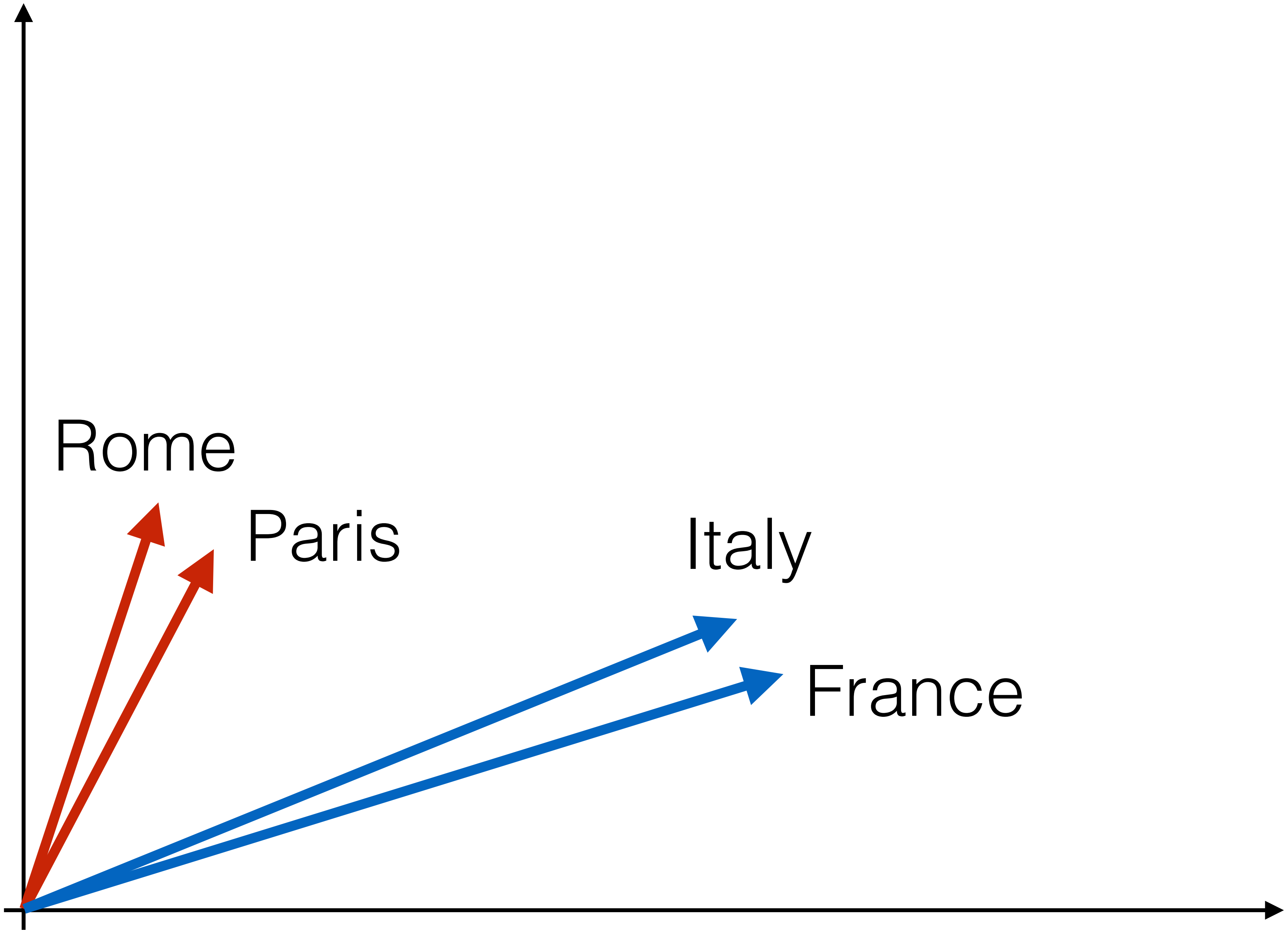
Rome = [0.91, 0.83, 0.17, …, 0.41]

Paris = [0.92, 0.82, 0.17, …, 0.98]

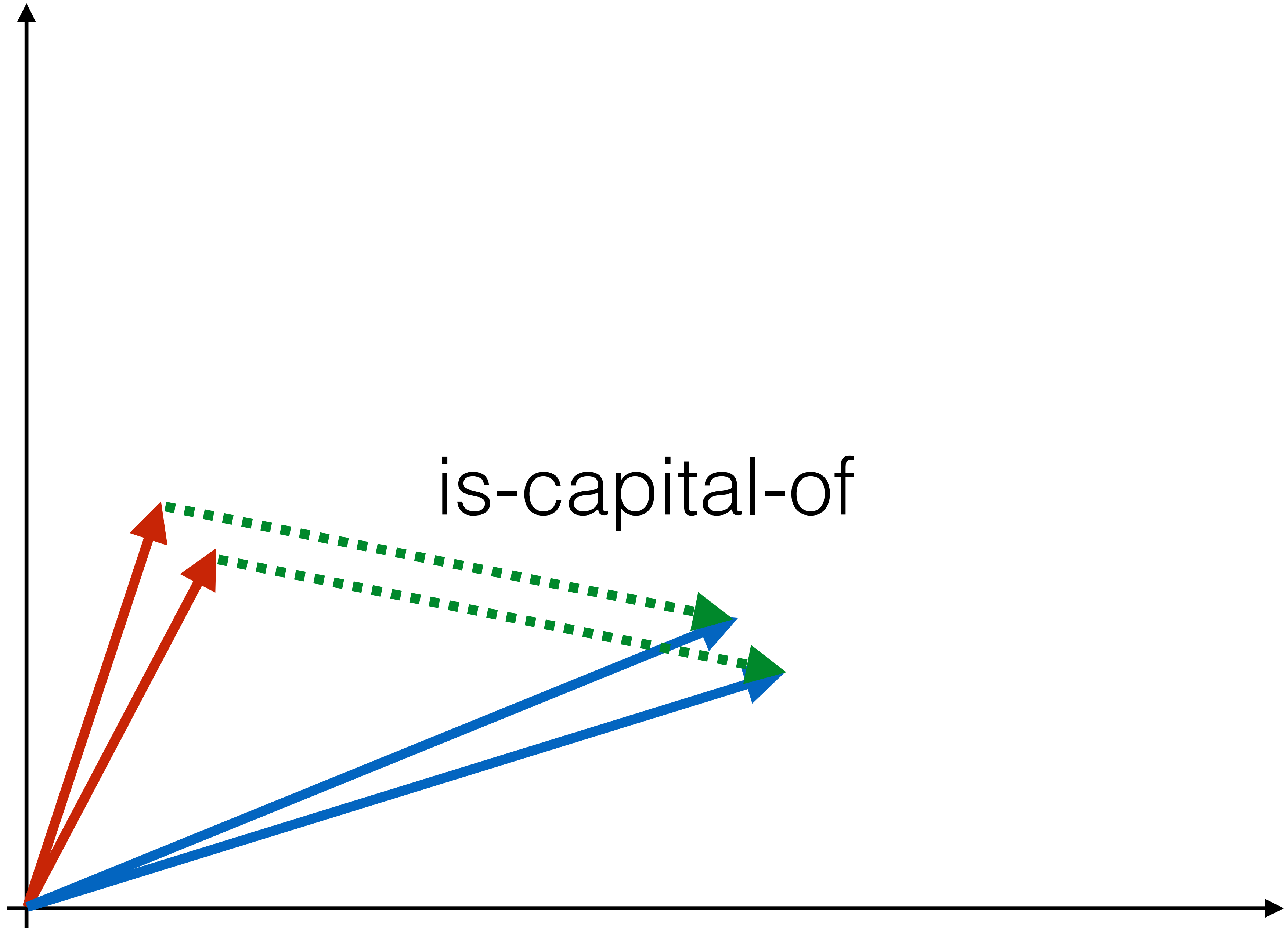
Italy = [0.32, 0.77, 0.67, …, 0.42]

France = [0.33, 0.78, 0.66, …, 0.97]

