

Deep Learning Course

Lecture 5: Word Vectors

Valentin Malykh

Based on Stanford CS224d

Как представить значение слова?

Definition: **Meaning** (Webster dictionary)

- the idea that is represented by a word, phrase, etc.
- the idea that a person wants to express by using words, signs, etc.
- the idea that is expressed in a work of writing, art, etc.

ак представить значение слова для компьютера?

Common answer: Use a taxonomy like WordNet that has hypernyms (is-a) relationships and

```
from nltk.corpus import wordnet as wn
panda = wn.synset('panda.n.01')
hyper = lambda s: s.hypernyms()
list(pandaclosure(hyper))
```

```
[Synset('procyonid.n.01'),
Synset('carnivore.n.01'),
Synset('placental.n.01'),
Synset('mammal.n.01'),
Synset('vertebrate.n.01'),
Synset('chordate.n.01'),
Synset('animal.n.01'),
Synset('organism.n.01'),
Synset('living_thing.n.01'),
Synset('whole.n.02'),
Synset('object.n.01'),
Synset('physical_entity.n.01'),
Synset('entity.n.01')]
```

synonym sets (good):

S: (adj) full, good
S: (adj) estimable, good, honorable, respectable
S: (adj) beneficial, good
S: (adj) good, just, upright
S: (adj) adept, expert, good, practiced, proficient, skillful
S: (adj) dear, good, near
S: (adj) good, right, ripe
...
S: (adv) well, good
S: (adv) thoroughly, soundly, good
S: (n) good, goodness
S: (n) commodity, trade good, good

Проблемы с дискретным представлением

- Great as resource but missing nuances, e.g.
synonyms:
adept, expert, good, practiced, proficient, skillful?
- Missing new words (impossible to keep up to date):
wicked, badass, nifty, crack, ace, wizard, genius, ninjia
- Subjective
- Requires human labor to create and adapt
- Hard to compute accurate word similarity →

Проблемы с дискретным представлением

The vast majority of rule-based **and** statistical NLP work regards words as atomic symbols: **hotel, conference, walk**

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

$$[0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 1 \ 0 \ 0 \ 0 \ 0]$$

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

We call this a “**one-hot**” representation. Its problem:

$$\begin{aligned} \text{motel } & [0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 1 \ 0 \ 0 \ 0 \ 0] \text{ AND} \\ \text{hotel } & [0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 1 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0] = 0 \end{aligned}$$

Distributional similarity based representations

You can get a lot of value by 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* ↗

How to make neighbors represent words?

Answer: With a cooccurrence matrix X

- 2 options: full document vs windows
- Word - document cooccurrence matrix will give general topics (all sports terms will have similar entries) leading to “Latent Semantic Analysis”
- Instead: Window around each word → captures both syntactic (POS) and semantic information

Window based cooccurrence matrix

- Window length 1 (more common: 5 - 10)
- Symmetric (irrelevant whether left or right context)
- Example corpus:
 - I like deep learning.
 - I like NLP.
 - I enjoy flying.

Window based cooccurrence matrix

- Example corpus:
 - I like deep learning.
 - I like NLP.
 - I enjoy flying.

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

Problems with simple cooccurrence vectors

Increase in size with vocabulary

Very high dimensional: require a lot of storage

Subsequent classification models have sparsity issues

→ Models are less robust

Solution: Low dimensional vectors

- Idea: store “most” of the important information in a fixed, small number of dimensions: a dense vector
- Usually around 25 – 1000 dimensions
- How to reduce the dimensionality?

Method 1: Dimensionality Reduction on X

Singular Value Decomposition of cooccurrence matrix X .

$$\begin{matrix} & m \\ n & \boxed{} \end{matrix} = n \begin{matrix} r \\ \boxed{| | |} \\ U_1 U_2 U_3 \dots \end{matrix} \begin{matrix} r \\ S_1 S_2 S_3 \dots \\ 0 \\ \vdots \\ S_r \end{matrix} \begin{matrix} m \\ V_1 \\ V_2 \\ V_3 \\ \vdots \end{matrix}$$

X U S V^T

$$\begin{matrix} & m \\ n & \boxed{} \end{matrix} = n \begin{matrix} k \\ \boxed{| | |} \\ U_1 U_2 U_3 \dots \end{matrix} \begin{matrix} k \\ S_1 S_2 S_3 \dots \\ 0 \\ \vdots \\ S_k \end{matrix} \begin{matrix} m \\ V_1 \\ V_2 \\ V_3 \\ \vdots \end{matrix}$$

\hat{X} \hat{U} \hat{S} \hat{V}^T

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

Simple SVD word vectors in Python

Corpus:

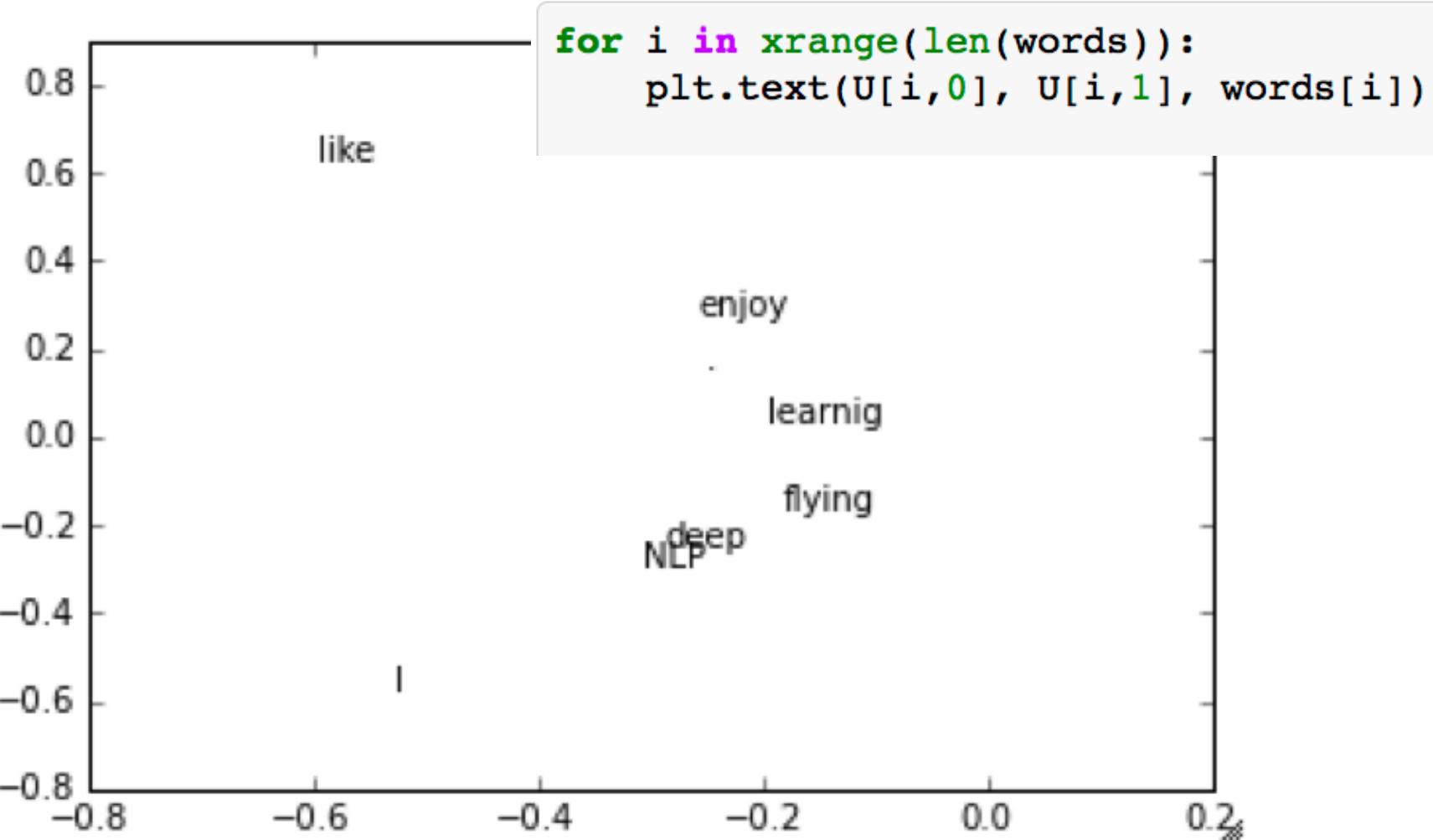
I like deep learning. I like NLP. I enjoy flying.

```
import numpy as np
la = np.linalg
words = ["I", "like", "enjoy",
          "deep", "learnig", "NLP", "flying", ".."]
X = np.array([[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]])
U, s, Vh = la.svd(X, full_matrices=False)
```

Simple SVD word vectors in Python

Corpus: I like deep learning. I like NLP. I enjoy flying.

Printing first two columns of U corresponding to the 2 biggest singular values



Word meaning is defined in terms of vectors

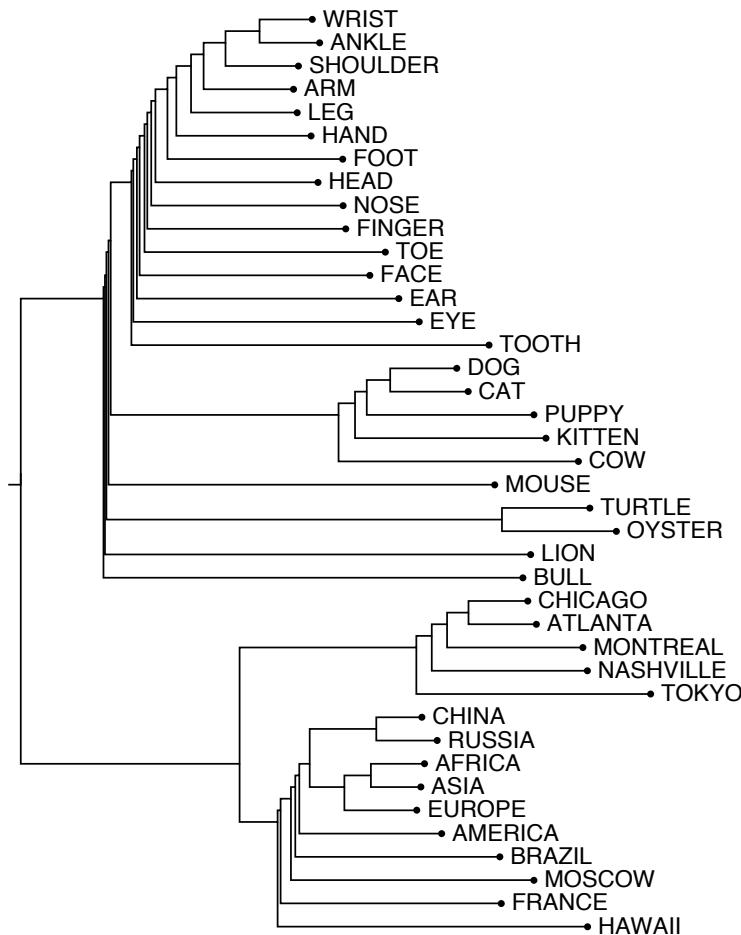
- In all subsequent models, including deep learning models, a word is represented as a dense vector

$$\text{linguistics} = \begin{pmatrix} 0.286 \\ 0.792 \\ -0.177 \\ -0.107 \\ 0.109 \\ -0.542 \\ 0.349 \\ 0.271 \end{pmatrix}$$

