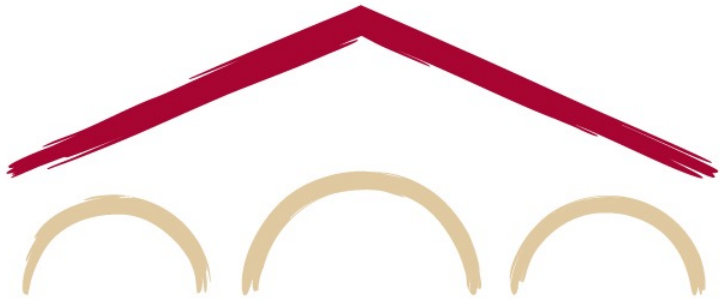
Natural Language Processing with Deep Learning CS224N/Ling284

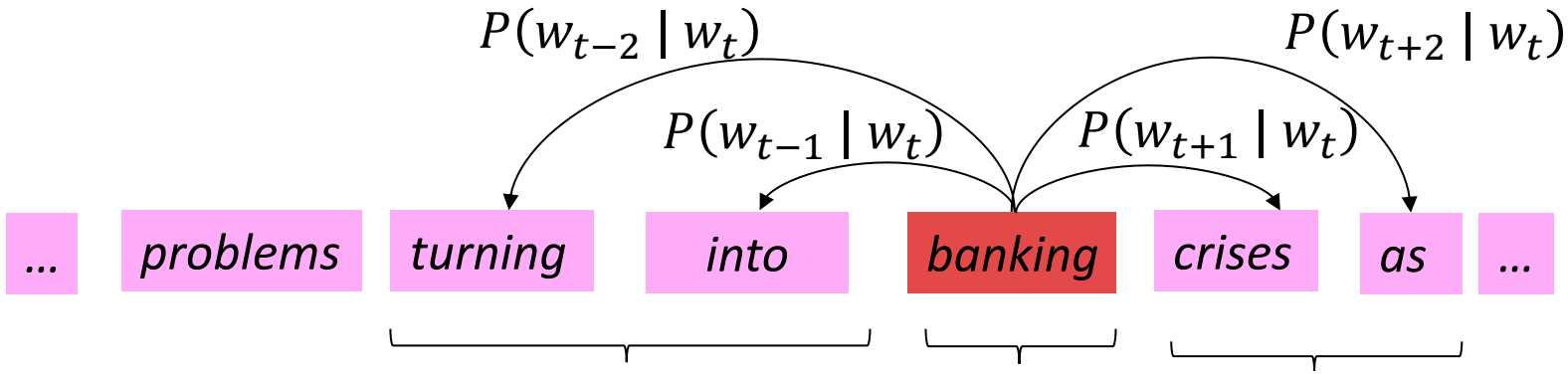


Christopher Manning

Lecture 2: Word Vectors, Word Senses, and Neural Classifiers

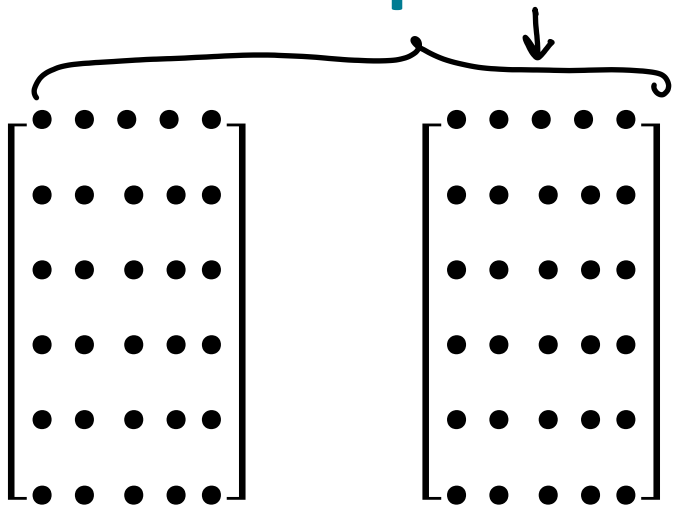
2. Review: Main idea of word2vec

- Start with random word vectors
- Iterate through each word position in the whole corpus
- Try to predict surrounding words using word vectors: $P(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{w \in V} \exp(u_w^T v_c)}$



- **Learning:** Update vectors so they can predict actual surrounding words better
- Doing no more than this, this algorithm learns word vectors that capture well word similarity and meaningful directions in a word space!

