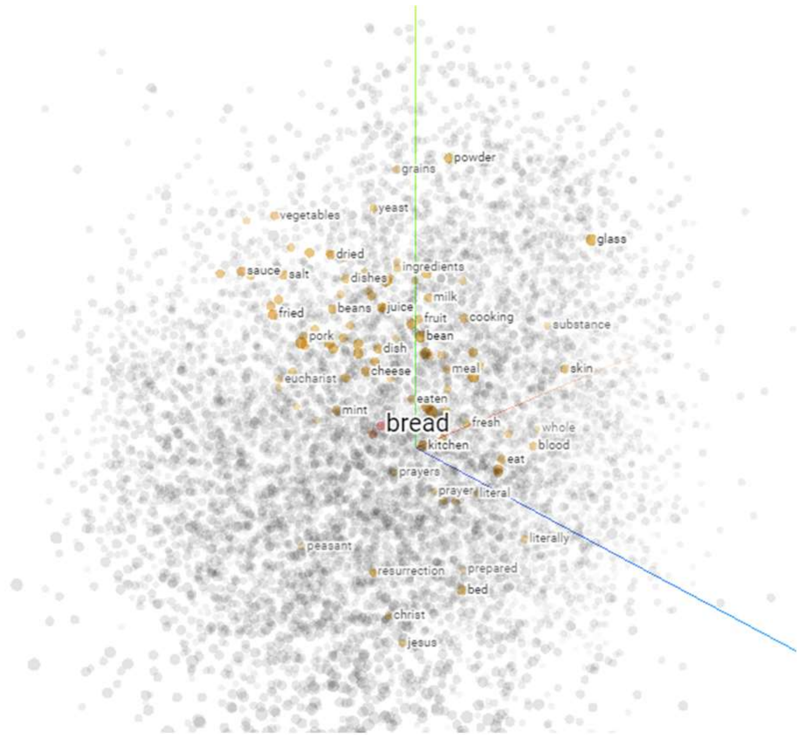
Introduction to word2vec (skip-gram)

Farid Khafizov

Aug 19, 2021



http://projector.tensorflow.org/

Content

1. Introduction
    a. Define “meaning”
    b. Learning (new) words
2. Problem statement
3. Skip-gram Algorithm [1,2]
4. Demonstration
5. How technique becomes technology
6. Some Fun Examples

[1] https://arxiv.org/pdf/1301.3781.pdf Mikolov 2013.09
[2] https://arxiv.org/pdf/1310.4546.pdf Mikolov 2013.10

Problem statement: identify words with similar meaning

Q1: Can machines learn the meaning of words?

What do we mean by "meaning"?

Winograd schema challenge

- The city councilmen refused the demonstrators a permit because they feared violence.
- The city councilmen refused the demonstrators a permit because they advocated violence.

Natural language understanding (NLU) is central in NLP.
Kevin Gimpel: “the biggest open problems (in NLP) are related to natural language understanding. [...] we should develop systems that read and understand text the way a person does”

How do we learn meaning of words?

1. He handed her a glass of bardiwac.
2. Pete staggered to his feet, face flushed from too much bardiwac.
3. Malbec, one of the lesser-known bardiwac grapes, responds well to Australia’s sunshine.
4. The drinks were delicious: blood-red bardiwac as well as light, sweet Rhenish.
5. I dined off bread and cheese and this excellent bardiwac.
6. Beef dished are made to complement the bardiwac.

Answer to Q1

- Machines **<u>can not</u>** learn “meaning” of words. We are not there yet.

- However machines can figure out if a word representation is **<u>close</u>** to another word’s representation in a given **<u>context</u>**.

Embedding

Text = "Ann was happy to be finally home.

Pete was thrilled to see Ann returned."

