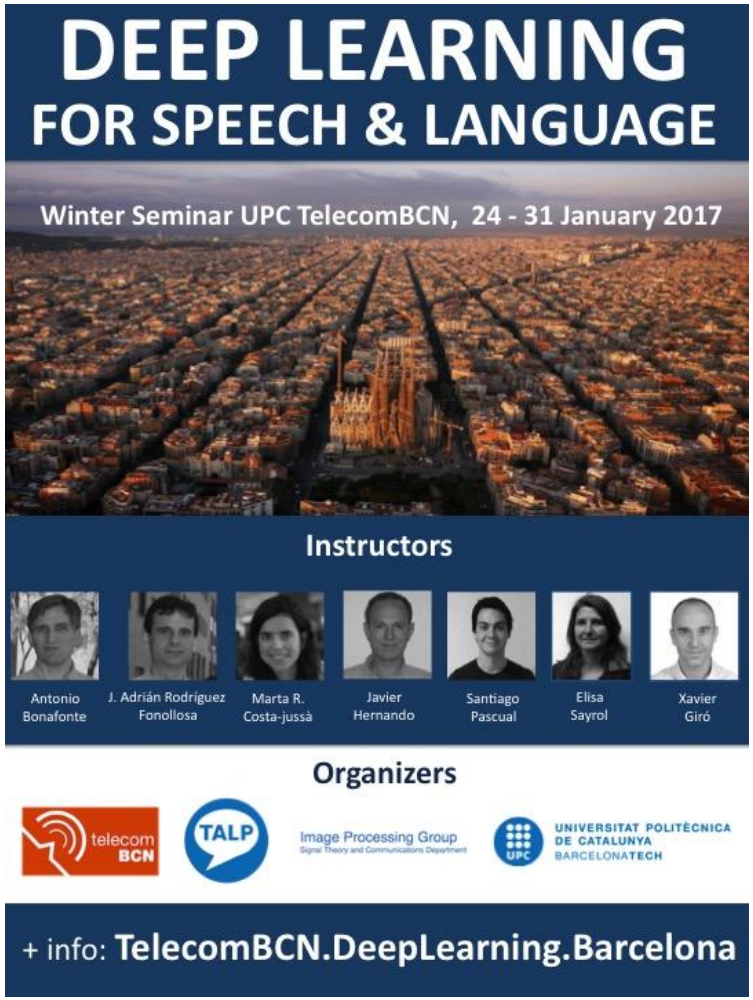


[course site]