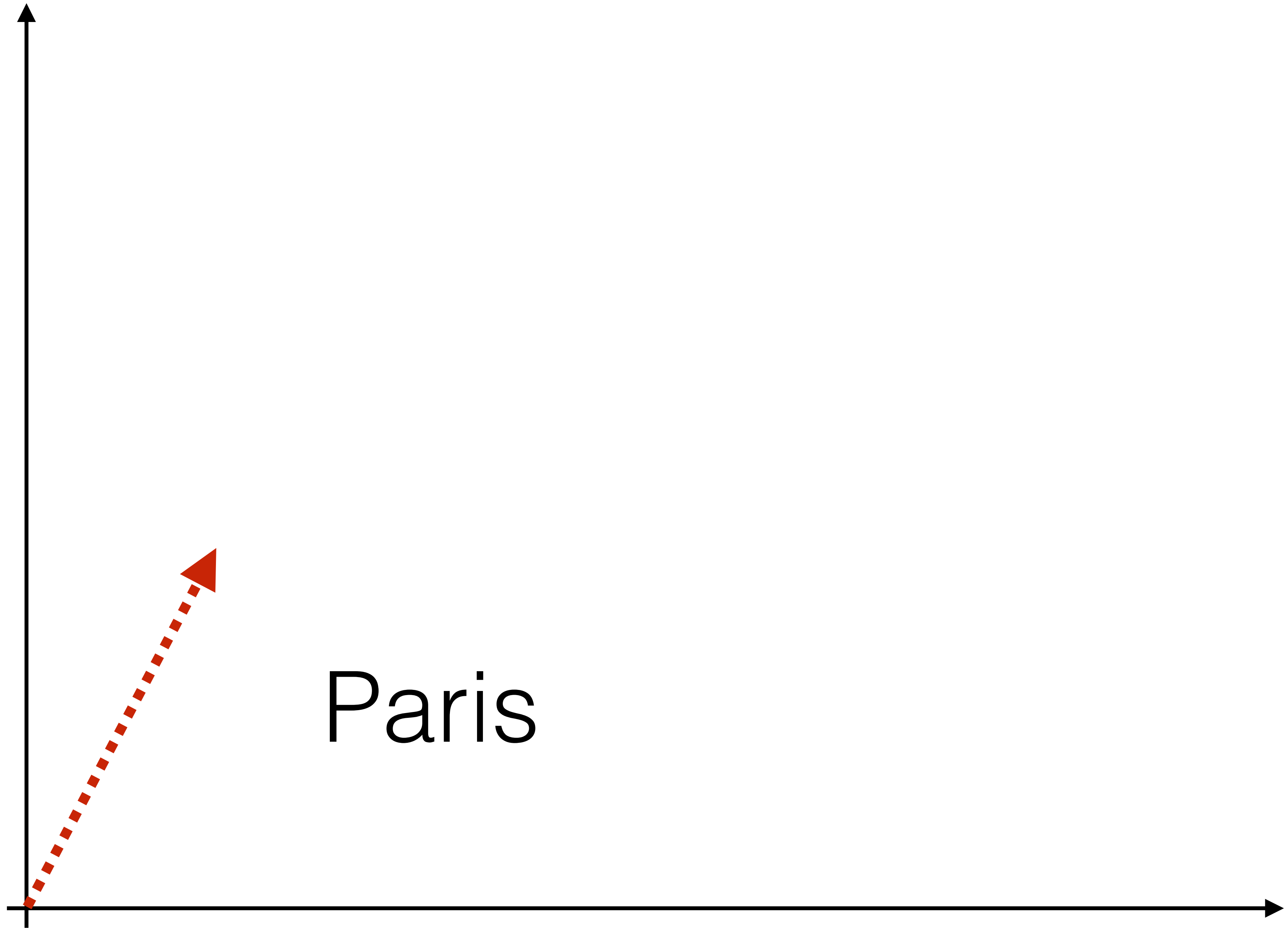
Word Embeddings



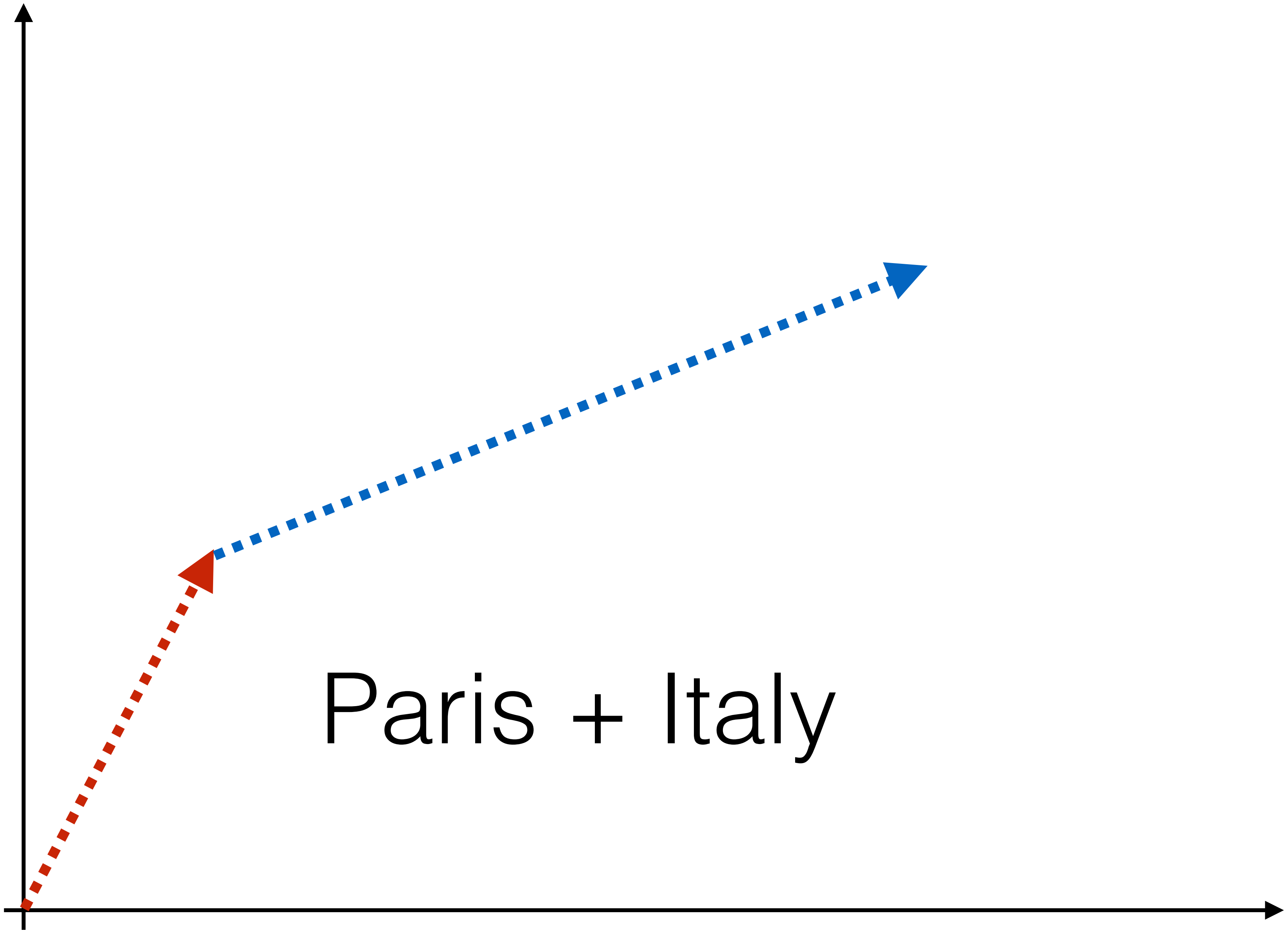
Word Embeddings



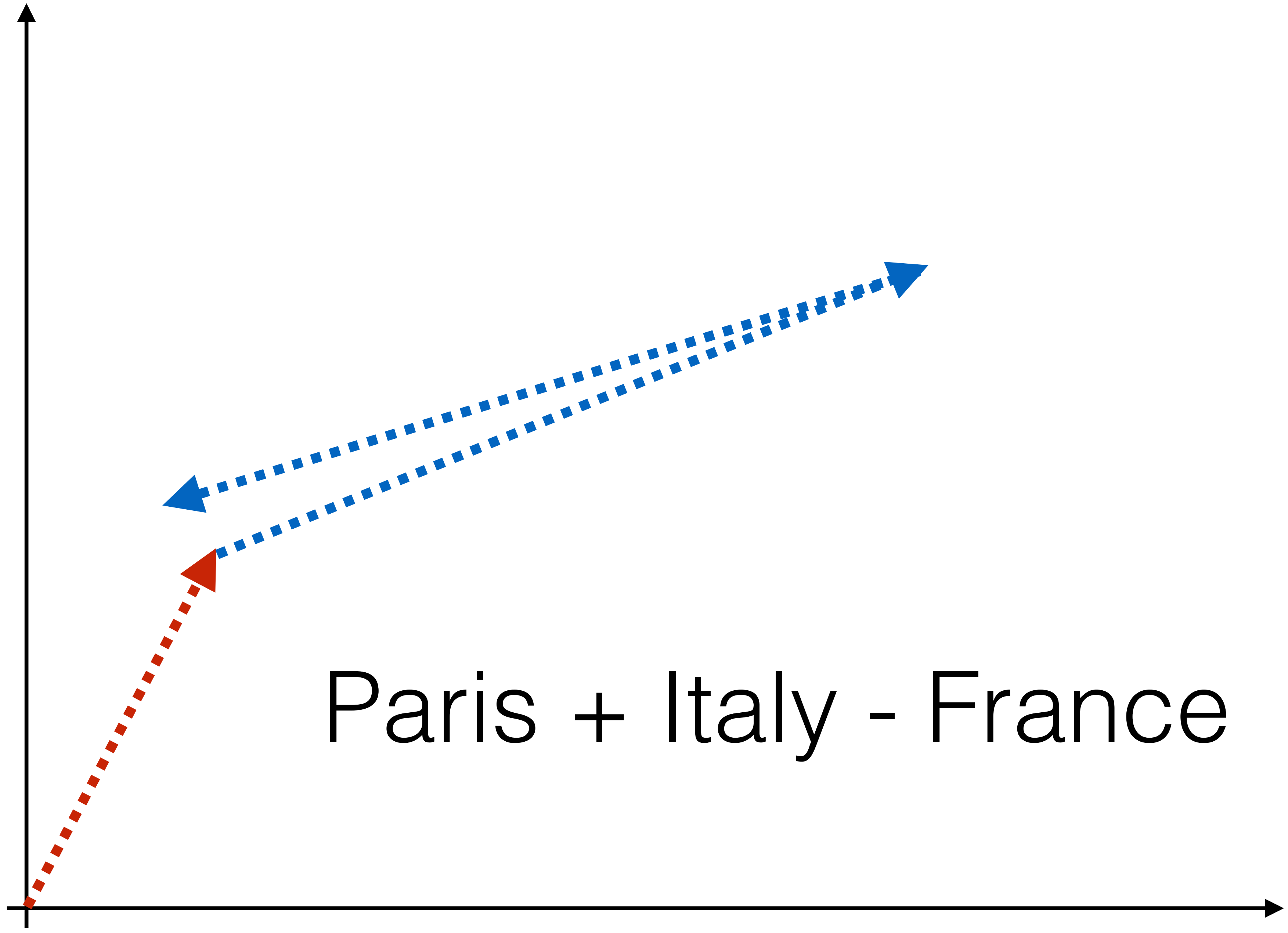
Word Embeddings



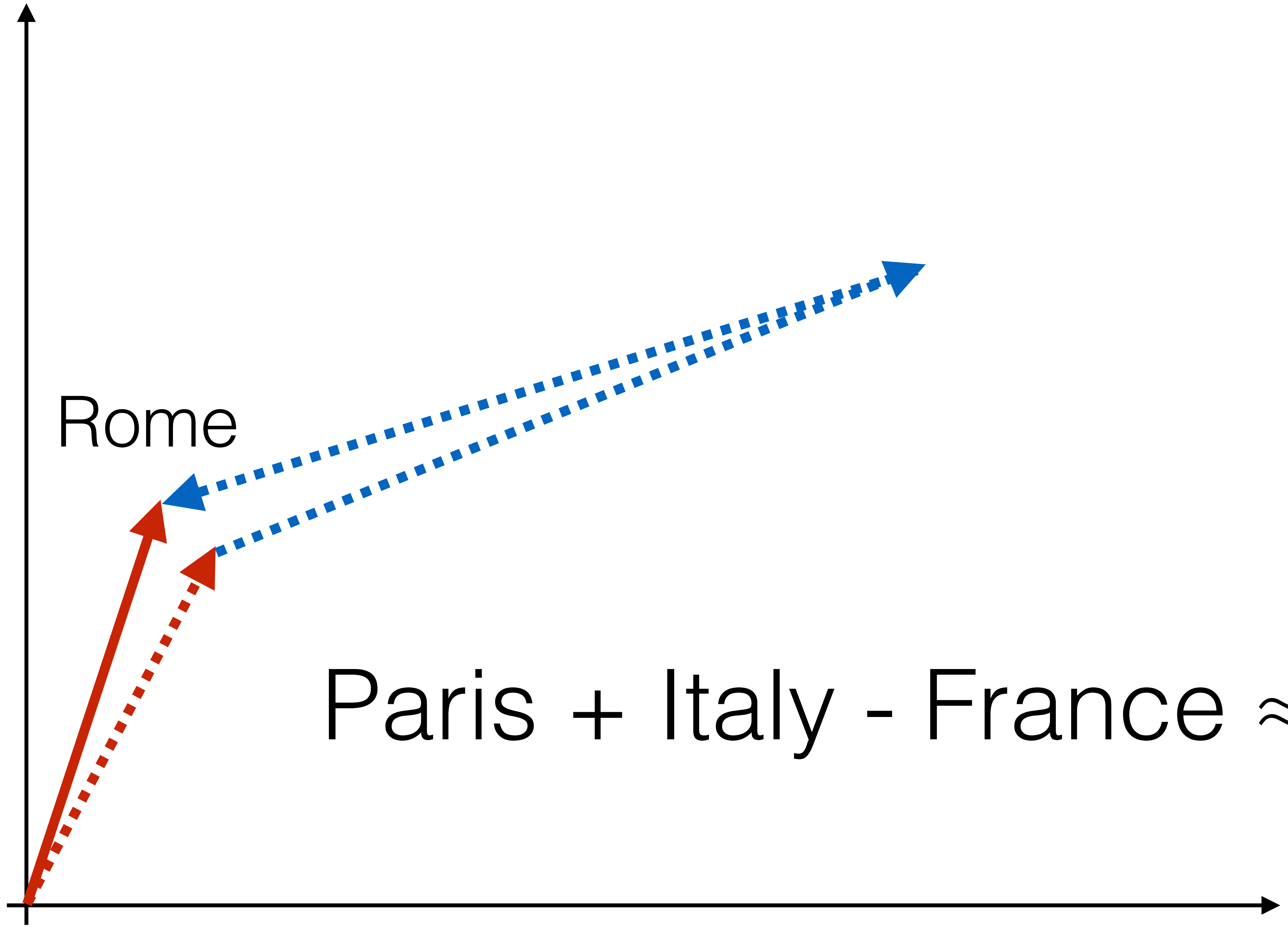
Word Embeddings



Word Embeddings



Word Embeddings



Paris + Italy - France ≈ Rome

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 $\boxed{\text{pizza}}$ at the restaurant

Word

I enjoyed eating some pizza at the restaurant

The company it keeps

I enjoyed eating some pizza at the restaurant

I enjoyed eating some pineapple at the restaurant

I enjoyed eating some pizza at the restaurant

I enjoyed eating some pineapple at the restaurant

I enjoyed eating some $\boxed{\text{pizza}}$ at the restaurant

I enjoyed eating some $\boxed{\text{pineapple}}$ at the restaurant

Same context

I enjoyed eating some pizza at the restaurant

I enjoyed eating some pineapple at the restaurant

Same context

Pizza = Pineapple ?

A BIT OF THEORY
word2vec

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

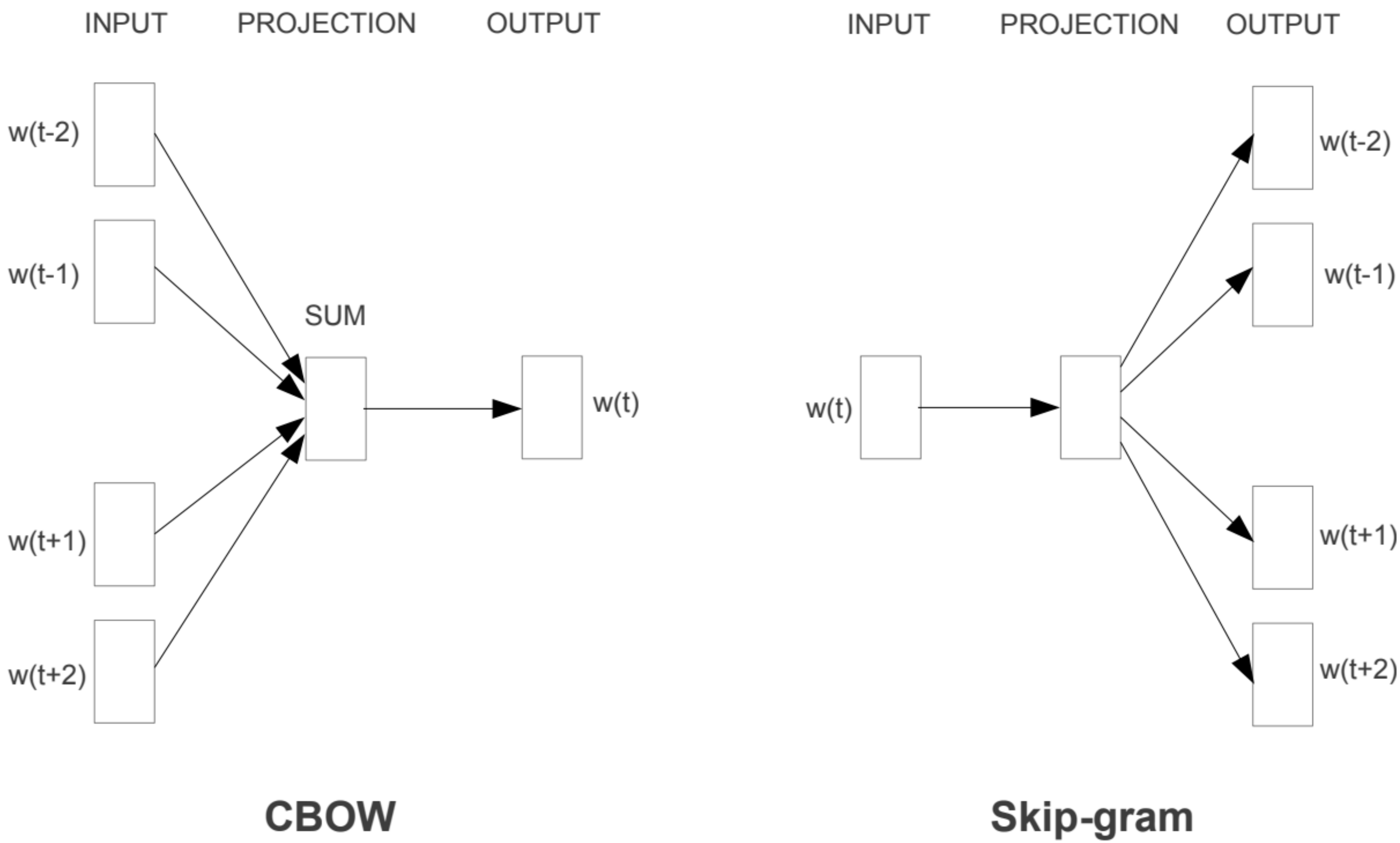
Greg Corrado
Google Inc.
Mountain View
gcorrado@google.com

