Word Embedding

CE-324: Modern Information Retrieval

Sharif University of Technology

M. Soleymani

Fall 2018

Many slides have been adopted from Socher lectures, cs224d, Stanford, 2017.

One-hot coding

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

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

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?

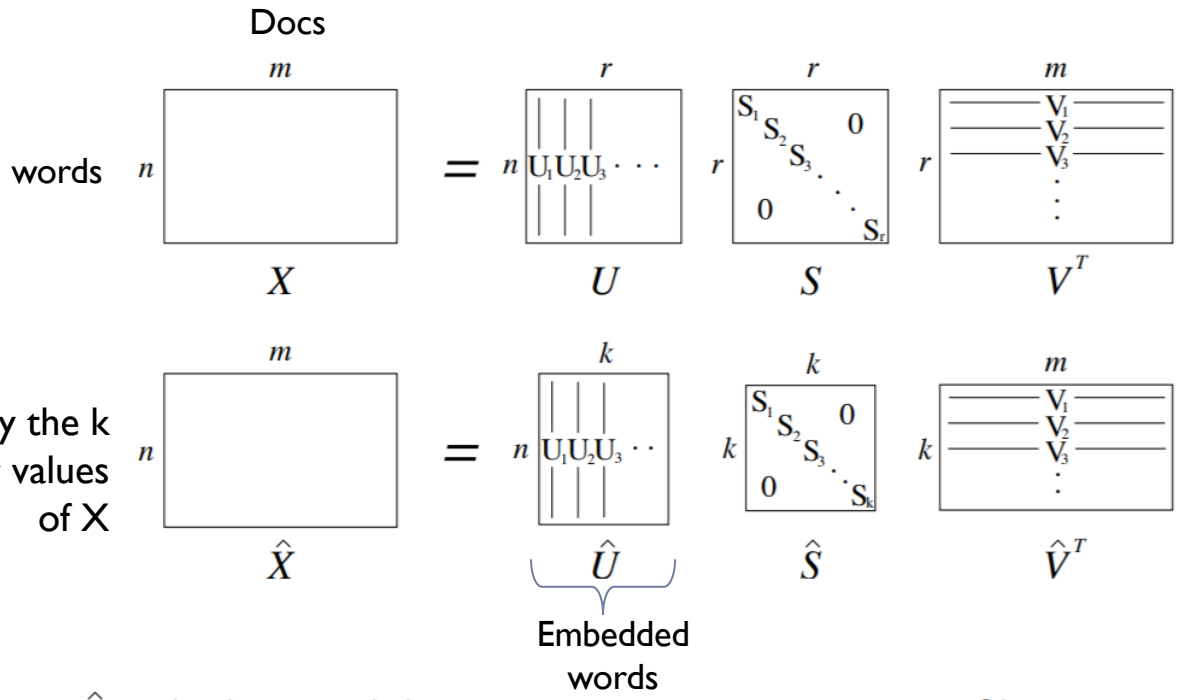
- Answer: With a co-occurrence matrix X
  - options: **full document** vs **windows**

- **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*.



Problems with SVD

- Its computational cost scales quadratically for n x m matrix: $O(mn^2)$ flops (when n<m)
  - Bad for millions of words or documents
- Hard to incorporate new words or documents
- Does not consider order of words in the documents

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

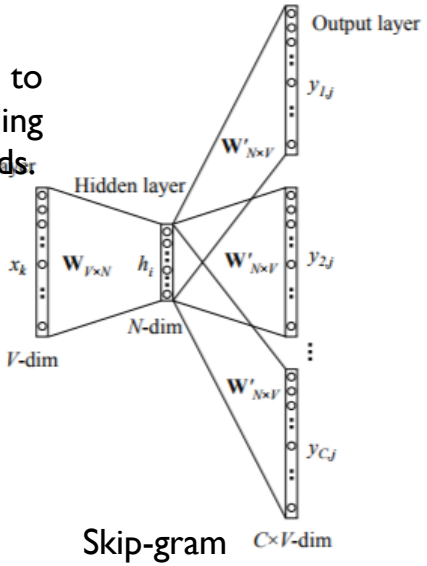
word2vec

- Key idea: The word vector can predict surrounding words
- word2vec: as originally described (Mikolov et al 2013), a NN model using a two-layer network (i.e., not deep!) to perform dimensionality reduction.
- Faster and can easily incorporate a new sentence/document or add a word to the vocabulary
- Very computationally efficient, good all-round model (good hyper-parameters already selected).

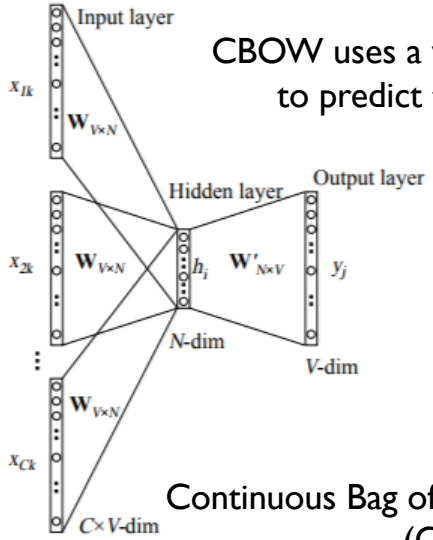
Skip-gram vs. CBOW

- 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 uses a word to predict the surrounding words.



