Amirkabir University of Technology
(Tehran Polytechnic)

Natural Language Processing

Lecture 8: Dense Word Representation

Amirkabir University of Technology
Dr Momtazi

Outline

- **Motivation**
- Dense Vectors via SVD
- Embeddings by neural language models
- Evaluation
- Applications

Taxonomy vs Context Vector

- Great as resource but missing nuances
- Missing new words (impossible to keep up to date)
- Subjective
- Requires human labor to create and adapt
- Hard to compute accurate word similarity

Sparse vs Dense Vectors

- Discrete representation
  - **long** (length |V|= 20,000 to 50,000)
  - **sparse** (most elements are zero)

- Dense representation
  - **short** (length 200-1000)
  - **dense** (most elements are non-zero)

Problems with the Discrete Representation

- In vector space terms, this is a vector with one 1 and a lot of zeroes
- We call this a "one-hot" representation

  [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)

car: [0 0 0 0 0 0 0 0 0 0 1 0 0 0 0]

automobile: [0 0 0 0 0 0 0 1 0 0 0 0 0 0 0]

Sparse vs Dense Vectors

- Why dense vectors?
  - Short vectors may be easier to use as features in machine learning (less weights to tune)
  - Dense vectors may generalize better than storing explicit counts
  - They may do better at capturing synonymy:
    - *car* and *automobile* are synonyms
    - But they are represented as distinct dimensions
    - This fails to capture similarity between a word with *car* as a neighbor and a word with *automobile* as a neighbor

Main Approaches

- Singular Value Decomposition (SVD)
  - A special case of this is called LSA – Latent Semantic Analysis
- “Neural Language Model”-inspired predictive models
  - Skip-grams and CBOW
- Brown clustering

Outline

- Motivation
- **Dense Vectors via SVD**
- Embeddings by neural language models
- Evaluation
- Applications

Intuition

- Approximate an N-dimensional dataset using fewer dimensions
- By first rotating the axes into a new space
- In which the highest order dimension captures the most variance in the original dataset
- And the next dimension captures the next most variance, etc.
- Many such (related) methods:
  - PCA – principle components analysis
  - Factor Analysis
  - SVD

Singular Value Decomposition (SVD)

Any rectangular w x c matrix X equals the product of 3 matrices:

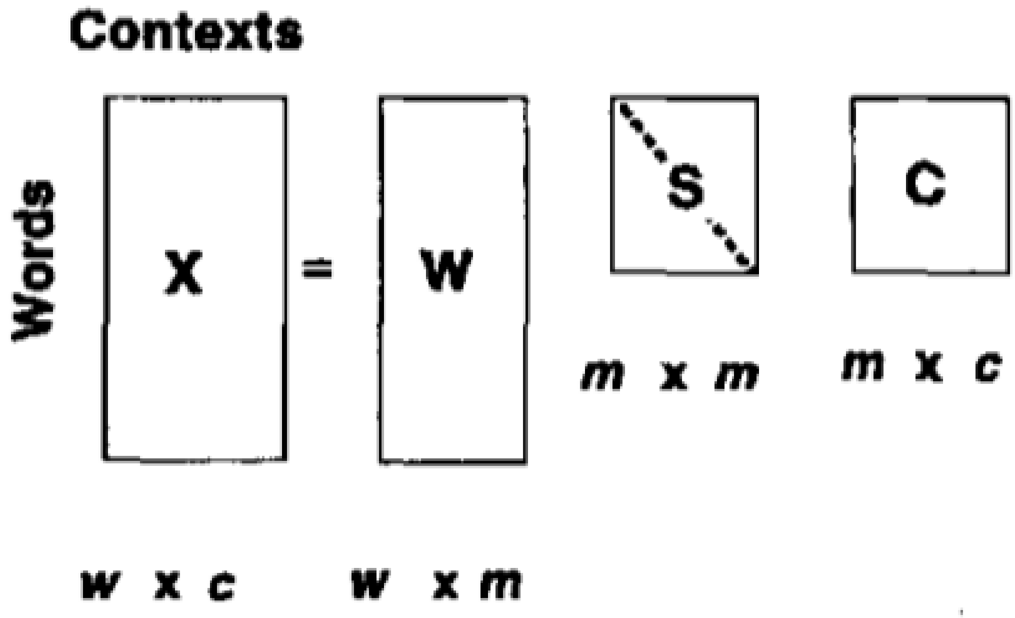
W: rows corresponding to original but m columns represents a dimension in a new latent space, such that

- M column vectors are orthogonal to each other
- Columns are ordered by the amount of variance in the dataset each new dimension accounts for

S: diagonal *m* x *m* matrix of **singular values** expressing the importance of each dimension.

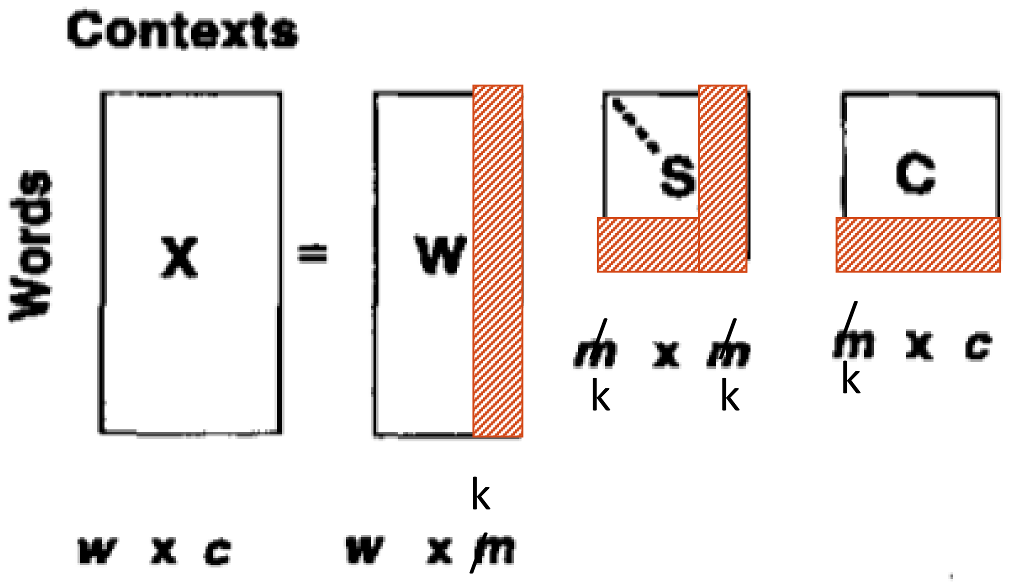
C: columns corresponding to original but m rows corresponding to singular values

Singular Value Decomposition (SVD)



Latent Semantic Analysis (LSA)

- SVD applied to term-document matrix
- Instead of keeping all m dimensions, we just keep the top k singular values. Let's say 300.
  - The result is a least-squares approximation to the original X
  - But instead of multiplying, we'll just make use of W.
- Each row of W:
  - A k-dimensional vector
  - Representing a word
- Each column of C:
  - A k-dimensional vector
  - Representing a document



Latent Semantic Analysis (LSA)

- 300 dimensions are commonly used
- The cells are commonly weighted by a product of two weights
  - Local weight: Log term frequency
  - Global weight: either idf or an entropy measure

Let’s return to PPMI word-word matrices

- Can we apply SVD to them?

