Text Analysis for Social Sciences in Python

Week 9: Text as Data – Word Embeddings

WINTER SEMESTER 21/22

HUYEN NGUYEN

HTTP://HUYENTTNGUYEN.COM

Last weeks…

Supervised Methods

Has anyone given a try yet at the bonus exercise?

…any questions with the materials?

Final presentation date

Monday 31.01: 14.00 – 18.00 (20-25’ presentation/group + 10’ Q&A)

(i.e. code & slides submission by Thursday 27.01 23.59 to Open Olat folder)

Today's agenda

Word Embeddings

- How do we represent meaning of a word?
- Word2vec
- Word vectors
- Optimization basics

Word meaning

Commonest linguistic way of thinking of meaning:

signifier (symbol) $\Leftrightarrow$ signified (idea or thing)

How do we have usable meaning in a computer?

Common solution: Use e.g. WordNet, a thesaurus containing lists of **synonym sets** and **hypernyms** ("is a" relationships).

e.g. synonym sets containing "good":

```
from nltk.corpus import wordnet as wn
poses = { 'n':'noun', 'v':'verb', 's':'adj (s)', 'a':'adj', 'r':'adv'}
for synset in wn.synsets("good"):
    print("{}: {}".format(poses[synset.pos()],
            ", ".join([l.name() for l in synset.lemmas()])))
```

```
noun: good
noun: good, goodness
noun: good, goodness
noun: commodity, trade_good, good
adj: good
adj (sat): full, good
adj: good
adj (sat): estimable, good, honorable, respectable
adj (sat): beneficial, good
adj (sat): good
adj (sat): good, just, upright
…
adverb: well, good
adverb: thoroughly, soundly, good
```

e.g. hypernyms of "panda":

```
from nltk.corpus import wordnet as wn
panda = wn.synset("panda.n.01")
hyper = lambda s: s.hypernyms()
list(panda.closure(hyper))
```

```
[Synset('procyonid.n.01'),
Synset('carnivore.n.01'),
Synset('placental.n.01'),
Synset('mammal.n.01'),
Synset('vertebrate.n.01'),
Synset('chordate.n.01'),
Synset('animal.n.01'),
Synset('organism.n.01'),
Synset('living_thing.n.01'),
Synset('whole.n.02'),
Synset('object.n.01'),
Synset('physical_entity.n.01'),
Synset('entity.n.01')]
```

Issues with resources like WordNet?

- Great as a resource but missing nuance
  e.g. “proficient” is listed as a synonym for “good”. This is only correct in some contexts.
- Missing new meanings of words
  e.g., wicked, badass, nifty, wizard, genius, ninja, bombest
- Impossible to keep up-to-date!
- Subjective
- Requires human labor to create and adapt
- Can’t compute accurate word similarity

Representing words as discrete symbols?

In traditional NLP, we regard words as discrete symbols: hotel, conference, motel – a localist representation

Words can be represented by ONE-HOT vectors:

$$hotel = [0\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ 0]$$

$$motel = [0\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0]$$

Vector dimension = number of words in vocabulary (e.g., 500,000)

Problem with words as discrete symbols?

- Example: in web search, if user searches for “Seattle motel”, we would like to match documents containing “Seattle hotel”.
- But:

$$hotel = [0\ 0\ 0\ 0\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ 0]$$
$$motel = [0\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0]$$

- These two vectors are orthogonal. There is no natural notion of similarity for one-hot vectors!
- Solution?
- Could try to rely on WordNet’s list of synonyms to get similarity?
- But it is well-known to fail badly: incompleteness, etc.
- Instead: learn to encode similarity in the vectors themselves

Representing words by their context?

