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:

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

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

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

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

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

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

"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

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):

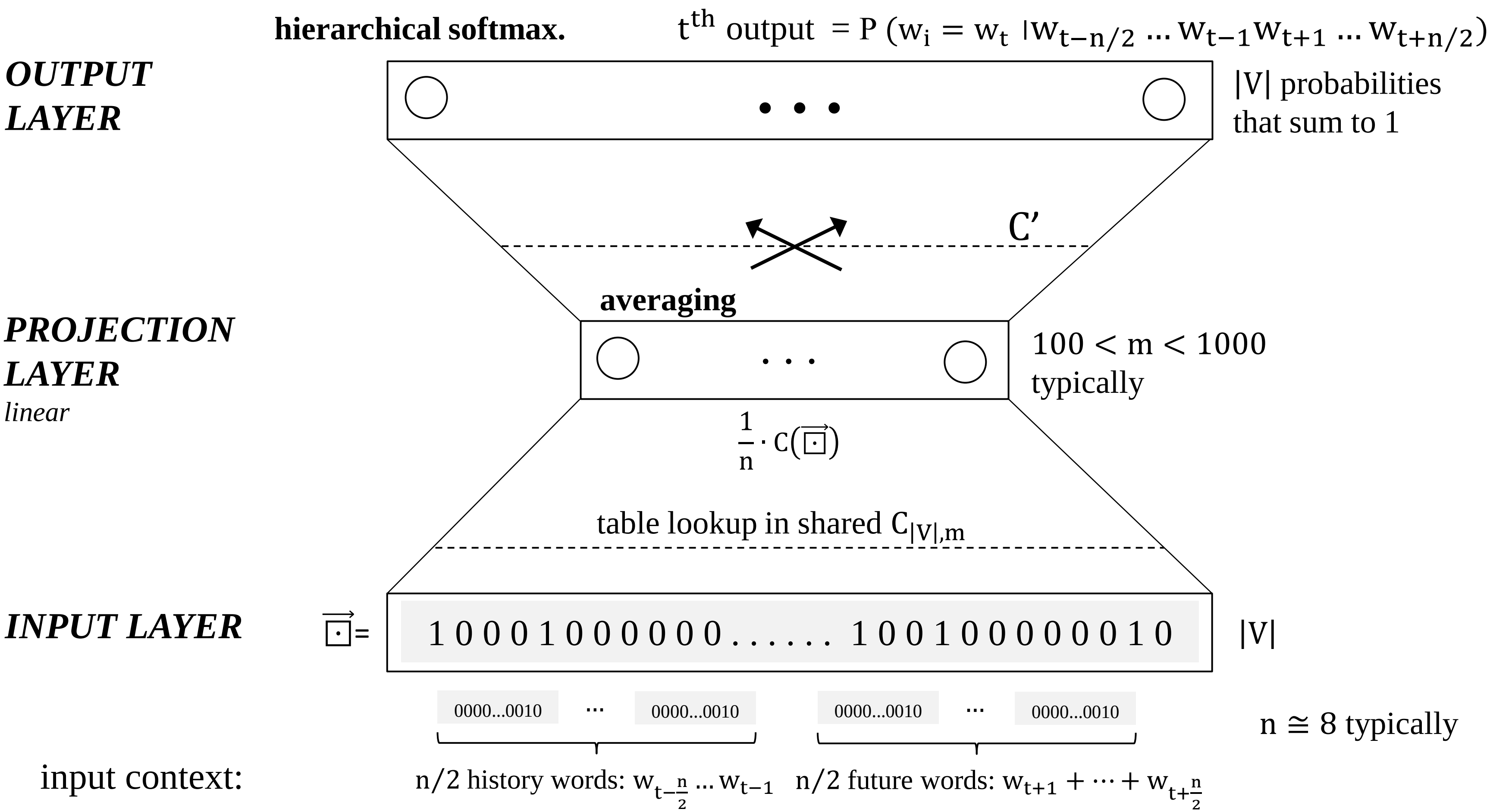
```
…Pelé has called Neymar an excellent player…
…At the age of just 22 years, Neymar had scored 40 goals in 58 internationals…
…occasionally as an attacking midfielder, Neymar was called a true phenomenon…
```