SVD Applied to Term-Term Matrix

$$\underbrace{\begin{bmatrix} & & \\ & X & \\ & & \end{bmatrix}}_{|V| \times |V|} = \underbrace{\begin{bmatrix} & & \\ & W & \\ & & \end{bmatrix}}_{|V| \times |V|} \underbrace{\begin{bmatrix} \sigma_1 & 0 & 0 & \dots & 0 \\ 0 & \sigma_2 & 0 & \dots & 0 \\ 0 & 0 & \sigma_3 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & \sigma_V \end{bmatrix}}_{|V| \times |V|} \underbrace{\begin{bmatrix} & & \\ & C & \\ & & \end{bmatrix}}_{|V| \times |V|}$$

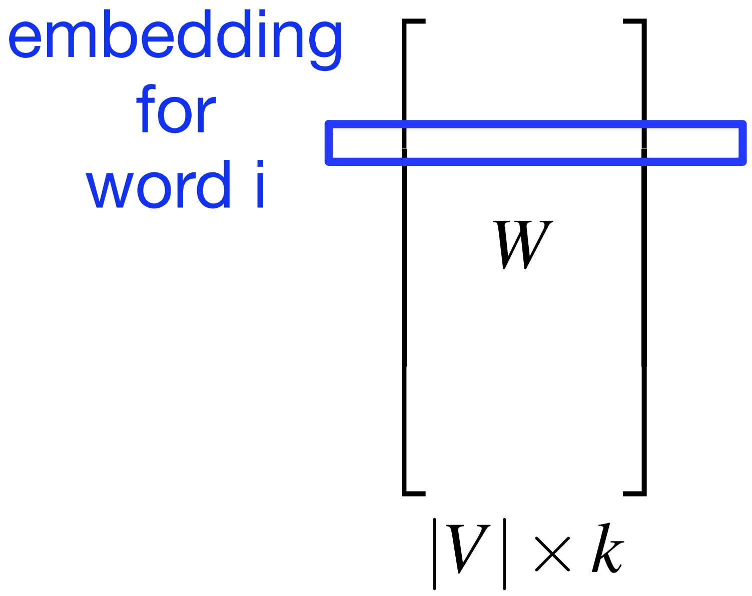
(assuming the matrix has rank |V|)

Truncated SVD on Term-Term Matrix

$$\underset{|V| \times |V|}{\begin{bmatrix} \\ \\ X \\ \\ \\ \end{bmatrix}} = \underset{|V| \times k}{\begin{bmatrix} \\ \\ W \\ \\ \\ \end{bmatrix}} \underset{k \times k}{\begin{bmatrix} \sigma_1 & 0 & 0 & \dots & 0 \\ 0 & \sigma_2 & 0 & \dots & 0 \\ 0 & 0 & \sigma_3 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & \sigma_k \end{bmatrix}} \underset{k \times |V|}{\begin{bmatrix} \quad C \quad \end{bmatrix}}$$

Truncated SVD Produces Embeddings

- Each row of W matrix is a k-dimensional representation of each word *w*
- K might range from 50 to 1000
- Generally we keep the top k dimensions, but some experiments suggest that getting rid of the top 1 dimension or even the top 50 dimensions is helpful



(Lapesa and Evert, A Large Scale Evaluation of Distributional Semantic Models: Parameters, Interactions and Model Selection, 2014).

Embeddings vs Sparse Vectors

- Dense SVD embeddings sometimes work better than sparse PPMI matrices at tasks like word similarity
  - Denoising: low-order dimensions may represent unimportant information
  - Truncation may help the models generalize better to unseen data.
  - Having a smaller number of dimensions may make it easier for classifiers to properly weight the dimensions for the task.
  - Dense models may do better at capturing higher order co-occurrence.

Problems with SVD

- Computational cost scales quadratically for n x m matrix:
  $O(mn^2)$ when $n<m$

→Bad for millions of words or documents

- Hard to incorporate new words or documents

Outline

- Motivation
- Dense Vectors via SVD
- **Embeddings by neural language models**
- Evaluation
- Applications

Prediction-based Models

- An alternative way to get dense vectors
- Idea:
    - Instead of capturing co-occurrence counts directly, predict surrounding words of every word
    - Learn embeddings as part of the process of word prediction.
    - Both are quite similar (see “Glove: Global Vectors for Word Representation” by Pennington et al. (2014) and Levy and Goldberg (2014))

- Train a neural network to predict neighboring words
    - Inspired by **neural net language models**.
    - In so doing, learn dense embeddings for the words in the training corpus.

Prediction-based Models

- Advantages:
  - Fast, easy to train (much faster than SVD)
  - Can easily incorporate a new sentence/document or add a word to the vocabulary
  - Including sets of pretrained embeddings!

- Available models
  - **Skip-gram** (Mikolov et al. 2013a)
  - **CBOW** (Mikolov et al. 2013b)

- Available online in the `word2vec` package

Word2Vec

- Idea: **predict** rather than **count**
- Instead of **counting** how often each word *w* occurs near "*apricot*“
- Train a model for a binary **prediction** task:
  - Is *w* likely to show up near "*apricot*"?
- We don’t actually care about this task
  - But we'll take the learned weights as the word embeddings