One-hot encoding (sparse embedding):

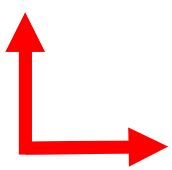
- "ann" = [1,0,…,0,…,0,…,0] = v_{ann}
- "was" = [0,1,…,0,…,0,…,0]
- "happy" = [0,0,…,1,…,0,…,0] = v_{happy}
- "thrilled" = [0,0,…,0,…,1,…,0] = $v_{thrilled}$
- "returned" = [0,0,…,0,…,0,…,1]

Sparse embedding
vs
Dense embedding

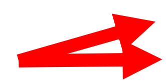
Which one is better?

Dense embedding:

- "ann" = [0.3, 0.2, 0.3, 0.2]
- "was" = [0.1, 0.1, 0.3, 0.5]
- "happy" = [0.00, 0.21, 0.11, 0.69]
- "thrilled" = [0.01, 0.20, 0.13, 0.66]
- "returned" = [0.4, 0.22, 0.28, 0.1]



Similarity(v_{happy}, $v_{thrilled}$) = 0



Similarity(v_{happy}, $v_{thrilled}$) = 0.98

Illustration

https://turbomaze.github.io/word2vecjson

Similar Words

Enter a word and see words with similar vectors.

happy [List words]

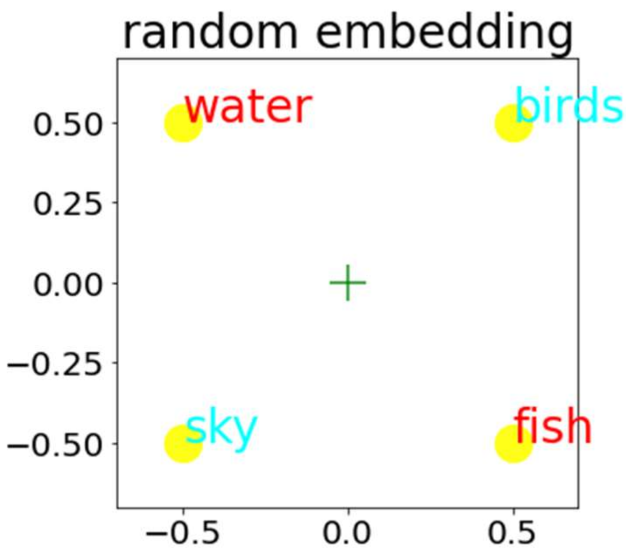
happy	1
glad	0.7408889975979509
pleased	0.6632168300400823
thrilled	0.651404662202906
satisfied	0.6437953159801051
proud	0.6360420824282996
delighted	0.6272380305363671
disappointed	0.6269950290513414
excited	0.6247663940787201
happier	0.624462516505867

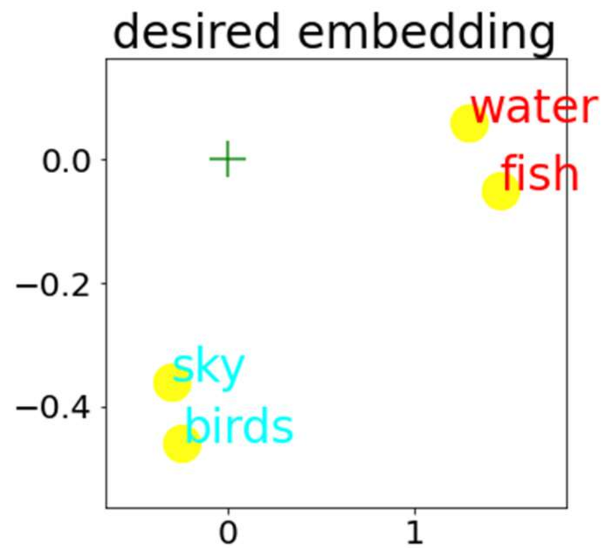
A simpler problem

Separate words according to their context
via embedding words to 2D space.