Day 2 Lecture 4

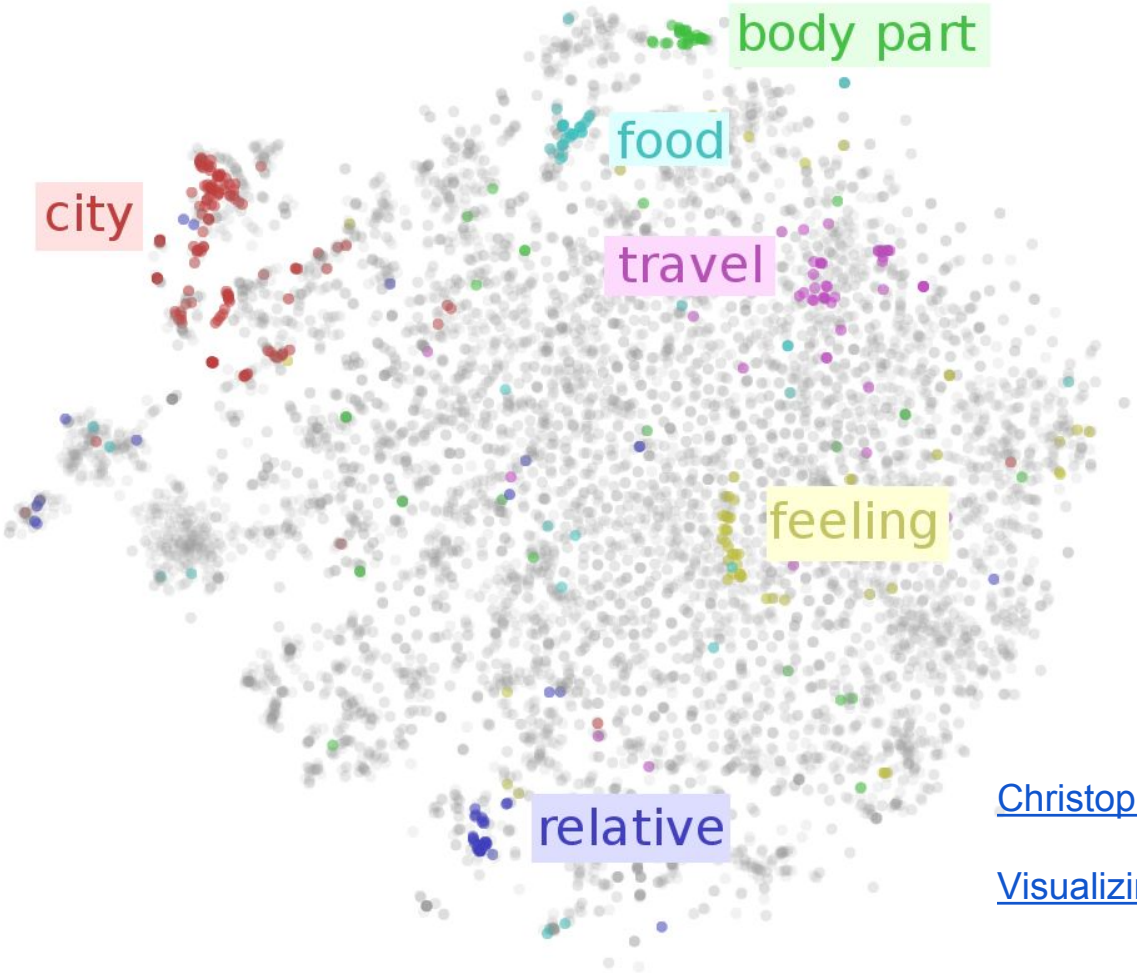
Word Embeddings

Word2Vec

Antonio Bonafonte

TALP UPC

telecom BCN



Christopher Olah

Visualizing Representations

Representation of categorical features

Variable which can take a limited number of possible values
E.g.: gender, blood types, countries, …, letters, words, phonemes

Newsgroup task:
Input: 1000 words (from a fixed vocabulary V, size |V|)

Phonetics transcription (CMU DICT):
Input: letters: (a b c…. z ‘ - .) (30 symbols)

One-hot (one-of-n) encoding

Example: letters. |V| = 30

‘a’: x^T = [1,0,0, ..., 0]
‘b’: x^T = [0,1,0, ..., 0]
‘c’: x^T = [0,0,1, ..., 0]
.
.
.

‘.’: x^T = [0,0,0, ..., 1]

One-hot (one-of-n) encoding

Example: words.

cat: x^T = [1,0,0, ..., 0]
dog: x^T = [0,1,0, ..., 0]
.
.
mamaguy: x^T = [0,0,0, …,0,1,0,...,0]
.
.
.

Number of words, |V| ?

