Deep Neural Networks for Natural Language Processing (AI6127)

JUNG-JAE KIM

WORD VECTORS

Contents

- Word vectors
- Word2vec: Learning word vectors
  - Stochastic gradient descent (SGD)
  - Negative sampling
- GloVe
- Word vector evaluation

Representing words as discrete symbols

- Recall: Binary term-document incidence matrix
- Words are 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)

Means one 1, the rest 0s

	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
Antony	1	1	0	0	0	1
Brutus	1	1	0	1	0	0
Caesar	1	1	0	1	1	1
Calpurnia	0	1	0	0	0	0
Cleopatra	1	0	0	0	0	0
mercy	1	0	1	1	1	1
worser	1	0	1	1	1	0

Source: IR

Problem with words as discrete symbols

- If “Seattle motel” belongs to ‘lodging’ category, “Seattle hotel” might as well
- But these two vectors are orthogonal
  - 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]
- There is no natural notion of similarity for one-hot vectors!
- Should represent lexical semantics (e.g. word similarity)

Source: CS224n

Denotational lexical semantics

- 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 (sat)',
  '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
…
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')]
```

https://wordnet.princeton.edu/

Source: CS224n

Problems with resources like 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, 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

Source: CS224n

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***

Source: CS224n

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

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

Source: CS224n

Word meaning as a neural word vector – visualization

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

Source: CS224n

Contents

- Word vectors
- Word2vec: Learning word vectors
  - Stochastic gradient descent (SGD)
  - Negative sampling
- GloVe
- Word vector evaluation

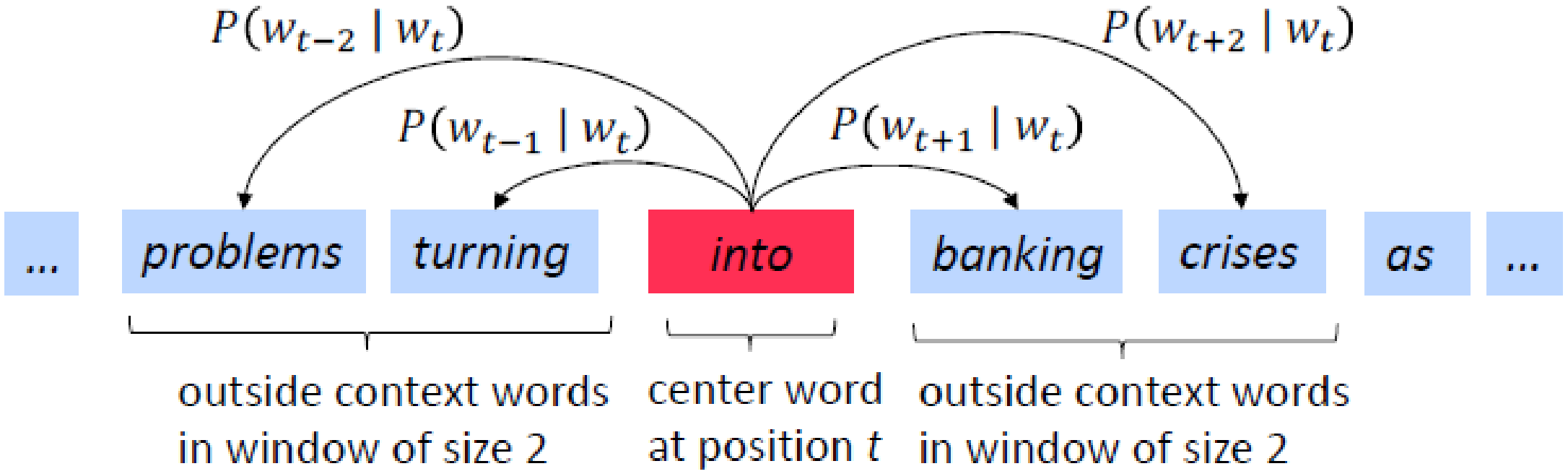
Word2vec: Overview

- Word2vec (Mikolov et al. 2013) is a framework for learning word vectors
- Idea:
  - We have a large corpus of texts
  - 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

Source: CS224n

Word2Vec Overview

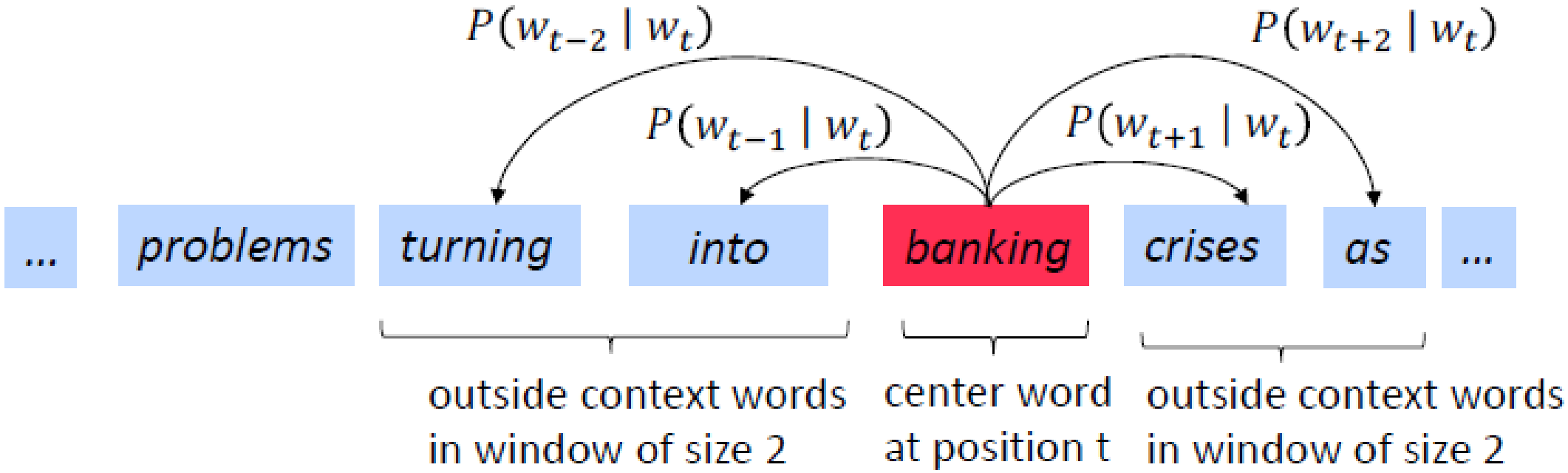
- Example windows and process for computing $p(w_{t+j}|w_t)$



Source: CS224n

Word2Vec Overview

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



Source: CS224n

Word2vec: objective function

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

Likelihood = $$L(\theta) = \prod_{t=1}^{T} \prod_{\substack{-m \le j \le m \\ j \neq 0}} p(w_{t+j} | w_t; \theta)$$

θ is all variables to be optimized

- The objective function $J(\theta)$ is the (average) negative log likelihood:

sometimes called *cost* or *loss* function

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

- Minimizing objective function $\Leftrightarrow$ Maximizing predictive accuracy

Source: CS224n

Word2vec: objective function

- We want to minimize the objective function:

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

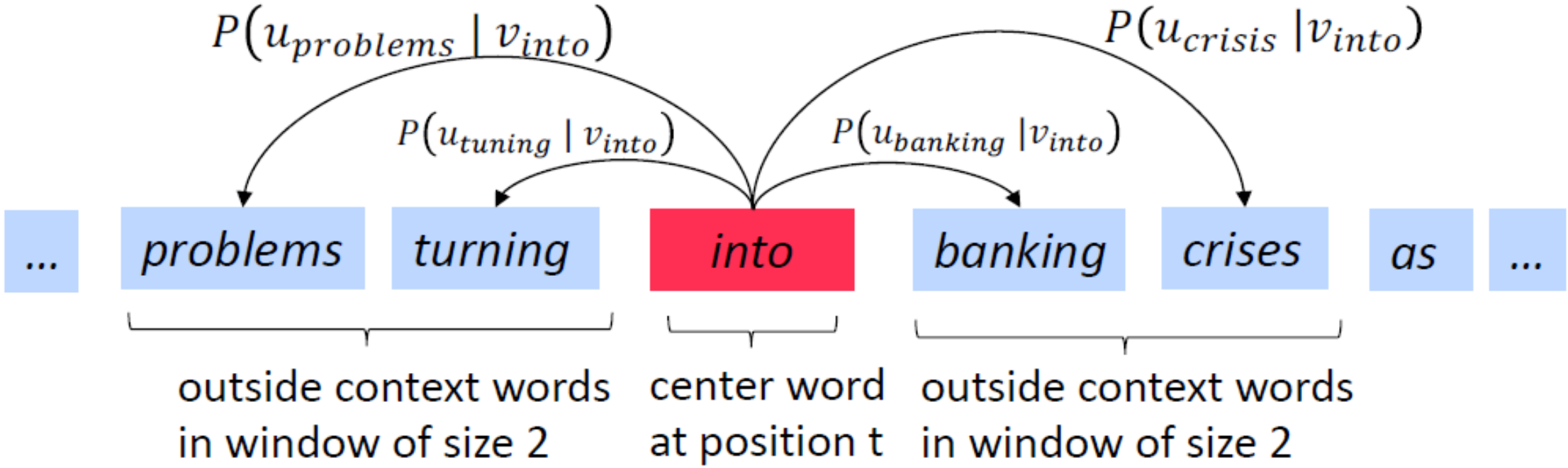
- Question: How to calculate $p\left(w_{t+j}|w_t;\theta\right)$?
- Answer: We will *use two* vectors per word *w*:
  - v_w: when *w* is a center word
  - u_w : when *w* is a context word
- Then for a center word *c* and a context word *o*:

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

Source: CS224n

Word2Vec Overview with vectors

- Example windows and process for computing
- $p(u_{problems}|v_{into})$ short for $p(problems|into; u_{problems}, v_{into}, \theta)$



Source: CS224n

Word2vec: prediction function

(3) Normalize over entire vocabulary to give probability distribution

(2) Exponentiation makes anything positive

(1) Dot product compares similarity of *o* and *c*.

$$u^T v = u \cdot v = \sum_{i=1}^{n} u_i v_i$$

Larger dot product = larger probability

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

- This is an example of the softmax function $\mathbb{R}^n \rightarrow (0,1)^n$

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

- The softmax function maps arbitrary values x_i to a probability distribution p_i
  - "max" because amplifies probability of largest x_i
  - "soft" because still assigns some probability to smaller x_i
  - Frequently used in Deep Learning

Source: CS224n

Word2vec: Training – Compute all vector gradients

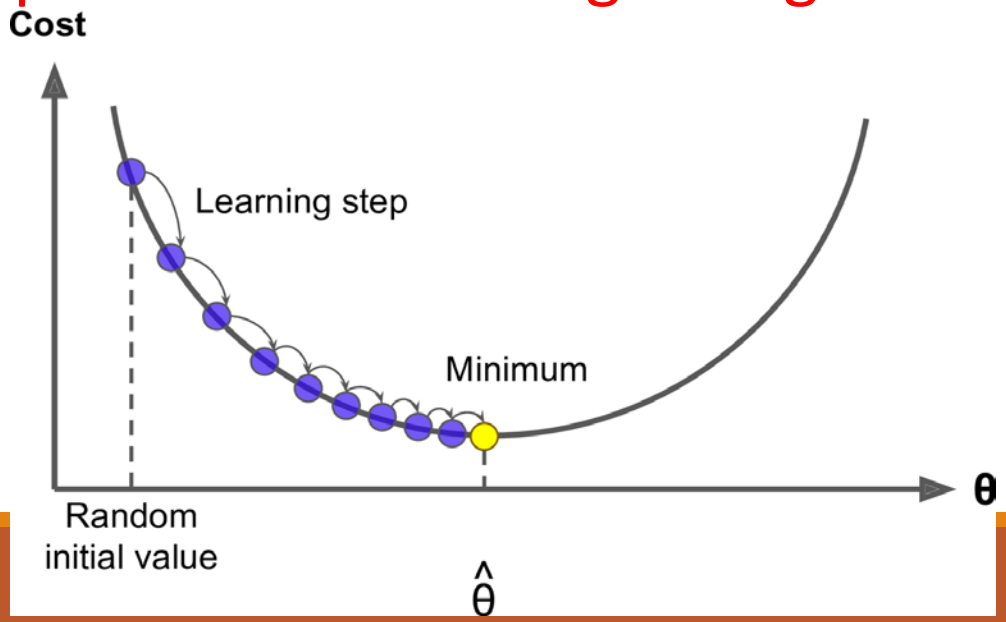
- Recall: θ represents all model parameters
- In our case, *d*-dimensional vectors for |V|-many words
- Remember: every word has two vectors
- We can optimize these parameters by walking down the gradient
  - Watch https://stanford-pilot.hosted.panopto.com/Panopto/Pages/Viewer.aspx?id=b2acdfb7-8038-49fb-941e-ab25012bd9ec for details

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

Source: CS224n

Word2vec: Gradient Descent

- In optimization, we have a cost function $J(\theta)$ we want to minimize
- Gradient Descent is an algorithm to minimize $J(\theta)$
- Idea: For current value of θ, calculate gradient of $J(\theta)$, $\nabla_\theta J(\theta)$ then take small step in direction of negative gradient. Repeat.



Note: Our objectives may not be convex like this :(

Source: CS224n

Gradient Descent

- Update equation (in matrix notation):

alpha: step size or learning rate

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