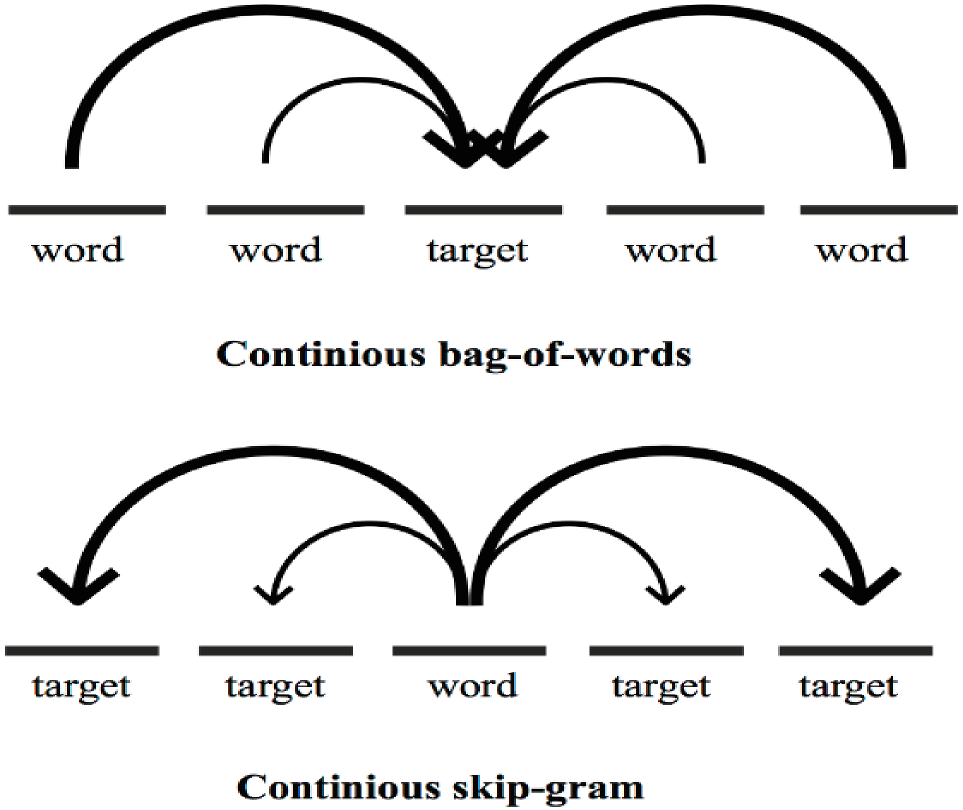
Outline

- Motivation
- Dense Vectors via SVD
- **Embedding by neural language models**
  - **Skip-grams and CBOW**
- Evaluation
- Applications

Training data!

- Brilliant insight: Use running text as implicitly supervised training data!
- A word *s* near *apricot*
  - Acts as gold 'correct answer' to the question
  - "Is word *w* likely to show up near *apricot*?"
- No need for hand-labeled supervision
- The idea comes from **neural language modeling**
  - Bengio et al. (2003)
  - Collobert et al. (2011)

Skip-gram vs CBOW



The Skip-gram Algorithm

- Predict each neighboring word
  - in a context window of $2C$ words
  - from the current word.

- So for C=2, we are given word w_t and predicting these 4 words:

$$\left[w_{t-2}, w_{t-1}, w_{t+1}, w_{t+2}\right]$$

- Training sentence:

... lemon, a tablespoon of **apricot** jam a pinch ...

c1 c2 target c3 c4

Skip-Gram Goal

- Given a tuple (t,c) = target, context
  - (*apricot*, *jam*)
  - (*apricot*, *data*)

- Return probability that c is a real context word:
  - $P(+|t,c)$

Setup

- Walking through corpus pointing at word w, whose index in the vocabulary is t, so we'll call it w_t $(1 < t < |V|)$.

- Let's predict w_{t+1}

- Hence our task is to compute $P(w_{t+1} | w_t)$.

Details of Skip-grams

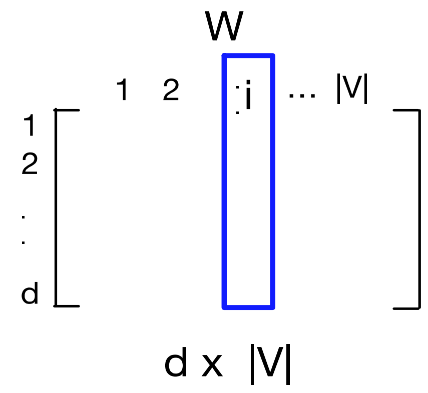
- Objective function: Maximize the log probability of any context word given the current center word

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

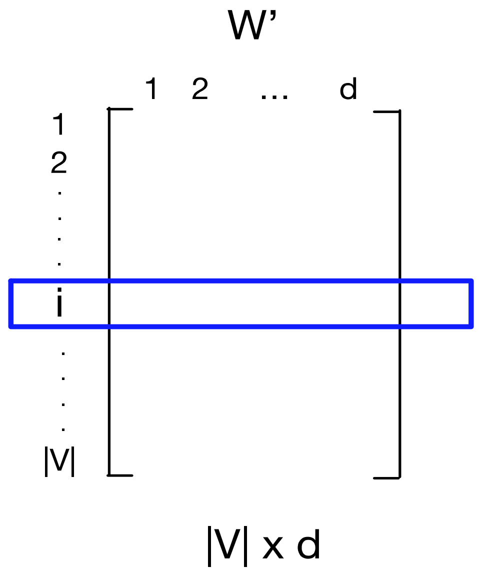
- Where θ represents all variables we optimize

Skip-grams Learn 2 Embeddings for Each w

- **Input embedding** v_i in the input matrix W
  - Column i of the input matrix W is the $1{\times}d$ embedding v_i for word i in the vocabulary.

