Word Embedding

M. Soleymani
Sharif University of Technology
Fall 2017

Many slides have been adopted from Socher lectures, cs224d, Stanford, 2017
and some slides from Hinton slides, “Neural Networks for Machine Learning”, coursera, 2015.

One-hot coding

In vector space terms, this is a vector with one 1 and a lot of zeroes

[0 0 0 0 0 0 0 0 0 0 1 0 0 0 0]

Dimensionality: 20K (speech) – 50K (PTB) – 500K (big vocab) – 13M (Google 1T)

Distributed similarity based representations

- representing a word by means of its neighbors
- "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

government debt problems turning into	banking	crises as has happened in
saying that Europe needs unified	banking	regulation to replace the hodgepodge

↖ These words will represent *banking* ↗

Word embedding

- Store "most" of the important information in a fixed, small number of dimensions: a dense vector
  - Usually around 25 – 1000 dimensions

- Embeddings: distributional models with dimensionality reduction, based on prediction

How to make neighbors represent words?

- Answer: With a co-occurrence matrix X
  - options: **full document** vs **windows**

- **Full** word-document co-occurrence matrix
  - will give general topics (all sports terms will have similar entries) leading to "Latent Semantic Analysis"

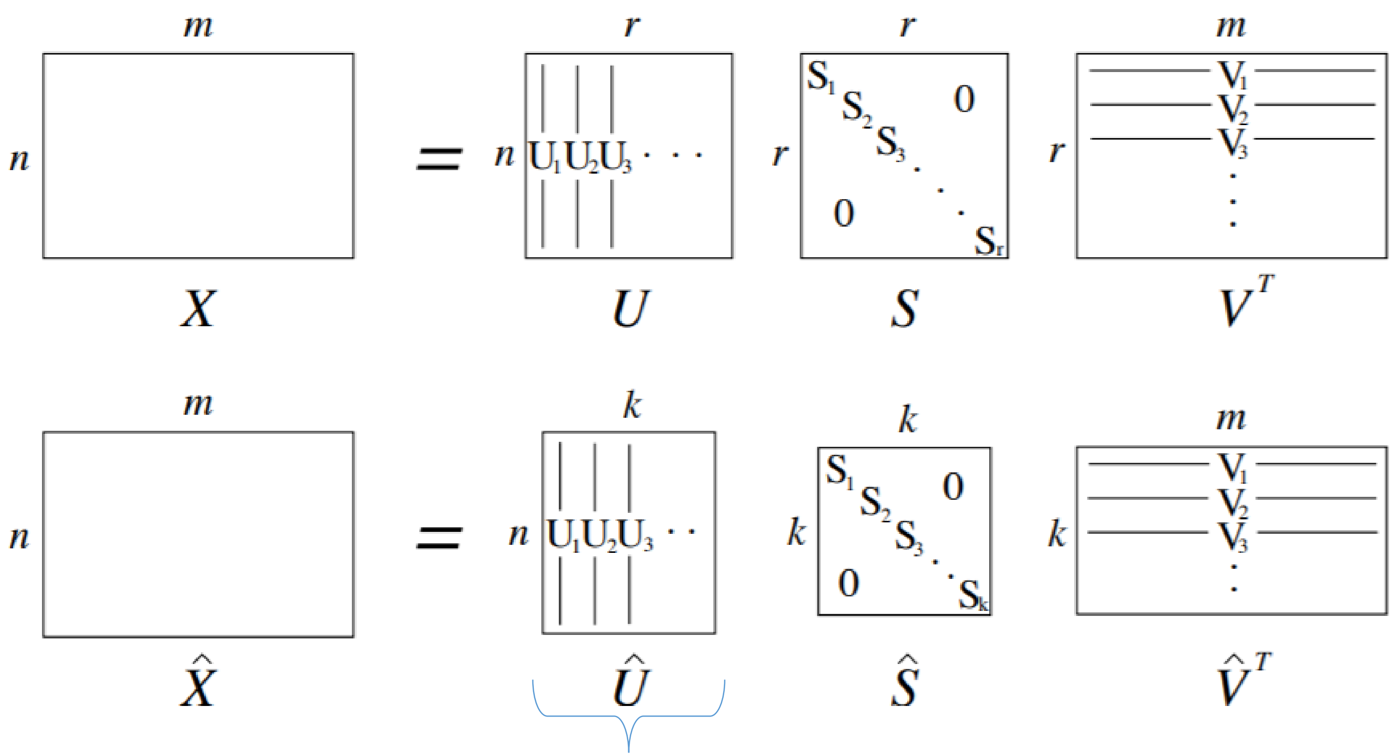
- **Window** around each word
  - captures both syntactic (POS) and semantic information

LSA: Dimensionality Reduction based on word-doc matrix

Singular Value Decomposition of cooccurrence matrix *X*.

Docs

words



Maintaining only the k largest singular values of X

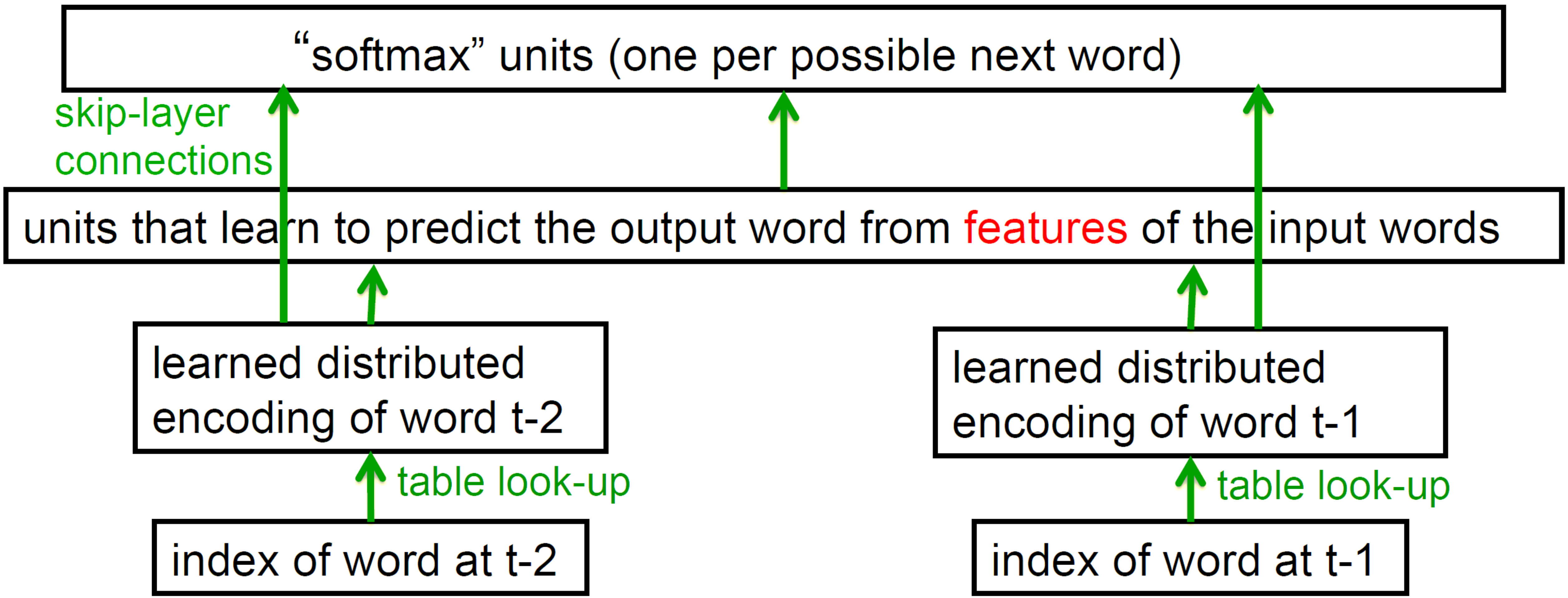
Embedded words

$\hat{X}$ is the best rank *k* approximation to *X* , in terms of least squares.