- Algorithm:

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

Word2vec: Variants

- Model variants
  - Skip-grams (SG): Predict context (“outside”) words (position independent) given center word
    - This lecture introduces skip-gram model
  - Continuous Bag of Words (CBOW): Predict center word from (bag of) context words
- Efficiency in training
  - ‘Naïve’ softmax (so far) vs. Negative sampling (later)

Source: CS224n

Word2vec: Another view

Training: Collecting input-output pairs

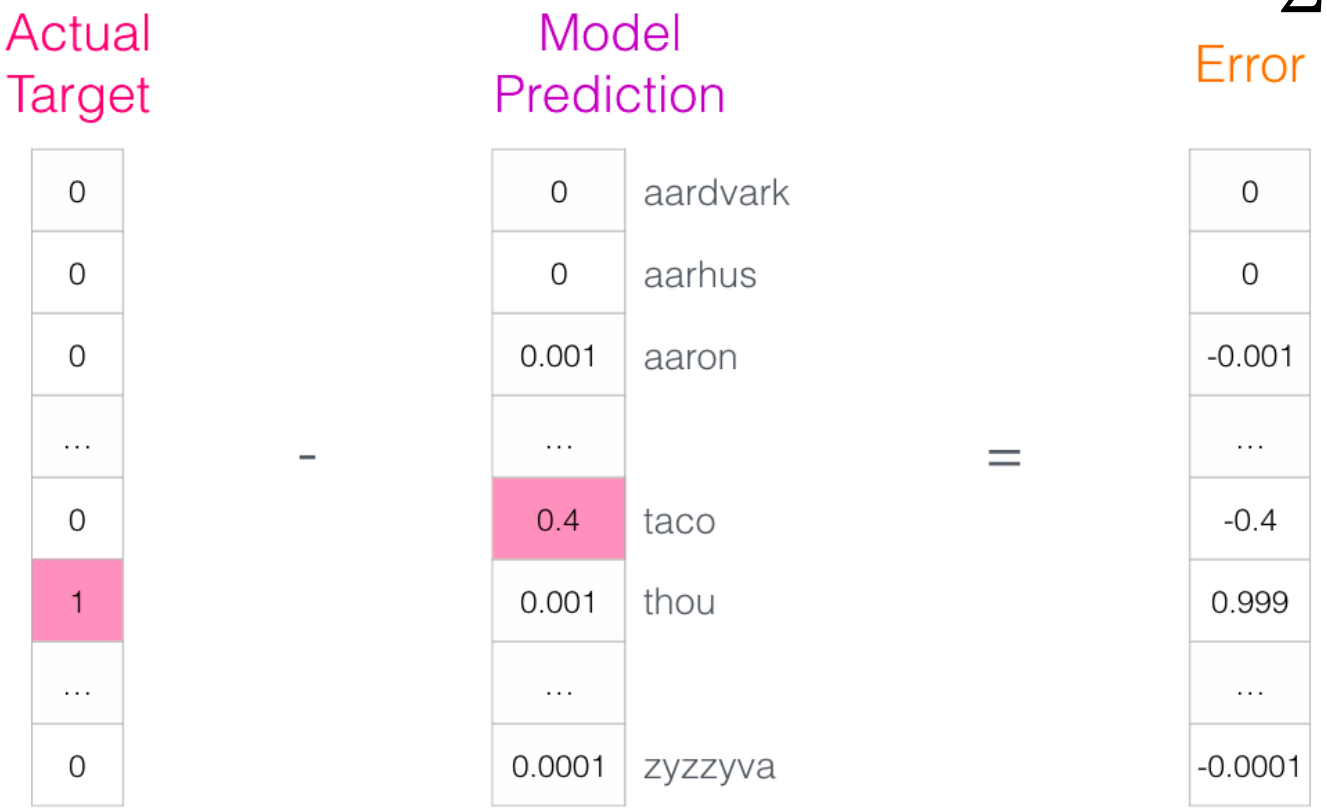
Thou shalt not make a machine in the likeness of a human mind

input word	target word
not	thou
not	shalt
not	make
not	a

Word2vec: Another view

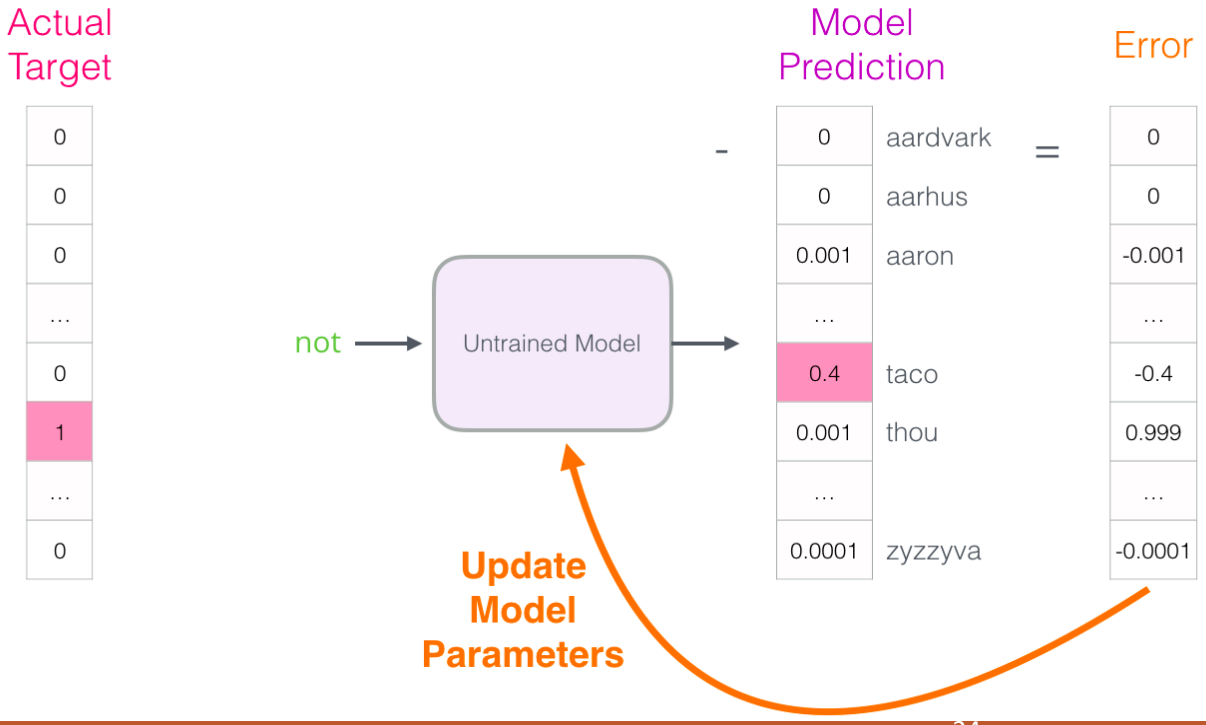
- Get error vector by subtraction

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



Word2vec: Another view

- We proceed to do the same process with the next samples in our dataset, until we cover all the samples in the dataset.
  - That concludes one epoch of training. We do it over again for a number of epochs.
- Then we'd have our trained model and we can extract the embedding matrix from it and use it for any other application.



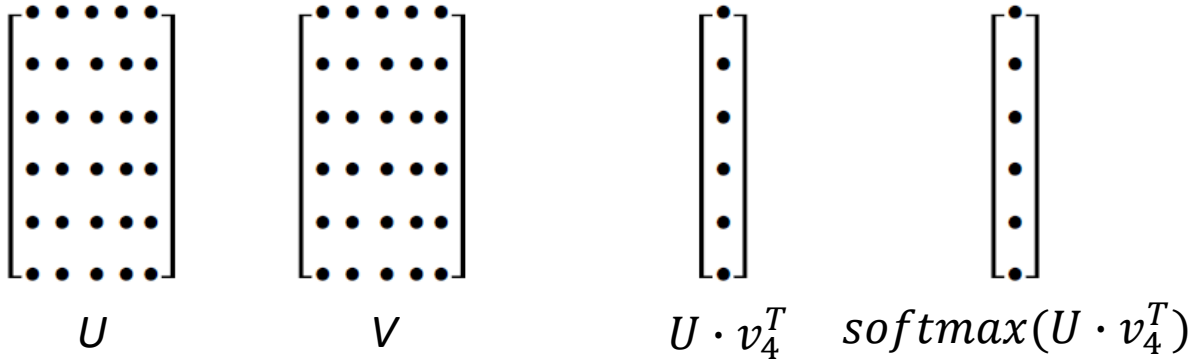
Contents

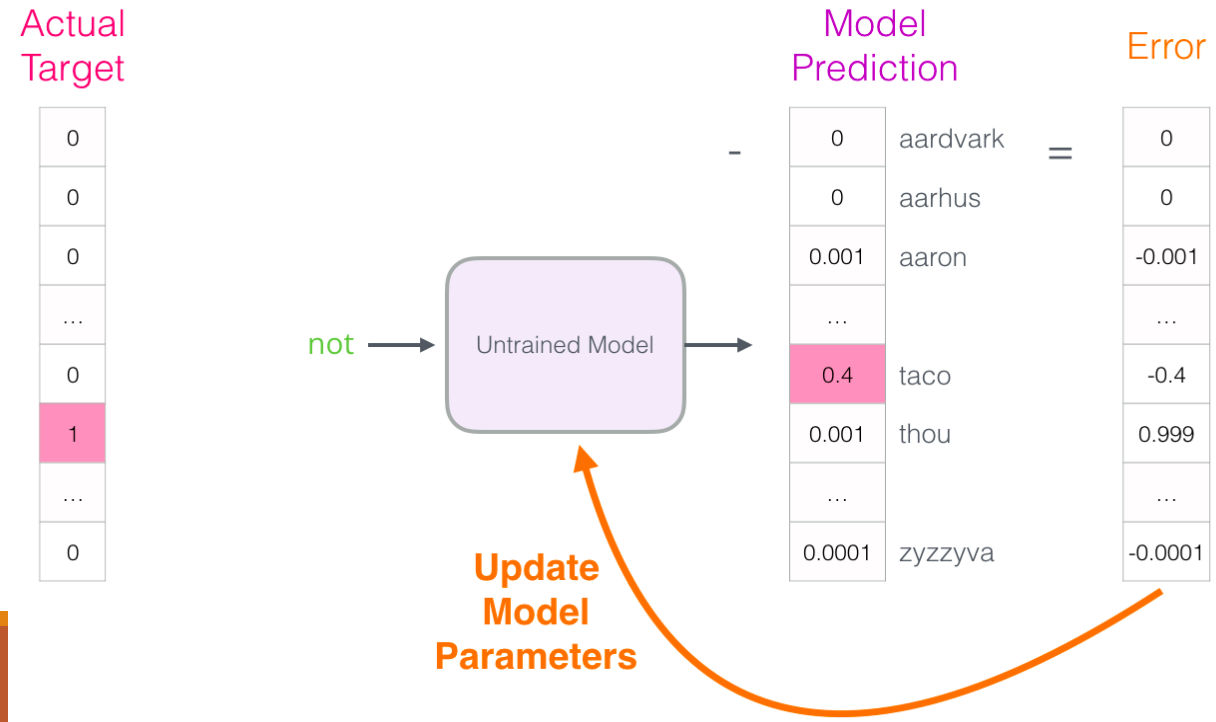
- Word vectors
- Word2vec: Learning word vectors
  - Stochastic gradient descent (SGD)
  - Negative sampling
- GloVe
- Word vector evaluation

Word2vec: Training

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

- One Issue: Model prediction is expensive (because of the denominator).
- We need to do it once for every training sample in our dataset (easily millions of times)





Recall

Gradient Descent

- Update equation (in matrix notation):

alpha: step size or learning rate

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

- Algorithm:

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

Word2vec: Stochastic Gradient Descent

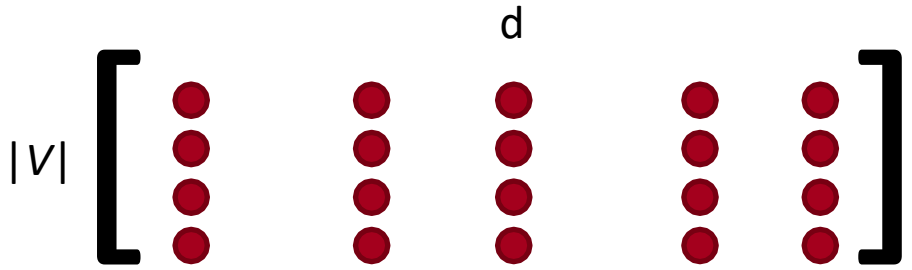
- Problem: $J(\theta)$ is a function of all windows in the corpus (potentially billions!)
  - So $\nabla_\theta J(\theta)$ is very expensive to compute
- You would wait a very long time before making a single update!
- Solution: Stochastic gradient descent (SGD)
  - Repeatedly sample windows, and update after each one
- Algorithm:
  - alpha: step size or learning rate

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


Source: CS224n

Word2vec: Stochastic Gradient Descent

- We might only update the word vectors that appear in the window!
- 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!

Source: CS224n

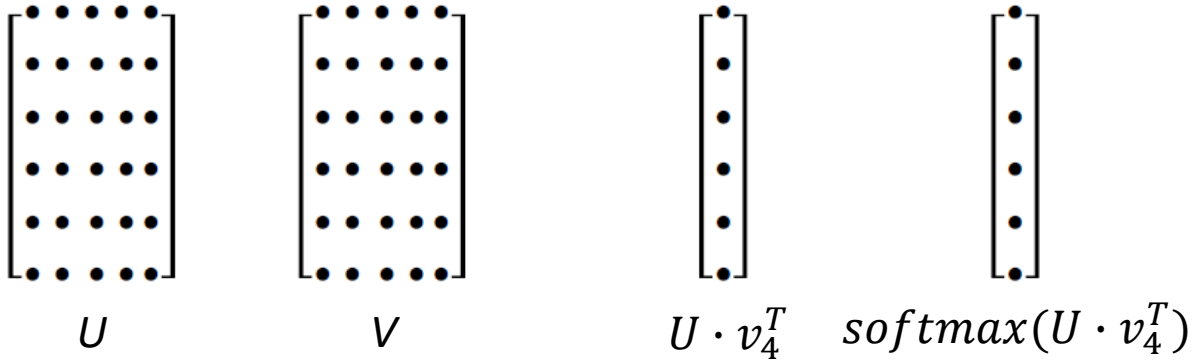
Contents