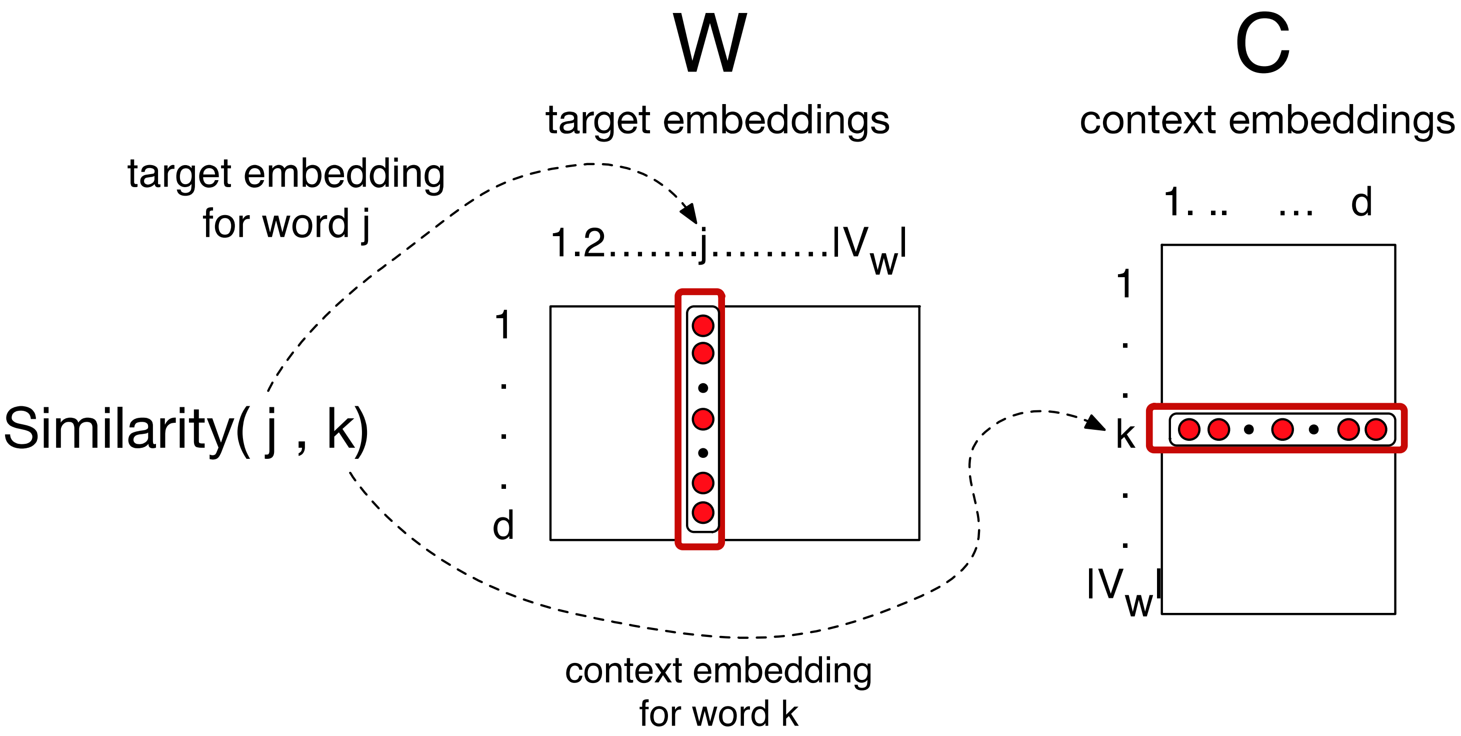


- **Output embedding** v'_i, in output matrix W'
  - Row i of the output matrix W' is a $d \times 1$ vector embedding v'_i for word i in the vocabulary.



Intuition

- Similarity as dot-product between a target vector and context vector



Similarity is computed from dot product

- Remember: two vectors are similar if they have a high dot product
  - Cosine is just a normalized dot product

- So:
  - Similarity(s,t) $\propto u_s \cdot v_t$

- We will need to normalize to get a probability

Turning dot products into probabilities

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

$$p(w_s|w_t) = \frac{\exp(u_s.v_t)}{\sum_{w\in|V|}\exp(u_w.v_t)}$$

Embeddings from W and W’

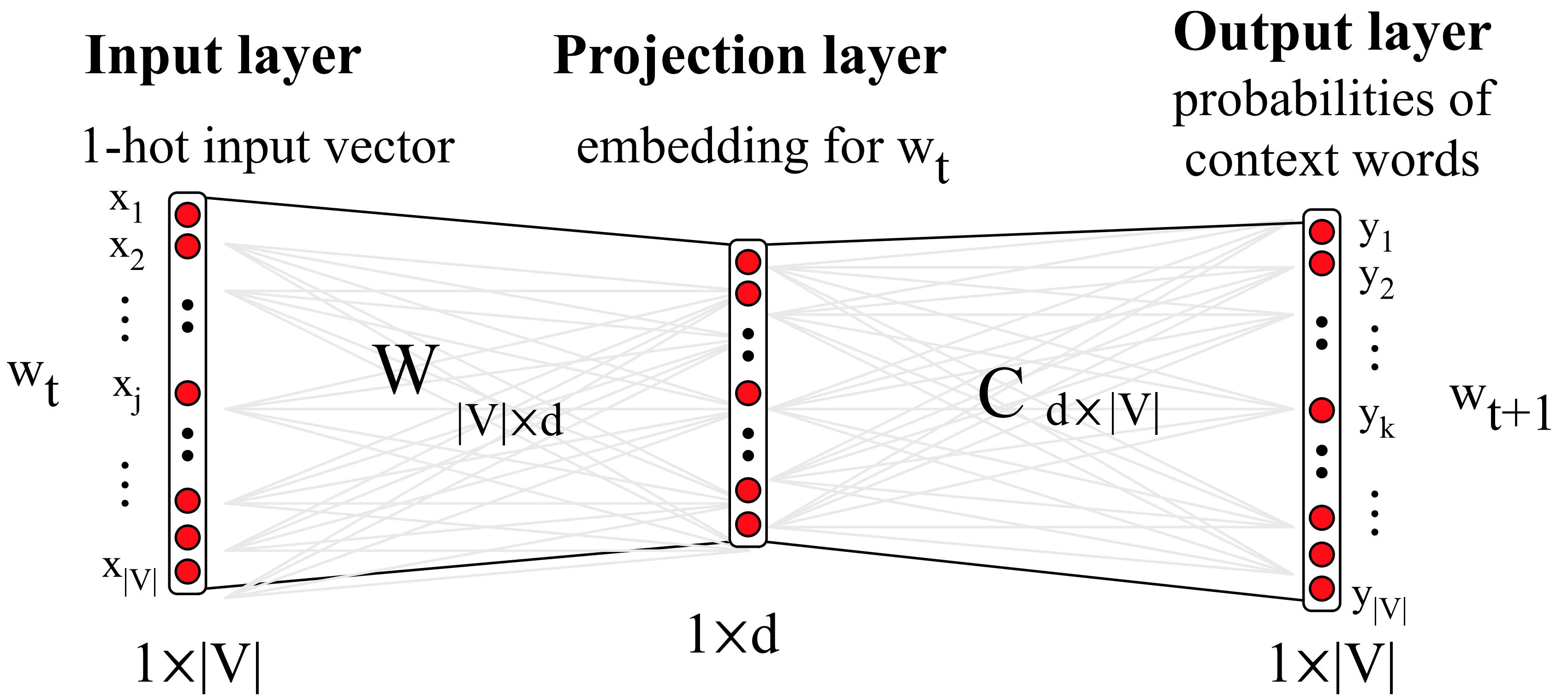
- Since we have two embeddings, v_t and u_t for each word w
- We can either:
  - Just use v_t
  - Sum them
  - Concatenate them to make a double-length embedding

