CS 584 Natural Language Processing

Vector Semantics

Department of Computer Science
Yue Ning
yue.ning@stevens.edu

Late Submission Policy

- **10%** penalty for late submission within **24 hours**.
- **40%** penalty for late submissions within **24-48 hours**.
- After 48 hours, you get **NO** points on the assignment.

Bias and Variance

- Assuming a training set($x_1,\dots,x_n$) and their real associated y values.
- There is a function with noise y=f(x)+e where the noise e has mean 0 and variance σ^2.
- We want to find a function p(x) that approximates the true function f(x).
- Expected squared prediction error at a point x is:

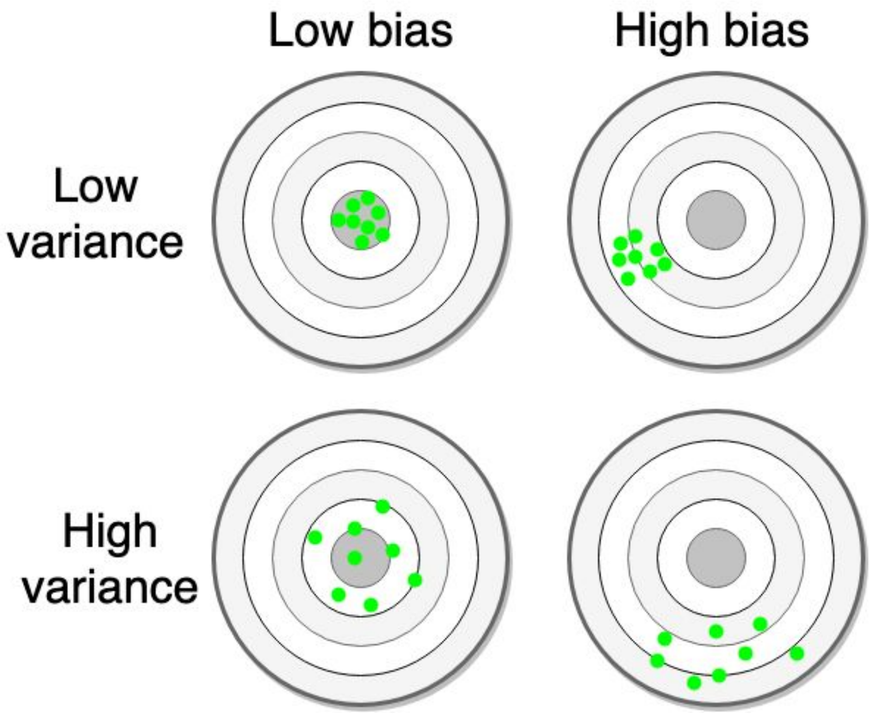
$$
\begin{aligned}
E[(y-p(x))^2] &= (E[p(x)] - f(x))^2 + (E[p(x)^2] - E^2[p(x)]) + \sigma^2 \\
&= \mathrm{Bias}^2 + \mathrm{Variance} + \sigma^2
\end{aligned}
$$

$$\mathrm{Bias}[p(x)] = E[p(x)] - f(x), \mathrm{Var}[p(x)] = E[p(x)^2] - E^2[p(x)]$$

➔ Bias: error from incorrect modeling assumption

➔ Variance: error from random noise

Bias and Variance



Diagnosis:

- If you model cannot even fit the training examples, then you have large bias (underfitting)

- If you can fit the training examples, but large error on testing data, then you probably have large variance (overfitting)

Figure from: https://www.machinelearningtutorial.net/2017/01/26/the-bias-variance-tradeoff/

What to do with large bias or variance?

Large bias:

- redesign your model
- add more features as input
- a more complex model

Large variance:

- more data (usually effective, but not always practical)
- regularization

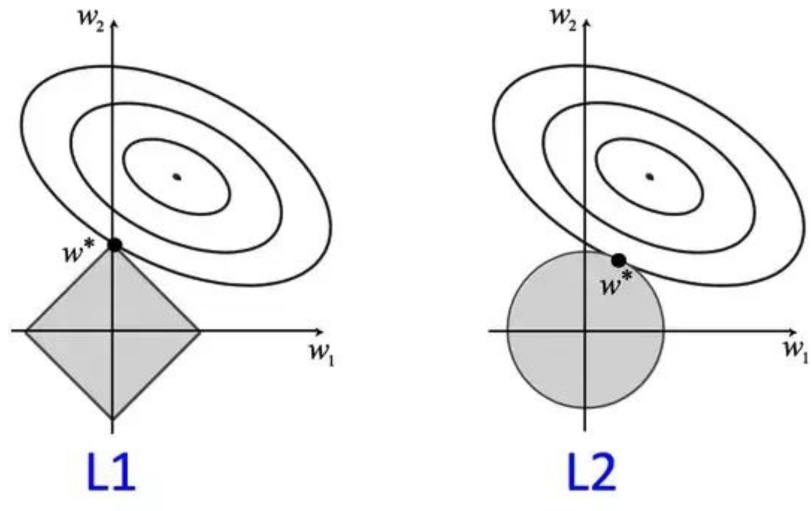
Regularization

L2 regularization for regression:

$$L_2(X, y, \mathbf{w}) = \sum_{n=1}^{N}(f_w(x_n, y_n, \mathbf{w}) - y_n)^2 + \underbrace{\alpha \mathbf{w}^T \mathbf{w}}_{\text{L2 regularization}}$$

L1 regularization for regression:

$$L_1(X, y, \mathbf{w}) = \sum_{n=1}^{N}(f_w(x_n, y_n, \mathbf{w}) - y_n)^2 + \underbrace{\alpha |\mathbf{w}|}_{\text{L1 regularization}}$$



Represent the meaning of a word

- 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 word of writing, art, etc.

- Commonest linguistic way of thinking of meaning:
  - signifier (symbol) <-> signified (idea or thing)

= denotational semantics

Usable meaning in a computer?

- Common solution: use e.g. WordNet, a thesaurus containing lists of synonym sets and hypernyms (“is a” relationships)

e.g. synonym sets containing “good”:

```
from nltk.corpus import wordnet as wn
poses = { 'n':'noun', 'v':'verb', 's':'adj (s)', 'a':'adj', 'r':'adv'}
for synset in wn.synsets("good"):
    print("{}: {}".format(poses[synset.pos()],
          ", ".join([l.name() for l in synset.lemmas()])))
```

```
noun: good
noun: good, goodness
noun: good, goodness
noun: commodity, trade_good, good
adj: good
adj (sat): full, good
adj: good
adj (sat): estimable, good, honorable, respectable
adj (sat): beneficial, good
adj (sat): good
adj (sat): good, just, upright
…
adverb: well, good
adverb: thoroughly, soundly, good
```

e.g. hypernyms of “panda”:

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

Problems with WordNet

- Great as a resource but missing nuance
  - e.g. “proficient” is listed as a synonym for “good”. This is only correct in some contexts.
- Missing new meanings of words
  - e.g., wicked, badass, nifty, wizard, genius, ninja, bombest Impossible to keep up-to-date!
- Subjective
- Requires human labor to create and adapt
- Can’t compute accurate word similarity

Similarity Metrics of Vectors

- Cosine similarity

$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\| \|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \sqrt{\sum_{i=1}^{n} B_i^2}},$$

- Inner product (dot product)

https://dataaspirant.com/2015/04/11/five-most-popular-similarity-measures-implementation-in-python/

Representing words as discrete symbols

- In traditional NLP, we regard words as discrete symbols: hotels, conference, motel - a locallist representation

Means one 1, the rest 0s

- Words can be represented by one-hot vectors:
  - motel = [0 0 0 0 0 0 0 0 0 0 1 0 0 0 0]
  - hotel = [0 0 0 0 0 0 0 1 0 0 0 0 0 0 0]
- Vector dimension = number of words in vocabulary (e.g. 500, 000)

Representing documents as bag of words

- Bag of words: count, TFIDF, etc

- Documents can be represented by vectors:
  - doc1 = [0 0 3 0 2 0 0 0 0 0 5 0 0 1 0]
  - doc2 = [0 1 0 0 0 2 0 0 4 0 0 1 1 0 0]
- Vector dimension = number of words in vocabulary (e.g. 500, 000)

Problem with words as discrete symbols

- Example: in web search, if user searches for "Seattle motel", we would like to match documents containing "Seattle hotel"
- But
  - motel = [0 0 0 0 0 0 0 0 0 0 1 0 0 0 0]
  - hotel = [0 0 0 0 0 0 0 1 0 0 0 0 0 0 0]
- These two vectors are orthogonal. There is no natural notion of similarity of one-hot vectors.
- Solution:
  - Could try to rely on WordNet's list of synonyms to get similarity?
    - but it is well-known to fail badly: incompleteness, etc
    - instead: learn to encode similarity in the vectors themselves

Representing words by their context

- Distributional semantics: A word's meaning is given by the words that frequently appear close-by
  - "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!
- When a word w appears in a text, its context is the set of words that appear nearby (within a fixed-size window).
- Use the many contexts of w to build up a representation of w

…government debt problems turning into	***banking***	*crises as happened in 2009…*
…saying that Europe needs unified	***banking***	*regulation to replace the hodgepodge…*
…India has just given its	***banking***	*system a shot in the arm…*

These context words will represent ***banking***

Word vectors

- We will build a dense vector for each word, chosen so that it is similar to vectors of words that appear in similar context.