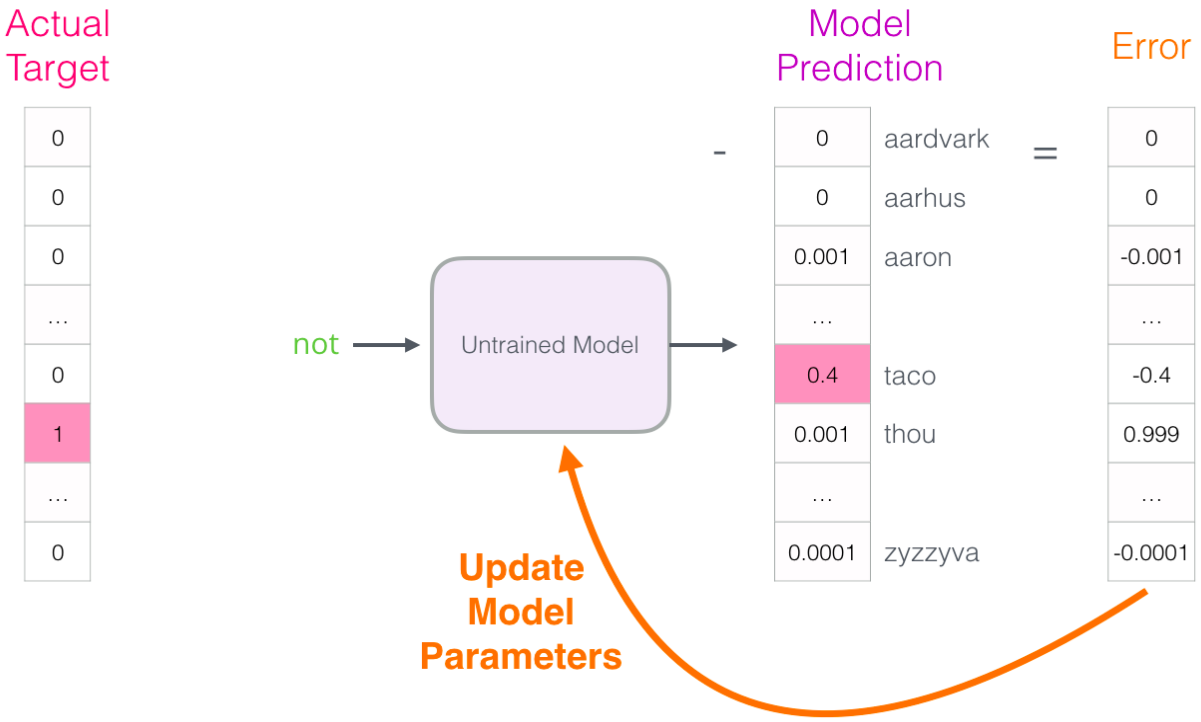
- Word vectors
- Word2vec: Learning word vectors
  - Stochastic gradient descent (SGD)
  - Negative sampling
- GloVe
- Word vector evaluation

Word2vec: Training

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

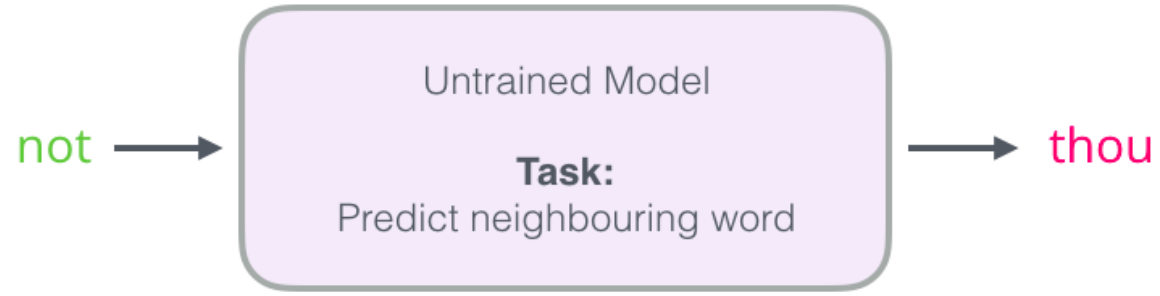
- Recall: Issue - The last step is expensive (because of the denominator).
  - We need to do it once for every training sample in our dataset (easily millions of times)
- Another solution: Negative sampling
  - Get rid of the denominator (next slide)





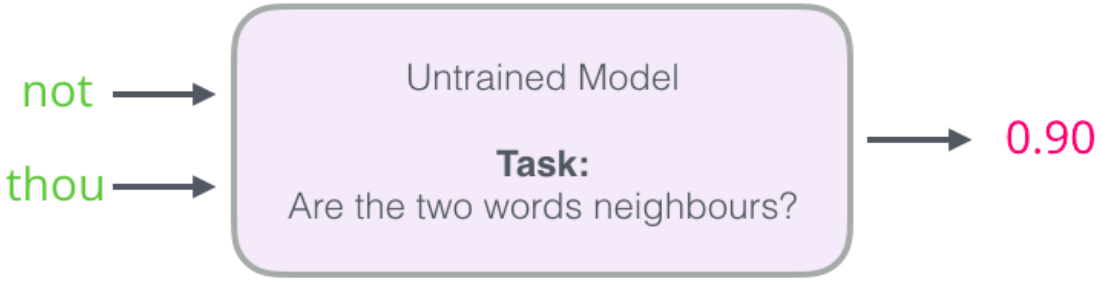
Word2vec: Negative sampling

Change task from:



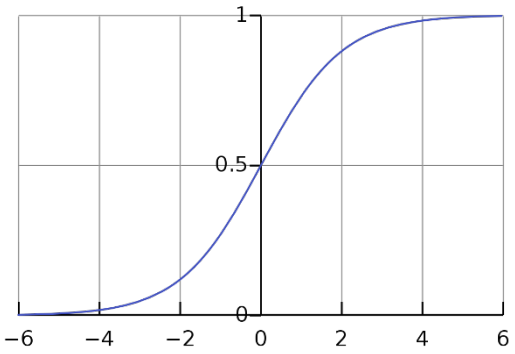
$$p(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{w \in V} \exp(u_w^T v_c)}$$

To:



$$\log p(t = 1|c, o) = \text{sig}(u_o^T v_c)$$

- This changes the model to a logistic regression model, thus much simpler and much faster to calculate



Logistic/ Sigmoid function

Word2vec: Negative sampling

- But there's one loophole! All our examples are positive
- Introduce negative samples – *k* samples of words that are not neighbors
  - Model should return 0 for those samples
  - Randomly sample words from vocabulary
- Negative sampling is a version of noise-contrastive estimation.
  - Contrast the actual signal (positive examples of neighboring words) with noise (randomly selected words that are not neighbors)

input word	output word	target
not	thou	**1**
not		**0**
not		**0**
not	shalt	**1**
not	make	**1**

Negative examples

Word2vec: Skipgram with negative sampling

- Maximize prob. that real outside word appears (co-occur) around center word
- Minimize prob. that random word appears around center word

Skipgram

shalt	not	make	a	machine

input	output
make	shalt
make	not
make	a
make	machine

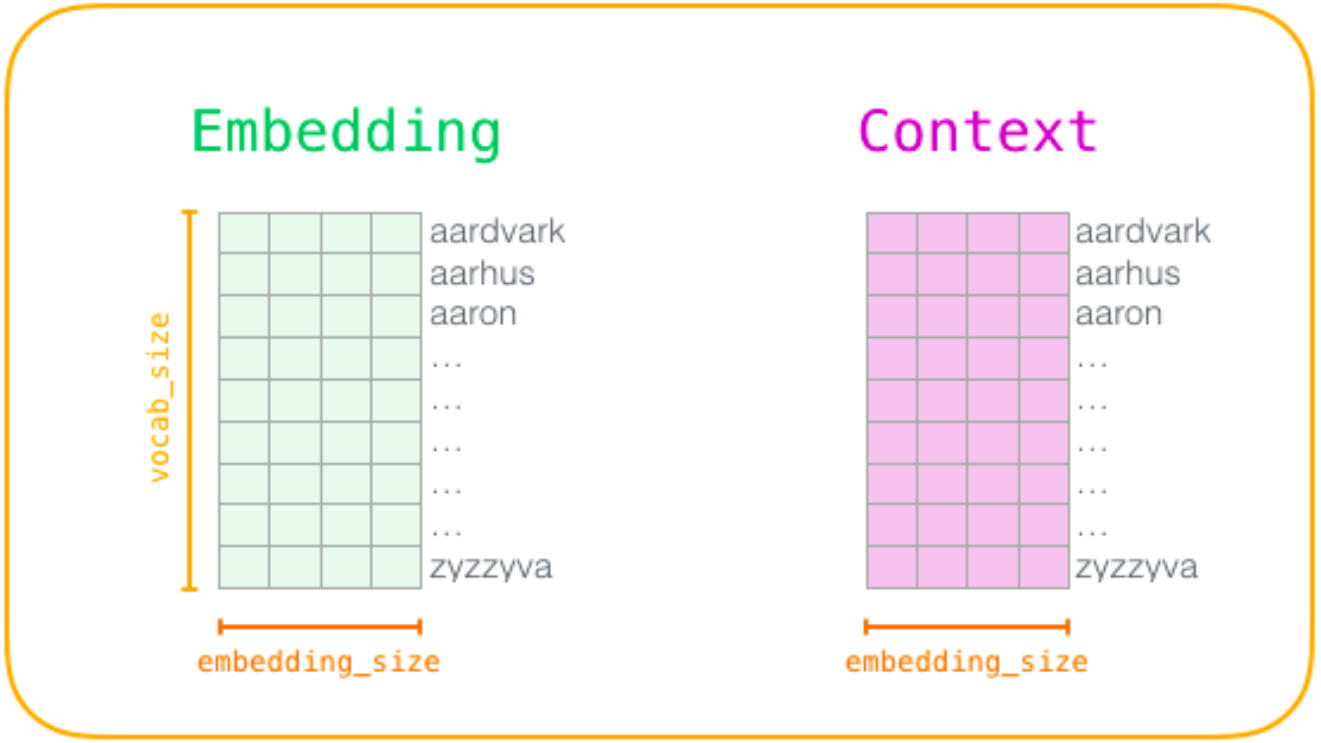
Negative Sampling

input word	output word	target
make	shalt	1
make	aaron	0
make	taco	0

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

Word2vec: Skipgram with negative sampling - Training

- Create two matrices – an Embedding matrix and a Context matrix.
- Initialize these matrices with random values



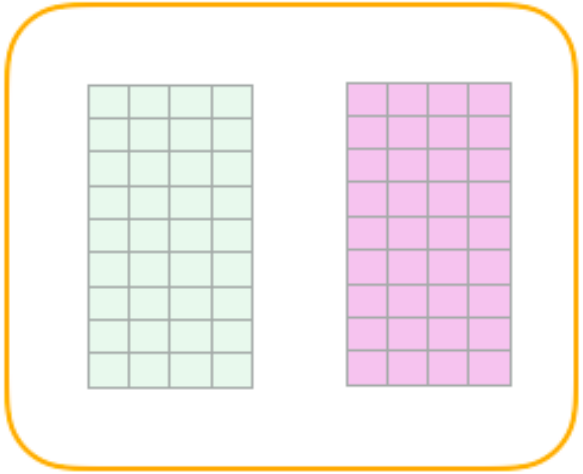
Word2vec: Skipgram with negative sampling - Training

- In each training step, take one positive example and its associated negative examples.

dataset

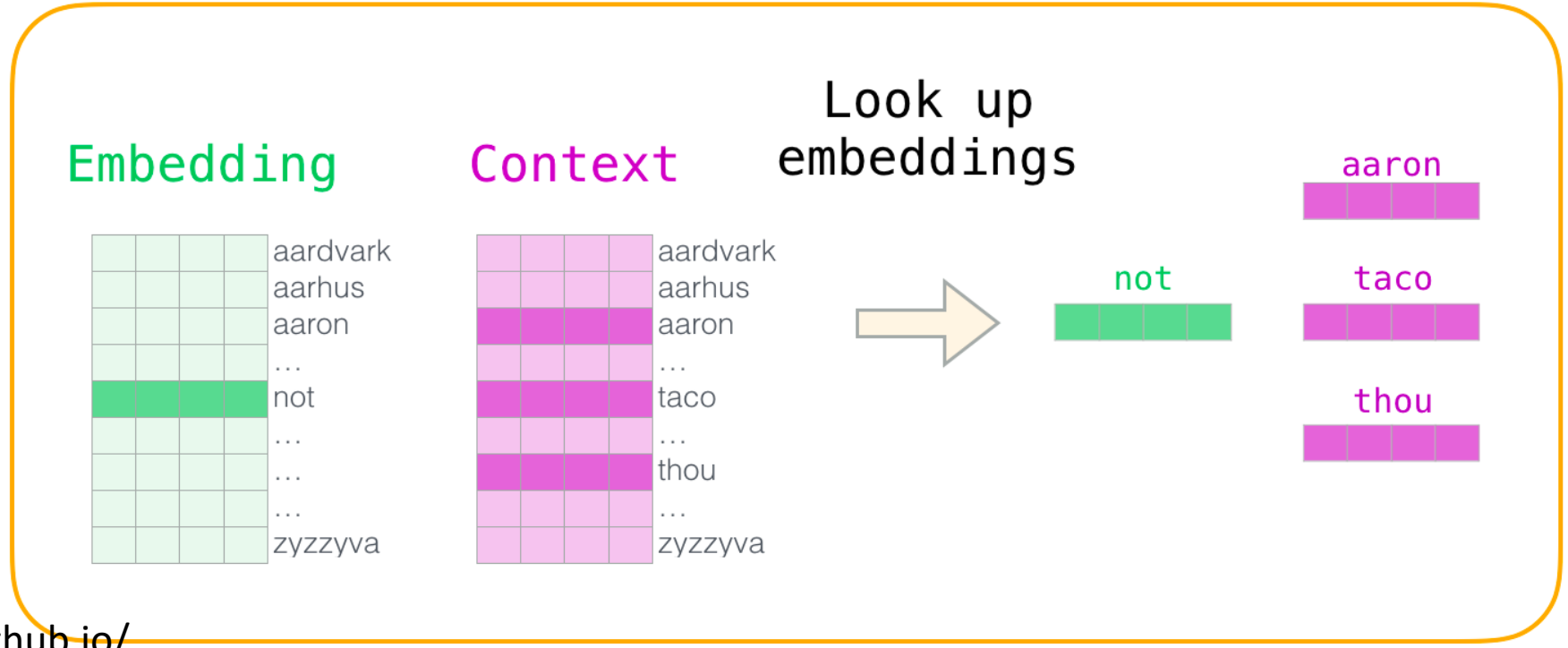
input word	output word	target
not	thou	1
not	aaron	0
not	taco	0
not	shalt	1
not	mango	0
not	finglonger	0
not	make	1
not	plumbus	0
...	...	...

model



Word2vec: Skipgram with negative sampling - Training

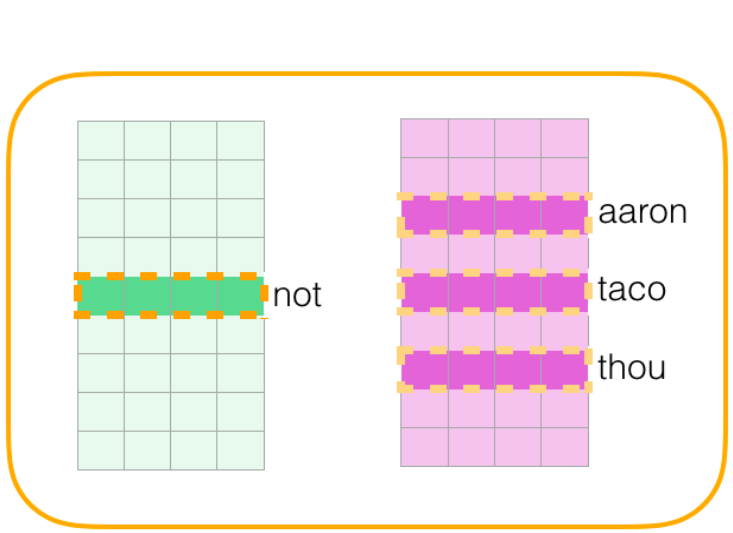
- Look up their embeddings
