Unit 3 – Word Embeddings

- **Starting** with some details on term-term matrices, PPMI, etc
- **Main Part:** Word embeddings

Term-Term Matrices

- In IR systems we typically use Term-Document matrices
- But in many NLP applications we are interested in **term-term co-occurrence** matrices
- Co-occurrence can be measured in different ways, for example
  - Within a unit like a **sentence** or **paragraph**
  - Within a **word window** (left and/or right) of the target word – eg. a word window of 5
  - Q: when could a co-occurrence matrix be useful?

Co-occurrence matrix - Example

1. I enjoy flying.
2. I like NLP.
3. I like deep learning.

The resulting counts matrix will then be:

$$X = \begin{array}{c} \\ I \\ like \\ enjoy \\ deep \\ learning \\ NLP \\ flying \\ . \end{array} \begin{array}{c} \begin{array}{cccccccc} I & like & enjoy & deep & learning & NLP & flying & . \end{array} \\ \begin{bmatrix} 0 & 2 & 1 & 0 & 0 & 0 & 0 & 0 \\ 2 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \end{bmatrix} \end{array}$$

Co-occurrence Matrices

- **Distributional hypothesis** – "a word is characterized by the company it keeps"
- Words are defined by their context (words)

A bottle of **tesgüino** is on the table
Everybody likes **tesgüino**
Tesgüino makes you drunk
We make **tesgüino** out of corn.

Co-occurrence Matrices

- These matrices are often input to further processing, eg. for word embeddings, SVD, etc.
- **Weighting** of the matrix:
  - **Raw counts:** but not best option
    - “the” and “of” are very frequent, but maybe not the most discriminative
  - **PPMI** (see next slide)
    - Goal: context is informative about the target word
  - …

Co-occurrence Matrices – PPMI

- Starts from co-occurrence counts

PMI between two words: (Church & Hanks 1989)

Do words x and y co-occur more than if they were independent?

$$\mathrm{PMI}(word_1, word_2) = \log_2 \frac{P(word_1, word_2)}{P(word_1)P(word_2)}$$

- PMI goes from neg. inf. to pos. inf.
- **PPMI:** we just set all negative values to 0.
  - Low counts are unreliable, and neg. relation hard to understand for humans

Co-occurrence Matrices

- Co-occ. matrices are **sparse** representations
- **Next** we look at **dense** representations, like word embeddings

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 ...

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 $\mathrm{P}(s = \mathrm{w}_1 \ldots \mathrm{w}_\mathrm{k})$ in some domain of interest

$$\mathrm{P}(s) = \prod_{\mathrm{i}=1}^{\mathrm{k}} \mathrm{P}\left(\mathrm{w}_\mathrm{i} \mid \mathrm{w}_1 \ldots \mathrm{w}_{\mathrm{i}-1}\right)$$

$$\text{e.g., } \mathrm{P}\left(\mathrm{w}_1\mathrm{w}_2\mathrm{w}_3\right) = \mathrm{P}\left(\mathrm{w}_1\right) \mathrm{P}\left(\mathrm{w}_2 \mid \mathrm{w}_1\right) \mathrm{P}\left(\mathrm{w}_3 \mid \mathrm{w}_1\mathrm{w}_2\right)$$

- Traditional n-gram language model assumption:
"the probability of a word depends only on **context** of $\mathrm{n} - 1$ previous words"

$$\Rightarrow \widehat{\mathrm{P}}(s) = \prod_{\mathrm{i}=1}^{\mathrm{k}} \mathrm{P}\left(\mathrm{w}_\mathrm{i} \mid \mathrm{w}_{\mathrm{i}-\mathrm{n}+1} \ldots \mathrm{w}_{\mathrm{i}-1}\right)$$

- Typical ML-smoothing learning process (e.g., Katz 1987):
  1. compute $\widehat{\mathrm{P}}\left(\mathrm{w}_\mathrm{i} \mid \mathrm{w}_{\mathrm{i}-\mathrm{n}+1} \ldots \mathrm{w}_{\mathrm{i}-1}\right) = \frac{\#\mathrm{w}_{\mathrm{i}-\mathrm{n}+1} \ldots \mathrm{w}_{\mathrm{i}-1}\mathrm{w}_\mathrm{i}}{\#\mathrm{w}_{\mathrm{i}-\mathrm{n}+1} \ldots \mathrm{w}_{\mathrm{i}-1}}$ on training corpus
  2. smooth to avoid zero probabilities

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 ↔ assigning a probability to each of the 100.000^{10} slots
- **probability mass vanishes** → more data is needed to fill the huge space
- the more data, the more unique words! → vicious circle
- what about corpuses of 10^6 unique words?