Skip-gram

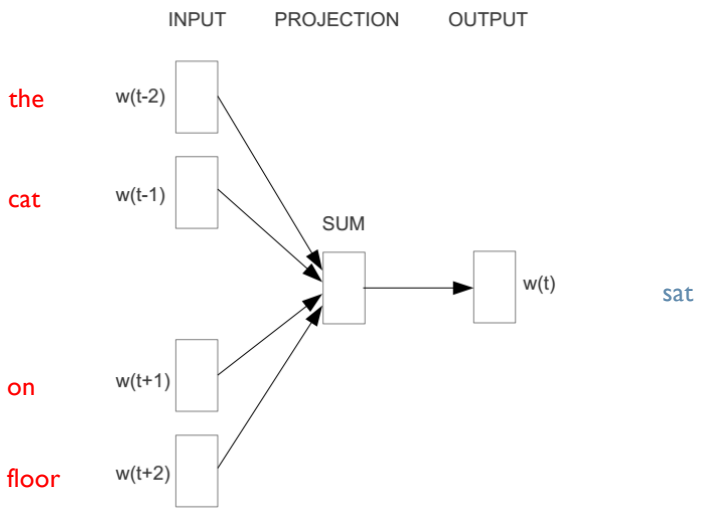


CBOW uses a window of word to predict the middle word

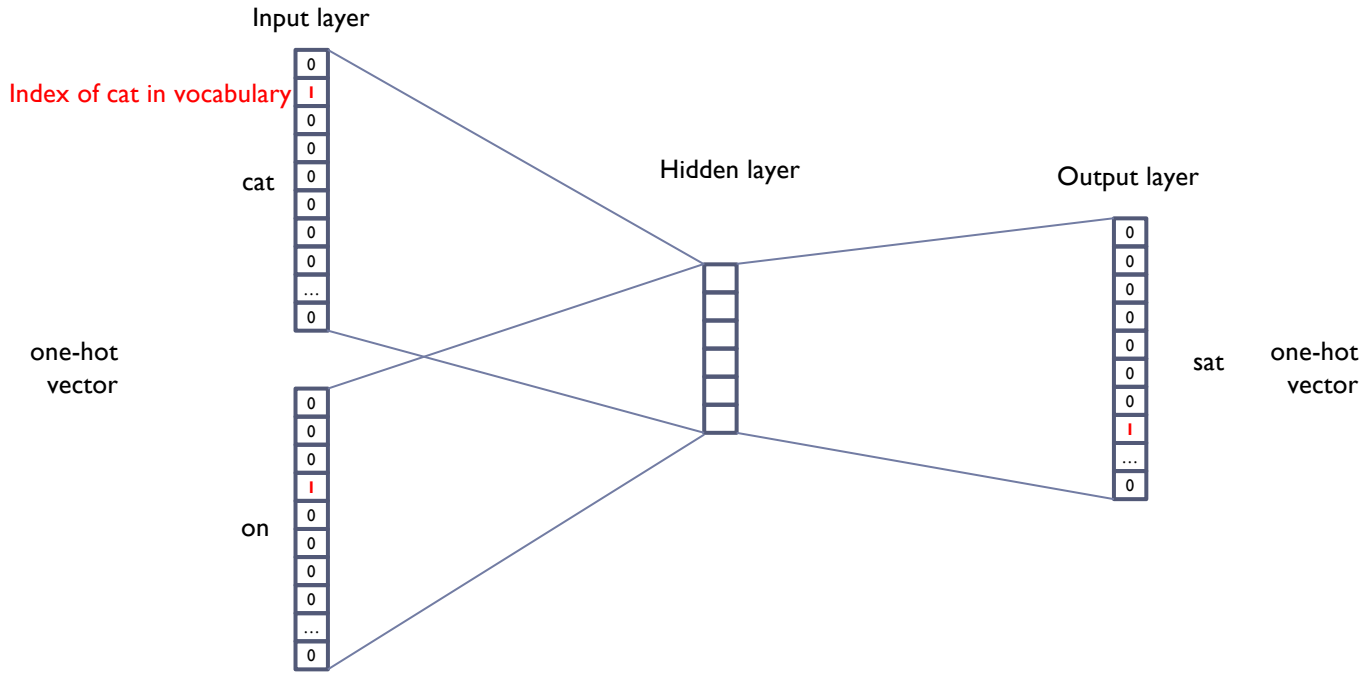
Continuous Bag of words (CBOW)

Continuous Bag of Word: Example

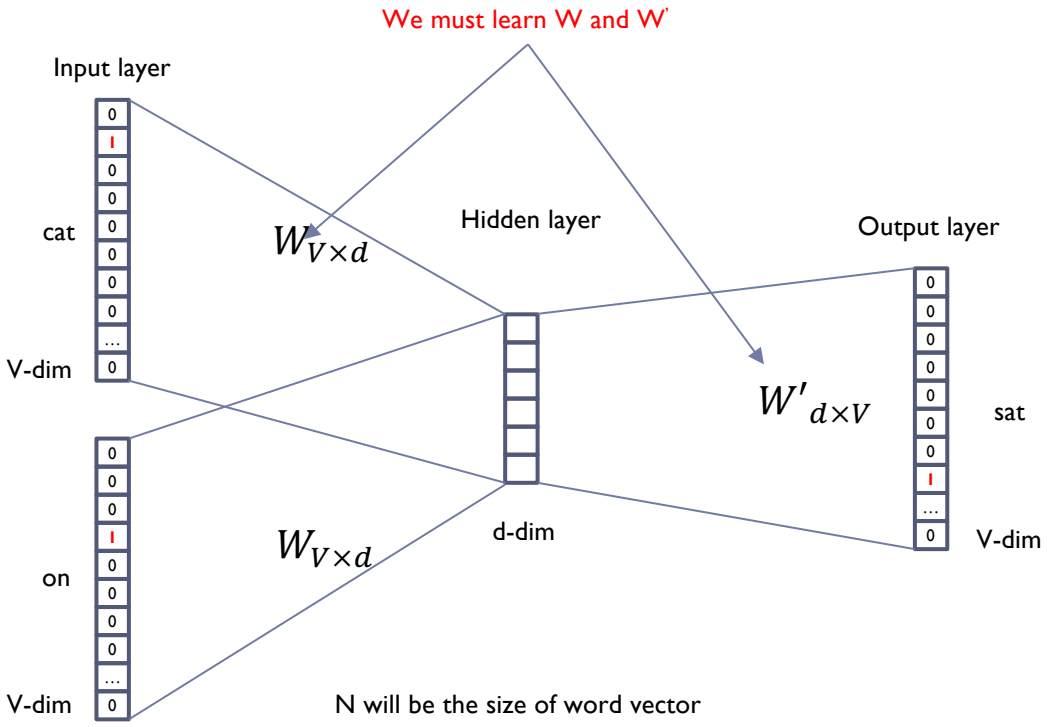
- E.g. "The cat sat on floor"
  - Window size = 2



Continuous Bag of Word: Example

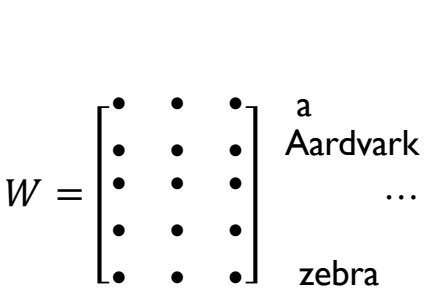


Continuous Bag of Word: Example



Word embedding matrix

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

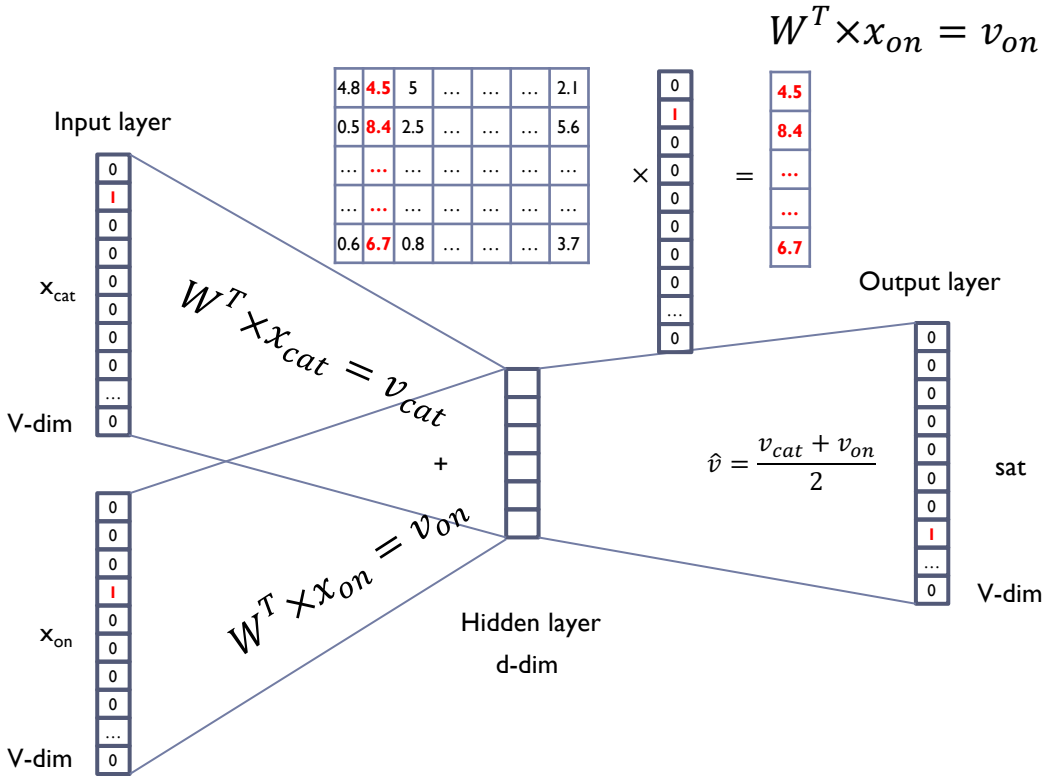


$$x = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \\ 0 \end{bmatrix} \quad (x_k = 1)$$