Word2vec parameters … and computations



U V

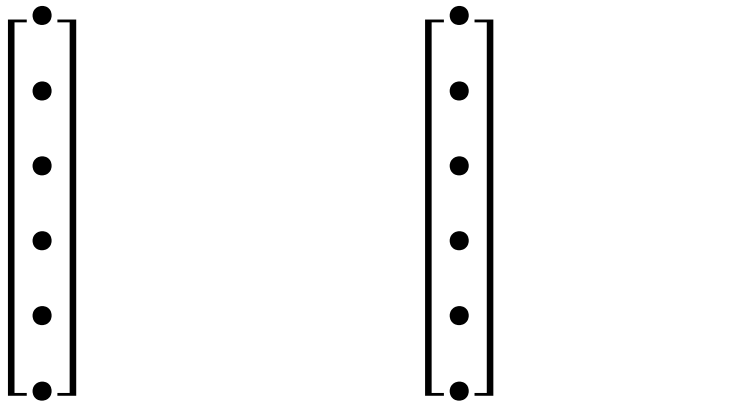
outside center

WORD VECTORS WORD VECTORS

"Bag of words" model!

FOR EACH WORD

AND THEN WE TAKE THE DOT PRODUCT



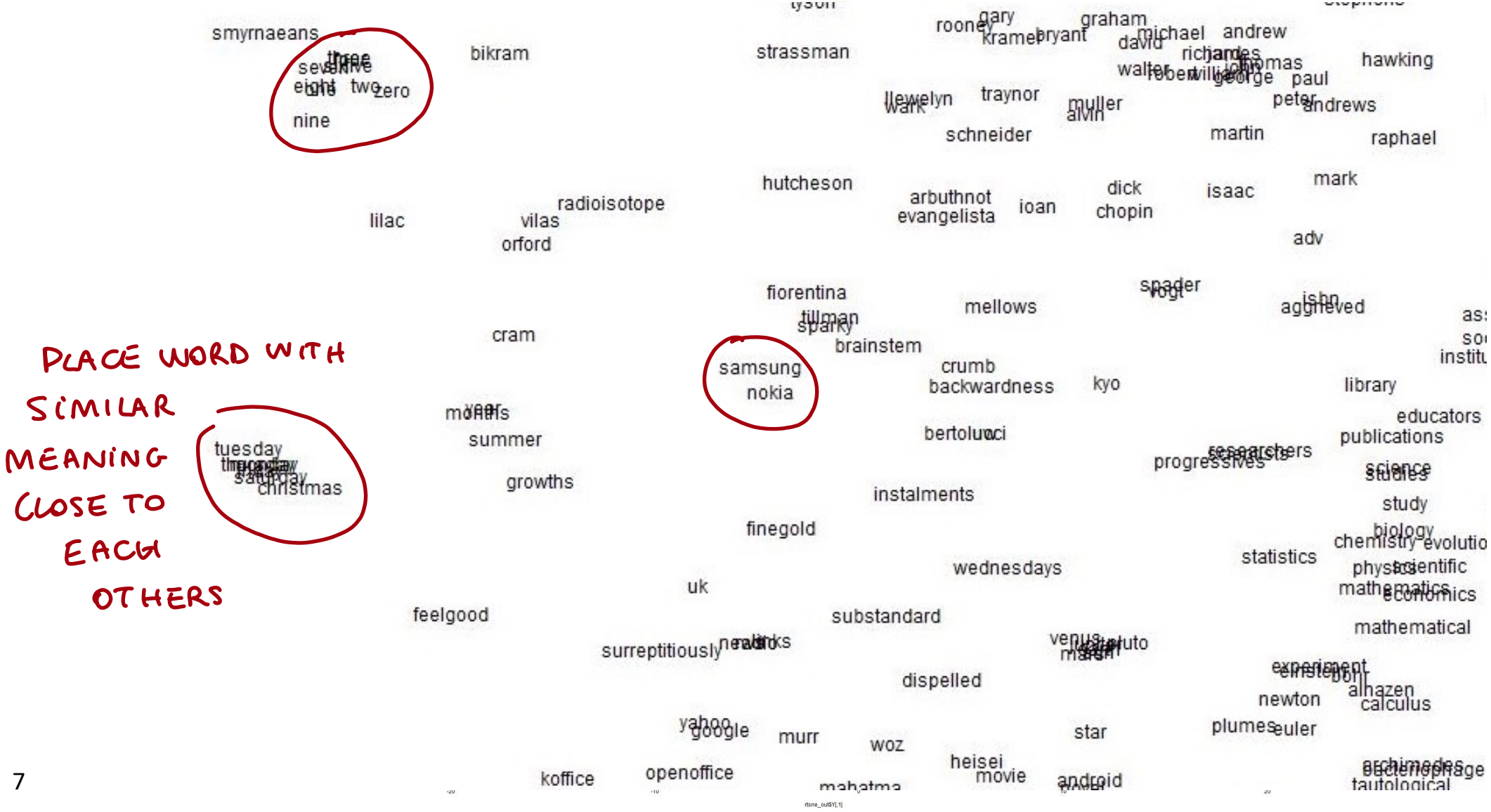
$U.v_4{}^T$ $\text{softmax}(U.v_4{}^T)$

dot product probabilities

The model makes the same predictions at each position

We want a model that gives a reasonably high probability estimate to *all* words that occur in the context (at all often)

Word2vec maximizes objective function by putting similar words nearby in space



The skip-gram model with negative sampling (HW2)

- The normalization term is computationally expensive (when many output classes):

NAIVE SOFTMAX

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

A big sum over words

- Hence, in standard word2vec and HW2 you implement the skip-gram model with **negative sampling**

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

Word2vec algorithm family (Mikolov et al. 2013): More details

Why two vectors? → Easier optimization. Average both at the end

- But can implement the algorithm with just one vector per word … and it helps a bit

Two model variants:

1. Skip-grams (SG)

   Predict context ("outside") words (position independent) given center word

2. Continuous Bag of Words (CBOW)

   Predict center word from (bag of) context words

We presented: **Skip-gram model**

Loss functions for training:

1. Naïve softmax (simple but expensive loss function, when many output classes)
2. More optimized variants like hierarchical softmax
3. Negative sampling