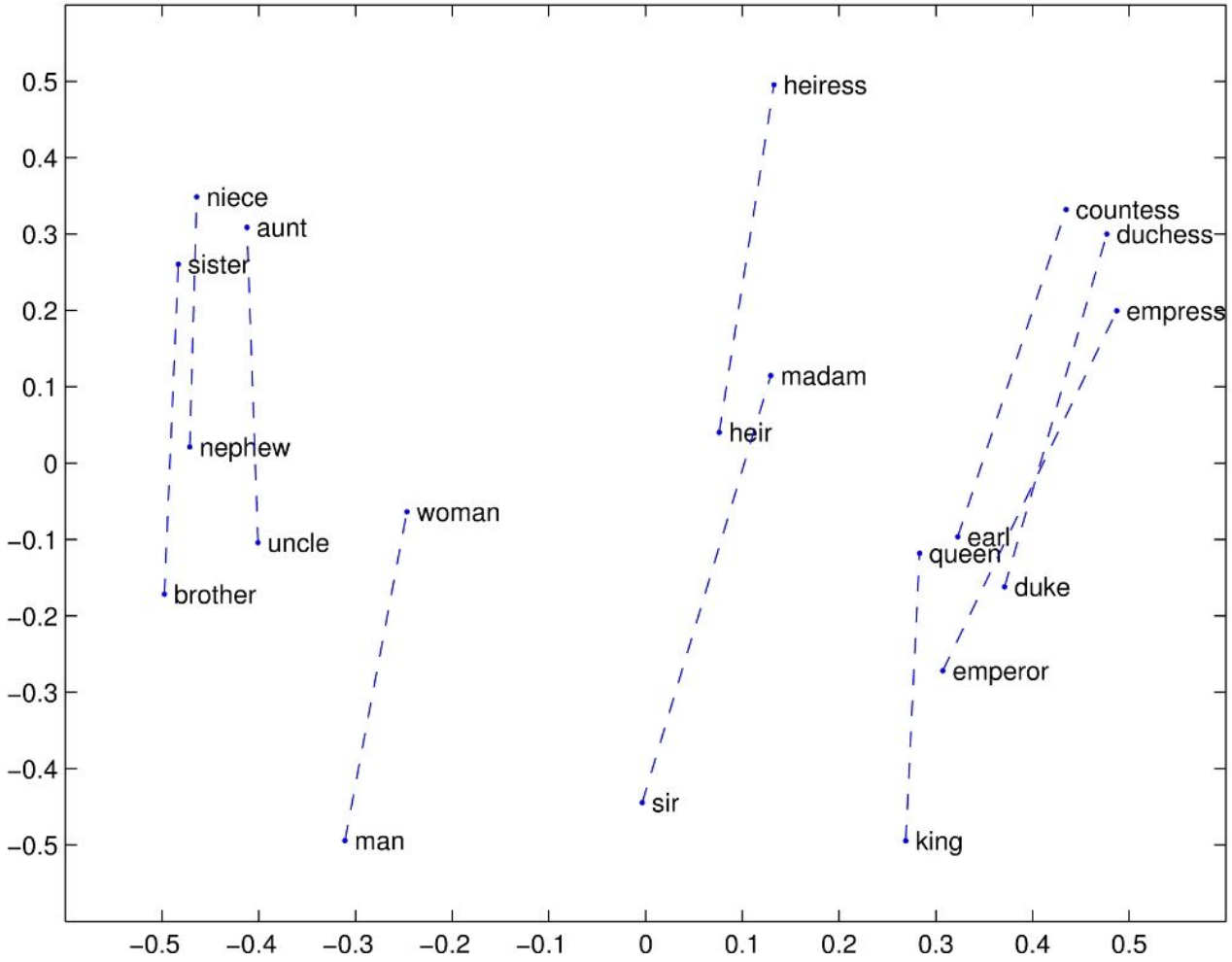
B2: 5K
C2: 18K
LVSR: 50-100K
Wikipedia (1.6B): 400K
Crawl data (42B): 2M