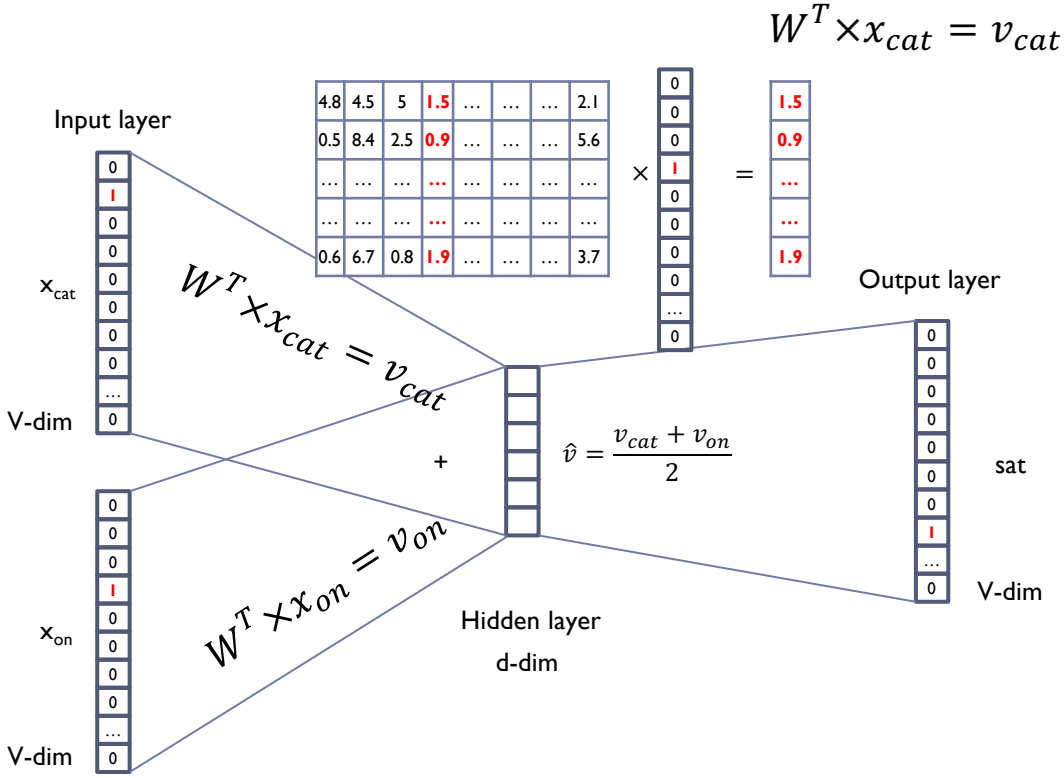
$$h = x^T W = W_{k,\cdot} = v_k$$

k-th row of the matrix W

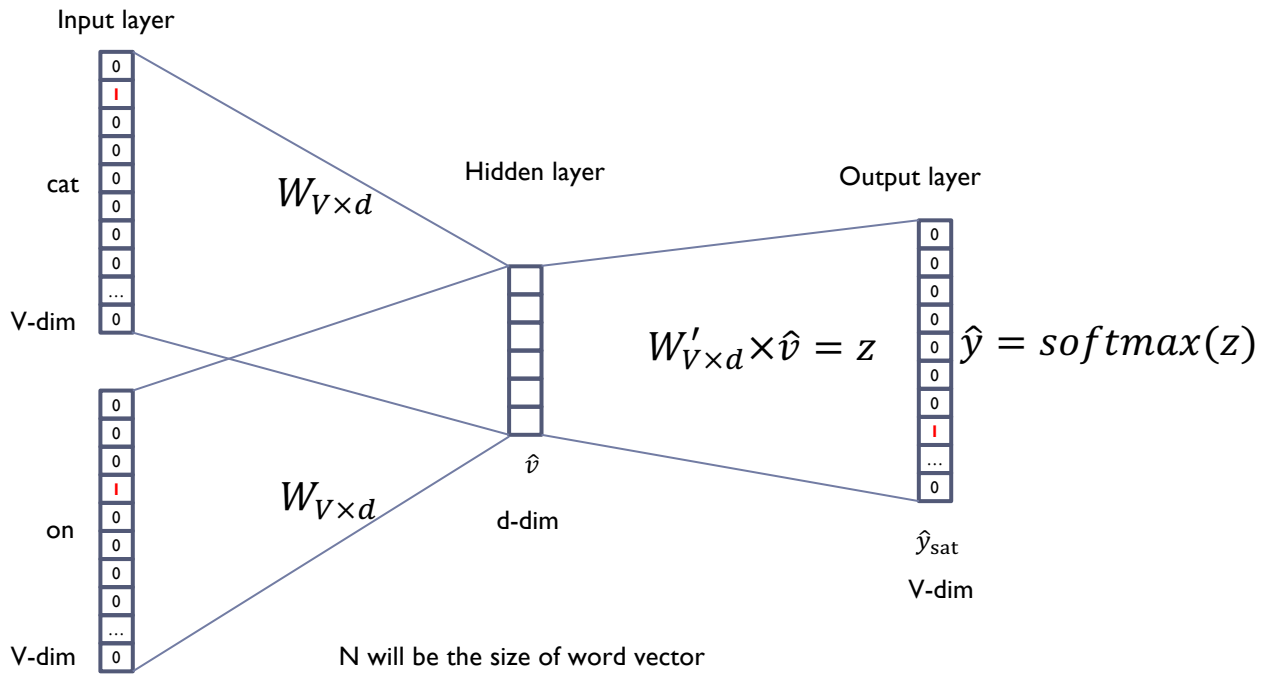
Continuous Bag of Word: Example



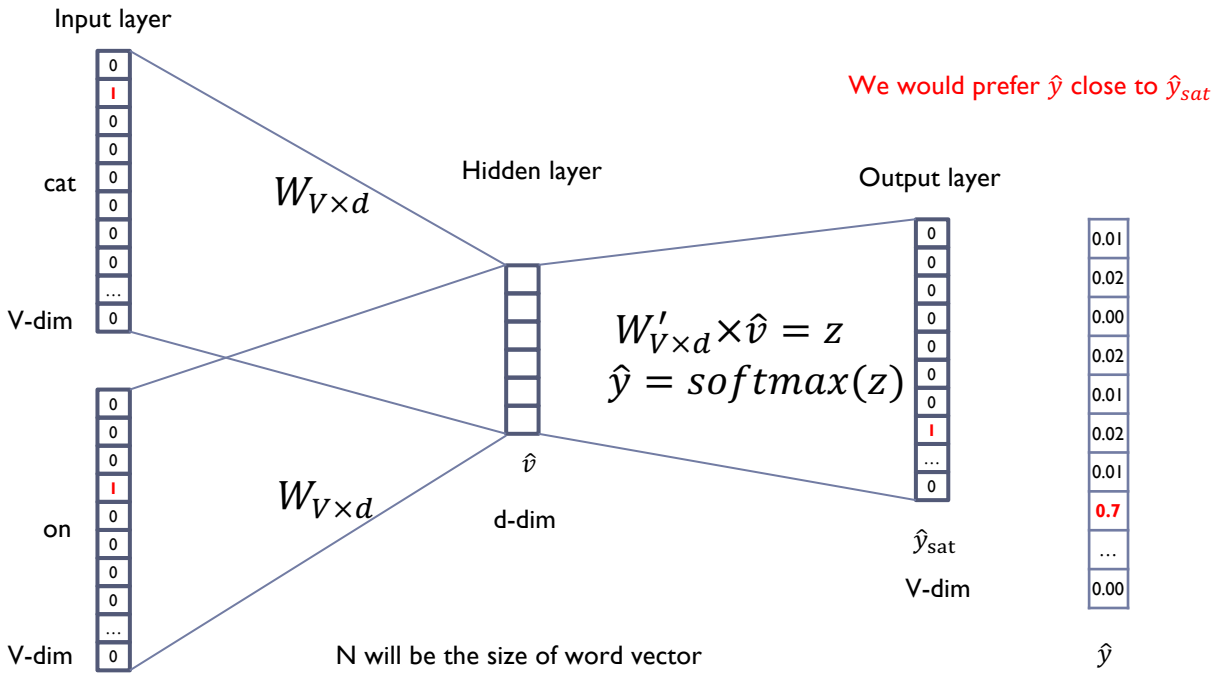
Continuous Bag of Word: Example



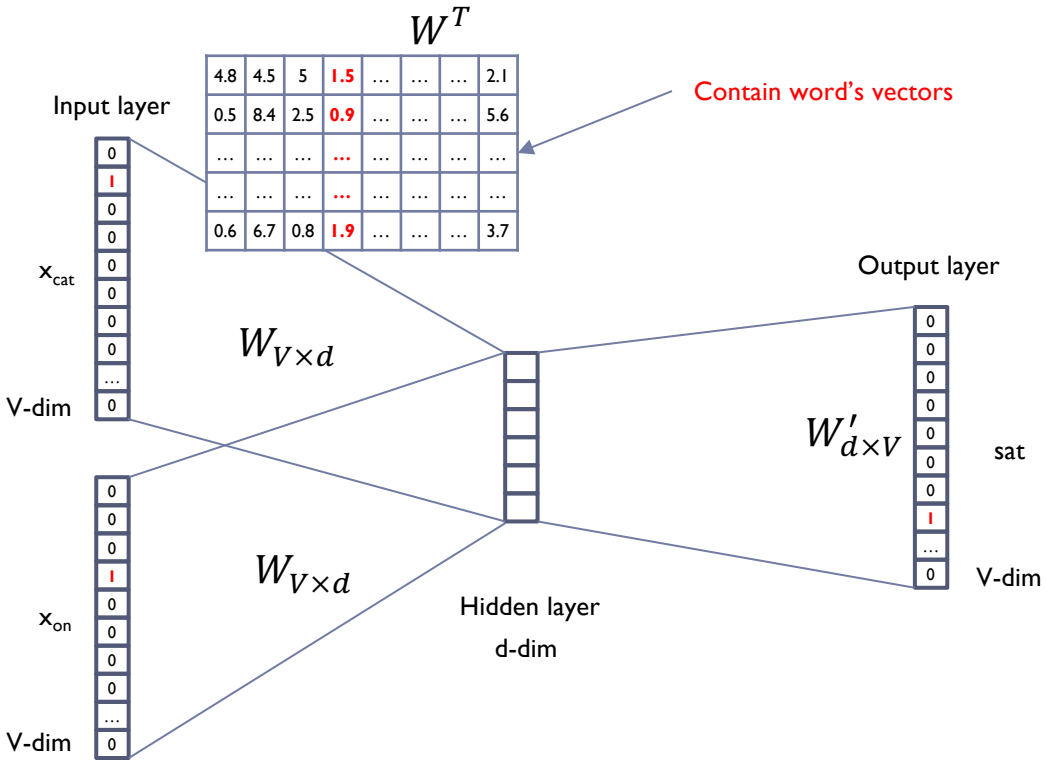