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

- Note: word vectors are sometimes called word embeddings or word representations. They are a distributed representation.

Word meaning as a word vector
- visualization

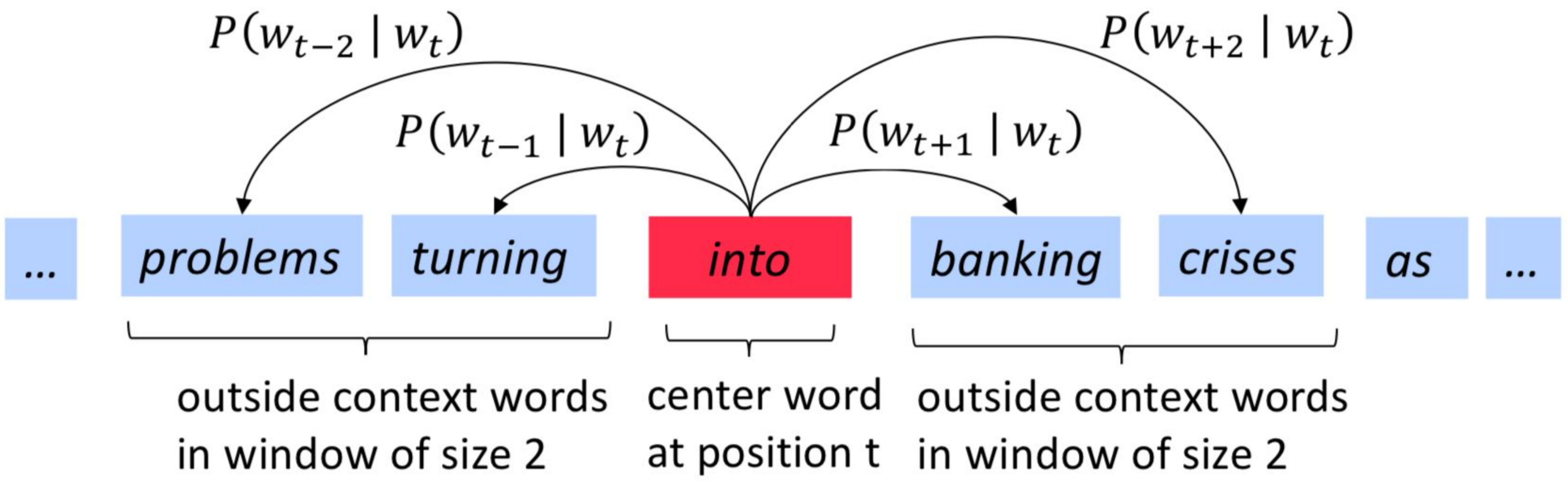
$$\textit{expect} = \begin{pmatrix} 0.286 \\ 0.792 \\ -0.177 \\ -0.107 \\ 0.109 \\ -0.542 \\ 0.349 \\ 0.271 \\ 0.487 \end{pmatrix}$$

Word2vec: Overview

- Word2vec (Mikolov et al. 2013) is a framework for learning word vectors
- Idea:
  - We have a large corpus of text
  - Every word in a fixed vocabulary is represented by a vector
  - Go through each position t in the text, which has a center word c and context ("outside") words o
  - Use the similarity of the word vectors for c and o to calculate the probability of o given c (or vice versa)
  - Keep adjusting the word vectors to maximize this probability

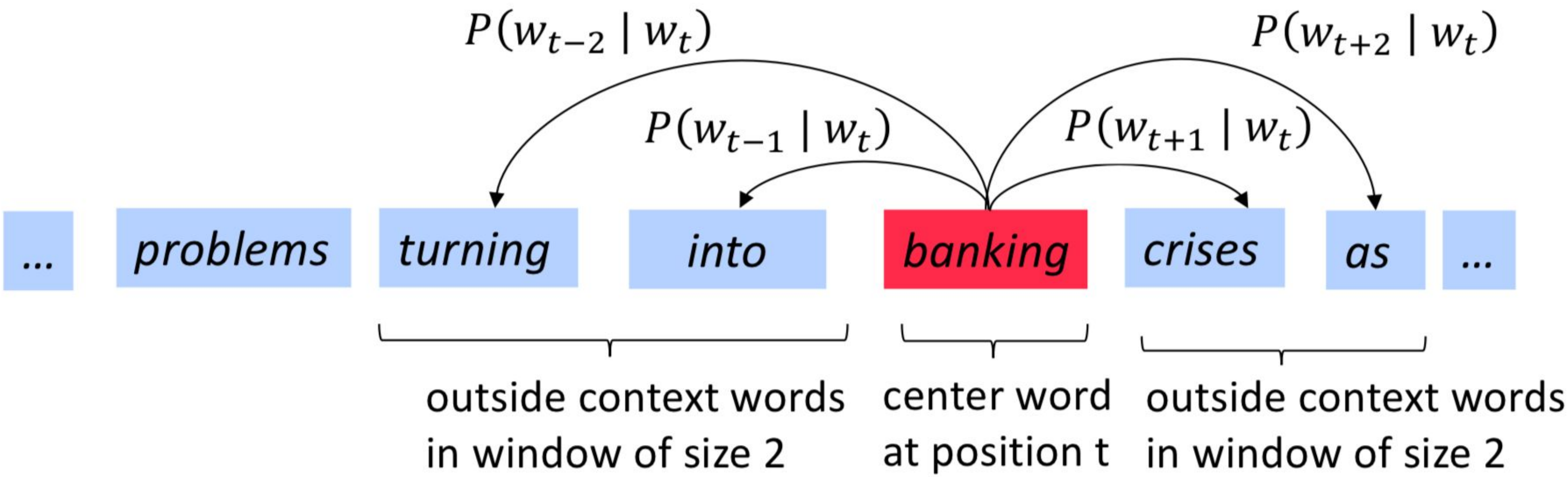
Word2vec: Overview

- Example windows and process for computing $P(w_{t+j}|w_t)$



Word2vec: Overview

- Example windows and process for computing $P(w_{t+j}|w_t)$



Word2vec: objective function

- For each position t = 1,...,T, predict context words within a window of fixed size *m*, given center word w_j.

$$\text{Likelihood} = L(\theta) = \prod_{t=1}^{T} \prod_{-m \le j \le m, j \ne 0} P(w_{t+j} | w_t; \theta)$$

- The objective function J is the (average) negative log likelihood

$$J(\theta) = -\frac{1}{T} \log L(\theta) = -\frac{1}{T} \sum_{t=1}^{T} \sum_{-m \le j \le m, j \ne 0} \log P(w_{t+j} | w_t; \theta)$$

Minimizing objective function ⟺ Maximizing predictive accuracy

Word2vec overview with vectors

- We want to minimize the objective function:

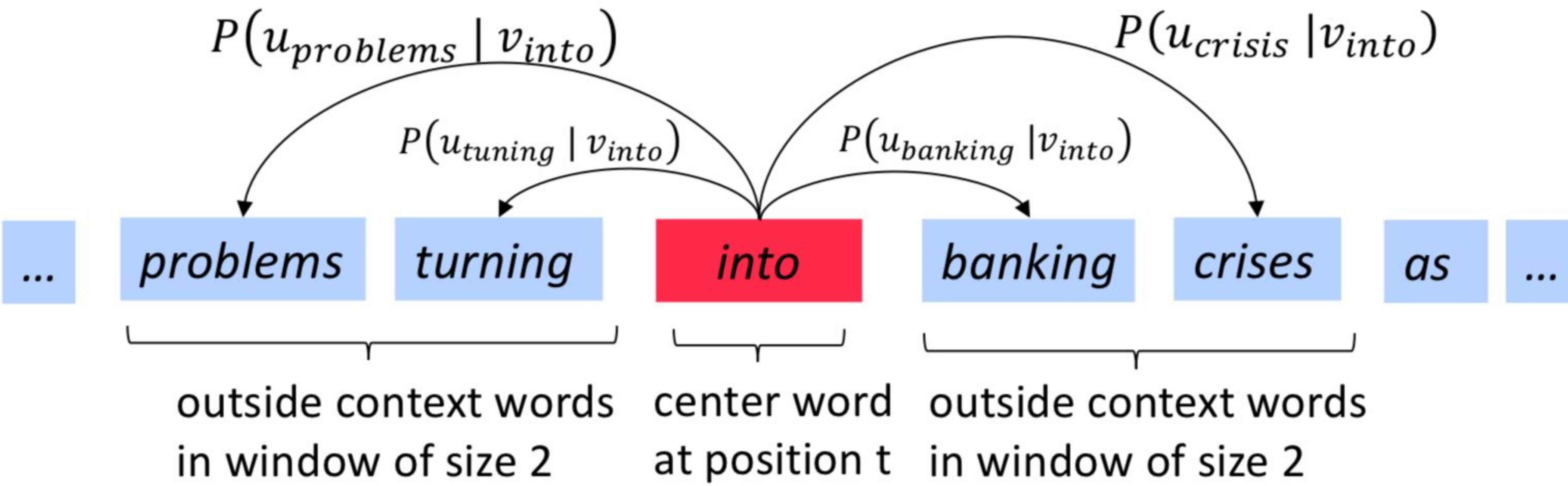
$$J(\theta) = -\frac{1}{T}\log L(\theta) = -\frac{1}{T}\sum_{t=1}^{T}\sum_{-m\leq j\leq m, j\neq 0}\log P(w_{t+j}|w_t;\theta)$$