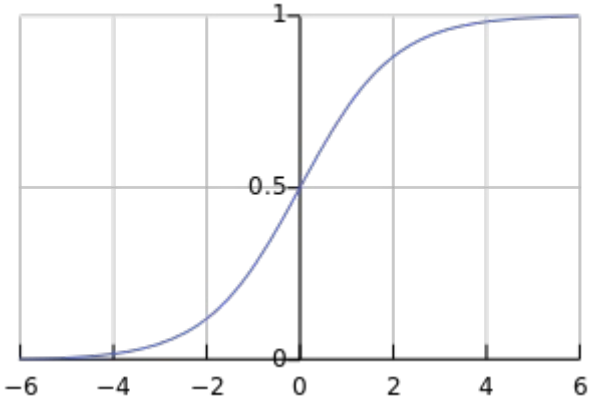
So far, we explained **naïve softmax**

The skip-gram model with negative sampling (HW2)

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

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

- The logistic/sigmoid function: $\sigma(x) = \frac{1}{1+e^{-x}}$
  (we'll become good friends soon)
- We maximize the probability of two words co-occurring in first log and minimize probability of noise words in second part



The skip-gram model with negative sampling (HW2)

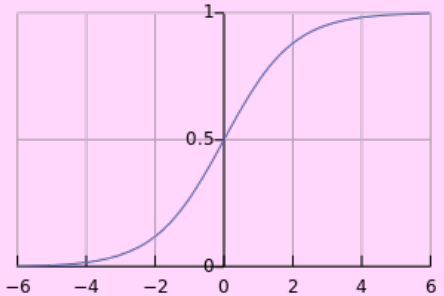
NEGATIVE LOG LIKELIHOOD BUT OF RANDOM WORDS

- Using notation consistent with this class and HW2:

$$J_{neg-sample}(\boldsymbol{u}_o, \boldsymbol{v}_c, U) = -\log \sigma(\boldsymbol{u}_o^T \boldsymbol{v}_c) - \sum_{k \in \{K \text{ sampled indices}\}} \log \sigma(\boldsymbol{u}_k^T \boldsymbol{v}_c)$$

NEGATIVE LOG LIKELIHOOD OF THE SIGMOID OF THE DOT PRODUCT

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



- We take *k* negative samples (using word probabilities)
- Maximize probability that real outside word appears; minimize probability that random words appear around center word

- Sample with $P(w)=U(w)^{3/4}/Z$, the unigram distribution U(w) raised to the 3/4 power (We provide this function in the starter code).
- The power makes less frequent words be sampled more often

Stochastic gradients with negative sampling [aside]

- We iteratively take gradients at each window for SGD
- In each window, we only have at most $2m + 1$ words plus $2km$ negative words with negative sampling, so $\nabla_\theta J_t(\theta)$ is very sparse!