Continuous Bag of Word: Example



Continuous Bag of Word: Example



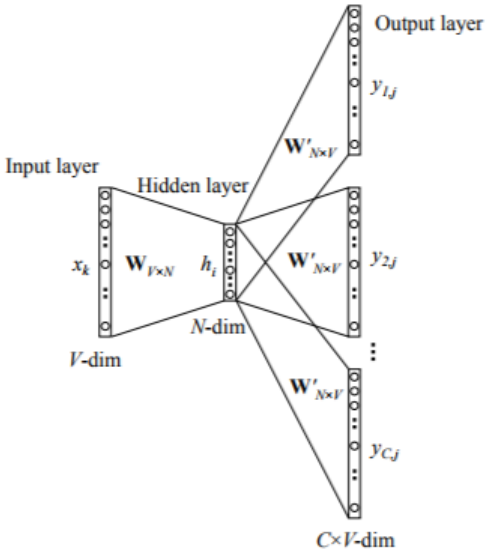
Continuous Bag of Word: Example

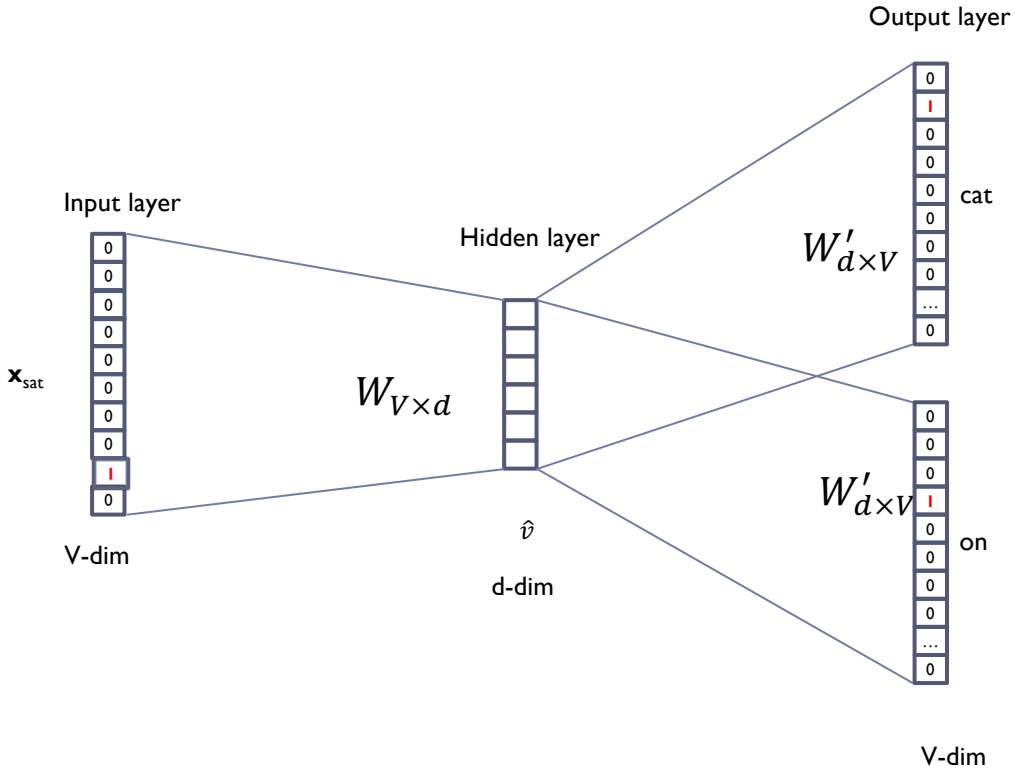


We can consider either W or W' as the word's representation. Or even take the average.