Learning

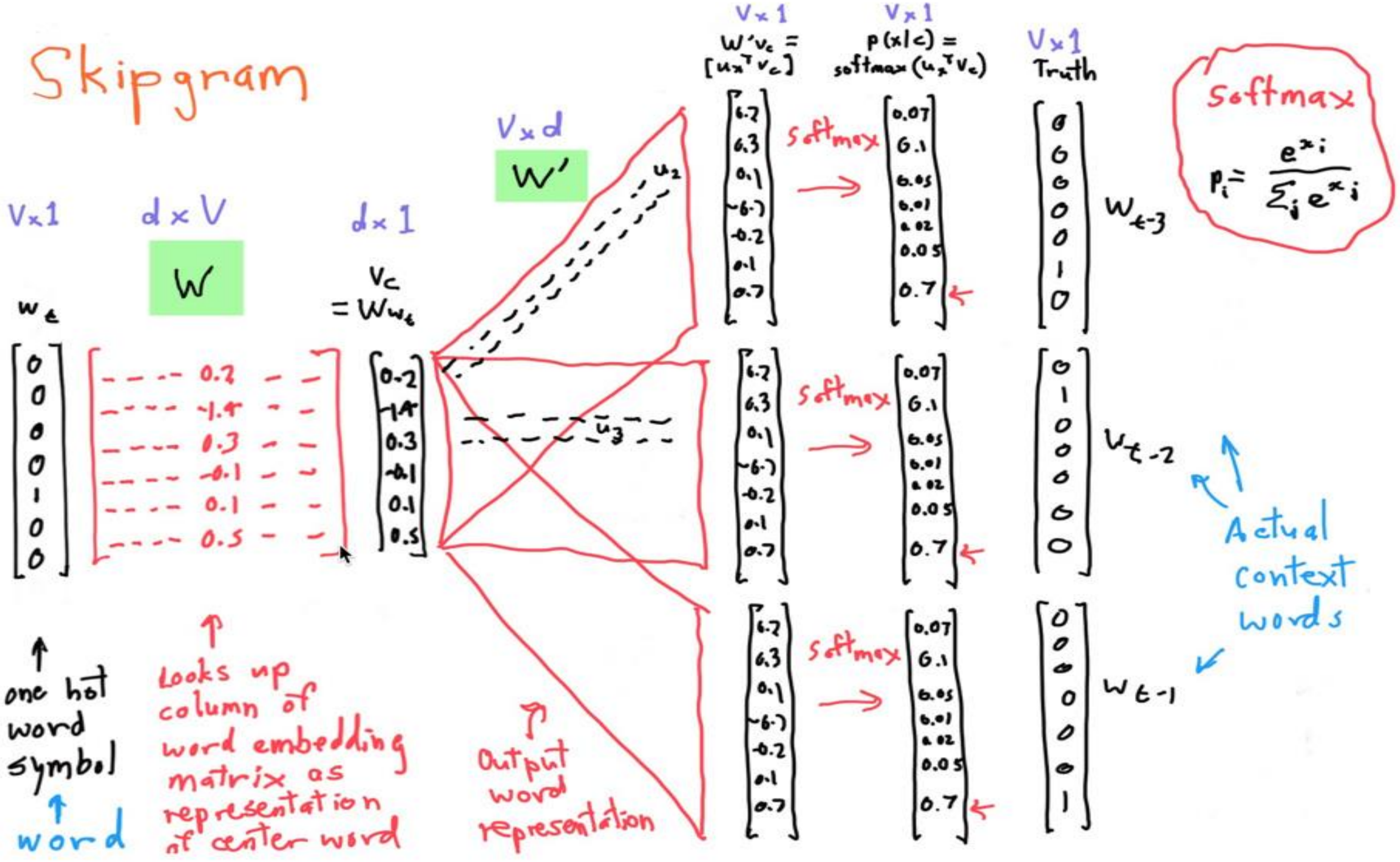
- Start with some initial embeddings (e.g., random)
- Iteratively make the embeddings for a word
  - more like the embeddings of its neighbors
  - less like the embeddings of other words.

Visualization

- Visualizing W and C as a network for doing error back propagation



Visualization



Problem with the Softamx

- The denominator: have to compute over every word in vocab

$$p(w_s|w_t) = \frac{\exp(u_s^T . v_t)}{\sum_{w \in |V|} \exp(u_w^T . v_t)}$$

- Instead: just sample a few of those negative words

Goal in learning

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

- Make the word like the context words

```
lemon,  a [tablespoon of apricot preserves or] jam
              c1          c2     w     c3        c4
```

- We want this to be high:

$$\sigma(c1 \cdot w) + \sigma(c2 \cdot w) + \sigma(c3 \cdot w) + \sigma(c4 \cdot w)$$

- And not like *k* randomly selected "noise words"

```
[cement metaphysical dear coaxial    apricot attendant whence forever puddle]
 n1     n2                n3   n4                n5         n6     n7         n8
```

- We want this to be low:

$$\sigma(n1 \cdot w) + \sigma(n2 \cdot w) + ... + \sigma(n8 \cdot w)$$

Skip-Gram Training

- Training sentence:

... lemon, a tablespoon of **apricot** jam a pinch ...

c1 c2 t c3 c4

positive examples +	
t	c
apricot	tablespoon
apricot	of
apricot	preserves
apricot	or

- For each positive example, we'll create *k* negative examples.
- Using *noise* words
- Any random word that isn't *t*

Skip-Gram Training

- Training sentence:

... lemon, a tablespoon of apricot jam a pinch ...

c1 c2 t c3 c4

positive examples +

t	c
apricot	tablespoon
apricot	of
apricot	preserves
apricot	or

negative examples -

t	c	t	c
apricot	aardvark	apricot	twelve
apricot	puddle	apricot	hello
apricot	where	apricot	dear
apricot	coaxial	apricot	forever

Objective Criteria

- Focusing on one target word t

$$
\begin{aligned}
L(\theta) &= \log P(+|t,c) + \sum_{i=1} \log P(-|t,n_i) \\
&= \log \sigma(c \cdot t) + \sum_{i=1}^{k} \log \sigma(-n_i \cdot t) \\
&= \log \frac{1}{1+e^{-c \cdot t}} + \sum_{i=1}^{k} \log \frac{1}{1+e^{n_i \cdot t}}
\end{aligned}
$$

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

Relation between Skip-grams and PMI!