  - For the input word, look in the Embedding matrix
  - For the context words, look in the Context matrix



Word2vec: Skipgram with negative sampling - Training

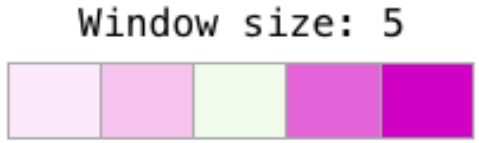
- Compute the predictions, the loss, gradients and model updates

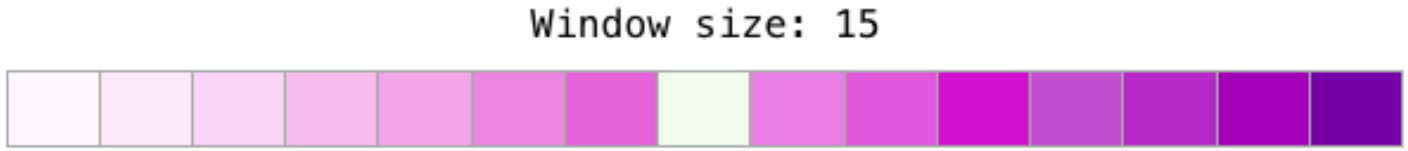
input word	output word	target	input • output	sigmoid()	Error
not	thou	1	0.2	0.55	0.45
not	aaron	0	-1.11	0.25	-0.25
not	taco	0	0.74	0.68	-0.68



Word2vec: Skipgram with negative sampling – Hyper-parameters

- Window size





- Number of negative samples

Negative samples: 2

input word	output word	target
make	shalt	1
make	aaron	0
make	taco	0

Negative samples: 5

input word	output word	target
make	shalt	1
make	aaron	0
make	taco	0
make	finglonger	0
make	plumbus	0
make	mango	0

Contents

- Word vectors
- Word2vec: Learning word vectors
  - Stochastic gradient descent (SGD)
  - Negative sampling
- GloVe
- Word vector evaluation

Capturing co-occurrence counts directly

- Word2Vec only takes local contexts into account
- It does not take advantage of global count statistics (explicitly) (cf. IDF)
- We can compute a word co-occurrence matrix (|V|x|V|)
- Similar to word2vec, use window around each word - captures both syntactic and semantic information
  - E.g. all sports terms will have similar entries
  - Well-studied topic before deep learning: “Latent Semantic Analysis”

Source: NTUNLP

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

Source: CS224n

Problems with simple co-occurrence vectors

- Very high dimensional: requires a lot of storage
- Subsequent classification models have sparsity issues
- Thus, models are less robust

Source: CS224n

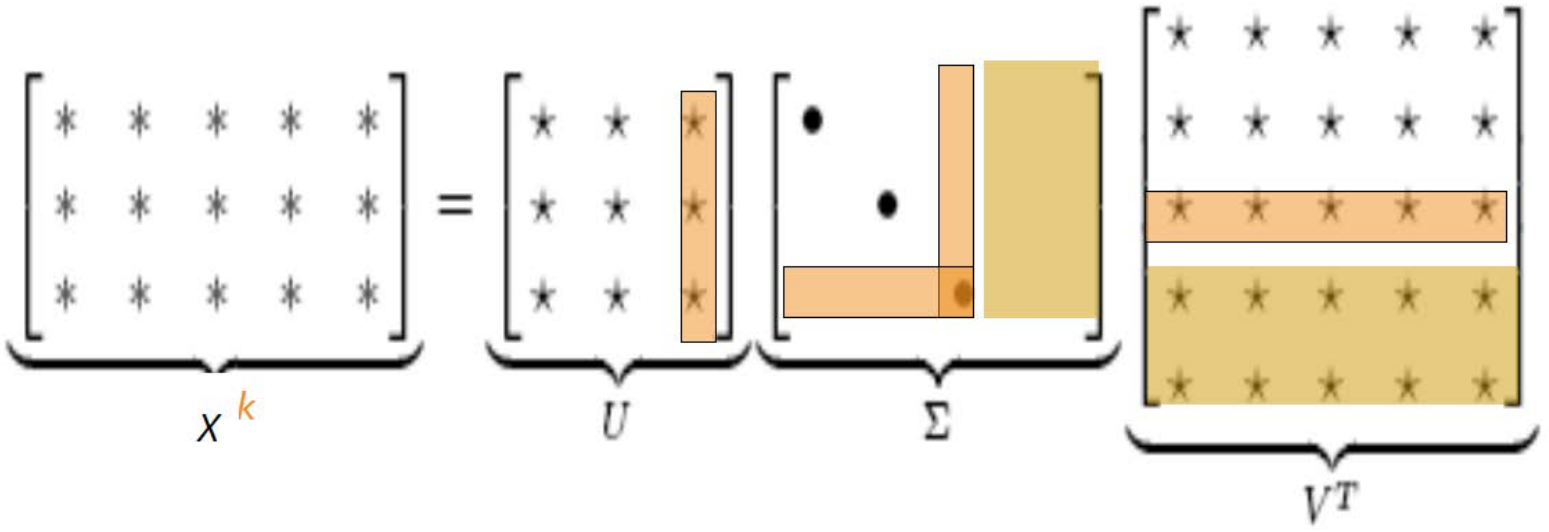
Solution: Low dimensional vectors

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

Source: CS224n

Method: Dimensionality Reduction ("Latent Semantic Analysis")

- Singular Value Decomposition (SVD) of co-occurrence matrix *X*
- Factorizes *X* into $U\Sigma V^T$, where *U* and *V* are orthonormal*
  - *: each column vector has length one and is orthogonal to the other column vectors
- Retain only *k* singular values
- $\widehat{t_j} = \Sigma_k^{-1} U_k^T t_j$



Source: CS224n

Simple SVD word vectors in Python

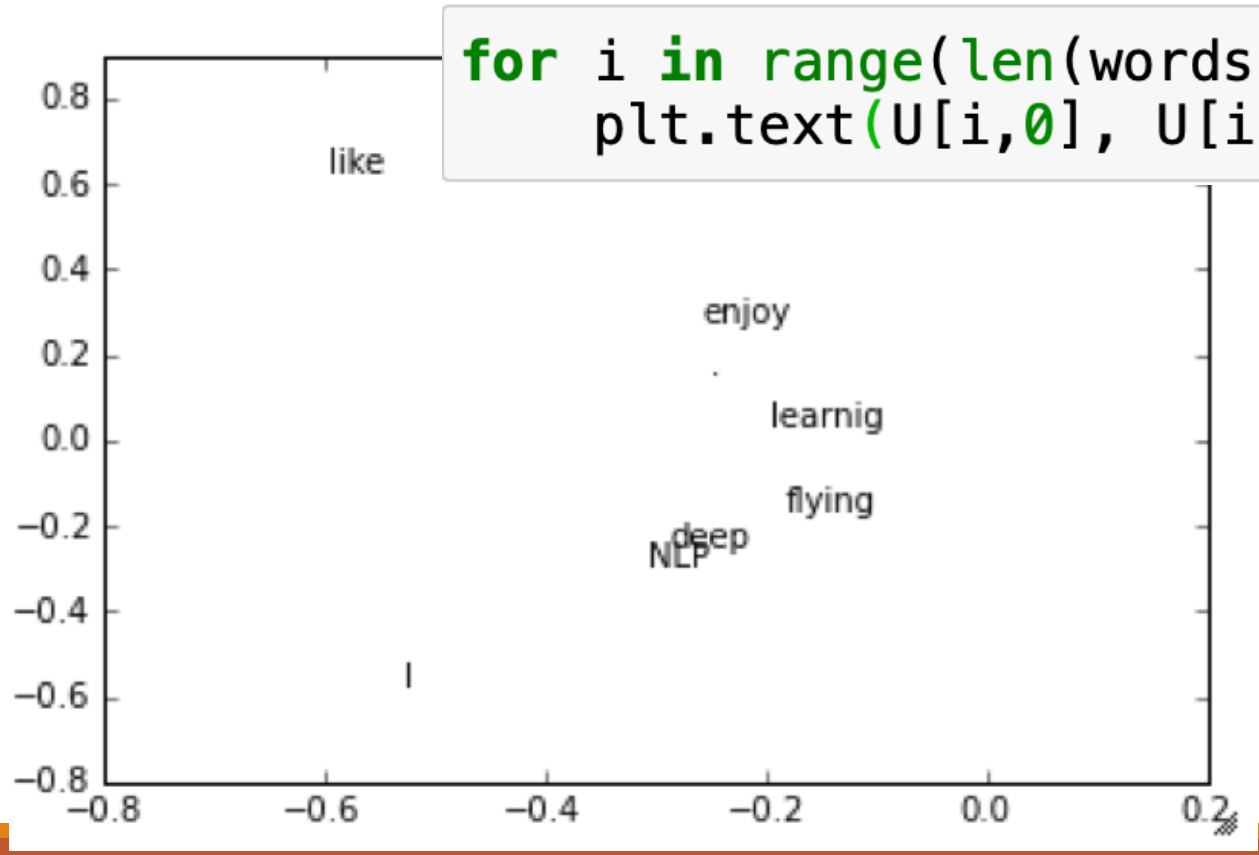
```
import numpy as np
la = np.linalg
words = ["I", "like", "enjoy",
         "deep","learnig","NLP","flying","."]
X = np.array([[0,2,1,0,0,0,0,0],
              [2,0,0,1,0,1,0,0],
              [1,0,0,0,0,0,1,0],
              [0,1,0,0,1,0,0,0],
              [0,0,0,1,0,0,0,1],
              [0,1,0,0,0,0,0,1],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