Directly learn low-dimensional word vectors

- Old idea. Relevant for this lecture:
  - Learning representations by back-propagating errors. (Rumelhart et al., 1986)
  - NNLM: A neural probabilistic language model (Bengio et al., 2003)
  - NLP (almost) from Scratch (Collobert & Weston, 2008)
  - A recent, even simpler and faster model: word2vec (Mikolov et al. 2013)-> intro now

NNLM: Trigram (Language Modeling)



Bengio et al., NNLM: A neural probabilistic language model, 2003.

NNLM

- Semantic and syntactic features of previous words can help to predict the features of the next word.

- Word embedding in the NNLM model helps us to find similarities between pairs of words

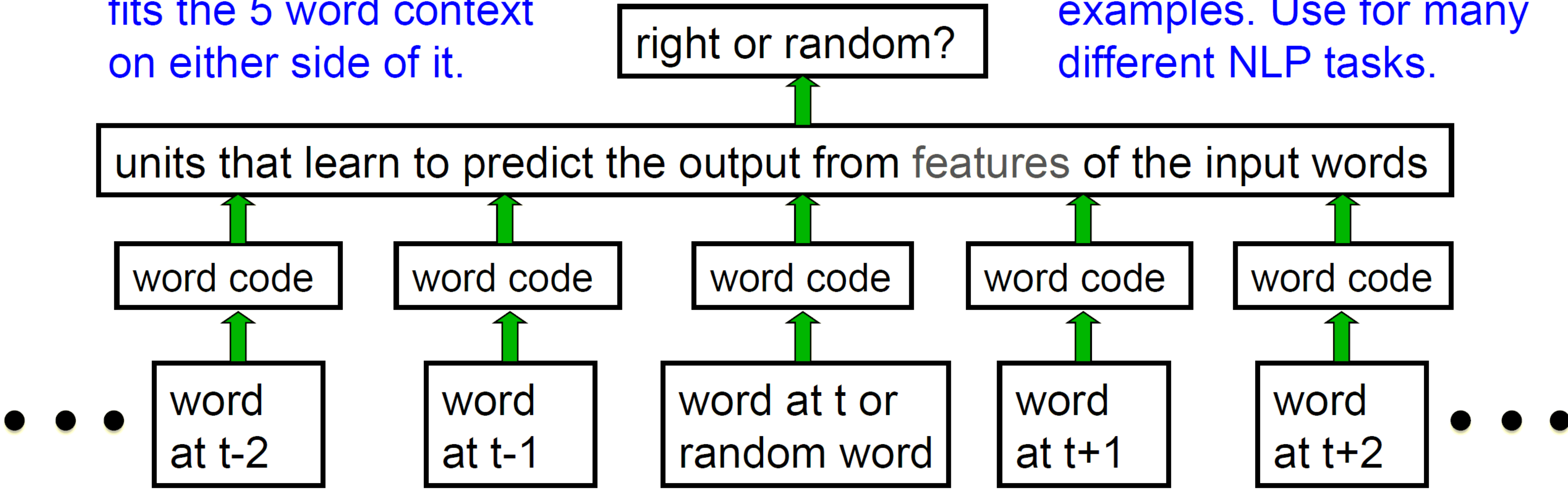
“the cat got squashed in the garden on friday”

“the dog got flattened in the yard on monday”

A simpler way to learn feature vectors for words

Learn to judge if a word fits the 5 word context on either side of it.

Train on ~600 million examples. Use for many different NLP tasks.



Collobert and Weston, A Unified Architecture for Natural Language Processing: Deep Neural Networks with Multitask Learning, ICML 2008.

Part of a 2-D map of the 2500 most common words

rather
increasingly
further
later
otherwise
which
that
entirely
completely
newly
fully
greatly
what
whom
far
only
just
both
how
whether
why
either
even
ever
briefly
already
heavily
easily
quickly
successfully
well
closely
widely
directly
then
once
yet
officially
commonly
frequently
specifically
regularly
initially
originally
actually
simply
nor
but
as
if
so
where
soon
shortly
immediately
never
n't
largely
currently
now
also
still
not
primarily
mainly
mostly
because
when
though
although
while
whilst
generally
eventually
again
normally
usually
typically
apparently
subsequently
ultimately
finally
particularly
especially
formerly
often
occasionally
sometimes
before
twice
instead
meanwhile
notably
except
probably
likely
possibly
perhaps
thus
therefore
however
hence
here
today
there
afterwards
ago
none

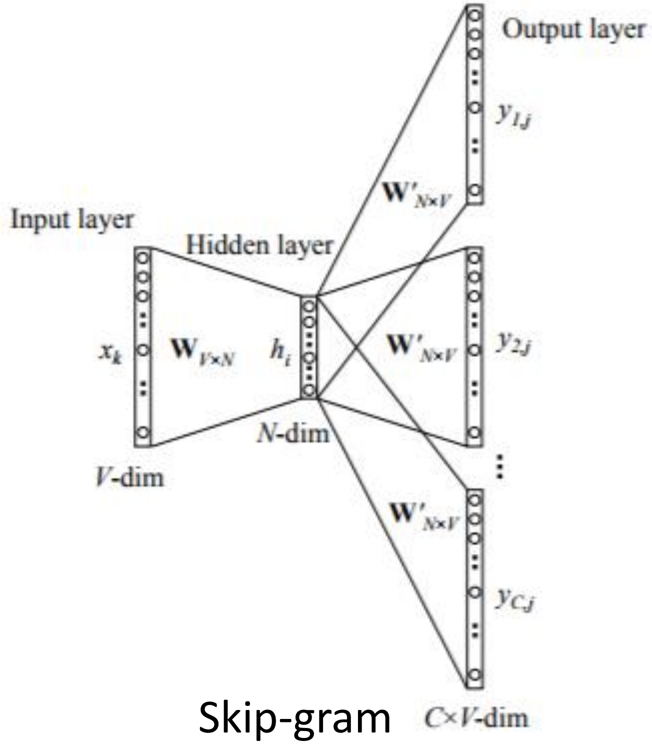
Word2vec embedding

- word2vec: as originally described (Mikolov et al 2013), a NN model using a two-layer network (i.e., not deep!) to perform dimensionality reduction.

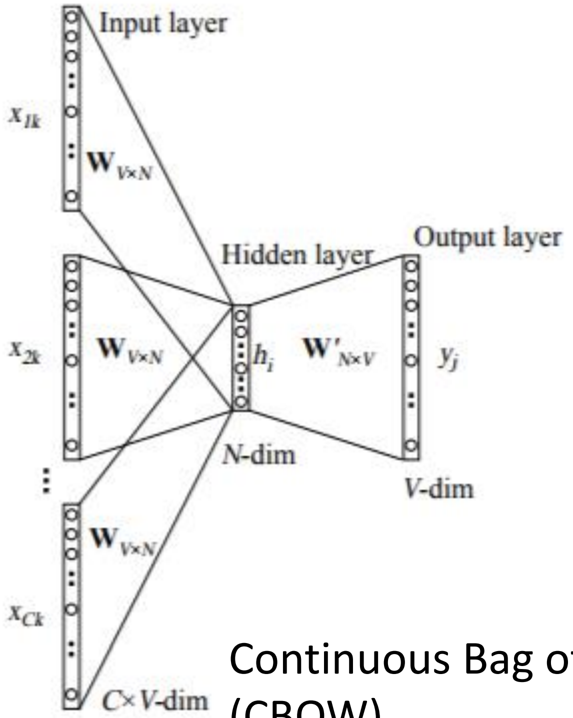
- Very computationally efficient, good all-round model (good hyperparameters already selected).