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

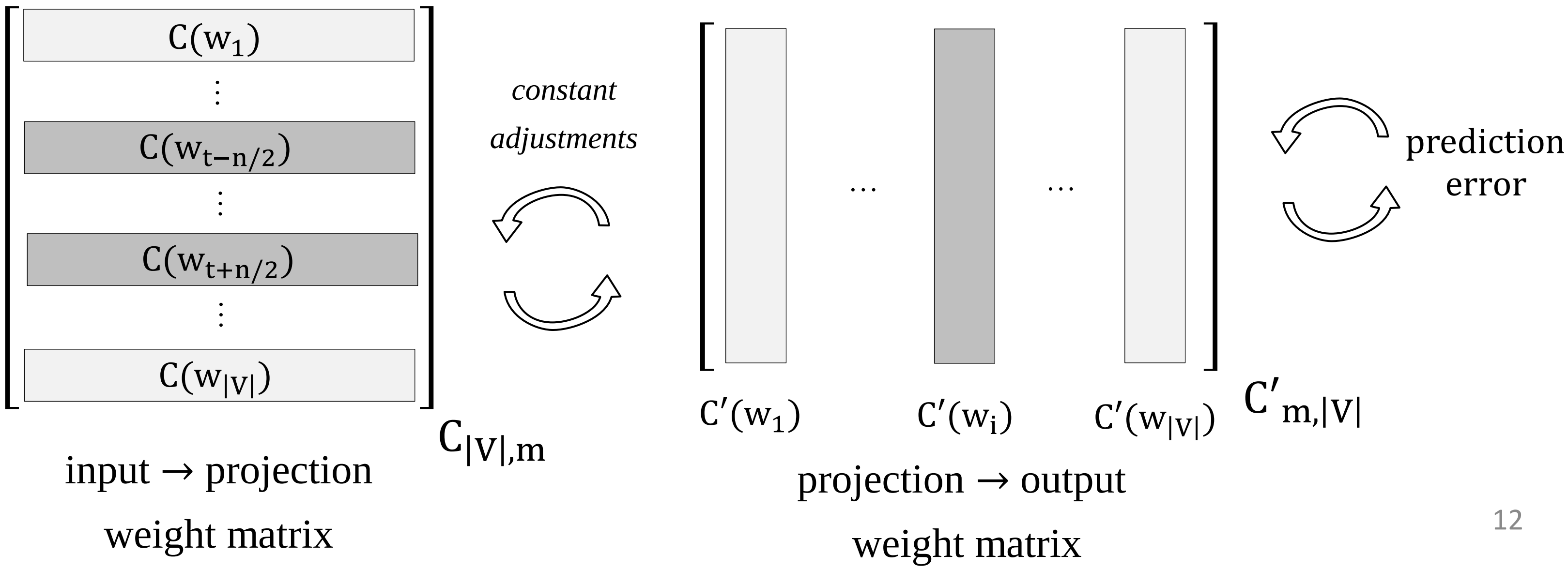


Weight updating intuition

- 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 ≅ 100K-5M words/s
- Quality of the word vectors:
  - ↗ 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:

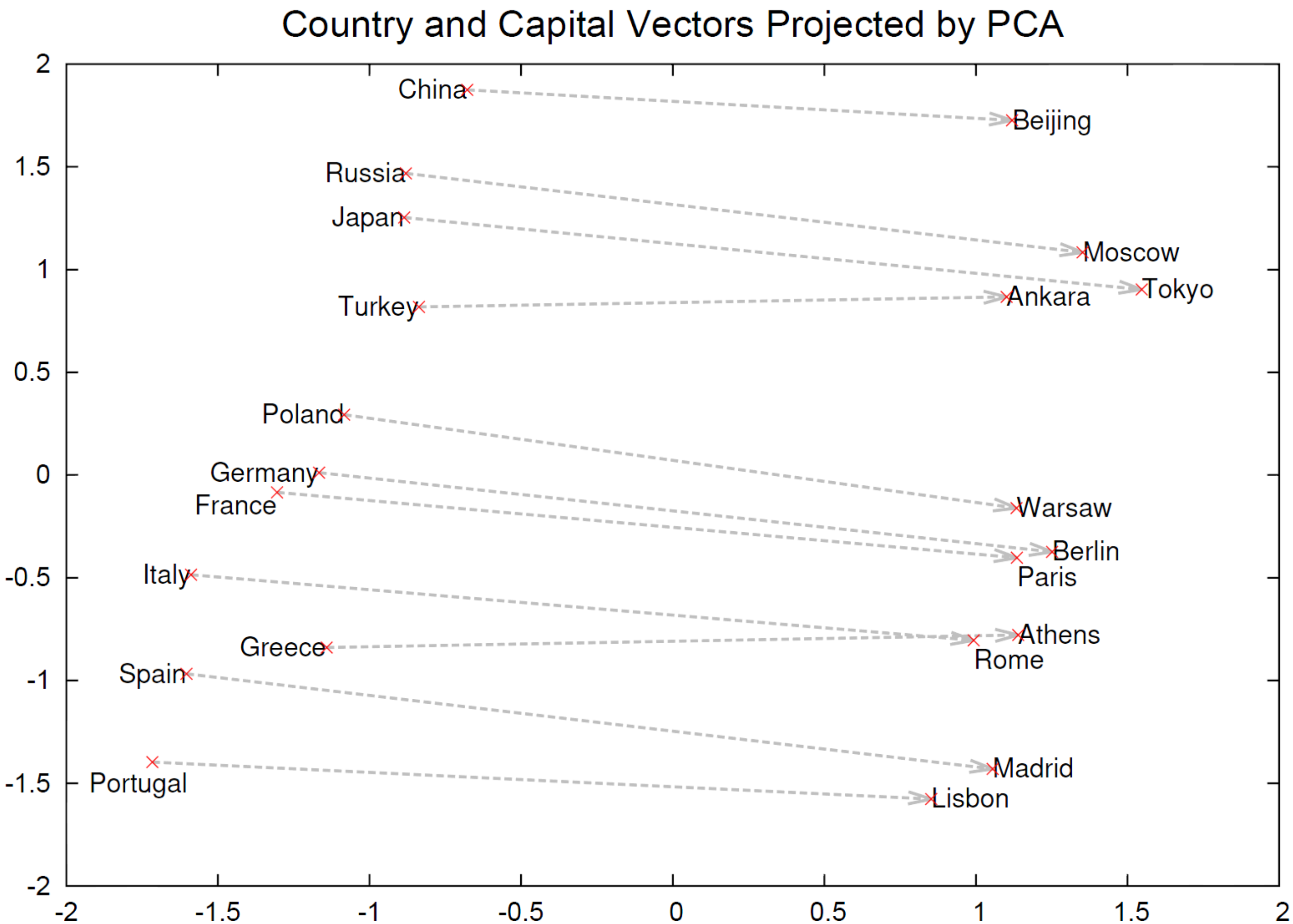
Capital-Country	Past tense	Superlative	Male-Female	Opposite
Athens: **Greece**	walking: **walked**	easy: **easiest**	brother: **sister**	ethical: **unethical**