- → in practice, contexts are typically limited to size 2 (trigram model)
  e.g., famous Katz (1987) smoothed trigram model

- → 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:

$$\left.\begin{array}{r}\text{obama} = [0\ 0\ 0\ 0\ \ \ldots\ \ 0\ 1\ 0\ 0] \\ \text{president} = [0\ 0\ 0\ 1\ \ \ldots\ \ 0\ 0\ 0\ 0]\end{array}\right\} \quad \overrightarrow{\text{obama}}.\overrightarrow{\text{president}} = \vec{0}$$

$$\left.\begin{array}{r}\text{speaks} = [0\ 0\ 1\ 0\ \ \ldots\ \ 0\ 0\ 0\ 0] \\ \text{addresses} = [0\ 0\ 0\ 0\ \ \ldots\ \ 0\ 0\ 1\ 0]\end{array}\right\} \quad \overrightarrow{\text{speaks}}.\overrightarrow{\text{addresses}} = \vec{0}$$

$$\left.\begin{array}{r}\text{illinois} = [1\ 0\ 0\ 0\ \ \ldots\ \ 0\ 0\ 0\ 0] \\ \text{chicago} = [0\ 1\ 0\ 0\ \ \ldots\ \ 0\ 0\ 0\ 0]\end{array}\right\} \quad \overrightarrow{\text{illinois}}.\overrightarrow{\text{chicago}} = \vec{0}$$

- 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, $|V| > 10^6$, $100 < m < 500$

$$w_i \in V \xrightarrow{\text{mapping } C} \mathbb{R}^m$$

$$\text{obama} = \underbrace{[\overset{w_1}{\downarrow}\!\!0 \dots \dots \overset{\text{obama}}{\downarrow}\!\!1 \dots \dots \overset{w_{|V|}}{\downarrow}\!\!0]}_{|V|} \qquad \text{obama} = \underbrace{[\overset{\text{feature}_1}{\downarrow}\!\!0.12 \dots \overset{\text{feature}_m}{\downarrow}\!\!-0.25]}_{m \ll |V|}$$

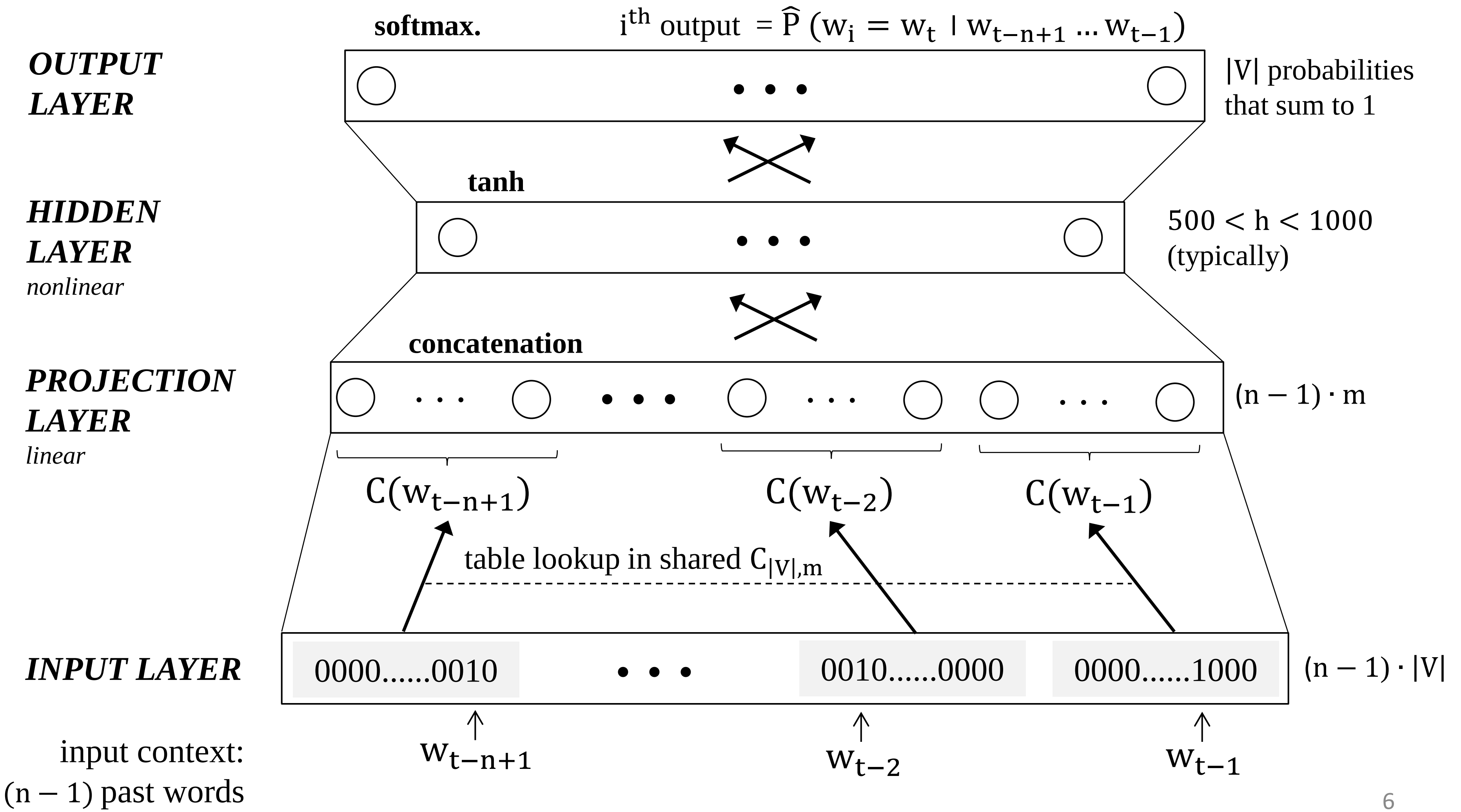
“one-hot” vector → feature vector

- 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: $(w_{t-n+1} \ldots w_{t-1}, w_t)$

