Constructing and Evaluating Word Embeddings

Dr Marek Rei and Dr Ekaterina Kochmar
Computer Laboratory
University of Cambridge

Representing words as vectors

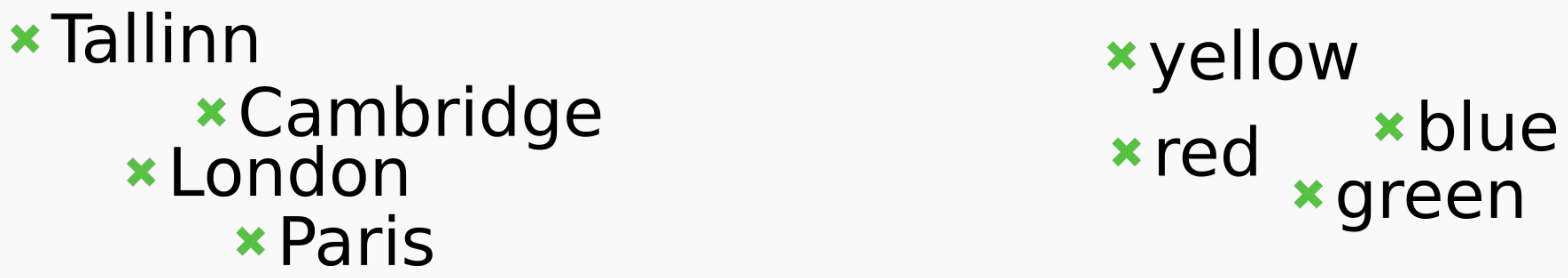
Let’s represent words (or any objects) as vectors.
We want to construct them so that similar words have similar vectors.

Sequence
I live in Cambridge
I live in Paris
I live in Tallinn
I live in yellow

Representing words as vectors

Let's represent words (or any objects) as vectors.
We want to construct them so that similar words have similar vectors.

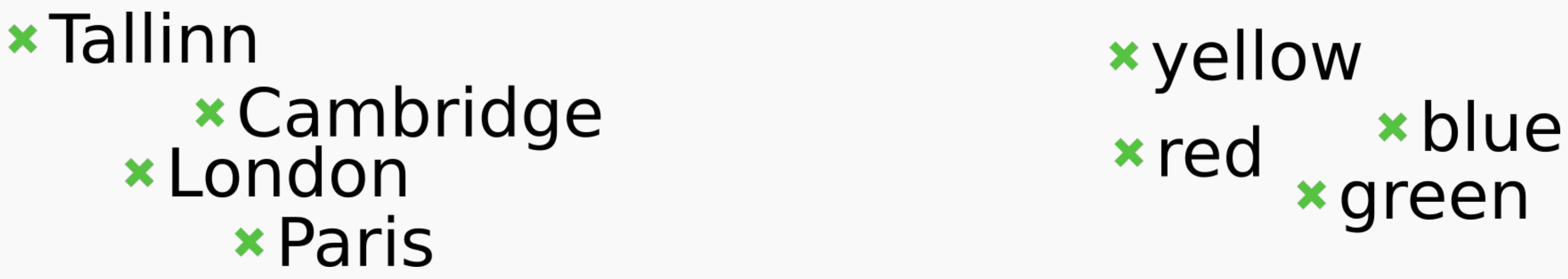
Sequence	Count
I live in Cambridge	19
I live in Paris	68
I live in Tallinn	0
I live in yellow	0



Representing words as vectors

Let’s represent words (or any objects) as vectors.
We want to construct them so that similar words have similar vectors.

Sequence	Count
I live in Cambridge	19
I live in Paris	68
I live in Tallinn	0
I live in yellow	0



1-hot vectors

How can we represent words as vectors?

Option 1: each element represents a different word.

Also known as “1-hot” or “1-of-V” representation.

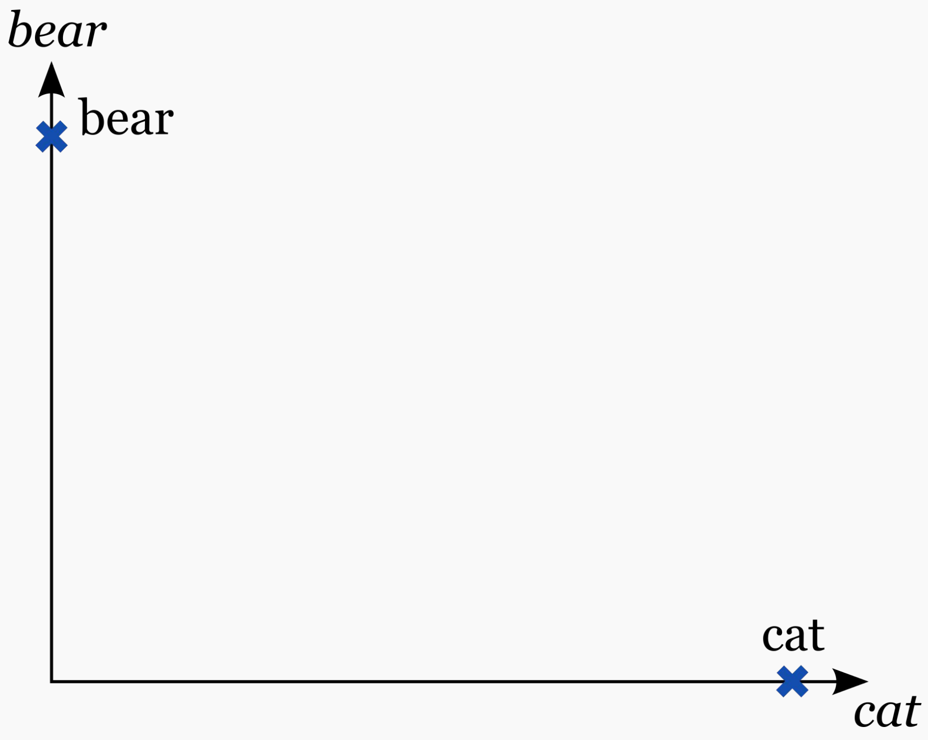
	bear	*cat*	*frog*
bear	1	0	0
cat	0	1	0
frog	0	0	1

```
bear=[1.0, 0.0, 0.0]          cat=[0.0, 1.0, 0.0]
```

1-hot vectors

When using 1-hot vectors, we can’t fit many and they tell us very little.

Need a separate dimension for every word we want to represent.



Distributed vectors

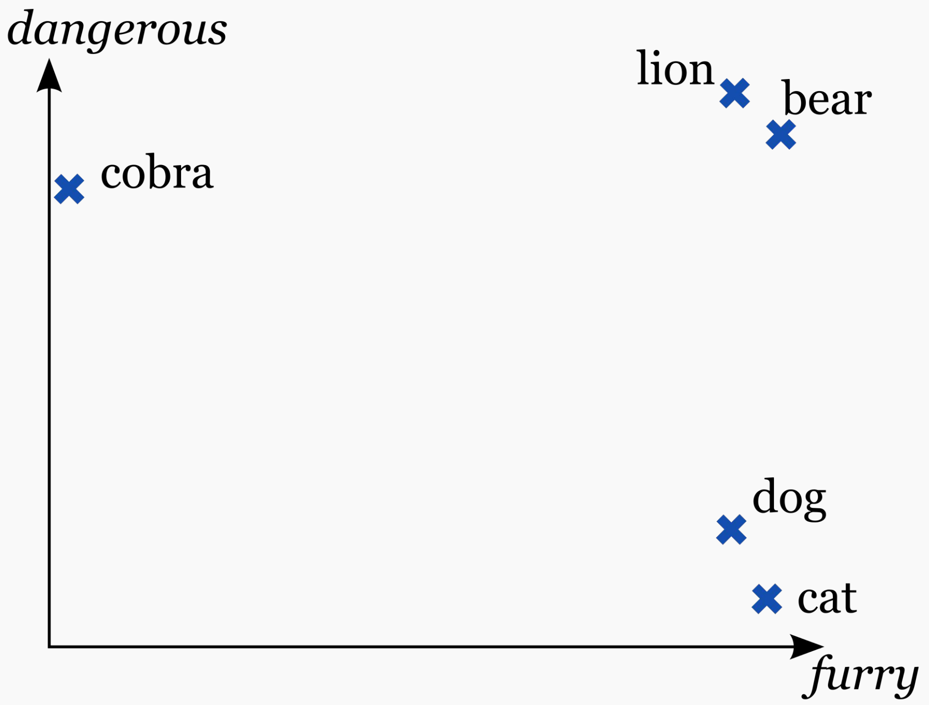
Option 2: each element represents a property, and they are shared between the words.

