

Unit 3: Word embeddings

Goal today:

- Understand word embeddings:
 - Theoretical introduction
 - Training a model with Gensim
 - Using a model eg. for sentence similarity
- **Use case 1:** digital humanities, knowledge extraction
- **Use case 2:** Linked Data, natural language interfaces to Wikidata / ontodia

Motivation

Word embeddings are used:

- Stand-alone, eg. for word similarity, relation extraction
- In **ML/DL models**
 - WE usually the input to (NLP) **deep learning systems** as numeric word representations
 - Used in many tasks ...

Resources / Intro

- **Great introduction for the understanding of embeddings:**
<http://jalammarm.github.io/illustrated-word2vec/>
- Use the illustrations in the link above to explain basic ideas on why to use embeddings and how

Introduction to word embeddings

Agenda

- language modeling
- limitations of traditional n-gram language models
- Bengio et al. (2003)'s NNLM
- Google's word2vec (Mikolov et al. 2013)

Antoine Tixier, [DaSciM team](#), LIX
November 2015

Language model

- Goal: determine $P(s = w_1 \dots w_k)$ in some domain of interest

$$P(s) = \prod_{i=1}^k P(w_i \mid w_1 \dots w_{i-1})$$

$$\text{e.g., } P(w_1 w_2 w_3) = P(w_1) P(w_2 \mid w_1) P(w_3 \mid w_1 w_2)$$

- Traditional n-gram language model assumption:
“the probability of a word depends only on **context** of $n - 1$ previous words”

$$\Rightarrow \hat{P}(s) = \prod_{i=1}^k P(w_i \mid w_{i-n+1} \dots w_{i-1})$$

- Typical ML-smoothing learning process (e.g., Katz 1987):
 1. compute $\hat{P}(w_i \mid w_{i-n+1} \dots w_{i-1}) = \frac{\#w_{i-n+1} \dots w_{i-1} w_i}{\#w_{i-n+1} \dots w_{i-1}}$ on training corpus
 2. smooth to avoid zero probabilities

- How could we create a language model? Ideas?
- Explain the basic idea of neural networks and why word embeddings are better as input than one-hot encodings.

N-Grams – motivation

books.google.com/ngrams

Traditional n-gram language model

Limitation 1): curse of dimensionality

- Example
 - train a 10-gram LM on a corpus of 100.000 unique words
 - space: 10-dimensional hypercube where each dimension has 100.000 slots
 - model training \leftrightarrow assigning a probability to each of the 100.000^{10} slots
 - **probability mass vanishes** \rightarrow more data is needed to fill the huge space
 - the more data, the more unique words! \rightarrow vicious circle
 - what about corpuses of 10^6 unique words?
- \rightarrow in practice, contexts are typically limited to size 2 (trigram model)
e.g., famous Katz (1987) smoothed trigram model
- \rightarrow such short context length is a limitation: a lot of information is not captured

Traditional n-gram language model

Limitation 2): word similarity ignorance

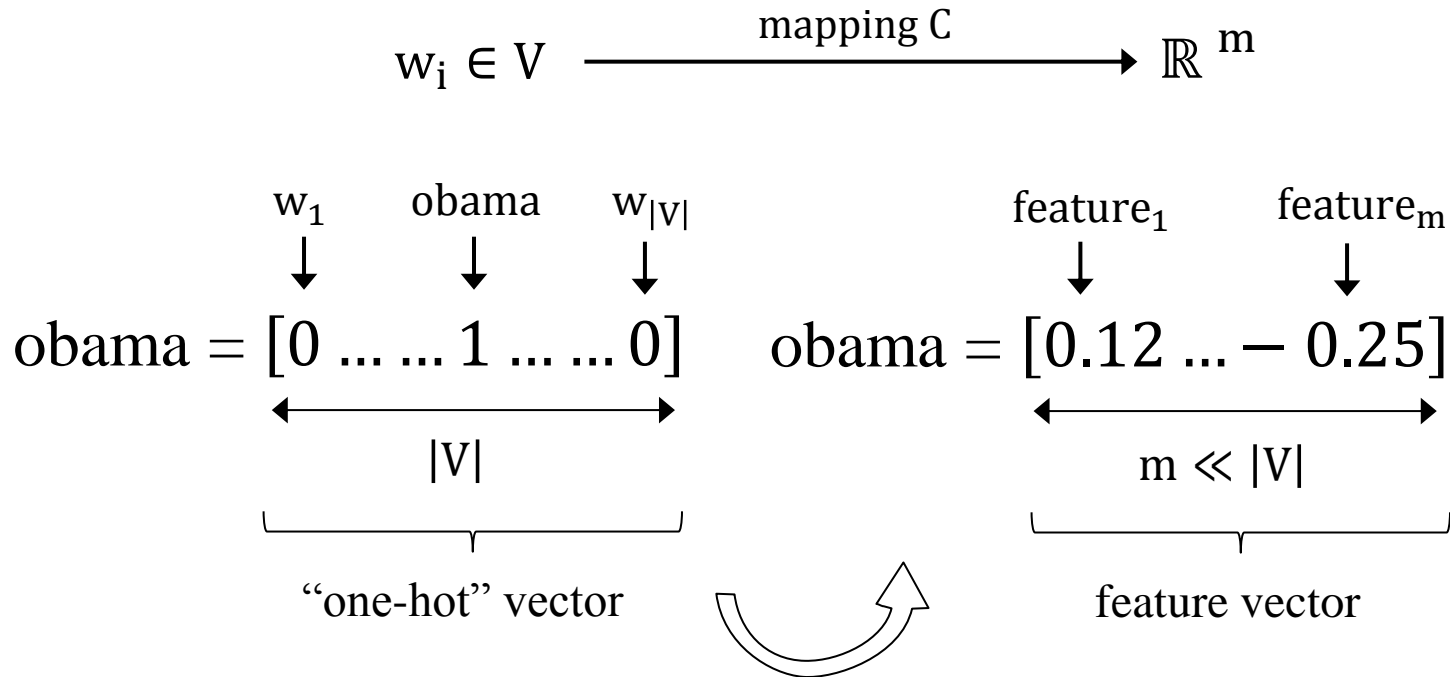
- We should assign similar probabilities to Obama speaks to the media in Illinois **and** the President addresses the press in Chicago
- This does not happen because of the “one-hot” vector space representation:

$$\begin{array}{lcl} \text{obama} = [0 \ 0 \ 0 \ 0 \ \dots \ 0 \ 1 \ 0 \ 0] & \left. \vphantom{\begin{array}{l} \text{obama} \\ \text{president} \end{array}} \right\} & \overrightarrow{\text{obama}} \cdot \overrightarrow{\text{president}} = \overrightarrow{0} \\ \text{president} = [0 \ 0 \ 0 \ 1 \ \dots \ 0 \ 0 \ 0 \ 0] & & \\ \\ \text{speaks} = [0 \ 0 \ 1 \ 0 \ \dots \ 0 \ 0 \ 0 \ 0] & \left. \vphantom{\begin{array}{l} \text{speaks} \\ \text{addresses} \end{array}} \right\} & \overrightarrow{\text{speaks}} \cdot \overrightarrow{\text{addresses}} = \overrightarrow{0} \\ \text{addresses} = [0 \ 0 \ 0 \ 0 \ \dots \ 0 \ 0 \ 1 \ 0] & & \\ \\ \text{illinois} = [1 \ 0 \ 0 \ 0 \ \dots \ 0 \ 0 \ 0 \ 0] & \left. \vphantom{\begin{array}{l} \text{illinois} \\ \text{chicago} \end{array}} \right\} & \overrightarrow{\text{illinois}} \cdot \overrightarrow{\text{chicago}} = \overrightarrow{0} \\ \text{chicago} = [0 \ 1 \ 0 \ 0 \ \dots \ 0 \ 0 \ 0 \ 0] & & \end{array}$$

- In each case, word pairs share no similarity
- This is obviously wrong
- We need to encode **word similarity** to be able to **generalize**

Word embeddings: distributed representation of words

- Each unique word is mapped to a point in a real continuous m-dimensional space
- Typically, $|V| > 10^6$, $100 < m < 500$

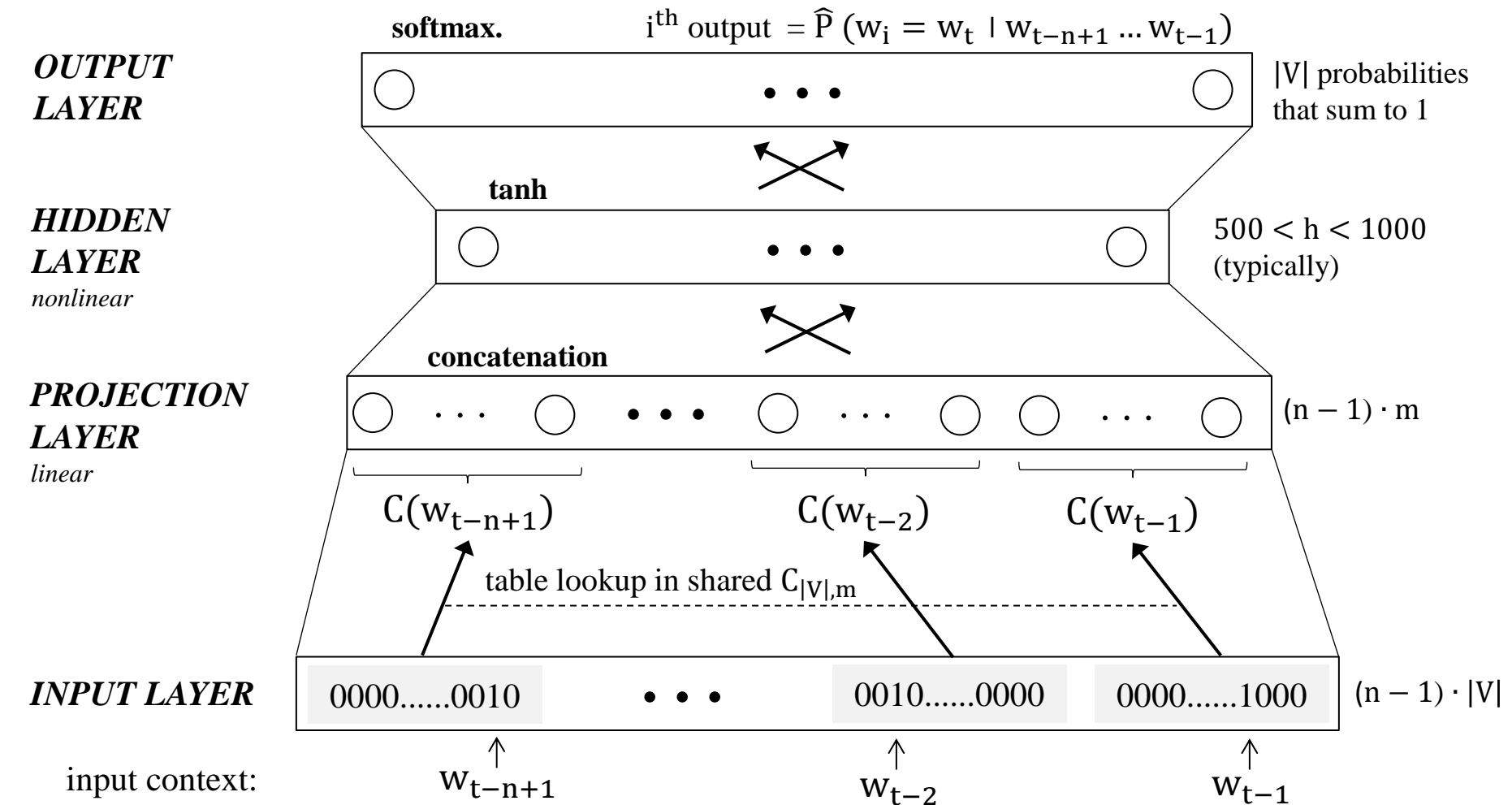


- Fighting the curse of dimensionality with:
 - **compression** (*dimensionality reduction*)
 - **smoothing** (*discrete to continuous*)
 - **densification** (*sparse to dense*)
- Similar words end up close to each other in the feature space

Neural Net Language Model (Bengio et al. 2003)

For each training sequence: input = (context, target) pair: $(w_{t-n+1} \dots w_{t-1}, w_t)$