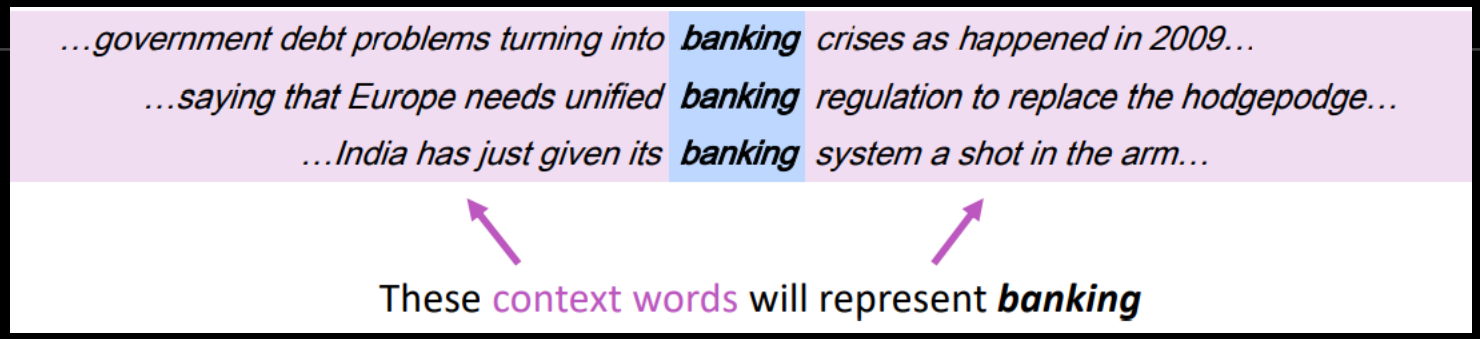
- Distributional semantics: A word's meaning is given by the words that frequently appear close-by

"You shall know a word by the company it keeps"

(J. R. Firth 1957: 11)

- One of the most successful ideas of modern statistical NLP:
- When a word w appears in a text, its context is the set of words that appear nearby (within a fixed-size window).
- Use the many contexts of w to build up a representation of w

Word vectors



We will build a dense vector for each word, chosen so that it is similar to vectors of words that appear in similar contexts.

Note: word vectors are sometimes called word embeddings or word representations. They are a distributed representation.

Word meaning as a neural word vector – visualization

$$\textit{expect} = \begin{pmatrix} 0.286 \\ 0.792 \\ -0.177 \\ -0.107 \\ 0.109 \\ -0.542 \\ 0.349 \\ 0.271 \\ 0.487 \end{pmatrix}$$