Also known as “distributed” representation.

	furry	*dangerous*	*mammal*
bear	0.9	0.85	1
cat	0.85	0.15	1
frog	0	0.05	0

```
bear = [0.9, 0.85, 1.0]    cat = [0.85, 0.15, 1.0]
```

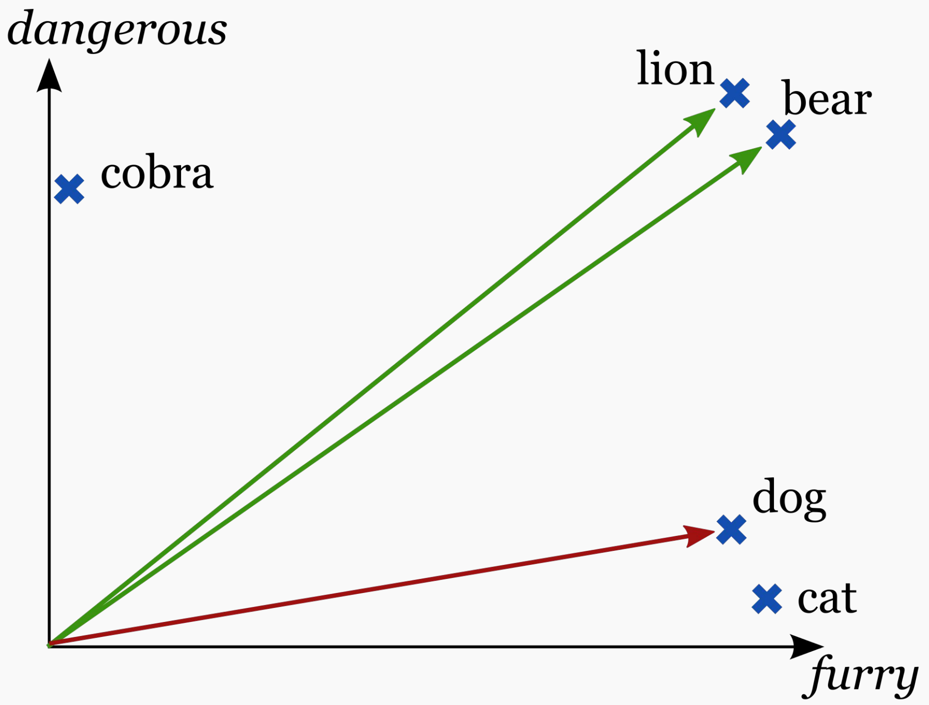
Distributed vectors



	furry	*dangerous*
bear	0.9	0.85
cat	0.85	0.15
cobra	0.0	0.8
lion	0.85	0.9
dog	0.8	0.15

Distributed vectors group similar words/objects together

Distributed vectors



$$cos(a,b) = \frac{\sum_i a_i \cdot b_i}{\sqrt{\sum_i a_i^2} \cdot \sqrt{\sum_i b_i^2}}$$

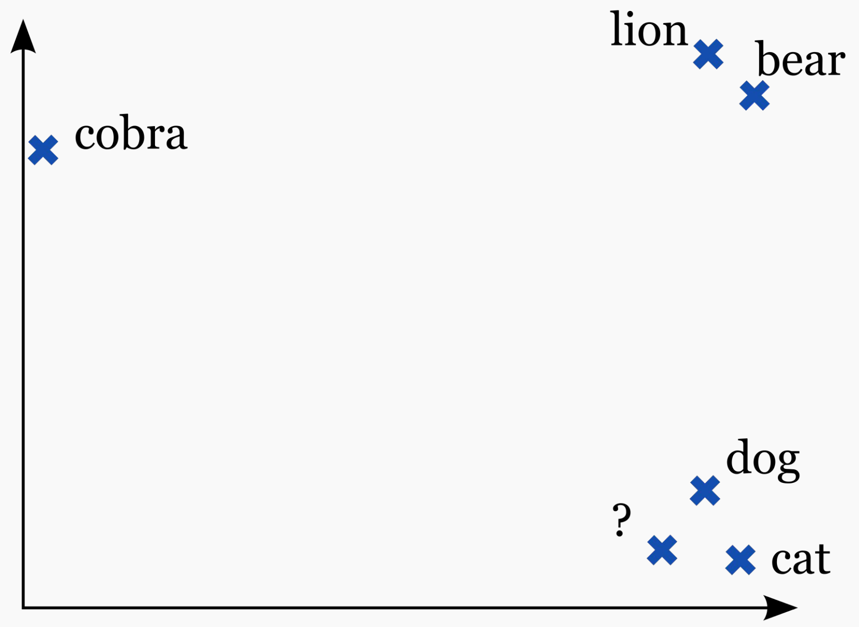
cos(lion, bear) = 0.998

cos(lion, dog) = 0.809

cos(cobra, dog) = 0.727

Can use cosine to calculate similarity between two words

Distributed vectors



We can infer some information, based only on the vector of the word

We don’t even need to know the labels on the vector elements

Distributional hypothesis

Words which are similar in meaning occur in similar contexts.
(Harris, 1954)

You shall know a word by the company it keeps
(Firth, 1957)

He is reading a **magazine**	I was reading a **newspaper**
This **magazine** published my story	The **newspaper** published an article
She buys a **magazine** every month	He buys this **newspaper** every day

Count-based vectors

One way of creating a vector for a word:
Let’s count how often a word occurs together with specific other words.

He is reading a **magazine**

I was reading a **newspaper**

This **magazine** published my story

The **newspaper** published an article

She buys a **magazine** every month

He buys this **newspaper** every day

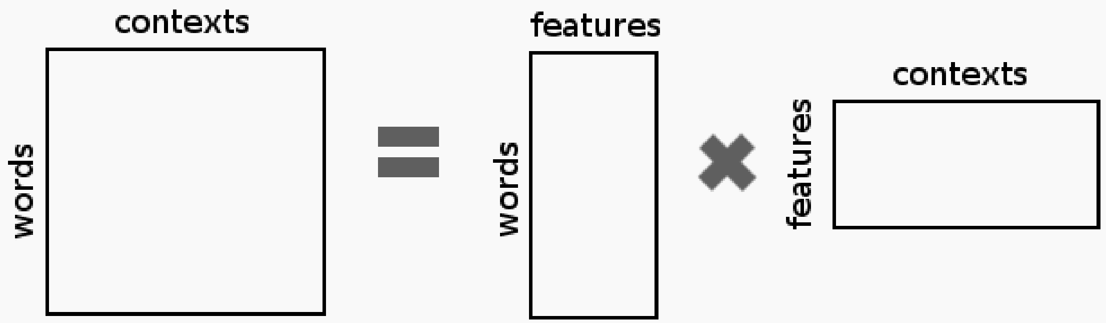
	reading	a	this	published	my	buys	the	an	every	month	day
magazine	1	2	1	1	1	1	0	0	1	1	0
newspaper	1	1	1	1	0	1	1	1	1	0	1

Count-based vectors

- **More frequent words dominate the vectors.**
  Can use a weighting scheme like PMI or TF-IDF.

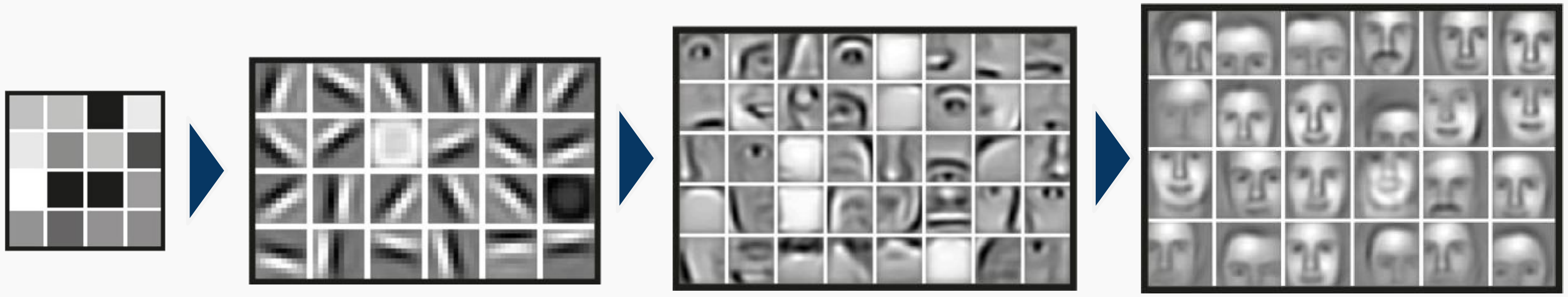
$$pmi(x,z) = log\frac{p(x,z)}{p(x)p(z)} \qquad \text{tf-idf}(w,z) = freq_{w,z} \cdot log\frac{V}{n_z}$$

- **Large number of sparse features**
  Can use matrix decomposition like Singular Value Decomposition (SVD) or Latent Dirichlet Allocation (LDA).