Hacks to X

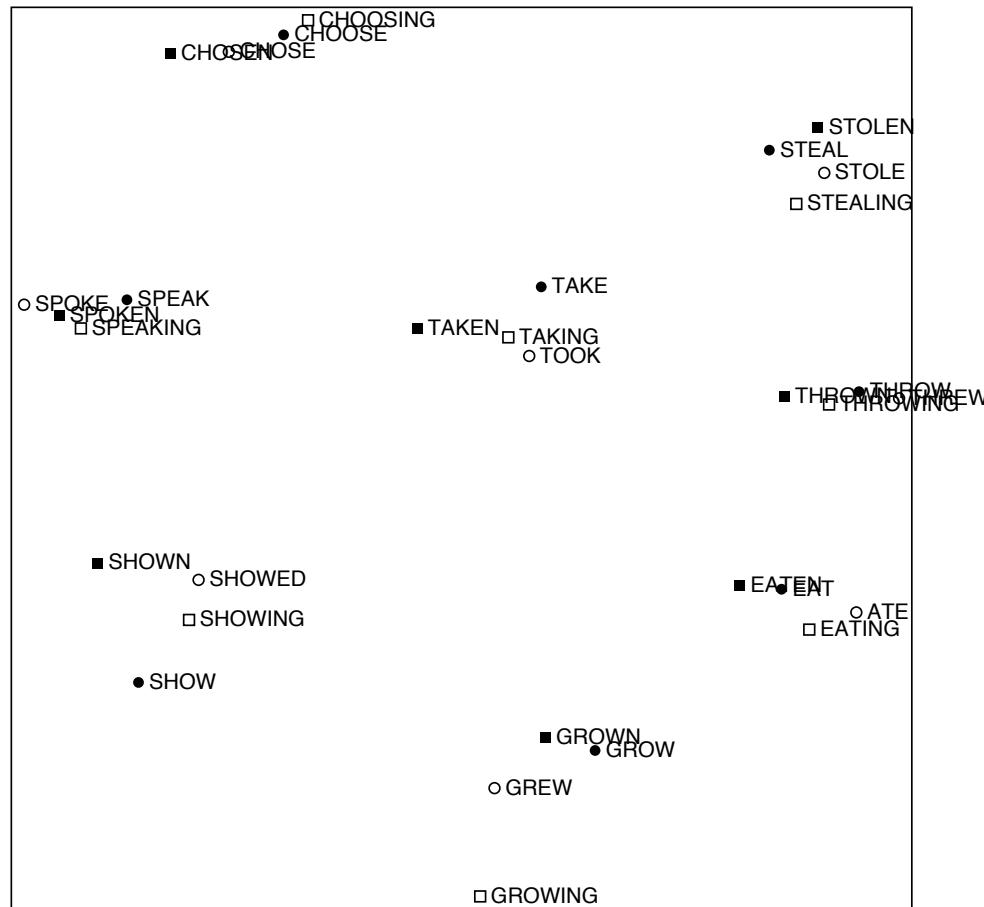
- Problem: function words (the, he, has) are too frequent → syntax has too much impact. Some fixes:
 - $\min(X, t)$, with $t \sim 100$
 - Ignore them all
- Ramped windows that count closer words more
- Use Pearson correlations instead of counts, then set negative values to 0
- +++

Interesting semantic patterns emerge in the vectors



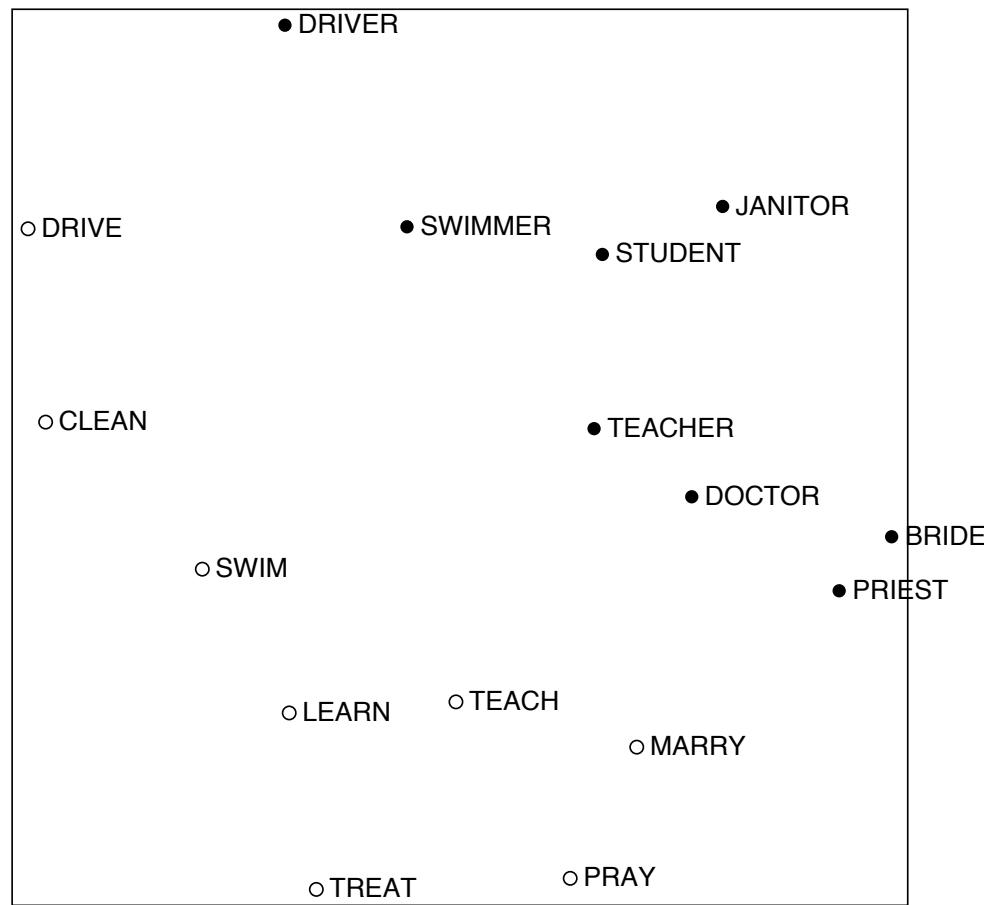
An Improved Model of Semantic Similarity Based on Lexical Co-Occurrence
Rohde et al. 2005

Interesting syntactic patterns emerge in the vectors



An Improved Model of Semantic Similarity Based on Lexical Co-Occurrence
Rohde et al. 2005

Interesting semantic patterns emerge in the vectors



An Improved Model of Semantic Similarity Based on Lexical Co-Occurrence
Rohde et al. 2005

проблемы с SVD

Computational cost scales quadratically for $n \times m$ matrix:

$O(mn^2)$ flops (when $n < m$)

→ Bad for millions of words or documents

Hard to incorporate new words or documents

Different learning regime than other DL models

Idea: Directly learn low-dimensional word vectors

- Old idea. Relevant for this lecture & deep learning:
 - Learning representations by back-propagating errors. (Rumelhart et al., 1986)
 - 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

Main Idea of word2vec

- Instead of capturing cooccurrence counts directly
- Predict surrounding words of every word
- Both are quite similar, see “*Glove: Global Vectors for Word Representation*” by Pennington et al. (2014) and Levy and Goldberg (2014) ... more later
- Быстрее и проще добавлять слово или документ

Details of Word2Vec

- 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:

$$\bullet J(\theta) = \frac{1}{T} \sum_{t=1}^T \sum_{-m \leq j \leq m, j \neq 0} \log p(w_{t+j} | w_t)$$

- Where θ represents all variables we optimize

Details of Word2Vec

- Predict surrounding words in a window of length m of every word
- For $p(w_{t+j} | w_t)$ the simplest first formulation is

$$p(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{w=1}^W \exp(u_w^T v_c)}$$

- where o is the outside (or output) word id, c is the center word id, u and v are “center” and “outside” vectors of o and c
- Every word has two vectors!
- This is essentially “dynamic” logistic regression

Cost/Objective functions

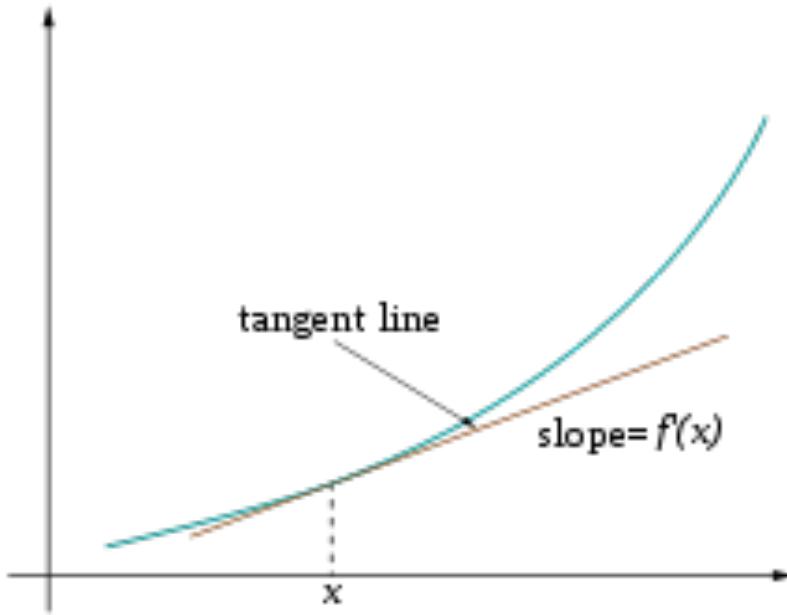
We will optimize (maximize or minimize) our objective/cost functions