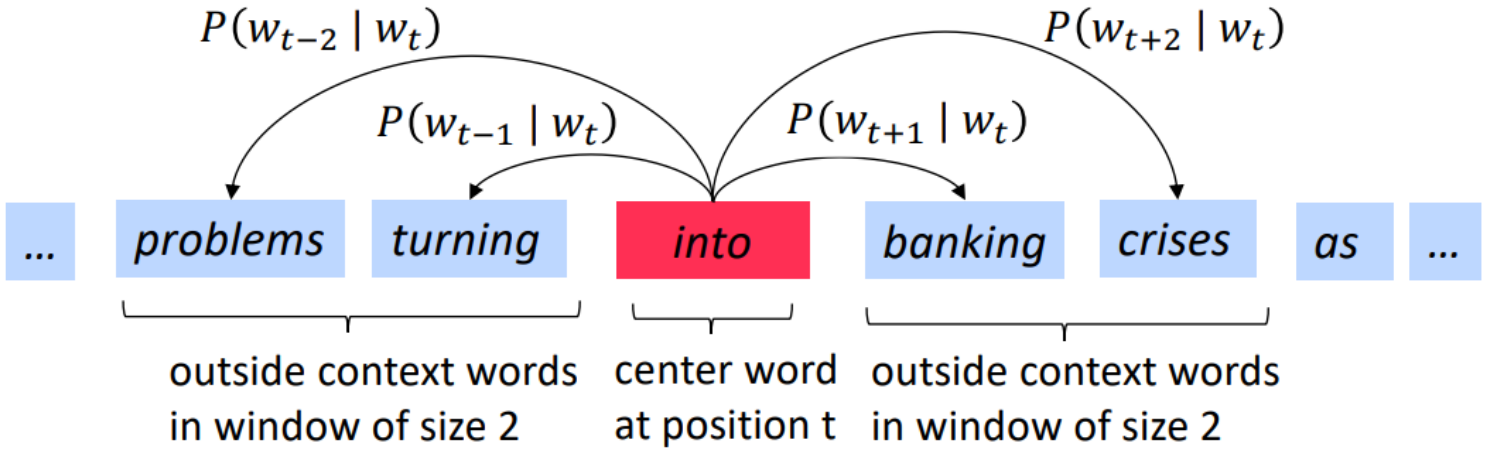
Word2vec: key idea

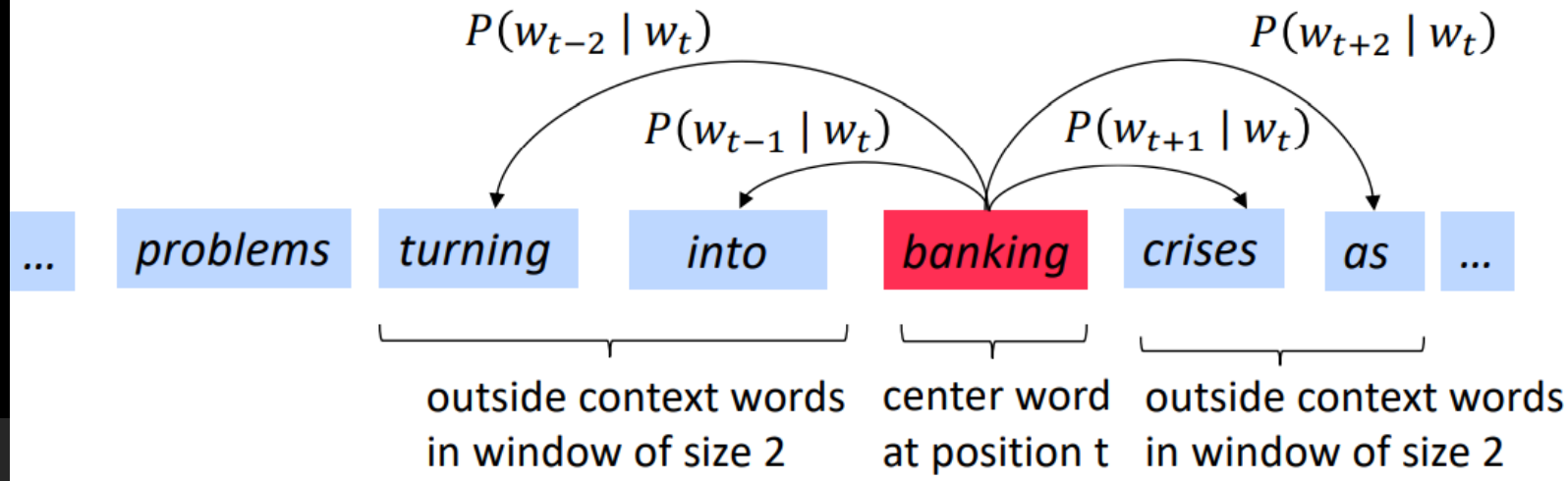
Word2vec (Mikolov et al. 2013) is a framework for learning word vectors.

Idea: With a large corpus of text....

- Every word in a fixed vocabulary is represented by a vector
- Go through each position t in the text, which has a center word c and context ("outside") words o
- Use the similarity of the word vectors for c and o to calculate the probability of o given c (or vice versa)
- Keep adjusting the word vectors to maximize this probability

Example windows and process for computing $P(w_{t+j} \mid w_t)$





Word2vec objective function

We want to minimize the objective function:

$$J(\theta) = -\frac{1}{T}\sum_{t=1}^{T}\sum_{\substack{-m\leq j\leq m \\ j\neq 0}} \log P\left(w_{t+j} \mid w_t; \theta\right)$$

Question: How to calculate $P\left(w_{t+j} \mid w_t; \theta\right)$?