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

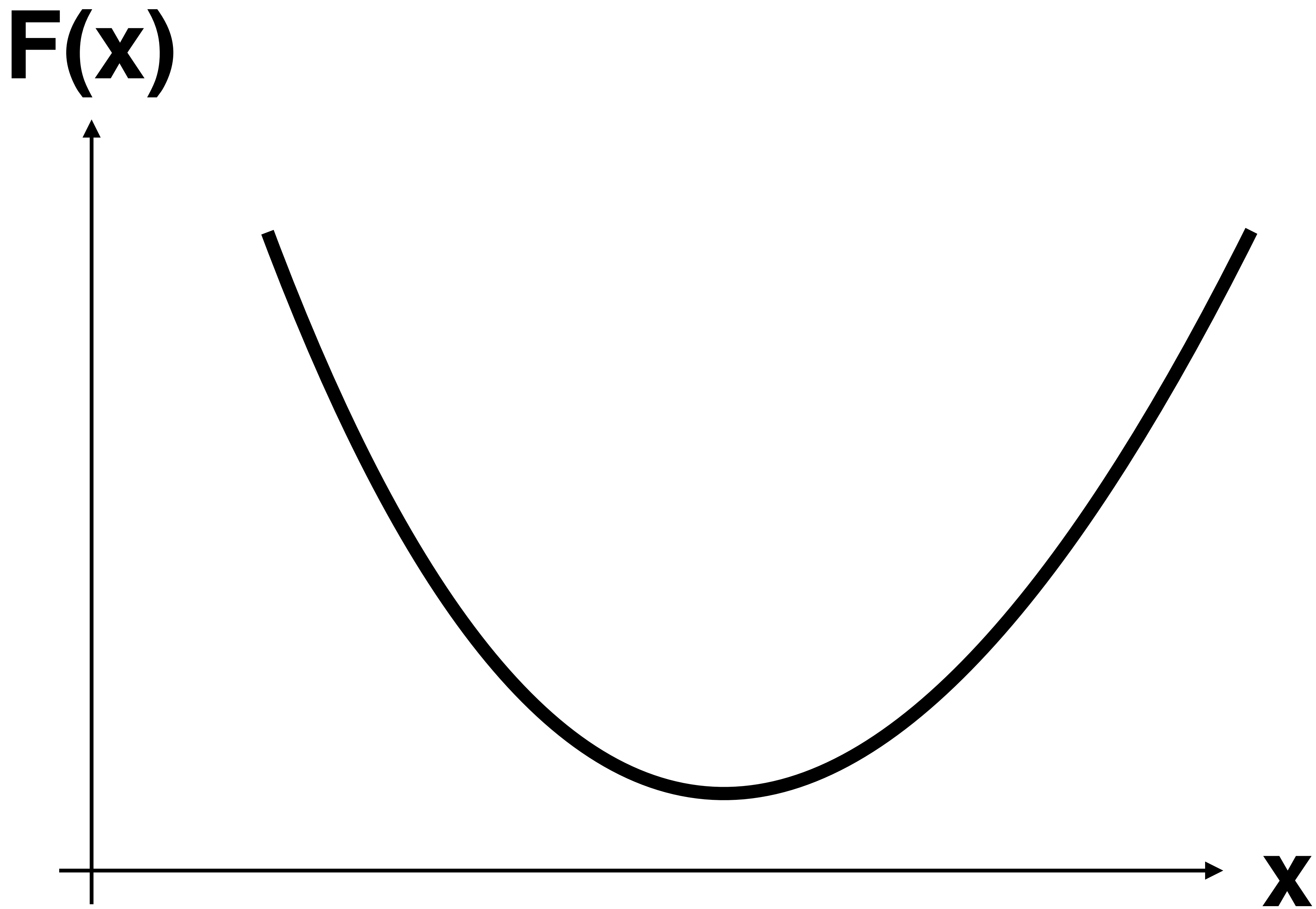
Vector Calculation

Goal: learn vec(word)

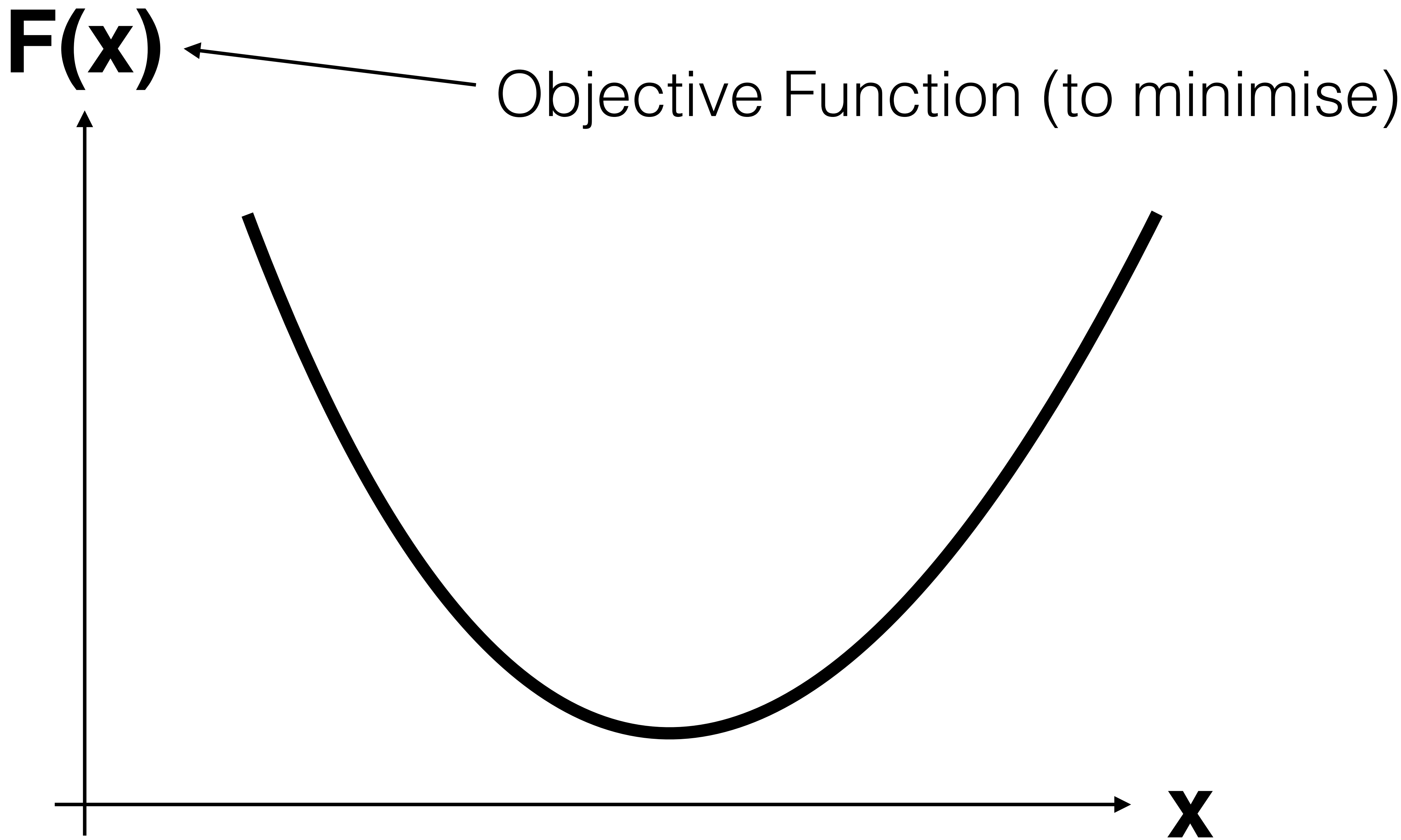
1. Choose objective function
2. Init: random vectors
3. Run stochastic gradient descent

Intermezzo (Gradient Descent)

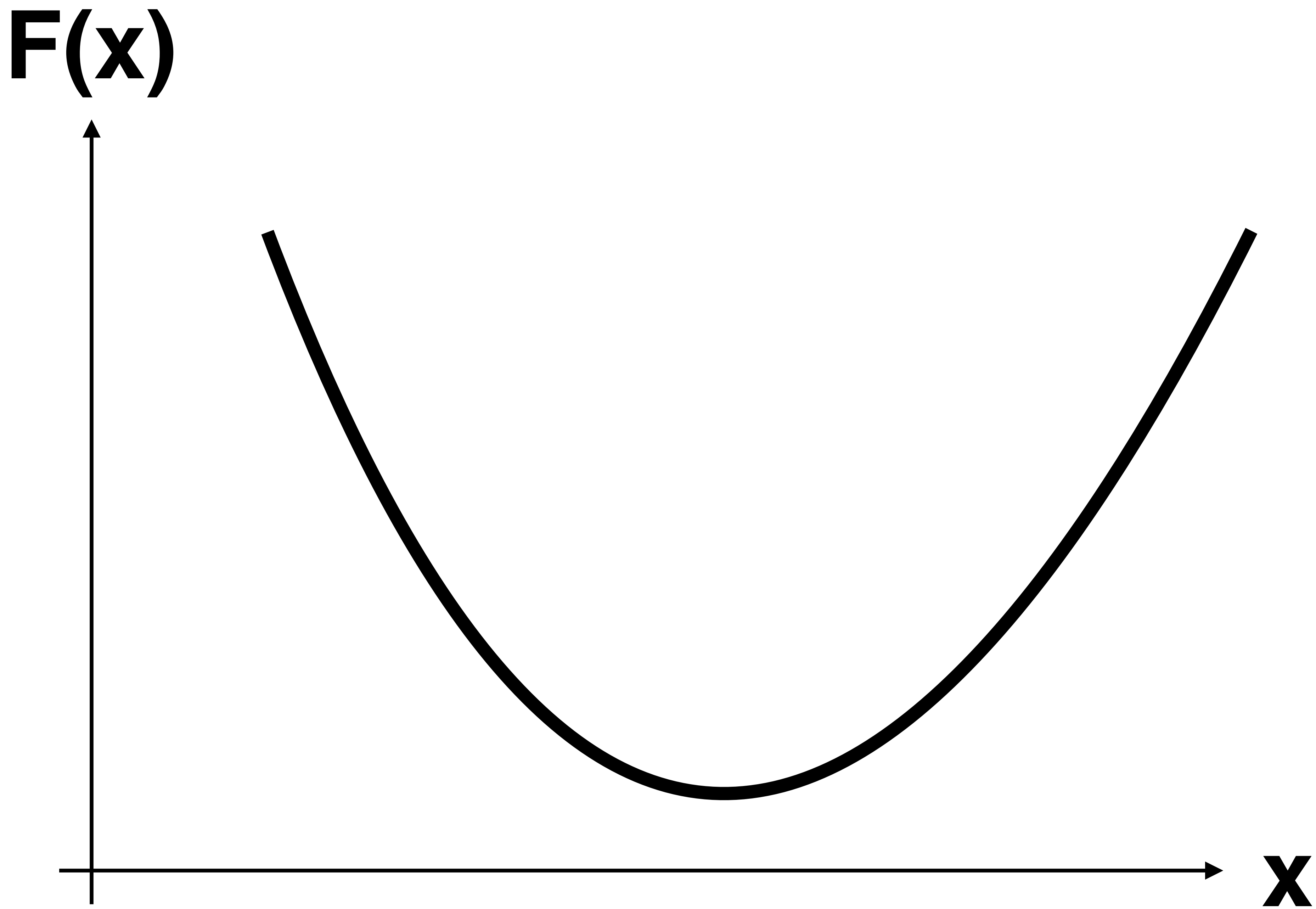
Intermezzo (Gradient Descent)



