[course site]

#DLUPC

Day 1 Lecture 4

Embeddings

Antonio Bonafonte
antonio.bonafonte@upc.edu

Associate Professor
Universitat Politècnica de Catalunya

Nominal Features

Real-value features

Most classifiers, including NN, are based on input features which are real numbers, or discrete numbers with a natural measure of numbers. E.g.: spectrum magnitude, pixel values, speed, age

Nominal or Categorical Features

Many problems are based on *nominal* features: discrete without natural order notion.

E.g.: sex, nationality, color, blood type, `new_user`, `bought_product-X`, etc.

This is particular relevant in language: phonemes, Part-of-Speech (Verb, Noun, etc.), and **words**.

Natural Language Processing

Natural language processing has a huge range of interesting applications:

Example

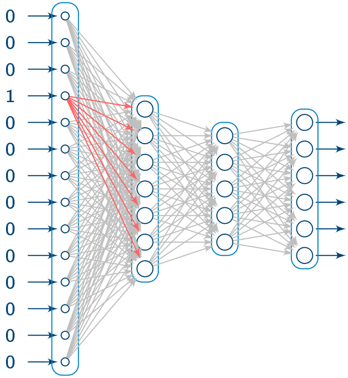
- Google search.
- 2008 U.S Presidential Elections: prediction of results from tweets (correct: 49/50).
- Machine translation.

Representation of *words* has a huge impact on a broad range of applications

One-hot (one-of-n) encoding

One-hot encoding of letters. $|V| = 30$

$$
\begin{array}{ll}
\texttt{'a'} & \mathbf{x}^T = (1, 0, 0, \ldots, 0) \\
\texttt{'b'} & \mathbf{x}^T = (0, 1, 0, \ldots, 0) \\
\texttt{'c'} & \mathbf{x}^T = (0, 0, 1, \ldots, 0) \\
\vdots & \\
\texttt{'.'} & \mathbf{x}^T = (0, 0, 0, \ldots, 1)
\end{array}
$$



One-hot encoding of words

$$
\begin{array}{ll}
\texttt{'cat'} & \mathbf{x}^T = (1, 0, 0, \ldots, 0) \\
\texttt{'dog'} & \mathbf{x}^T = (0, 1, 0, \ldots, 0) \\
\texttt{'mamaguy'} & \mathbf{x}^T = (0, 0, 1, \ldots, 0) \\
\vdots & \\
\texttt{'.'} & \mathbf{x}^T = (0, 0, 0, \ldots, 1)
\end{array}
$$

One-hot encoding of words

How Many Words ($|V|$)?

$$
\begin{array}{ll}
\texttt{'cat'} & \mathbf{x}^T = (1, 0, 0, \ldots, 0) \\
\texttt{'dog'} & \mathbf{x}^T = (0, 1, 0, \ldots, 0) \\
\texttt{'mamaguy'} & \mathbf{x}^T = (0, 0, 1, \ldots, 0) \\
\vdots & \\
\texttt{'.'} & \mathbf{x}^T = (0, 0, 0, \ldots, 1)
\end{array}
$$

One-hot encoding of words

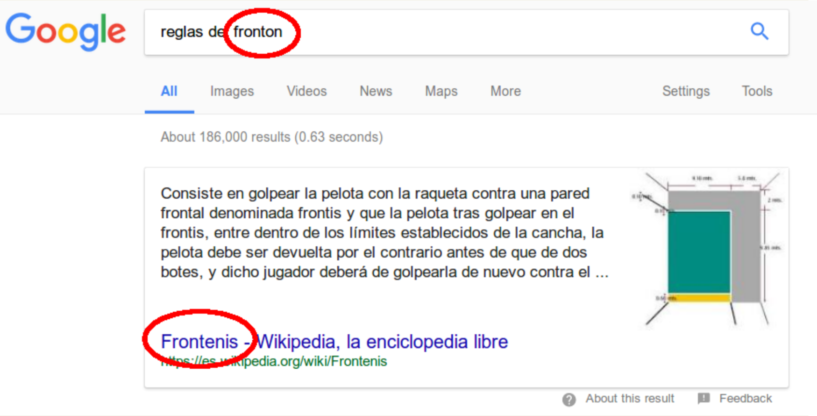
`'cat'`	$\mathbf{x}^T = (1, 0, 0, \ldots, 0)$
`'dog'`	$\mathbf{x}^T = (0, 1, 0, \ldots, 0)$
`'mamaguy'`	$\mathbf{x}^T = (0, 0, 1, \ldots, 0)$
$\vdots$	
`'.'`	$\mathbf{x}^T = (0, 0, 0, \ldots, 1)$