Skip-gram vs. CBOW

- Two possible architectures:
  - given some context words, predict the center (CBOW)
    - Predict center word from sum of surrounding word vectors
  - given a center word, predict the contexts (Skip-gram)



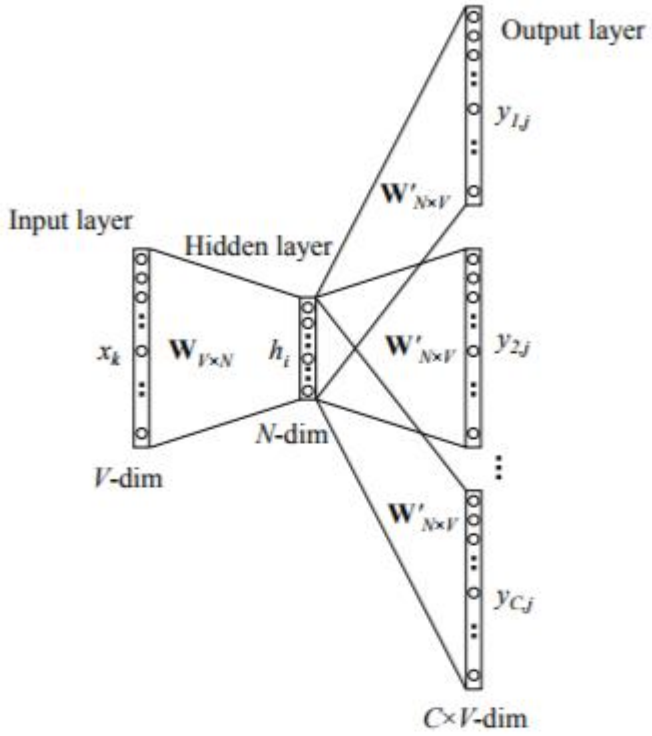
Skip-gram



Continuous Bag of words (CBOW)

Skip-gram

- Embeddings that are good at predicting neighboring words are also good at representing similarity



Details of Word2Vec

- Learn to predict surrounding words in a window of length m of every word.

- Objective function: Maximize the log probability of any context word given the current center word:

$$J(\theta) = \frac{1}{T}\sum_{t=1}^{T}\sum_{\substack{-m\le j\le m \\ j\ne 0}} \log p\left(w_{t+j}|w_t\right)$$

T: training set size
m: context size
w_j : vector representation of the jth word
θ: whole parameters of the network

- Use a large training corpus to maximize it

m is usually 5~10

Word embedding matrix

- You will get the word-vector by left multiplying a one-hot vector by W

$$W = \left[\begin{array}{ccc} \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet \\ \bullet & \bullet & \bullet \end{array}\right] \begin{array}{l} \text{a} \\ \text{Aardvark} \\ \dots \\ \\ \text{zebra} \end{array}$$

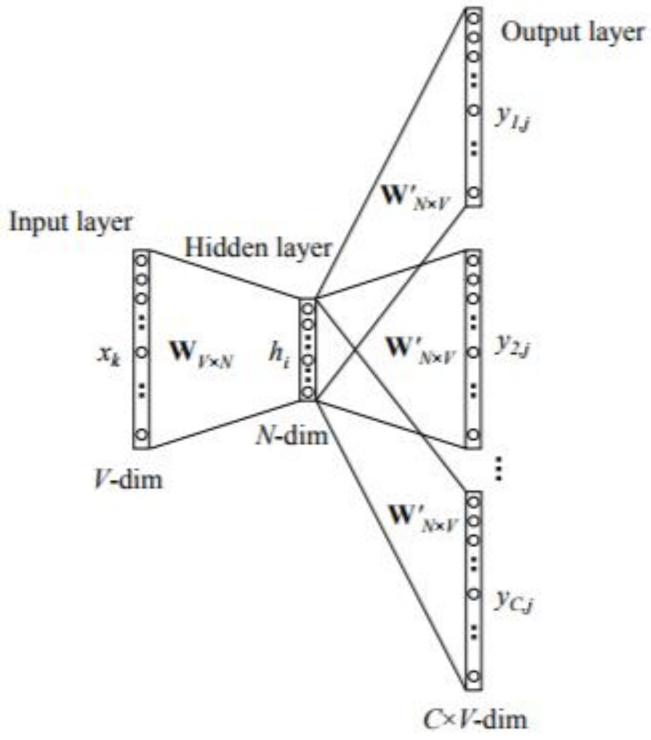
$$x = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \\ 0 \end{bmatrix} \quad (x_k = 1)$$

$$h = x^T W = W_{k,.} = v_k$$

k-th row of the matrix W

Skip-gram

- w_o: context or output (outside) word
- w_I: center or input word



$$score(w_o, w_I) = h^T W'_{.,o} = v_I^T u_o$$
$$h = x_I^T W = W_{I,.} = v_I$$
$$W'_{.,o} = u_o$$

$$P(w_o|w_I) = \frac{e^{u_o^T v_I}}{\sum_i \ e^{u_k^T v_I}}$$

Every word has 2 vectors
v_W: when w is the center word
u_W: when w is the outside word (context word)

Details of Word2Vec

- Predict surrounding words in a window of length m of every word:

$$P(w_o|w_I) = \frac{e^{u_o^T v_I}}{\sum_{i=1}^{|V|} e^{u_i^T v_I}}$$

$$J(\theta) = \frac{1}{T}\sum_{t=1}^{T} \sum_{\substack{-m\le j\le m \\ j\ne 0}} \log p\left(w_{t+j}|w_t\right)$$

Parameters

- We often define the set of ALL parameters in a model in terms of one long vector θ
- In our case with d-dimensional vectors and V many words:

$$\theta = \begin{bmatrix} v_{aardvark} \\ v_a \\ \vdots \\ v_{zebra} \\ u_{aardvark} \\ u_a \\ \vdots \\ u_{zebra} \end{bmatrix} \in \mathbb{R}^{2dV}$$

Gradient

$$\frac{\partial \log p(w_o|w_I)}{\partial v_I} = \frac{\partial}{\partial v_I} \log \frac{e^{u_o^T v_I}}{\sum_{x=1}^{|V|} e^{u_x^T v_I}}$$

$$= \frac{\partial}{\partial v_I} \left(\log e^{u_o^T v_I} - \log \sum_{x=1}^{|V|} e^{u_x^T v_I} \right)$$

$$= u_o - \frac{1}{\sum_{x=1}^{|V|} e^{u_x^T v_I}} \sum_{x=1}^{|V|} u_x \, e^{u_x^T v_I}$$

$$= u_o - \sum_{x=1}^{|V|} p(w_x|w_I) u_x$$

Training difficulties

- With large vocabularies, it is not scalable!

$$\frac{\partial \log p(w_o|w_I)}{\partial v_I} = u_o - \sum_{x=1}^{|V|} p(w_x|w_I)u_x$$

- Define negative prediction that only samples a few words that do not appear in the context
  - Similar to focusing on mostly positive correlations

Negative sampling

- k is the number of negative samples

$$\log \sigma(u_o^T v_I) + \sum_{w_j \sim P(w)} \log \sigma\left(-u_j^T v_I\right)$$

- Maximize probability that real outside word appears, minimize prob. that random words appear around center word
- $P(w) = U(w)^{\frac{3}{4}}/Z$
  - the unigram distribution U(w) raised to the 3/4rd power.
  - The power makes less frequent words be sampled more often

Mikolov et al., Distributed Representations of Words and Phrases and their Compositionality, 2013.

What to do with the two sets of vectors?