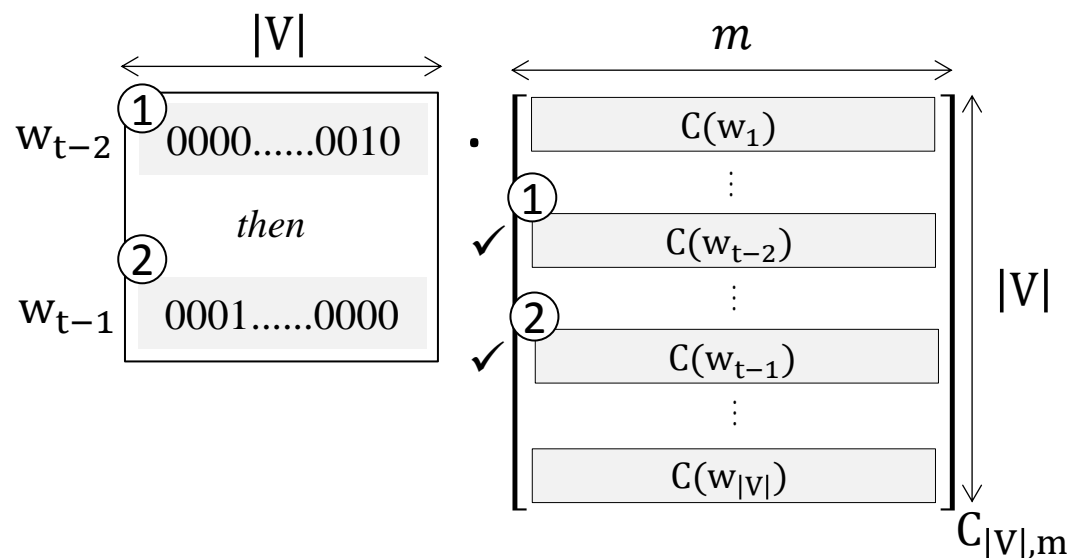
objective: minimize $E = -\log \hat{P}(w_t | w_{t-n+1} \dots w_{t-1})$



NNLM Projection layer

- Performs a simple table lookup in $C_{|V|,m}$: concatenate the rows of the shared mapping matrix $C_{|V|,m}$ corresponding to the context words

Example for a two-word context $w_{t-2}w_{t-1}$:



Concatenate ① and ② \rightarrow

$C(w_{t-2})$

$C(w_{t-1})$

- $C_{|V|,m}$ is **critical**: it contains the weights that are tuned at each step. After training, it contains what we're interested in: the **word vectors**

NNLM hidden/output layers and training

- Softmax (log-linear classification model) is used to output positive numbers that sum to one (a multinomial probability distribution):

for the i^{th} unit in the output layer:
$$\hat{P}(w_i = w_t \mid w_{t-n+1} \dots w_{t-1}) = \frac{e^{y w_i}}{\sum_{i'=1}^{|V|} e^{y w_{i'}}}$$

Where:

- $y = b + U \cdot \tanh(d + H \cdot x)$
 - \tanh : nonlinear squashing (link) function
 - x : concatenation $C(w)$ of the context weight vectors seen previously
 - b : output layer biases ($|V|$ elements)
 - d : hidden layer biases (h elements). Typically $500 < h < 1000$
 - U : $|V| * h$ matrix storing the *hidden-to-output* weights
 - H : $(h * (n - 1)m)$ matrix storing the *projection-to-hidden* weights
- $\theta = (\mathbf{b}, \mathbf{d}, \mathbf{U}, \mathbf{H}, \mathbf{C})$

- Complexity per training sequence: $n * m + n * m * h + \mathbf{h} * |\mathbf{V}|$
computational bottleneck: **nonlinear hidden layer** ($h * |V|$ term)
- **Training** is performed via stochastic gradient descent (learning rate ε):

$$\theta \leftarrow \theta + \varepsilon \cdot \frac{\partial E}{\partial \theta} = \theta + \varepsilon \cdot \frac{\partial \log \hat{P}(w_t \mid w_{t-n+1} \dots w_{t-1})}{\partial \theta}$$

(weights are initialized randomly, then updated via backpropagation)

NNLM facts

- - tested on Brown (1.2M words, $|V| \cong 16K$, 200K test set) and AP News (14M words, $|V| \cong 150K$ reduced to 18K, 1M test set) corpuses
- - Brown: $h = 100$, $n = 5$, $m = 30$
 - AP News: $h = 60$, $n = 6$, $m = 100$, **3 week** training using **40 cores**
 - 24% and 8% relative improvement (resp.) over traditional smoothed n-gram LMs in terms of test set perplexity: geometric average of $1/\hat{P}(w_t \mid w_{t-n+1} \dots w_{t-1})$
- Due to **complexity**, NNLM can't be applied to large data sets → poor performance on rare words
- Bengio et al. (2003) initially thought their main contribution was a more accurate LM. They left the interpretation and use of the word vectors as **future work**
- On the opposite, Mikolov et al. (2013) focus on the **word vectors**

Google's word2vec (Mikolov et al. 2013a)

- Key idea of word2vec: achieve better performance not by using a more complex model (i.e., with more layers), but by allowing a **simpler (shallower) model** to be trained on **much larger amounts of data**
- Two algorithms for learning words vectors:
 - **CBOW**: from context predict target (focus of what follows)
 - **Skip-gram**: from target predict context
- Compared to Bengio et al.'s (2003) NNLM:
 - no hidden layer (leads to 1000X speedup)
 - projection layer is shared (not just the weight matrix)
 - context: words from both **history & future**:
“You shall know a word by the company it keeps” (John R. Firth 1957:11):