- If we multiply WW'^T

- We get a |V|x|V| matrix M, each entry m_{ij} corresponding to some association between input word i and output word j

- Levy and Goldberg (2014b) show that skip-gram reaches its optimum just when this matrix is a shifted version of PMI:

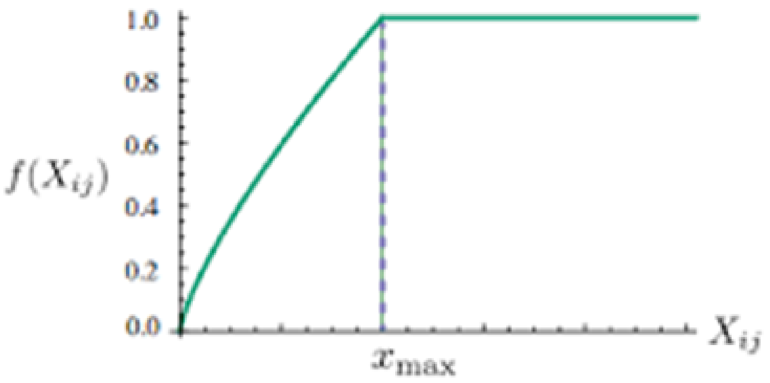
$$WW'^T = M^{\text{PMI}} - \log k$$

- So skip-gram is implicitly factoring a shifted version of the PMI matrix into the two embedding matrices.

GloVe: Combining the Best of Both Worlds

$$J(\theta) = \frac{1}{2}\sum_{i,j=1}^{W} f(P_{ij})\left(u_i^T v_j - \log P_{ij}\right)^2$$

- X_{ij}: the number of times word *j* occurs in the context of word *i*
- $X_i = \sum_k X_{ik}$: the number of times any word appears in the context of word *i*
- $P_{ij} = P(j|i) = {}^{X_{ij}}/_{X\_i}$: probability that word *j* appear in the context of word *i*



GloVe: Combining the Best of Both Worlds

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

Outline

- Motivation
- Dense Vectors via SVD
- Embeddings by neural language models
- **Evaluation**
- Applications

Properties of Embeddings

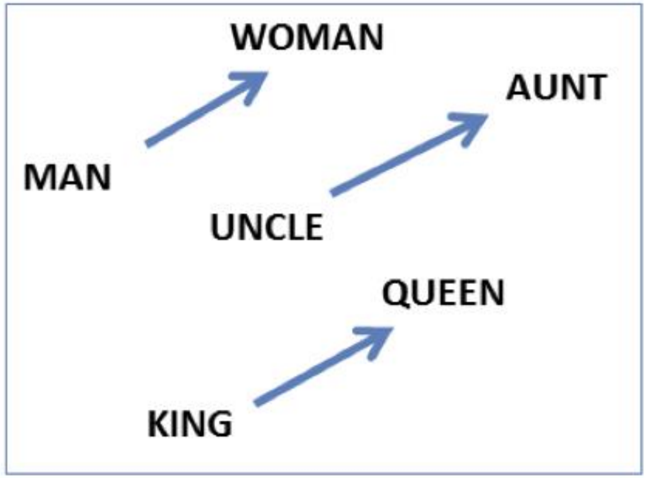
- Nearest words to some embeddings

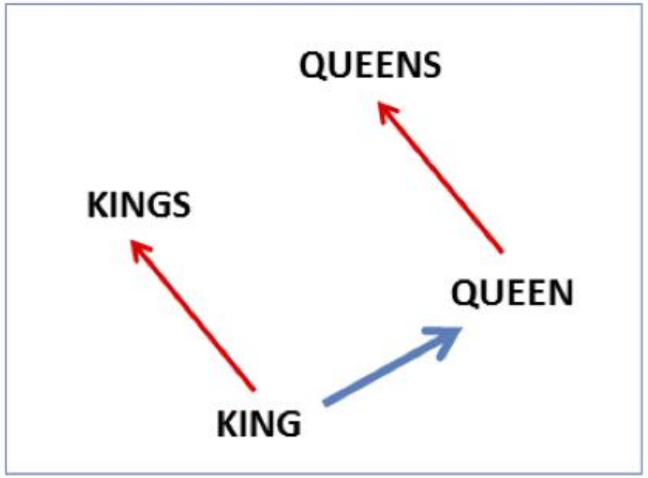
FRANCE	JESUS	XBOX	REDDISH	SCRATCHED	MEGABITS
AUSTRIA	GOD	AMIGA	GREENISH	NAILED	OCTETS
BELGIUM	SATI	PLAYSTATION	BLUISH	SMASHED	MB/S
GERMANY	CHRIST	MSX	PINKISH	PUNCHED	BIT/S
ITALY	SATAN	IPOD	PURPLISH	POPPED	BAUD
GREECE	KALI	SEGA	BROWNISH	CRIMPED	CARATS
SWEDEN	INDRA	PSNUMBER	GREYISH	SCRAPED	KBIT/S
NORWAY	VISHNU	HD	GRAYISH	SCREWED	MEGAHERTZ
EUROPE	ANANDA	DREAMCAST	WHITISH	SECTIONED	MEGAPIXELS
HUNGARY	PARVATI	GEFORCE	SILVERY	SLASHED	GBIT/S
SWITZERLAND	GRACE	CAPCOM	YELLOWISH	RIPPED	AMPERES

Embeddings Capture Relational Meaning!

vector(*'king'*) - vector(*'man'*) + vector(*'woman'*) ≈ vector('queen')

vector(*'Paris'*) - vector(*'France'*) + vector(*'Italy'*) ≈ vector('Rome')





Evaluating Embeddings

- Internal Evaluation
  - Word similarity
  - Word analogy

- External Evaluation

Evaluation based on Word Similarity

- Compare to human scores on word similarity-type tasks:
  - WordSim-353 (Finkelstein et al., 2002)
  - SimLex-999 (Hill et al., 2015)
  - Stanford Contextual Word Similarity (SCWS) dataset (Huang et al., 2012)
  - TOEFL dataset: *Levied is closest in meaning to: imposed, believed, requested, correlated*