Skip-gram

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




Output layer
Input layer
Hidden layer
cat
on
$W'_{d\times V}$
$W'_{d\times V}$
$W_{V\times d}$
$\mathbf{x}_{sat}$
$\hat{v}$
V-dim
d-dim
V-dim

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:

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

- Use a large training corpus to maximize it

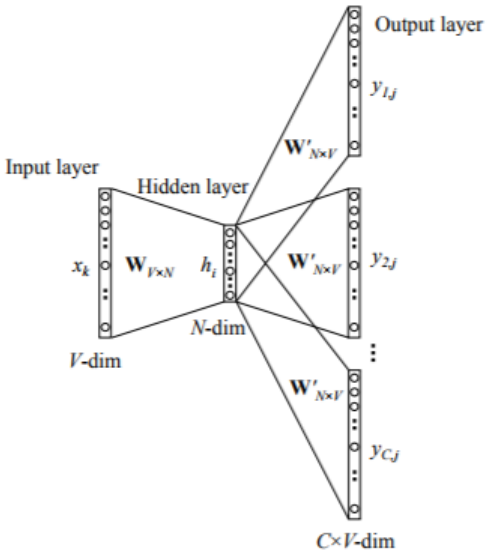
T: training set size
m: context size
w_j : vector representation of the jth word
θ: whole parameters of the network

m is usually 5~10

Skip-gram

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

$$P(w_o|w_I) = \frac{e^{score(w_o,w_I)}}{\sum_x e^{score(w_x,w_I)}}$$



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

$$h = x_I^T W = W_{I,.} = v_I$$
$$W'_{.,o} = u_o$$

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

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:

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

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

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:

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

Review: Iterative optimization of objective function

- Objective function: $J(\boldsymbol{\theta})$
- Optimization problem: $\widehat{\boldsymbol{\theta}} = \underset{\boldsymbol{\theta}}{\operatorname{argmax}} J(\boldsymbol{\theta})$
- Steps:
  - Start from $\boldsymbol{\theta}^0$
  - Repeat
    - Update $\boldsymbol{\theta}^t$ to $\boldsymbol{\theta}^{t+1}$ in order to increase J
    - $t \leftarrow t + 1$
  - until we hopefully end up at a maximum

Review: Gradient ascent

- First-order optimization algorithm to find $\widehat{\boldsymbol{\theta}} = \underset{\boldsymbol{\theta}}{\operatorname{argmax}} J(\boldsymbol{\theta})$
  - Also known as "**steepest ascent**"
- In each step, takes steps proportional to the negative of the gradient vector of the function at the current point $\boldsymbol{\theta}^t$:
  - $J(\boldsymbol{\theta})$ increases fastest if one goes from $\boldsymbol{\theta}^t$ in the direction of $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}^t)$
  - Assumption: $J(\boldsymbol{\theta})$ is defined and differentiable in a neighborhood of a point $\boldsymbol{\theta}^t$

Review: Gradient ascent

- Maximize $J(\boldsymbol{\theta})$

$$\boldsymbol{\theta}^{t+1} = \boldsymbol{\theta}^{t} + \eta \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}^{t})$$

Step size (Learning rate parameter)

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{w}) = [\frac{\partial J(\boldsymbol{\theta})}{\partial \theta_1}, \frac{\partial J(\boldsymbol{\theta})}{\partial \theta_2}, \dots, \frac{\partial J(\boldsymbol{\theta})}{\partial \theta_d}]$$

- If η is small enough, then $J(\boldsymbol{\theta}^{t+1}) \geq J(\boldsymbol{\theta}^{t})$.
- η can be allowed to change at every iteration as η_t.

Gradient

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

$$= \frac{\partial}{\partial v_I} \left(\log e^{u_o^T v_I} - \log \sum_x e^{u_x^T v_I} \right)$$

$$= u_o - \frac{1}{\sum_x e^{u_x^T v_I}} \sum_x u_x \, e^{u_x^T v_I}$$

$$= u_o - \sum_x p(w_x|w_I) u_x$$

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

Word2vec: Negative Sampling

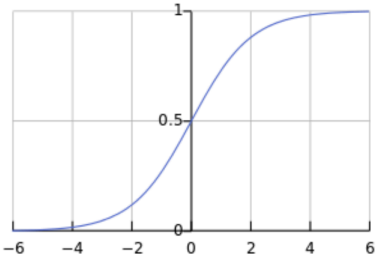
- Computing $\sum_{x=1}^{|V|} p(w_x|w_I)u_x$ is very time consuming.

- Main idea: train binary classifier for a true pair (center word and word in its context window) and a couple of random pairs (the center word with a random word)

Negative sampling

- k is the number of negative samples

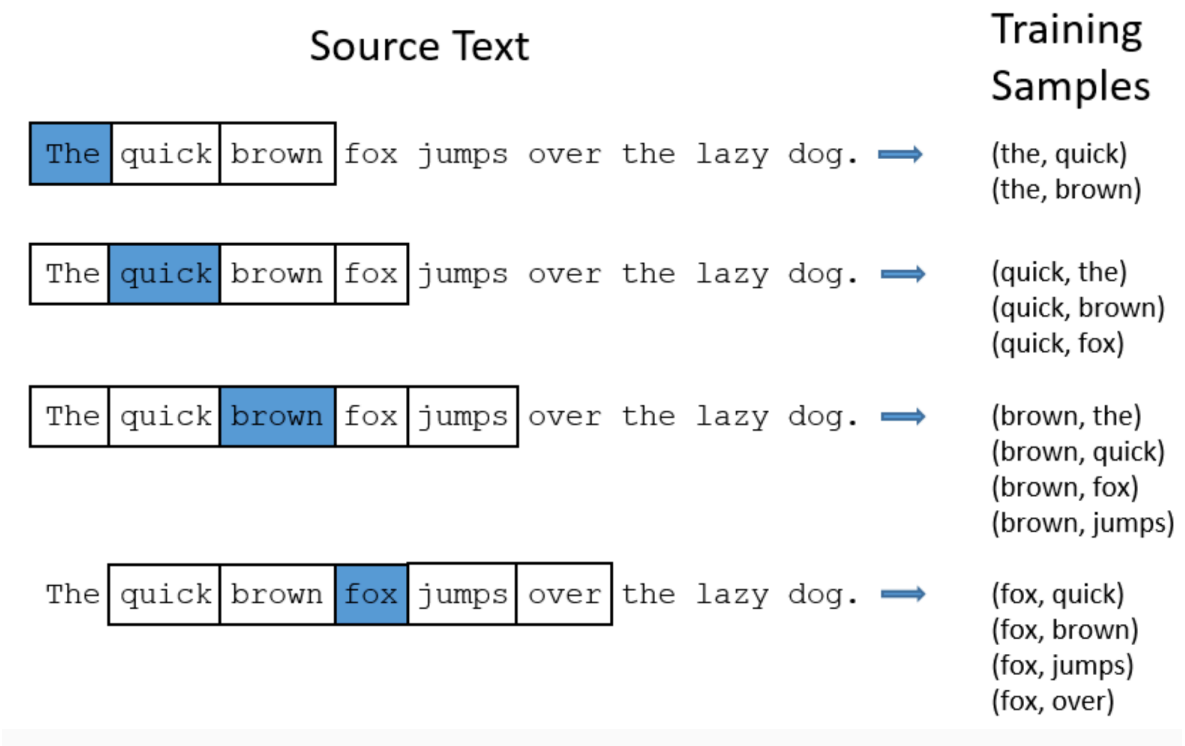
$$\log \sigma(u_o^T v_I) + \sum_{w_j \sim P(w)} \log \sigma\left(-u_j^T v_I\right)$$



- 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.

Example



http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model

What to do with the two sets of vectors?