Source: CS224n

Simple SVD word vectors in Python

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

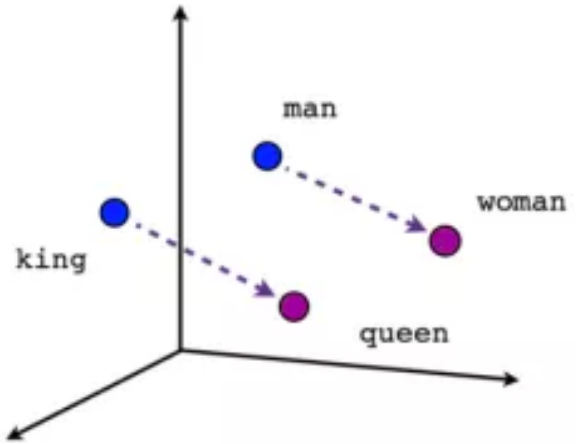
```
for i in range(len(words)):
    plt.text(U[i,0], U[i,1], words[i])
```



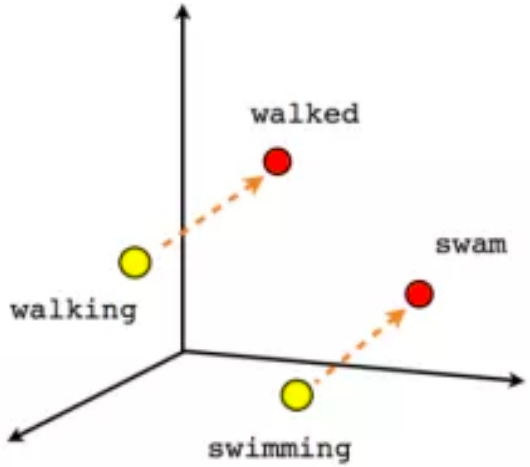
Source: CS224n

Problems with LSA vectors

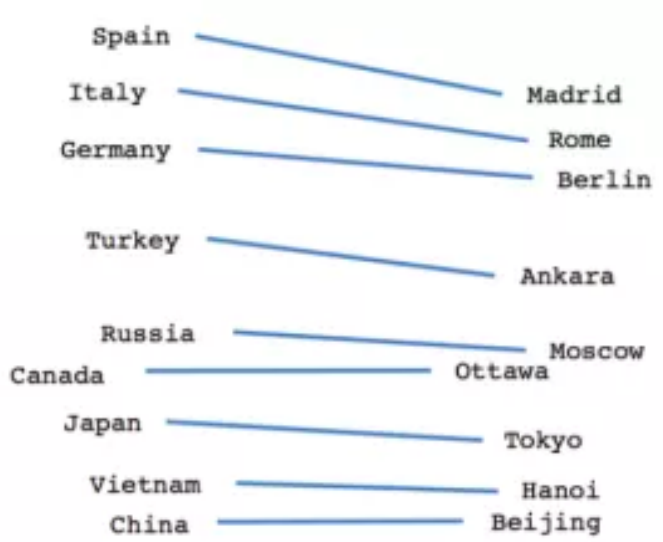
- Although LSA considers global statistics, difficult to show word analogies (without hacks)



Male-Female



Verb tense



Country-Capital

Source: NTUNLP

GloVe: Encoding meaning in vector differences

- Insights: 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

$$p(i|j) = \frac{\#\text{ of times word } j \text{ appears with word } i}{\#\text{ of times word } j \text{ appears}} = \frac{n(i,j)}{n(j)}$$

https://www.aclweb.org/anthology/D14-1162/

Source: CS224n

GloVe: Encoding meaning in vector differences

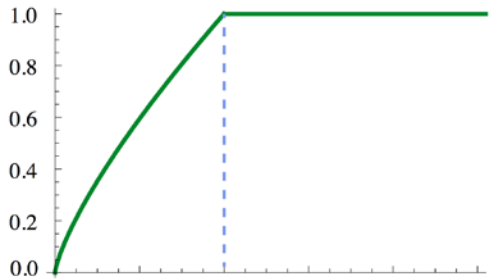
- Q: How can we capture ratios of co-occurrence probabilities as linear meaning components in a word vector space?
- A: Log-bilinear model $w_i \cdot w_j = w_i^T w_j = \log p(i|j)$

  with vector differences $w_x \cdot (w_a - w_b) = \log \frac{p(x|a)}{p(x|b)}$

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

$$p(i|j) = \frac{\#\text{ of times word } j \text{ appears with word } i}{\#\text{ of times word } j \text{ appears}} = \frac{n(i,j)}{n(j)} = X_{ij}$$

$f(X_{ij})$



https://www.aclweb.org/anthology/D14-1162/

GloVe: Encoding meaning in vector differences

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

Source: CS224n

GloVe: Encoding meaning in vector differences

- Nearest words to ‘frog’
  1. frogs
  2. toad
  3. litoria
  4. leptodactylidae
  5. rana
  6. lizard
  7. eleutherodactylus

litoria

leptodactylidae

rana

eleutherodactylus

Source: CS224n

Contents

- Word vectors
- Word2vec: Learning word vectors
  - Stochastic gradient descent (SGD)
  - Negative sampling
- GloVe
- Word vector evaluation

How to evaluate word vectors?

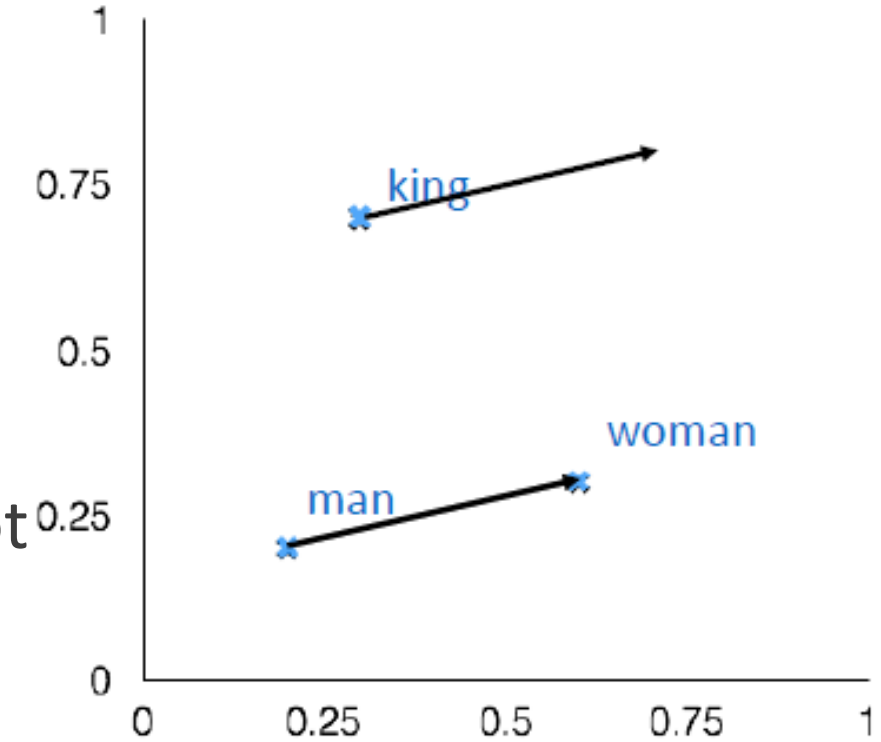
- Related to general evaluation in NLP: Intrinsic vs. extrinsic
- Intrinsic:
  - Evaluation on a specific/intermediate subtask
  - Fast to compute
  - Helps to understand that system
  - Not clear if really helpful unless correlation to real task is established
- Extrinsic:
  - Evaluation on a real task
  - 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!