One-hot encoding of words: limitations

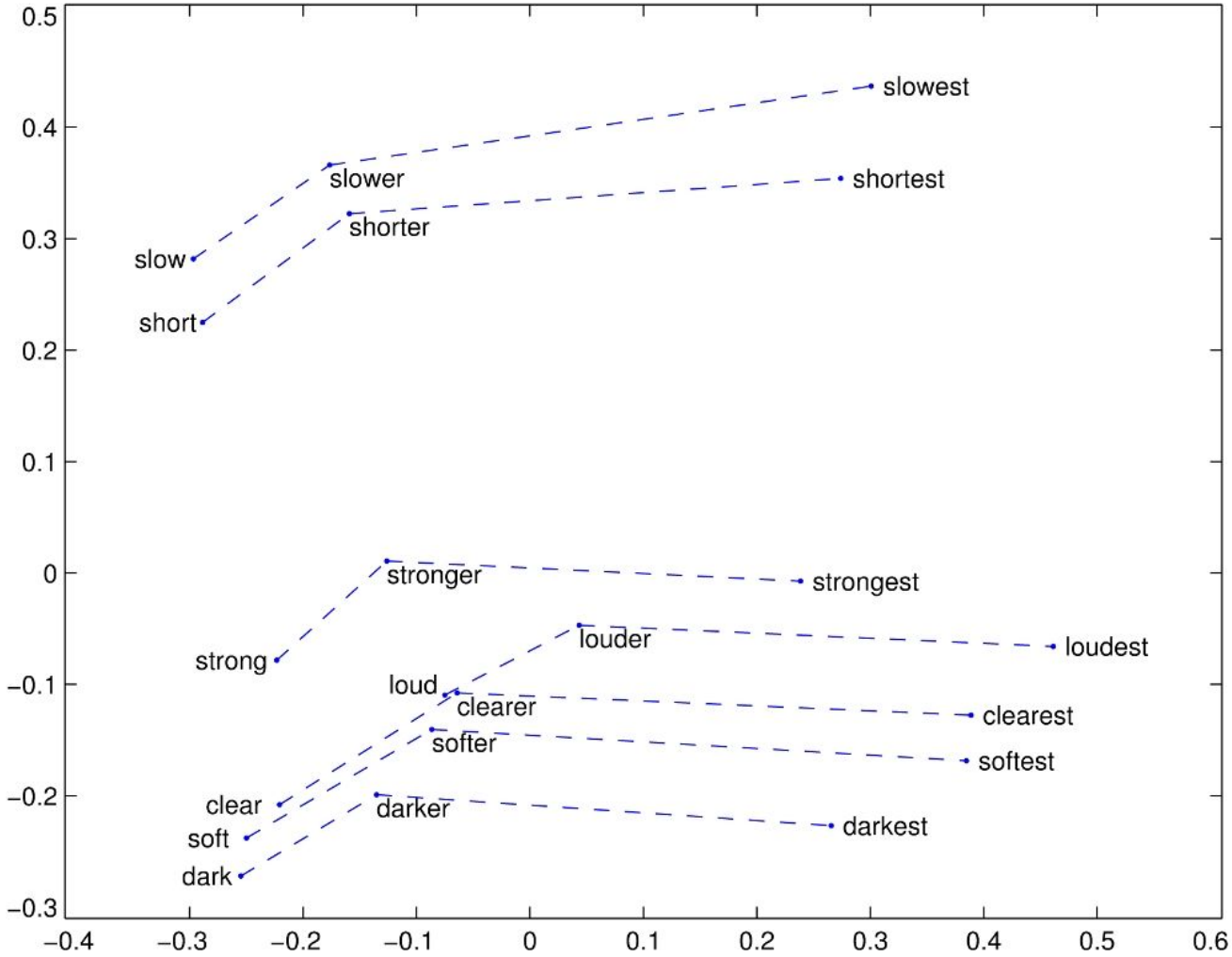
- Large dimensionality
- Sparse representation (mostly zeros)
- Blind representation
  - Only operators: ‘!=’ and ‘==’

Word embeddings

- Represent words using vectors of dimension *d* (~100 - 500)
- Meaningful (semantic, syntactic) distances
- Dominant research topic in last years in NLP conferences (EMNLP)
- *Good* embeddings are useful for *many* other tasks



GloVe (Stanford)



GloVe (Stanford)

Evaluation of the Representation

Word analogy:

a is to *b* as *c* is to ….

Find *d* such as w_d is closest to $w_a - w_b + w_c$

- Athens is to Greece as Berlin to ….
- Dance is to dancing as fly to ….

Word similarity:

Closest word to ...

How to define word representation?

> You shall know a word by the company it keeps.
> Firth, J. R. 1957

Some relevant examples:

Latent semantic analysis (LSA):

- Define co-ocurrence matrix of words w_i in documents w_j
- Apply SVD to reduce dimensionality

GloVe (Global Vectors):