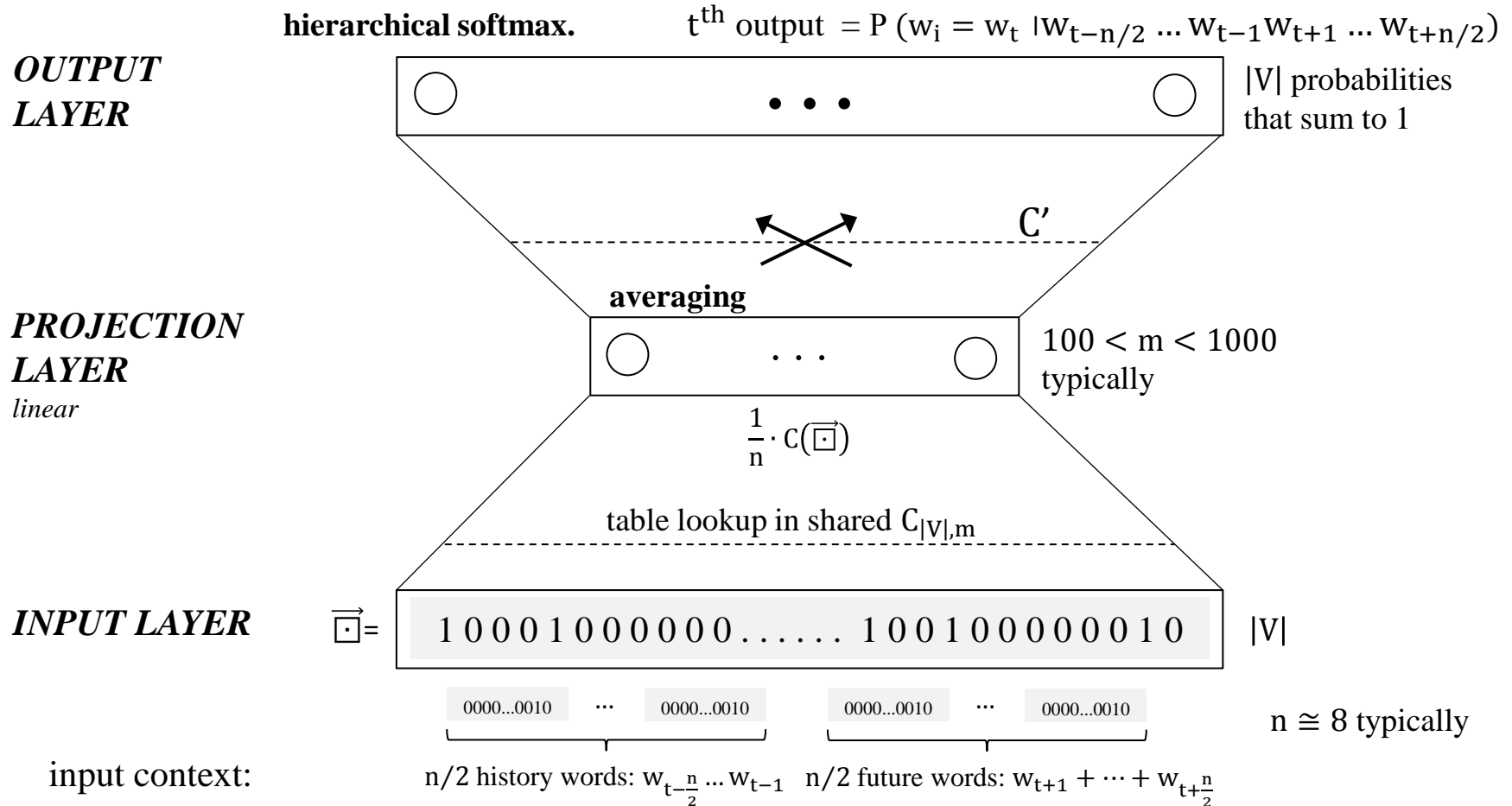
...Pelé has called **Neymar** an excellent player...
...At the age of just 22 years, **Neymar** had scored 40 goals in 58 internationals...
...occasionally as an attacking midfielder, **Neymar** was called a true phenomenon...

← These words will represent **Neymar** →

word2vec's Continuous Bag-of-Words (CBOW)

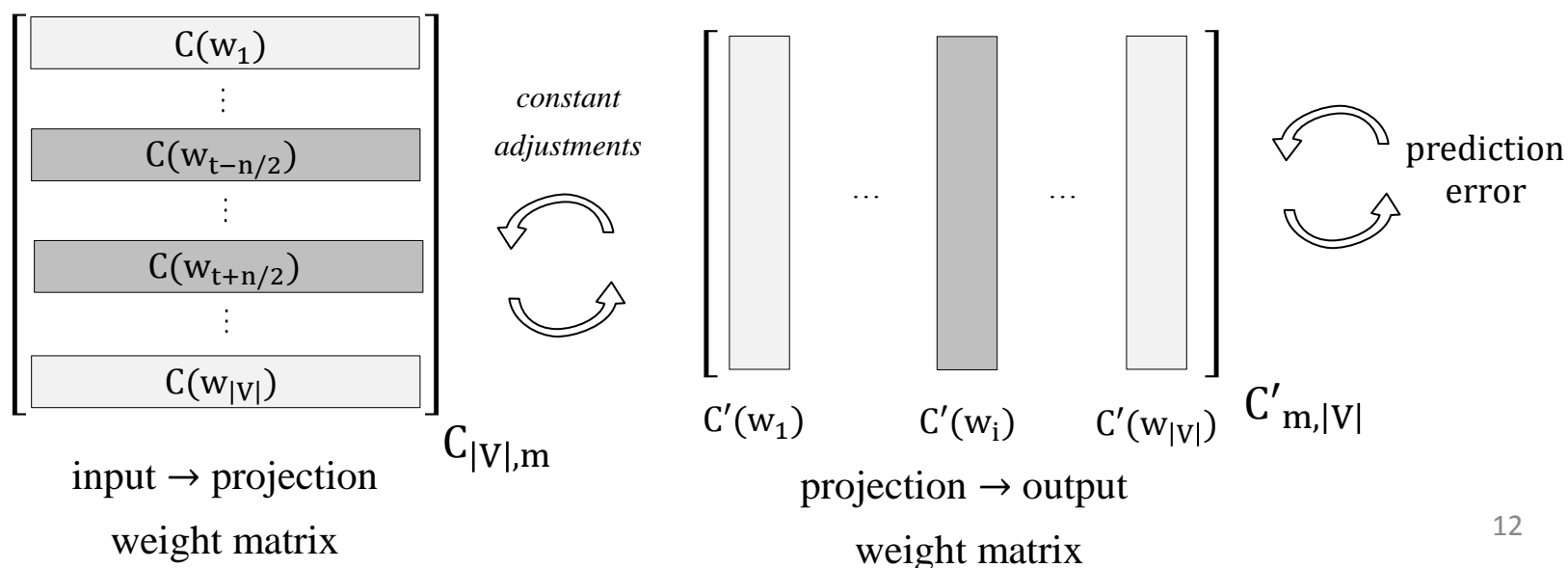
For each training sequence: input = (context, target) pair: $(w_{t-\frac{n}{2}} \dots w_{t-1} w_{t+1} \dots w_{t+\frac{n}{2}}, w_t)$

objective: minimize $E = -\log \hat{P}(w_t | w_{t-\frac{n}{2}} \dots w_{t-1} w_{t+1} \dots w_{t+\frac{n}{2}})$