Note: for us a word embedding is a 2D vector, or
a point on a plane

Random Embedding and Desired Embedding





word	desired_embedding
birds	[-0.24, -0.46]
fish	[1.46, -0.05]
sky	[-0.3, -0.36]
water	[1.29, 0.06]

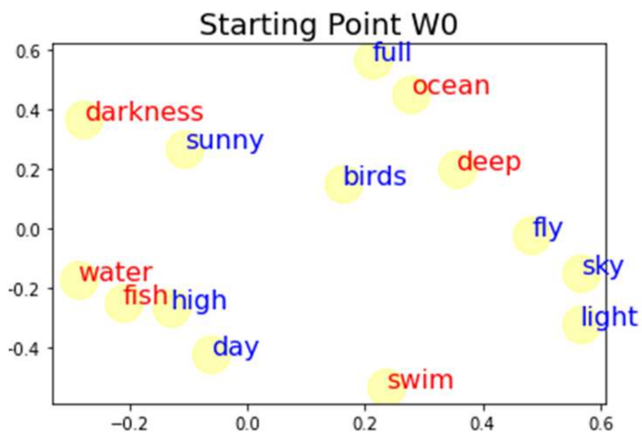
Find embedding reflecting semantics of words in a given context.

GIVEN TEXT:

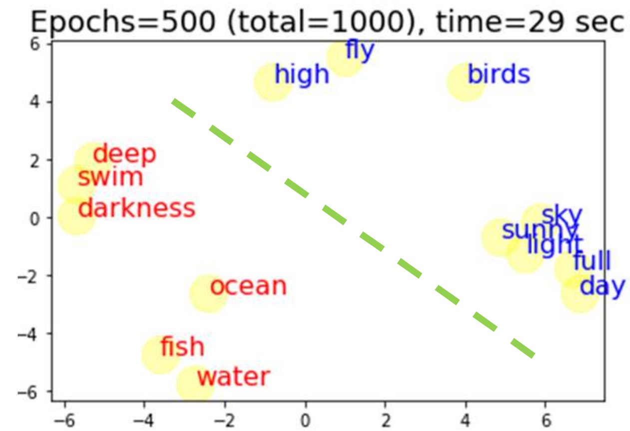
- Fish swim in deep water.
- Ocean is very deep.
- Fish swim in darkness.
- Birds are high in the sky.
- Birds fly very high.
- On a sunny day the sky is full of light.

VOCABULARY (14 words)= ['birds', 'darkness', 'day', 'deep', 'fish', 'fly', 'full', 'high', 'light', 'ocean', 'sky', 'sunny', 'swim', 'water']

and RANDOM embedding:



Find a BETTER embedding:



APPROACH:

- Use context
- WINDOW SIZE = 2
- Embedding Dim=2
- 44 CONTEXT PAIRS

	input	label
0	fish	swim
1	fish	deep
2	swim	fish
3	swim	deep
4	swim	water
43	light	full

Method

Pairs of words for training

	center	context
0	fish	swim
1	fish	deep
2	swim	fish
3	swim	deep
4	swim	water

...

39	full	day
40	full	sky
41	full	light
42	light	sky
43	light	full

- Fish swim in deep water.
- Ocean is very deep.
- Fish swim in darkness.
- Birds are high in the sky.
- Birds fly very high.
- On a sunny day the sky is full of light.

Random initialization of W0

vocab	init_embedding
birds	[0.04, 0.15]
darkness	[-0.8, -0.68]
day	[0.27, 1.54]
deep	[1.57, -0.06]
fish	[0.61, 1.68]
fly	[1.2, 1.08]
full	[-1.68, -2.54]
high	[-0.83, 0.72]
light	[0.69, -0.93]
ocean	[-1.51, -3.47]
sky	[-0.54, 0.37]
sunny	[-0.97, -0.8]
swim	[-1.3, -0.15]
water	[0.89, 0.31]