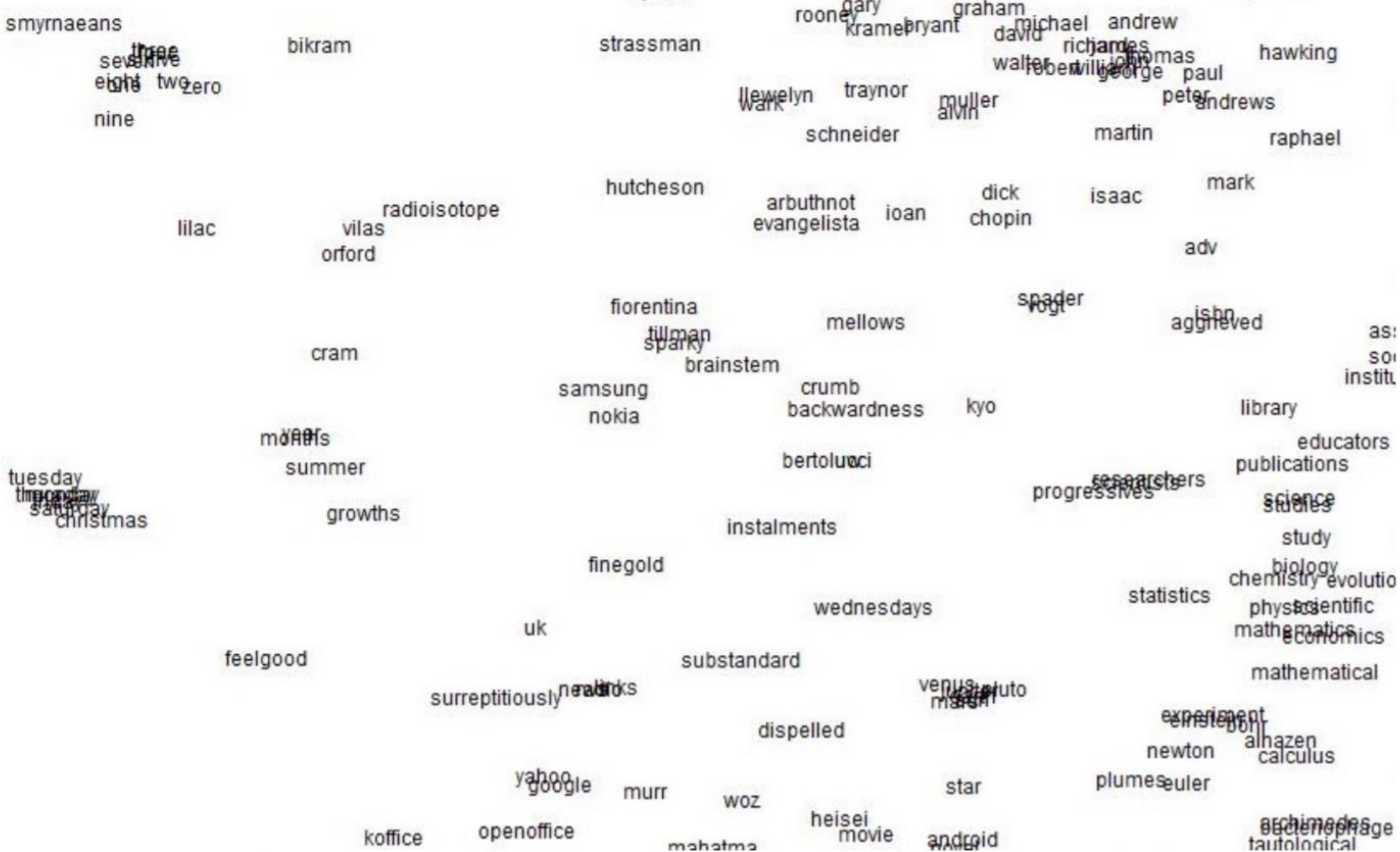
- We end up with U and V from all the vectors u and v (in columns)
  - Both capture similar co-occurrence information.

- The best solution is to simply sum them up:

$$Xfinal = U + V$$

- One of many hyperparameters explored in GloVe

Pennington et al., Global Vectors for Word Representation, 2014.


smyrnaeans
bikram
strassman
rooney
gary
kramer
bryant
graham
david
michael
andrew
walter
paul
hawking
nine
zero
llewelyn
traynor
muller
peter
andrews
schneider
martin
raphael
hutcheson
arbuthnot
evangelista
ioan
dick
chopin
isaac
mark
radioisotope
lilac
vilas
orford
adv
fiorentina
mellows
brainstem
cram
samsung
nokia
crumb
backwardness
kyo
library
educators
publications
summer
tuesday
christmas
growths
instalments
progressives
study
finegold
statistics
chemistry
wednesdays
uk
mathematical
feelgood
substandard
surreptitiously
dispelled
newton
calculus
murr
woz
star
plumes
euler
heisei
movie
android
koffice
openoffice

Summary of word2vec

- Go through each word of the whole corpus
- Predict surrounding words of each word
- This captures co-occurrence of words one at a time
- Why not capture co-occurrence counts directly?

Window based co-occurrence matrix: Example

Corpus

- I like deep learning.
- I like NLP.
- I enjoy flying.

Window length 1 (more common: 5 - 10)
Symmetric (irrelevant whether lel or right context)

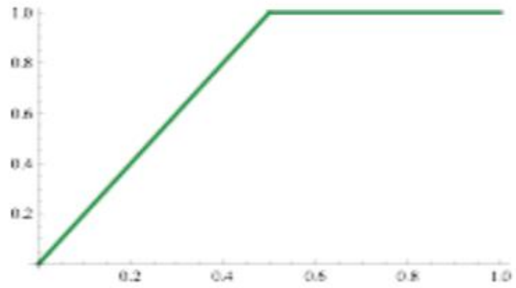
P

counts	I	like	enjoy	deep	learning	NLP	flying	.
I	0	2	1	0	0	0	0	0
like	2	0	0	1	0	1	0	0
enjoy	1	0	0	0	0	0	1	0
deep	0	1	0	0	1	0	0	0
learning	0	0	0	1	0	0	0	1
NLP	0	1	0	0	0	0	0	1
flying	0	0	1	0	0	0	0	1
.	0	0	0	0	1	1	1	0

GloVe

$$J(\theta) = \frac{1}{2}\sum_{i,j=1}^{W} f(P_{ij})(u_i^T v_j - \log P_{ij})^2$$

$f \sim$



- Fast training
- Scalable to huge corpora
- Good performance even with small corpus, and small vectors

Pennington et al., Global Vectors for Word Representation, 2014.

How to evaluate word vectors?

- Related to general evaluation in NLP: Intrinsic vs extrinsic
- Intrinsic:
  - Evaluation on a specific/intermediate subtask
  - Fast to compute
  - Helps to understand that system
  - Not clear if really helpful unless correlation to real task is established
- Extrinsic:
  - Evaluation on a real task
  - Can take a long time to compute accuracy
  - Unclear if the subsystem is the problem or its interaction or other subsystems
  - If replacing exactly one subsystem with another improves accuracy -> Winning!

Intrinsic evaluation by word analogies

- Performance in completing word vector analogies:

$$d = \operatorname*{argmax}_i \frac{(x_b - x_a + x_c)^T x_i}{\|x_b - x_a + x_c\|}$$

- Evaluate word vectors by how well their cosine distance after addition captures intuitive semantic and syntactic analogy questions
- Discarding the input words from the search!
- Problem: What if the information is there but not linear?

Mikolov et al., Distributed Representations of Words and Phrases and their Compositionality, 2013.

Word Analogies

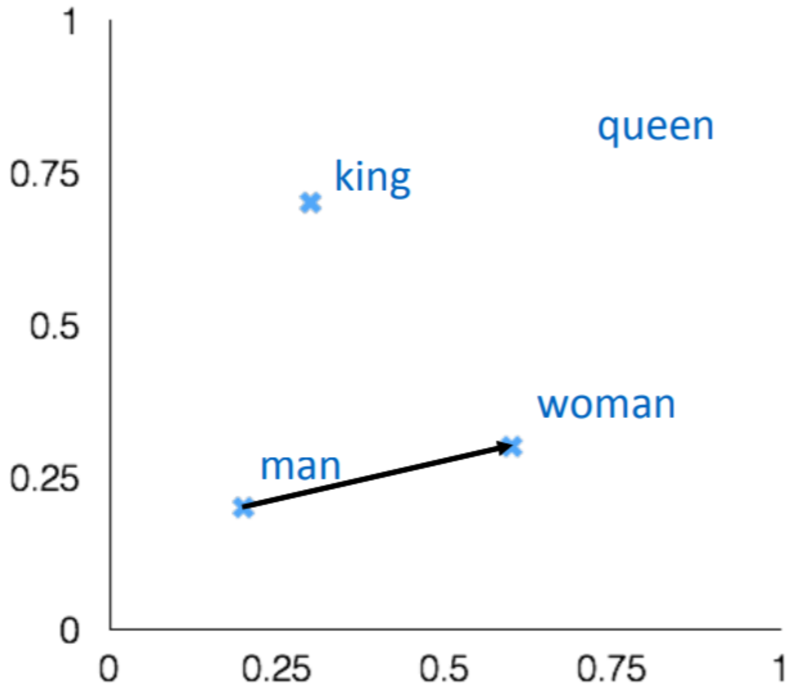
Test for linear relationships, examined by Mikolov et al. (2014)

a:b :: c:?