- Question: how to calculate $P(w_{t+j} \mid w_t; \theta)$?
- Answer: we will use two vectors per word w:
  - v_w when w is a center word
  - u_w when w is a context word
- Then for a center word c and a context word o:

$$P(o|c) = \frac{\exp(u_o^\top v_c)}{\sum_{w\in V}\exp(u_w^\top v_c)}$$

Word2vec: objective function

- Example windows and process for comput $P(w_{t+j}|w_t)$
- $P(u_{problems} \mid v_{into})$ short for $P(problems|into;\ u_{problems},\ v_{into},\theta)$



Word2vec: prediction function

Exponentiation makes anything positive

Dot product compares similarity of o and c.
larger dot product = larger probability

Normalize over entire vocabulary to give probability distribution

$$P(o|c) = \frac{\exp(\boxed{u_o^\top v_c})}{\boxed{\sum_{w \in V} \exp(u_w^\top v_c)}}$$

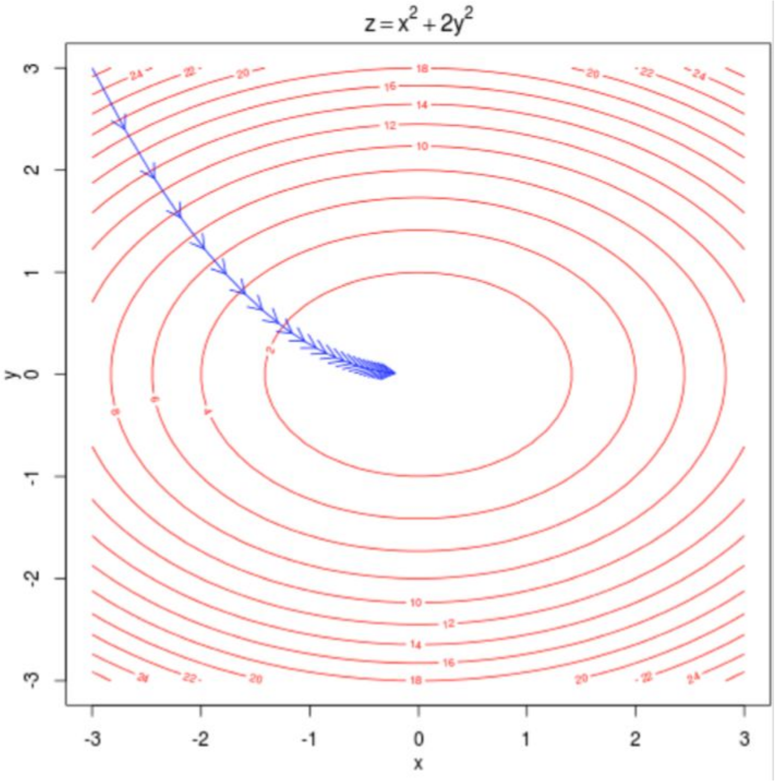
- This is an example of the softmax function

$$\text{softmax}(x_i) = \frac{\exp(x_i)}{\sum_{j=1}^{n} \exp(x_j)} = p_i$$

- The softmax function maps arbitrary values xi to a probability distribution pi
  - "max" because amplifies probability of largest xi
  - "soft" because still assigns some probability to smaller xi
  - frequently used in deep learning

Training a model by optimizing parameters

- To train a model, we adjust parameters to minimize a loss e.g. below, for a simple convex function over two parameters. Contour lines show levels of objective function



To train the model: compute all vector gradients

- Recall: θ represents all model parameters, in one long vector
- In our case with d-dimensional vectors and V-many words:

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

- Remember: every word has two vectors
- We optimize these parameters by walking down the gradient

Word2vec derivations of gradient

- Recall: θ represents all model parameters, in one long vector
- In our case with d-dimensional vectors and V-many words:

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

- Remember: every word has two vectors
- We optimize these parameters by walking down the gradient

Word2vec derivations of gradient

- Useful basics:

$$\frac{\partial \mathbf{x}^\top \mathbf{a}}{\partial \mathbf{x}} = \frac{\partial \mathbf{a}^\top \mathbf{x}}{\partial \mathbf{x}} = \mathbf{a}$$

- If in doubt: write out with indices
- Chain rule: if y = f(u) and u = g(x), i.e. y = f(g(x)), then

$$\frac{\partial y}{\partial x} = \frac{\partial y}{\partial u}\frac{\partial u}{\partial x}$$

Chain rule

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

$$\frac{\partial y}{\partial x} = \frac{\partial y}{\partial u}\frac{\partial u}{\partial x}$$

- Simple example $\frac{\partial y}{\partial x} = \frac{\partial 5(x^3+7)^4}{\partial x}$

$$y = f(u) = 5u^4$$
$$u = g(x) = x^3 + 7$$
$$\frac{dy}{du} = 20u^3$$
$$\frac{du}{dx} = 3x^2$$
$$\frac{dy}{dx} = 20(x^3+7)^3 \times 3x^2$$

Exercise

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

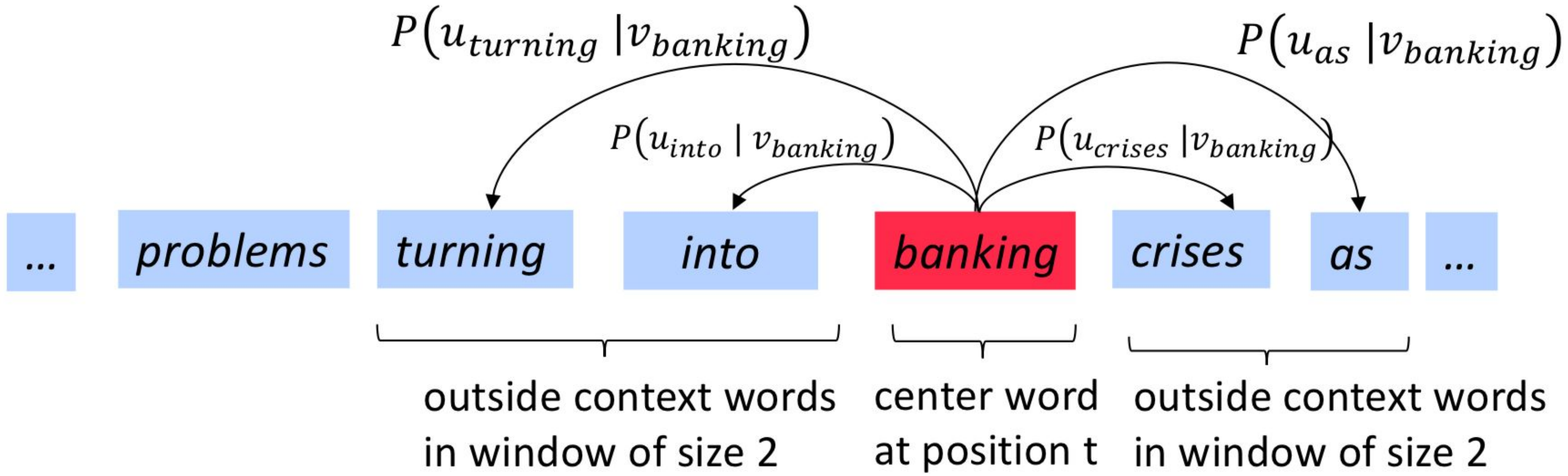
- Lets derive gradient for center word
- For one example in window and one example outside window:

$$\log P(o|c) = \log \frac{\exp(u_o^\top v_c)}{\sum_{w=1}^{V}\exp(u_w^\top v_c)}$$

- You then also need the gradient for context words (it's similar). That's all of the parameter θ here.

Calculating all gradients!

- We went through gradient 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:



Word2vec: more variations

- Why two vectors? -> easier optimization, average both at the end
- Two model variants:
  - Skip-grams(SG): predict context (“outside”) words (position independent) given center word
  - Continuous Bag of Words(CBOW):predict center word from context words
  - This lecture so far: skip-gram model

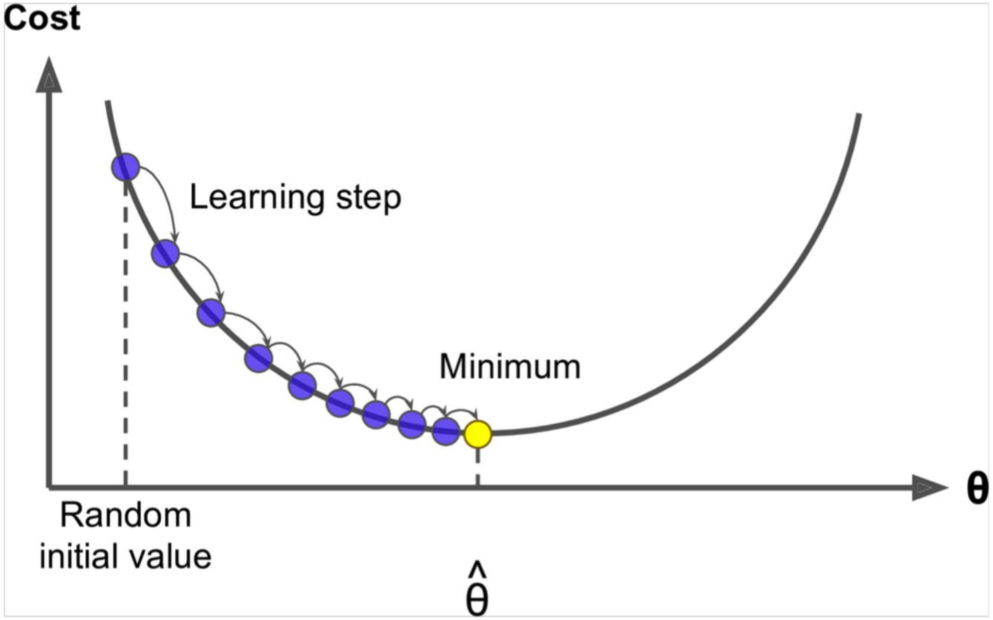
Additional efficiency in training:

Negative sample

so far: focus on naive softmax (simpler training method)

Optimization: Gradient descent

- We have a cost function J(θ) we want to minimize
- Gradient descent is an algorithm to minimize J(θ)
- Idea: for current value of θ, calculate gradient of J(θ), then take small step in direction of negative gradient. Repeat.