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

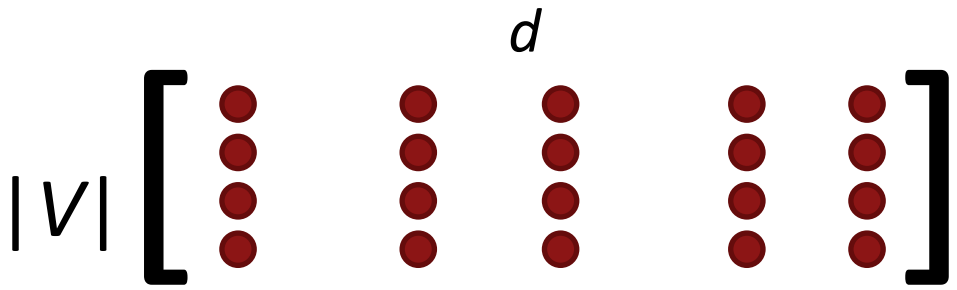
Stochastic gradients with with negative sampling [aside]

- 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

Rows not columns in actual DL packages!

MORE EFFICIENT TO MANIPULATE THEM



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

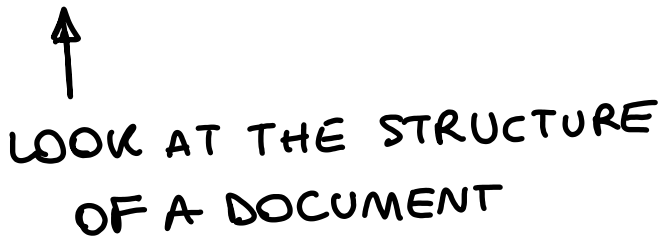
This is also a particular issue with more advanced optimization methods in the Adagrad family

3. Why not capture co-occurrence counts directly?

There's something weird about iterating through the whole corpus (perhaps many times); why don't we just accumulate all the statistics of what words appear near each other?!?

Building a co-occurrence matrix *X*

- 2 options: windows vs. full document
- Window: Similar to word2vec, use window around each word → captures some syntactic and semantic information ("word space")
- Word-document co-occurrence matrix will give general topics (all sports terms will have similar entries) leading to "Latent Semantic Analysis" ("document space")



Example: Window based co-occurrence matrix

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

HOW MANY TIMES WORDS CO-OCCURE IN A WINDOW

- 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

Co-occurrence vectors

- Simple count co-occurrence vectors
  - Vectors increase in size with vocabulary
  - Very high dimensional: require a lot of storage (though sparse)
  - Subsequent classification models have sparsity issues → Models are less robust

MODELS ARE MORE SENSIBLE AND TEND TO BE NOISY

- 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?

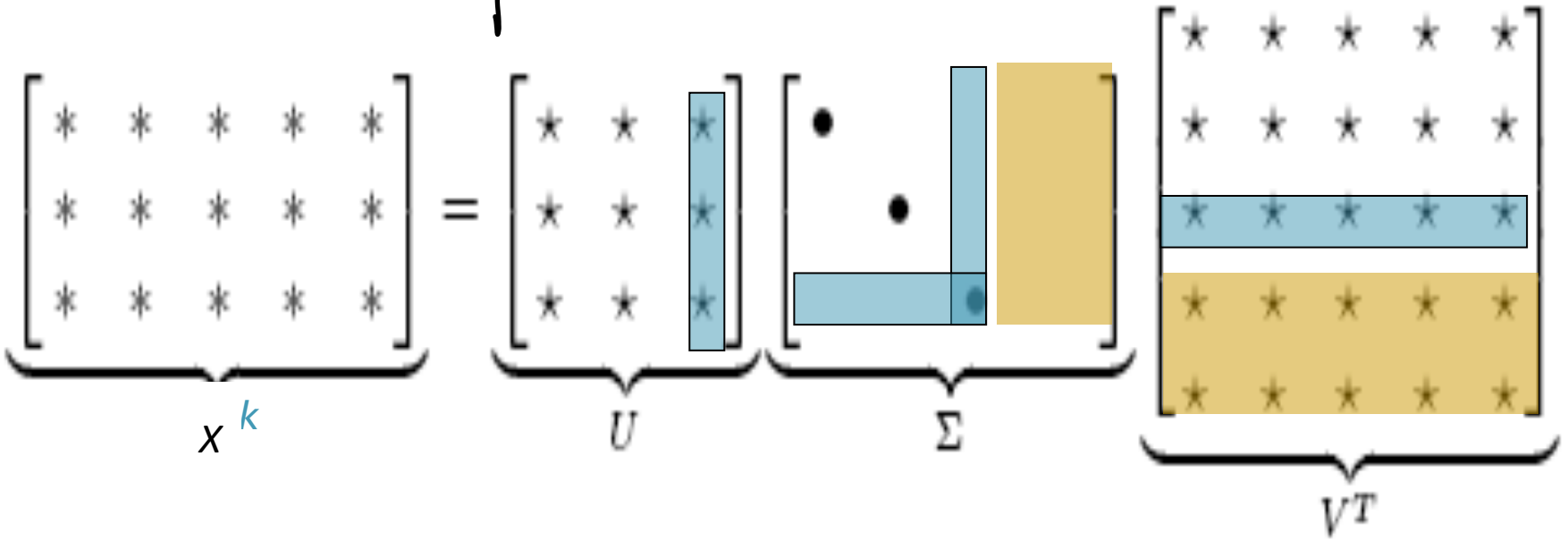
Classic Method: Dimensionality Reduction on X (HW1)