$$d = \arg\max_x \frac{(w_b - w_a + w_c)^T w_x}{||w_b - w_a + w_c||}$$

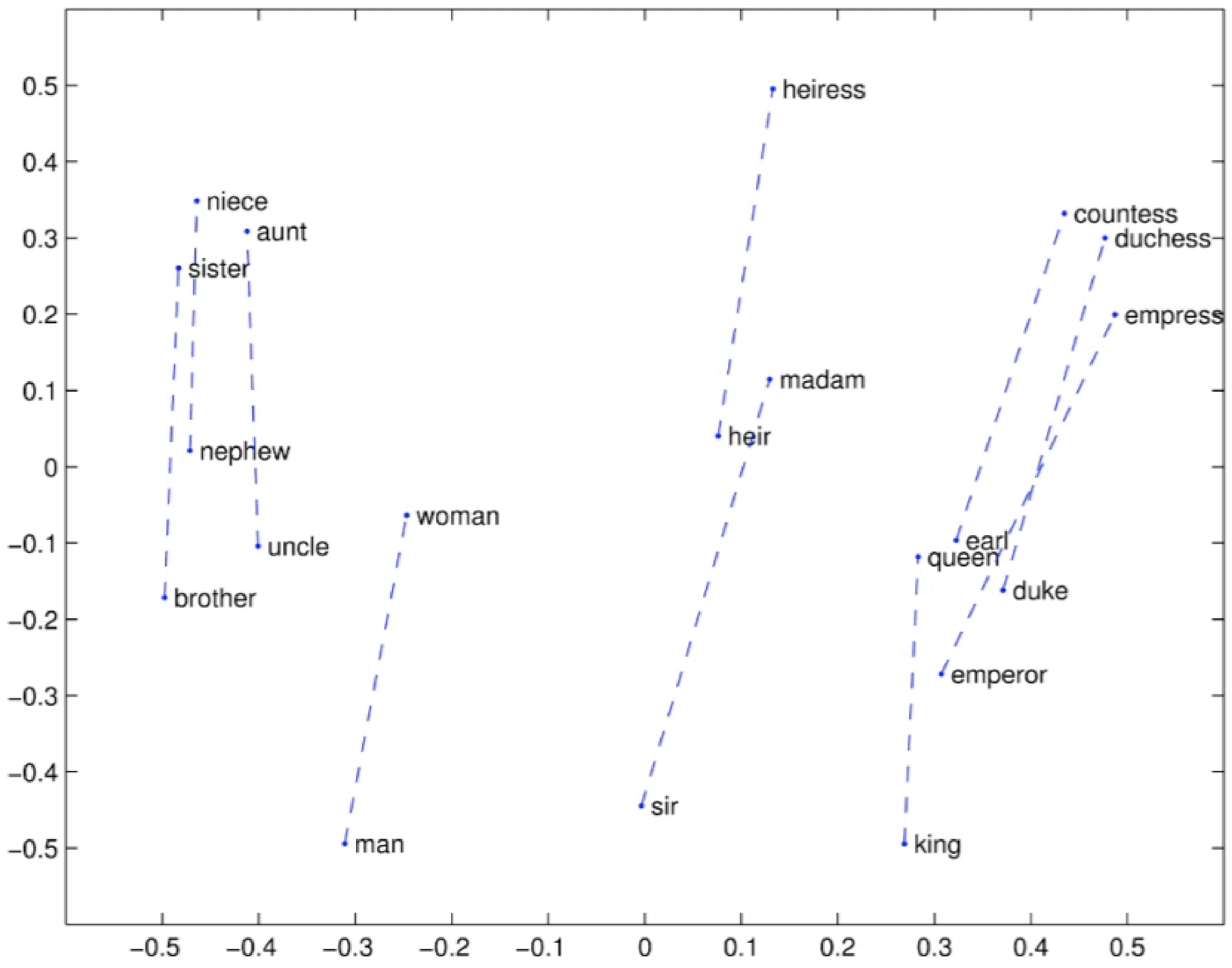
man:woman :: king:?

+	king	[0.30 0.70]
-	man	[0.20 0.20]
+	woman	[0.60 0.30]
	queen	[0.70 0.80]

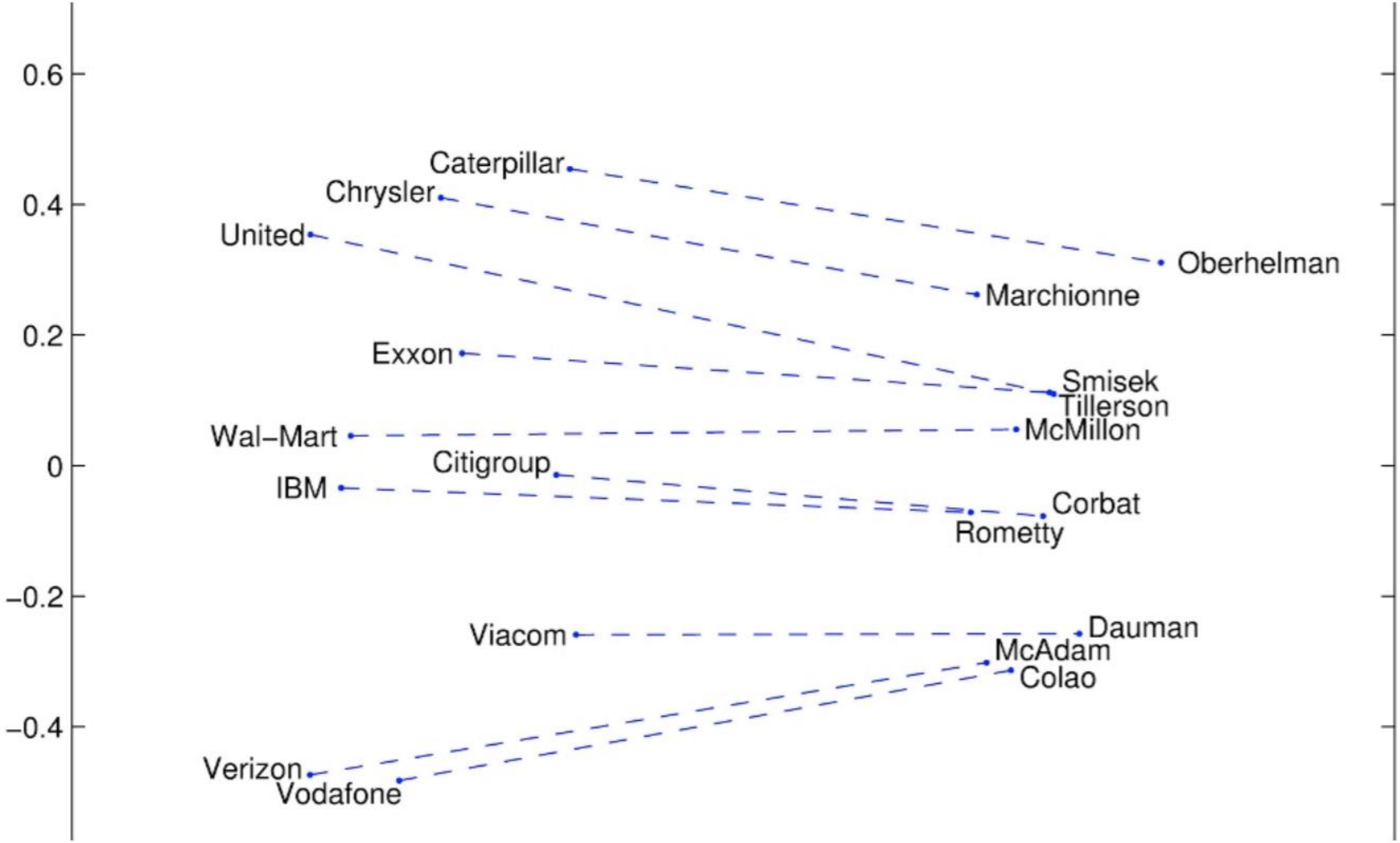


The linearity of the skip-gram model makes its vectors more suitable for such linear analogical reasoning