Sketch of the Training Process

For (‘birds’ , ‘fly’)
birds: x ---> $x = W_0[0,:] = (0.5, -0.4)$
fly: y ---> $y' = W_1[:,5] = (-0.5, 0.5)^t$

x

W_0 =
```
[ 0.5 -0.4]
[-0.4  0.6]
[ 0.5 -0.3]
[ 0.5 -0.1]
[-0.4 -0.4]
[ 0.2 -0.2]
[-0.2 -0.2]
[ 0.2  0.5]
[-0.2  0.1]
[ 0.1 -0.1]
[ 0.5  0.1]
[-0.6 -0.  ]
[ 0.3 -0.2]
[-0.6 -0.  ]
```

y’

W_1 =
```
[-0.2  0.5  0.   0.5 -0.3 -0.5 -0.1  0.1  0.2 -0.1 -0.3 -0.5 -0.1  0.2]
[-0.3 -0.2 -0.6 -0.6  0.6  0.5  0.3  0.3  0.6  0.1 -0.1  0.1 -0.1  0.1]
```

x.y’

$x.W_1$ =
```
[0.02  0.33  0.24  0.49 -0.39 -0.45 -0.17 -0.07 -0.14 -0.09 -0.11 -0.29 -0.01  0.06]
```

$x = W_0[0,:]$

$x . W_1$

dot-product(s) = $x . W_1$
$y' = W_1[:,5]$
x.y’
Compute Utility Function
(average log likelihood)

Optimize Weights W_0, W_1

Целевая функция

'fly' 'birds'

$$P(w_{context}|w_{center}) = P(y|x) = \frac{\exp(x \cdot y')}{\sum_{v=1}^{|V|} \exp(x \cdot y'_v)} = \frac{\exp(x \cdot y')}{D(x, W_1)}$$

$$\log P(y_{t+j}|x_t) = x_t \cdot y'_{t+j} - \log D(x_t, W_1)$$

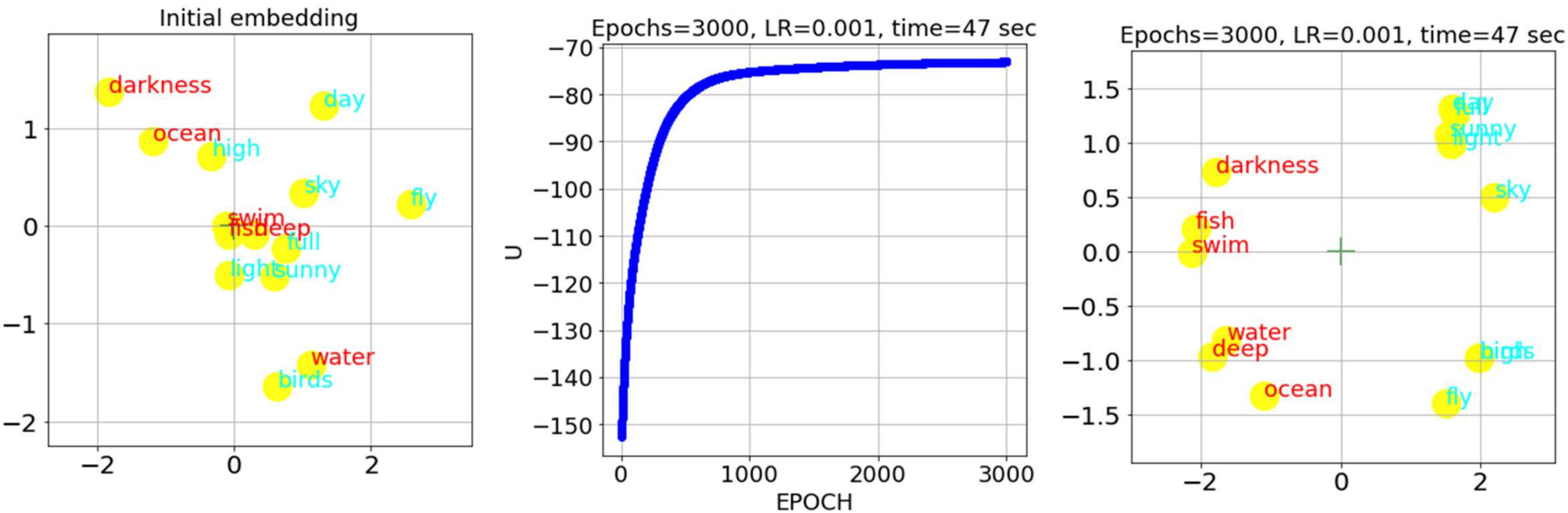
$$U(W_0, W_1) = \frac{1}{T} \sum_{t=1}^{T} \sum_{-c \leq j \leq c, j \neq 0} \log P(y_{t+j}|x_t)$$