Singular Value Decomposition of co-occurrence matrix *X*

USED TO REDUCE DIMENSIONALITY

Factorizes *X* into $U\Sigma V^T$, where *U* and *V* are orthonormal

DECOMPOSITION INTO 3 OTHER MATRICES



Retain only *k* singular values, in order to generalize.

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

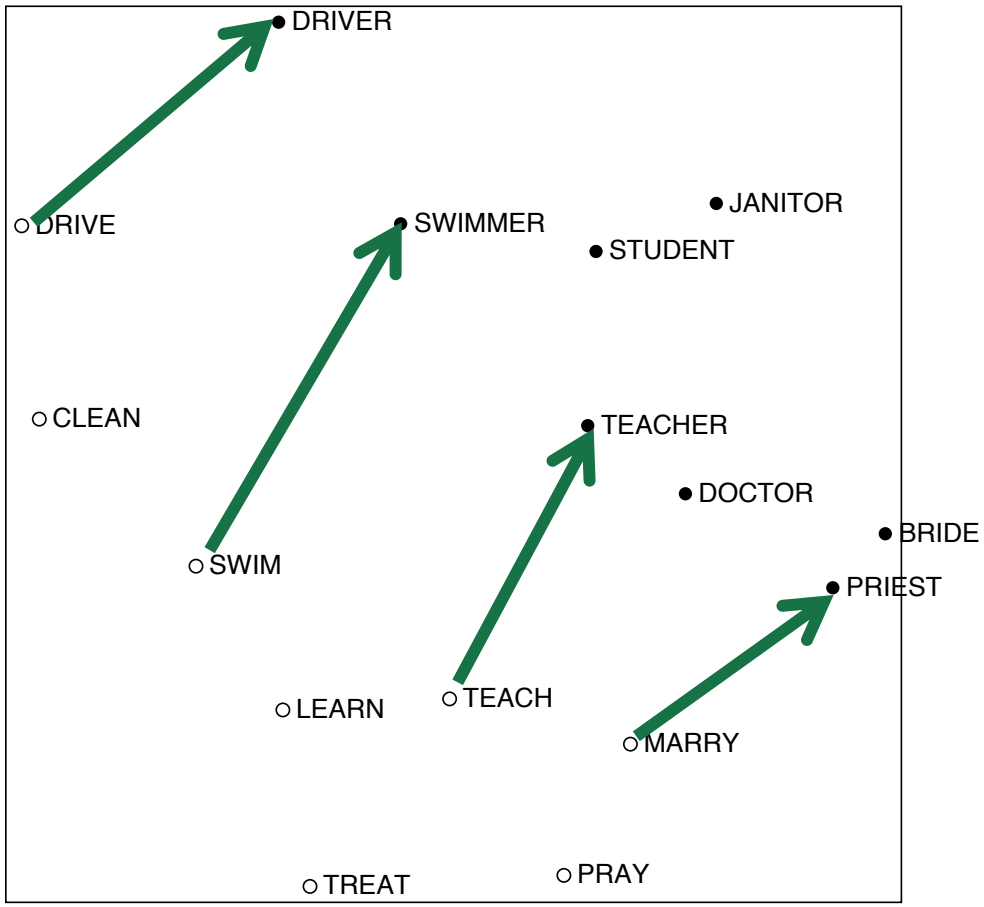
Classic linear algebra result. Expensive to compute for large matrices.