For now: minimize → gradient descent

Refresher with trivial example: (from Wikipedia)

Find a local minimum of the function

$f(x)=x^4-3x^3+2$, with derivative $f'(x)=4x^3-9x^2$.



```
x_old = 0
x_new = 6 # The algorithm starts at x=6
eps = 0.01 # step size
precision = 0.00001

def f_derivative(x):
    return 4 * x**3 - 9 * x**2

while abs(x_new - x_old) > precision:
    x_old = x_new
    x_new = x_old - eps * f_derivative(x_old)

print("Local minimum occurs at", x_new)
```

Derivations of gradient

- The basic Lego piece
- Useful basics: $\frac{\partial \mathbf{x}^T \mathbf{a}}{\partial \mathbf{x}} = \frac{\partial \mathbf{a}^T \mathbf{x}}{\partial \mathbf{x}} = \mathbf{a}$
- If in doubt: write out with indices
- Chain rule! If $y = f(u)$ and $u = g(x)$, i.e. $y=f(g(x))$, then:

$$\frac{dy}{dx} = \frac{dy}{du} \frac{du}{dx}$$

Chain Rule

- Chain rule! If $y = f(u)$ and $u = g(x)$, i.e. $y=f(g(x))$, then:

$$\frac{dy}{dx} = \frac{dy}{du} \frac{du}{dx} = \frac{df(u)}{du} \frac{dg(x)}{dx}$$

- Simple example: $\frac{dy}{dx} = \frac{d}{dx} 5(x^3 + 7)^4$

$$y = f(u) = 5u^4 \quad u = g(x) = x^3 + 7$$

$$\frac{dy}{du} = 20u^3 \quad \frac{du}{dx} = 3x^2$$

$$\frac{dy}{dx} = 20(x^3 + 7)3x^2$$

Linear Relationships in word2vec

These representations are *very good* at encoding dimensions of similarity!

- Analogies testing dimensions of similarity can be solved quite well just by doing vector subtraction in the embedding space

Syntactically

- $x_{apple} - x_{apples} \approx x_{car} - x_{cars} \approx x_{family} - x_{families}$
- Similarly for verb and adjective morphological forms

Semantically (Semeval 2012 task 2)

- $x_{shirt} - x_{clothing} \approx x_{chair} - x_{furniture}$
- $x_{king} - x_{man} \approx x_{queen} - x_{woman}$

Count based vs direct prediction

LSA, HAL (Lund & Burgess),
COALS (Rohde et al),
Hellinger-PCA (Lebret & Collobert)

- Fast training
- Efficient usage of statistics
- Primarily used to capture word similarity
- Disproportionate importance given to large counts

• NNLM, HLBL, RNN, Skip-gram/CBOW, (Bengio et al; Collobert & Weston; Huang et al; Mnih & Hinton; Mikolov et al; Mnih & Kavukcuoglu)

- Scales with corpus size
- Inefficient usage of statistics
- Generate improved performance on other tasks
- Can capture complex patterns beyond word similarity

Word Analogies

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

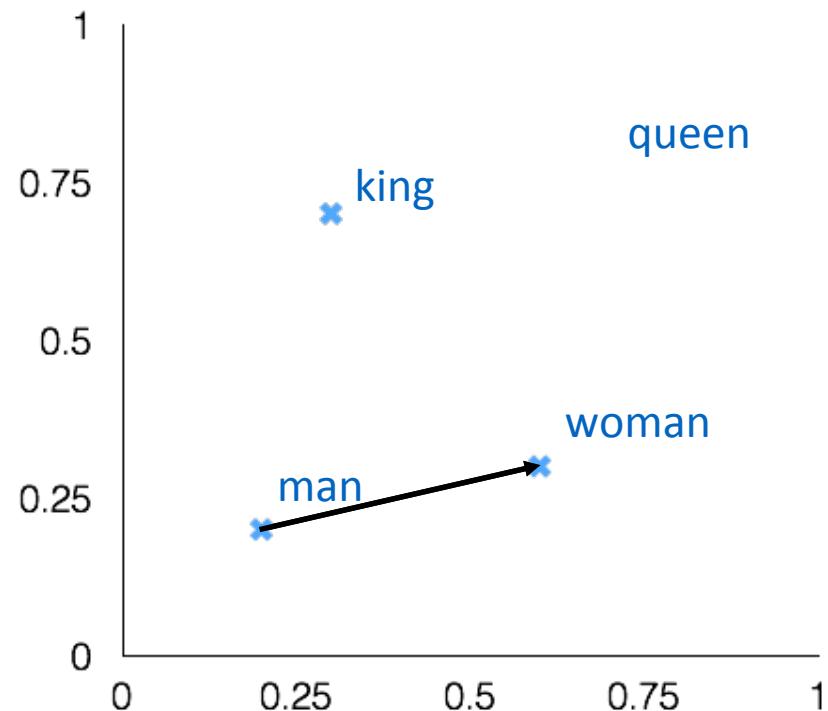
a:b :: c:?



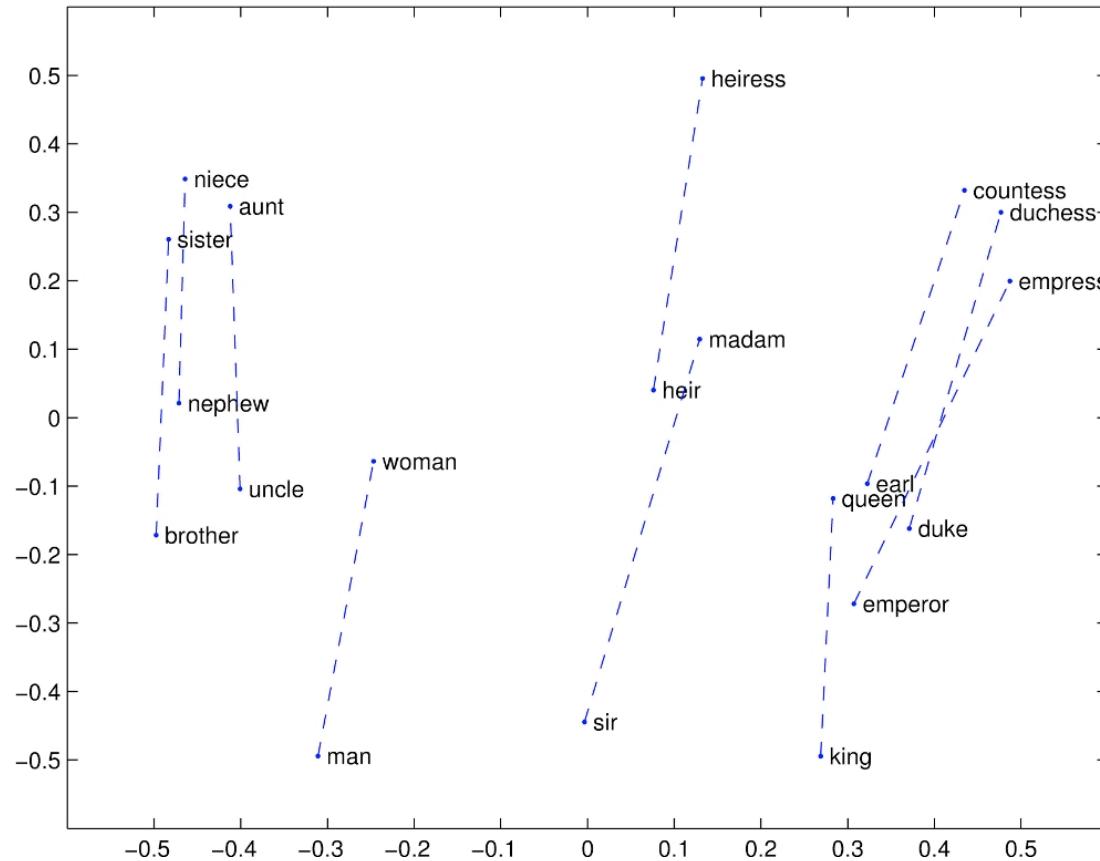
$$d = \arg \max_x \frac{(w_b - w_a + w_c)^T w_x}{\|w_b - w_a + w_c\|}$$

man:woman :: king:?

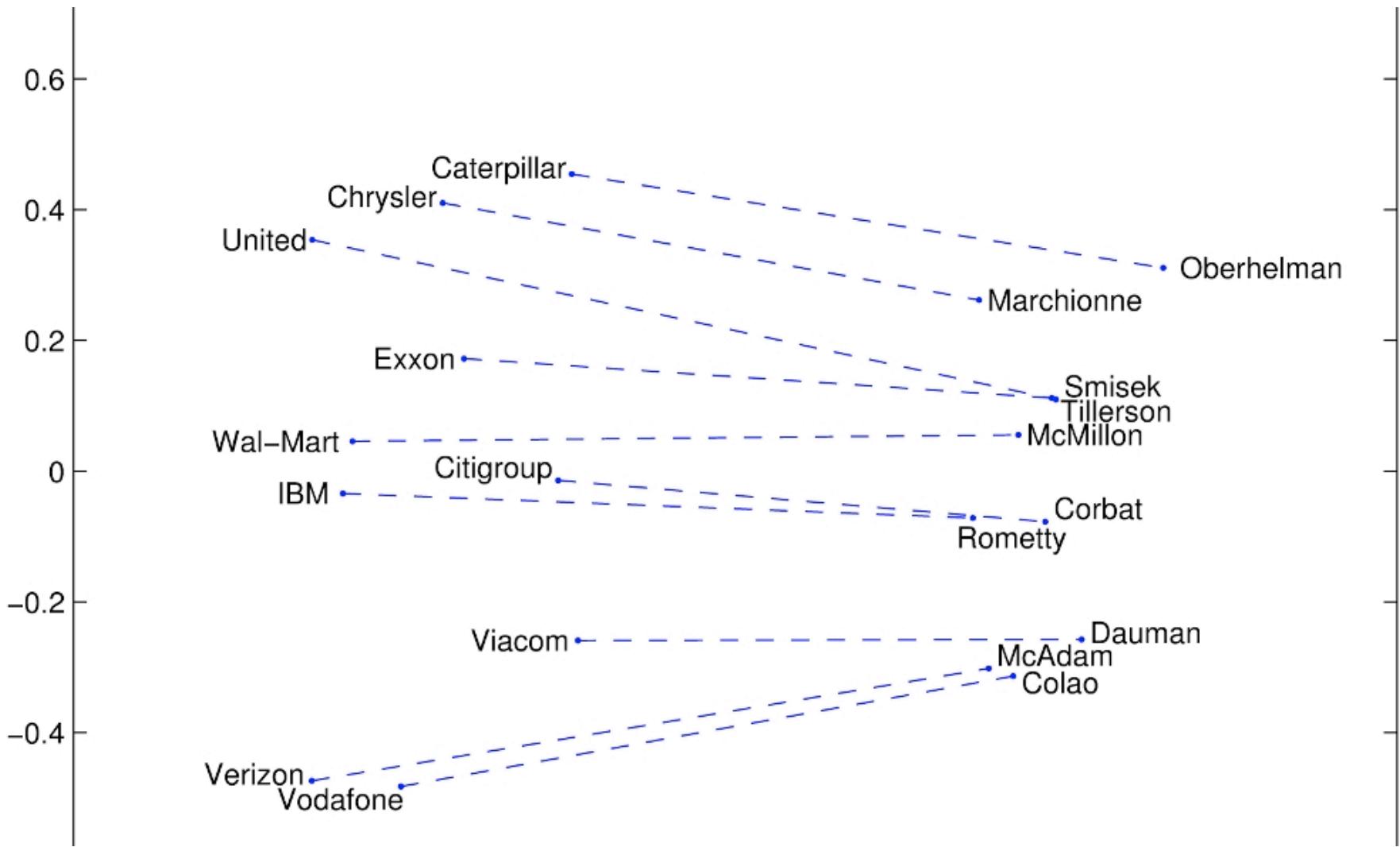
+ king	[0.30 0.70]
- man	[0.20 0.20]
+ woman	[0.60 0.30]
<hr/>	
queen	[0.70 0.80]