Neural word embeddings

Neural networks will automatically try to discover useful features in the data, given a specific task.



Idea: Let’s allocate a number of parameters for each word and allow the neural network to automatically learn what the useful values should be.

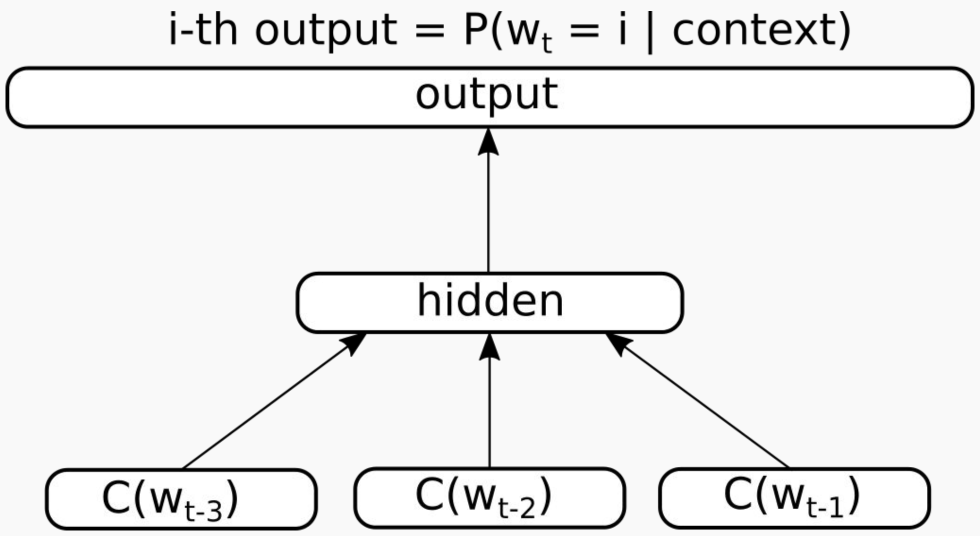
Often referred to as “**word embeddings**”, as we are embedding the words into a real-valued low-dimensional space.

Embeddings through language modelling

Predict the next word in a sequence, based on the previous words.

Use this to guide the training for word embeddings.

Bengio et. al. 2003. *A Neural Probabilistic Language Model.*



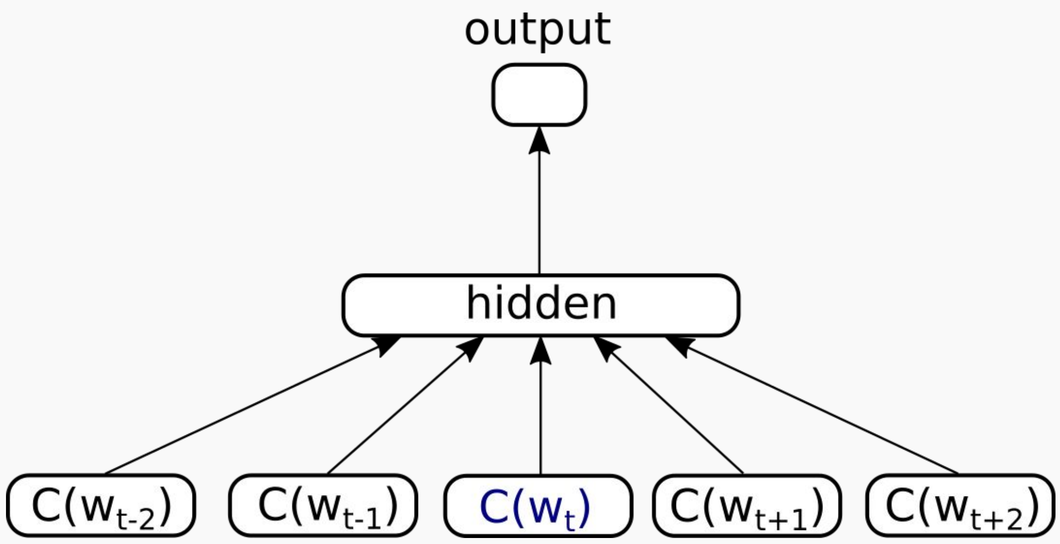
```
I read at my          desk

I study at my         desk
```

Embeddings through error detection

Take a grammatically correct sentence and create a corrupted counterpart.

Train the neural network to assign a higher score to the correct version of each sentence.

Collobert et. al. 2011. *Natural Language Processing (Almost) from Scratch.*

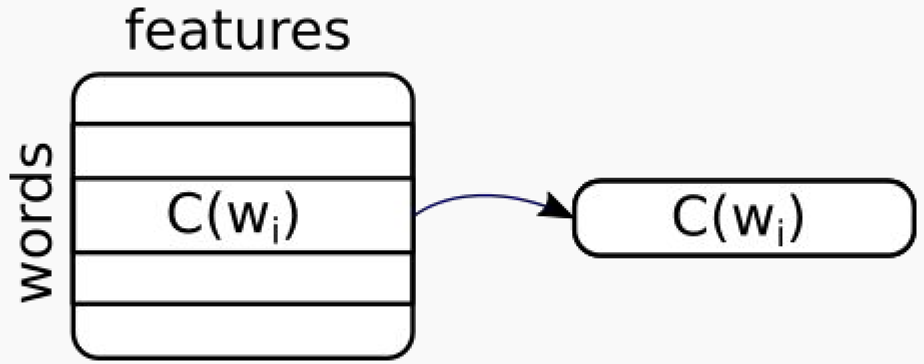


```
my cat climbed a tree

my cat bridge  a tree
```

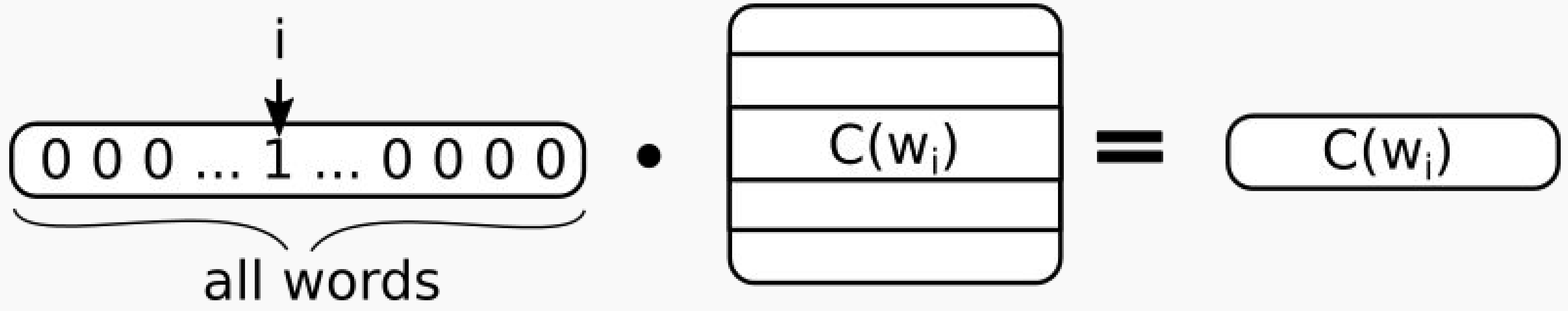
Embedding matrix

Two ways of thinking about the embedding matrix.

1. **Each row** contains a word embedding, which we need to extract



2. It is a normal **weight matrix**, working with a 1-hot input vector



Word2vec

A popular tool for creating word embeddings.