Given a sequence of training words $w_1, w_2, w_3, \ldots, w_T$, the objective of the Skip-gram model is to maximize $U(W_0, W_1)$.

$$\max_{W_0, W_1} U(W_0, W_1)$$

In our example: |V|=14, T=44 is the size of training set, c=2 is the context window size.

Results



Initial embedding
darkness
ocean
high
day
sky
fly
swim
full
light
sunny
water
birds
Epochs=3000, LR=0.001, time=47 sec
U
EPOCH
Epochs=3000, LR=0.001, time=47 sec
darkness
fish
swim
sky
water
deep
ocean
fly

Q2: Can we simplify skip-gram?

Let $W_1 = W_0^t$

For (‘birds’ , ‘fly’)
birds: x ---> $x = W_0[0,:] = (0.5, -0.4)$
fly: y ---> $y' = W_0^t[:,5] = (0.2, -0.2)^t$

$$W_0 = \begin{array}{c} x \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \end{array} \begin{bmatrix} \boxed{0.5 & -0.4} \\ -0.4 & 0.6 \\ 0.5 & -0.3 \\ 0.5 & -0.1 \\ -0.4 & -0.4 \\ \boxed{0.2 & -0.2} \\ -0.2 & -0.2 \\ 0.2 & 0.5 \\ -0.2 & 0.1 \\ 0.1 & -0.1 \\ 0.5 & 0.1 \\ -0.6 & -0. \\ 0.3 & -0.2 \\ -0.6 & -0. \end{bmatrix} \begin{array}{c} \\ \\ \\ \\ \\ y \\ \\ \\ \\ \\ \\ \\ \\ \\ \end{array}$$

$x = W_0[0,:]$

$x \,.\, W_0^t$

dot-product(s) = $x \,.\, W_0$
$y' = W_0^t[:,5]$
$x.y'$
Compute Utility Function
(average log likelihood)

Optimize Weights W_0

Part4: Demo

https://github.com/fkhafizov/w2v_intro

How technique becomes technology

Q3: How to scale?

How many words are out there?

- 470k in Webser’s 3rd New International Dictionary
- 170k in current use per Oxford English Dictionary
- 20k-35k used by average native speaker

Technique ==> Technology

1. Distributed architecture and vocab optimization

2. Subsampling frequent words ==> Speed up computation

Idea: Discard each training word w_i with probability $P(w_i)$

$$P(w_i) = \begin{cases} \sqrt{1 - \frac{t}{f(w_i)}} & , \quad \text{if } f(w_i) > t \\ 0 & , \quad \text{otherwise} \end{cases}$$

3. Negative sampling (NEG) ==> Improves optimization.

Idea: add words from "negative" sample to computation of utility function

Subsampling of frequent words

In very large corpora, the most frequent words can easily occur hundreds of millions of times (e.g., "in", "the", and "a"). Such words usually provide less information value than the rare words. For example, while the Skip-gram model benefits from observing the co-occurrences of "France" and "Paris", it benefits much less from observing the frequent co-occurrences of "France" and "the", as nearly every word co-occurs frequently within a sentence with "the". This idea can also be applied in the opposite direction; the vector representations of frequent words do not change significantly after training on several million examples.