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

- Running an SVD on raw counts doesn't work well!!!

- Scaling the counts in the cells can help ***a lot***
  - Problem: function words (*the, he, has*) are too frequent → syntax has too much impact. Some fixes:
    - log the frequencies
    - min(*X*, *t*), with $t \approx 100$
    - Ignore the function words
- Ramped windows that count closer words more than further away words
- Use Pearson correlations instead of counts, then set negative values to 0
- Etc.

Interesting semantic patterns emerge in the scaled vectors



COALS model from
Rohde et al. ms., 2005. An Improved Model of Semantic Similarity Based on Lexical Co-Occurrence

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

RELATIONSHIP KING → QUEEN, DRIVING → DRIVER, ...

Q: How can we capture ratios of co-occurrence probabilities as linear meaning components in a word vector space? → RATIOS OF CO-OCCURRENCE PROBABILITIES

A: Log-bilinear model: $w_i \cdot w_j = \log P(i|j)$

DOT PRODUCT OF 2 VECTORS

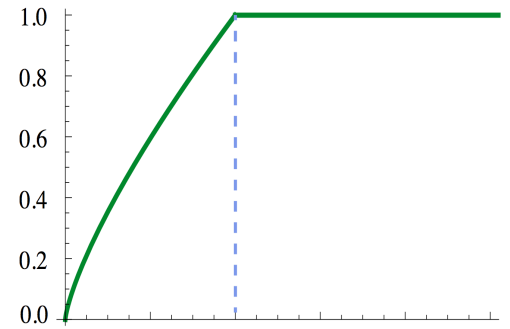
LOG PROB OF CO-OCCURENCE

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

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

SCALE BASED ON THE FREQUENCY OF A WORD

$f \sim$



- Fast training
- Scalable to huge corpora

4. How to evaluate word vectors?

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

Intrinsic word vector evaluation

- Word Vector Analogies

WE USE ANALYSIS TO ASK THE MODEL WHICH IS WORD THAT IS CLOSEST

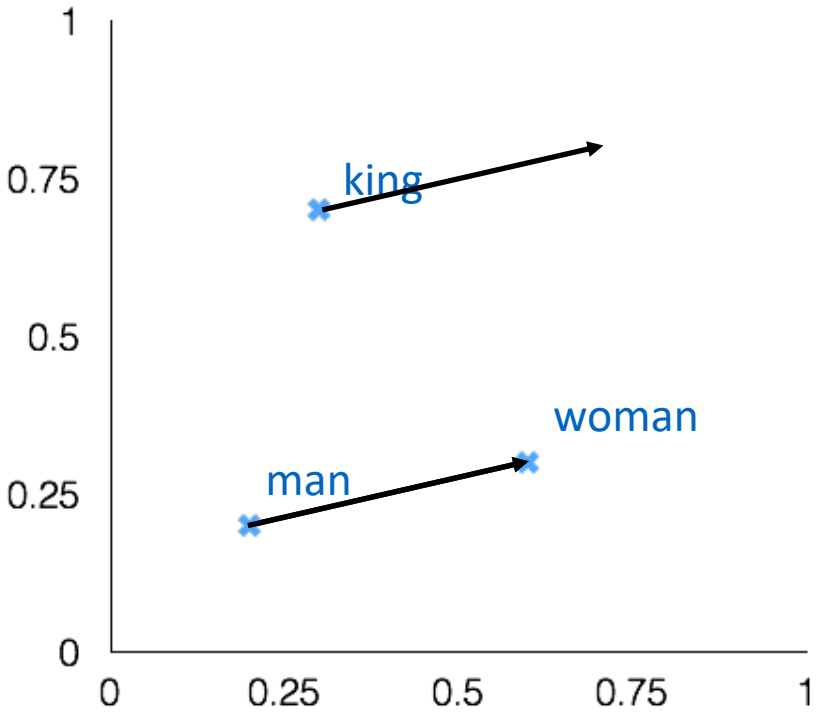
a:b :: c:?

man:woman :: king:?