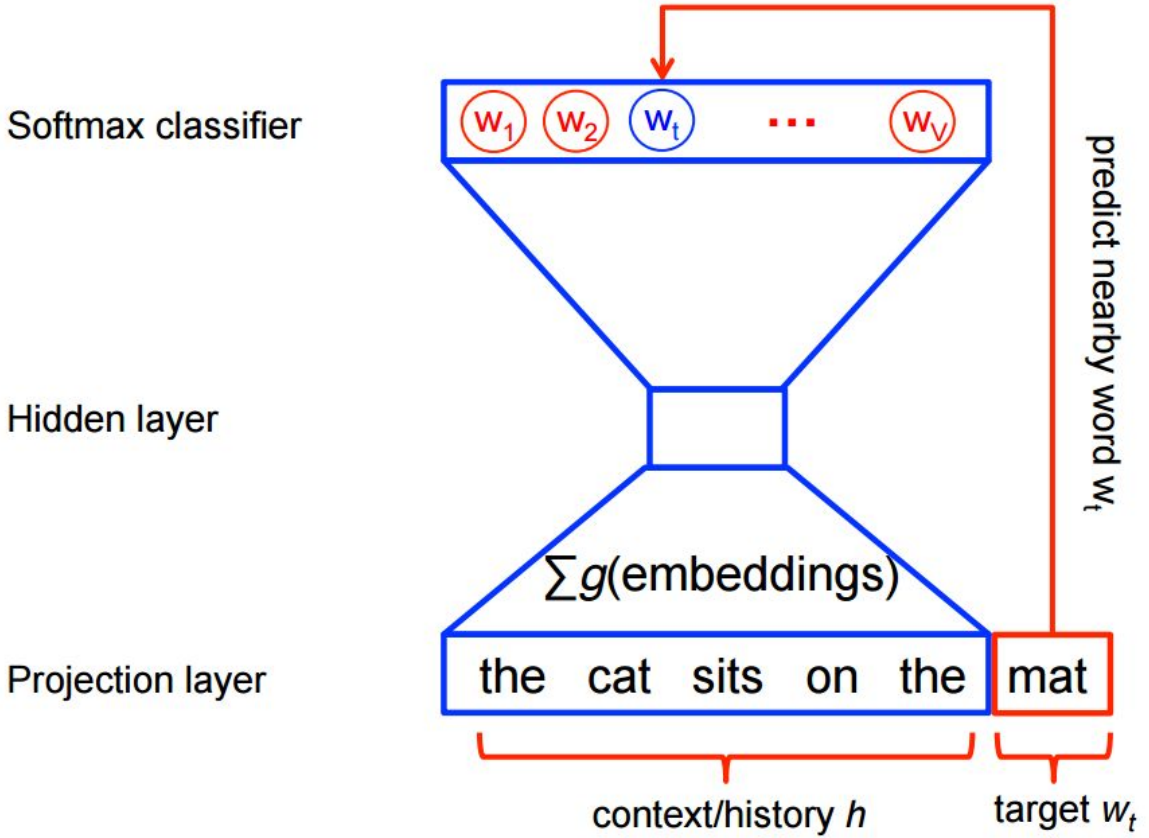
- Start with co-ocurrences of word w_i and w_j in context of w_i
- Fit log-bilinear regression model of the embeddings

Using NN to define embeddings

One-hot encoding + fully connected

→ embedding (projection) layer

Example:
Language Model (predict next word given previous words) produces word embeddings (Bengio 2003)



From tensorflow word2vec tutorial: https://www.tensorflow.org/tutorials/word2vec/

Toy example: predict next word

Corpus:

```
the dog saw a cat
the dog chased a cat
the cat climbed a tree
```

|V| = 8

One-hot encoding:

```
a:        [1,0,0,0,0,0,0,0]
cat:      [0,1,0,0,0,0,0,0]
chased:   [0,0,1,0,0,0,0,0]
climbed:  [0,0,0,1,0,0,0,0]
dog:      [0,0,0,0,1,0,0,0]
saw:      [0,0,0,0,0,1,0,0]
the:      [0,0,0,0,0,0,1,0]
tree:     [0,0,0,0,0,0,0,1]
```

Toy example: predict next word

Architecture:

Input Layer: $h_1(x) = W^I \cdot x$

Hidden Layer: $h_2(x) = g(W^H \cdot h^1(x))$

Output Layer: $z(x) = o(W^O \cdot h^2(x))$

Training sample:

cat → climbed

```
x = y = zeros(8,1)
x(2) = y(4) = 1
WI = rand(3,8) - 0.5;
WO = rand(8,3) - 0.5;
WH = rand(3,3);
h1 = WI * x
a2 = WH * h1            h2 = tanh(h2)
a3 = WO * h2
z3 = exp(a3)            z3 = h3/sum(z3)
```