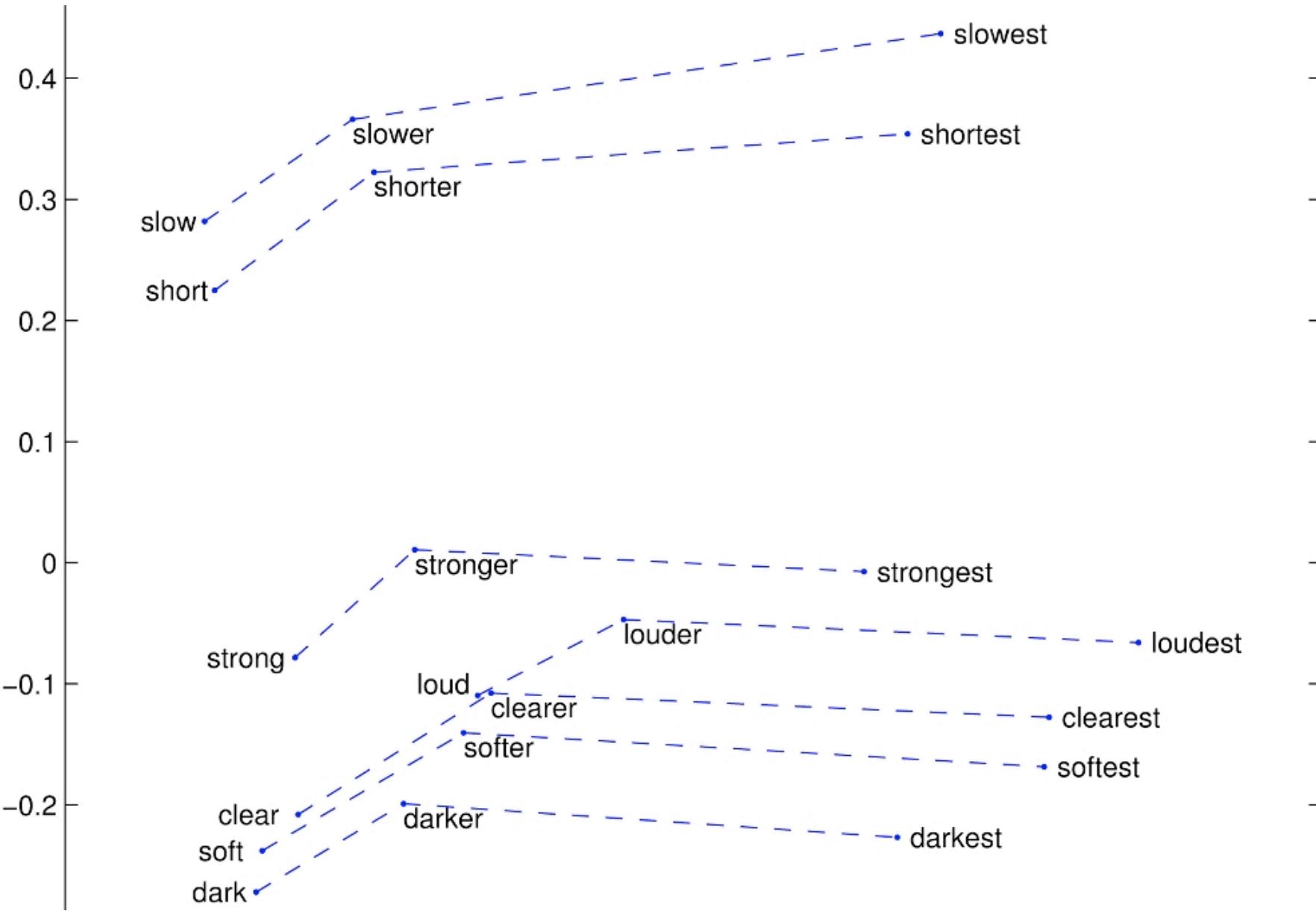
Glove Visualizations



Glove Visualizations: Company - CEO



Glove Visualizations: Superlatives



Word embedding matrix

- Initialize most word vectors of future models with our “pre-trained” embedding matrix $L \in \mathbb{R}^{n \times |V|}$

$$L = \begin{bmatrix} \vdots & \vdots & \vdots & \dots & \vdots & \vdots \\ \vdots & \vdots & \vdots & & \vdots & \vdots \\ \vdots & \vdots & \vdots & & \vdots & \vdots \\ \vdots & \vdots & \vdots & & \vdots & \vdots \\ \vdots & \vdots & \vdots & & \vdots & \vdots \end{bmatrix}^n$$

|V|

aardvark a at ...

- Also called a look-up table
 - Conceptually you get a word’s vector by left multiplying a one-hot vector e (of length $|V|$) by L : $x = Le$

Advantages of low dimensional word vectors

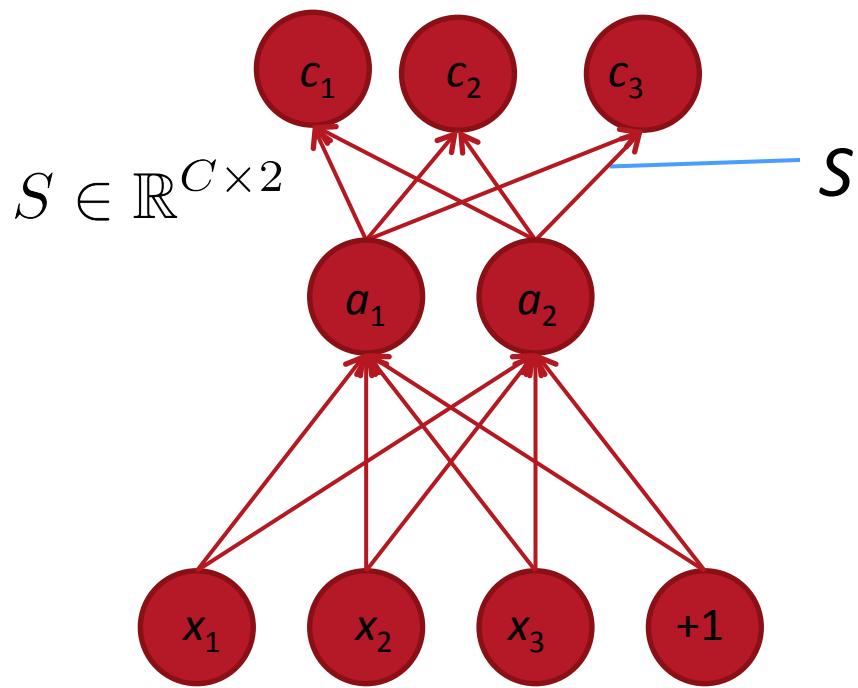
What is the major benefit of deep learned word vectors?

Ability to also propagate **any** information into them via neural networks (next lecture).

$$P(c|d, \lambda) = \frac{e^{\lambda^\top f(c,d)}}{\sum_{c'} e^{\lambda^\top f(c',d)}}$$



$$p(c|x) = \frac{\exp(S_c \cdot a)}{\sum_{c'} \exp(S_{c'} \cdot a)}$$



Advantages of low dimensional word vectors

- Word vectors will form the basis for all subsequent lectures.
- All our semantic representations will be vectors!
- Next lecture:
 - Some more details about word vectors
 - Predict labels for words in context for solving lots of different tasks

Перерыв на демонстрацию Word2Vec

Refresher: The simple word2vec model

- Main cost function J :

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

- With probabilities defined as: $p(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{w=1}^W \exp(u_w^T v_c)}$
- We derived the gradient for the internal vectors v_c

Calculating all gradients!

- We went through gradients for each center vector v in a window
 - We also need gradients for outside vectors u
 - Derive at home!
-
- Generally in each window we will compute updates for all parameters that are being used in that window.
 - For example window size $c = 1$, sentence:
“I like learning.”
 - First window computes gradients for:
 - internal vector v_{like} and external vectors u_i and u_{learning}
 - Next window in that sentence?

Compute all vector gradients!