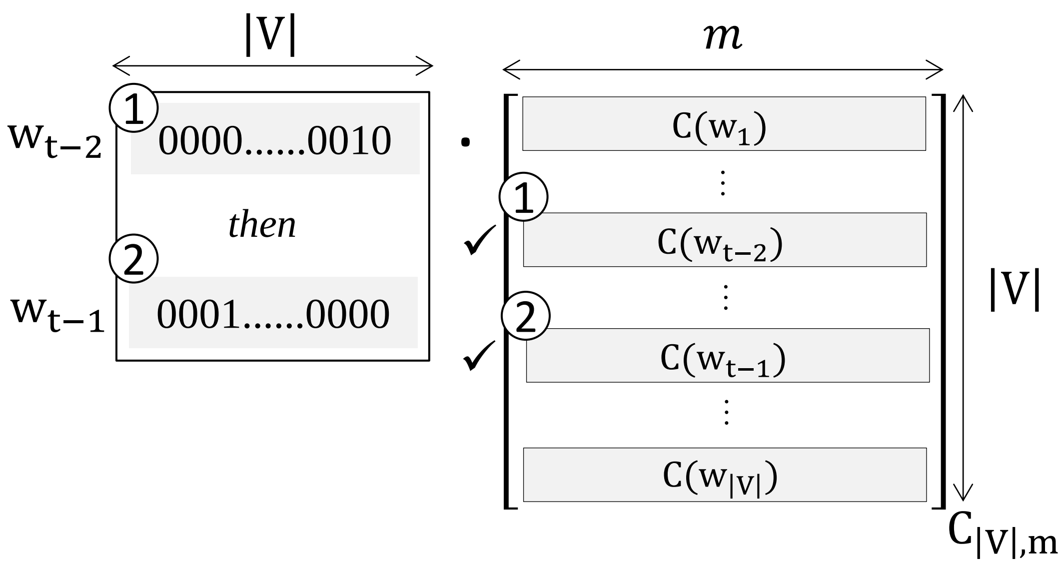
objective: minimize $E = -\log \widehat{P}(w_t \mid w_{t-n+1} \ldots w_{t-1})$



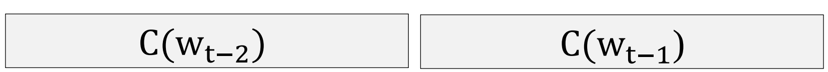
NNLM Projection layer

- 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 $\rightarrow$



- $C_{|V|,m}$ 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: $\widehat{P}(w_i = w_t \mid w_{t-n+1} \ldots w_{t-1}) = \frac{e^{y_{w_i}}}{\sum_{i'=1}^{|V|} e^{y_{w_{i'}}}}$

  Where:
  - $y = b + U.\tanh(d + H.x)$
  - 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

  $\rightarrow \boldsymbol{\theta} = (\mathbf{b}, \mathbf{d}, \mathbf{U}, \mathbf{H}, \mathbf{C})$

- 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):

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

(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 $1/\hat{P}(w_t \mid w_{t-n+1} \ldots w_{t-1})$

- 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 let 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):