Intermezzo (Gradient Descent)

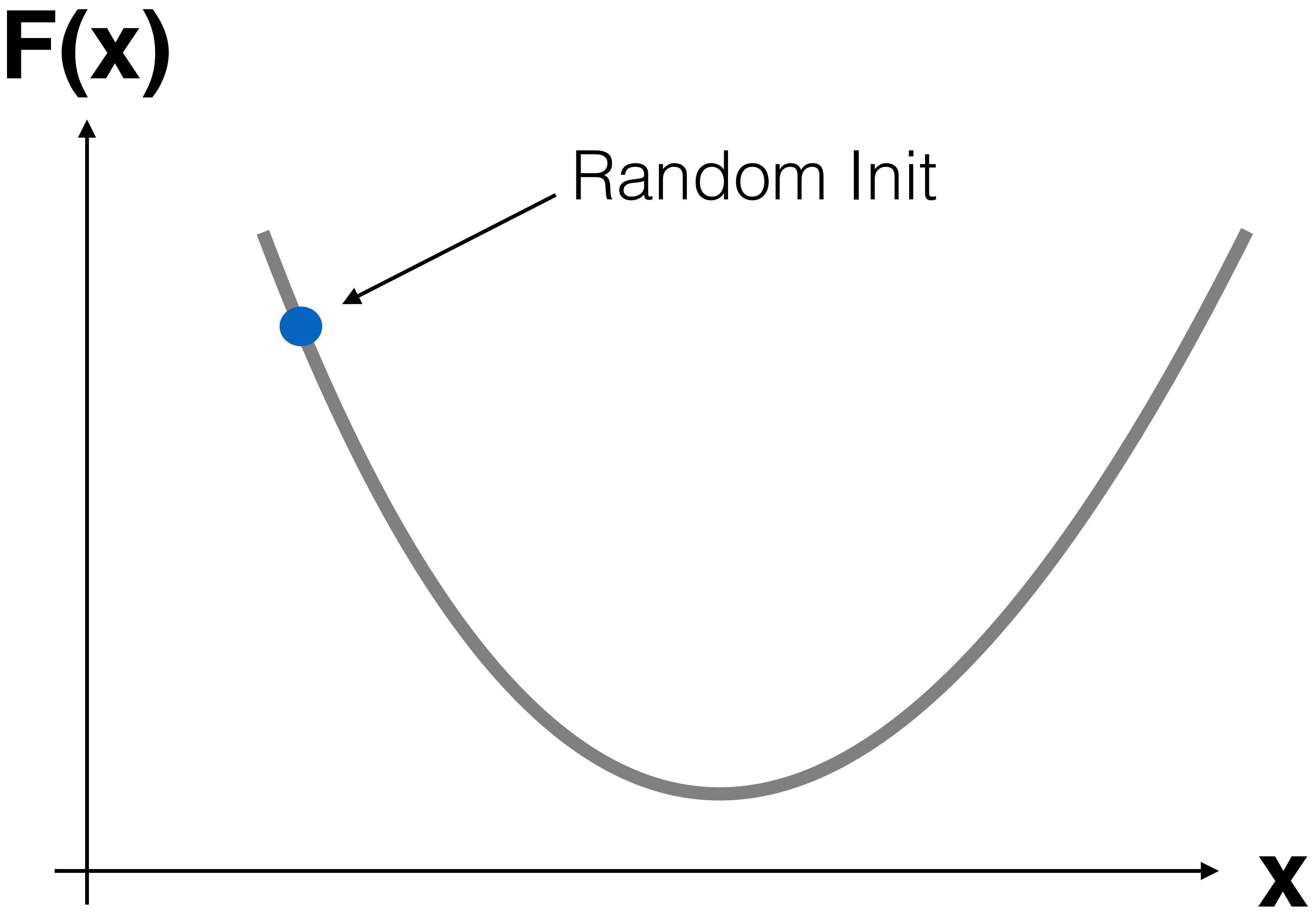


Intermezzo (Gradient Descent)

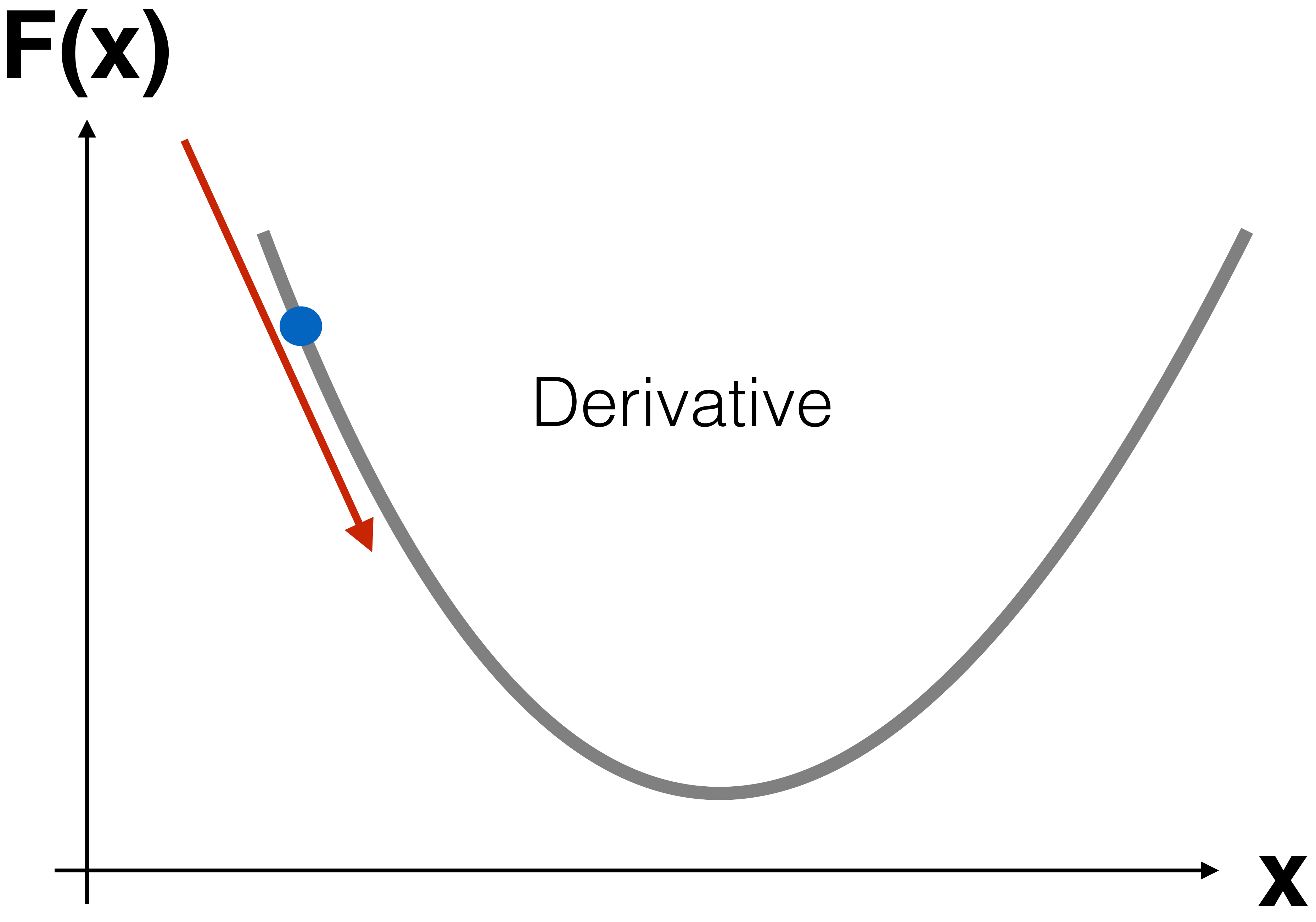


Find the optimal “x”

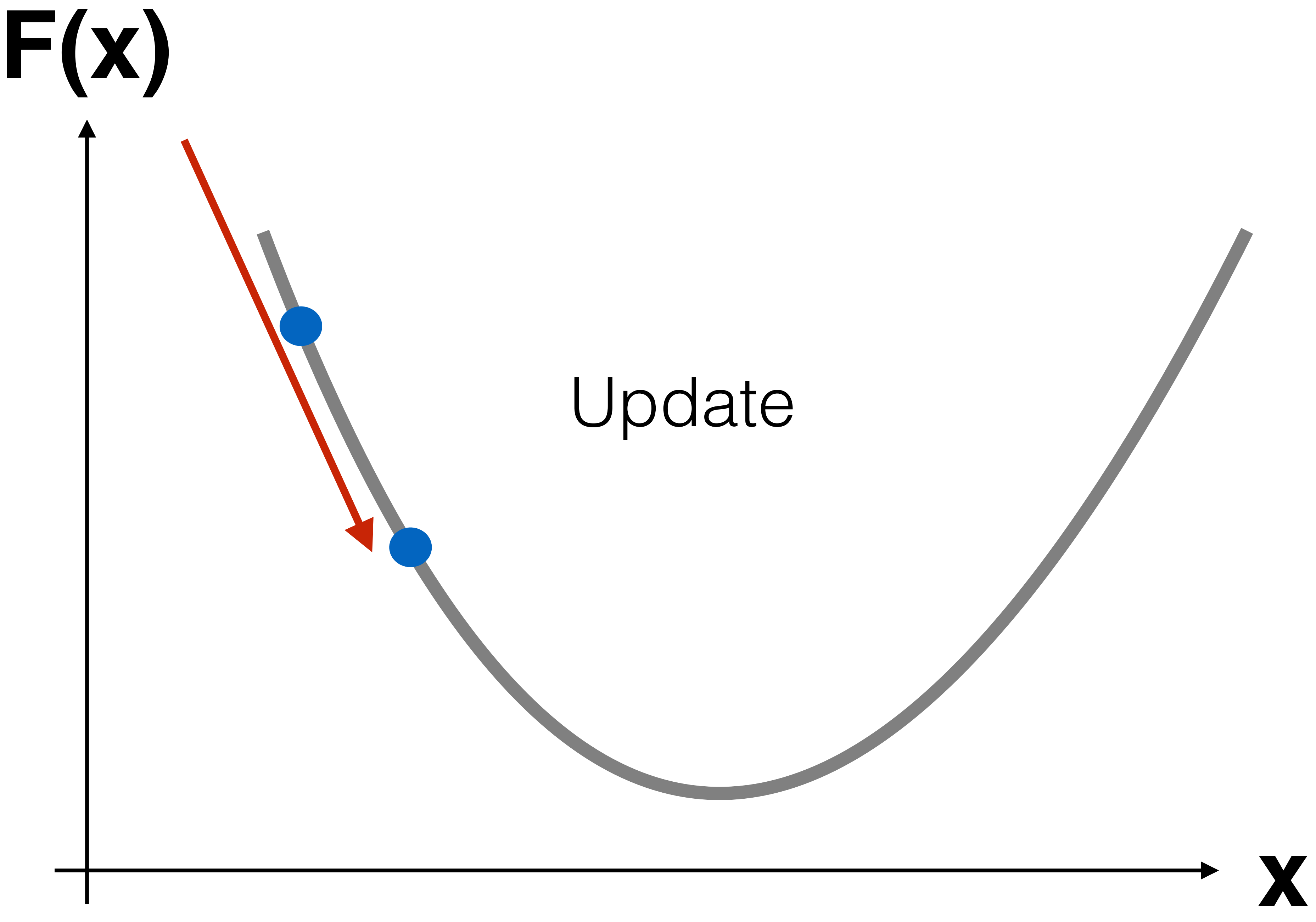
Intermezzo (Gradient Descent)