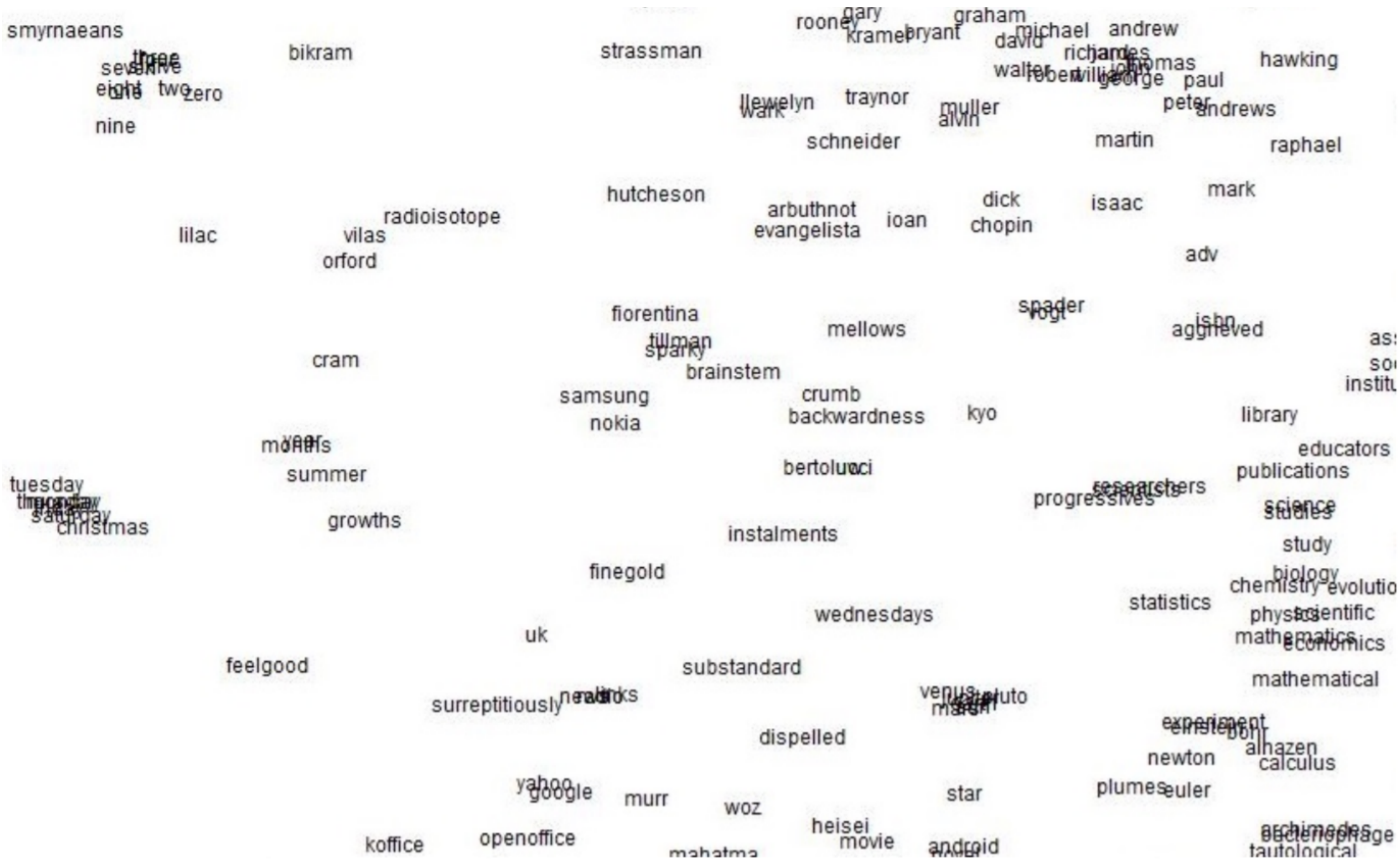
- We end up with U and V from all the vectors u and v (in columns)
  - Both capture similar co-occurrence information.

- The best solution is to simply sum them up:

$$X_{final} = U + V$$

- 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
richards
james
walter
thomas
george
paul
hawking
eight
two
zero
nine
llewelyn
traynor
muller
alvin
peter
andrews
schneider
martin
raphael
hutcheson
arbuthnot
evangelista
ioan
dick
chopin
isaac
mark
radioisotope
lilac
vilas
orford
adv
fiorentina
mellows
spader
vogt
aggrieved
tillman
sparky
brainstem
cram
samsung
nokia
crumb
backwardness
kyo
library
months
summer
bertolucci
educators
publications
tuesday
christmas
growths
progressives
researchers
science
studies
instalments
study
finegold
biology
chemistry
statistics
wednesdays
scientific
mathematics
economics
uk
feelgood
substandard
mathematical
surreptitiously
venus
pluto
mars
experiment
dispelled
alhazen
calculus
newton
yahoo
google
murr
woz
star
plumes
euler
heisei
movie
android
koffice
openoffice
mahatma
archimedes
bacteriophage
tautological

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?

LSI vs. Skip-gram

Count based prediction	Direct prediction
LSA	Skip-gram/CBOW
• Fast training • Efficient usage of statistics • Primarily used to capture word similarity • Disproportionate importance given to large counts	• Scales with corpus size • Inefficient usage of statistics • Generate improved performance on other tasks • Can capture complex patterns beyond word similarity

Slide by M. Korniyenko, S. Samson
http://www.sfs.uni-tuebingen.de/~ddekok/dl4nlp/glove-presentation.pdf

LSI disadvantages

- The co-occurrence matrix changes very often (new words are added).
- The matrix is extremely sparse since most words do not co-occur.
- Quadratic cost to train.
- Requires the incorporation of some hacks on X to account for the drastic imbalance in word frequency.

- Iteration based methods solve many of these issues in a far more elegant manner.

Slide by M. Korniyenko, S. Samson
http://www.sfs.uni-tuebingen.de/~ddekok/dl4nlp/glove-presentation.pdf

Main Idea of word2vec

- Instead of capturing co-occurrence counts directly, predict surrounding words of every word

- For many tasks, word2vec (skip-gram) outperforms standard count-based vectors.
  - But mainly due to the hyperparameters (see Levy et al)

Window based co-occurrence matrix: Example

Corpus

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

Window length 1 (more common: 5 -
Symmetric (irrelevant whether lel or right contex

X

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

More about Word2Vec – relation to LSA

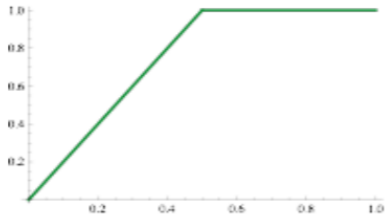
- LSA factorizes a matrix of co-occurrence counts v

- (Levy and Goldberg 2014) proves that skip-gram model implicitly factorizes a (shifted) PMI matrix!

$$PMI(w,c) = \log \frac{\hat{P}(w,c)}{\hat{P}(w)\hat{P}(c)} = \log \frac{\#(w,c)\cdot|D|}{\#(w)\cdot\#(c)}$$

GloVe

$$J(\theta) = \sum_{i,j} f(X_{ij})(u_i^T v_j - \log X_{ij})$$

$f \sim$ 

- 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: Word analogy tasks

- Performance in completing analogy tasks:
  - Analogy queries
  - Example: "*man* is to *woman* as *king* is to — ?"

$$x_b - x_a \approx x_d - x_c$$
$$d^* = \underset{i}{\text{argmax}}\, sim(x_b - x_a, x_i - x_c)$$
$$x_b - x_a + x_c \approx x_d$$
$$d^* = \underset{i}{\text{argmax}}\, sim(x_b - x_a + x_c, x_i)$$

$$d = \underset{i}{\text{argmax}} \frac{(x_b - x_a + x_c)^T x_i}{\|x_b - x_a + x_c\|}$$

- 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

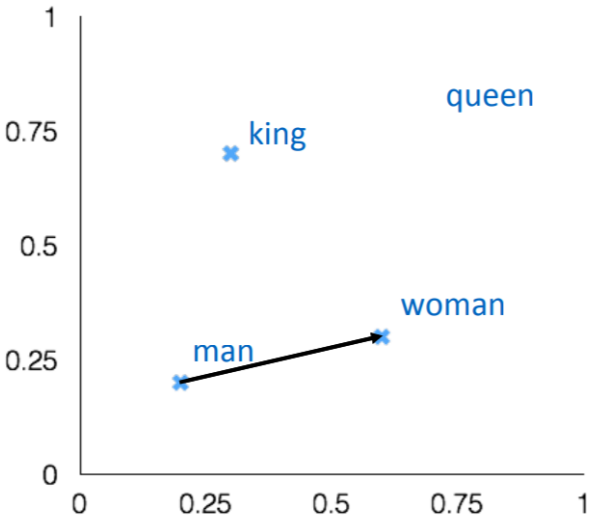
Test for linear relationships, examined by Mikolov et al. (2014)

a:b :: c:?