Available from: https://code.google.com/archive/p/word2vec/

Can also download embeddings that are pretrained on 100 billion words.

Preprocess the data!

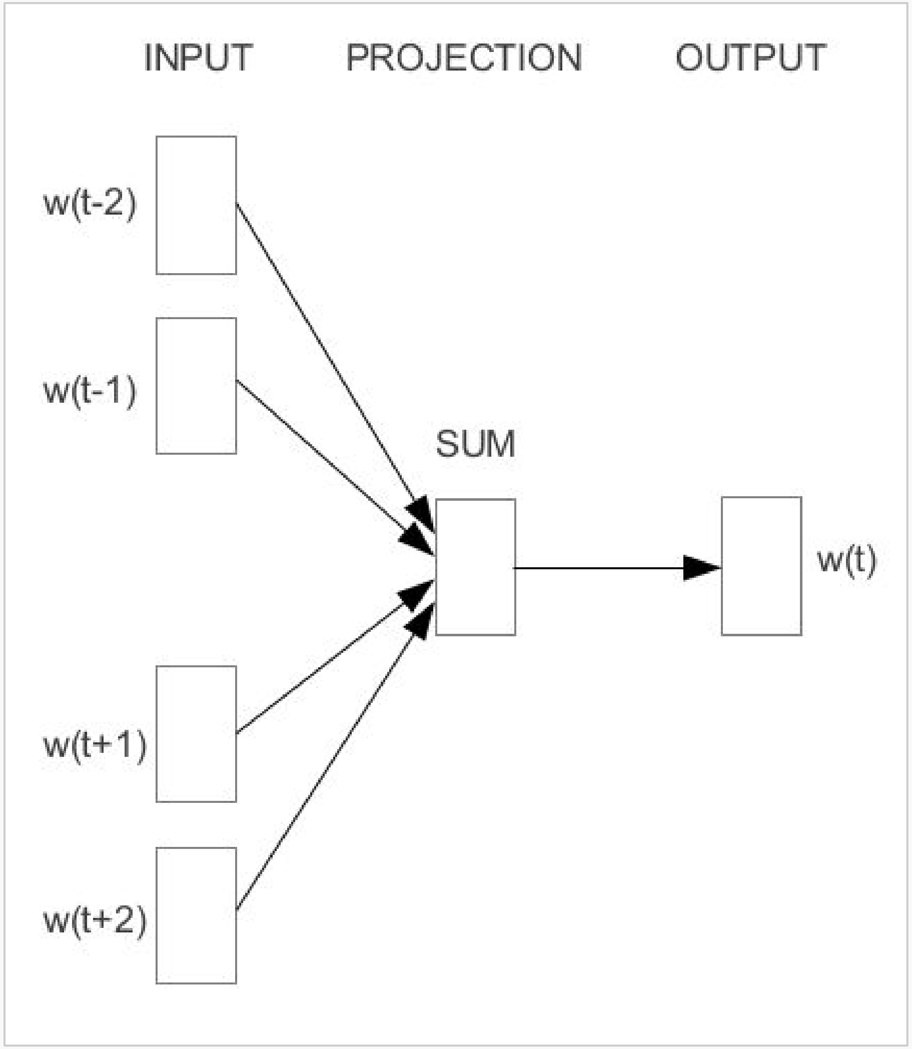
- Tokenise
- Lowercase (usually)

```
./word2vec -train input.txt -output vectors.txt -cbow 0 -size 100
-window 5 -negative 5 -hs 0 -sample 1e-3 -threads 8
```

Continuous Bag-of-Words (CBOW) model

Predict the current word, based on the surrounding words

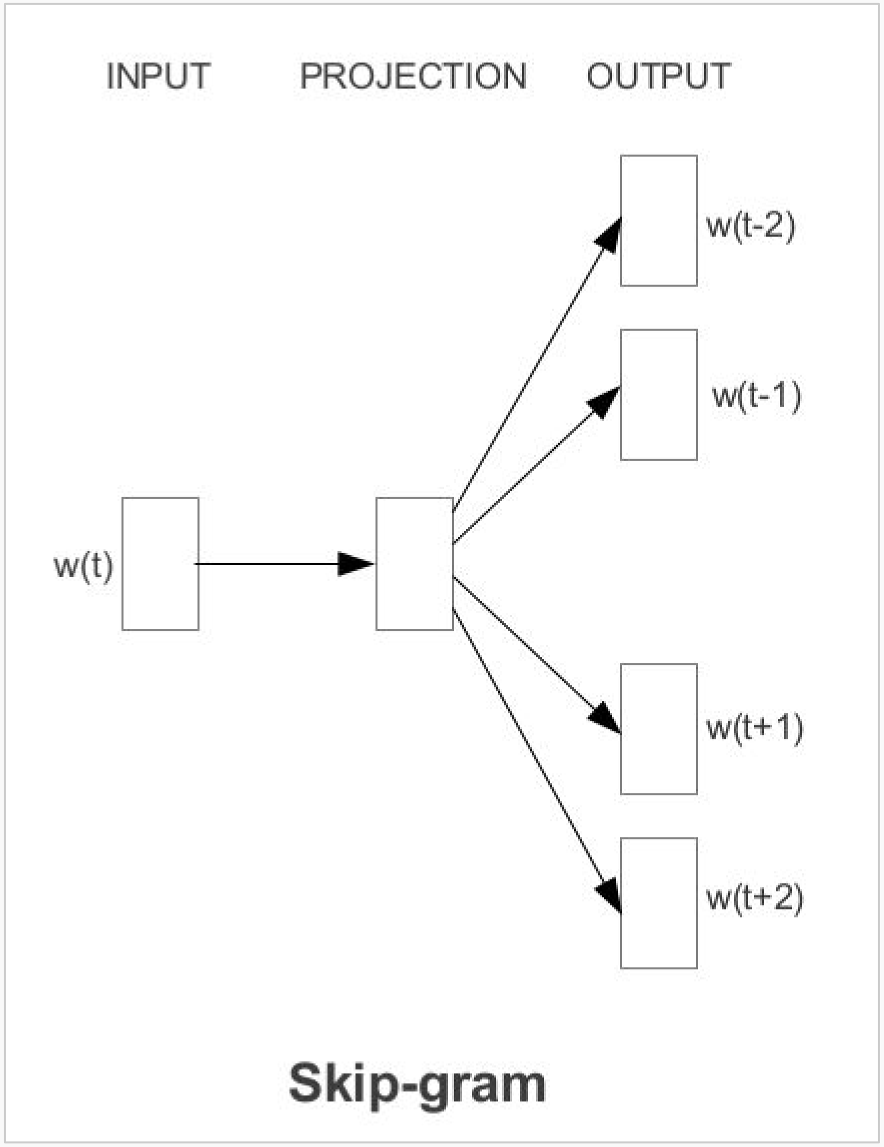
Mikolov et. al. 2013. *Efficient Estimation of Word Representations in Vector Space.*