Weight updating intuition

- For each (context, target= w_t) pair, only the word vectors from matrix C corresponding to the context words are updated
 - Recall that we compute $P(w_i = w_t \mid \text{context}) \forall w_i \in V$. We compare this distribution to the true probability distribution (1 for w_t , 0 elsewhere)
 - If $P(w_i = w_t \mid \text{context})$ is **overestimated** (i.e., > 0 , happens in potentially $|V| - 1$ cases), some portion of $C'(w_i)$ is **subtracted** from the context word vectors in C , proportionally to the magnitude of the error
 - Reversely, if $P(w_i = w_t \mid \text{context})$ is **underestimated** (< 1 , happens in potentially 1 case), some portion of $C'(w_i)$ is **added** to the context word vectors in C
- at each step the words move away or get closer to each other in the feature space → clustering
 → analogy with a **spring force** layout. See online [demo](#) with Chrome



word2vec facts

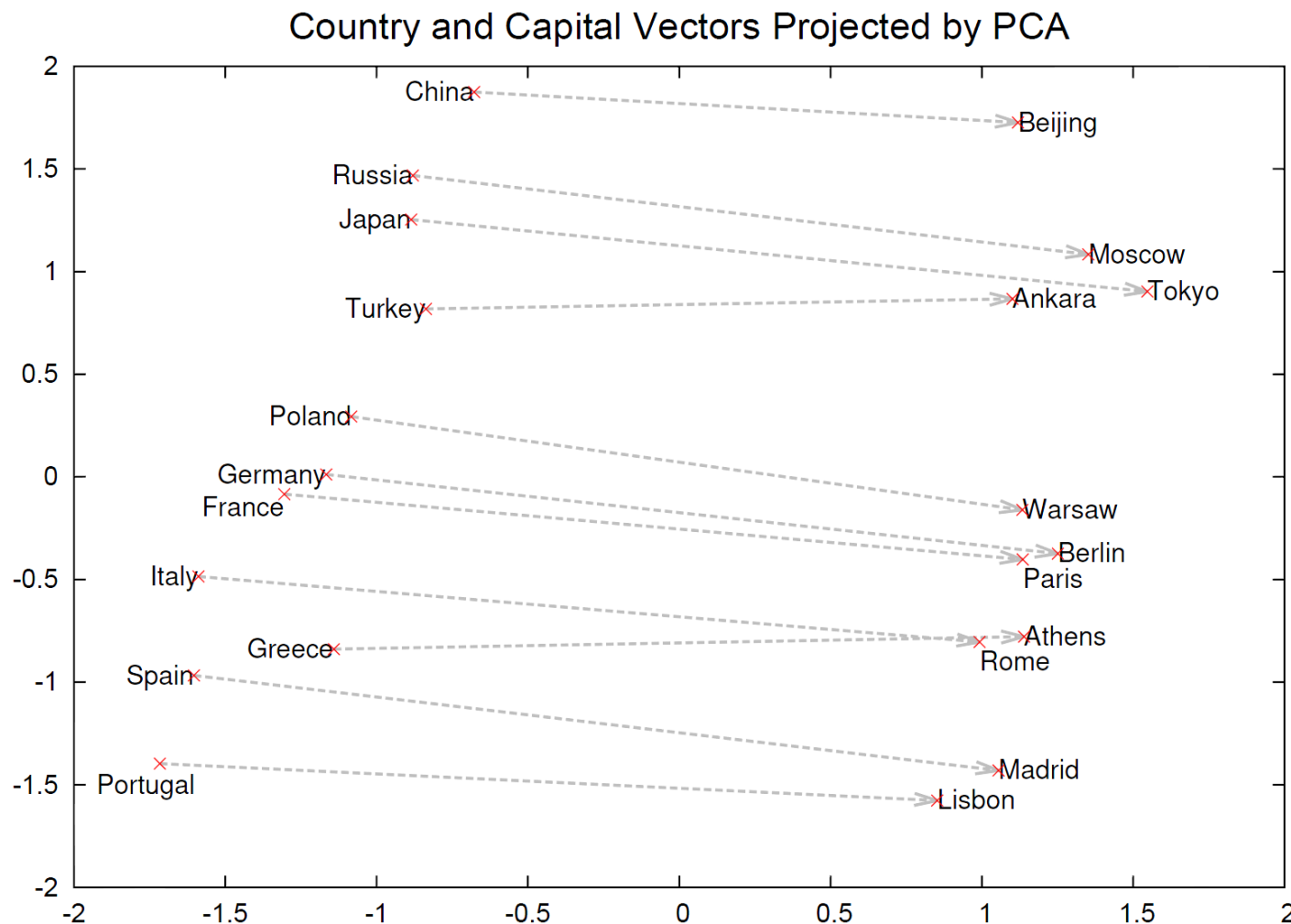
- Complexity is $n * m + m * \log|V|$ (Mikolov et al. 2013a)
- On Google news 6B words training corpus, with $|V| \sim 10^6$:
 - CBOW with $m = 1000$ took **2 days** to train on **140 cores**
 - Skip-gram with $m = 1000$ took **2.5 days** on **125 cores**
 - NNLM (Bengio et al. 2003) took **14 days** on **180 cores**, for $m = 100$ only!
(note that $m = 1000$ was not reasonably feasible on such a large training set)
- word2vec training speed $\cong 100K$ -5M words/s
- Quality of the word vectors:
 - \nearrow significantly with **amount of training data** and **dimension of the word vectors** (m),
with diminishing relative improvements
 - measured in terms of accuracy on 20K semantic and syntactic association tasks.
e.g., words in **bold** have to be returned:

Capital-Country	Past tense	Superlative	Male-Female	Opposite
Athens: Greece	walking: walked	easy: easiest	brother: sister	ethical: unethical

Adapted from Mikolov et al. (2013a)

- Best NNLM: 12.3% overall accuracy. Word2vec (with Skip-gram): 53.3%

Remarkable properties of word2vec's word vectors



Mikolov et al. (2013b)

regularities between words are encoded in the difference vectors
e.g., there is a constant **country-capital** difference vector

After p14: Training Models (simple starter)

- See:
<https://radimrehurek.com/gensim/models/word2vec.html>
- Step One: show example and gensim calls
`1_create_and_test_model.py`
- Show testing the model

Exercise 1 (Mini, 5min):

- Download a book from www.gutenberg.org (plain txt)
- Gensim expects a list of lists (of sentences and words)
 - Eg like in `gutenberg.sents(fileid)`
- Train model
- Test it ... (`most_similar()` etc)