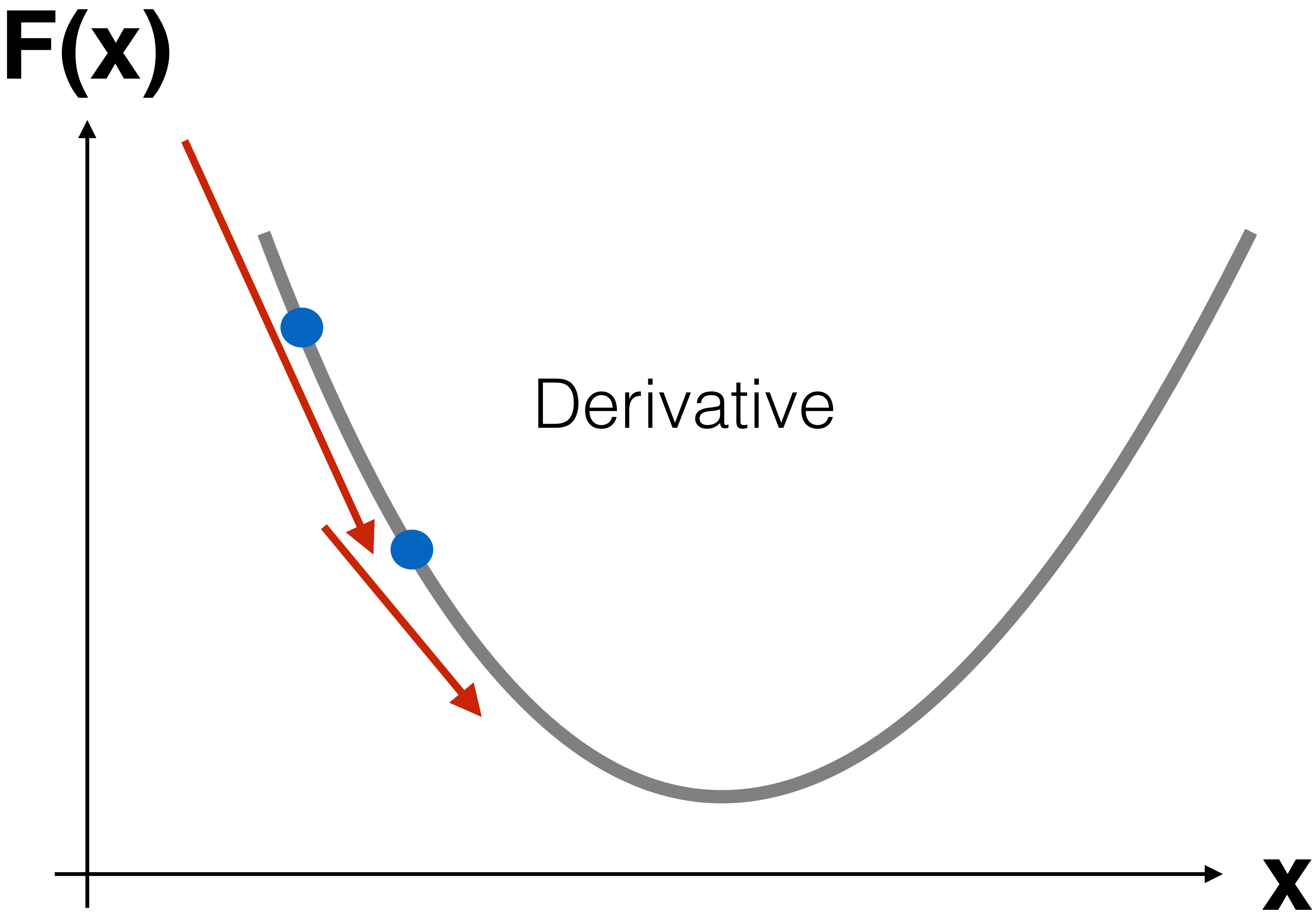
Intermezzo (Gradient Descent)



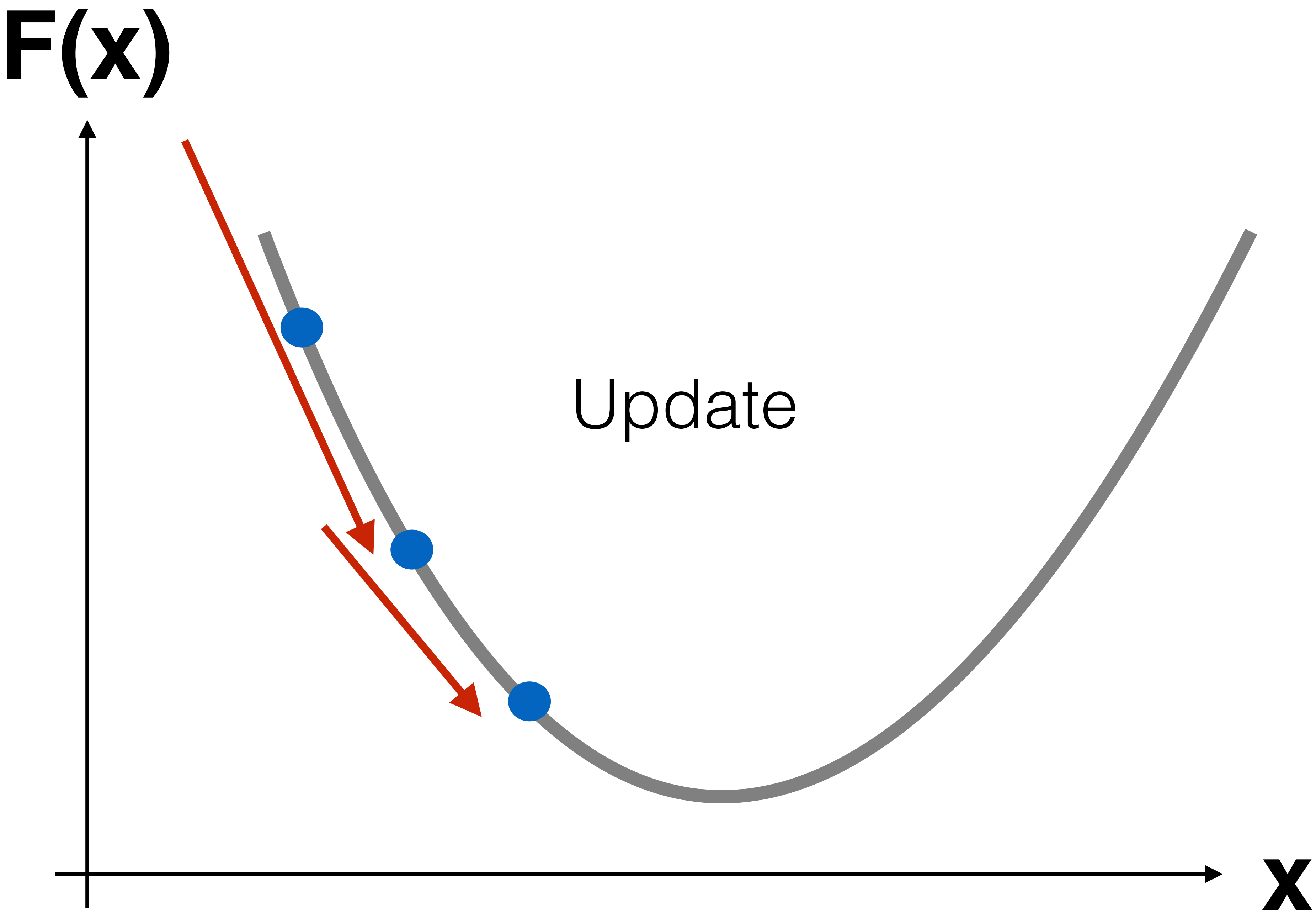
Intermezzo (Gradient Descent)



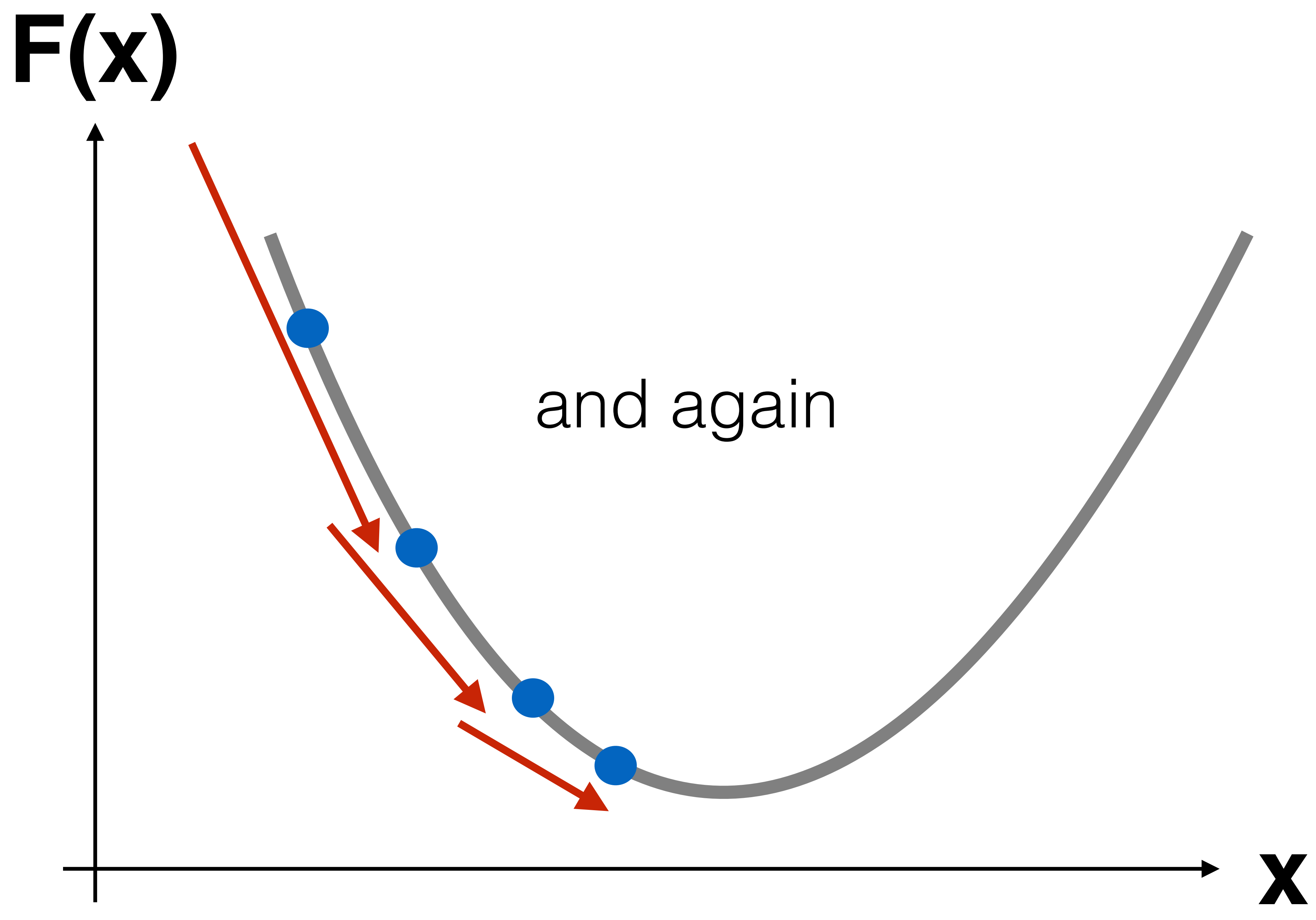
Intermezzo (Gradient Descent)



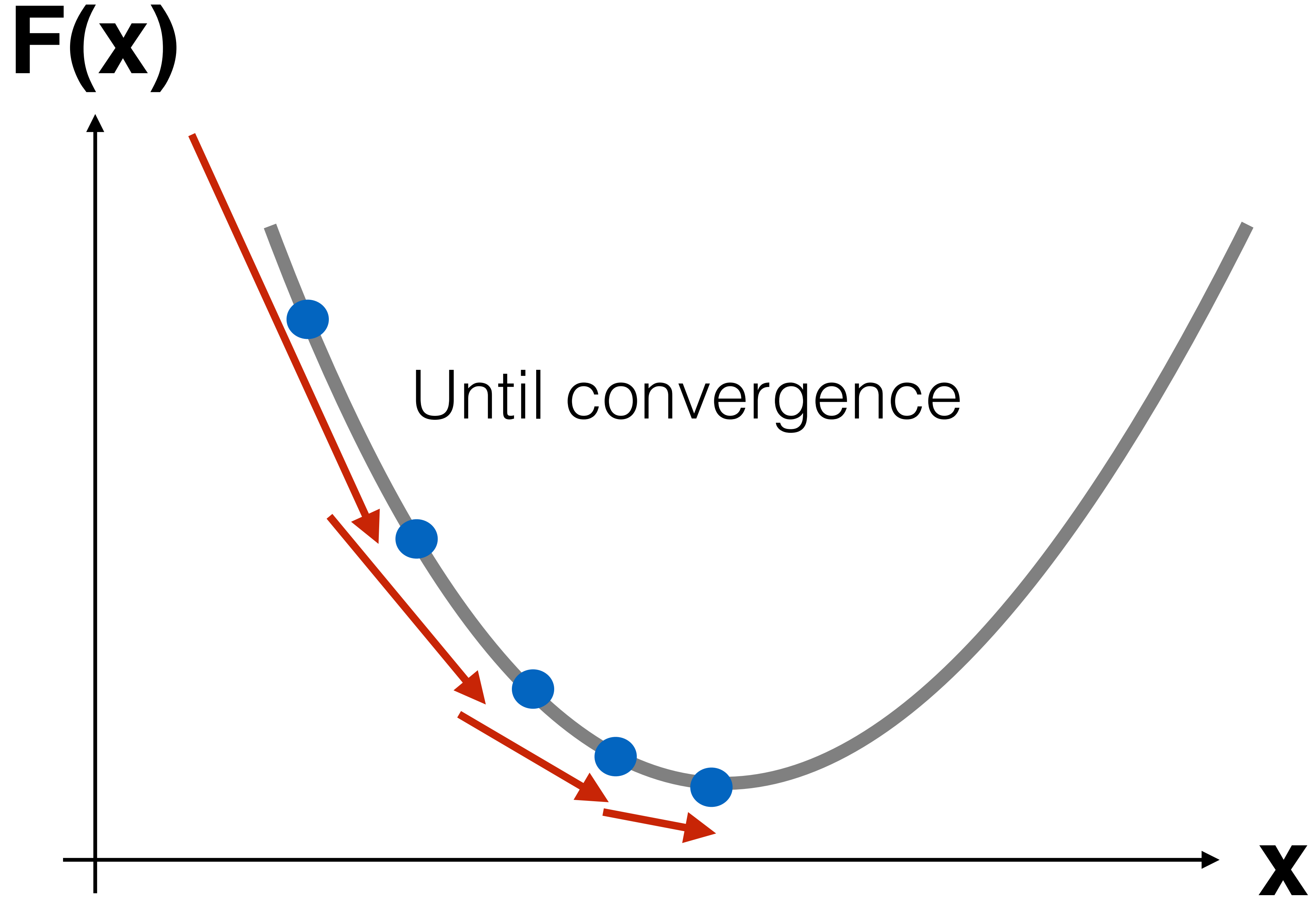
Intermezzo (Gradient Descent)