Input Layer: $h_1(x) = W^I \cdot x$ (projection layer)

```
>> x'
ans =

   0   1   0   0   0   0   0   0

>> WI
WI =

  -0.0399348  -0.4823444   0.2111808  -0.0604554  -0.2916515   0.1254983   0.4865081   0.0495719
   0.2828681   0.3160292  -0.0206030  -0.4471545  -0.1573812  -0.0890795  -0.2779747   0.3646263
  -0.1617508   0.2179165  -0.1890760  -0.0692710  -0.2752995   0.2690168  -0.2928326  -0.0095162

>> h1 = WI * x
h1 =

  -0.48234
   0.31603
   0.21792
```

Note that non-lineal activation function in projection layer is irrelevant

Hidden Layer: $h_2(x) = g(W^H \cdot h_1(x))$

```
>> a2 = WH * h1;
>> h2 = tanh(a2)
h2 =

   0.250580
   0.020785
   0.120591
```

Softmax Layer: $z(x) = o(W^O \cdot h_2(x))$

```
>> a3 = WO * h2; a3'
ans =

  -0.0553628  -0.0997195  -0.0955033  -0.1315899  -0.0577650   0.0015831  -0.1180911  -0.0609728

>> z  = exp(a3);
>> z  = z/sum(z); z'
ans =

   0.12765   0.12212   0.12263   0.11828   0.12735   0.13513   0.11989   0.12694

>> y'
ans =

   0   0   0   1   0   0   0   0
```

Computational complexity

Example:

Num training data: 1B
Vocabulary size: 100K
Context: 3 previous words
Embeddings: 100
Hidden Layers: 300 units

Projection Layer: - (copy row)
Hidden Layer: 300 · 300 products, 300 *tanh(.)*
Softmax Layer: 100 · 100K products, 100K exp(.)

Total: 90K + 10M !!
The softmax is the network's main bottleneck.

Word embeddings: requirements

We can get embeddings implicitly from any task that involves words.

However, good generic embeddings are good for other tasks which may have much less data (transfer learning).

Sometimes, the embeddings can be fine-tuned to the final task.

Word embeddings: requirements

How to get good embedings:

- Very large lexicon
- Huge amount of learning data
- Unsupervised (or trivial labels)
- Computational efficient

Word2Vec [Mikolov 2013]

Architecture specific for producing embeddings.

It is learnt with huge amount of data.

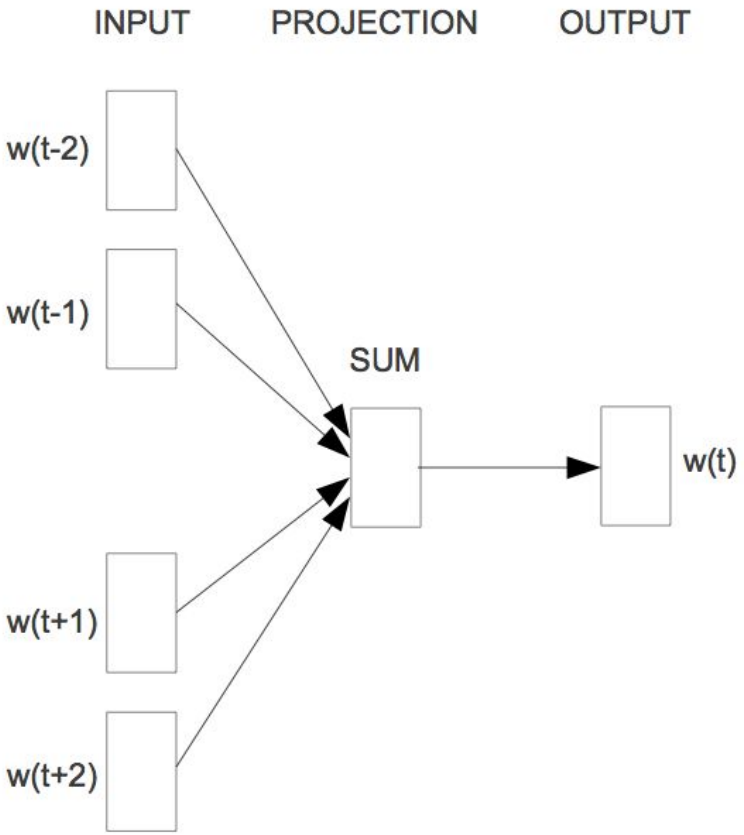
Simplify the architecture: remove hidden layer.