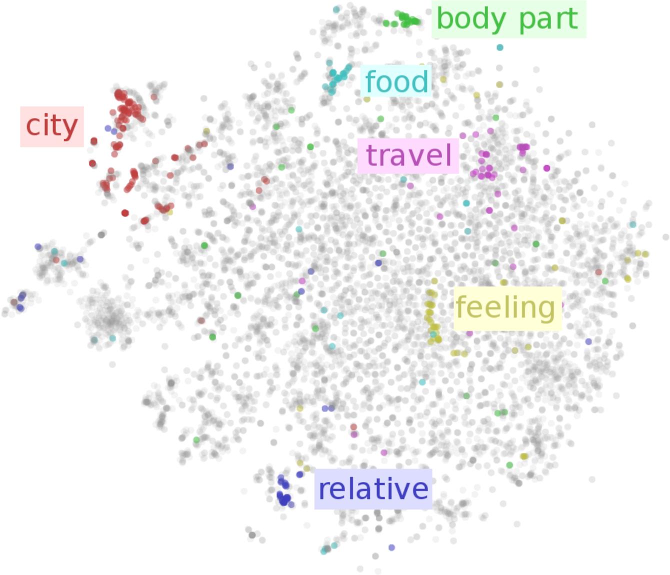
How Many Words ($|V|$)?

Foreign language B2:	5K
Foreign language C2:	18K
Speech Recognition:	50K – 100K
Wikipedia (1.6B):	400K
Crawl data (42B):	2M

- Large dimensionality, sparse representation
- Blind representation: only `equal` and `not_equal`

Example (Google search: frontón)