Some operations in Gensim

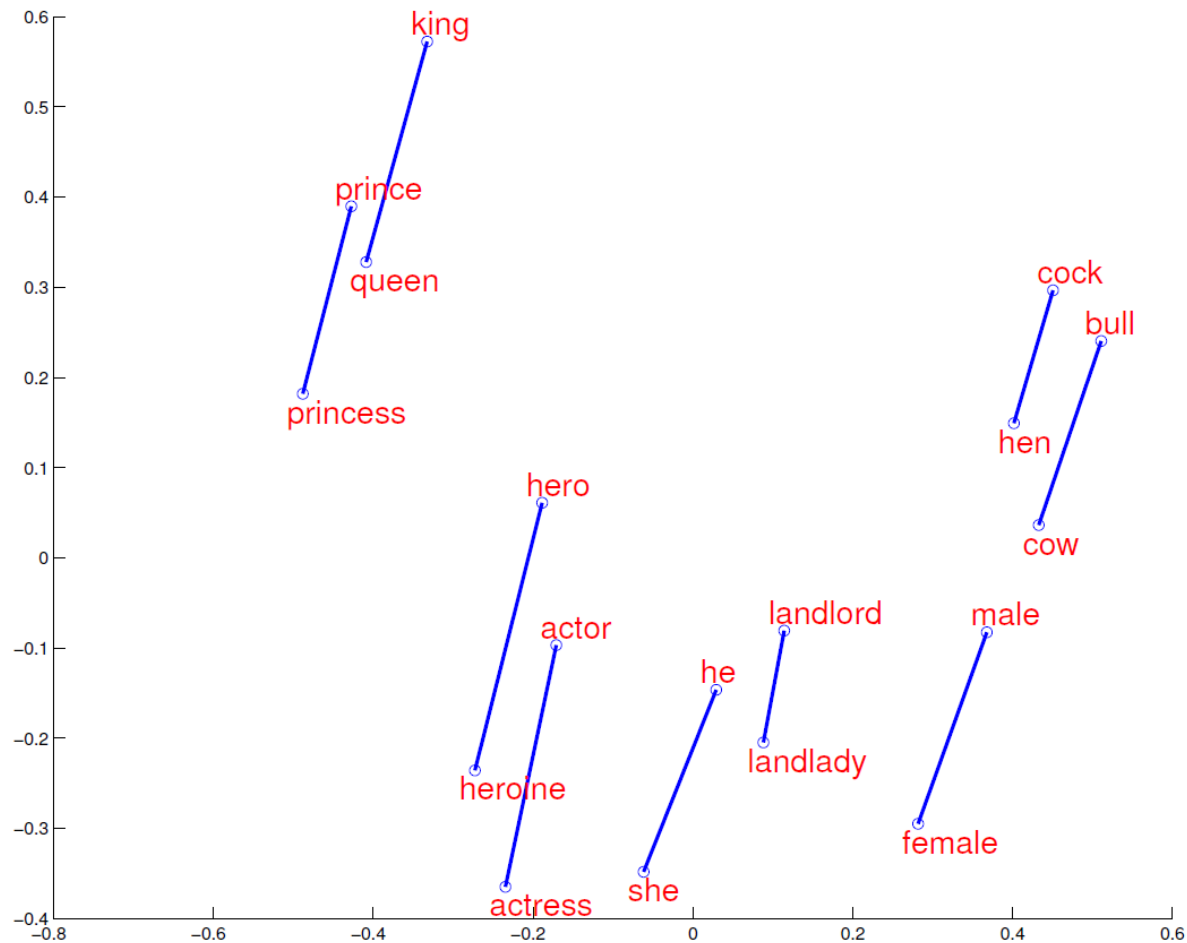
- `similarity(a,b)`
- `most_similar(a)`
- `most_similar(positive=[], negative=[])`
 - “analogy operation!” (famous king-queen example)
- All these operations work MUCH better with big corpora (and not just a book)
- **Often** people use pretrained models (eg on Wikipedia corpus or Google News corpus, etc)

Show code example ([2_playing_around.py](#))

Exercise 2: test these functions with your model

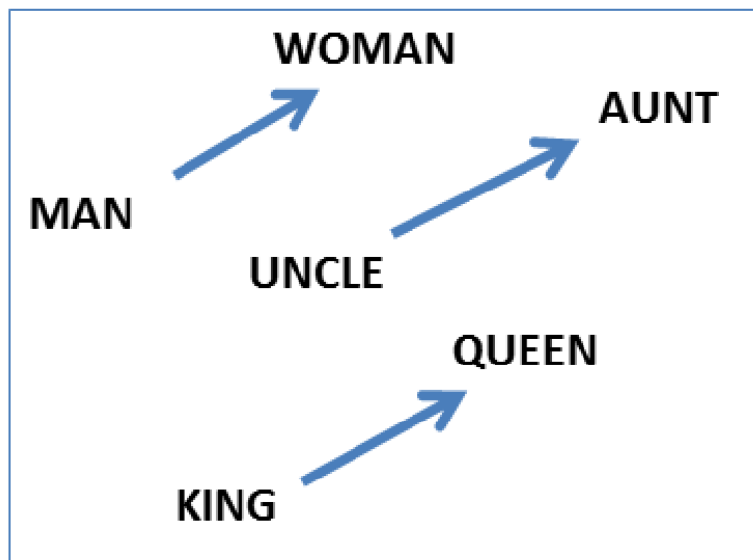
See also: <http://ahogrammer.com/2017/01/20/the-list-of-pretrained-word-embeddings/>

Remarkable properties of word2vec's word vectors

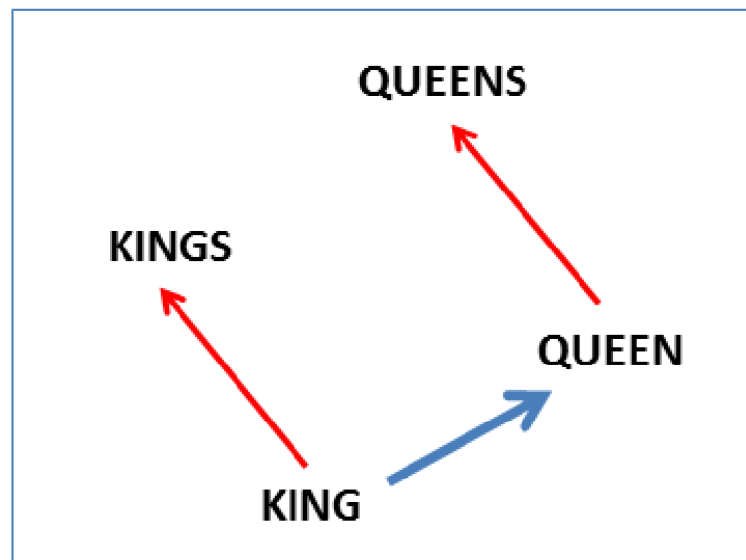


constant **female-male** difference vector

Remarkable properties of word2vec's word vectors



constant **male-female** difference vector



constant **singular-plural** difference vector

- Vector operations are supported and make intuitive sense:

$$w_{king} - w_{man} + w_{woman} \cong w_{queen}$$

$$w_{einstein} - w_{scientist} + w_{painter} \cong w_{picasso}$$

$$w_{paris} - w_{france} + w_{italy} \cong w_{rome}$$

$$w_{his} - w_{he} + w_{she} \cong w_{her}$$

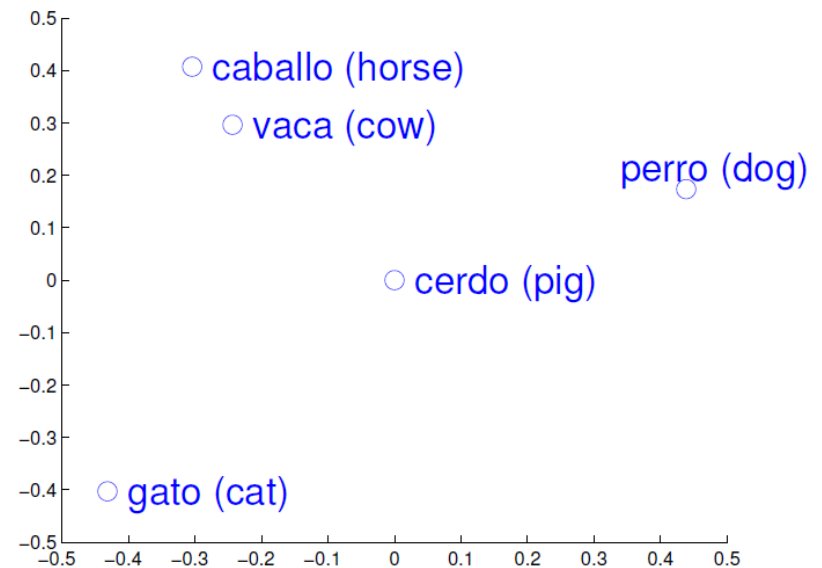
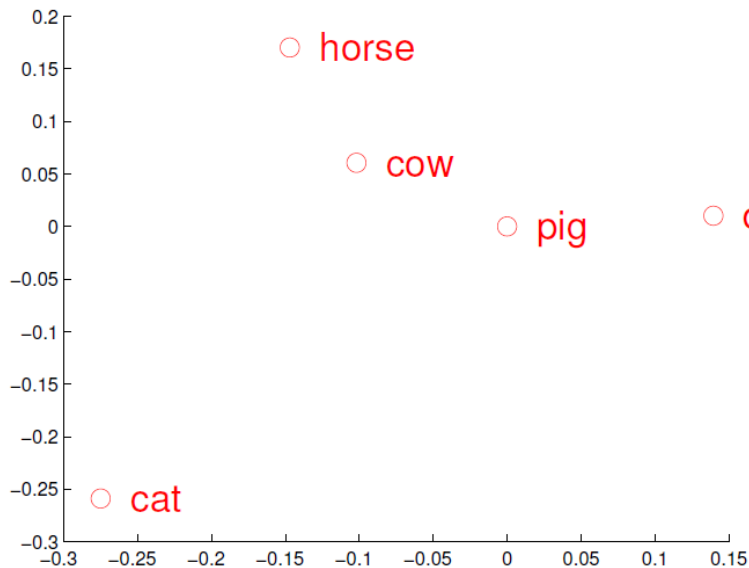
$$w_{windows} - w_{microsoft} + w_{google} \cong w_{android}$$

$$w_{cu} - w_{copper} + w_{gold} \cong w_{au}$$

- Online [demo](#) (scroll down to end of tutorial)

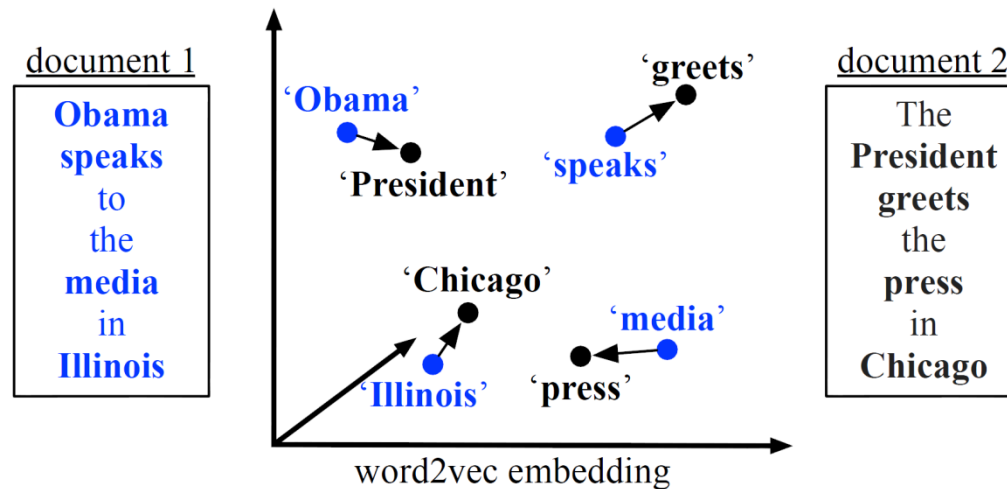
Applications

- High quality word vectors boost performance of all NLP tasks, including document classification, machine translation, information retrieval...
- Example for English to Spanish machine translation:

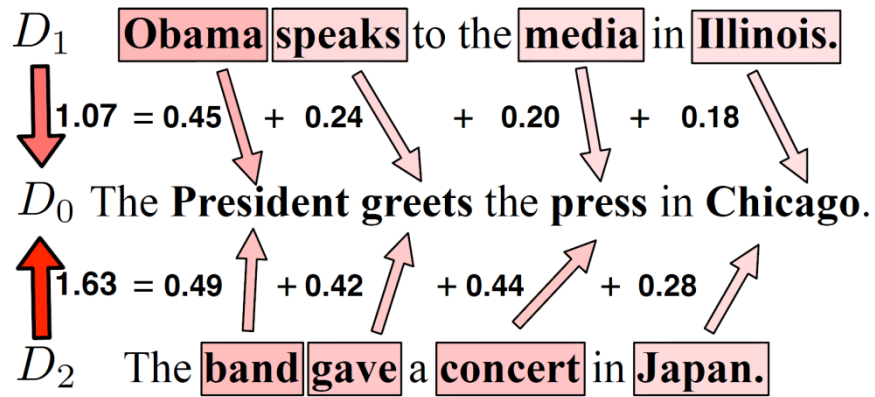


About 90% reported accuracy (Mikolov et al. 2013c)

Application to document classification



With the BOW representation D_1 and D_2 are at equal distance from D_0 . Word embeddings allow to capture the fact that D_1 is closer.