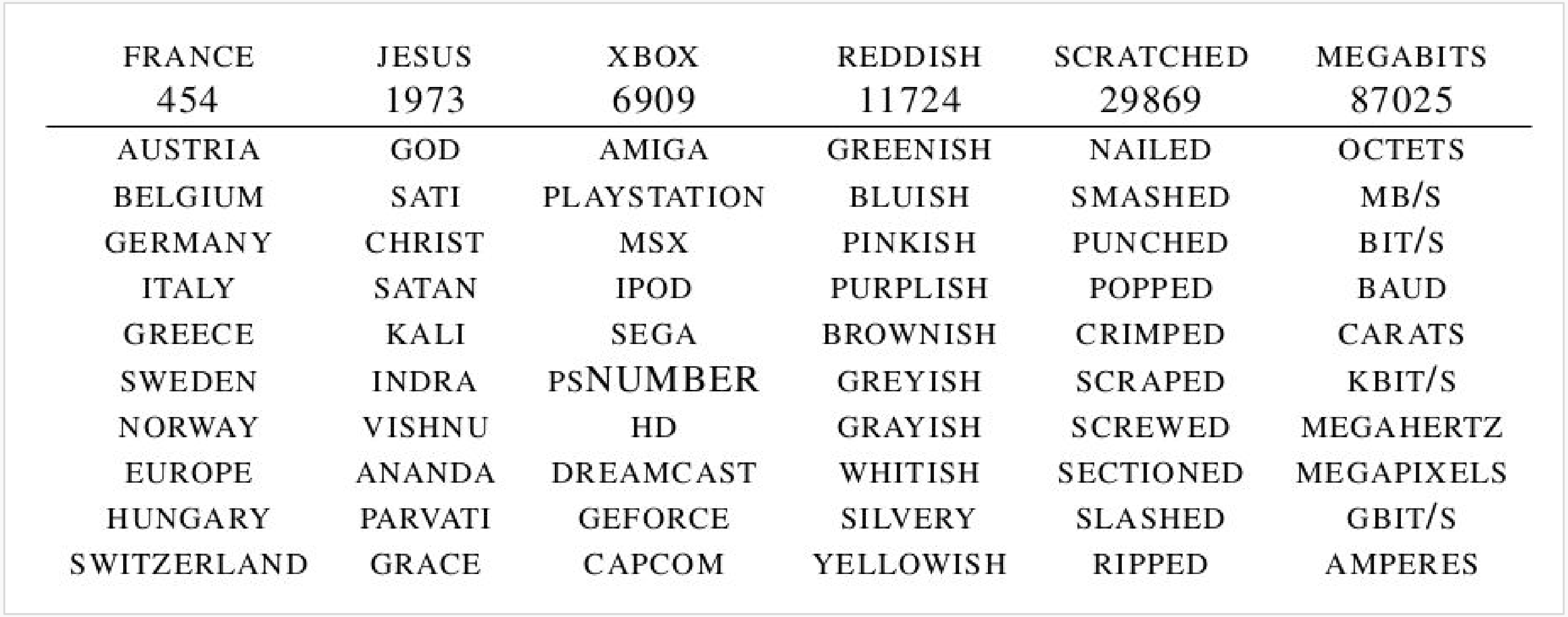
Skip-gram model

Predict the surrounding words, based on the current word.

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



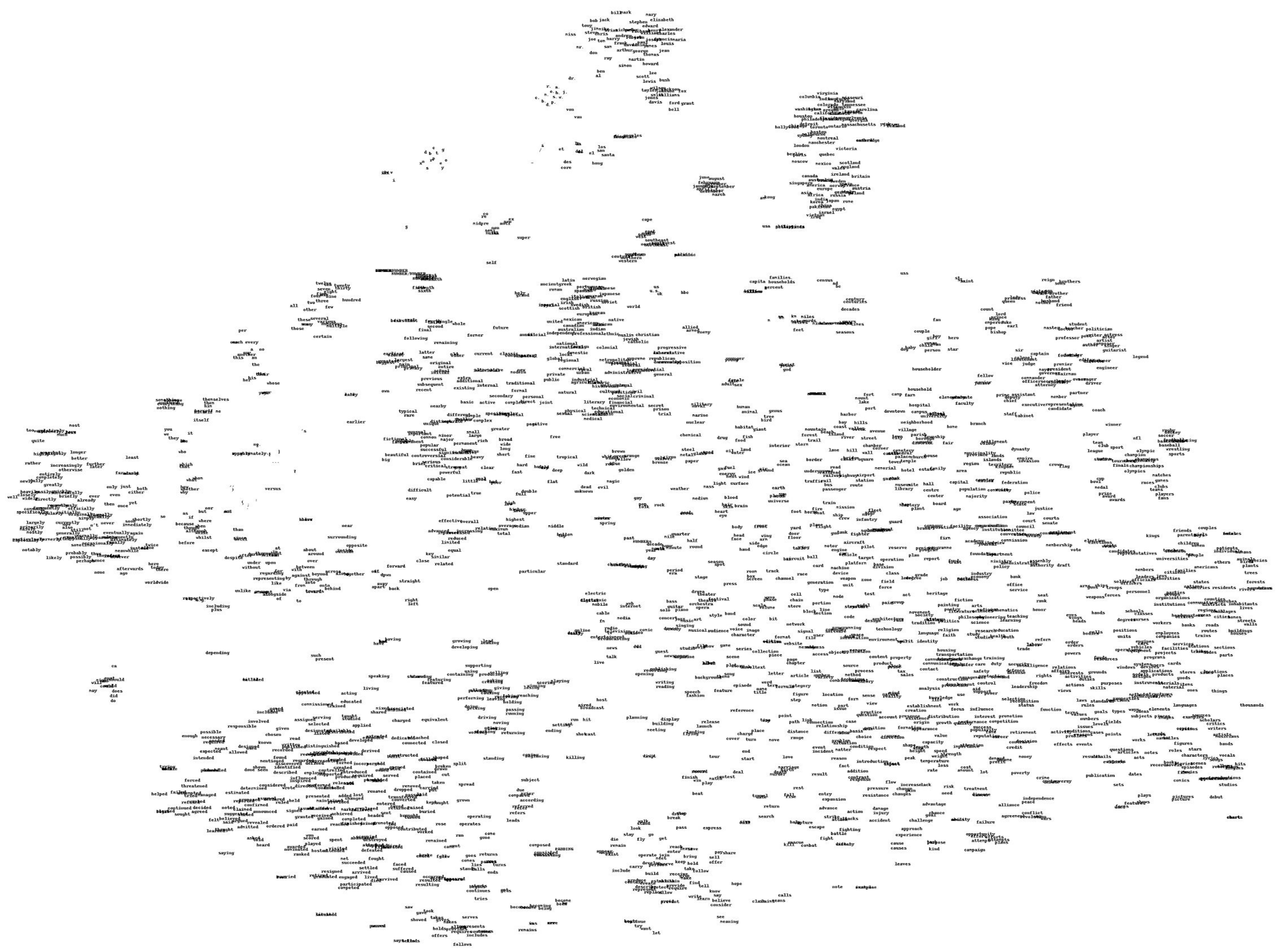
Word similarity



FRANCE 454	JESUS 1973	XBOX 6909	REDDISH 11724	SCRATCHED 29869	MEGABITS 87025
AUSTRIA	GOD	AMIGA	GREENISH	NAILED	OCTETS
BELGIUM	SATI	PLAYSTATION	BLUISH	SMASHED	MB/S
GERMANY	CHRIST	MSX	PINKISH	PUNCHED	BIT/S
ITALY	SATAN	IPOD	PURPLISH	POPPED	BAUD
GREECE	KALI	SEGA	BROWNISH	CRIMPED	CARATS
SWEDEN	INDRA	PSNUMBER	GREYISH	SCRAPED	KBIT/S
NORWAY	VISHNU	HD	GRAYISH	SCREWED	MEGAHERTZ
EUROPE	ANANDA	DREAMCAST	WHITISH	SECTIONED	MEGAPIXELS
HUNGARY	PARVATI	GEFORCE	SILVERY	SLASHED	GBIT/S
SWITZERLAND	GRACE	CAPCOM	YELLOWISH	RIPPED	AMPERES