Note: our objectives may not be convex like this.

Gradient descent

- Update equation (in matrix notation)
- Update equation (for single parameter):

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

- Algorithm:

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

Stochastic Gradient Descent

- Problem: $J(\theta)$ is a function of all windows in the corpus (potentially billions!)
  - so $\nabla_\theta J(\theta)$ is very expensive to compute
- You would wait a long time before making a single update
- Very bad idea for pretty much all neural nets!
- Solution: Stochastic gradient descent (SGD)
  - Repeatedly sample windows and update after each one
- Algorithm:

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

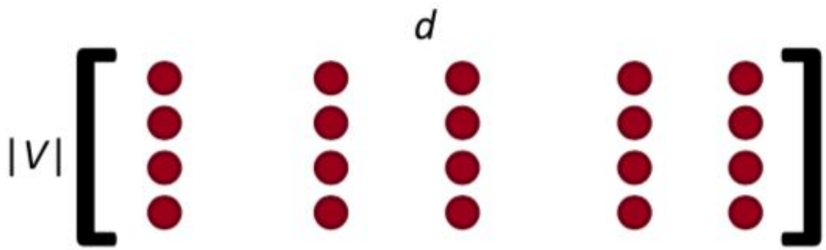
Stochastic gradients with word vectors

- Iteratively take gradients at each such window for SGD
- But in each window, we only have at most 2m+1 words, so $\nabla_\theta J(\theta)$ is 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 might only update the word vectors that actually appear!
- Solution: either you need sparse matrix update operations to only update certain rows of full embedding matrices U and V, or you need to keep around a hash for word vectors



- If you have millions of word vectors and do distributed computing, it is important to not have to send gigantic updates around!

Skip-gram model with negative sampling

- The normalization factor is too computationally expensive.

$$P(o|c) = \frac{\exp(u_o^\top v_c)}{\sum_{w \in V} \exp(u_w^\top v_c)}$$

- Hence, in standard word2vec you implement the skip-gram model with negative sampling

- Main idea: train binary logistic regressions for a true pair (center word and word in its context window) versus several noise pairs (the center word paired with a random word)

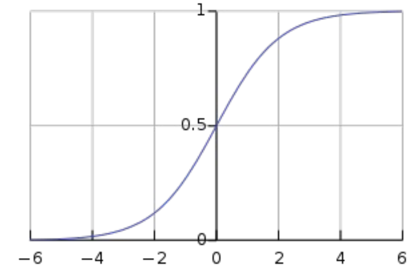
Skip-gram model with negative sampling

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

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

- The sigmoid function:

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$



- We maximize the probability of two words co-occurring in first log

Skip-gram model with negative sampling

- We take k negative samples (using word probabilities)
- Maximize 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 ¾ power
- The power makes less frequent words be sampled more often

Why not capture co-occurrence counts directly?

With a co-occurrence matrix X

- 2 options: windows vs. full document
- Window: similar to word2vec, use window around each word -> captures both syntactic (POS) and semantic information
- Word-document co-occurrence matrix will give general topics (all sports terms will have similar entries) leading to “latent semantic analysis”

Example: window based co-occurrence 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 co-occurrence 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 co-occurrence vectors

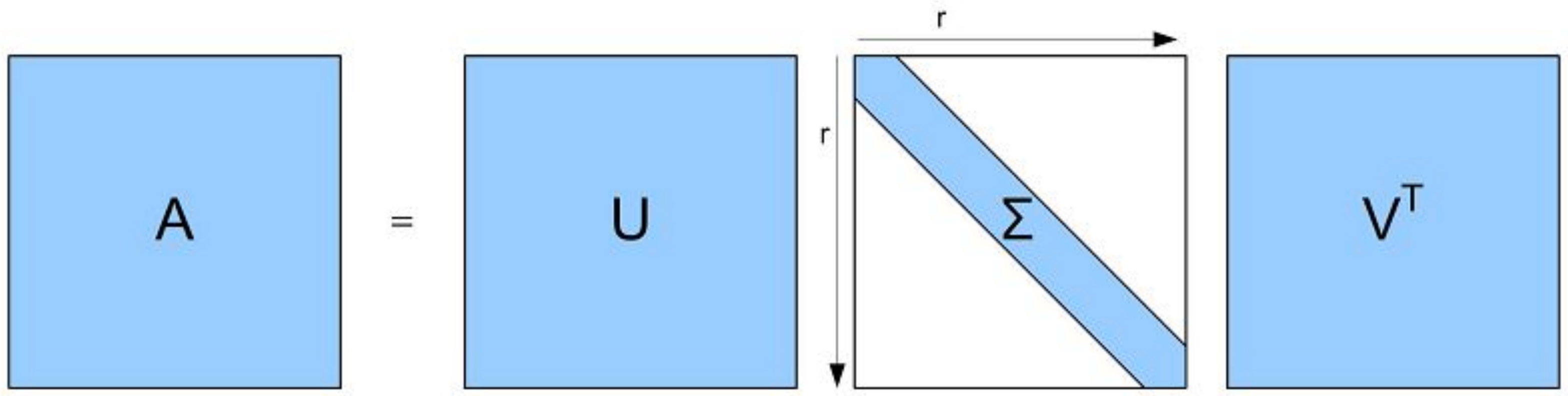
- Increase in size with vocabulary
- Very high dimensional: requires a lot of storage
- Subsequent classification models have sparsity issues
- Models are less robust

Solution: low dimensional vectors

- Idea: store "most" of the important information in a fixed, small number of dimensions: a dense vector
- Usually 25-1000 dimensions, similar to word2vec
- How to reduce the dimensionality?

Method 1: dimensionality reduction on X

- Singular value decomposition of co-occurrence matrix X
- Factorizes X into $U\Sigma V^T$, where U and V are orthonormal



Retain only k singular values, in order to generalize.
A is the best rank k approximation to X , in terms of least squares. Classic linear algebra result. Expensive to compute for large matrices.