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

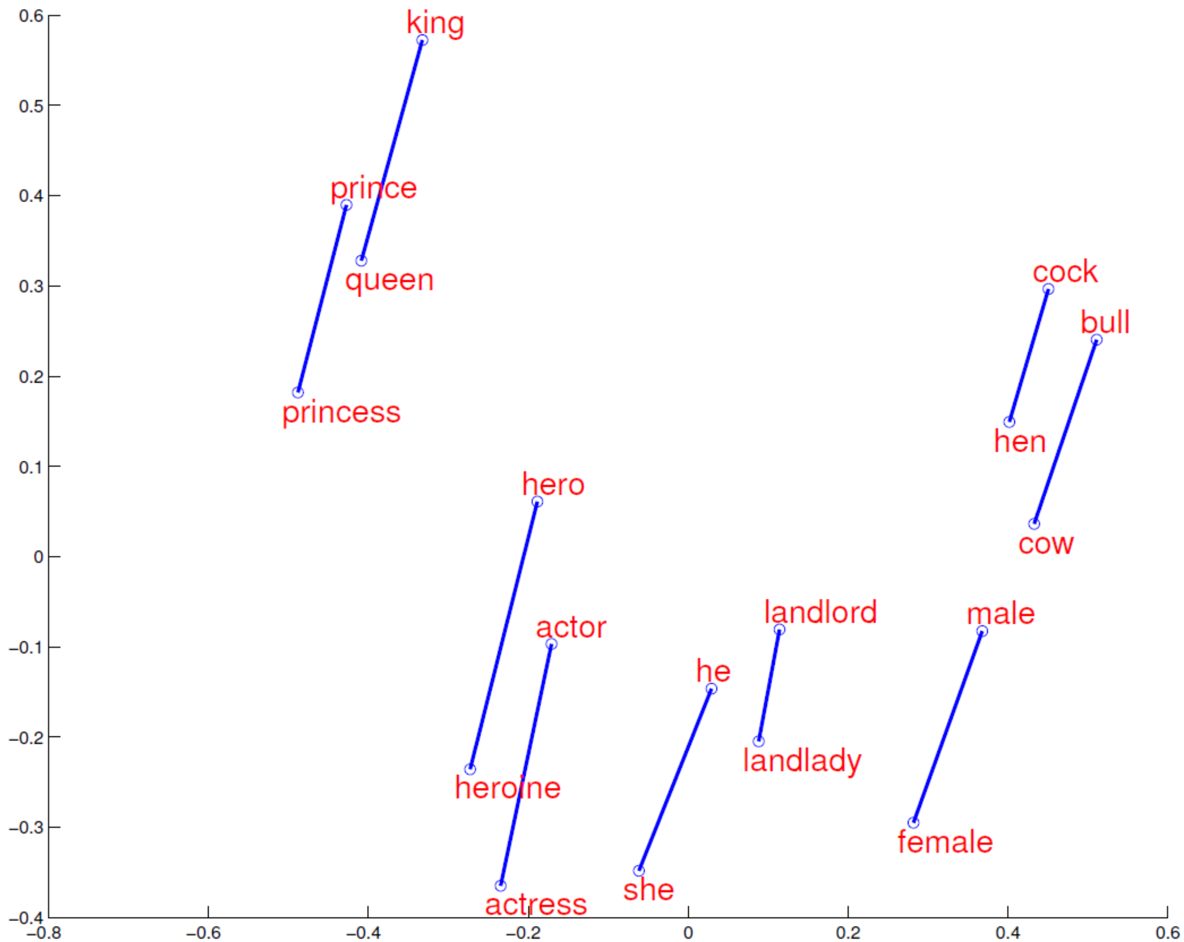


Mikolov et al. (2013b)

regularities between words are encoded in the difference vectors

e.g., there is a constant **country-capital** difference vector

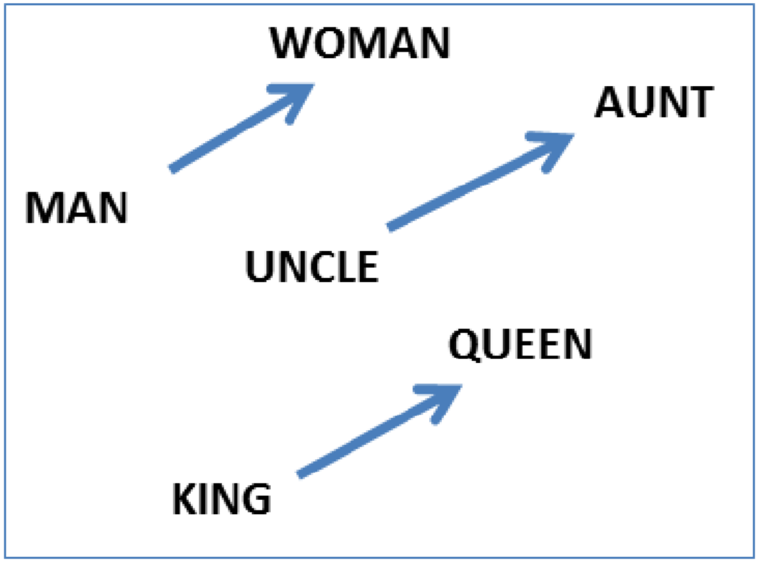
Remarkable properties of word2vec's word vectors