To counter the imbalance between the rare and frequent words, we used a simple subsampling approach: each word w_i in the training set is discarded with probability computed by the formula

$$P(w_i) = 1 - \sqrt{\frac{t}{f(w_i)}} \tag{5}$$

where $f(w_i)$ is the frequency of word w_i and t is a chosen threshold, typically around 10^{-5}. We chose this subsampling formula because it aggressively subsamples words whose frequency is greater than t while preserving the ranking of the frequencies. Although this subsampling formula was chosen heuristically, we found it to work well in practice. It accelerates learning and even significantly improves the accuracy of the learned vectors of the rare words, as will be shown in the following sections.

[Mikolov 2013.10]

Negative Sampling

simplify NCE as long as the vector representations retain their quality. We define Negative sampling (NEG) by the objective

$$\log \sigma({v'_{w_O}}^\top v_{w_I}) + \sum_{i=1}^{k} \mathbb{E}_{w_i \sim P_n(w)} \left[\log \sigma(-{v'_{w_i}}^\top v_{w_I}) \right] \tag{4}$$

which is used to replace every $\log P(w_O|w_I)$ term in the Skip-gram objective. Thus the task is to distinguish the target word w_O from draws from the noise distribution $P_n(w)$ using logistic regression, where there are k negative samples for each data sample. Our experiments indicate that values of k in the range 5–20 are useful for small training datasets, while for large datasets the k can be as small as 2–5. The main difference between the Negative sampling and NCE is that NCE needs both

[Mikolov 2013.10]

BAD SOLUTION: w_I=x='birds' w_O=y='ocean' w_n=n='fly'

x= [1.1 -3.9] n= [0.1 -4.0] xn= 15.7, sig(-xn)=1.5e-07, log(sig(-uv))=-15.7 << 0

x= [1.1 -3.9] y= [-0.1 3.2] xy= -12.6, sig(xy)=3.4e-06, log(sig(uv))= -12.6 << 0

GOOD SOLUTION: w_I=x='birds' w_O=y='fly' w_n=n='ocean'

x= [1.1 -3.9] y= [0.1 -4.0] xy= 15.7, sig(xy)=0.99, log(sig(xy))= -1.5e-07 ~ 0

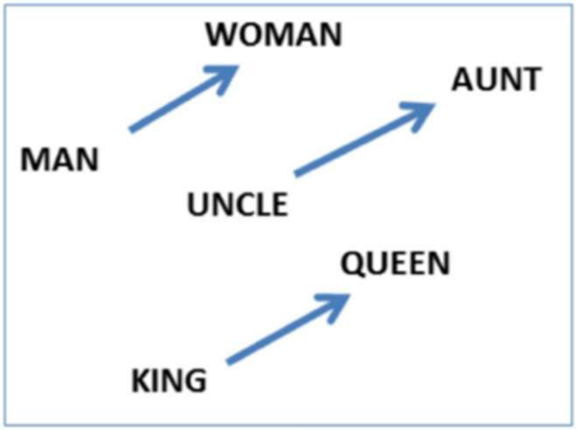
x= [1.1 -3.9] n= [-0.1 3.2] xn= -12.6, sig(-xn)=0.99, log(sig(-xn))= -3.4e-06 ~ 0

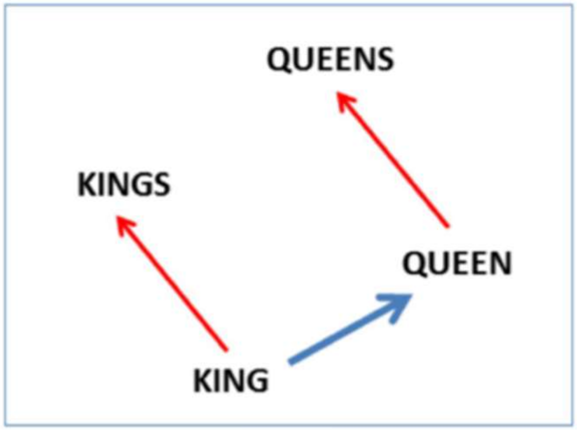
More details on NEG can be found in section 2 of [Goldberg, 2014.02] https://arxiv.org/pdf/1402.3722.pdf