- 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 Descent

- To minimize $J(\theta)$ over the full batch (the entire training data) would require us to compute gradients for all windows
- Updates would be for each element of μ :

$$\theta_j^{new} = \theta_j^{old} - \alpha \frac{\partial}{\partial \theta_j^{old}} J(\theta)$$

- With step size α
- In matrix notation for all parameters:

$$\theta^{new} = \theta^{old} - \alpha \frac{\partial}{\partial \theta^{old}} J(\theta)$$

$$\theta^{new} = \theta^{old} - \alpha \nabla_{\theta} J(\theta)$$

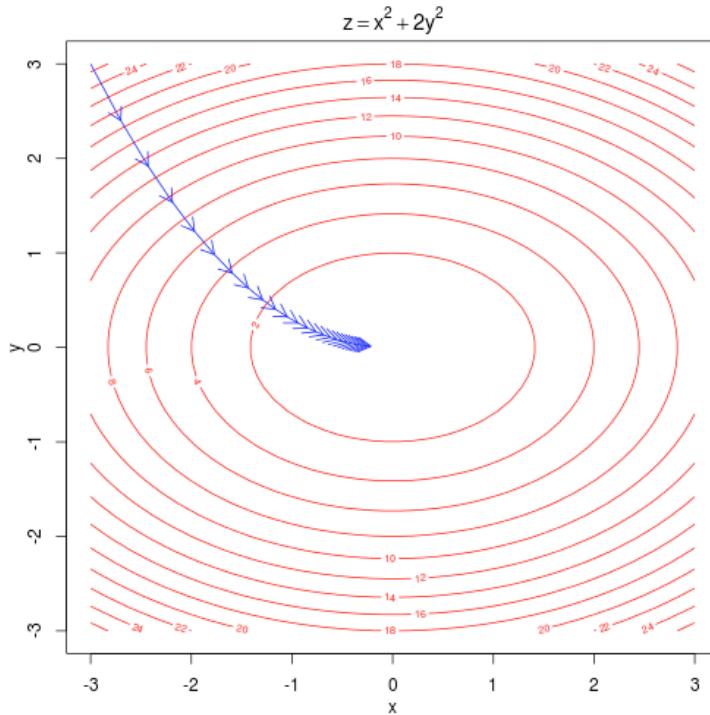
од стандартного Gradient Descent

$$\theta^{new} = \theta^{old} - \alpha \nabla_{\theta} J(\theta)$$

```
while True:  
    theta_grad = evaluate_gradient(J,corpus,theta)  
    theta = theta - alpha * theta_grad
```

Intuition

- For a simple convex function over two parameters.
- Contour lines show levels of objective function



Stochastic Gradient Descent

- But Corpus may have 40B tokens and windows
- You would wait a very long time before making a single update!
- Very bad idea for pretty much all neural nets!
- Instead: We will update parameters after each window t
→ Stochastic gradient descent (SGD)

$$\theta^{new} = \theta^{old} - \alpha \nabla_{\theta} J_t(\theta)$$

```
while True:  
    window = sample_window(corpus)  
    theta_grad = evaluate_gradient(J,window,theta)  
    theta = theta - alpha * theta_grad
```

Stochastic gradients with word vectors!

- But in each window, we only have at most $2c - 1$ words, so $\nabla_{\theta} J_t(\theta)$ is very sparse!

$$\nabla_{\theta} J_t(\theta) = \begin{bmatrix} 0 \\ \vdots \\ \nabla_{v_{like}} \\ \vdots \\ 0 \\ \nabla_{u_I} \\ \vdots \\ \nabla_{u_{learning}} \\ \vdots \end{bmatrix} \in \mathbb{R}^{2dV}$$

Stochastic gradients with word vectors!

- We may as well only update the word vectors that actually appear!
- Solution: either keep around hash for word vectors or only update certain columns of full embedding matrix U and V

$$d \begin{bmatrix} \vdots & \vdots & |V| \\ \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots \end{bmatrix}$$

- Important if you have millions of word vectors and do distributed computing to not have to send gigantic updates around.

Approximations

- The normalization factor is too computationally expensive

$$p(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{w=1}^W \exp(u_w^T v_c)}$$

- Hence, you should implement the skip-gram model
- Main idea: train binary logistic regressions 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)

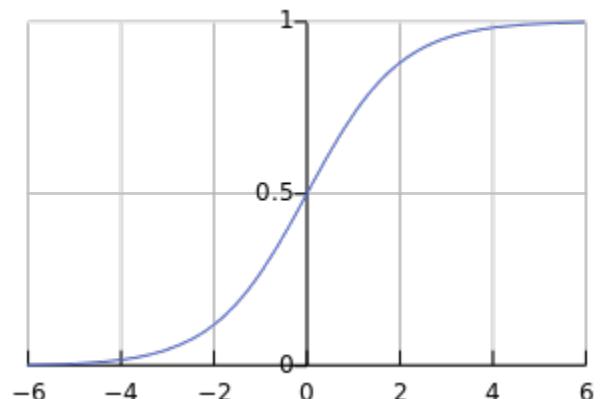
The skip-gram model and negative sampling

- From paper: “Distributed Representations of Words and Phrases and their Compositionality” (Mikolov et al. 2013)
- Overall objective function: $J(\theta) = \frac{1}{T} \sum_{t=1}^T J_t(\theta)$

$$J_t(\theta) = \log \sigma(u_o^T v_c) + \sum_{i=1}^k \mathbb{E}_{j \sim P(w)} [\log \sigma(-u_j^T v_c)]$$

- Where k is the number of negative samples and we use,

- The sigmoid function! $\sigma(x) = \frac{1}{1+e^{-x}}$
(we'll become good friends soon)
- So we maximize the probability
of two words co-occurring in first log
→



The skip-gram model and negative sampling

- Slightly clearer notation:

$$J_t(\theta) = \log \sigma(u_o^T v_c) + \sum_{j \sim P(w)} [\log \sigma(-u_j^T v_c)]$$

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

The continuous bag of words model

- Main idea for continuous bag of words (CBOW): Predict center word from sum of surrounding word vectors instead of predicting surrounding single words from center word as in skip-gram model

Мы разберем имплементацию после лекции.

Count based vs direct prediction

LSA, HAL (Lund & Burgess),
COALS (Rohde et al),
Hellinger-PCA (Lebret & Collobert)

- Fast training
- Efficient usage of statistics
- Primarily used to capture word similarity
- Disproportionate importance given to large counts

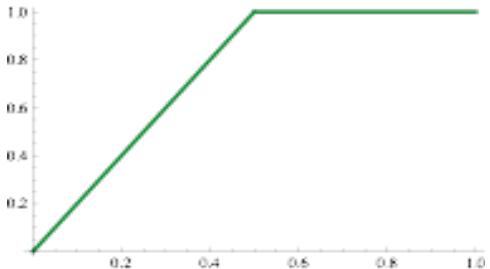
• NNLM, HLBL, RNN, Skip-gram/CBOW, (Bengio et al; Collobert & Weston; Huang et al; Mnih & Hinton; Mikolov et al; Mnih & Kavukcuoglu)

- Scales with corpus size
- Inefficient usage of statistics
- Generate improved performance on other tasks
- Can capture complex patterns beyond word similarity

Combining the best of both worlds: 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

Glove results

Nearest words to
[frog](#):

1. frogs
2. toad
3. litoria
4. leptodactylidae
5. rana
6. lizard
7. eleutherodactylus



[litoria](#)



[leptodactylidae](#)



[rana](#)



[eleutherodactylus](#)

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. It turns out, the best solution is to simply sum them up:

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

- One of many hyperparameters explored in *GloVe: Global Vectors for Word Representation* (Pennington et al. (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 one subsystem with another improves accuracy → Winning!

Intrinsic word vector evaluation

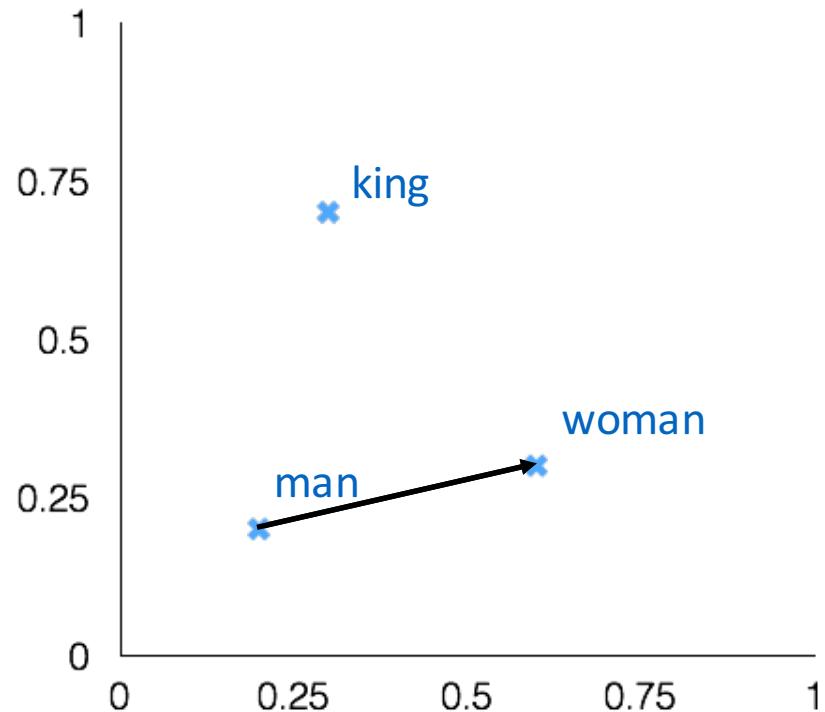
- Word Vector Analogies

a:b :: c:?
man:woman :: king:?