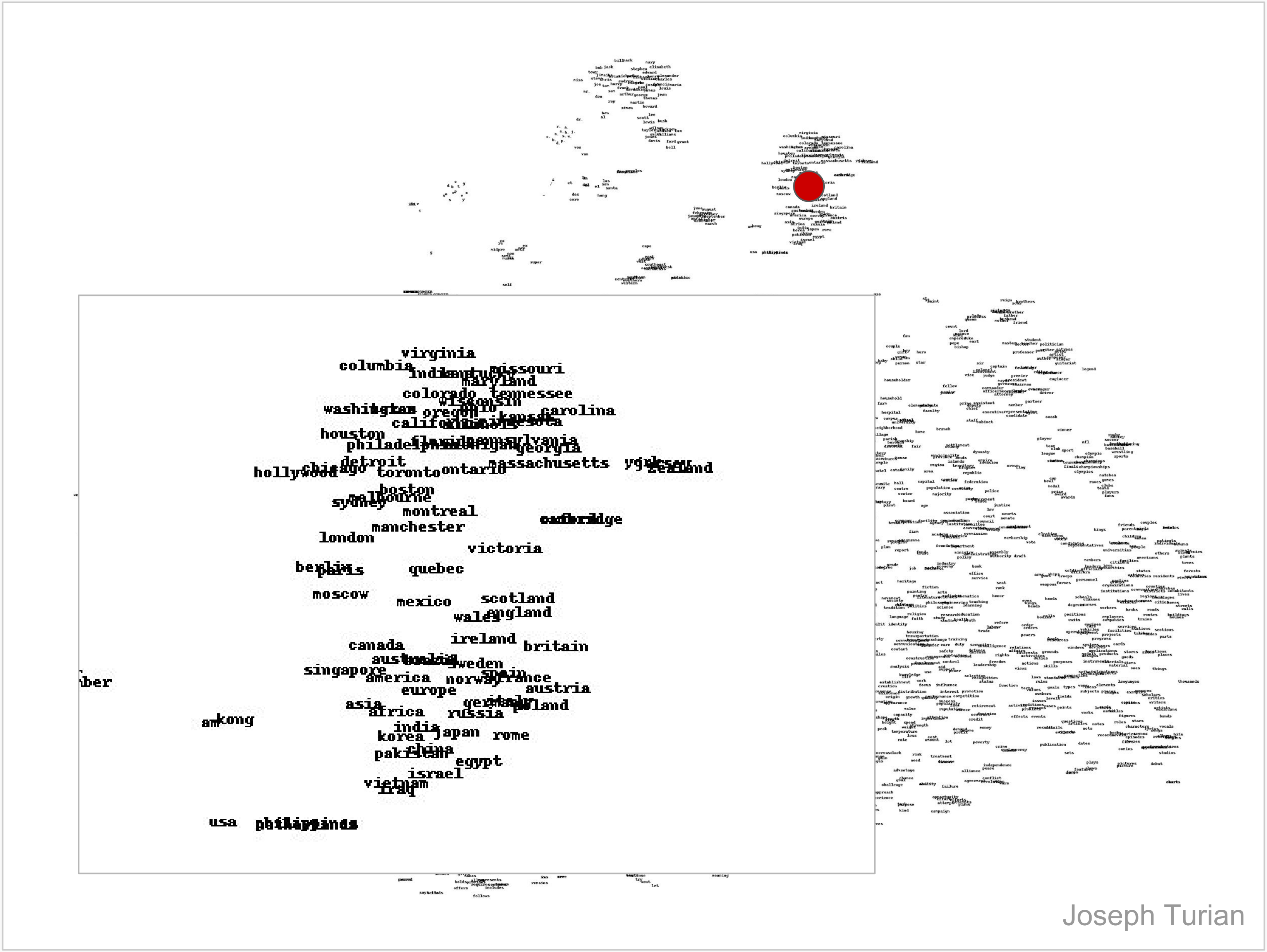
Collobert et. al. 2011. *Natural Language Processing (Almost) from Scratch.*

Word similarity



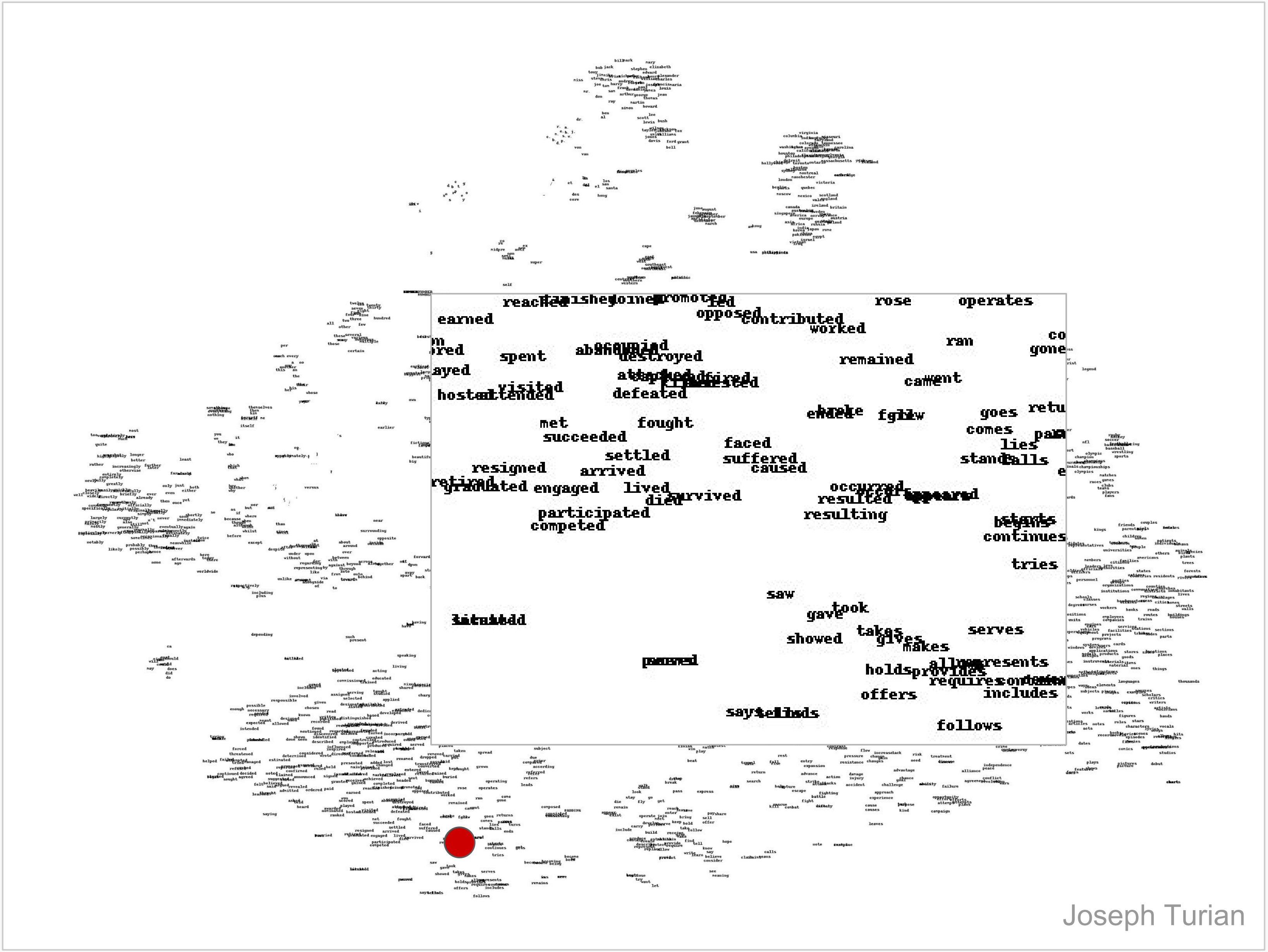
Joseph Turian

Word similarity



Joseph Turian

Word similarity



Joseph Turian

Analogy recovery

The task of **analogy recovery**. Questions in the form:

a is to *b* as *c* is to *d*

The system is given words *a, b, c*, and it needs to find *d*. For example:

‘apple’ is to ‘apples’ as ‘car’ is to ?

or

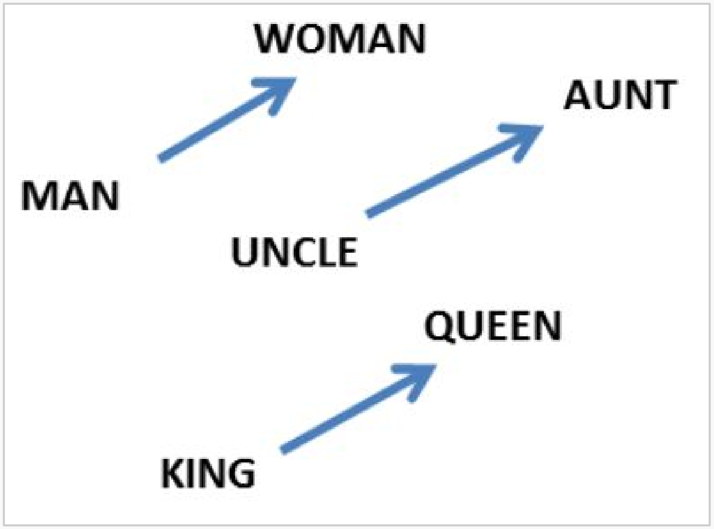
‘man’ is to ‘woman’ as ‘king’ is to ?