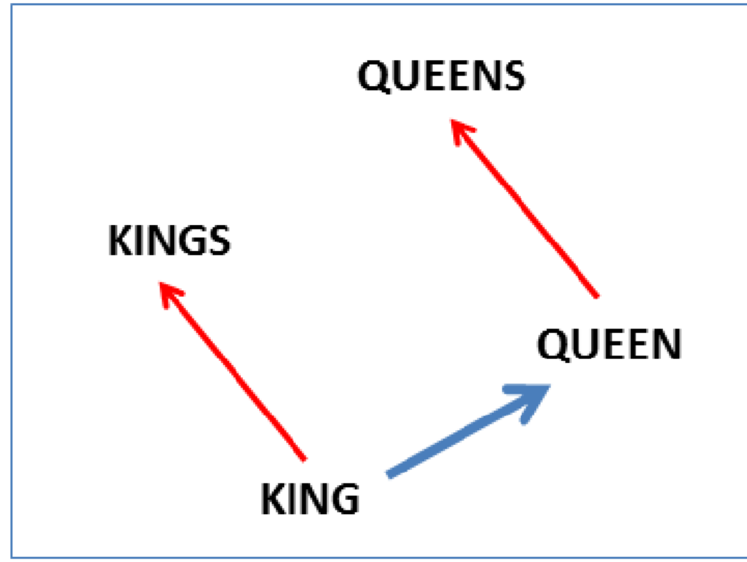
constant **female-male** difference vector

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:

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

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

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

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

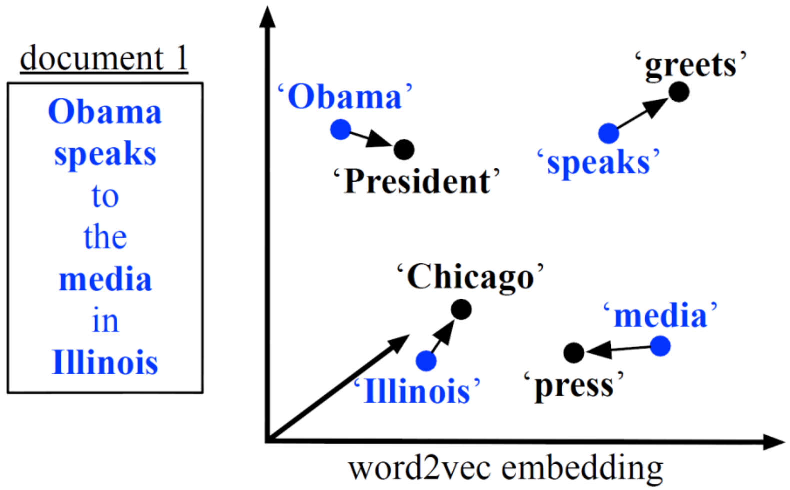
$$w_{windows} - w_{microsoft} + w_{google} \cong w_{android}$$

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

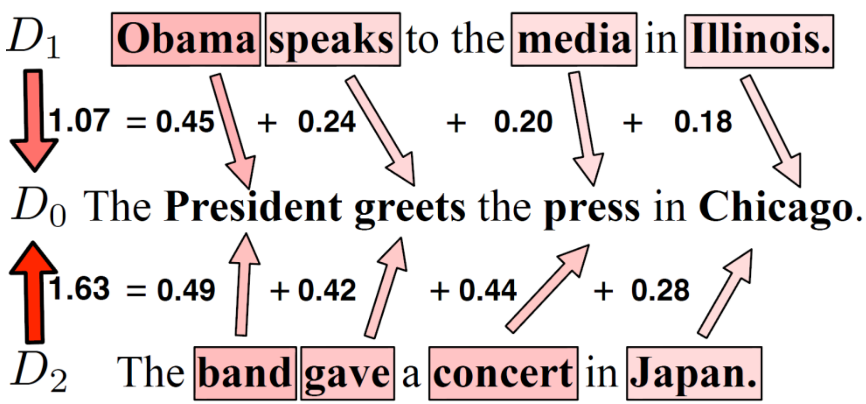
- Online demo (scroll down to end of tutorial)

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

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.]