Intermezzo (Gradient Descent)



Intermezzo (Gradient Descent)



Intermezzo (Gradient Descent)

- Optimisation algorithm

Intermezzo (Gradient Descent)

- Optimisation algorithm
- Purpose: find the min (or max) for F

Intermezzo (Gradient Descent)

- Optimisation algorithm
- Purpose: find the min (or max) for F
- Batch-oriented (use all data points)

Intermezzo (Gradient Descent)

- Optimisation algorithm
- Purpose: find the min (or max) for F
- Batch-oriented (use all data points)
- Stochastic GD: update after each sample

Objective Function

Objective Function

I enjoyed eating some pizza at the restaurant

Objective Function

I enjoyed eating some $\boxed{\text{pizza}}$ at the restaurant

Objective Function

I enjoyed eating some pizza at the restaurant

Objective Function

I enjoyed eating some pizza at the restaurant

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

Objective Function

I enjoyed eating some | pizza | at the restaurant

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

P(i | pizza)
P(enjoyed | pizza)
…
P(restaurant | pizza)

Example

I enjoyed eating some pizza at the restaurant

Example

I enjoyed eating some pizza at the restaurant

Iterate over context words

Example

I enjoyed eating some pizza at the restaurant

bump P(i | pizza)

Example

I $\boxed{\text{enjoyed}}$ eating some $\boxed{\text{pizza}}$ at the restaurant

bump P(enjoyed | pizza)

Example

I enjoyed $\boxed{\text{eating}}$ some $\boxed{\text{pizza}}$ at the restaurant

bump P(eating | pizza)

Example

I enjoyed eating $\boxed{\text{some}}$ $\boxed{\text{pizza}}$ at the restaurant

bump P(some | pizza)

Example

I enjoyed eating some $\boxed{\text{pizza}}$ $\boxed{\text{at}}$ the restaurant

bump P(at | pizza)

Example

I enjoyed eating some pizza at the restaurant

bump P(the | pizza)

Example

I enjoyed eating some $\boxed{\text{pizza}}$ at the $\boxed{\text{restaurant}}$

bump P(restaurant | pizza)

Example

I enjoyed eating some pizza $\boxed{\text{at}}$ the restaurant

Move to next focus word and repeat

Example

I enjoyed eating some pizza at the restaurant

bump P(i | at)

Example

I enjoyed eating some pizza at the restaurant

bump P(enjoyed | at)

Example

I enjoyed eating some pizza $\boxed{\text{at}}$ the restaurant

… you get the picture

$$P(\text{ eating } \mid \text{ pizza })$$

$P(\text{ eating } | \text{ pizza })$ **??**

Output word **Input word**

$$P(\boxed{\text{eating}} \mid \boxed{\text{pizza}})$$

Output word **Input word**

$$P(\boxed{\text{eating}} \mid \boxed{\text{pizza}})$$

$$P(\text{vec(eating)} \mid \text{vec(pizza)})$$

Output word **Input word**

$$P(\boxed{\text{eating}} \mid \boxed{\text{pizza}})$$

$$P(\, \text{vec(eating)} \mid \text{vec(pizza)} \,)$$

$$P(\, v_{out} \mid v_{in} \,)$$

Output word **Input word**

$$P(\boxed{\text{eating}} \mid \boxed{\text{pizza}})$$

$$P(\ \text{vec(eating)} \mid \text{vec(pizza)}\ )$$

$$P(\ v_{out} \mid v_{in}\ )$$ **???**

$$P(v_{out} \mid v_{in})$$

$$\text{cosine}(\ v_{out},\ v_{in}\ )$$

$$\text{cosine}(\, v_{out},\ v_{in}\,) \quad \mathbf{[-1, 1]}$$

$$\mathrm{softmax}(\mathrm{cosine}(\ v_{out},\ v_{in}\ ))$$

$$\text{softmax}(\text{cosine}(\ v_{out},\ v_{in}\ ))$$ **[0, 1]**

softmax(cosine(v_{out}, v_{in}))

$$P(\mathrm{v}_{out}|\mathrm{v}_{in}) = \frac{\exp(\mathrm{cosine}(\mathrm{v}_{out}, \mathrm{v}_{in}))}{\sum_{k \in V} \exp(\mathrm{cosine}(\mathrm{v}_k, \mathrm{v}_{in}))}$$

Vector Calculation Recap

Vector Calculation Recap

Learn vec(word)

Vector Calculation Recap

Learn vec(word)

by gradient descent

Vector Calculation Recap

Learn vec(word)

by gradient descent

on the softmax probability

Plot Twist

Distributed Representations of Sentences and Documents

Quoc Le QVL@GOOGLE.COM
Tomas Mikolov TMIKOLOV@GOOGLE.COM
Google Inc, 1600 Amphitheatre Parkway, Mountain View, CA 94043

Abstract

Many machine learning algorithms require the input to be represented as a fixed-length feature vector. When it comes to texts, one of the most common fixed-length features is bag-of-words. Despite their popularity, bag-of-words features have two major weaknesses: they lose the ordering of the words and they also ignore semantics of the words. For example, "powerful," "strong" and "Paris" are equally distant. In this paper, we propose *Paragraph Vector*, an unsupervised algorithm that learns fixed-length feature representations from variable-length pieces of texts, such as sentences, paragraphs, and documents. Our algo-

tages. The word order is lost, and thus different sentences can have exactly the same representation, as long as the same words are used. Even though bag-of-n-grams considers the word order in short context, it suffers from data sparsity and high dimensionality. Bag-of-words and bag-of-n-grams have very little sense about the semantics of the words or more formally the distances between the words. This means that words "powerful," "strong" and "Paris" are equally distant despite the fact that semantically, "powerful" should be closer to "strong" than "Paris."

In this paper, we propose *Paragraph Vector*, an unsupervised framework that learns continuous distributed vector representations for pieces of texts. The texts can be of variable-length, ranging from sentences to documents. The

Distributed Representations of Sentences and Documents

Quoc Le QVL@GOOGLE.COM
Tomas Mikolov TMIKOLOV@GOOGLE.COM
Google Inc, 1600 Amphitheatre Parkway, Mountain View, CA 94043

Abstract

Many machine learning algorithms require the input to be represented as a fixed-length feature vector. When it comes to texts, one of the most common fixed-length features is bag-of-words. Despite their popularity, bag-of-words features have two major weaknesses: they lose the ordering of the words and they also ignore semantics of the words. For example, "powerful," "strong" and "Paris" are equally distant. In this paper, we propose *Paragraph Vector*, an unsupervised algorithm that learns fixed-length feature representations from variable-length pieces of texts, such as sentences, paragraphs, and documents. Our algo-

tages. The word order is lost, and thus different sentences can have exactly the same representation, as long as the same words are used. Even though bag-of-n-grams considers the word order in short context, it suffers from data sparsity and high dimensionality. Bag-of-words and bag-of-n-grams have very little sense about the semantics of the words or more formally the distances between the words. This means that words "powerful," "strong" and "Paris" are equally distant despite the fact that semantically, "powerful" should be closer to "strong" than "Paris."

In this paper, we propose *Paragraph Vector*, an unsupervised framework that learns continuous distributed vector representations for pieces of texts. The texts can be of variable-length, ranging from sentences to documents. The

Paragraph Vector

a.k.a.

doc2vec

i.e.

$$\mathbf{P(v_{out} \mid v_{in}, \underline{label})}$$

A BIT OF PRACTICE

gensim – Topic Modelling in Python

build passing release v1.0.1 wheel yes Mailing List gitter join chat → Follow 2k

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

build passing release v1.0.1 wheel yes Mailing List gitter join chat → Follow 2k

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

Case Study 1: Skills and CVs

Case Study 1: Skills and CVs

Data set of ~300k resumes

Each experience is a “sentence”

Each experience has 3-15 skills

Approx 15k unique skills

Case Study 1: Skills and CVs

```
from gensim.models import Word2Vec
fname = 'candidates.jsonl'
corpus = ResumesCorpus(fname)
model = Word2Vec(corpus)
```

Case Study 1: Skills and CVs

```
model.most_similar('chef')

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

Case Study 1: Skills and CVs

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

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

Case Study 1: Skills and CVs

Useful for:

Data exploration

Query expansion/suggestion

Recommendations

Case Study 2: Beer!

Case Study 2: Beer!

Data set of ~2.9M beer reviews

89 different beer styles

635k unique tokens

185M total tokens

Case Study 2: Beer!

```
from gensim.models import Doc2Vec
fname = 'ratebeer_data.csv'
corpus = RateBeerCorpus(fname)
model = Doc2Vec(corpus)
```

Case Study 2: Beer!

```
from gensim.models import Doc2Vec
fname = 'ratebeer_data.csv'
corpus = RateBeerCorpus(fname)
model = Doc2Vec(corpus)
```

3.5h on my laptop

… remember to pickle

Case Study 2: Beer!