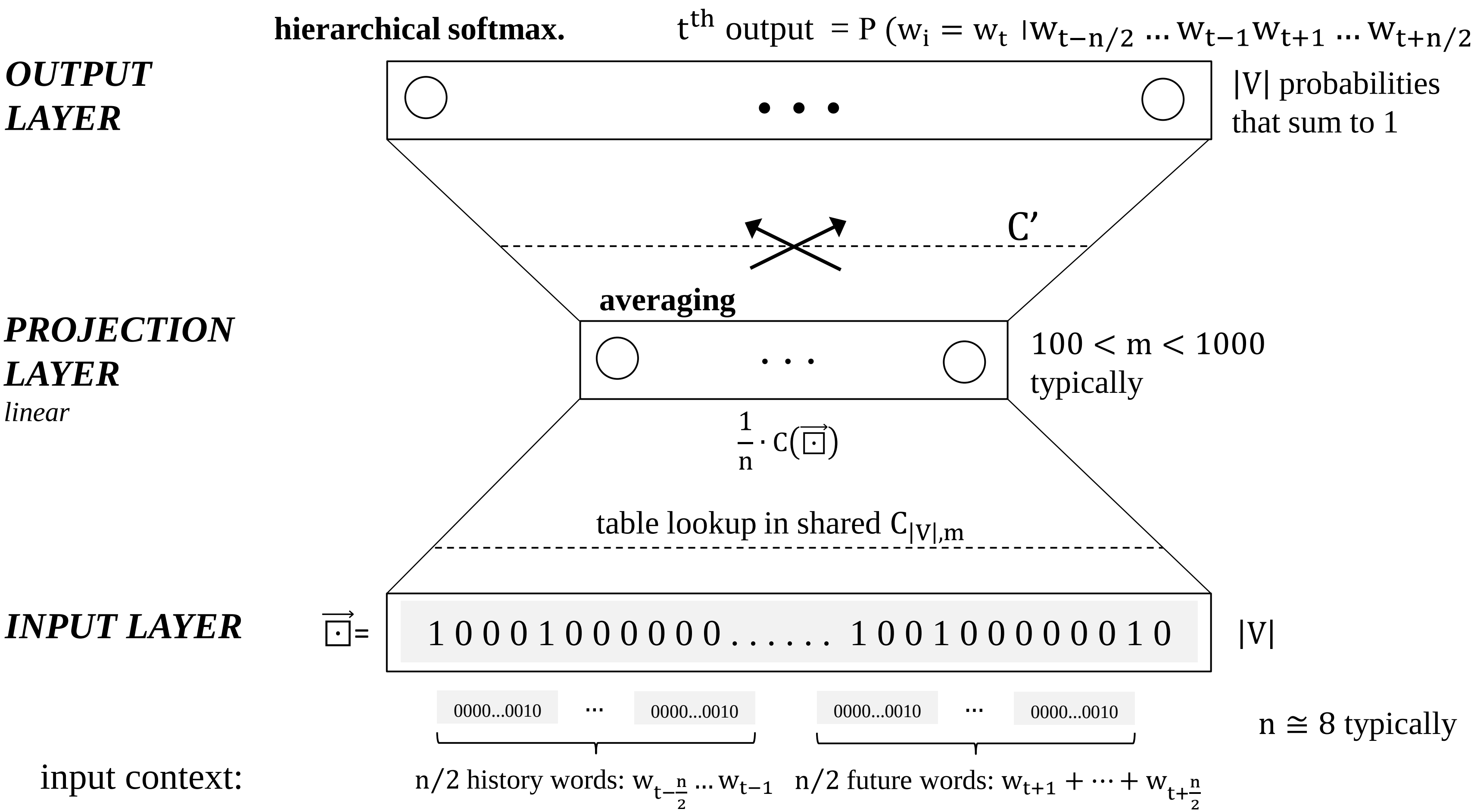
```
…Pelé has called Neymar an excellent player…
…At the age of just 22 years, Neymar had scored 40 goals in 58 internationals…
…occasionally as an attacking midfielder, Neymar was called a true phenomenon…
```

These words will represent **Neymar**

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

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

objective: minimize $E = -\log \hat{P}\left(w_t \mid w_{t-n/2} \ldots w_{t-1} w_{t+1} \ldots w_{t+n/2}\right)$

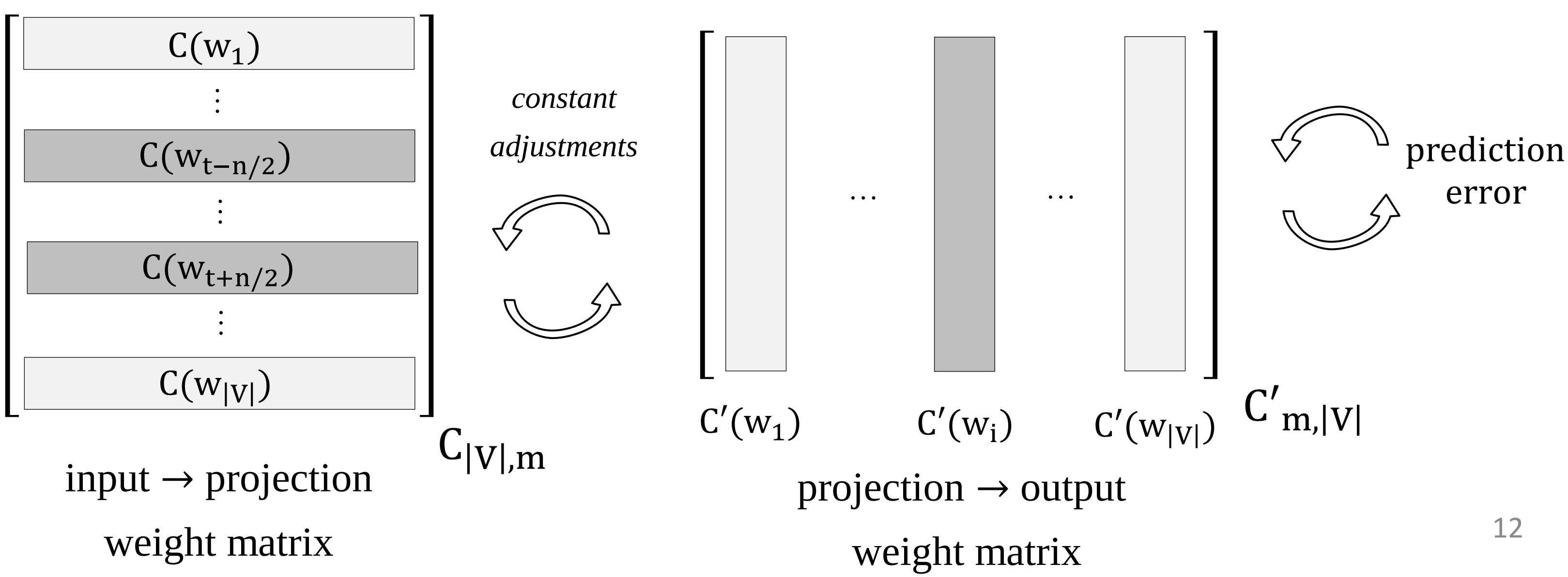


Weight updating intuition

- 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 $n * m + m * \log|\mathbf{V}|$ (Mikolov et al. 2013a)
- On Google news 6B words training corpus, with $|\mathbf{V}| \sim 10^6$:
  - 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:

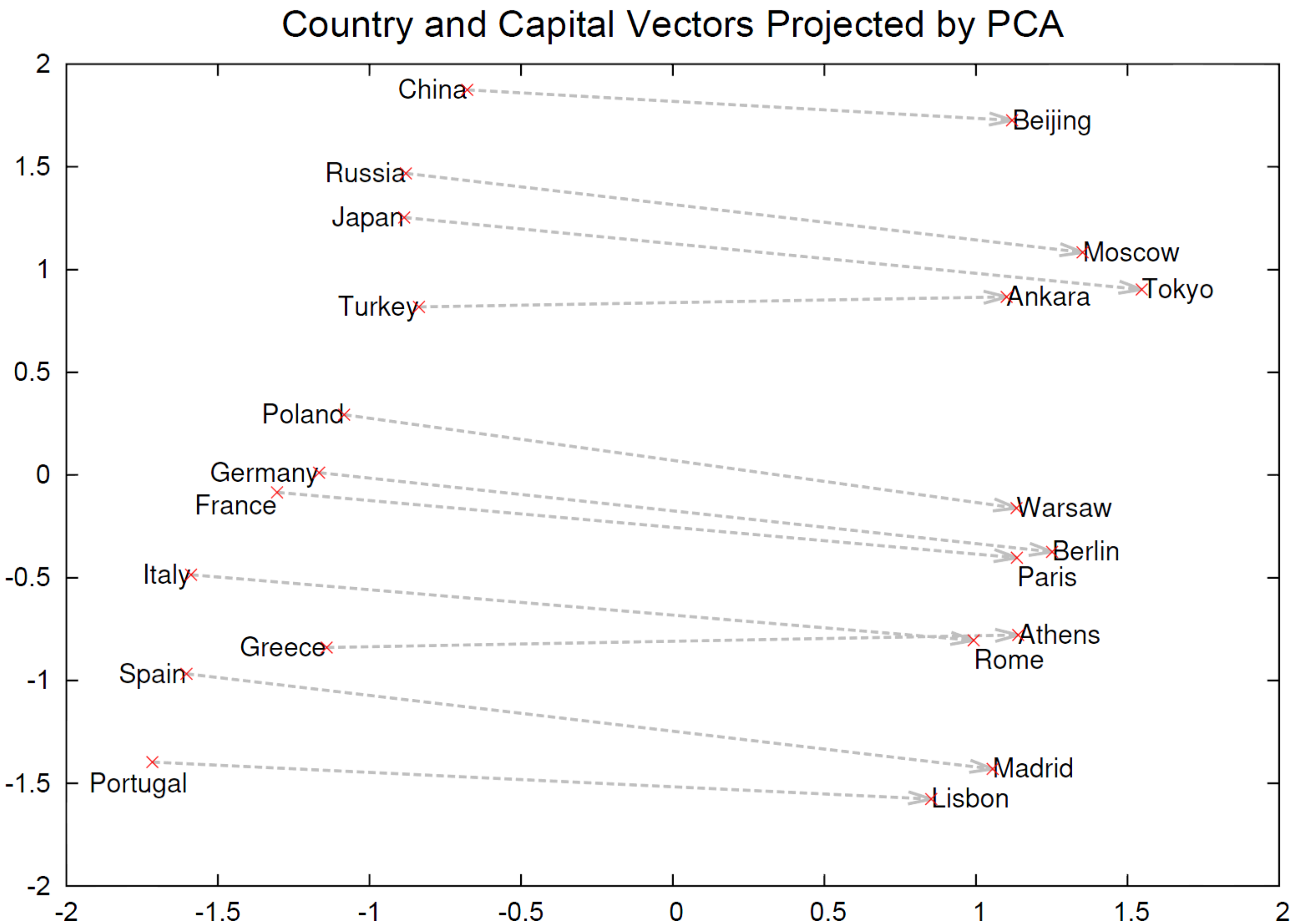
Capital-Country	Past tense	Superlative	Male-Female	Opposite
Athens: **Greece**	walking: **walked**	easy: **easiest**	brother: **sister**	ethical: **unethical**

Adapted from Mikolov et al. (2013a)

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

References: http://www.scribd.com/doc/285890694/NIPS-DeepLearningWorkshop-NNforText#scribd
https://code.google.com/p/word2vec/