Simple SVD word vectors in Python

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

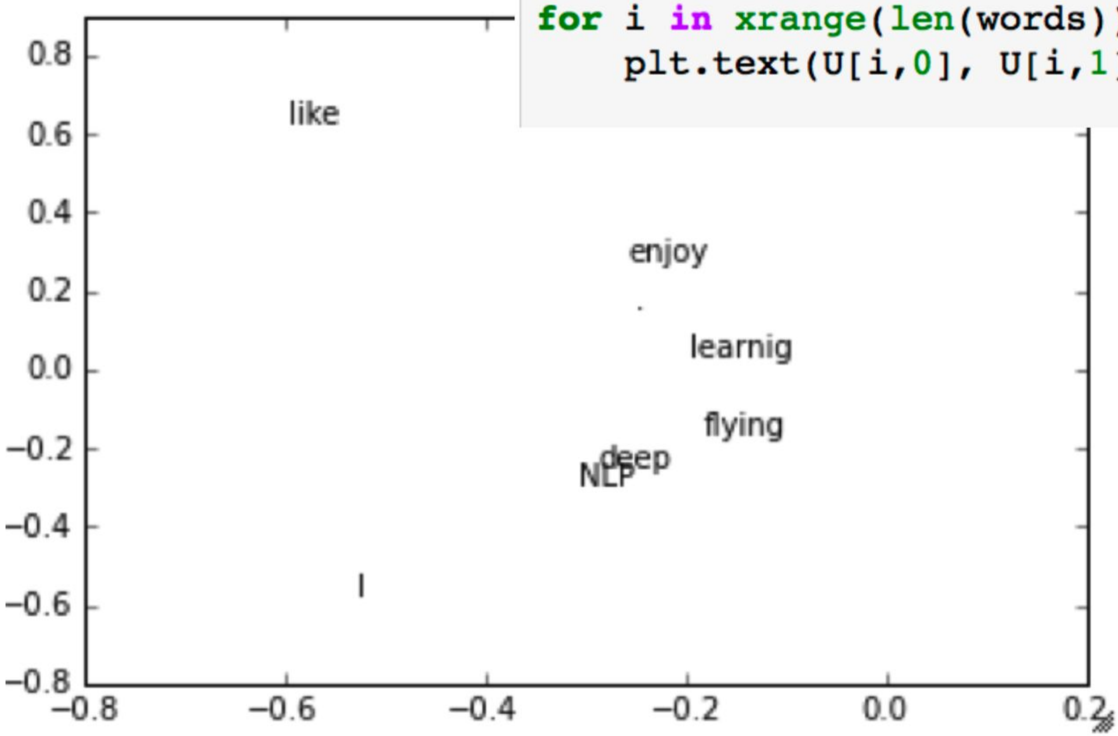
```
import numpy as np
la = np.linalg
words = ["I", "like", "enjoy",
         "deep","learnig","NLP","flying","."]
X = np.array([[0,2,1,0,0,0,0,0],
              [2,0,0,1,0,1,0,0],
              [1,0,0,0,0,0,1,0],
              [0,1,0,0,1,0,0,0],
              [0,0,0,1,0,0,0,1],
              [0,1,0,0,0,0,0,1],
              [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

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

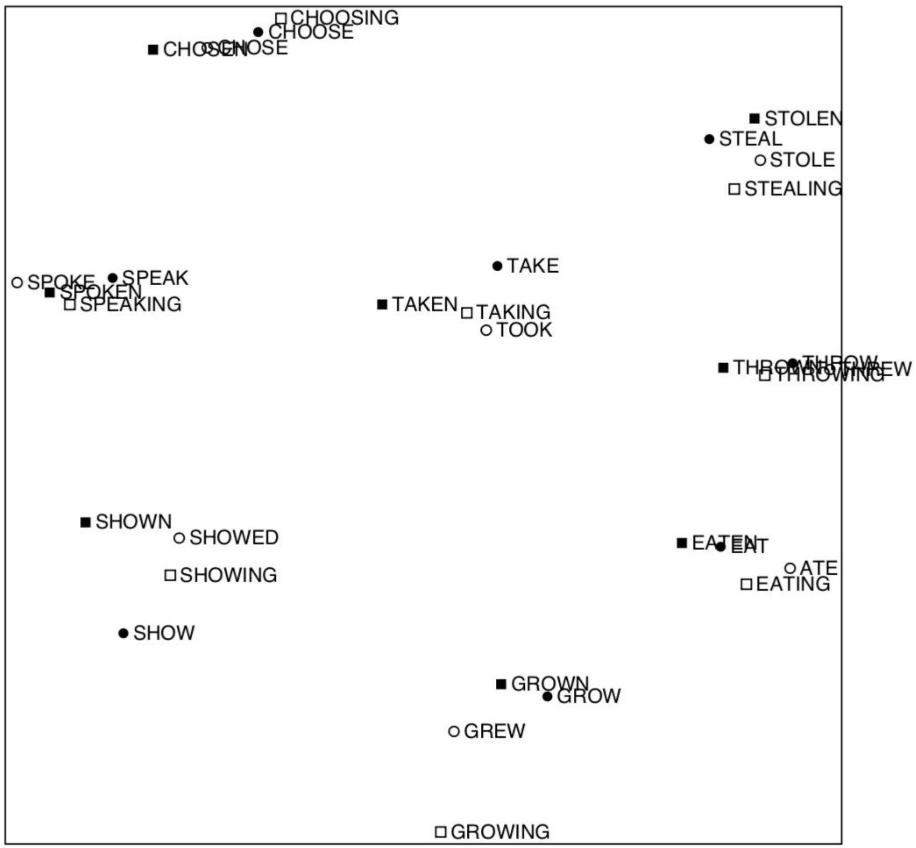
```
for i in xrange(len(words)):
    plt.text(U[i,0], U[i,1], words[i])
```



Hacks to X
(several used in Rohde et al. 2005)

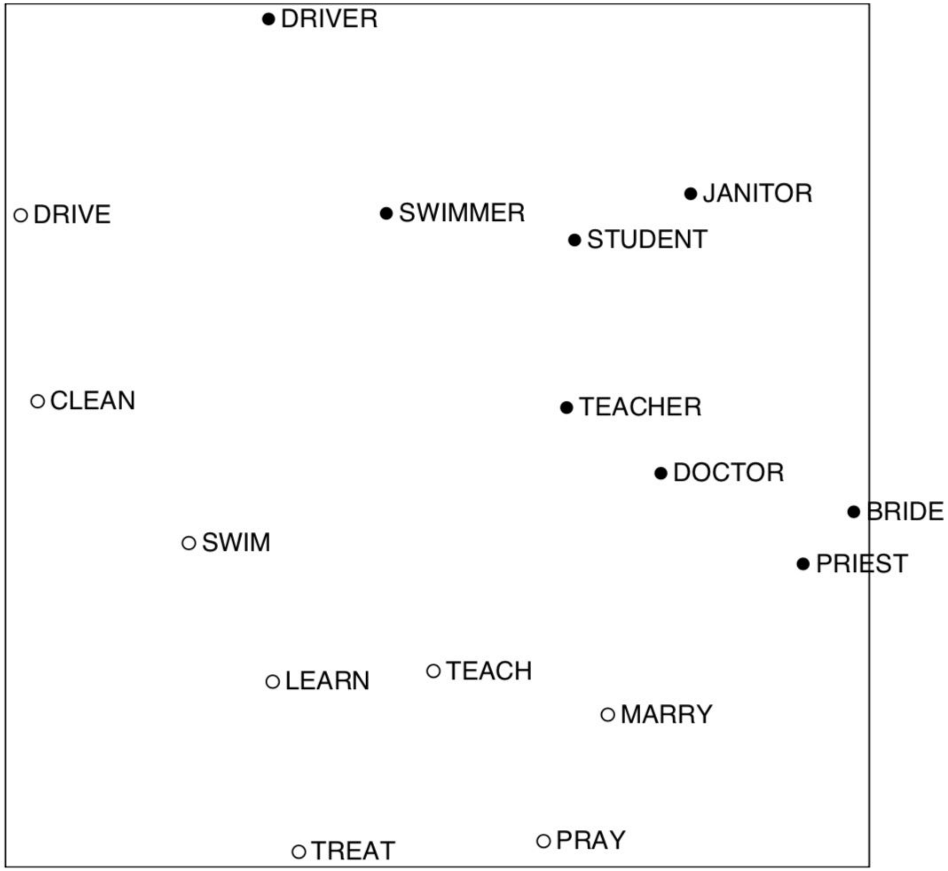
- Scaling the counts in the cells can help a lot
- Problem: function words (the, he, has) are too frequent, thus syntax has too much impact. Some fixes:
  - min(X,t), with t ≈ 100
  - Ignore them all
- Ramped windows that count closer words more (weighted sum of all occurrences of b in proximity to a)
- Use Pearson correlations instead of counts, then set negative values to 0

Interesting syntactic patterns emerge in the vectors



from “*An Improved Model of Semantic Similarity Based on Lexical Co-Occurrence*”. Rohde et al., 2005

Interesting syntactic patterns emerge in the vectors



from “*An Improved Model of Semantic Similarity Based on Lexical Co-Occurrence*”. Rohde et al., 2005

Count based vs. direct prediction

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

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

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

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

Encoding meaning in vector differences
[Pennington, Socher, and Manning, EMNLP 2014]

- Crucial insight: Ratios of co-occurrence probabilities can encode meaning components

	x = solid	x = gas	x = water	x = random
$P(x\|\text{ice})$	large	small	large	small
$P(x\|\text{steam})$	small	large	large	small