Answer: We will *use two* vectors per word *w*:

- v_w when *w* is a center word
- u_w when *w* is a context word

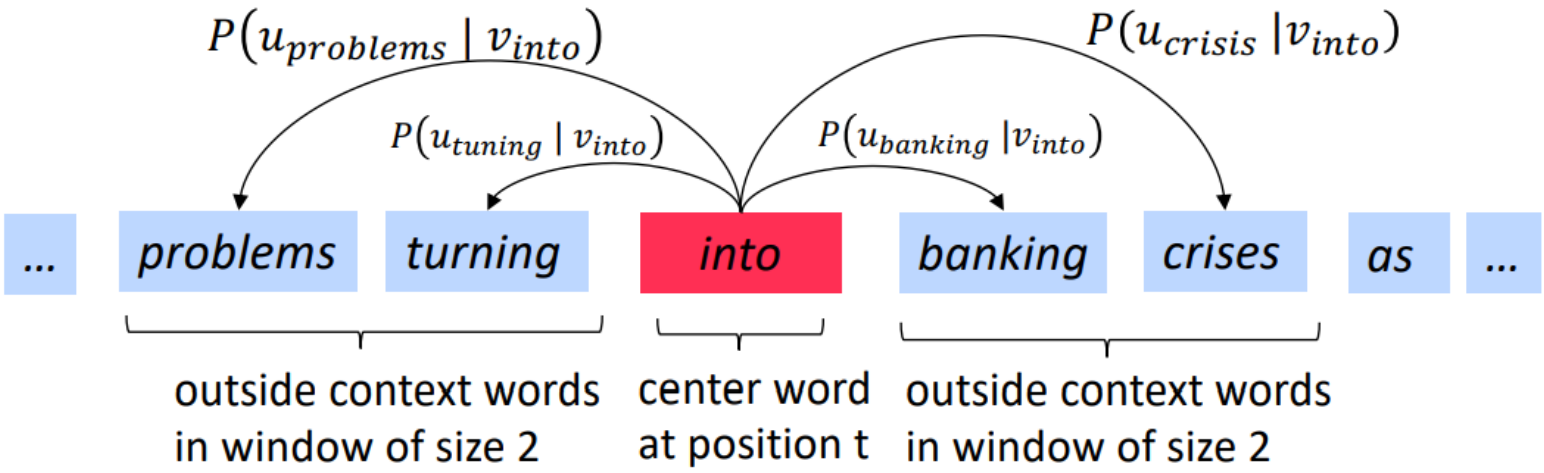
Then for a center word *c* and a context word *o*:

$$P(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{w\in V}\exp(u_w^T v_c)}$$

Word2vec overview with vectors

Example windows and process for computing $P(w_{t+j} \mid w_t)$

$P(u_{problems} \mid v_{into})$ short for $P(problems \mid into\ ; u_{problems}, v_{into}, \theta)$



Word2vec prediction function

Exponentiation makes anything positive

$$P(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{w \in V} \exp(u_w^T v_c)}$$

Dot product compares similarity of *o* and *c*.
$u^T v = u.v = \sum_{i=1}^{n} u_i v_i$
Larger dot product = larger probability

Normalize over entire vocabulary to give probability distribution

- This is an example of the **softmax function** $\mathbb{R}^n \to \mathbb{R}^n$

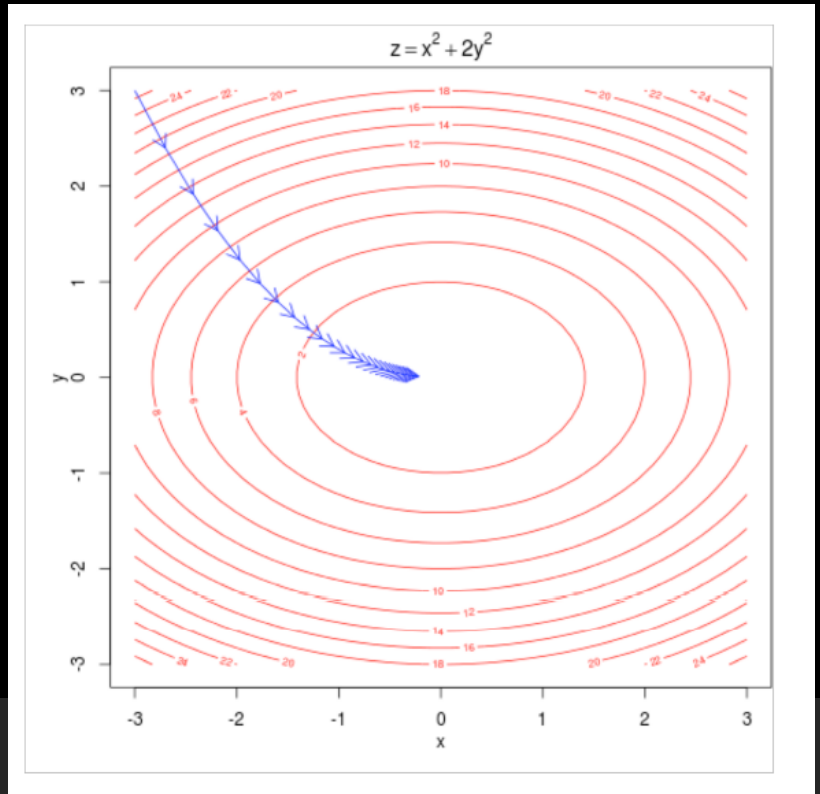
$$\text{softmax}(x_i) = \frac{\exp(x_i)}{\sum_{j=1}^{n} \exp(x_j)} = p_i$$