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

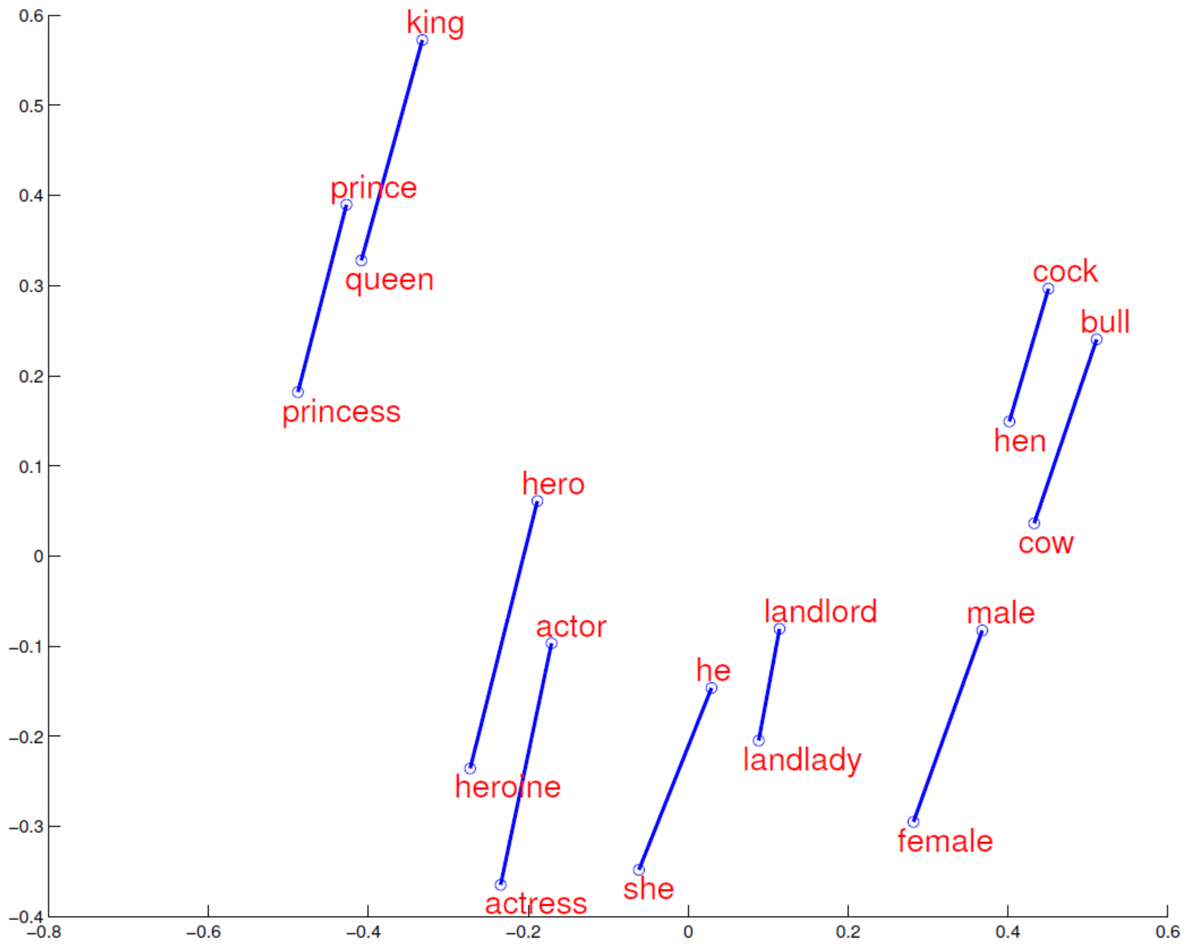
- `similarity(a,b)`
- `most_similar(a)`
- `most_similar(positive=[], negative=[])`
  – "analogy operation!" (famous king-queen example)

- 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/`

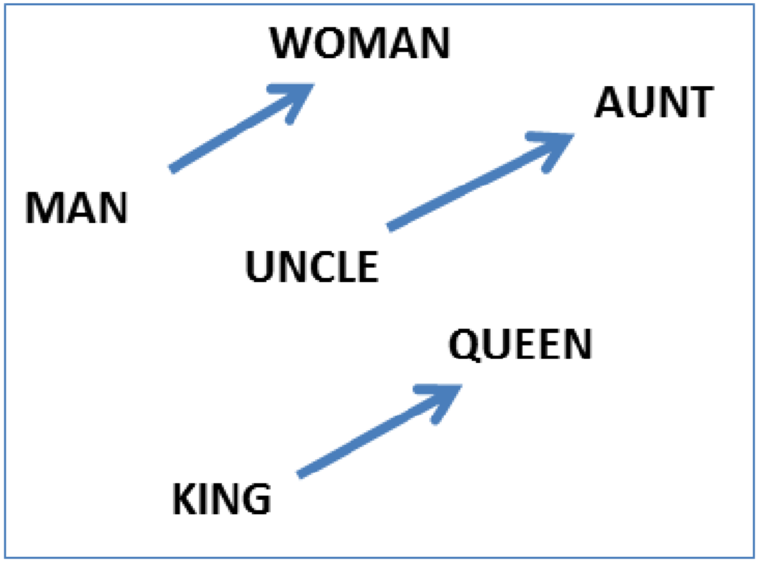
Remarkable properties of word2vec's word vectors