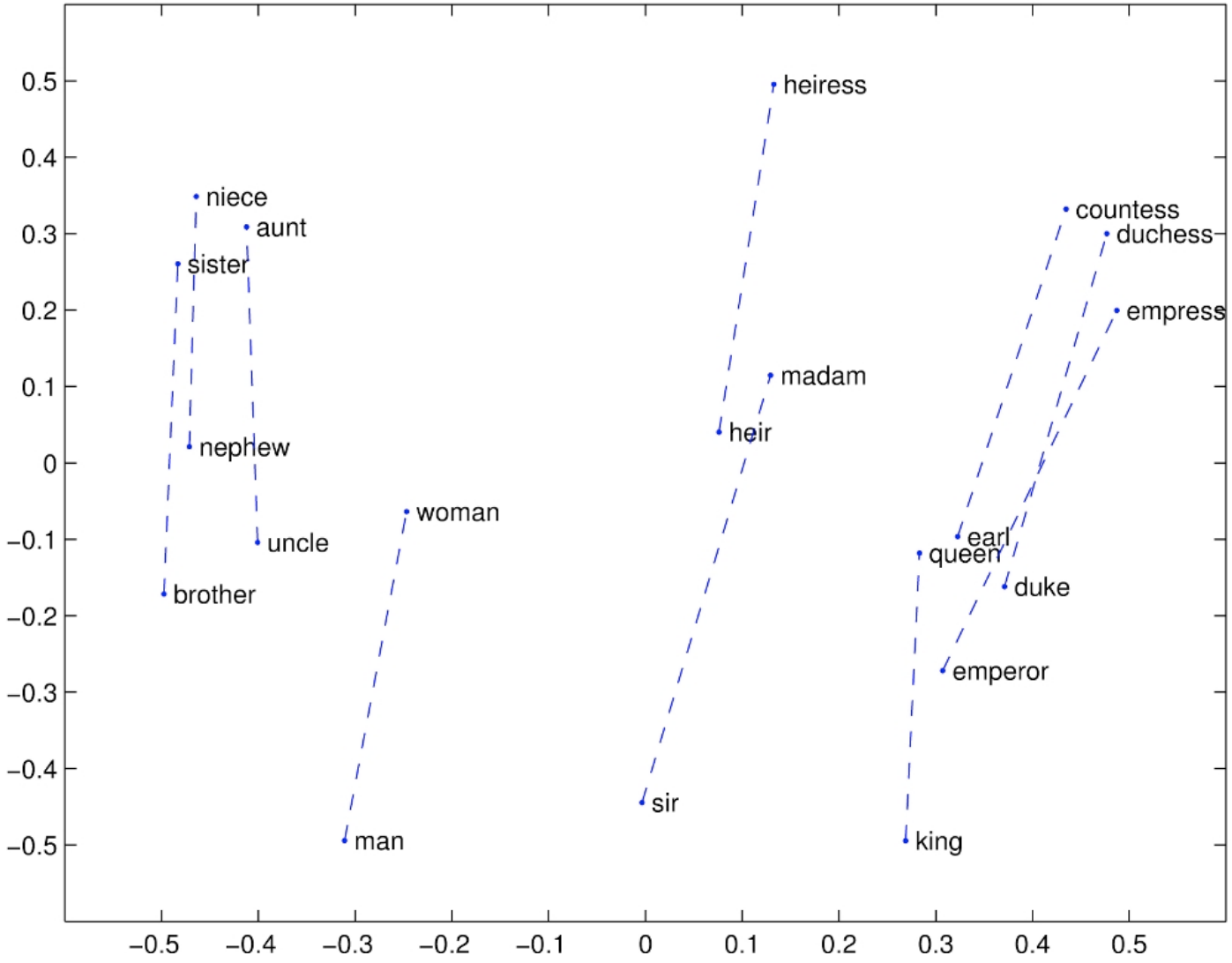
ACCURACY SCORE

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

- 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 Visualization



Meaning similarity: 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

similarity judgement done by humans

Compare it with the similarity judgement done by our model

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

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

Extrinsic word vector evaluation

- One example where good word vectors should help directly: **named entity recognition**: identifying references to a person, organization or location: Chris Manning lives in Palo Alto.

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**

- Subsequent NLP tasks in this class are other examples. So, more examples soon.

5. 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?