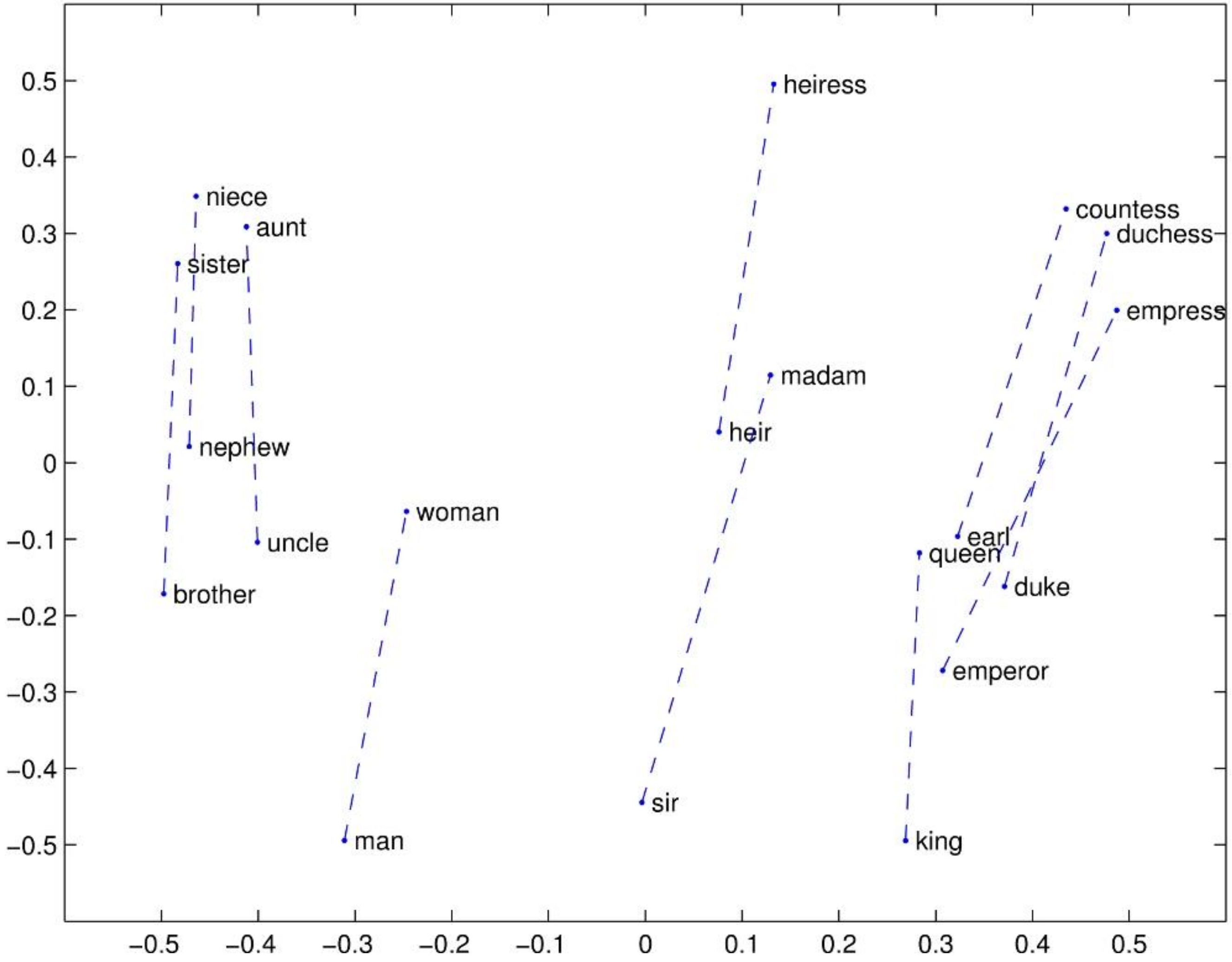
WordSim-353

- Rank word pairs based on gold data and vector representation outputs
- Calculate Spearman or Pearson correlation between two sets

computer	keyboard	7.62
planet	galaxy	8.11
OPEC	country	5.63
country	citizen	7.31
Maradona	football	8.62
money	bank	8.50
president	medal	3.00
peace	insurance	2.94
Mars	water	2.94
drink	ear	1.31
stock	jaguar	0.92
sugar	approach	0.88

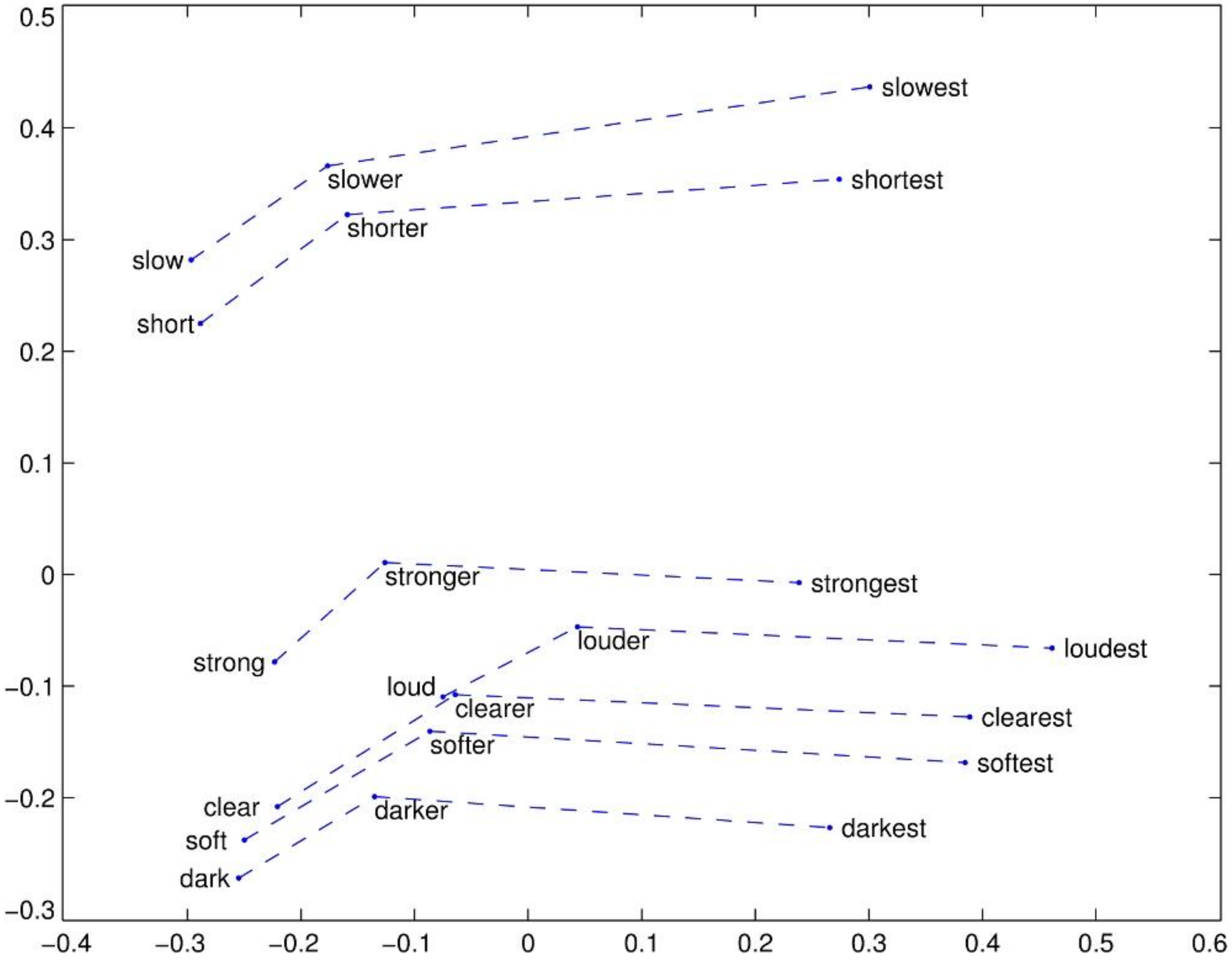
Evaluation based on Word Analogy

- Embeddings capture relational meaning!