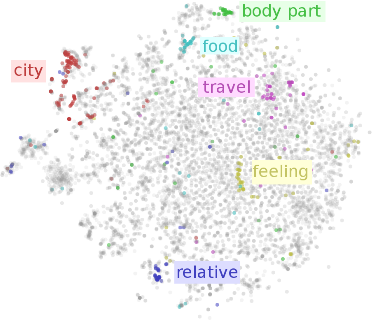


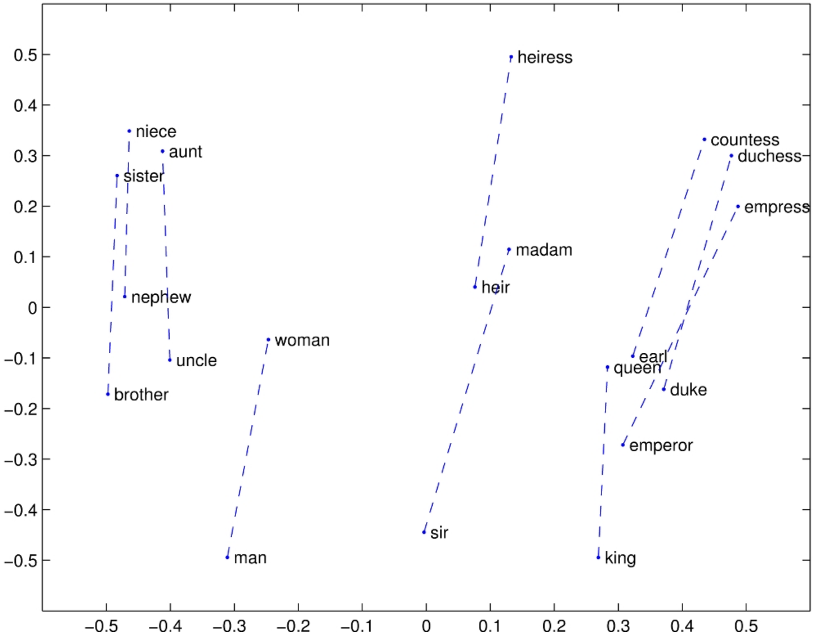


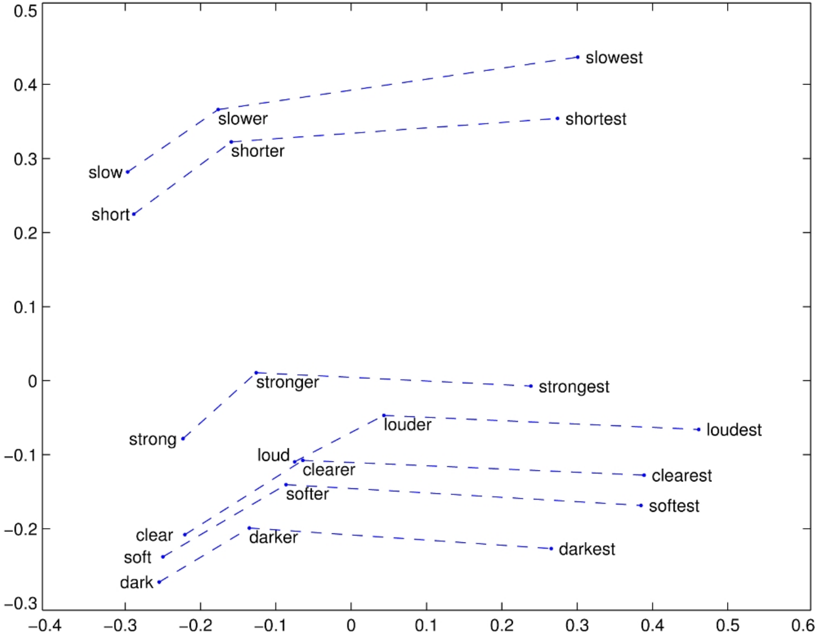
2D projection (t-sne) of word embeddings

Word Embeddings

- Represent words using vectors of dimension $d \approx [100 - 500]$
- Meaningful (semantic, syntactic) distances
- Dominant research topic in last years in NLP conferences (EMNLP)
- Transfer Learning: good embeddings are useful for many other tasks




heiress
niece
aunt
sister
countess
duchess
empress
madam
heir
nephew
woman
uncle
earl
queen
duke
brother
emperor
sir
man
king
0.5
0.4
0.3
0.2
0.1
0
−0.1
−0.2
−0.3
−0.4
−0.5
−0.5
−0.4
−0.3
−0.2
−0.1
0
0.1
0.2
0.3
0.4
0.5


slowest
slower
shortest
shorter
slow
short
stronger
strongest
louder
loudest
strong
loud
clearer
clearest
softer
softest
clear
darker
darkest
soft
dark
0.5
0.4
0.3
0.2
0.1
0
−0.1
−0.2
−0.3
−0.4
−0.3
−0.2
−0.1
0
0.1
0.2
0.3
0.4
0.5
0.6

Word Embeddings: evaluation

Word similarity

Closest word to w_a

$w_b = \text{argmin}_w \, \mathcal{D}(w, w_a)$

Word analogy

w_a is to w_b as $\bar{w}_a$ is to ?

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

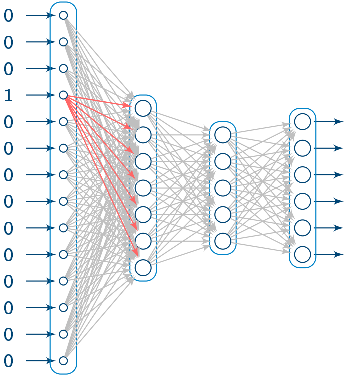
$$\bar{w}_b = \underset{w}{\text{argmin}} \, \mathcal{D}(w, w_b - w_a + \bar{w}_a)$$

Greece - Athens + Berlin → ?
dancing - dance + fly → ?

Projection Layer (or embedding layer)

The output of any neural network layer can be used as embeddings.