```
model.docvecs.most_similar('Stout')
[('Sweet Stout', 0.9877),
 ('Porter', 0.9620),
 ('Foreign Stout', 0.9595),
 ('Dry Stout', 0.9561),
 ('Imperial/Strong Porter', 0.9028),
 ...]
```

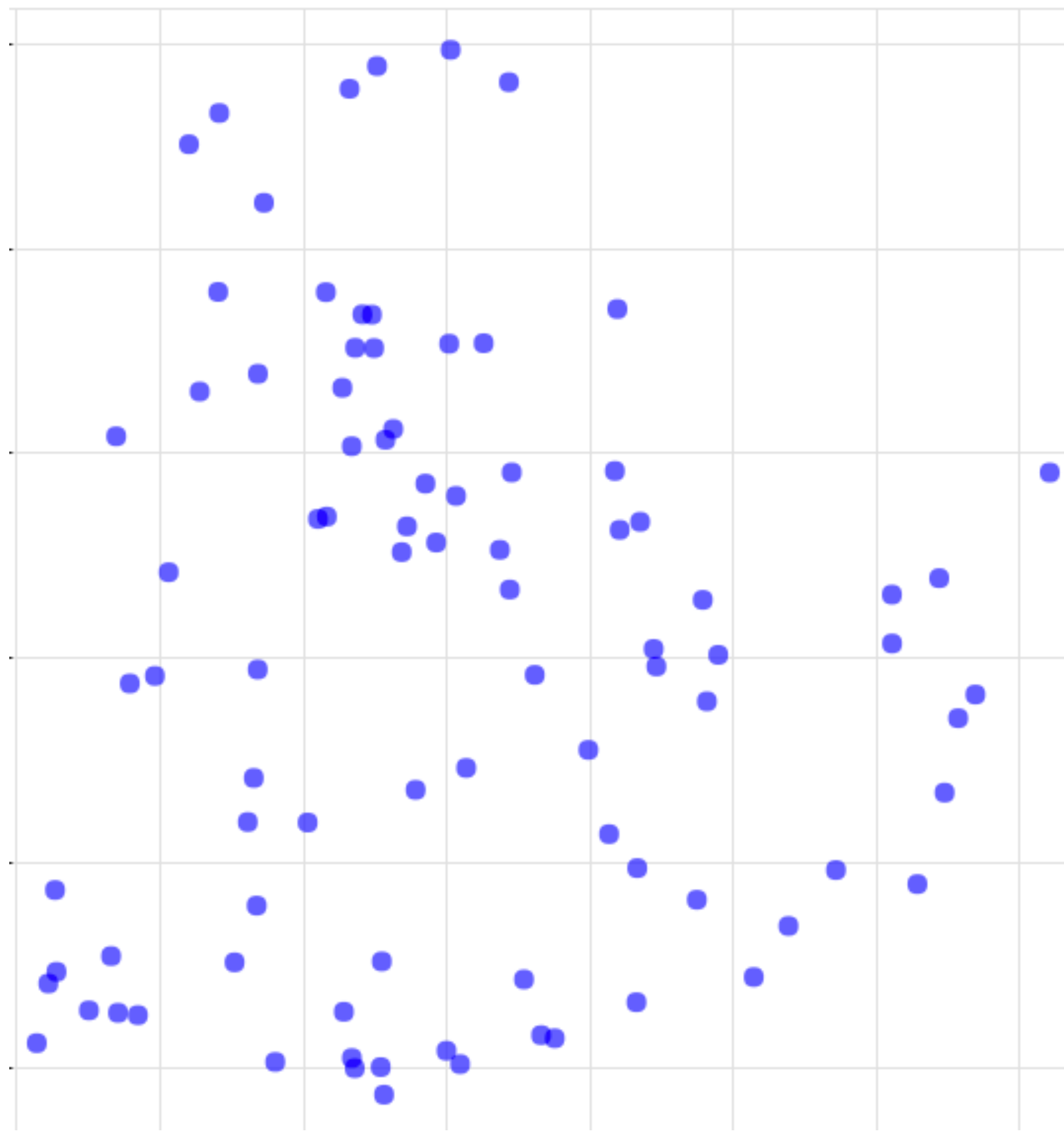
Case Study 2: Beer!

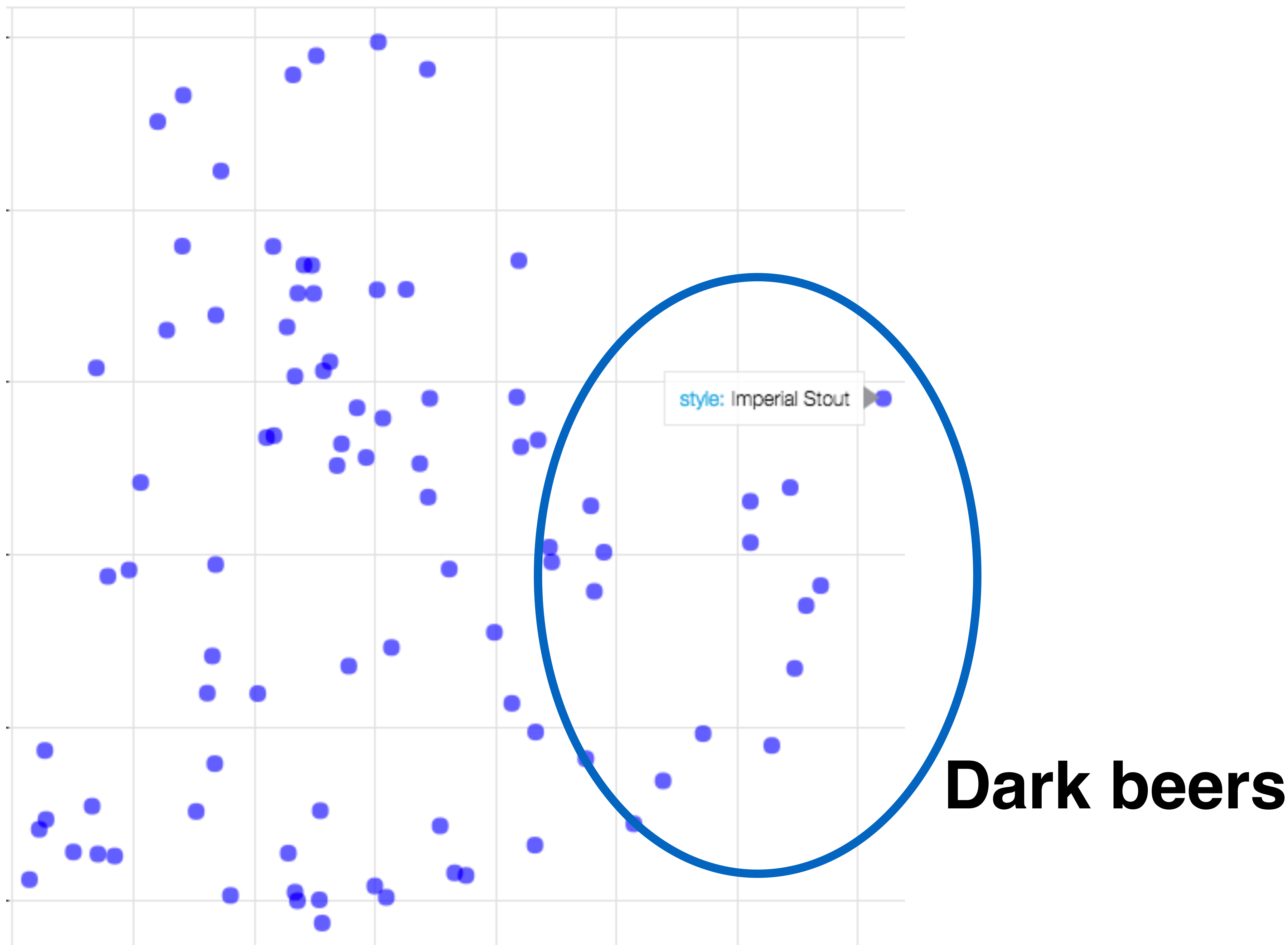
```
model.most_similar([model.docvecs['Stout']])

[('coffee', 0.6342),
 ('espresso', 0.5931),
 ('charcoal', 0.5904),
 ('char', 0.5631),
 ('bean', 0.5624),
 ...]
```

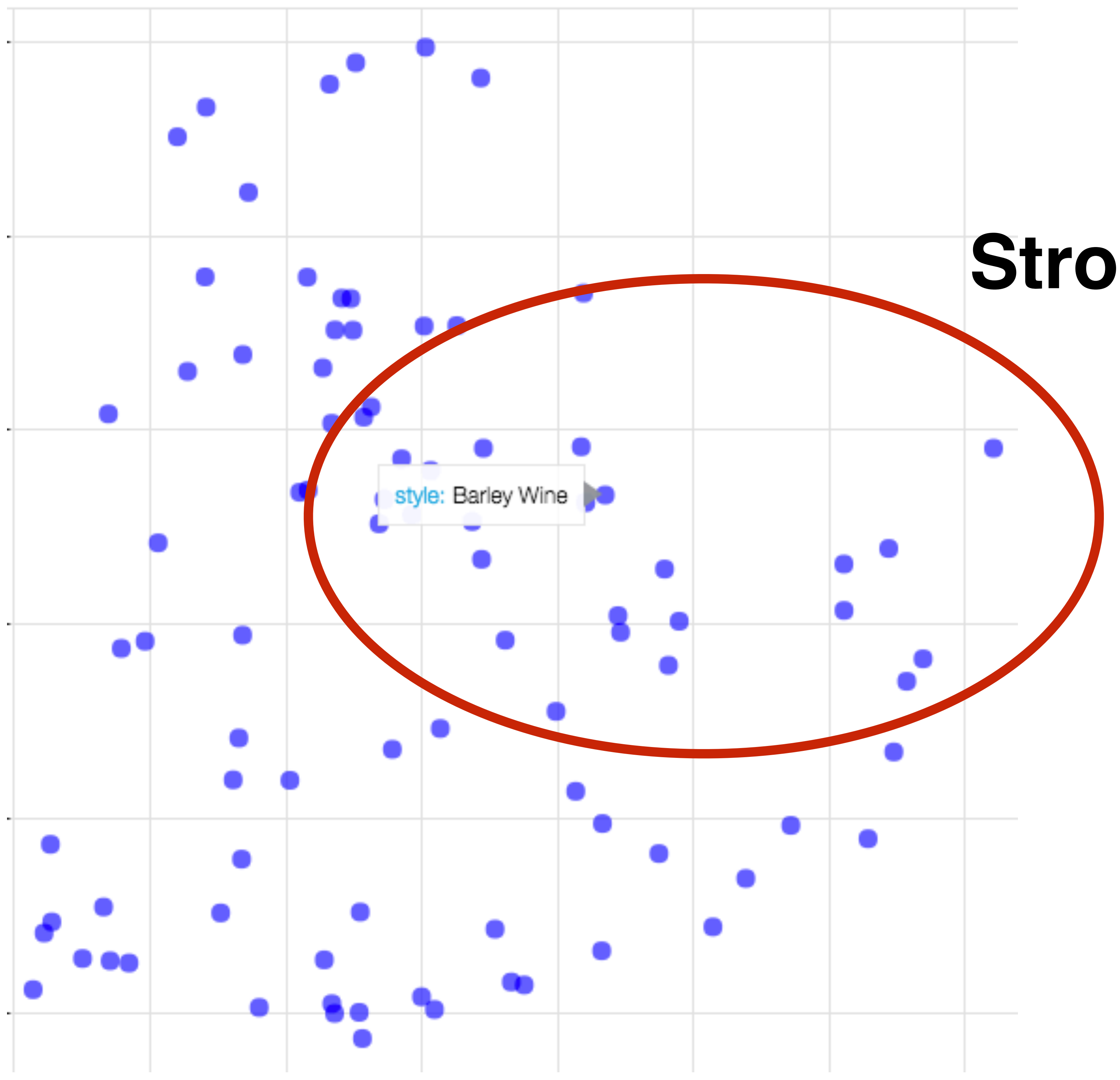
Case Study 2: Beer!

```
model.most_similar([model.docvecs['Wheat Ale']])