Evaluation based on Word Analogy

- Embeddings capture relational meaning!



Google Analogy

- 19544 items in 14 categories

$$\sim v(D) = v(A) - v(B) + v(C)$$

Relation Type	Example
Capital common country	Baghdad:Iraq::Tehran:Iran
Capital world	Lusaka:Zambia::Tehran:Iran
Currency	Canada:dollar::Iran:rial
City in state	Miami:Florida::Irving:Texas
Family	boy:girl::she:he
Gram1 adjective to adverb	calm:calmly::rare:rarely
Gram2 opposite	aware:unaware::sure:unsure
Gram3 comparative	bad:worse::big:bigger
Gram4 superlative	wide:widest::bad:worst
Gram5 present participle	fed:feeding::fly:flying
Gram6 nationality adjective	India:Indian::England:English
Gram7 past tense	going:went::running:ran
Gram8 plural	bird:birds::cow:cows
Gram9 plural verb	eat:eats::walk:walks

External Evaluation

- Apply word embedding in any NLP application
- Compare the result with different embeddings

Properties of Embeddings

- Similarity depends on window size C

C = ±5 The nearest words to *Hogwarts:*

- *Dumbledore*
- *Malfoy*
- *Halfblood*

C = ±2 The nearest words to *Hogwarts:*

- *Sunnydale*
- *Evernight*

- Similarity depends on training data

Similar words to "کبک" when training on Wikipedia

- میسیسکوا
- لورانتید

Similar words to "کبک" when training on irBlog

- قرقاول
- تیهو

Outline

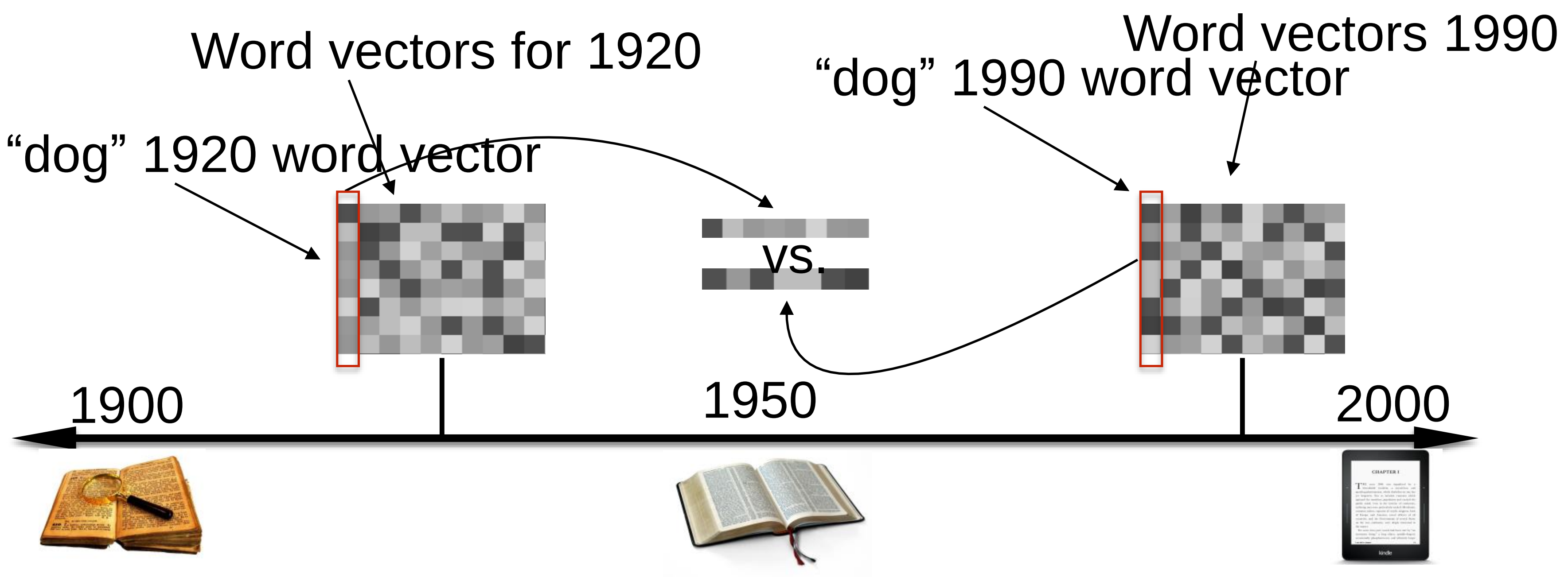
- Motivation
- Dense Vectors via SVD
- Embeddings by neural language models
- Evaluation
- **Applications**

Language Processing

- Replace sparse vectors with dense vector in any NLP task
  - Document classification
  - Document clustering
  - Sentiment analysis
  - POS tagging
  - Question answering
  - …

Study Culture and History

- Train embeddings on old books to study changes in word meaning!!



Study Culture and History

- “Paris : France :: Tokyo : x”
  - x = Japan
- “father : doctor :: mother : x”
  - x = nurse
- Psychological findings on US participants:
  - African-American names are associated with unpleasant words (more than European-American names)
  - Male names associated more with math, female names with arts
  - Old people's names with unpleasant words, young people with pleasant words.

Extensions of Word Embedding

- Subword-level embeddings
- Sense Embedding
- Embeddings for multiple languages
- OOV handling
- Phrases and multi-word expressions
- Task and domain-specific embeddings

Summary

- **Concepts** or word senses
  - Have a complex many-to-many association with **words** (homonymy, multiple senses)
  - Have relations with each other
    - Synonymy, Antonymy, Superordinate
  - But are hard to define formally (necessary & sufficient conditions)

- **Embeddings** = vector models of meaning
  - More fine-grained than just a string or index
  - Especially good at modeling similarity/analogy
    - Just download them and use cosines!!
  - Can use sparse models (tf-idf) or dense models (word2vec, GLoVE)
  - Useful in practice but know they encode cultural stereotypes

Further Reading

- Speech and Language Processing (3rd ed. draft)
  - Chapter 6