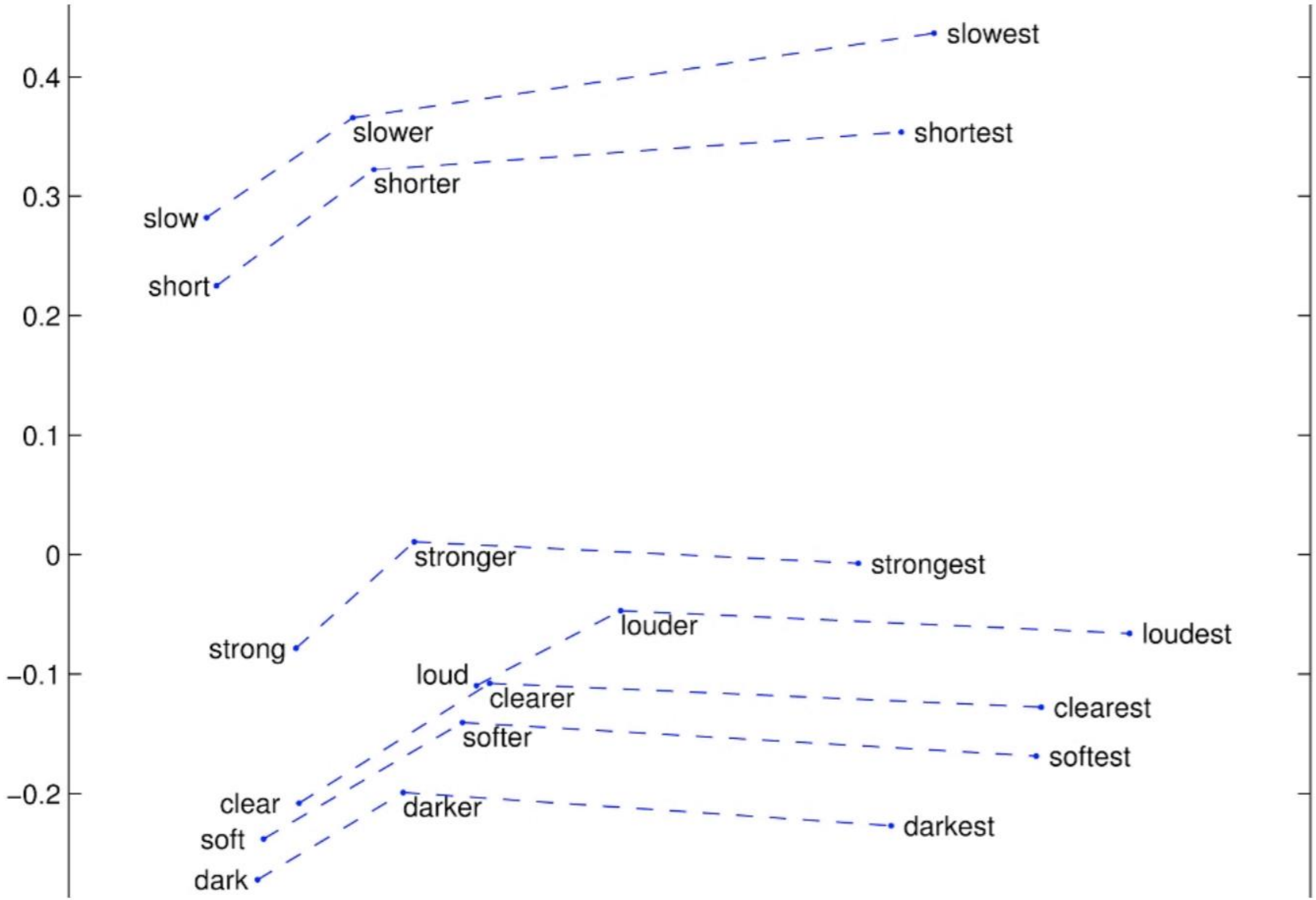
Visualizations



GloV Visualizations: Company - CEO



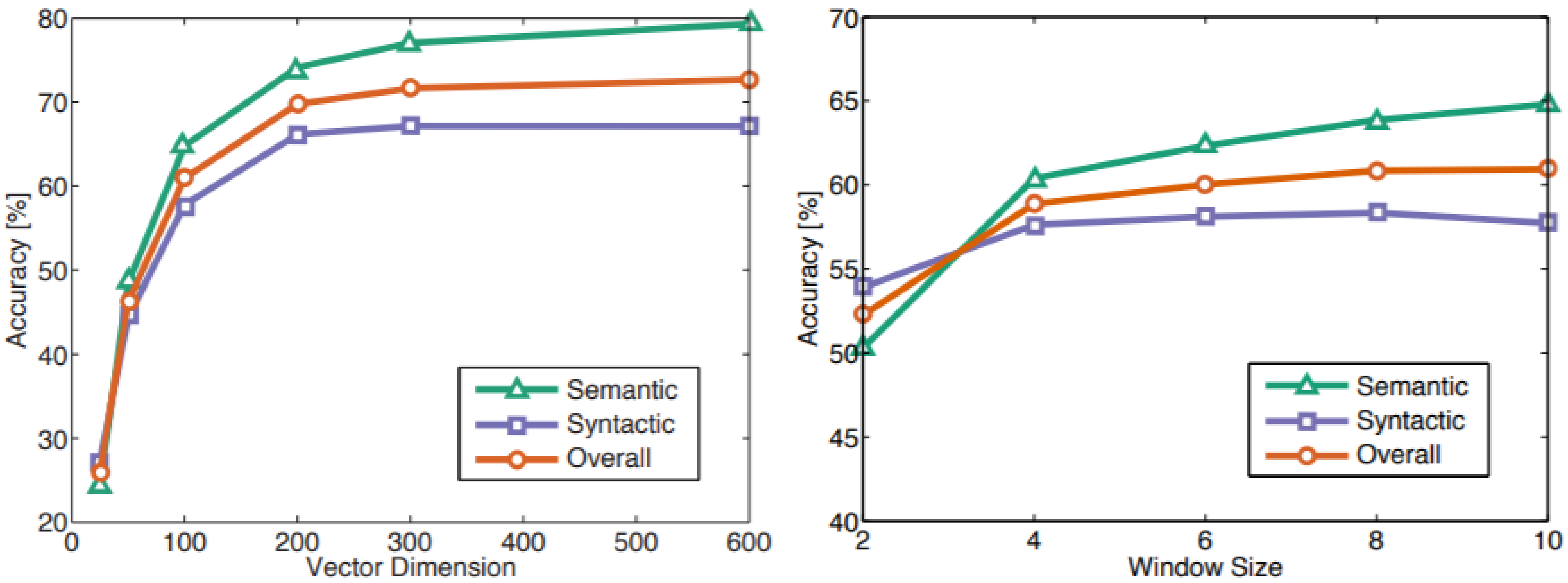
Glov Visualizations: Superlatives



Other fun word2vec analogies

Expression	*Nearest token*
Paris - France + Italy	Rome
bigger - big + cold	colder
sushi - Japan + Germany	bratwurst
Cu - copper + gold	Au
Windows - Microsoft + Google	Android
Montreal Canadiens - Montreal + Toronto	Toronto Maple Leafs