[('lemon', 0.6103),
 ('lemony', 0.5909),
 ('wheaty', 0.5873),
 ('germ', 0.5684),
 ('lemongrass', 0.5653),
 ('wheat', 0.5649),
 ('lime', 0.55636),
 ('verbena', 0.5491),
 ('coriander', 0.5341),
 ('zesty', 0.5182)]
```

PCA: scikit-learn — Data Viz: Bokeh

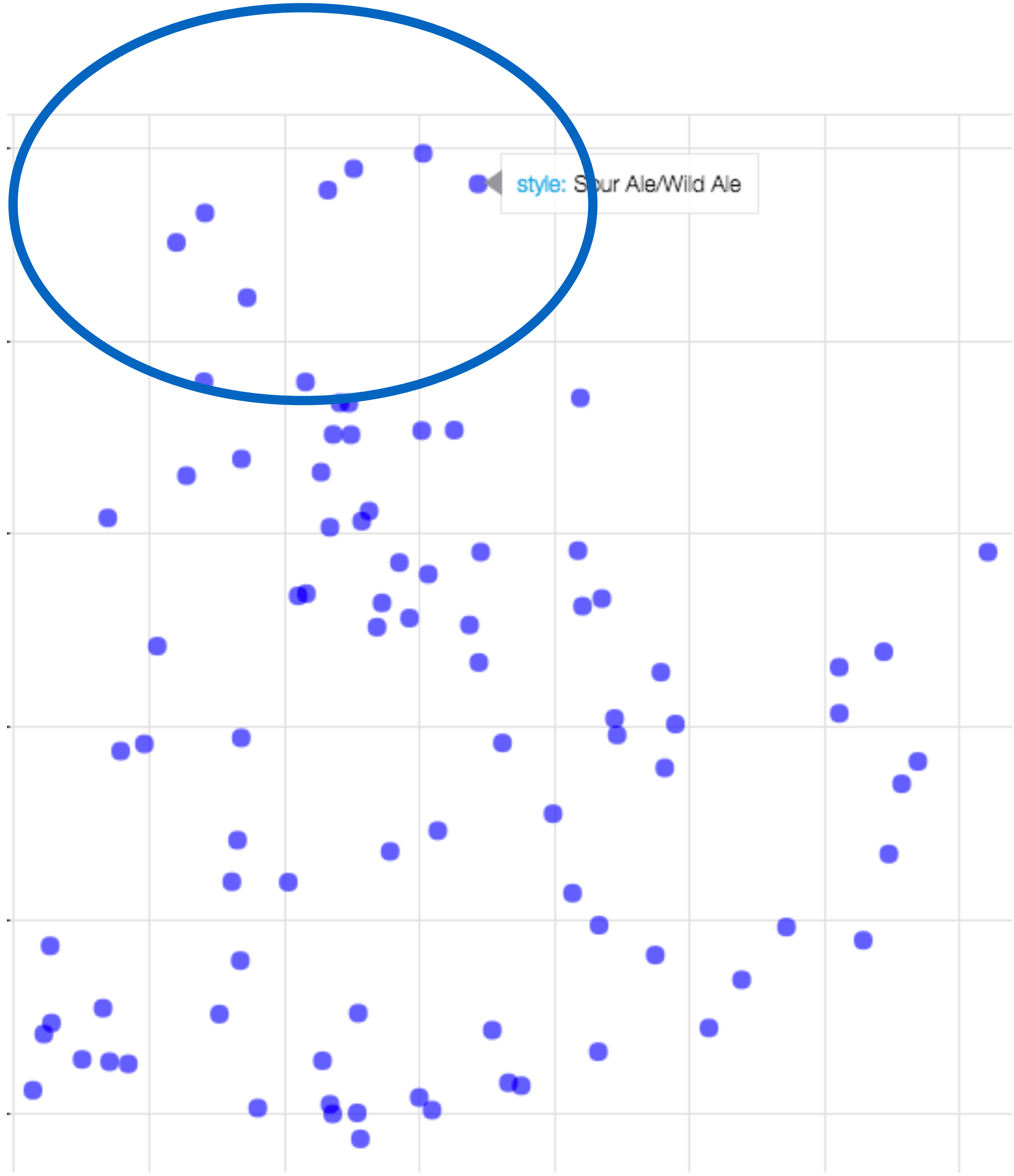


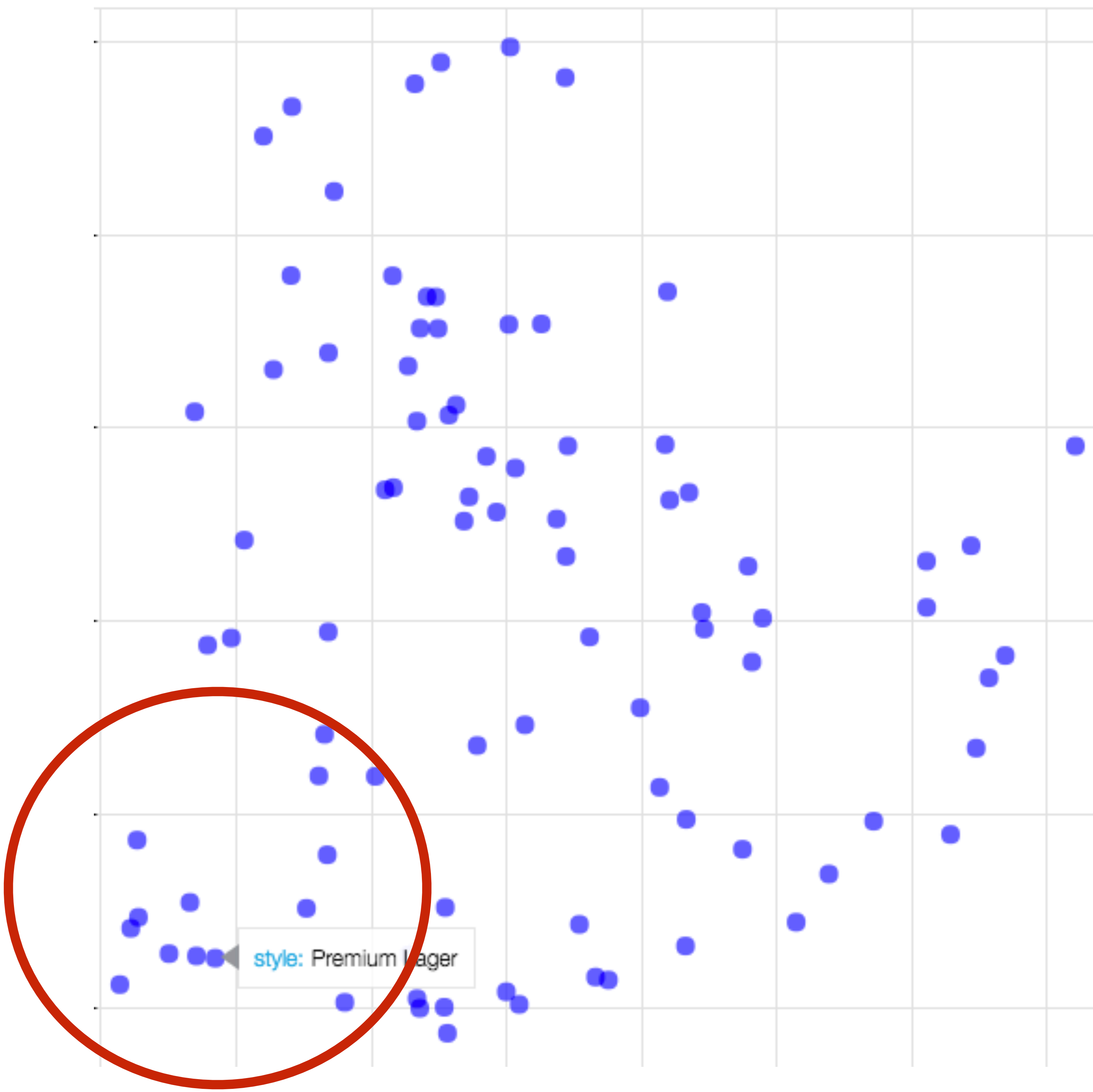

style: Imperial Stout
Dark beers



Strong beers

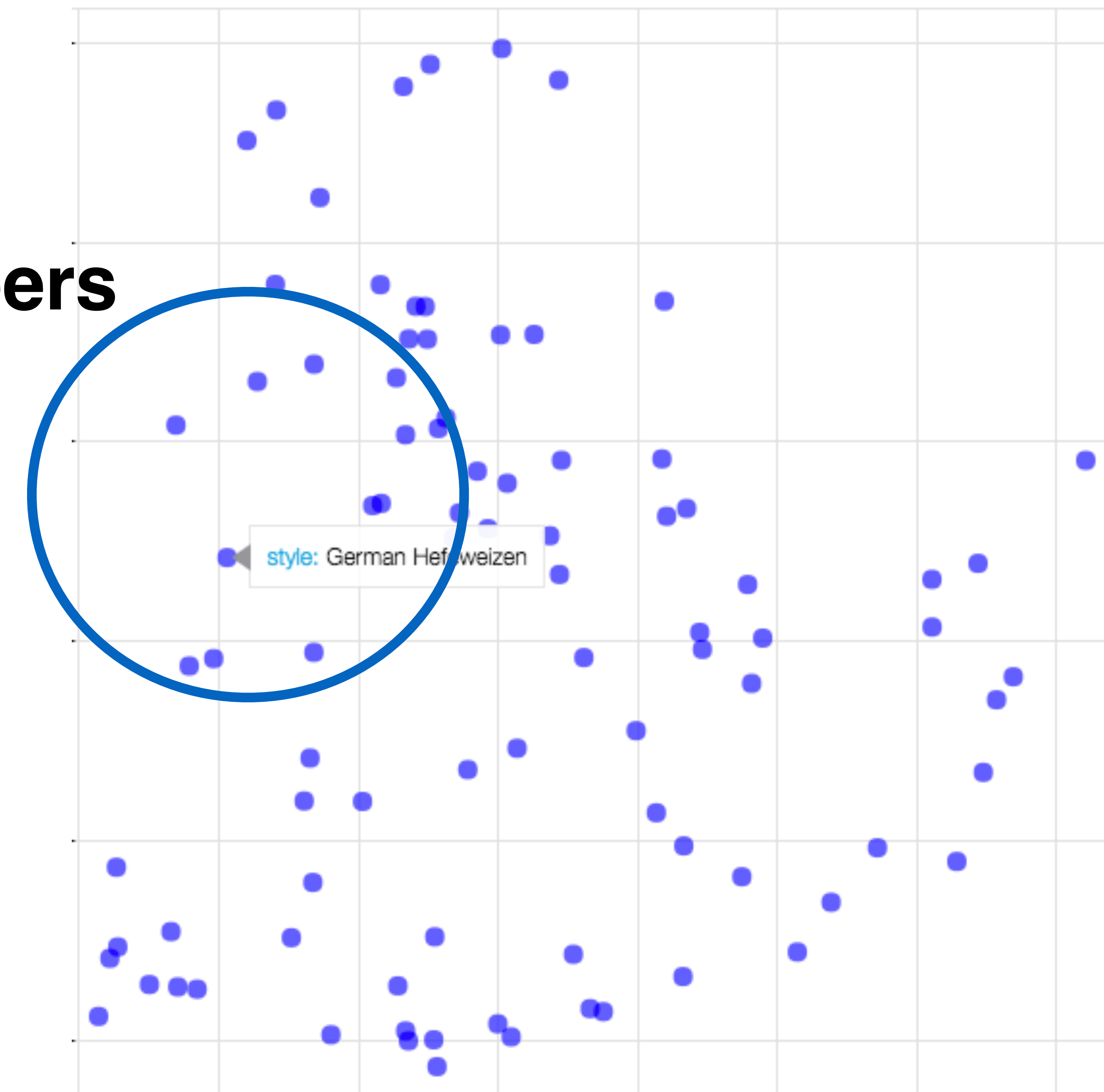
Sour beers





Lagers

Wheat beers



Case Study 2: Beer!

Useful for:

Understanding the language of beer enthusiasts

Planning your next pint

Classification

Case Study 3: Evil AI

Case Study 3: Evil AI

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

fname = ‘GoogleNews-vectors.bin'

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

Case Study 3: Evil AI

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

Case Study 3: Evil AI

```
model.most_similar(
    positive=['king', ‘woman'],
    negative=[‘man’]
)
    [('queen', 0.7118),
     ('monarch', 0.6189),
     ('princess', 0.5902),
     ('crown_prince', 0.5499),
     ('prince', 0.5377),
    …]
```

Case Study 3: Evil AI

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

Case Study 3: Evil AI

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

Case Study 3: Evil AI

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

Case Study 3: Evil AI

```
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),
   …]
```

Case Study 3: Evil AI

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

Case Study 3: Evil AI

```
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),
    …]
```

Case Study 3: Evil AI

- Culture is biased

Case Study 3: Evil AI

- Culture is biased
- Language is biased

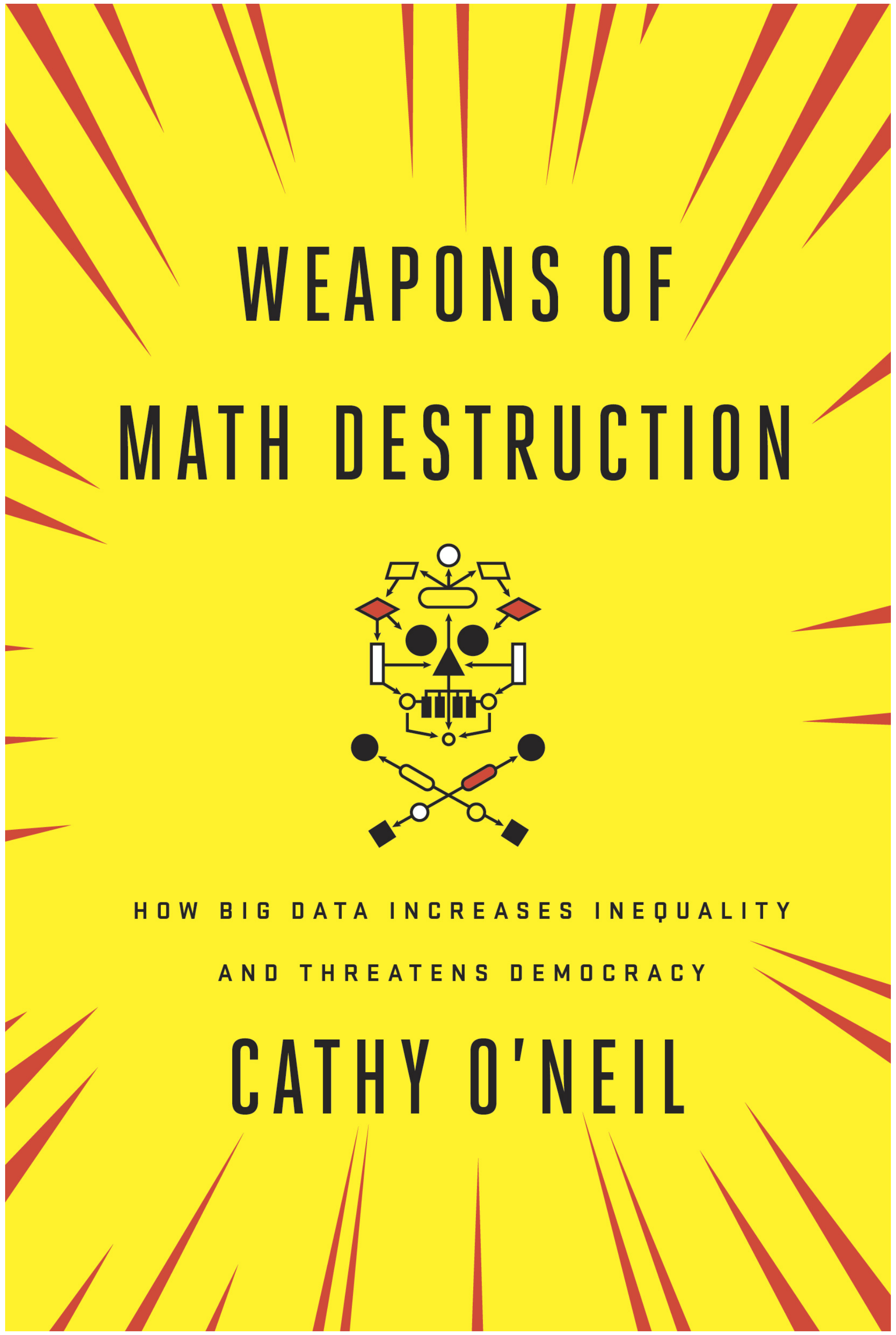
Case Study 3: Evil AI

- Culture is biased
- Language is biased
- Algorithms are not?

Case Study 3: Evil AI

- Culture is biased
- Language is biased
- Algorithms are not?
- “Garbage in, garbage out”

Case Study 3: Evil AI



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
- Think SVD / LSA / LDA
- … but they are usually outperformed by word2vec
- … and don't scale as well as word2vec

Efficiency

Efficiency

- There is no co-occurrence matrix (vectors are learned directly)
- Softmax has complexity O(V) Hierarchical Softmax only O(log(V))

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

Credits & Readings

Credits & Readings

Credits

- Lev Konstantinovskiy (`@gensim_py`)
- Chris E. Moody (`@chrisemoody`) see videos on lda2vec

Readings

- Deep Learning for NLP (R. Socher) `http://cs224d.stanford.edu/`
- “word2vec parameter learning explained” by Xin Rong

More readings

- “GloVe: global vectors for word representation” by Pennington et al.
- “Dependency based word embeddings” and “Neural word embeddings as implicit matrix factorization” by O. Levy and Y. Goldberg

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`

THANK YOU

@MarcoBonzanini

GitHub.com/bonzanini

marcobonzanini.com