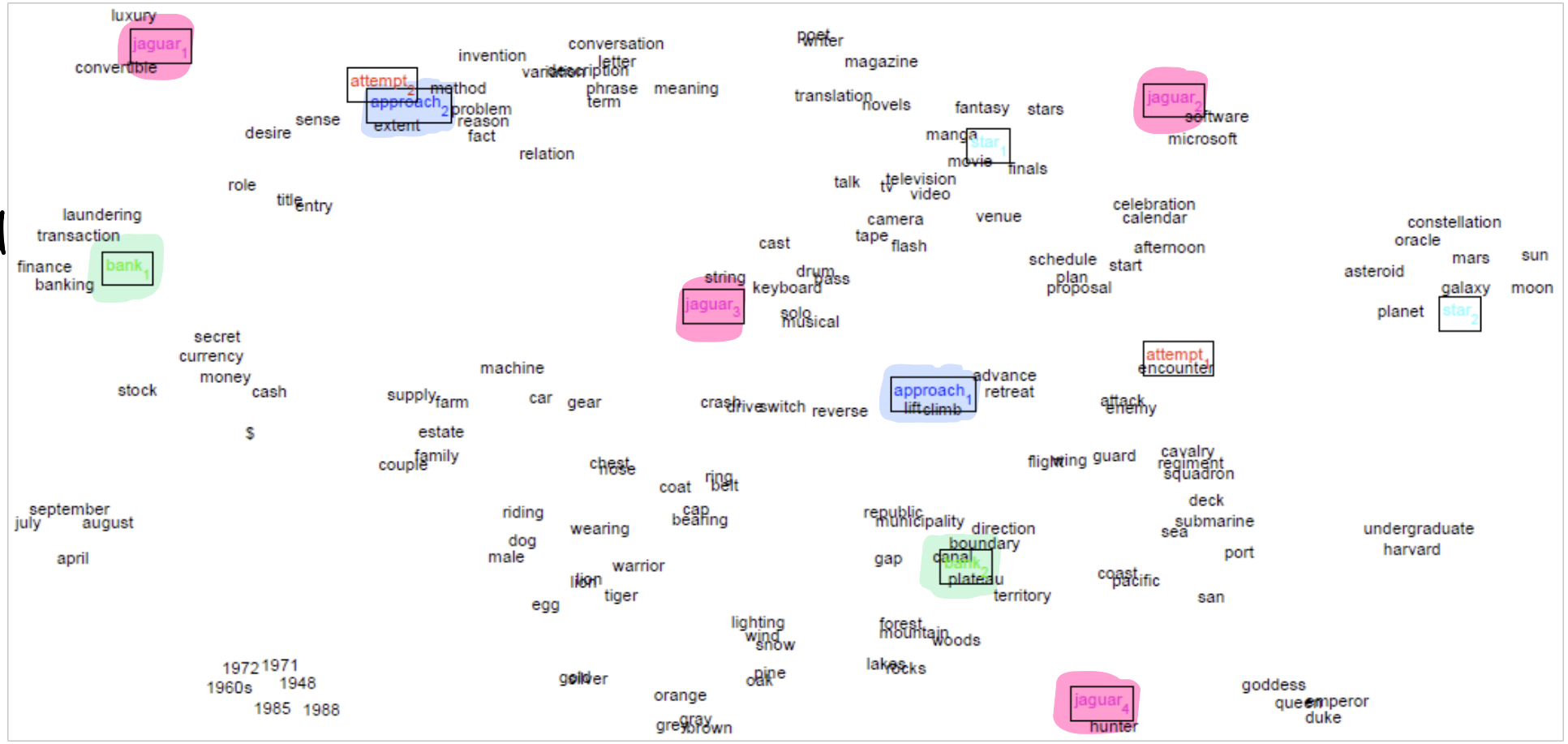
pike

- 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!*

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 $bank_1$, $bank_2$, etc.

NOT VERY USED IN PRACTICE. - complicated



Linear Algebraic Structure of Word Senses, with Applications to Polysemy (Arora, …, Ma, …, TACL 2018)

IN PRACTICE WE USE JUST ONE WORD VECTOR TO REPRESENT EACH WORD, USING THE WEIGHTED SUM TO CONSIDER ALL THE SENSES

- Different senses of a word reside in a linear superposition (weighted sum) in standard word embeddings like word2vec
- $v_{\text{pike}} = \alpha_1 v_{\text{pike}_1} + \alpha_2 v_{\text{pike}_2} + \alpha_3 v_{\text{pike}_3}$
- Where $\alpha_1 = \frac{f_1}{f_1+f_2+f_3}$, etc., for frequency f
- Surprising result:
  - Because of ideas from *sparse coding* you can actually separate out the senses (providing they are relatively common)!

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