Analogy evaluation and hyperparameters



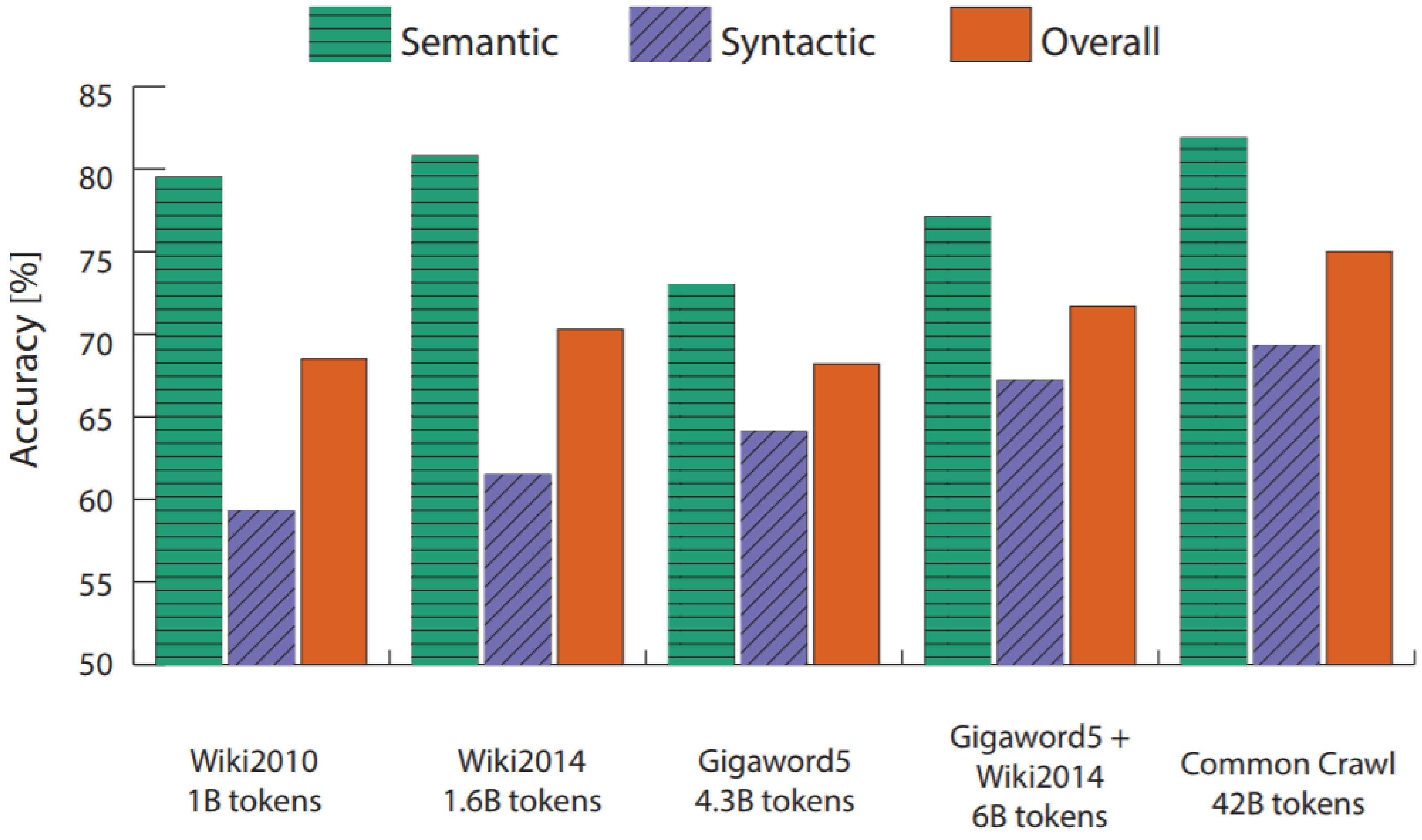
- Best dimensions ~300, slight drop-off afterwards
- But this might be different for downstream tasks!

- Window size of 8 around each center word is good for Glove vectors

Pennington et al., Global Vectors for Word Representation, 2014.

Analogy evaluation and hyperparameters

- More data helps, Wikipedia is better than news text!



Pennington et al., Global Vectors for Word Representation, 2014.

Another intrinsic word vector evaluation

- Word vector distances and their correlation with human judgments
- Example dataset: WordSim353
  http://www.cs.technion.ac.il/~gabr/resources/data/wordsim353/

Word 1	Word 2	Human (mean)
tiger	cat	7.35
tiger	tiger	10.00
book	paper	7.46
computer	internet	7.58
plane	car	5.77
professor	doctor	6.62
stock	phone	1.62
stock	CD	1.31
stock	jaguar	0.92

Closest words to “Sweden” (cosine similarity)

Word	Cosine distance
norway	0.760124
denmark	0.715460
finland	0.620022
switzerland	0.588132
belgium	0.585835
netherlands	0.574631
iceland	0.562368
estonia	0.547621
slovenia	0.531408

Extrinsic word vector evaluation

- Extrinsic evaluation of word vectors: All subsequent NLP tasks can be considered as down stream task
- One example where good word vectors should help directly: named entity recognition
  - finding a person, organization or location

Example: word classification

- Two options: train only classifier and fix word vectors or also train word vectors

- Question: What are the advantages and disadvantages of training the word vectors?
  - Pro: better fit on training data
  - Con: Worse generalization because the words move in the vector space

Example: word classification

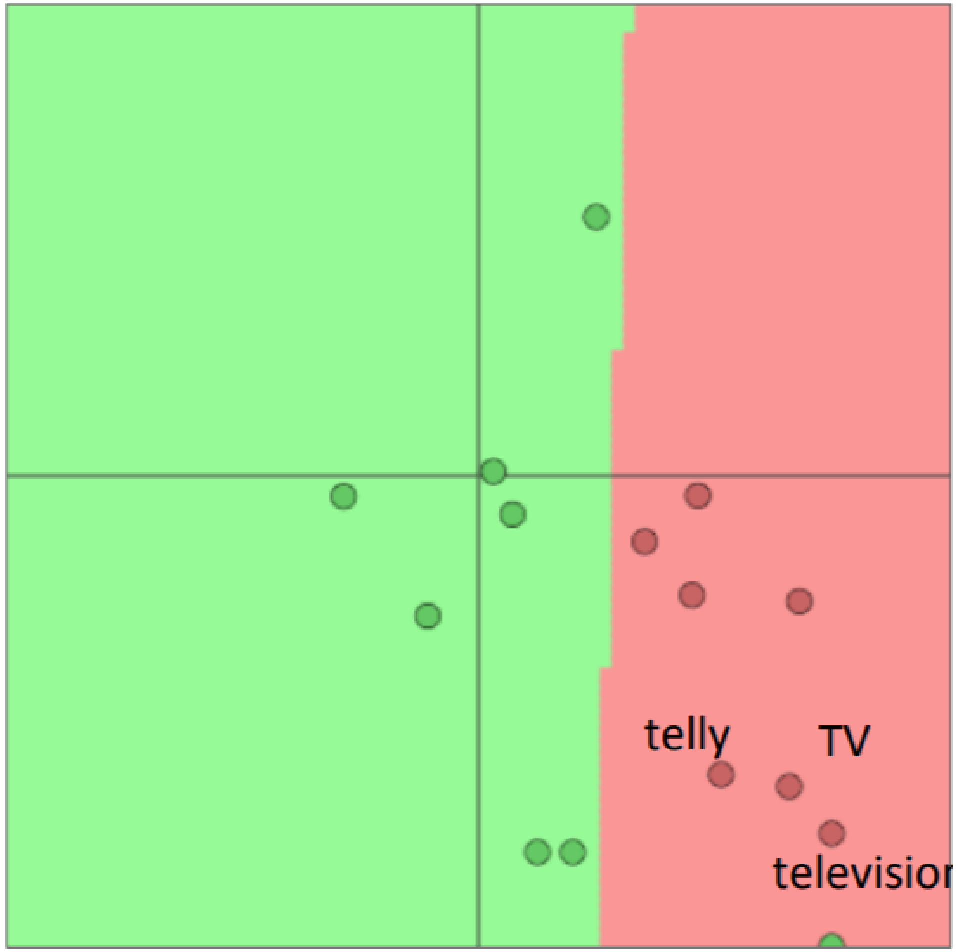
- What is the major benefit of word vectors obtained by skip-gram or GloV?
  - Ability to also classify words accurately
    - Countries cluster together -> classifying location words should be possible with word vectors (even for countries that do not exist in the labeled training set)
  - Fine tune (or learn from scratch for any task) and incorporate more information
    - Project sentiment into words to find most positive/negative words in corpus