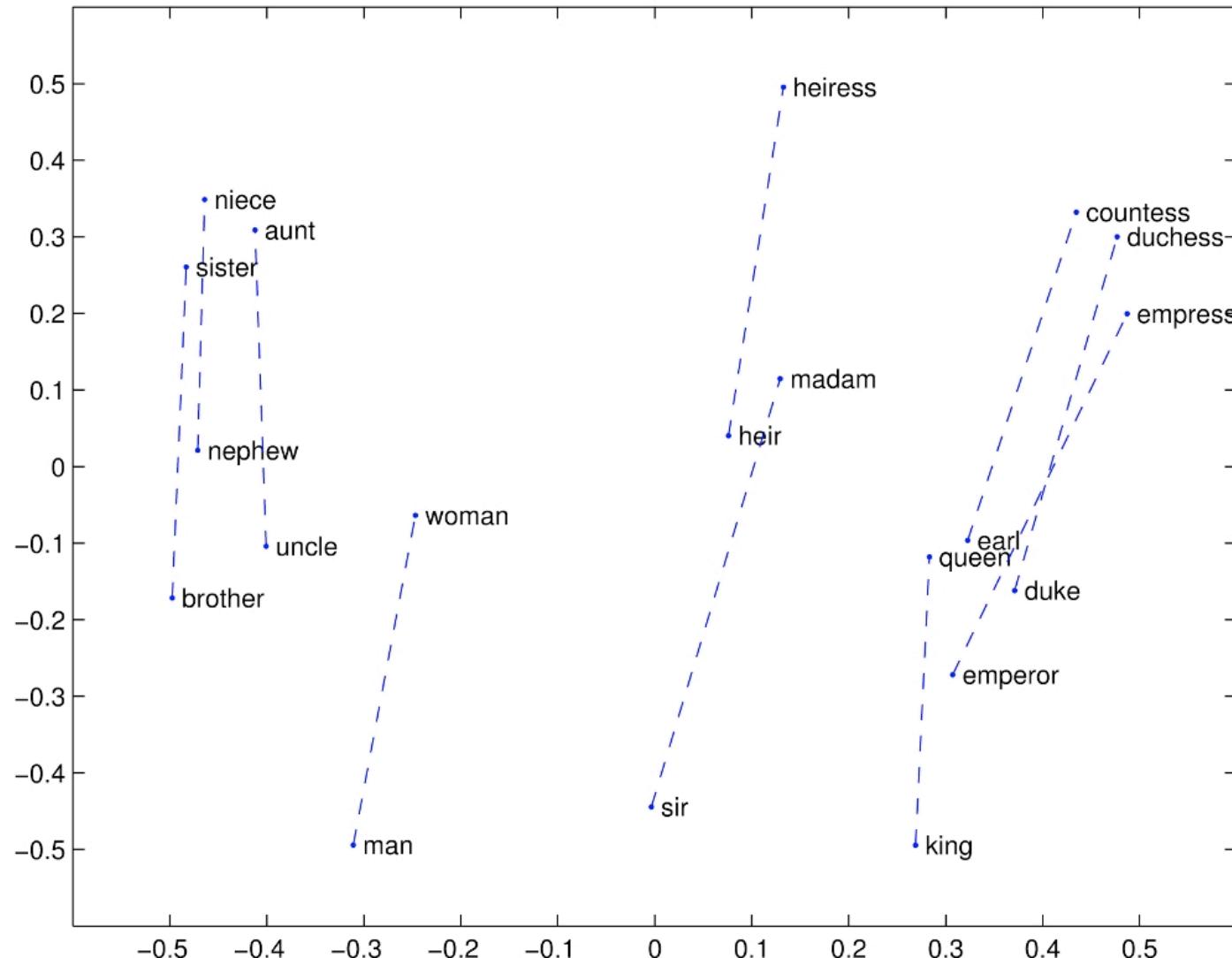


$$d = \arg \max_i \frac{(x_b - x_a + x_c)^T x_i}{\|x_b - x_a + x_c\|}$$

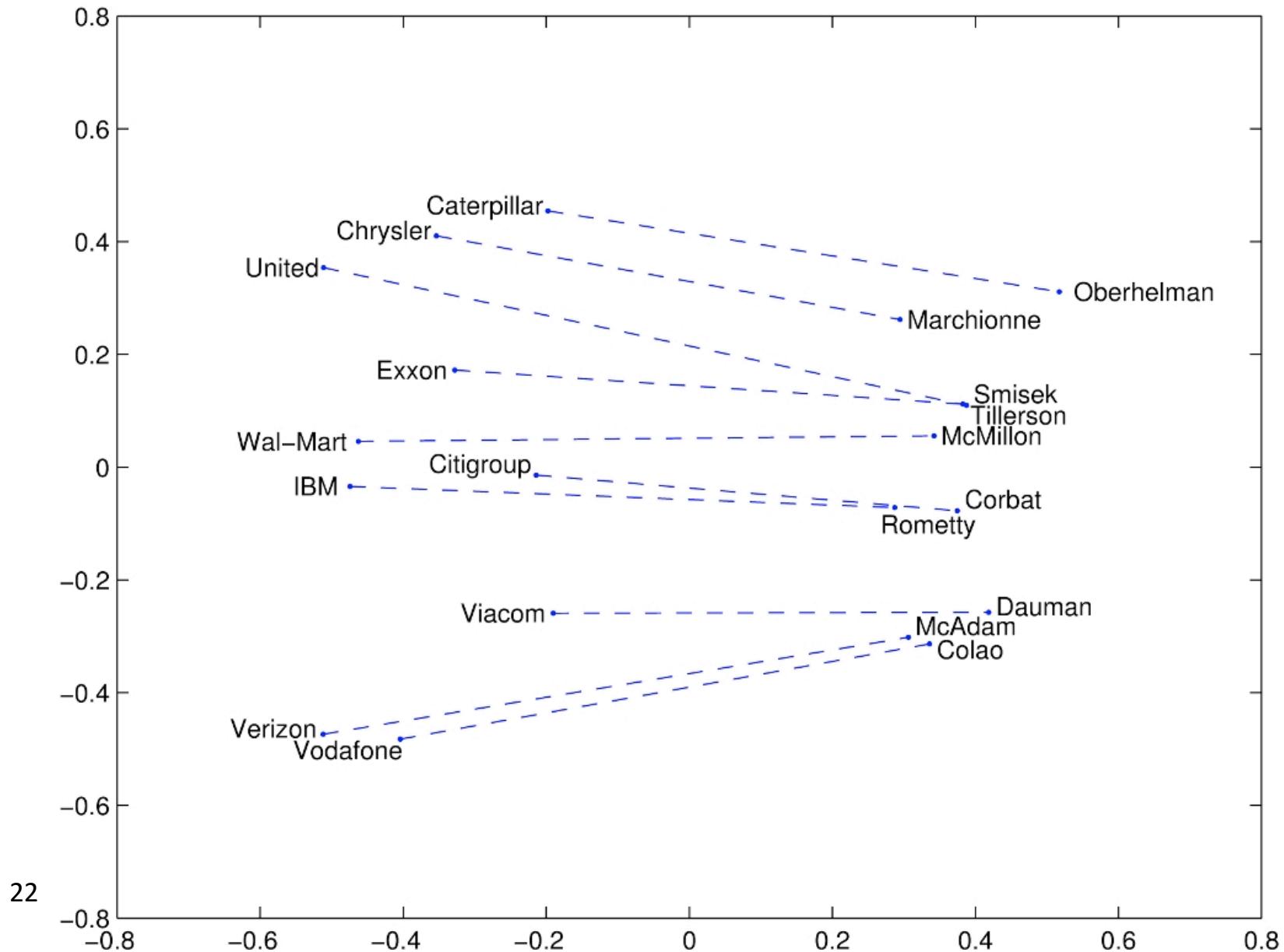
- Evaluate word vectors by how well their cosine distance after addition captures intuitive semantic and syntactic analogy questions
- Discarding the input words from the search!
- Problem: What if the information is there but not linear?



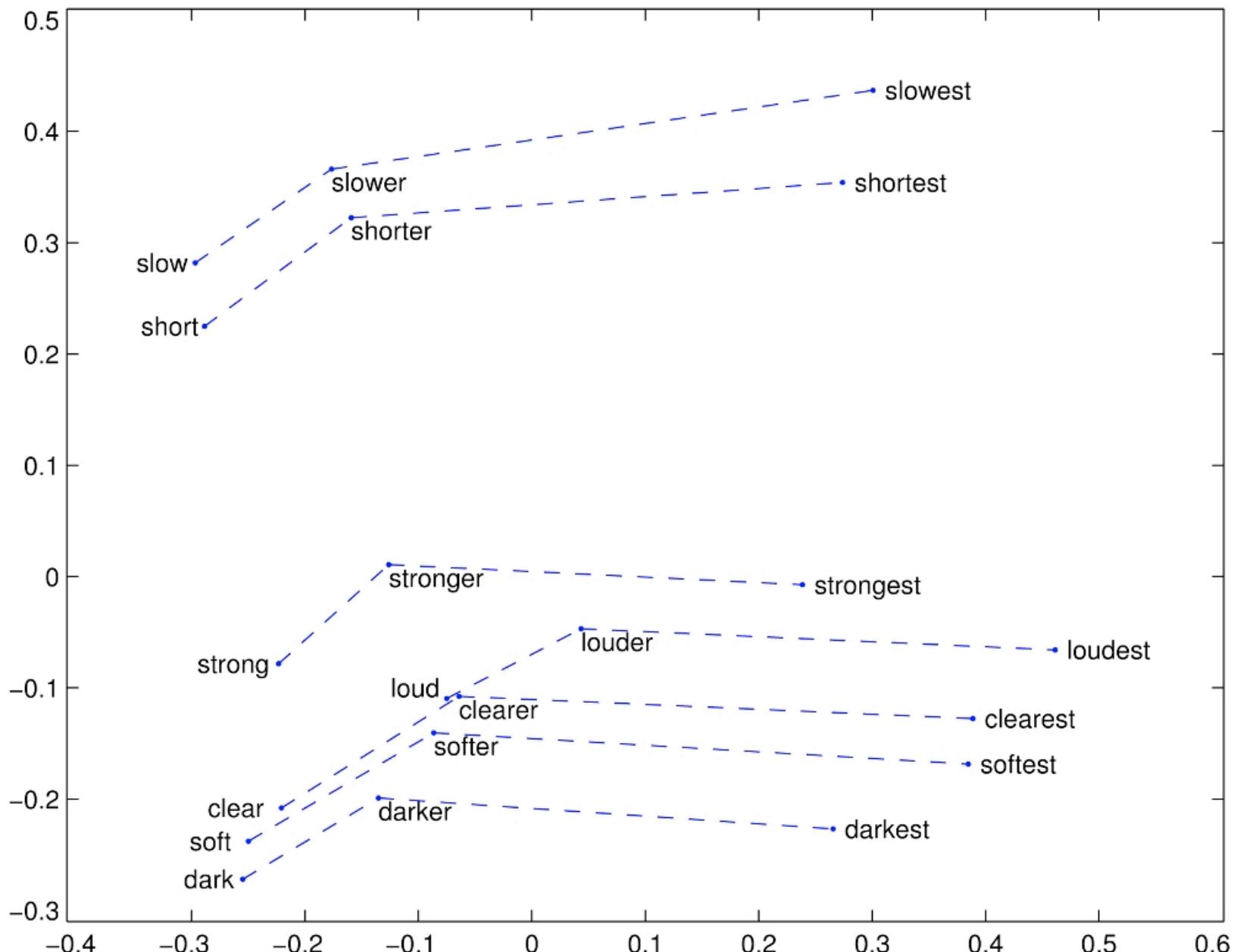
Glove Visualizations



Glove Visualizations: Company - CEO



Glove Visualizations: Superlatives



Details of intrinsic word vector evaluation

- Word Vector Analogies: Syntactic and **Semantic** examples from
<http://code.google.com/p/word2vec/source/browse/trunk/questions-words.txt>