In particular:
one-hot encoding + fully connected layer = embedding layer



Example: Embedding of dimension M, vocabulary size $|V|$
The input is the n value: $x^T = (0, \dots, 1, \dots, 0)$

$$\mathbf{W} = (\mathbf{w}_1, \mathbf{w}_2, \dots, \mathbf{w}_{|V|}) = \begin{bmatrix} w_{11} & w_{12} & w_{13} & \dots & w_{1|V|} \\ w_{21} & w_{22} & w_{23} & \dots & w_{2|V|} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ w_{M1} & w_{M2} & w_{M3} & \dots & w_{M|V|} \end{bmatrix}$$

$$\mathbf{h} = \mathbf{W} \cdot \mathbf{x} = \begin{bmatrix} w_{11} & \dots & w_{1n} & \dots & w_{1|V|} \\ w_{21} & \dots & w_{2n} & \dots & w_{2|V|} \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ w_{M1} & \dots & w_{Mn} & \dots & w_{M|V|} \end{bmatrix} \cdot \begin{bmatrix} 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \end{bmatrix} = \begin{bmatrix} w_{1n} \\ w_{2n} \\ \vdots \\ w_{Mn} \end{bmatrix} = \mathbf{w}_n$$

(In *DL frameworks*, embedding layer is just one matrix and the column is selected according to the index.)

How to define word representation?

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

Some relevant techniques:

Latent semantic analysis (LSA)

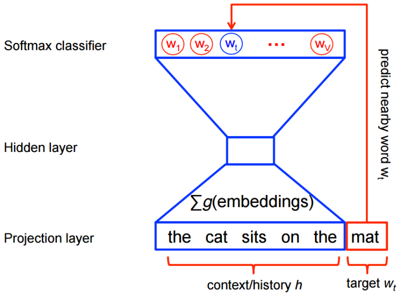
Define co-ocurrence matrix of words w_i in documents d_j
Apply SVD to reduce dimensionality

GloVe (Global Vectors)

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

Using NN to define word embeddings

First Idea: Get embedding from Language Model (predict next word given previous words) (Bengio 2003)



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

OK .. but computationally expensive!

Toy example: predict next word (bigram) I

(Just to see the operations and sizes)

Corpus

`the dog saw a cat`

`the dog chased a cat`

`the cat climbed a tree`

`a`	ID=1	$\mathbf{x}_1^T = (1,0,0,0,0,0,0,0)$
`cat`	ID=2	$\mathbf{x}_2^T = (0,1,0,0,0,0,0,0)$
`chased`	ID=3	$\mathbf{x}_3^T = (0,0,1,0,0,0,0,0)$
`climbed`	ID=4	$\mathbf{x}_4^T = (0,0,0,1,0,0,0,0)$
`dog`	ID=5	$\mathbf{x}_5^T = (0,0,0,0,1,0,0,0)$
`saw`	ID=6	$\mathbf{x}_6^T = (0,0,0,0,0,1,0,0)$
`the`	ID=7	$\mathbf{x}_7^T = (0,0,0,0,0,0,1,0)$
`tree`	ID=8	$\mathbf{x}_8^T = (0,0,0,0,0,0,0,1)$

Toy example: predict next word (bigram) II

Architecture

- Input layer: embedding size: 3; only last word
- Hidden layer: 3 units, tanh activation
- Output layer: softmax activation (8 outputs)

Forward: `cat` (ID=2) $\rightarrow$ `climbed` (ID=4)

```
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(a2);
a3 = WO * h2;
z3 = exp(a3);    z3 = z3/sum(z3);
```

Toy example: predict next word (bigram) III

Embedding Layer: `h1(x)= WI · x`

```
>> x'
ans =
   0    1    0    0    0    0    0    0

>> WI
WI =
  -0.051    0.439    0.006    0.055   -0.348   0.291    0.059   -0.482
  -0.488   -0.266    0.440    0.149   -0.257   0.245   -0.211    0.062
   0.247    0.010    0.311   -0.068    0.495  -0.218    0.220   -0.007

>> h1 = WI * x
h1 =
   0.439
  -0.266
   0.010
```

Toy example: predict next word (bigram) IV

Hidden Layer: `h2(x)= g(WH · h1(x))`

```
>> a2 = WH * h1
a2 =
  -0.078
  -0.101
   0.117

>> h2 = tanh(a2)
h2 =
  -0.078
  -0.101
   0.117
```