Source: CS224n

Intrinsic word vector evaluation

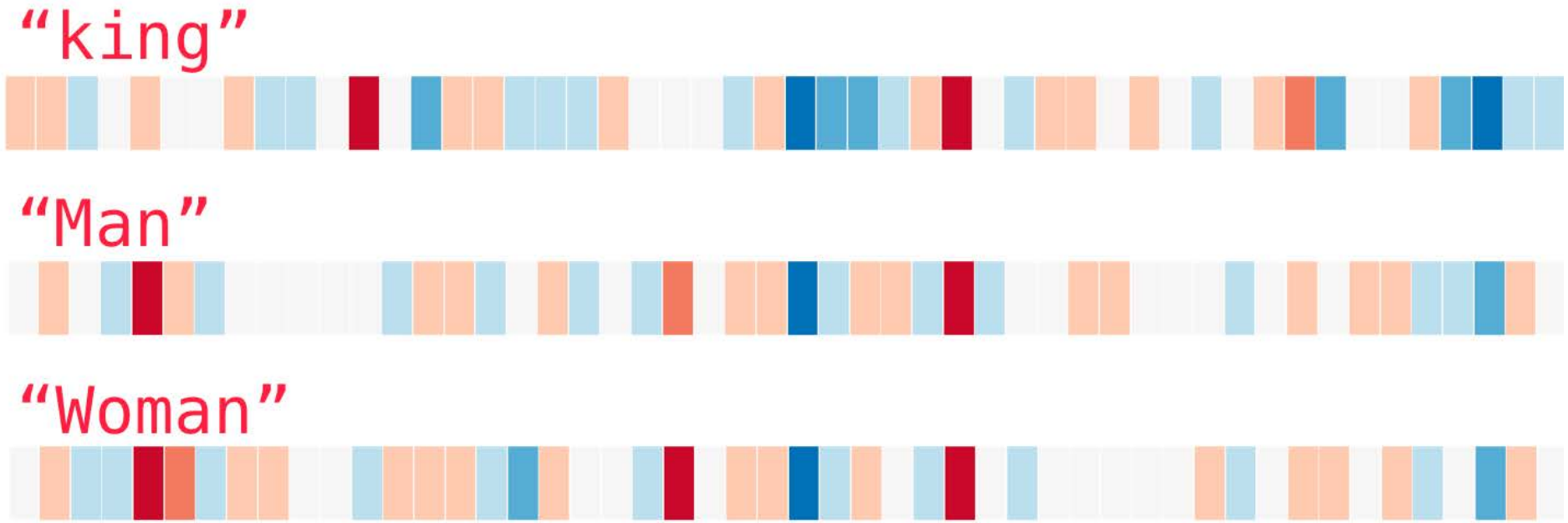
- Word Vector Analogies
  - a:b :: c:? (e.g. man:woman :: king:?)
- Evaluate word vectors by how well their cosine distance after addition captures intuitive semantic and syntactic analogy questions
- Discarding the input words from the search!
- Problem: What if the information is there but not linear?