- The softmax function maps arbitrary values x_i to a probability distribution p_i
  - "max" because amplifies probability of largest x_i
  - "soft" because still assigns some probability to smaller x_i
  - Frequently used in Deep Learning

Model training by optimizing parameters

To train a model, we adjust parameters to minimize a loss

- E.g., below, for a simple convex function over two parameters Contour lines show levels of objective function.

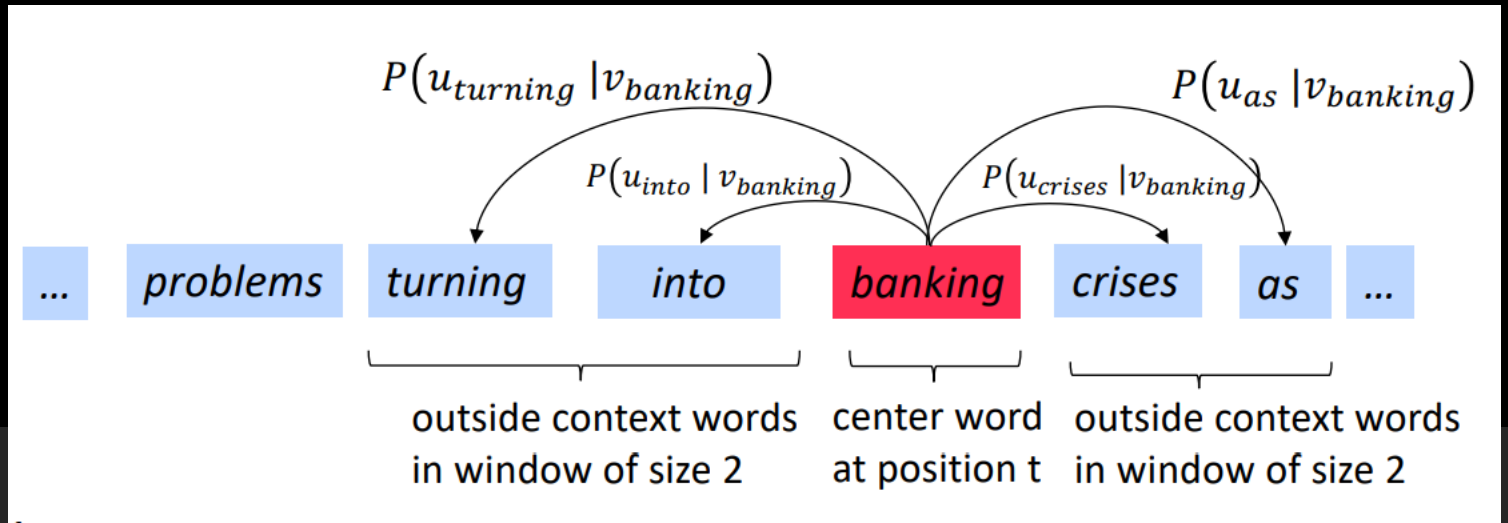


Model training: Compute ALL vector gradients

We represent all model parameters, in one long vector. In our case, with d-dimensional vectors and V-many words.