Resources

Papers:

[Chen, S. F., & Goodman, J. \(1999\). An empirical study of smoothing techniques for language modeling. *Computer Speech & Language*, 13\(4\), 359-393.](#)

[Katz, S. M. \(1987\). Estimation of probabilities from sparse data for the language model component of a speech recognizer. *Acoustics, Speech and Signal Processing, IEEE Transactions on*, 35\(3\), 400-401.](#)

[Bengio, Yoshua, et al. "A neural probabilistic language model." *The Journal of Machine Learning Research* 3 \(2003\): 1137-1155.](#)

[Mikolov, T., Chen, K., Corrado, G., & Dean, J. \(2013a\). Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.](#)

[Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., & Dean, J. \(2013b\). Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems* \(pp. 3111-3119\).](#)

[Mikolov, T., Le, Q. V., & Sutskever, I. \(2013c\). Exploiting similarities among languages for machine translation. *arXiv preprint arXiv:1309.4168*.](#)

[Rong, X. \(2014\). word2vec Parameter Learning Explained. *arXiv preprint arXiv:1411.2738*.](#)

Google word2vec webpage (with link to C code):

<https://code.google.com/p/word2vec/>

Python implementation:

<https://radimrehurek.com/gensim/models/word2vec.html>

Kaggle tutorial on movie review classification with word2vec:

<https://www.kaggle.com/c/word2vec-nlp-tutorial/details/part-2-word-vectors>

Insightful blogpost: <http://colah.github.io/posts/2014-07-NLP-RNNs-Representations/>

Visualizing models

After intro slides:

- We need to reduce the model from 300 (or whatever) to exactly 2 dimensions
 - This is typically done with t-SNE or PCA
 - Show graph: <https://www.quora.com/How-do-I-visualise-word2vec-word-vectors>
- See file `visualize_model.py`
- Run it ..
- **Exercise 3: Visualize your model..**
 - just copy the code
 - experiment with vocabulary size ..

Types of Word Embedding Models

- **Word2vec:** already discussed
- **GloVe:** applies dimension-reduction on a word-word co-occurrence matrix.
- **FastText:** is based on the skip-gram model, but also makes use of word morphology information. By using sub-word information, fastText can also supply vectors for out-of-vocabulary words.
- **Word2vec-f:** trained on dependency-parsed data (input in conll format) – well suited from some tasks, but not for all. Functional similarity not semantic similarity.

Big exercise: sentence similarity

- Use word embeddings to find **similar sentences to a given sentence**
- **Use:** `melville-moby_dick.txt` (nltk gutenbergl) and for example sentence: “Ahab boat”
- Possible starting point: represent sentence as **sum of word vectors**
- One solution is to use numpy to create avg vectors and use the ‘dot’ operation for similarity – another option is the Gensim `n_similarity()` function
- Did you normalize sentence vectors? Does it change anything?
- How can we improve the sentence vectors?

Repetition

- What are 1-hot encodings?
- What are dense encodings?
- Why have those representations anyway?
- Word embeddings: count-based vs prediction-based?
- What are language models?
- How to train embeddings?
- What basic functions does `gensim` provide?

Evaluation of Word Embeddings

- How can we evaluate the quality of embedding models
 - ideas?

WE Evaluation

- Any ideas on how we could evaluate the quality of a word embedding model?

Evaluation of Word Embeddings

- Intrinsic
 - Word similarity (with datasets)
 - Word analogy (with datasets)
- Extrinsic
 - On ‘downstream’ task
 - Give example with eg **NER**

see eg: <https://arxiv.org/abs/1801.09536>

Intrinsic Evaluation

- Performance in Analogy tasks
word2vec original dataset:

<https://github.com/nicholas-leonard/word2vec/blob/master/questions-words.txt>

- Performance in Word Similarity Tasks:
 - There are many datasets for this: WordSim-353, SimLex-999, MEN-3000, etc

- Eg:

https://github.com/kliegr/word_similarity_relatedness_datasets/blob/master/win353.csv

How can we use this dataset to measure the quality of the WE model?

Intrinsic Evaluation

- Creation of semantic similarity datasets?
 - 1.) Pick word pairs
 - 2.) Human ratings
 - 3.) Avg rating of humans
- We created such datasets for Thai language in 2019:
<https://arxiv.org/abs/1904.04307>
(published in IEEE Access)

Limitations of word embeddings

- Which limitations come to mind?

Use Case 1: Digital humanities

- **Goal:** we want to find out how good word-embedding-models work for the **extraction of specific relations** in literature.
- Did research paper where input text was the books from “A song of ice and fire” (GRR Martin) and “Harry Potter”
- For basic procedure:
 - Train word embedding models for the book corpus
 - Set up test relations, we did
 - Analogies
 - doesnt_match
 - Evaluate the system

Use Case 1

- git clone repository (in Pycharm) from
https://github.com/gwohlgen/nlp4is_word_embeddings
- Read README.md
- Explore:
 - a) datasets, dataset formats, and create_questions.py
 - b) model
 - c) src folder

Further reading:

Gerhard Wohlgenannt, Ekaterina Chernyak and Dmitry Ilvovsky:
Extracting Social Networks from Literary Text with Word Embedding Tools
<https://www.clarin-d.net/images/lt4dh/pdf/LT4DH04.pdf>

Use Case1: Exercise

- Easy: make a little script that loads the model and prints the 10 closest terms to:
 - Cersei, Kingslayer, dragon
 - What is more similar: “Jaime” and “Tyrion”, or “Jaime” and “Sansa”
 - Analogy:
 - Try: “man” to “king” like “woman” to ?
 - Try: “Cersei” to “Lannister” like “Theon” to ?
- Add relations to the datafiles (if you don’t know the books – just add random relations)
 - Analogies
 - doesnt_match
- Re-run the evaluations: compare results with and without new relations