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

Analogy recovery

Task: a is to b as c is to d

Idea: The direction of the relation should remain the same.



$$a - b \approx c - d$$

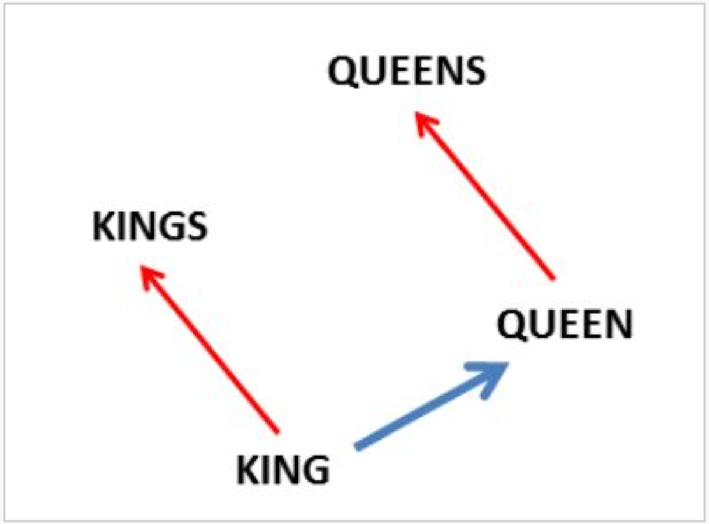
$$man - woman \approx king - queen$$

$$d_w = argmax_{d'_w \in V}(cos(a - b, c - d'))$$

Analogy recovery

Task: a is to b as c is to d

Idea: The offset of vectors should reflect their relation.



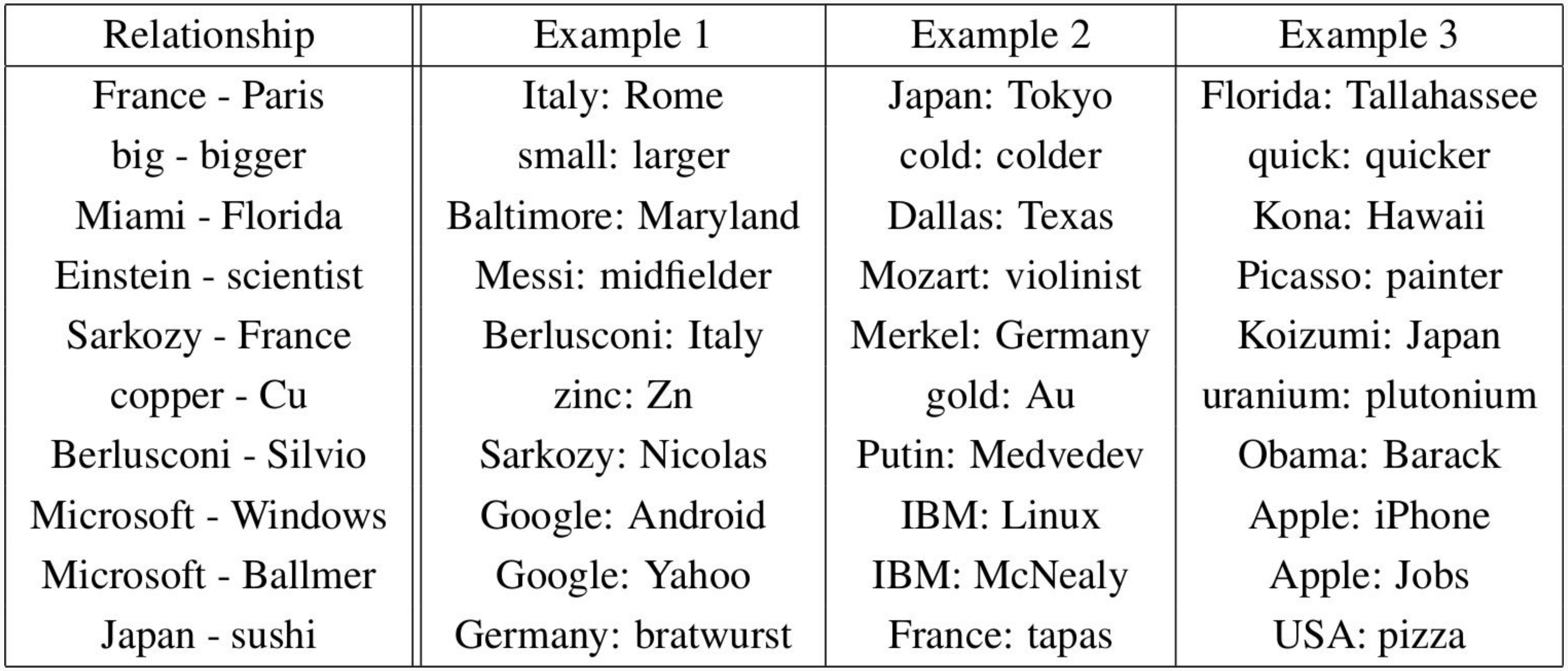
$$a - b \approx c - d$$

$$d \approx c - a + b$$

$$queen \approx king - man + woman$$

$$d_w = argmax_{d'_w \in V}(cos(d', c - a + b))$$

Analogy recovery



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

Example output using word2vec vectors.

Word embeddings in practice

Word2vec is often used for pretraining.

- It will help your models start from an **informed** position
- Requires only **plain text** - which we have a lot
- Is very **fast** and easy to use
- Already **pretrained** vectors also available (trained on 100B words)

However, for best performance it is important to continue training (fine-tuning).

Raw word2vec vectors are good for predicting the surrounding words, but not necessarily for your specific task.

Simply treat the embeddings the same as other parameters in your model and keep updating them during training.

Problems with word embeddings

Word embeddings allow us to learn similar representations for semantically or functionally similar words.

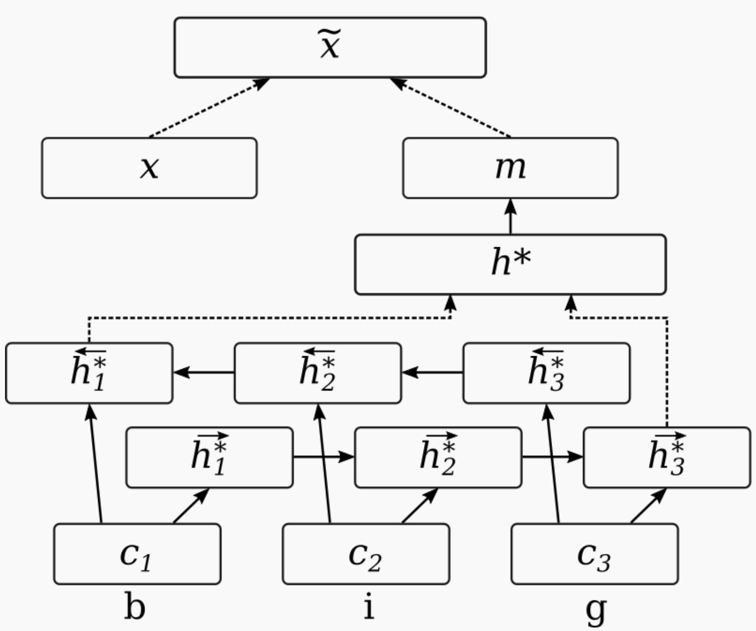
BUT

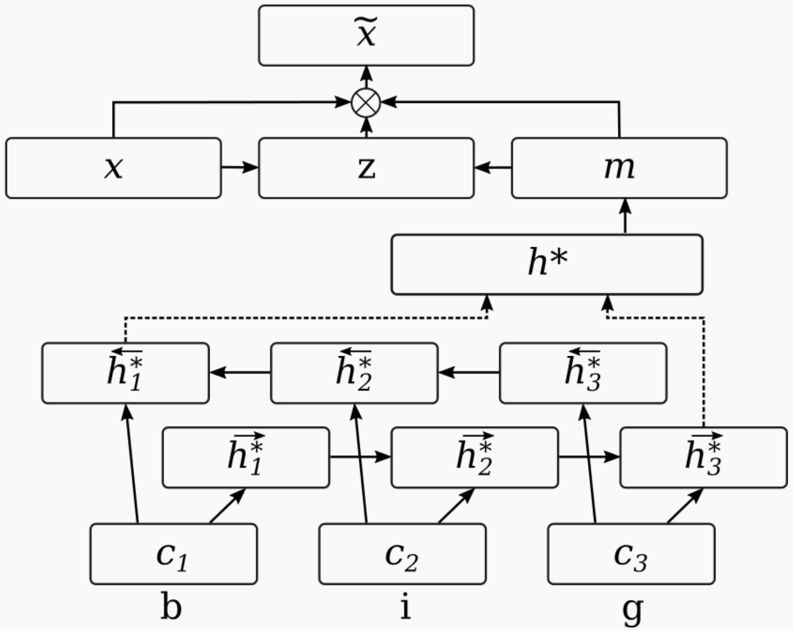
1. If a token has not been seen during training, we have to use a generic OOV (out-of-vocabulary) token to represent it.