$\frac{P(x\|\text{ice})}{P(x\|\text{steam})}$	large	small	~1	~1

Encoding meaning in vector differences
[Pennington, Socher, and Manning, EMNLP 2014]

- Crucial insight: Ratios of co-occurrence probabilities can encode meaning components

	x = solid	x = gas	x = water	x = fashion
$P(x\|\text{ice})$	1.9×10^{-4}	6.6×10^{-5}	3.0×10^{-3}	1.7×10^{-5}
$P(x\|\text{steam})$	2.2×10^{-5}	7.8×10^{-4}	2.2×10^{-3}	1.8×10^{-5}
$\frac{P(x\|\text{ice})}{P(x\|\text{steam})}$	8.9	8.5×10^{-2}	1.36	0.96

Encoding meaning in vector differences
[Pennington, Socher, and Manning, EMNLP 2014]

- Q: How can we capture ratios of co-occurrence probabilities as linear meaning components in a word vector space?
- A: log-bilinear model with vector differences

$$w_i \cdot w_j = \log P(i|j)$$

$$w_x \cdot (w_a - w_b) = \log \frac{P(x|a)}{P(x|b)}$$

Combining the best of both worlds: Glove

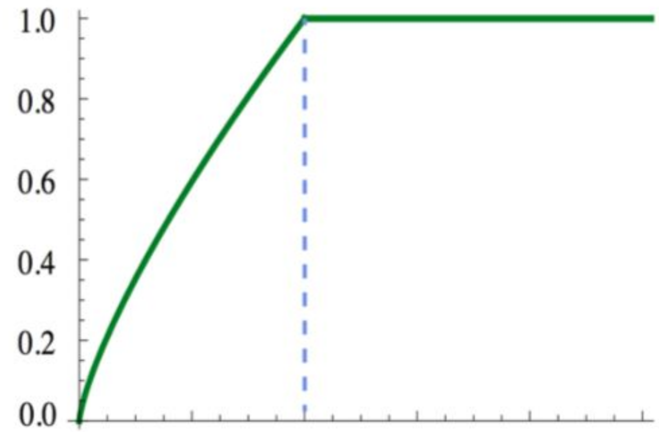
[Pennington et al., EMNLP 2014]

$$w_i \cdot w_j = \log P(i|j)$$

$$J = \sum_{i,j=1}^{V} f(X_{ij})(w_i^\top \hat{w}_j + b_i + \hat{b}_j - \log X_{ij})^2$$

- Fast training
- scalable to huge corpora
- good performance even with small corpus and small vectors

$f \sim$



Glove results
[Pennington et al., EMNLP 2014]

Nearest words to frog:

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

litoria

leptodactylidae

rana

eleutherodactylus

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 sub system is the problem or its interaction or other subsystems
  - If replacing exactly one subsystem with another improves accuracy Winning!

Intrinsic word vector evaluation

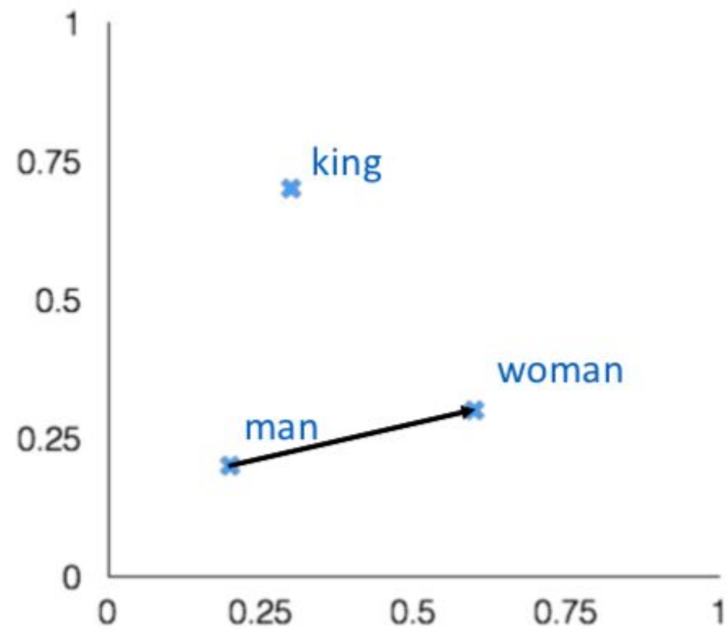
- Word vector analogies

a:b :: c:?

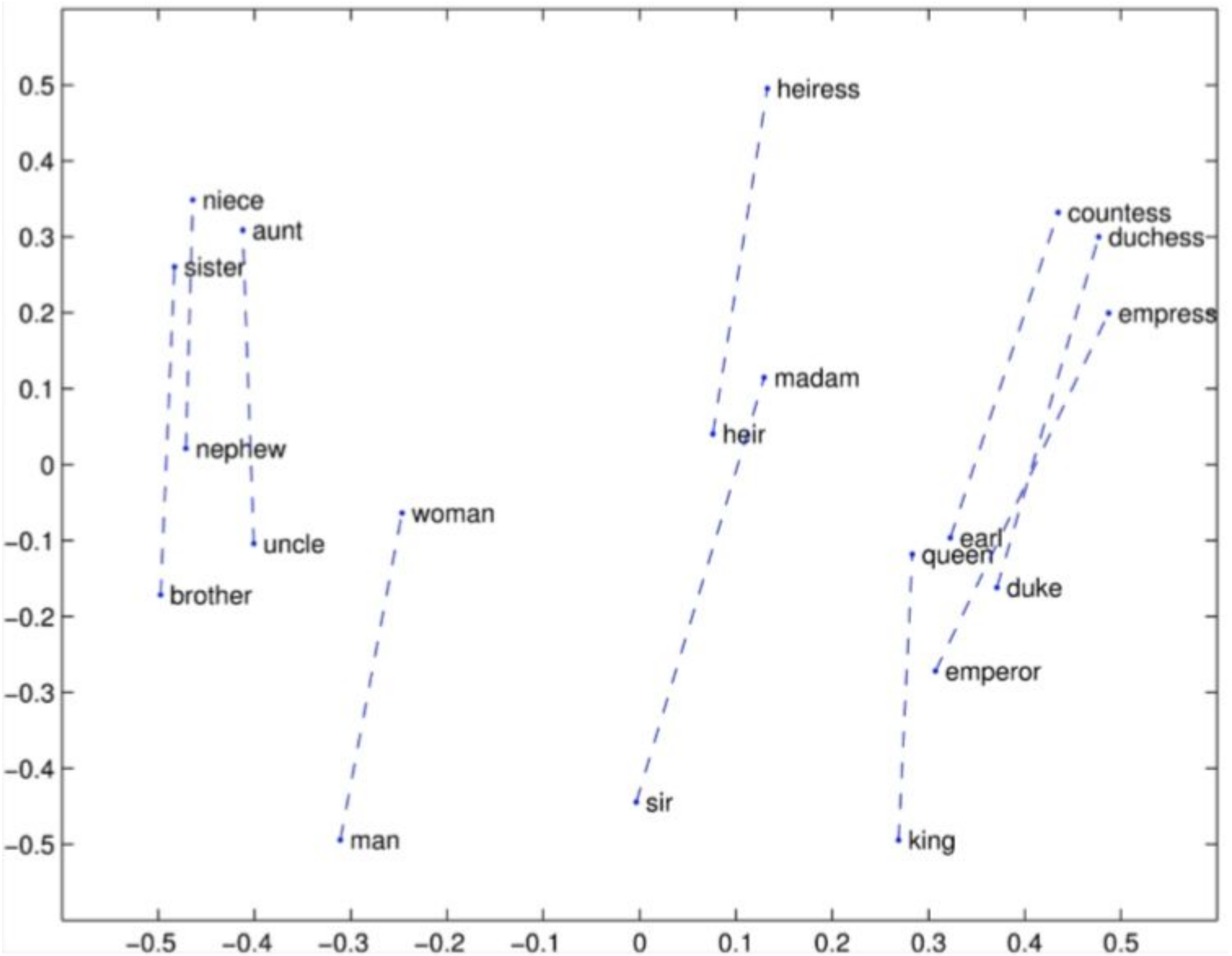
man:woman :: king:?

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

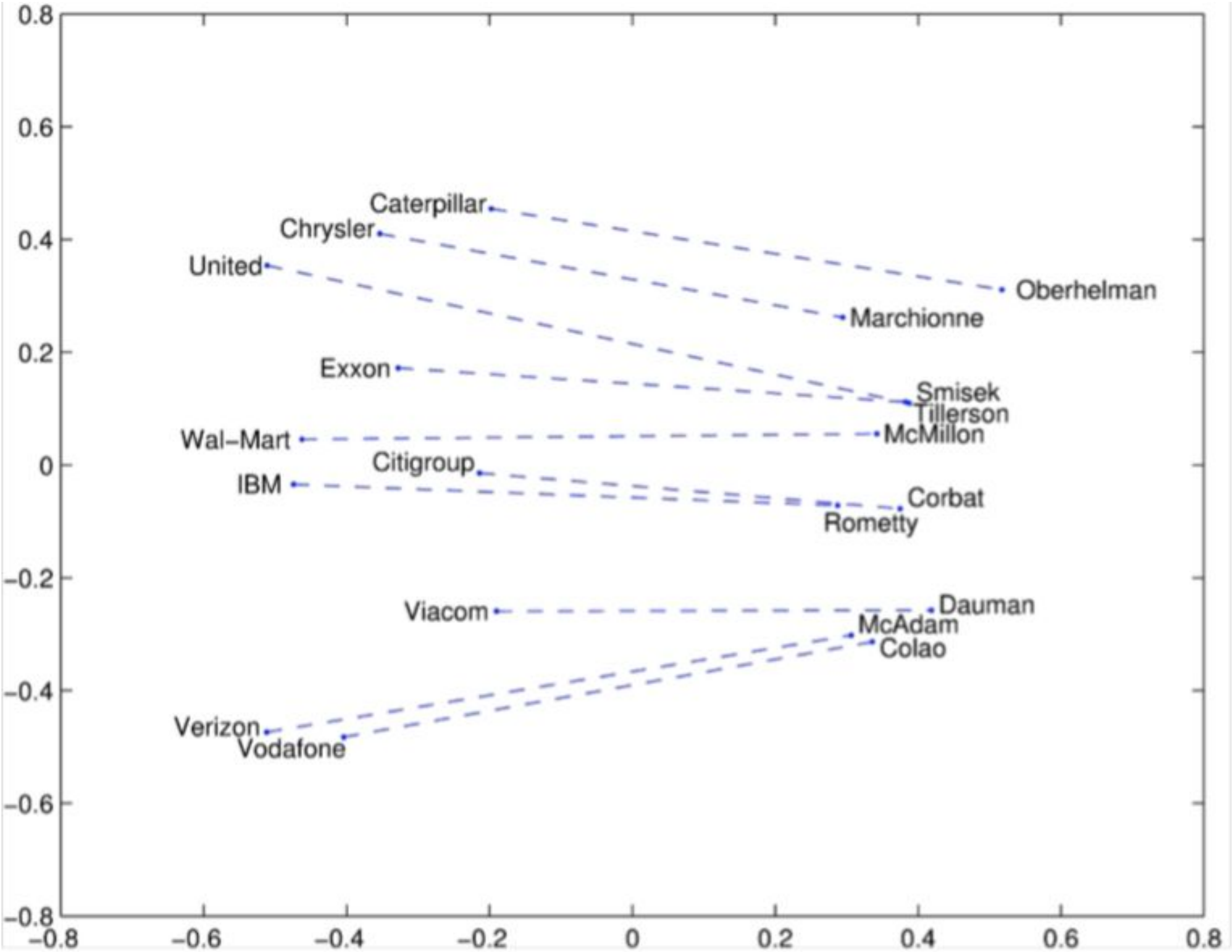
- Evaluate word vectors by how well their cosine distance after addition captures intuitive semantic and syntactic analogy questions
- Discarding the input words from the search!