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

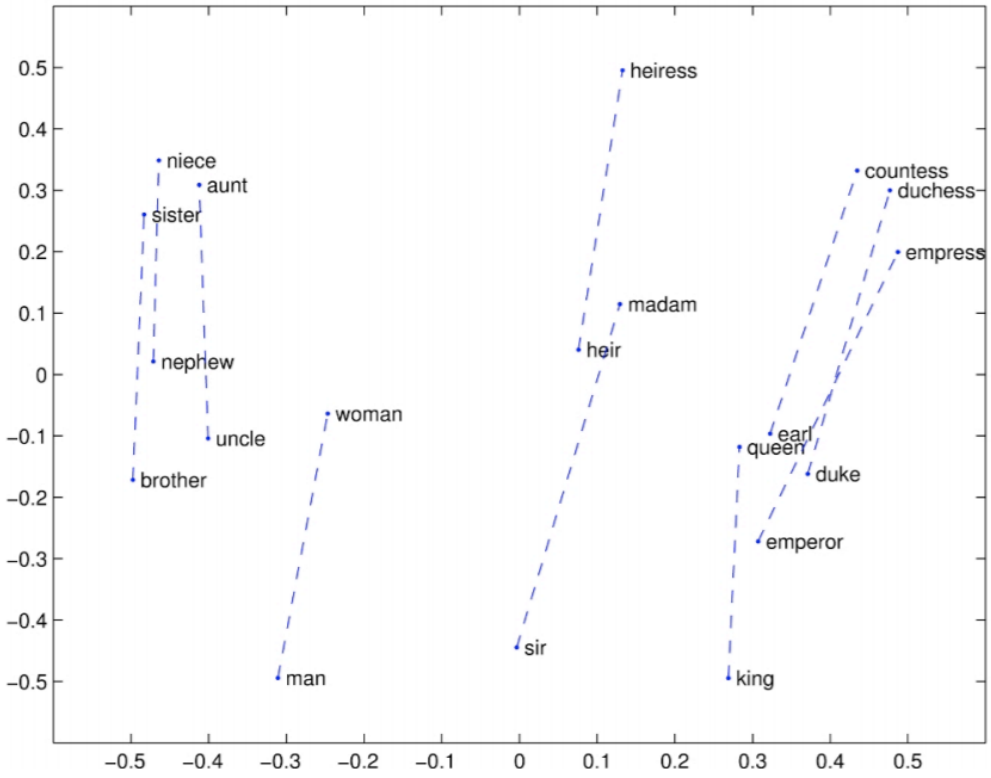
man:woman :: king:?

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

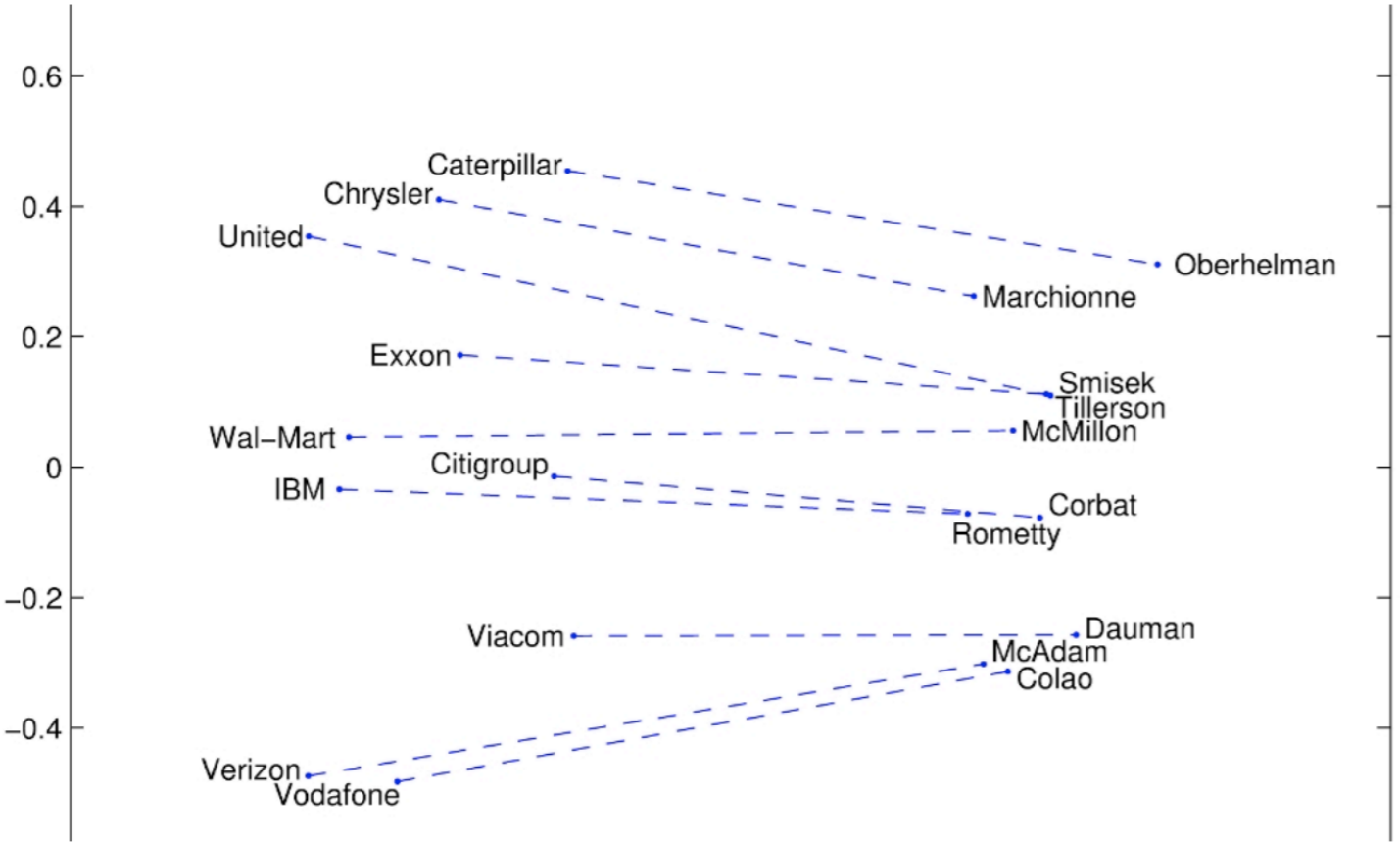


The linearity of the skip-gram model makes its vectors more suitable for such linear analogical reasoning