“Word Embedding” search trend

Fun stuff and Interesting observations

$v(\text{king}) - v(\text{man}) + v(\text{woman}) \approx v(\text{queen})$





v(Russia) + v(river) ≈ v(Volga River)
v(Germany) + v(capital) ≈ v(Berlin)

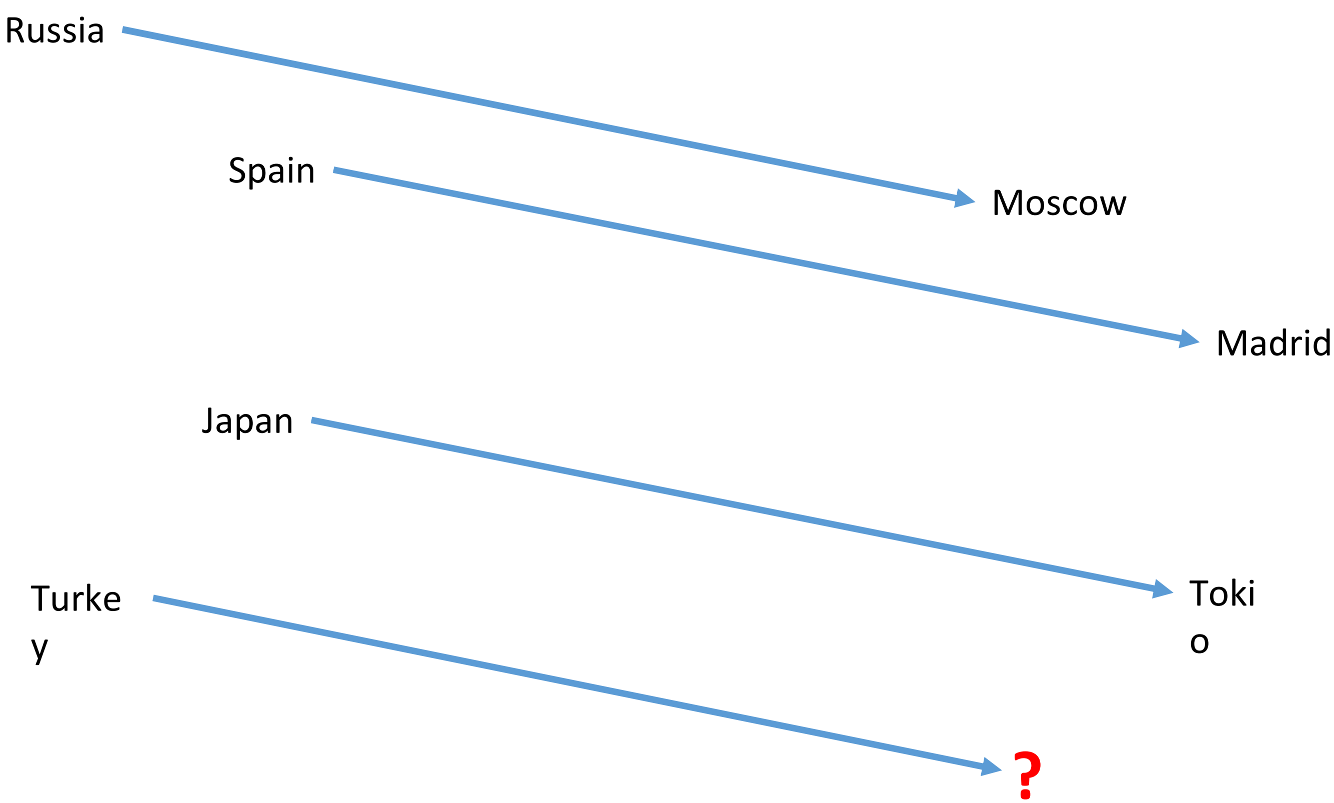
Ref: [1,2]

Word Algebra

France + (Moscow - Russia) = X

X = ?