: city-in-state

Chicago Illinois Houston Texas

Chicago Illinois Philadelphia Pennsylvania

Chicago Illinois Phoenix Arizona

Chicago Illinois Dallas Texas

Chicago Illinois Jacksonville Florida

Chicago Illinois Indianapolis Indiana

Chicago Illinois Austin Texas

Chicago Illinois Detroit Michigan

Chicago Illinois Memphis Tennessee

Chicago Illinois Boston Massachusetts

problem: different cities
may have same name

Details of intrinsic word vector evaluation

- Word Vector Analogies: Syntactic and **Semantic** examples from

: capital-world

problem: can change

Abuja Nigeria Accra Ghana

Abuja Nigeria Algiers Algeria

Abuja Nigeria Amman Jordan

Abuja Nigeria Ankara Turkey

Abuja Nigeria Antananarivo Madagascar

Abuja Nigeria Apia Samoa

Abuja Nigeria Ashgabat Turkmenistan

Abuja Nigeria Asmara Eritrea

Abuja Nigeria Astana Kazakhstan

Details of intrinsic word vector evaluation

- Word Vector Analogies: **Syntactic** and Semantic examples from

: gram4-superlative
bad worst big biggest
bad worst bright brightest
bad worst cold coldest
bad worst cool coolest
bad worst dark darkest
bad worst easy easiest
bad worst fast fastest
bad worst good best
bad worst great greatest

Details of intrinsic word vector evaluation

- Word Vector Analogies: **Syntactic** and Semantic examples from

: gram7-past-tense

dancing danced decreasing decreased

dancing danced describing described

dancing danced enhancing enhanced

dancing danced falling fell

dancing danced feeding fed

dancing danced flying flew

dancing danced generating generated

dancing danced going went

dancing danced hiding hid

dancing danced hitting hit

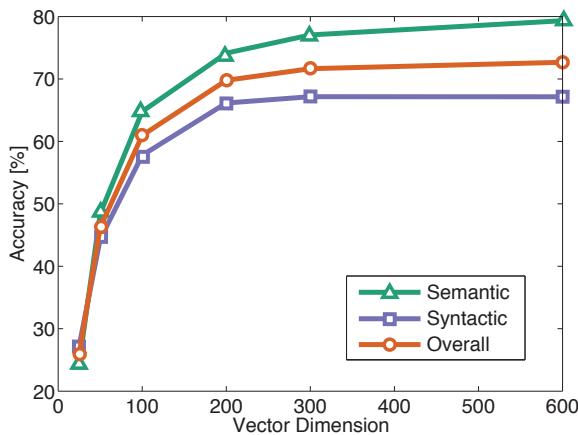
Analogy evaluation and hyperparameters

- Very careful analysis: Glove word vectors

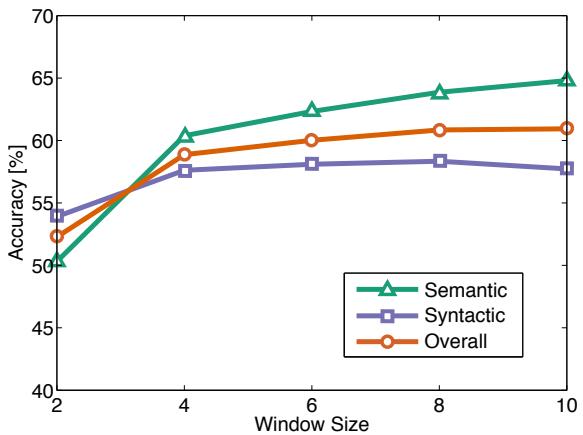
Model	Dim.	Size	Sem.	Syn.	Tot.
ivLBL	100	1.5B	55.9	50.1	53.2
HPCA	100	1.6B	4.2	16.4	10.8
GloVe	100	1.6B	<u>67.5</u>	<u>54.3</u>	<u>60.3</u>
SG	300	1B	61	61	61
CBOW	300	1.6B	16.1	52.6	36.1
vLBL	300	1.5B	54.2	<u>64.8</u>	60.0
ivLBL	300	1.5B	65.2	63.0	64.0
GloVe	300	1.6B	<u>80.8</u>	61.5	<u>70.3</u>
SVD	300	6B	6.3	8.1	7.3
SVD-S	300	6B	36.7	46.6	42.1
SVD-L	300	6B	56.6	63.0	60.1
CBOW [†]	300	6B	63.6	<u>67.4</u>	65.7
SG [†]	300	6B	73.0	66.0	69.1
GloVe	300	6B	<u>77.4</u>	67.0	<u>71.7</u>
CBOW	1000	6B	57.3	68.9	63.7
SG	1000	6B	66.1	65.1	65.6
SVD-L	300	42B	38.4	58.2	49.2
GloVe	300	42B	<u>81.9</u>	<u>69.3</u>	<u>75.0</u>

Analogy evaluation and hyperparameters

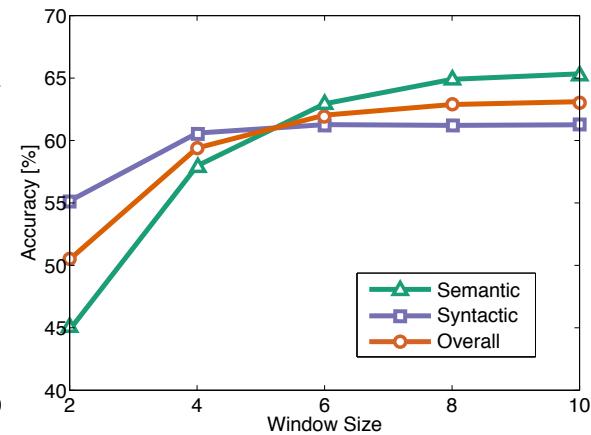
- Asymmetric context (only words to the left) are not as good