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



Source: CS224n

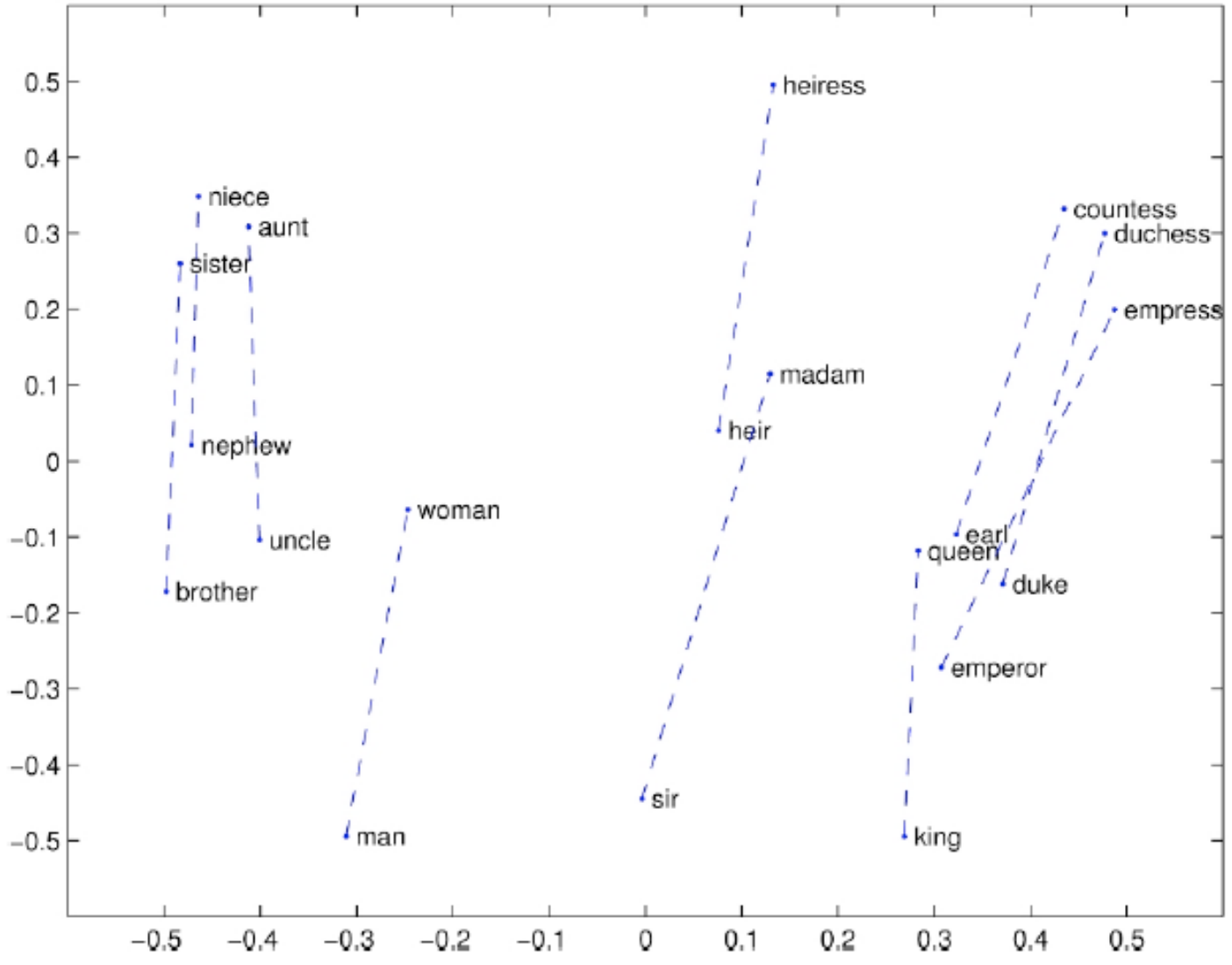
Intrinsic word vector evaluation



Intrinsic word vector evaluation

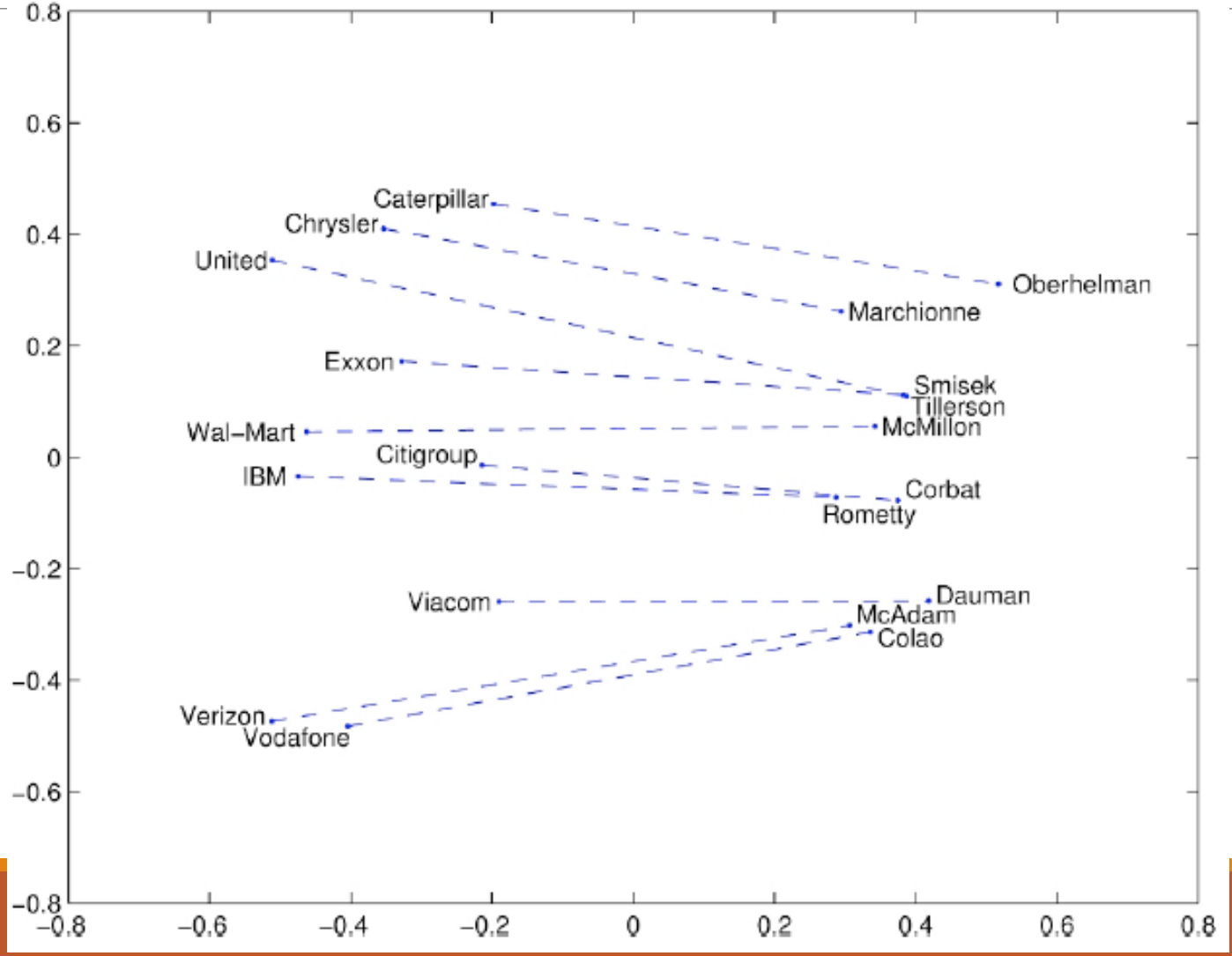
king − man + woman ~= queen

Glove Visualizations



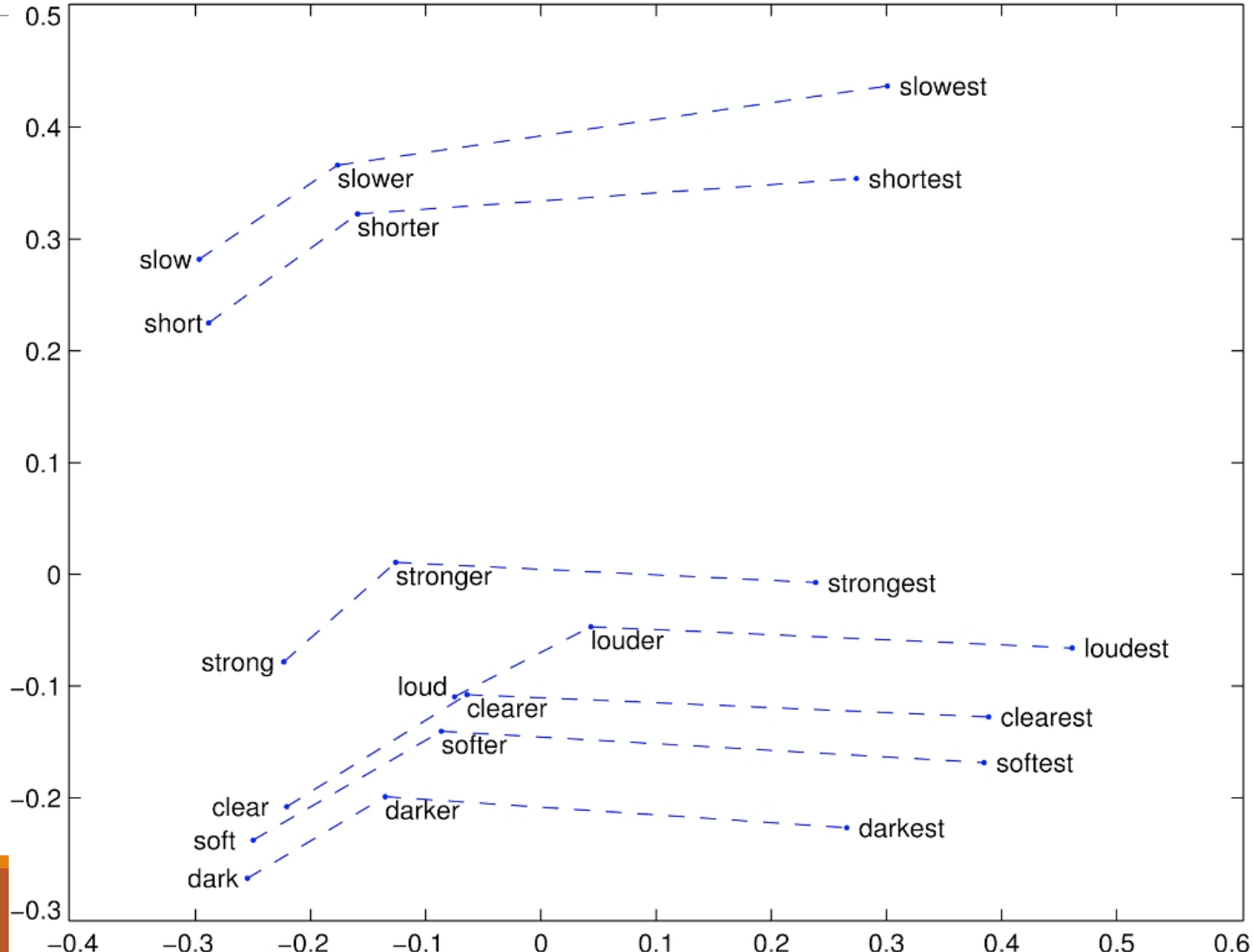
Source: CS224n

Glove Visualizations: Company - CEO



Source: CS224n

Glove Visualizations: Superlatives



Source: CS224n

Word Analogy Dataset

- : city-in-state problem: different cities
- 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

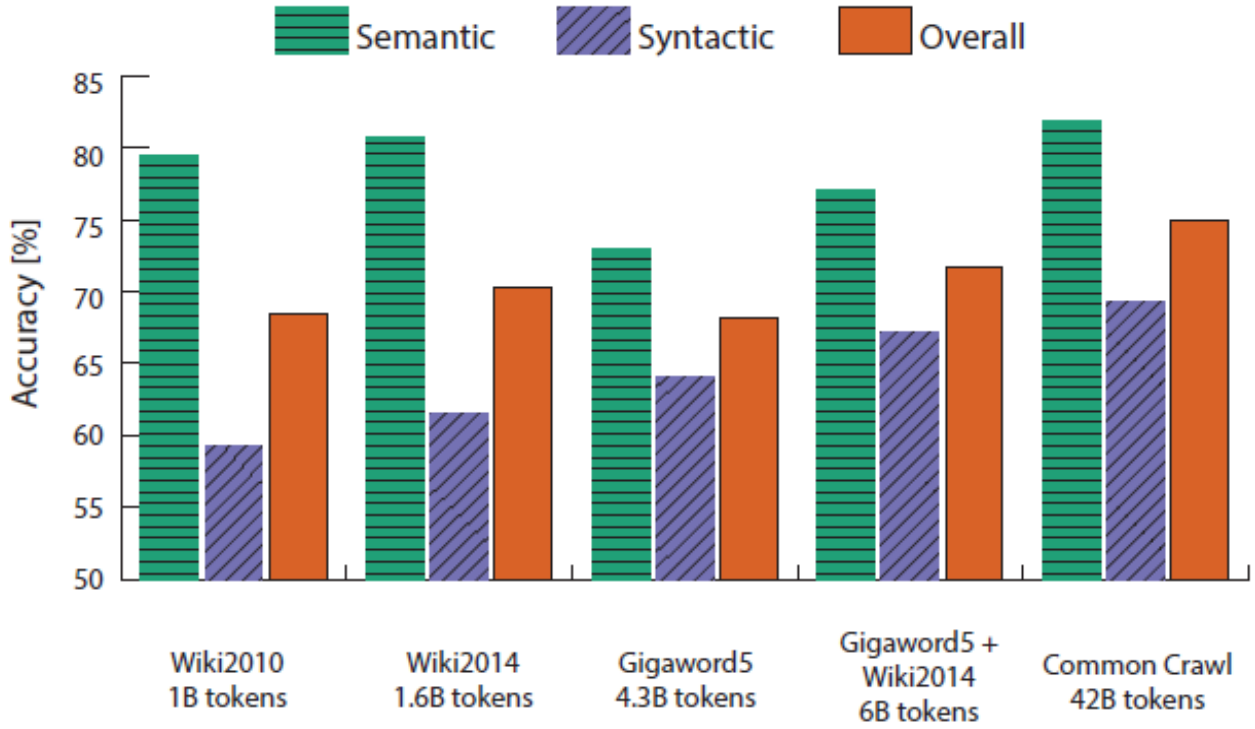
- : gram4-superlative
- bad worst big biggest
- bad worst bright brightest
- bad worst cold coldest
- bad worst cool coolest
- bad worst dark darkest
- bad worst easy easiest
- bad worst fast fastest
- bad worst good best
- bad worst great greatest

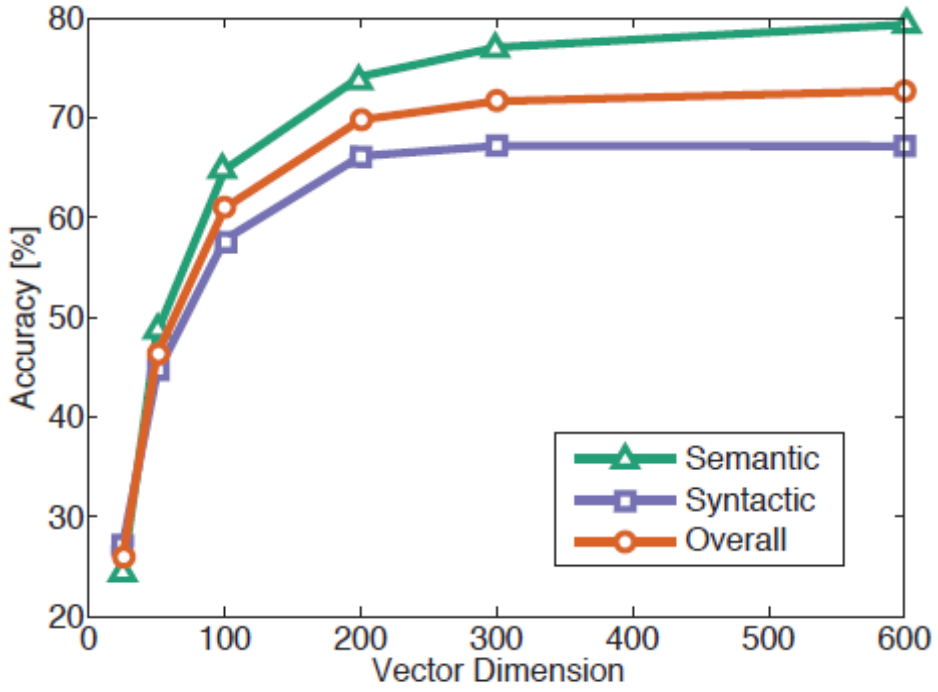
http://code.google.com/p/word2vec/source/browse/trunk/questions-words.txt

Source: CS224n

Analogy evaluation and hyper-parameters

- More data help
- Wikipedia is better than news text
- Good dimension is ~300





Source: CS224n

Another intrinsic word vector evaluation

- Word vector distances and their correlation with human judgments
- Example dataset: WordSim353 (similarity and relatedness)

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

http://alfonseca.org/eng/research/wordsim353.html

Source: CS224n

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	72.7	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	58.1	37.2
GloVe	6B	65.8	72.7	77.8	53.9	38.1
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

- WS353 (Mikolov et al. 2013): similar and related words
- RG (Rubenstein and Goodenough, 1965): 65 word pairs assessed by semantic similarity with a scale from 0 to 4
- MC (Miller and Charles, 1991): subset of RG containing 10 pairs with high similarity, 10 with middle similarity and 10
- with low similarity
- SCWS (Huang et al., 2012) ⇒ similarity ratings for different word senses
- RW (Luong et al., 2013) ⇒ 2,034 pairs of rare words assessed by semantic similarity with a scale from 0 to 10

https://www.cl.uni-heidelberg.de/courses/ss19/emb/material/evaluation.pdf

Source: CS224n

Extrinsic word vector evaluation

- Extrinsic evaluation of word vectors: All subsequent tasks in this course
- 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**

Source: CS224n

Other: Word sense

- Recall: Why is NLP hard? Human languages are ambiguous
  - E.g. I made her duck, Farmer Bill dies in house
- Most words have lots of meanings (or senses)!
  - Especially common words
  - Especially words that have existed for a long time
- "Contextual representation" (e.g. BERT) may help reduce the ambiguity issue

Source: CS224n