Simplify the cost (softmax)

Two variants:

- CBOW (continuos bag of words)
- Skip-gram

CBOW: Continuous Bag of Words



`the cat climbed a tree`

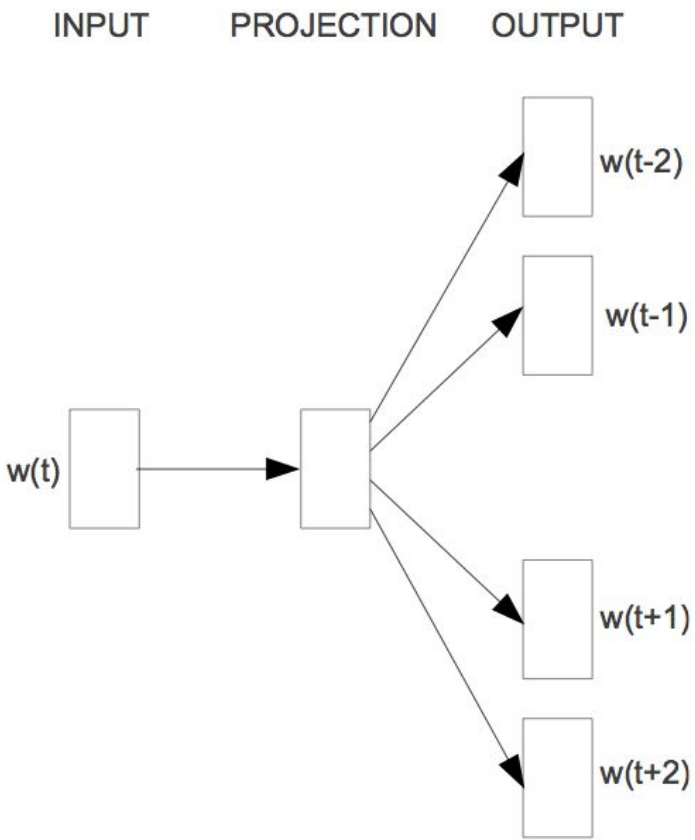
Given context:

`a, cat, the, tree`

Estimate prob. of

`climbed`

Skip-gram



`the cat climbed a tree`

Given word:

`climbed`

Estimate prob. of context words:

`a, cat, the, tree`

(It selects randomly the context length, till max of 10 left + 10 right)

Reduce cost: subsampling

Most frequent words (*is, the*) can appear hundred of millions of times.

Less important:

Paris ~ France OK

Paris ~ the ?

→ Each input word, w_i, is discarded with probability

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

Simplify cost: negative sampling

Softmax is required to get probabilities. But our goal here is just to get good embeddings.

Cost function: maximize $s(W^{O}_{j} \cdot W^{I}_{i})$

but add term negative for words which do not appear in the context (randomly selected).

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

Table 8: *Examples of the word pair relationships, using the best word vectors from Table 4 (Skip-gram model trained on 783M words with 300 dimensionality).*

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

Discussion

Word2Vec is not deep, but used in many tasks using deep learning

There are other approaches: this is a very popular toolkit, with trained embeddings, but there are others (GloVe).

Why does it works?

See paper from GloVe [Pennington et al]

References

Word2Vec

- Mikolov, Tomas; et al. "Efficient Estimation of Word Representations in Vector Space
- Linguistic Regularities in Continuous Space Word Representations
- Tensorflow tutorial https://www.tensorflow.org/tutorials/**word2vec**/

GloVe: http://nlp.stanford.edu/projects/glove/ (& paper)

Blog Sebastian Ruder
sebastianruder.com/word-embeddings-1/