2. Infrequent words have very low-quality embeddings, due to lack of data.

3. Morphological and character-level information is ignored when treating words as atomic units.

Character-based representations





Rei et al. (2016)

We can augment word embeddings by learning character-based representations.

Multimodal embeddings

We can map text and images into the same space

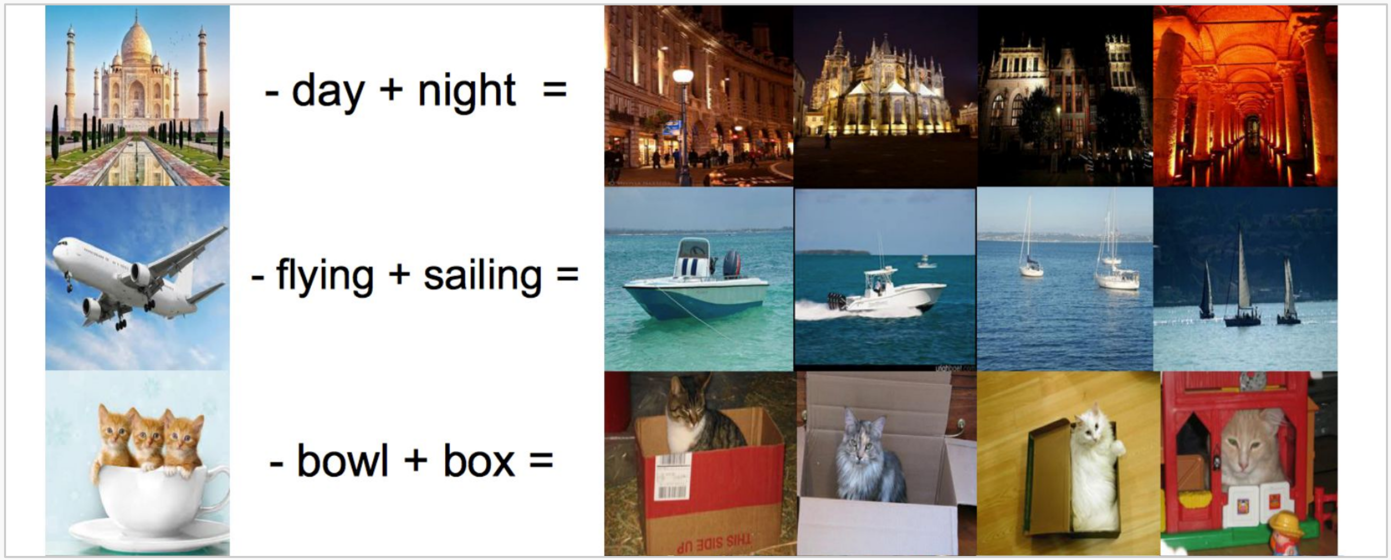
a kitchen with stainless steel appliances .

this is a herd of cattle out in the field .

a car is parked in the middle of nowhere .

a ferry boat on a marina with a group of people .

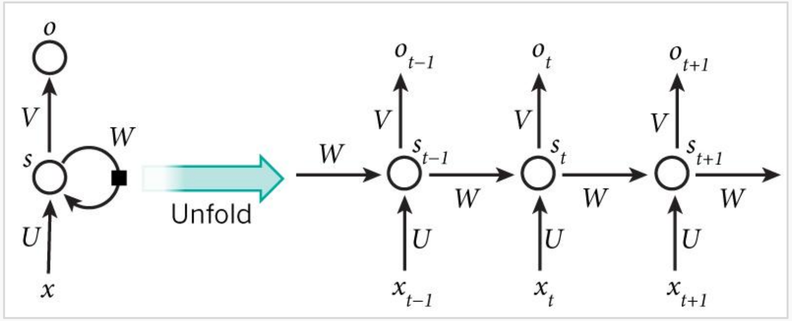
a little boy with a bunch of friends on the street .

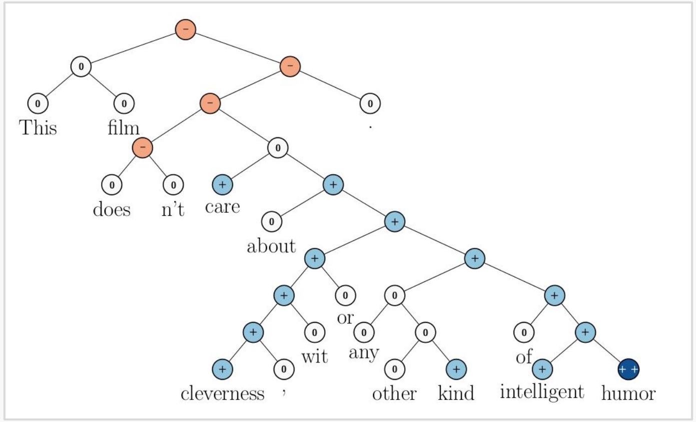


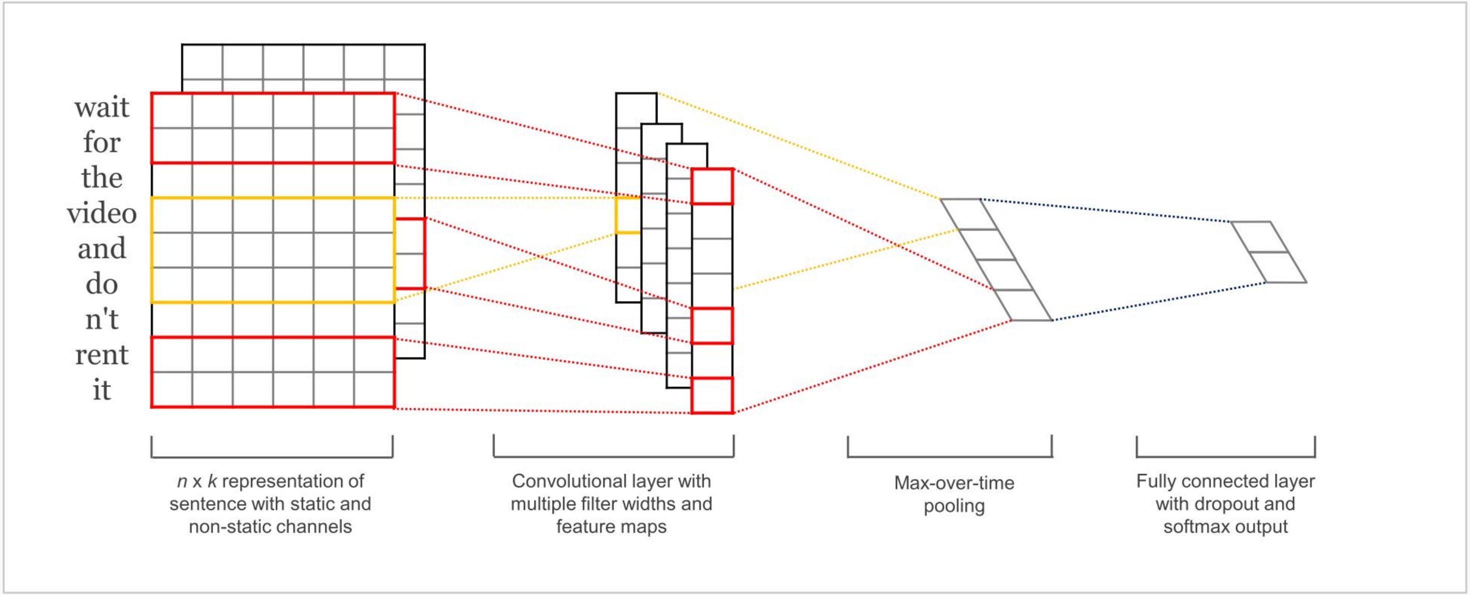
Kiros et al. (2014, 2015)

Conclusion

Word embeddings are the building blocks for higher-level models







wait
for
the
video
and
do
n't
rent
it
n x *k* representation of sentence with static and non-static channels
Convolutional layer with multiple filter widths and feature maps
Max-over-time pooling
Fully connected layer with dropout and softmax output

Questions?