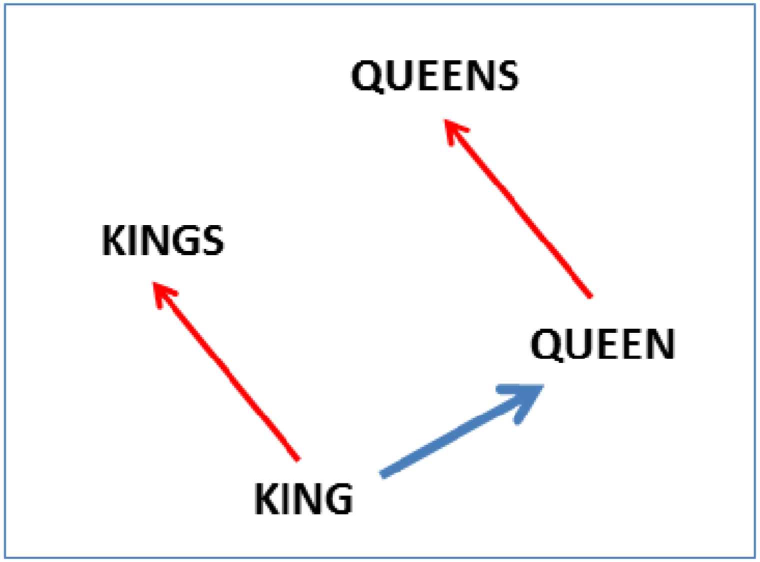
constant **female-male** difference vector

picture taken from http://www.scribd.com/doc/285890694/NIPS-DeepLearningWorkshop-NNforText#scribd

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:

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

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

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

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

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

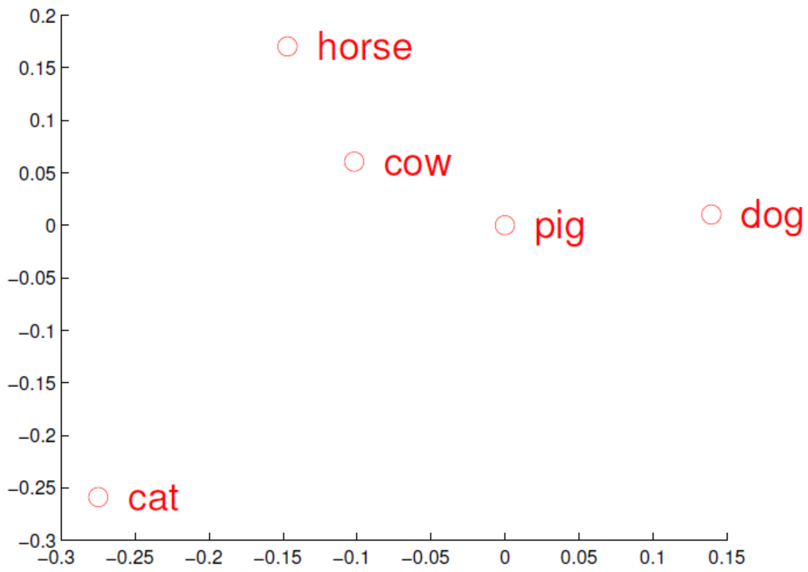
$$w_{cu} - w_{copper} + w_{gold} \cong w_{au}$$

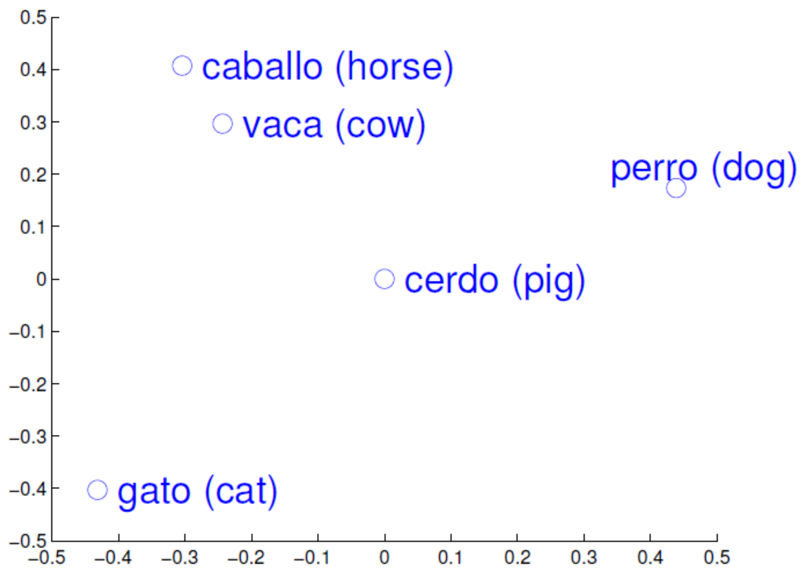
- Online demo (scroll down to end of tutorial)

picture taken from http://www.scribd.com/doc/285890694/NIPS-DeepLearningWorkshop-NNforText#scribd

Applications

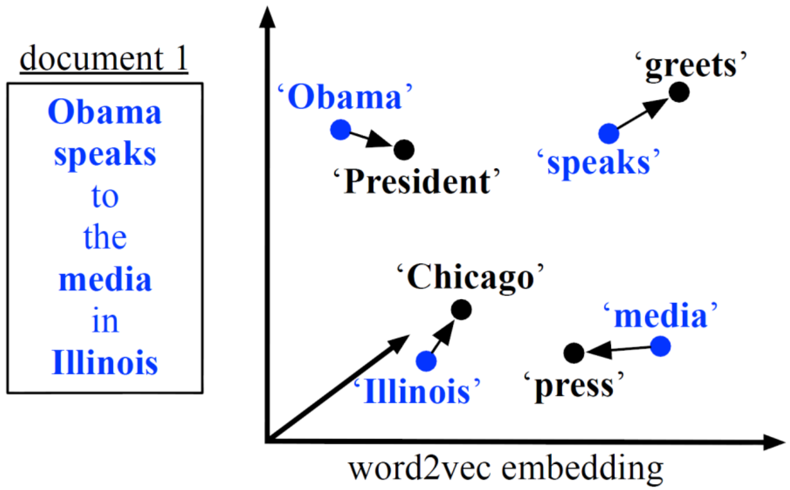
- High quality word vectors boost performance of all NLP tasks, including document classification, machine translation, information retrieval…
- Example for English to Spanish machine translation:



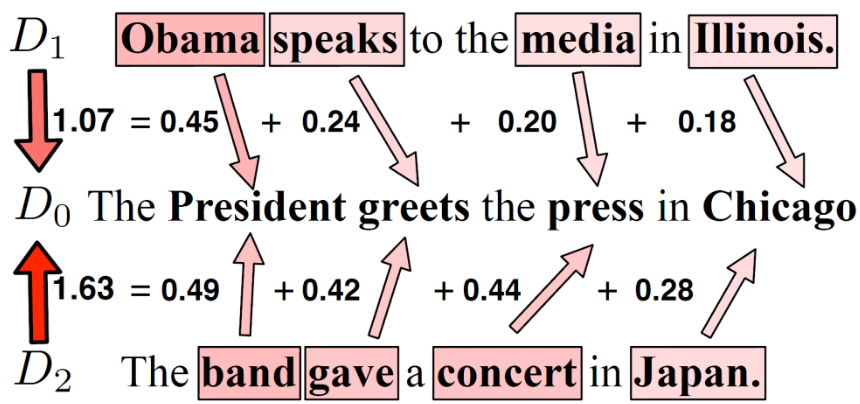


About 90% reported accuracy (Mikolov et al. 2013c)

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.



Kusner, M. J., Sun, E. Y., Kolkin, E. N. I., & EDU, W. From Word Embeddings To Document Distances. Proceedings of the 32nd International Conference on Machine Learning, Lille, France, 2015. JMLR: W&CP volume 37.

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/

After intro slides: Visualizing models

- 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` (ntlk gutenberg) and for example sentence: "Ahab boat"**
- Possible starting point: represent sentence as **sum of word vectors**
- On 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?

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

Use Case 2: Search in Ontodia / Wikidata

- **Ontodia**:
  - http://www.ontodia.org/
  - A diagramming, visual navigation tool for Linked Data (RDF, OWL, …)
  - Developed by ITMO-related VISmart company
  - In our Use Case applied to the Wikidata dataset
- **Wikidata**: www.wikidata.org
  - a free and open knowledge base readable both humans and machines.
  - Wikidata acts as central storage for the structured data of its Wikimedia sister projects including **Wikipedia**, ...

Search with Ontodia / Wikidata

- Entry point: https://wikidata.metaphacts.com/resource/Start
- Enter some search term → Ontodia → View in Ontodia
- Here you can explore the entity, add connections, etc. – give examples
- We are currently interested in exploring by properties
  - We want to search “family” related properties
  - No results for “Van Gogh”, what can we do to find **related properties**?

Search in Ontodia / Wikidata (2)

- How do Wikidata properties look like?
  - Example: https://www.wikidata.org/wiki/Property:P40
- Basic Ontodia search only matches in “label”
- What can we do?
- What could we do using word embeddings to find properties related to an input query?

Search process with embeddings

- Basically: use pretrained (on Wikipedia corpus) embeddings.
- Query-vector = AVG(vectors(query-words))
- Properties = AVG(vectors(words in label, aliases, descriptions(optional)))
- Property related to a query: word embeddings: most_similar() operation

Embeddings based system prototype

- Prototype at: http://ontodia-prop-suggest.apps.vismart.biz/wikidata.html
- Search for Van Gogh, and his “family”
- For the prototype we explored many settings:
  - Which word embedding type to use (fastText, word2vec, Glove, ..)
  - Use only terms from label + aliases, or also include description?

- Evaluation: take all aliases, and use the model to map them to related properties, winning model is the one with the highest accuracy
- Details about the prototype (ISWC2017 NLIWoD3 workshop):
  Using Word Embeddings for Visual Data Exploration with Ontodia and Wikidata
  Gerhard Wohlgenannt, Nikolay Klimov, Dmitry Mouromtsev, Daniil Razdyakonov, Dmitry Pavlov, Yury Emelyanov
  http://ceur-ws.org/Vol-1932/paper-03.pdf

Discussion

- Same method can be used to search in entites:
  - But what problems there?
- Implementation:
  - For properties ok, ca 3000 properties in Wikidata
    - We used the properties, and ca 300.000 words for user query vocabulary
  - Entities: >20M
    - Problem here?

Homework (15min to explain)

- **Part 1:** word embeddings, create and evaluate dataset for analogic reasoning and doesn’t-match
  - If you read the books (A song of Ice and Fire, or: Harry Potter) – we are happy if you use one of those and extend and evaluate this dataset
- ~~**Part 2:** apply VSM model (we learned last week) search to search in Wikidata~~