Toy example: predict next word (bigram) V

Softmax Layer: `z(x) = σ(WO · h2(x))`

```
>> a3 = WO * h2; a3'
ans =
   -0.0203  -0.0365  -0.0143   0.0037  -0.0245  -0.0276  -0.0161  0.0145

>> z = exp(a3);
>> z = z/sum(z); z'
ans =
   0.124  0.122  0.125  0.127  0.124  0.123  0.125  0.129

>> 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:	$(3 \times 100) \times 300$ products, 300 tanh(.)
Softmax Layer:	$300 \times 100K$ products, 100K exp(.)

Total: 90K + 30M

The softmax is the network's main bottleneck.

Word embeddings: requirements

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

Good embeddings

- **Very** large lexicon
- Huge amount of learning data
- Unsupervised (or trivial labels)
- Computational efficient
- Can be transfered to other tasks (transfer learning; w/o fine tuning)

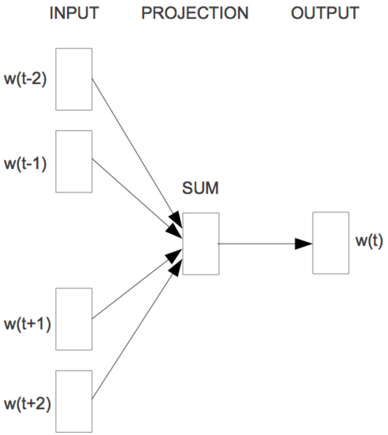
Word2Vec [Mikolov 2013]

- 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`

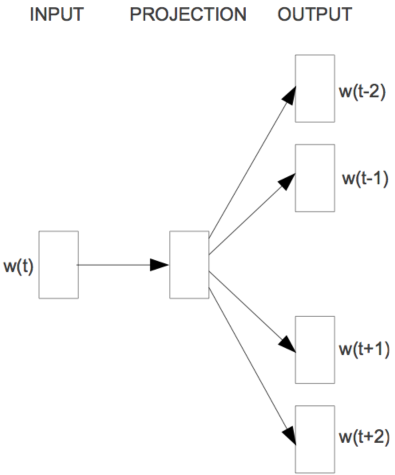
Given context:

`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`

The context length is selected randomly, till max of 10 left + 10 right

Word2Vec: efficiency I

subsampling

Most frequent words (*is*, *the*) can appear hundred of millions of times. They are less important:

- *Paris* – *France* $\rightarrow$ OK, interesting
- *Paris* – *the* $\rightarrow$ almost irrelevant

Each input word, w_i, is discarded with probability

$$p(w_i) = 1 - \sqrt{\frac{t}{freq(w_i)}}$$

Word2Vec: efficiency II

Negative sampling

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

- Maximize score: $\mathbf{w}_{Oj} \cdot \mathbf{w}_{Ik}$
- ... adding negative score for random terms not in the context.

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

Very efficient (100B words in one day):

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

Global Vectors for Word Representation (GloVe)

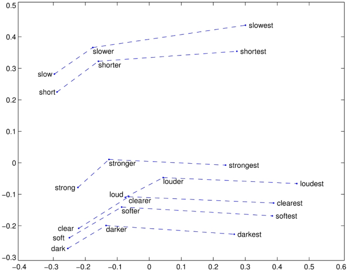
Trained on the non-zero entries of *word-coocurrence* matrix

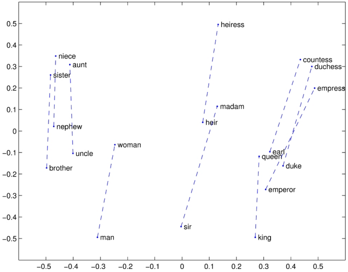
Toolkit & models available.

Training objective: learn vectors such that $\mathbf{v_i}^T \cdot \mathbf{v_j} = \log p(w_i, w_j)$

$$J = \sum_{i,j=1}^{V} f(X_{ij}) \cdot (\mathbf{v_i}^T \cdot \mathbf{v_j} + b_i + b_j - \log C_{ij})$$

The results of introduction are derived with GloVe





Representation of *embeddings*: t-sne

There are several techniques to reduce dimensionality and represent high dimension vector.

T-SNE: t-distributed Stochastic Neighbor Embedding. Iterative method that tries to minimize KL distance between original and reduced vectors.

Preserve local neighborhoods at the expense of global structure

Python package (tsne)
Tensor board (from tensor flow): continuous monitoring

[TensorBoard: embedding visualization]

[MNIST visualization]

[wikipedia paragraph visualization]

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]

It is still a hot research topic:

- Out-of-vocabulary, sublexical
- Paragraph embeddings
- Cross-lingual embeddings
- Task and domain specific