(a) Symmetric context



(b) Symmetric context

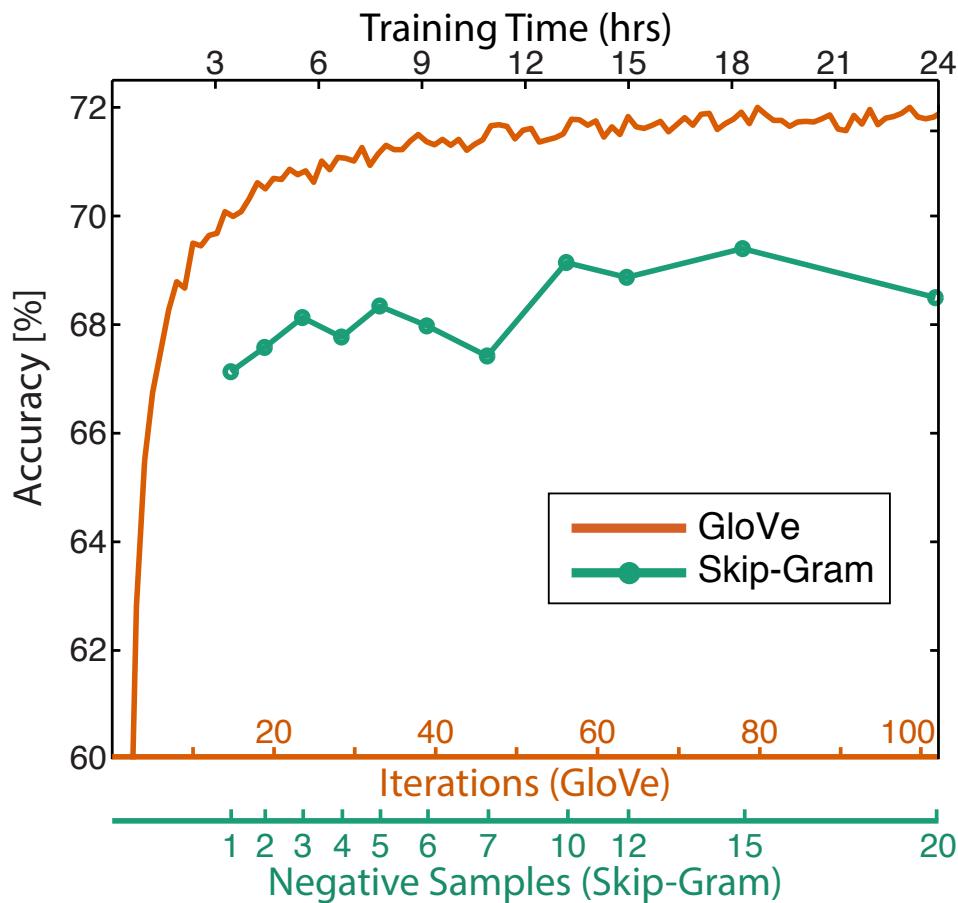


(c) Asymmetric context

- 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

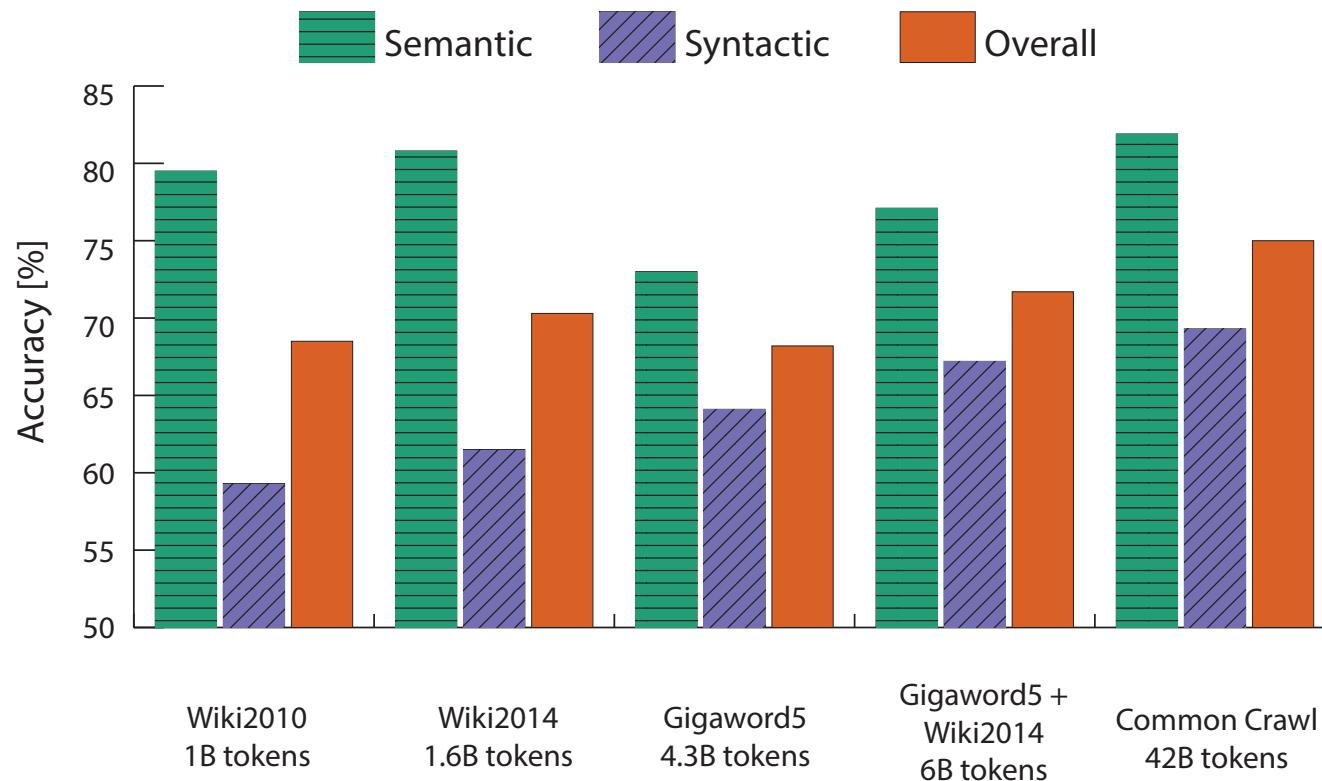
Analogy evaluation and hyperparameters

- More training time helps



Analogy evaluation and hyperparameters

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



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

Correlation evaluation

- Word vector distances and their correlation with human judgments

Model	Size	WS353	MC	RG	SCWS	RW
SVD	6B	35.3	35.1	42.5	38.3	25.6
SVD-S	6B	56.5	71.5	71.0	53.6	34.7
SVD-L	6B	65.7	<u>72.7</u>	75.1	56.5	37.0
CBOW [†]	6B	57.2	65.6	68.2	57.0	32.5
SG [†]	6B	62.8	65.2	69.7	<u>58.1</u>	37.2
GloVe	6B	<u>65.8</u>	<u>72.7</u>	<u>77.8</u>	53.9	<u>38.1</u>
SVD-L	42B	74.0	76.4	74.1	58.3	39.9
GloVe	42B	75.9	83.6	82.9	59.6	47.8
CBOW*	100B	68.4	79.6	75.4	59.4	45.5

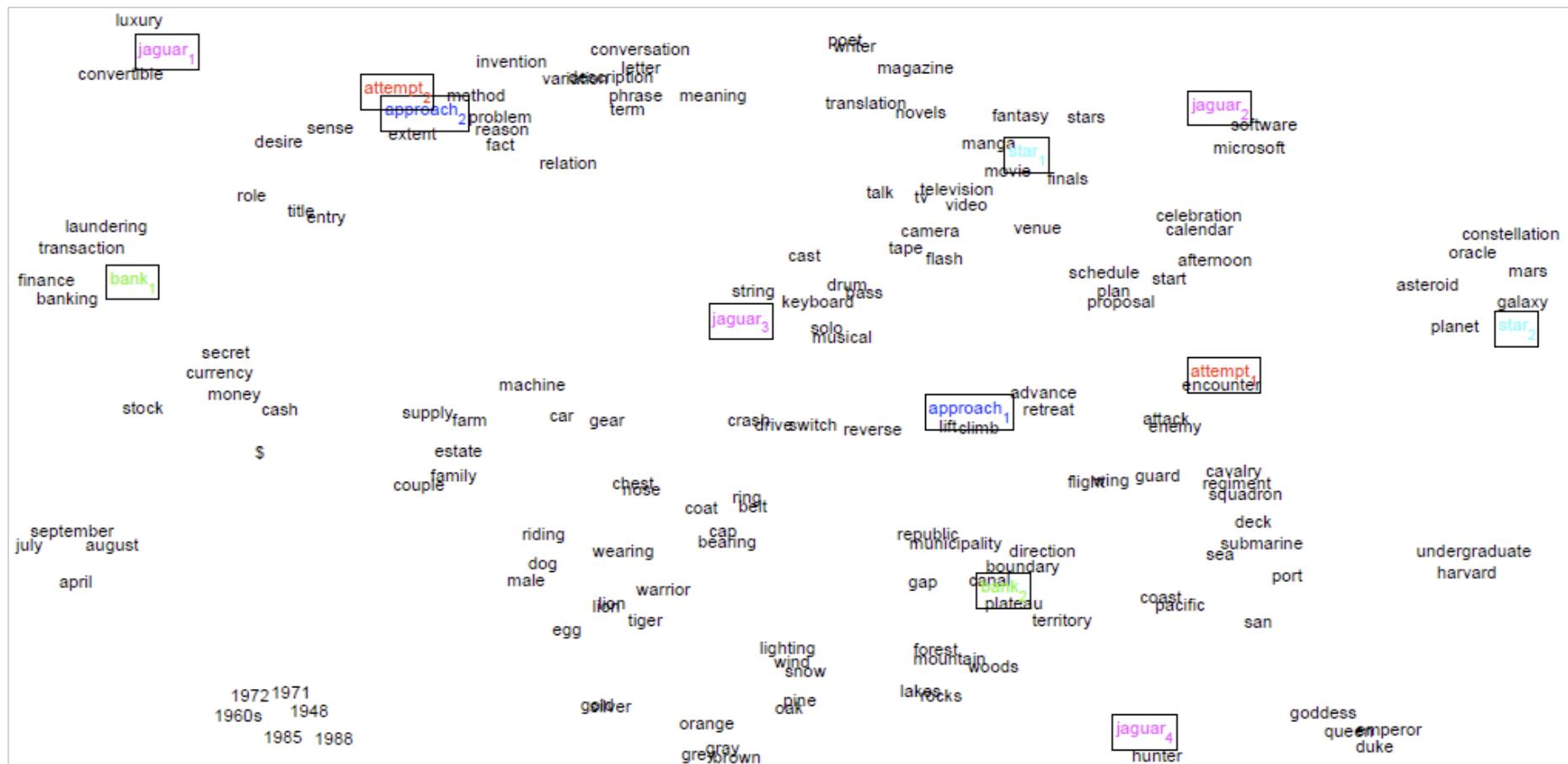
- Some ideas from Glove paper have been shown to improve skip-gram (SG) model also (e.g. sum both vectors)

But what about ambiguity?

- You may hope that one vector captures both kinds of information (run = verb and noun) but then vector is pulled in different directions
- Alternative described in: *Improving Word Representations Via Global Context And Multiple Word Prototypes* (Huang et al. 2012)
- Idea: Cluster word windows around words, retrain with each word assigned to multiple different clusters $\text{bank}_1, \text{bank}_2, \text{etc}$

But what about ambiguity?

- Improving Word Representations Via Global Context And Multiple Word Prototypes (Huang et al. 2012)



Extrinsic word vector evaluation

- Extrinsic evaluation of word vectors: All subsequent tasks in this class
- One example where good word vectors should help directly: named entity recognition: finding a person, organization or location

Model	Dev	Test	ACE	MUC7
Discrete	91.0	85.4	77.4	73.4
SVD	90.8	85.7	77.3	73.7
SVD-S	91.0	85.5	77.6	74.3
SVD-L	90.5	84.8	73.6	71.5
HPCA	92.6	88.7	81.7	80.7
HSMN	90.5	85.7	78.7	74.7
CW	92.2	87.4	81.7	80.2
CBOW	93.1	88.2	82.2	81.1
GloVe	93.2	88.3	82.9	82.2

- Next: How to use word vectors in neural net models!

Simple single word classification

- What is the major benefit of deep learned word vectors?
 - Ability to also classify words accurately
 - Countries cluster together → classifying location words should be possible with word vectors
 - Incorporate **any** information into them other tasks
 - Project sentiment into words to find most positive/negative words in corpus

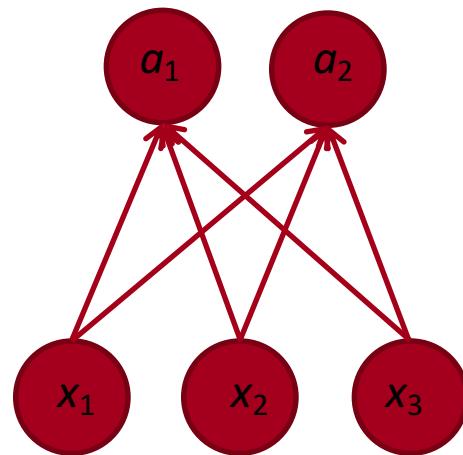
The softmax

Logistic regression = Softmax classification on word vector x to obtain probability for class y :

$$p(y|x) = \frac{\exp(W_y \cdot x)}{\sum_{c=1}^C \exp(W_c \cdot x)}$$

where: $W \in \mathbb{R}^{C \times d}$

Generalizes >2 classes
(for just binary sigmoid unit would suffice as in skip-gram)



The softmax - details

- Terminology: Loss function = cost function = objective function
- Loss for softmax: Cross entropy
- To compute $p(y|x)$: first take the y 'th row of W and multiply that with row with x :

$$W_y \cdot x = \sum_{i=1}^d W_{yi} x_i = f_y$$

- Compute all f_c for $c=1,\dots,C$
- Normalize to obtain probability with softmax function:

$$p(y|x) = \frac{\exp(f_y)}{\sum_{c=1}^C \exp(f_c)}$$

The softmax and cross-entropy error

- The loss wants to maximize the probability of the correct class y
- Hence, we minimize the negative log probability of that class:

$$-\log p(y|x) = -\log \left(\frac{\exp(f_y)}{\sum_{c=1}^C \exp(f_c)} \right)$$

- As before: we sum up multiple cross entropy errors if we have multiple classifications in our total error function over the corpus

Background: The Cross entropy error

- Assuming a ground truth (or gold or target) probability distribution that is 1 at the right class and 0 everywhere else: $p = [0, \dots, 0, 1, 0, \dots, 0]$ and our computed probability is q , then the cross entropy is:

$$H(p, q) = - \sum_{c=1}^C p(c) \log q(c)$$

- Because of one-hot p , the only term left is the negative probability of the true class
- Cross-entropy can be re-written in terms of the entropy and Kullback-Leibler divergence between the two distributions:

$$H(p, q) = H(p) + D_{KL}(p||q)$$

The KL divergence

- Cross entropy: $H(p, q) = H(p) + D_{KL}(p||q)$
- Because p is zero in our case (and even if it wasn't it would be fixed and have no contribution to gradient), to minimize this is equal to minimizing the KL divergence
- The KL divergence is **not a distance** but a non-symmetric measure of the difference between two probability distributions p and q

$$D_{KL}(p||q) = \sum_{c=1}^C p(c) \log \frac{p(c)}{q(c)}$$

Simple single word classification

- Example: Sentiment
- Two options: train only *softmax* weights W 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

Visualization of sentiment trained word vectors