Losing generalization by re-training word vectors

- Setting: Training classifier for movie review sentiment of words
  - in the training data we have "TV" and "telly"
  - In the testing data we have "television"

- Originally they were all similar (from pre-training word vectors)

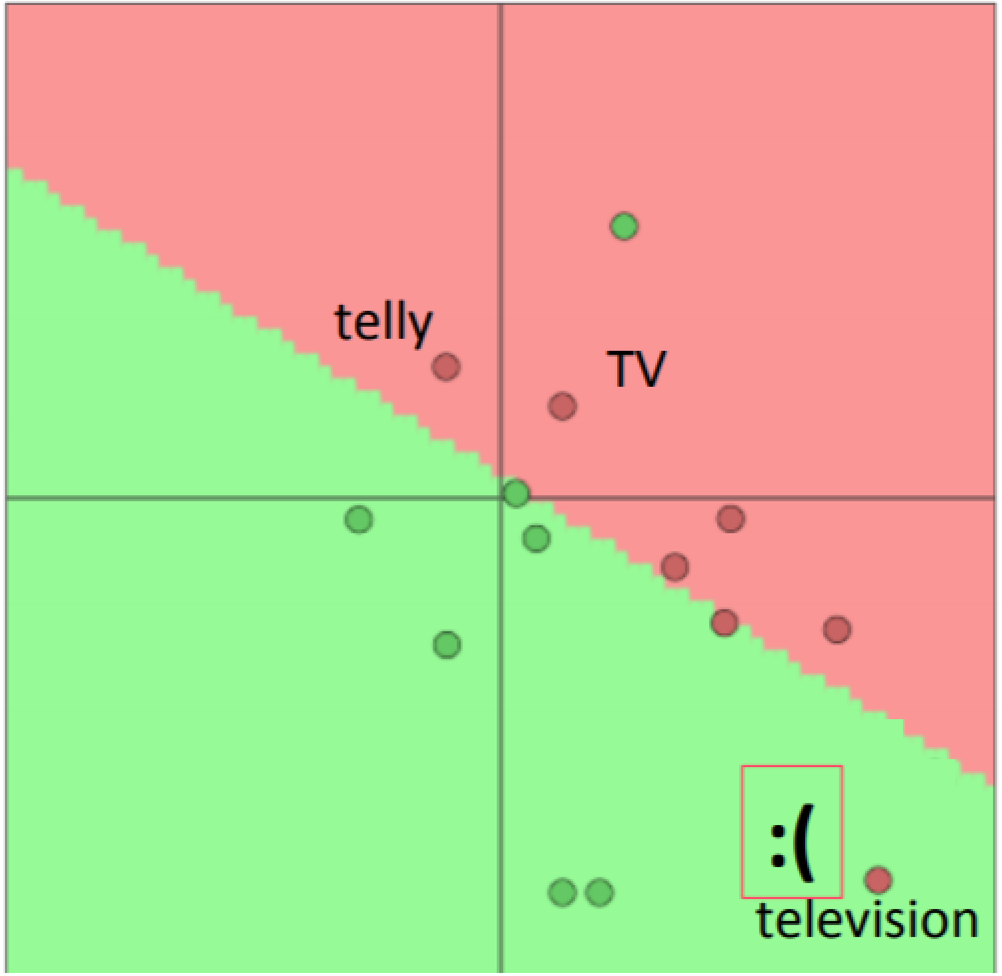
- What happens when we train the word vectors using labeled training data?



Losing generalization by re-training word vectors

- What happens when we train the word vectors?
  - Those that are in the training data move around
  - Words from pre-training that do NOT appear in training stay

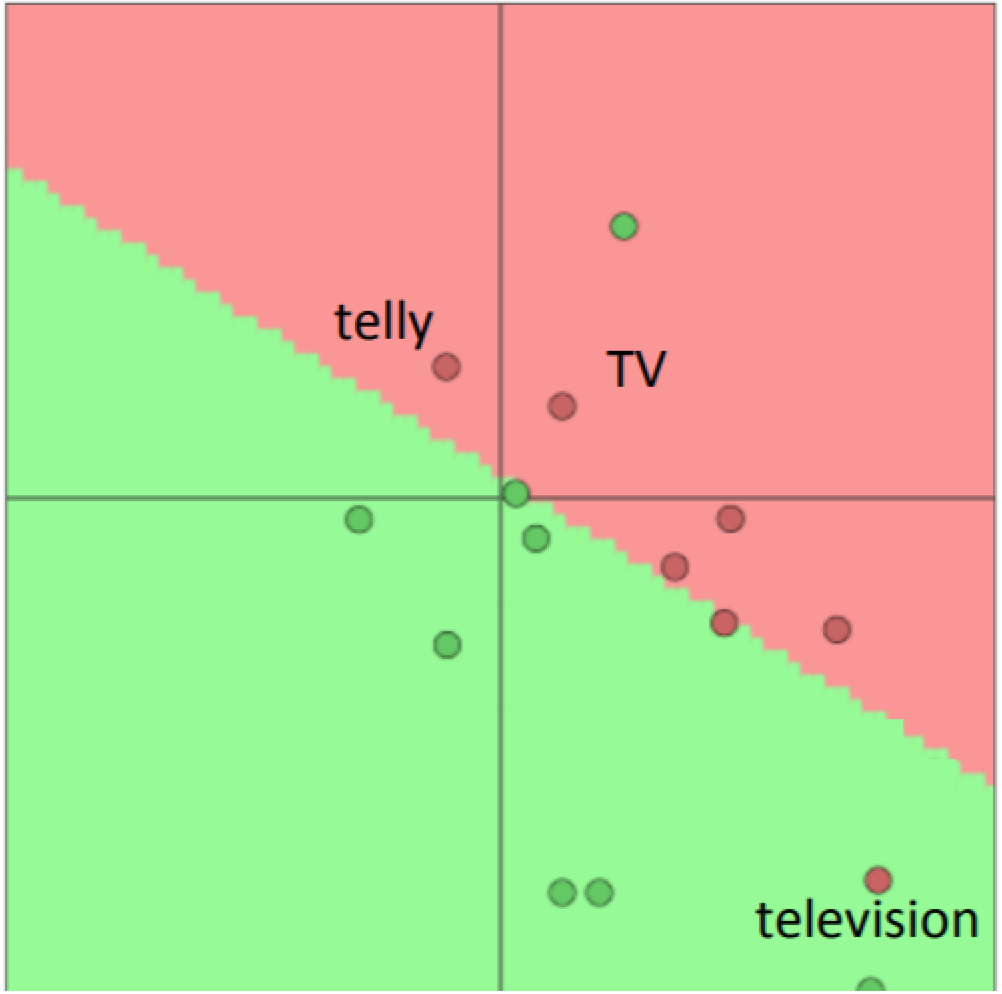
- Example:
  - In training data: “TV” and “telly”
  - Only in testing data: “television”



Losing generalization by re-training word vectors

If you only have a small training data set, don't train the word vectors.

If you have have a very large dataset, it may work better to train word vectors to the task.



Example: Using word2vec in Image Captioning

1. Takes words as inputs
2. Converts them to word vectors.
3. Uses word vectors as inputs to the sequence generation LSTM
4. Maps the output word vectors by this system back to natural language words
5. Produces words as answers

Word vectors: advantages

- It captures both syntactic (POS) and semantic information
- It scales
  - Train on billion word corpora in limited time
- Can easily incorporate a new sentence/ document or add a word to the vocabulary
- Word embeddings trained by one can be used by others.
- There is a nice Python module for word2vec
  - Gensim (word2vec: http://radimrehurek.com/2014/02/word2vec-tutorial/)

Resources

- Mikolov et al., Distributed Representations of Words and Phrases and their Compositionality, 2013.
- Pennington et al., Global Vectors for Word Representation, 2014.