- Problem: What if the information is there but not linear?



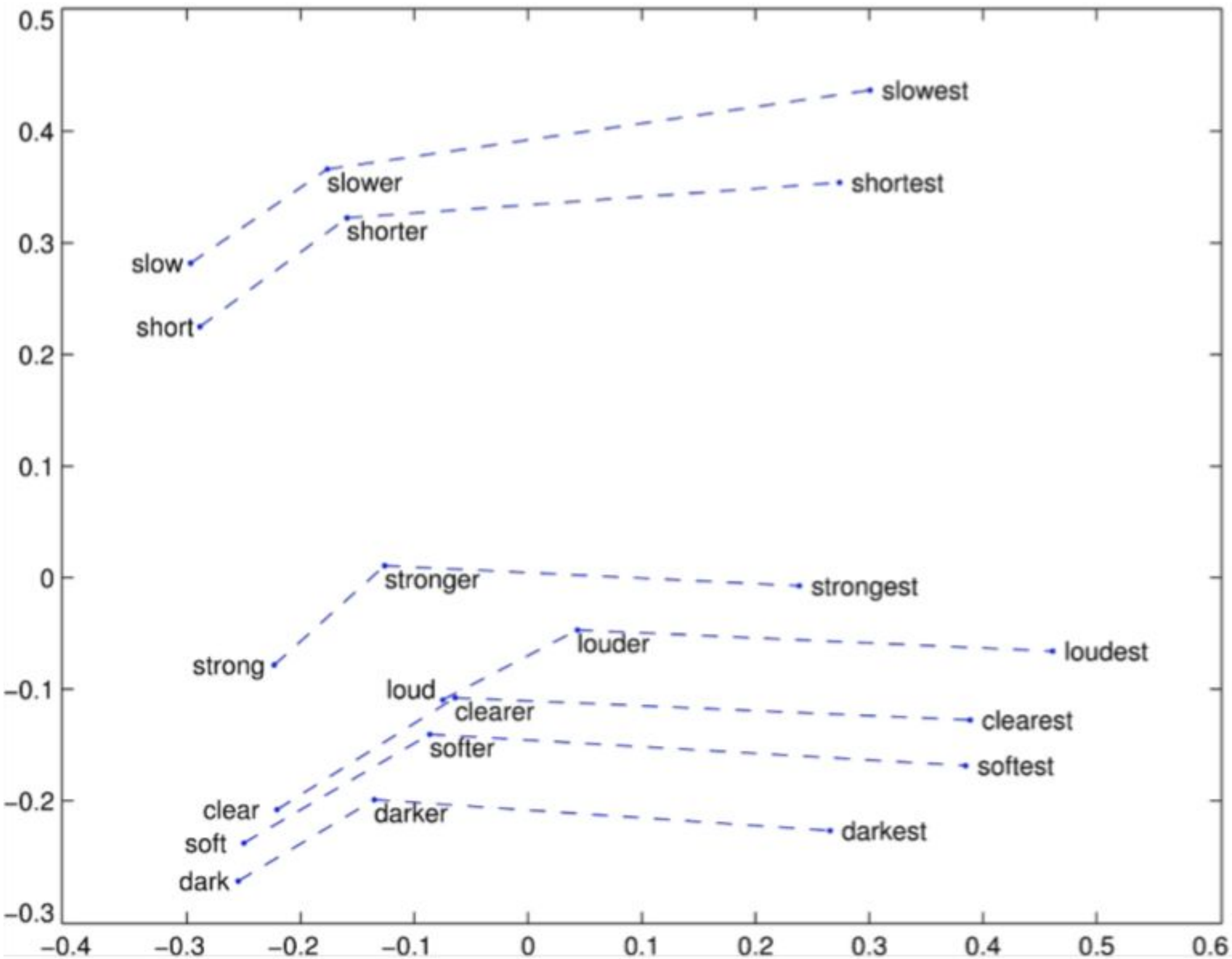
Glove visualizations



Glove visualizations: Company - CEO



Glove visualizations: Superlatives



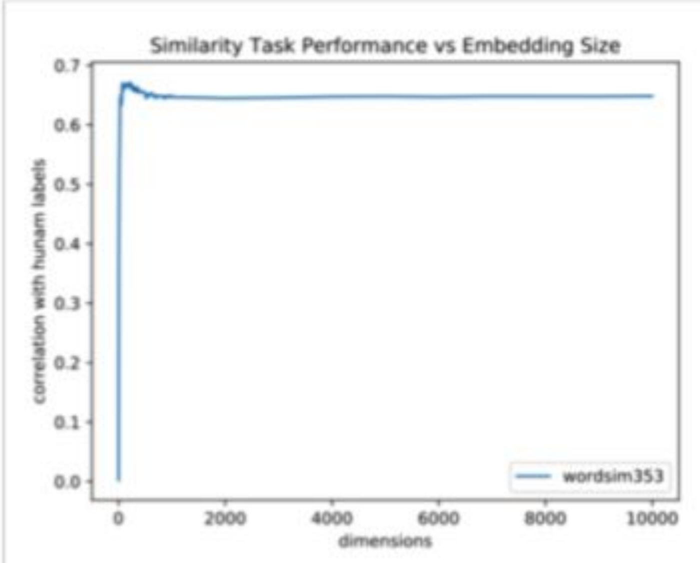
Analogy evaluation and hyperparameters

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	67.5	54.3	60.3
SG	300	1B	61	61	61
CBOW	300	1.6B	16.1	52.6	36.1
vLBL	300	1.5B	54.2	64.8	60.0
ivLBL	300	1.5B	65.2	63.0	64.0
GloVe	300	1.6B	80.8	61.5	70.3
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	67.4	65.7
SG†	300	6B	73.0	66.0	69.1
GloVe	300	6B	77.4	67.0	71.7
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	**81.9**	**69.3**	**75.0**

On the dimensionality of word embedding

[Zi Yin and Yuanyuan Shen, NeurIPS 2018]

- Using matrix perturbation theory, reveal a fundamental bias- variance trade-off in dimensionality selection for word embeddings

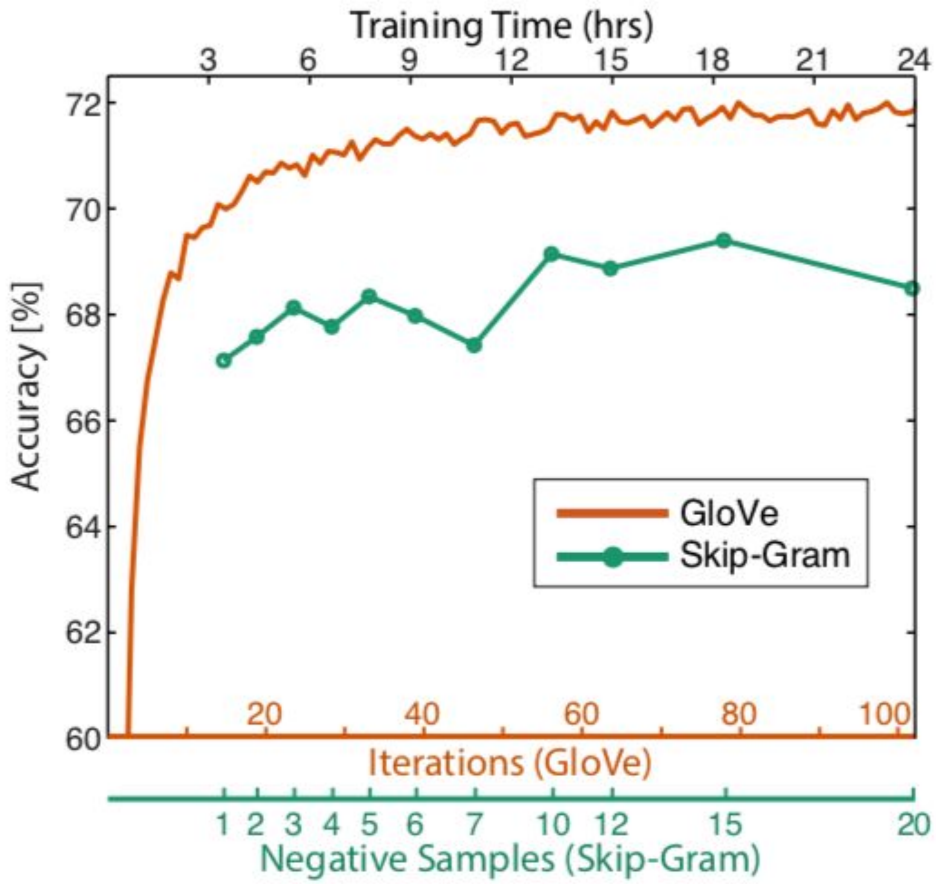


(b) WordSim353 Test

https://papers.nips.cc/paper/7368-on-the-dimensionality-of-word-embedding.pdf

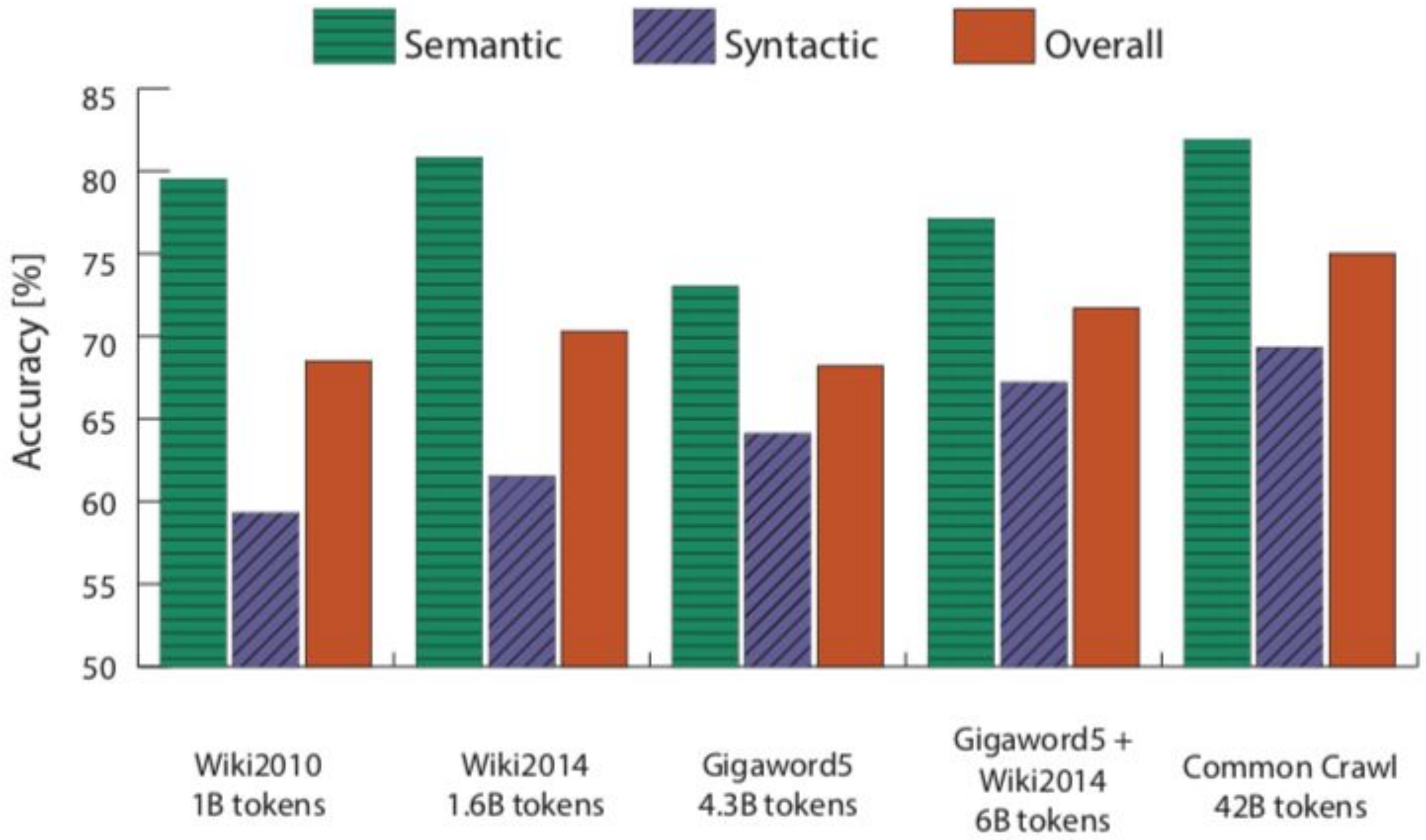
Analogy evaluation and hyperparameters

- More training time helps



Analogy evaluation and hyperparameters

- More data helps, wikipedia is better than news text



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
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

Closet words to “Sweden” (cosine)

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

Word senses and word sense ambiguity

- Most words have lots of meanings!
  - Especially common words
  - Especially words that have existed for a long time
- Example: pike
- Does one vector capture all these meanings or do we have a mess?

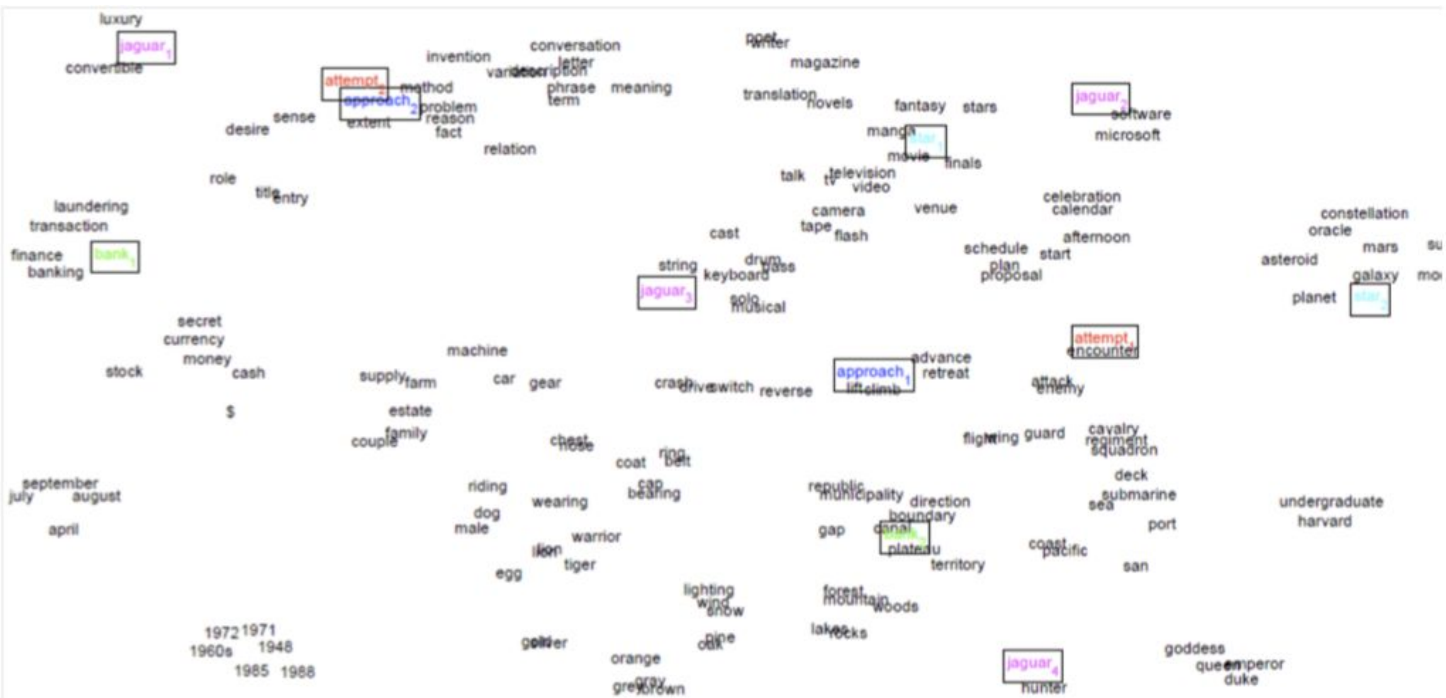
Word senses and word sense ambiguity

pike

- A sharp point or staff
- A type of elongated fish
- A railroad line or system
- A type of road
- The future (coming down the pike)
- A type of body position (as in diving)
- To kill or pierce with a pike