We go through gradient for each center vector v in a window

Generally, in each window we will compute updates for all parameters that are being used in that window. For example:



Word2vec: More details

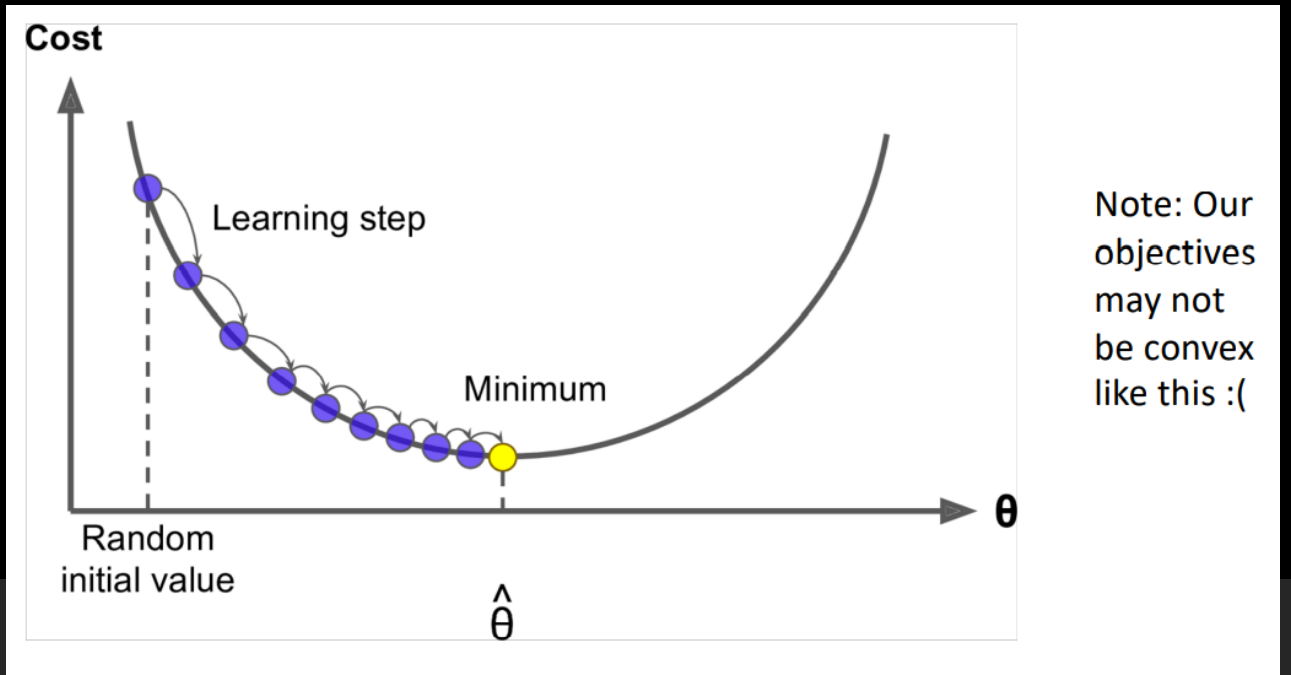
- Why two vectors? → Easier optimization. Average both at the end.
- Two model variants:
  - Skip-grams (SG): Predict context (”outside”) words (position independent) given center word
  - Continuous Bag of Words (CBOW): Predict center word from (bag of) context word

Optimization: Gradient Descent

We have a cost function $J(\theta)$ we want to minimize.

Gradient Descent is an algorithm to minimize $J(\theta)$.

Idea: for current value of θ, calculate gradient of $J(\theta)$, then take small step in direction of negative gradient. Repeat.



Note: Our objectives may not be convex like this :(

Practice time ☺

Open your Google Colab/ Jupyter Notebook

Practice time ☺

https://github.com/httn21uhh/Text-Analysis-for-Social-Sciences-in-Python

→ W9_word_doc_embeddings.ipynb + death-penalty-cases.csv + vocab.pkl + utils.py

- Download them
- Run the files on your own laptop
- The iPython file is NOT meant for passive scrolling!

This week…

- Word & doc embeddings in action
- Follow closely the illustrated examples and replicate yourself as the session proceeds.
- Unmute yourself to ask questions at any point, including when you think things go too fast/slow for you.