Geographical Words



Word Embedding Demo
https://turbomaze.github.io/word2vecjson/

Similar Words

Enter a word and see words with similar vectors.

doctor [List words]

doctor	1.0000000000000002
physician	0.7806019127031032
doctors	0.7476568731527384
surgeon	0.6793393714387082
dentist	0.6785442117848048
nurse	0.6319524227288814
psychiatrist	0.614703850361634
medical	0.5671389130686404
clinic	0.5499804910039348
therapist	0.5283346636619084

Word Algebra

Enter all three words, the first two, or the last two and see the words that result.

madrid + (russia - moscow) = [Get result]

spain	0.7905075552539405
madrid	0.7650632609053115

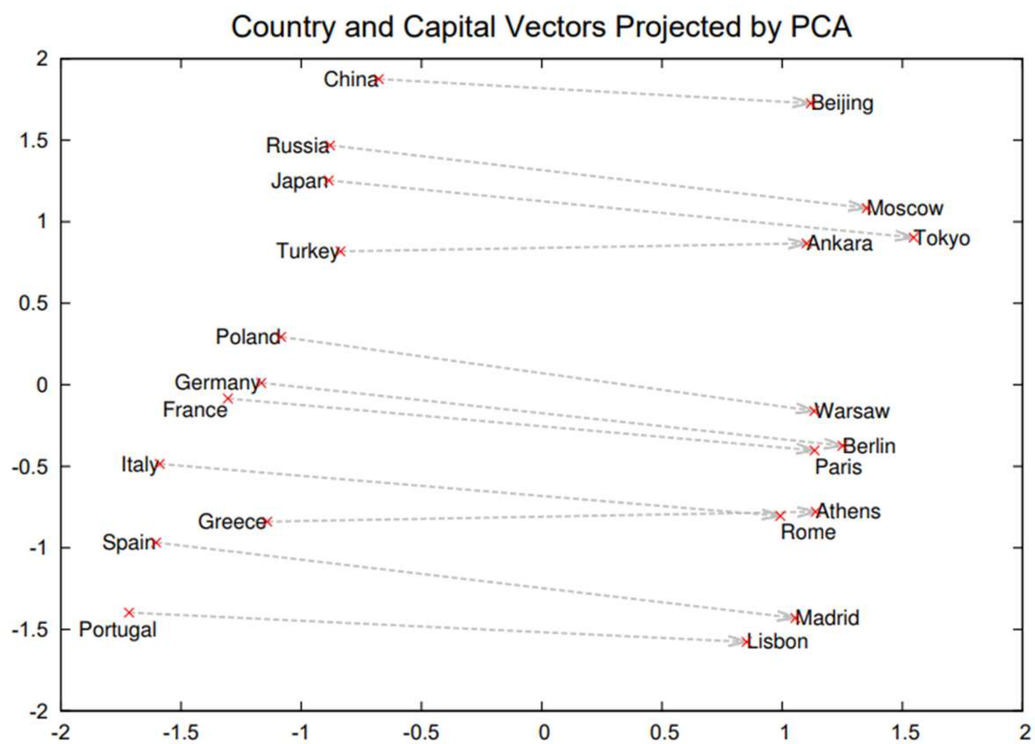
Word Algebra

Enter all three words, the first two, or the last two and see the words that result.

china + (moscow - russia) = [Get result]

shanghai	0.779147686225549
china	0.7691120887831315
chinese	0.6720659726186436
hu	0.5964189163973439
yuan	0.5946876191518002

PCA projection of the 1000-dimensional Skip-gram vectors



Mikolov et al, 2013

References

- [0] Mikolov et al, 2013, Linguistic Regularities in Continuous Space Word Representations, https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/rvecs.pdf
- [1] Mikolov et al 2013.09 https://arxiv.org/pdf/1301.3781.pdf
- [2] Mikolov et al 2013.10 https://arxiv.org/pdf/1310.4546.pdf