- To make one’s way (pike along)
- In Australian English, pike means to pull out from doing something: I reckon he could have climbed that cliff, but he piked!

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 bank1, bank2, etc



Linear algebraic structure of word senses with applications to polysemy
[Arora et al. TACL 2018]

- Different senses of a word reside in a linear superposition (weighted sum) in standard word embeddings like word2vec
- $v_{pike} = a_1 v_{pike1} + a_2 v_{pike2} + a_3 v_{pike3}$
- Where

$$a_1 = \frac{f_1}{f_1 + f_2 + f_3}$$

- Surprising result:
  - Because of ideas from sparse coding you can actually separate out the senses (providing they are relatively common)

Linear algebraic structure of word senses with applications to polysemy

[Arora et al. TACL 2018]

tie				
trousers	season	scoreline	wires	operatic
blouse	teams	goalless	cables	soprano
waistcoat	winning	equaliser	wiring	mezzo
skirt	league	clinching	electrical	contralto
sleeved	finished	scoreless	wire	baritone
pants	championship	replay	cable	coloratura

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

Limitations of distributional methods

- Definition of similarity
  - “words are similar if used in similar contexts”
- Black sheeps
  - people are less likely to mention known information than they are mention novel one
- Antonyms
  - good vs. bad, buy vs. sell
- Corpus biases
  - racial and gender stereotypes
- Lack of context
  - the distributional approaches aggregate the contexts in which a term occurs in a large corpus. The result is a “context independent” word representation

Using word embeddings

- Word similarity: cosine
- Word clustering
- Finding similar words
- Odd-one out
    - find the one that does not belong to a given list of words: computing the similarity between each word to the average word vector
- Short-document similarity
- Word analogies: $w_{king} - w_{man} + w_{woman} \approx w_{queen}$

STEVENS
INSTITUTE of TECHNOLOGY
THE INNOVATION UNIVERSITY®

stevens.edu

Thank You