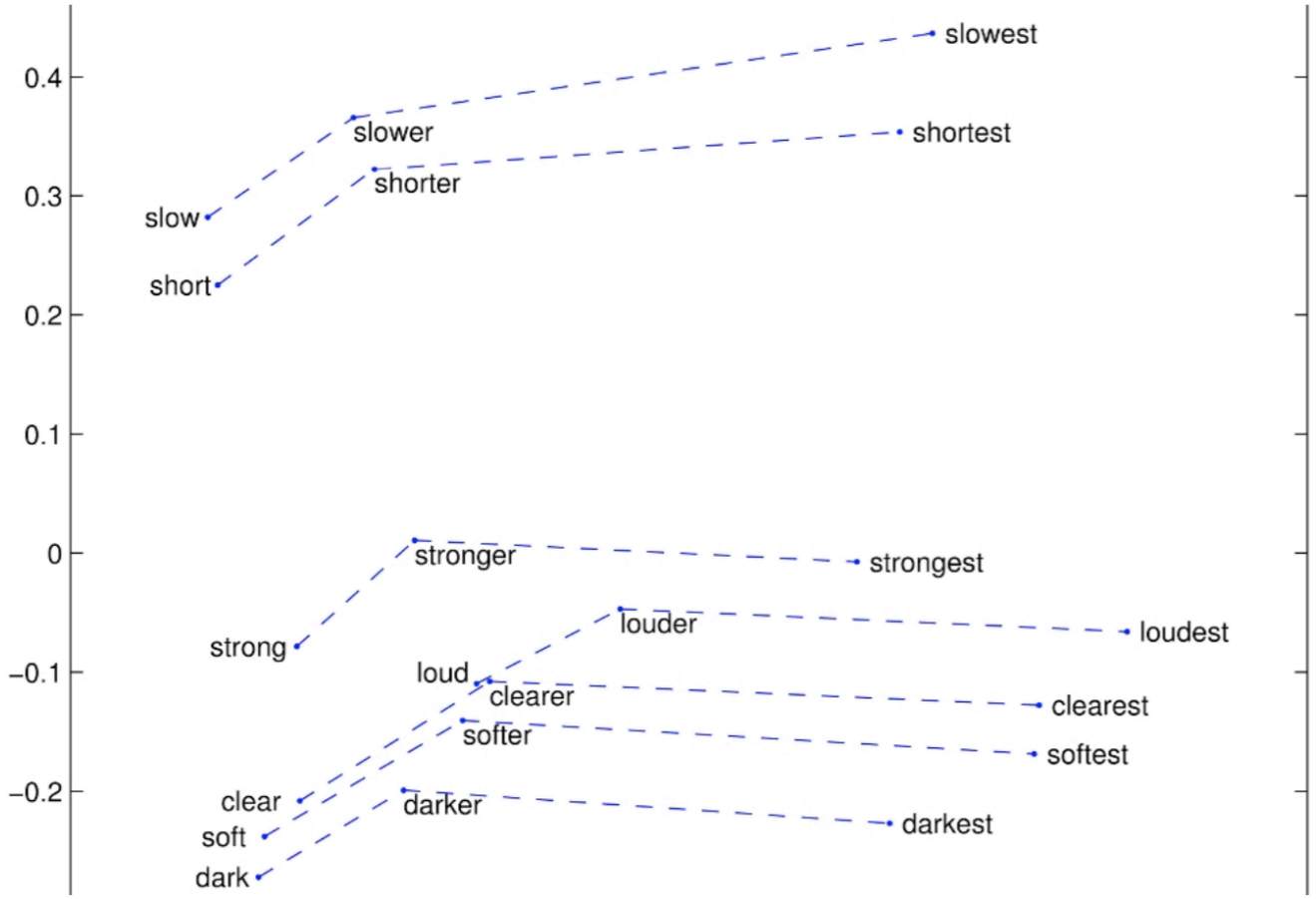
Visualizations



GloV Visualizations: Company - CEO



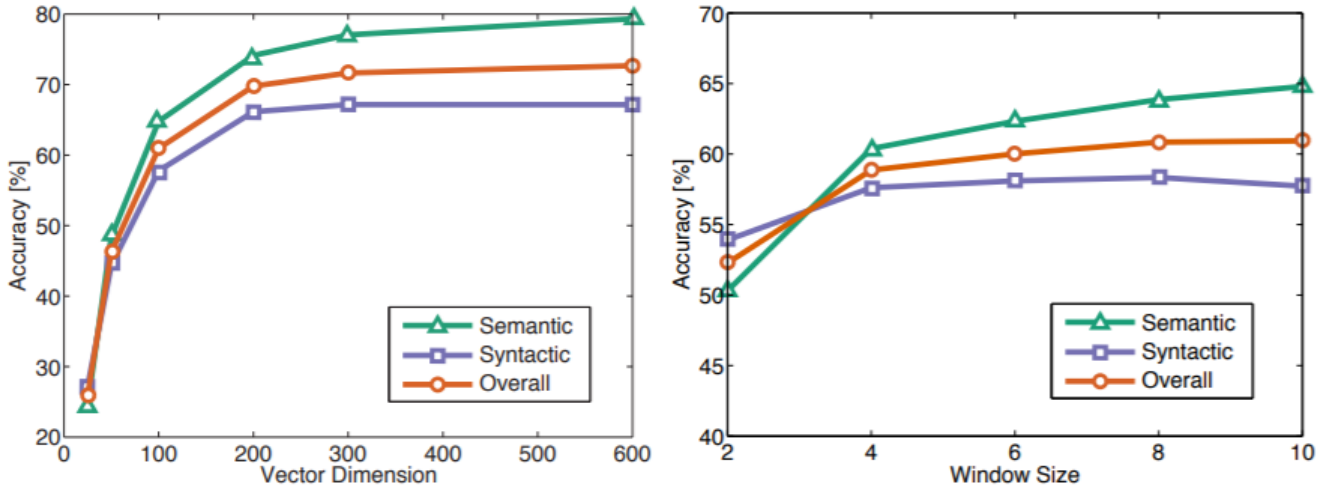
Glov Visualizations: Superlatives



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



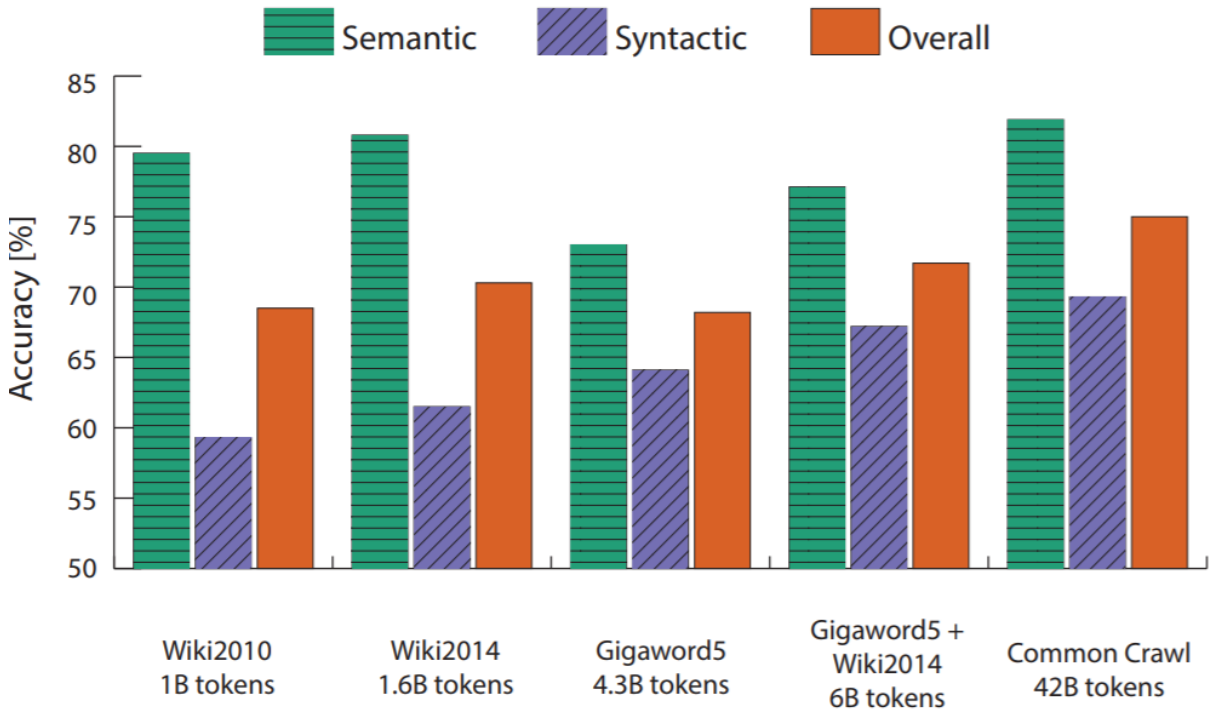
- Best dimensions ~300, slight drop-off afterwards
- But this might be different for downstream tasks!

- 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: text classification

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.