ML for Human Vision and Language

MSc Artificial Intelligence

Lecturer: Tejaswini Deoskar
t.deoskar@uu.nl

UiL-OTS, Utrecht University

Block 4, 2019

---

some slides by S. Goldwater, E. Shutova, C. Manning, and Jurafsky & Martin

Vector-based (distributional) lexical semantics

- Words as discrete symbols: hotel, motel
- 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. 50,000)]

- There is no natural notion of similarity for one-hot vectors! These two vectors are orthogonal.
- **Instead**: learn to encode similarity in the vectors themselves!

Distributional Hypothesis

"The degree of semantic similarity between two linguistic expressions A and B is a function of the similarity of the linguistic contexts in which A and B can appear." [Z. Harris (1954) *Distributional Structure*]

You shall know a word by the company it keeps (J. R. Firth, 1957)

Distributional semantics: family of techniques for representing word meaning based on (linguistic) contexts of use.

Distributional semantics

1. Count-based models:
   - a word is represented as a sparse and long vector
     - dimensions correspond to the words in the vocabulary (with vocabularies of 20,000 to 50,000)
     - interpretable

Distributional semantics

1. Count-based models:
   - a word is represented as a sparse and long vector
     - dimensions correspond to the words in the vocabulary (with vocabularies of 20,000 to 50,000)
     - interpretable
   - values in the vector are a function of the count of the word co-occurring with a context word
   - sparse; most values in the vector are zero (but dimensionality reduction techniques can be used to get dense vectors)

Distributional semantics

1. Count-based models:
   - a word is represented as a sparse and long vector
     - dimensions correspond to the words in the vocabulary (with vocabularies of 20,000 to 50,000)
     - interpretable
   - values in the vector are a function of the count of the word co-occurring with a context word
   - sparse; most values in the vector are zero (but dimensionality reduction techniques can be used to get dense vectors)
2. Predict-based models:
   - Train a (neural network) model to predict plausible contexts for a word
   - learn word representations in the process
   - **short dense** vectors with **latent** dimensions

Why short dense vectors

- Vectors that are short (length 50 -1000) and dense (most elements are non-zero)
- Machine Learning:
  - easier to include as features in machine learning systems
  - contain fewer parameters, hence less prone to overfitting, generalise better.
- Distributional semantics: Can we represent words in a smaller dimensional space that preserves the similarity relationships, results in meaningful abstract dimensions?

Today

- Predict-based models (Skipgram)
- Negative sampling in Skipgram
- GloVe

Count vs. Predict

- Instead of counting how often each word w occurs near e.g. "apricot"
- Train a classifier on 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 classifier weights as the word embeddings

Count vs. Predict

- Instead of counting how often each word w occurs near e.g. "apricot"
- Train a classifier on 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 classifier weights as the word embeddings

- A word c near "apricot"
  - Acts as gold 'correct answer' to the question
  - Is word w likely to show up near "apricot"

Count vs. Predict

- Instead of counting how often each word w occurs near e.g. "apricot"
- Train a classifier on 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 classifier weights as the word embeddings

- A word c near "apricot"
  - Acts as gold 'correct answer' to the question
  - Is word w likely to show up near "apricot"
- Use running text as implicitly supervised training data!
- No need for hand-labeled supervision

Prediction-based distributional models: Word2Vec

Word2Vec (Mikolov et al., 2013, Efficient Estimation of Word Representations in Vector Space)

- train a neural network to predict neighbouring words
- learn dense embeddings for the words in the training corpus in the process
- inspired by work on neural language models

- Consists of 2 models :
  - CBOW (continuous bag of words): predict a center word from the surrounding context
  - Skipgram : predict the distribution (probability) of context words from a center word

Context window

If context size $= 2$:

center word

$$w_{t-2} \quad w_{t-1} \quad w_t \quad w_{t+1} \quad w_{t+2}$$

More generally:

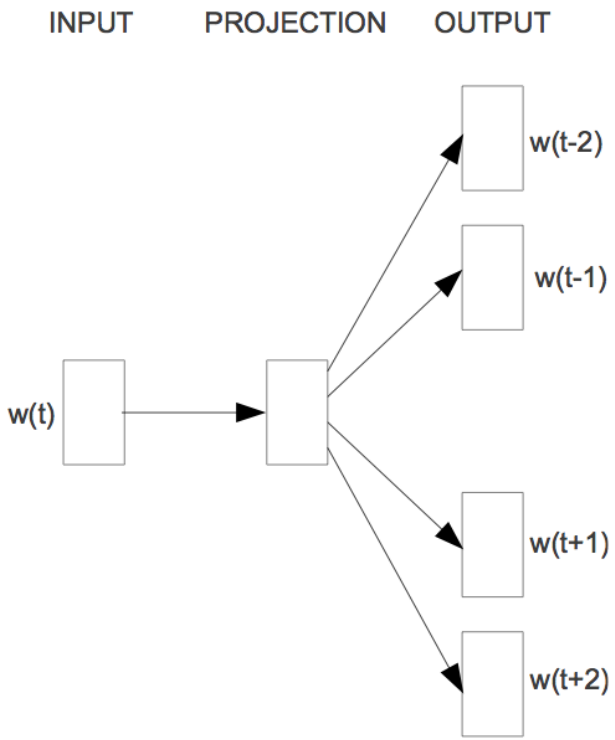
center word

$$w_{t-m} \quad \ldots \quad w_{t-1} \quad w_t \quad w_{t+1} \quad \ldots \quad w_{t+m}$$

Skip-gram

Given a word w_t :

- Predict each neighbouring word in a context window of $2L$ words
- For $L = 2$, we predict 4 neighbouring words : $[w_{t-2},\ w_{t-1},\ w_{t+1},\ w_{t+2}]$



Slide credit: Tomas Mikolov

Word2vec: Overview

- We have a large corpus; every word in a fixed vocabulary is represented by a vector
- Go through each position t in the text, which has a center word w and context words c

Word2vec: Overview

- We have a large corpus; every word in a fixed vocabulary is represented by a vector
- Go through each position t in the text, which has a center word w and context words c
- Use the similarity of the word vectors for w and c to calculate the probability of c given w (or vice versa), i.e $p(w|c)$ or $p(c|w)$
- Keep adjusting the word vectors to maximize this probability

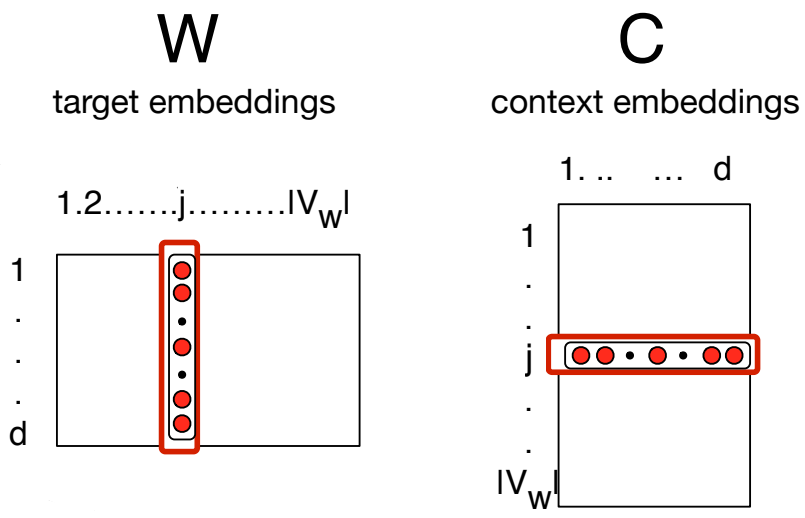
Two vectors (embeddings) per word

- We learn two vectors, for each word w
  - when the word is in the center : **word**/ **target** vector v (or input vector v)
  - when the word is in the context : **context** vector c (or output vector u)

Skip-gram: parameter matrices

For each word $w_j \in V_w$:

- word vector v, in word matrix W :
- context vector c, in context matrix C



Slide credit: Dan Jurafsky

Notation (J&M): Skip gram

- w_j : word j from vocabulary V_w
- $\mathcal{W} \in \mathbb{R}^{n \times |V_w|}$: Input word matrix
- $\mathcal{C} \in \mathbb{R}^{|V_w| \times n}$: Output word matrix
- n is dimension of embedding
- v_j : j^{th} column of $\mathcal{W}$, i.e. target embedding of word w_j (input word representation)
- c_j : j^{th} row of $\mathcal{C}$, i.e. context embedding of word w_j (output word representation)

Skipgram- setup

- Walk through the corpus pointing at word $w(t)$, whose index in the vocabulary is j — we will call it w_j
- We want to predict $w(t+1)$, whose index in the vocabulary is k — we will call it w_k
- to do this, we need to compute $p(w_k \mid w_j)$

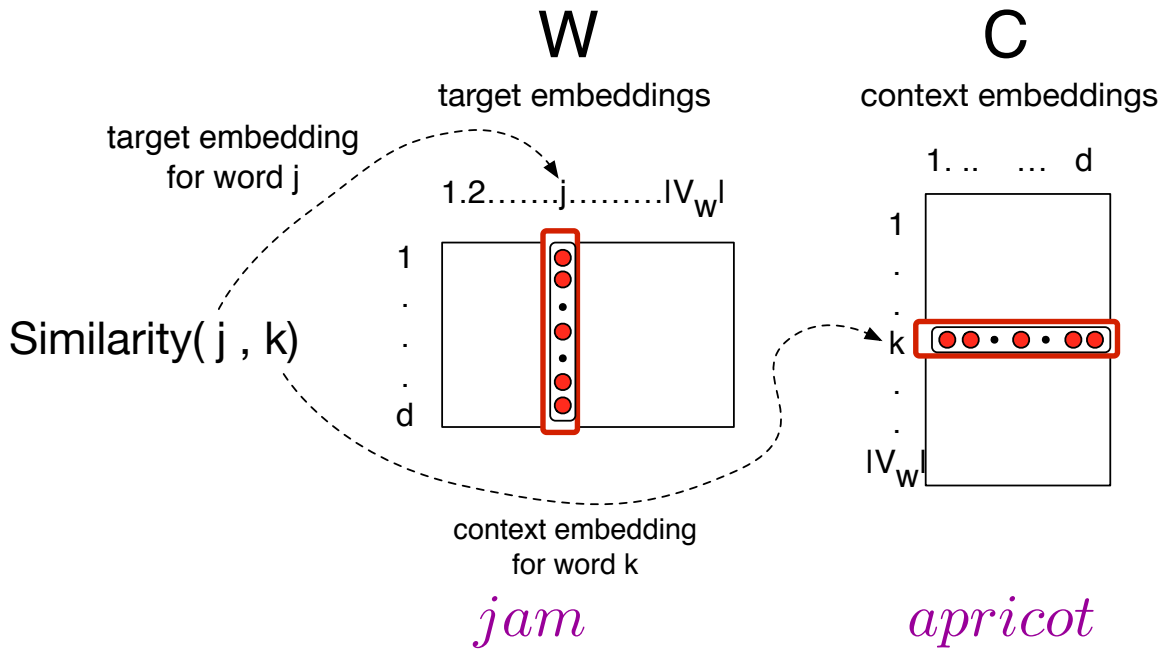
Skipgram- setup

- Walk through the corpus pointing at word $w(t)$, whose index in the vocabulary is j — we will call it w_j
- We want to predict $w(t+1)$, whose index in the vocabulary is k — we will call it w_k
- to do this, we need to compute $p(w_k \mid w_j)$
- Intuition behind skip gram : to compute this probability, we need to compute similarity between w_j and w_k

Skip-gram: Computing similarity

Similar vectors have a high dot product : $sim(c_k, v_j) \propto c_k \cdot v_j$

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



apricot jam

diagram: Dan Jurafsky

Skip-gram: Computing probability

- Dot product $c_k \cdot v_j$ is a number ranging from -inf. to +inf
- Use a **softmax** function to normalise the dot product into probabilities

$$p(w_k \mid w_j) = \frac{e^{c_k \cdot v_j}}{\sum_{c_i \in V} e^{c_i \cdot v_j}}$$

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

Skip-gram: Objective

Maximising the probability of the corpus (data likelihood)

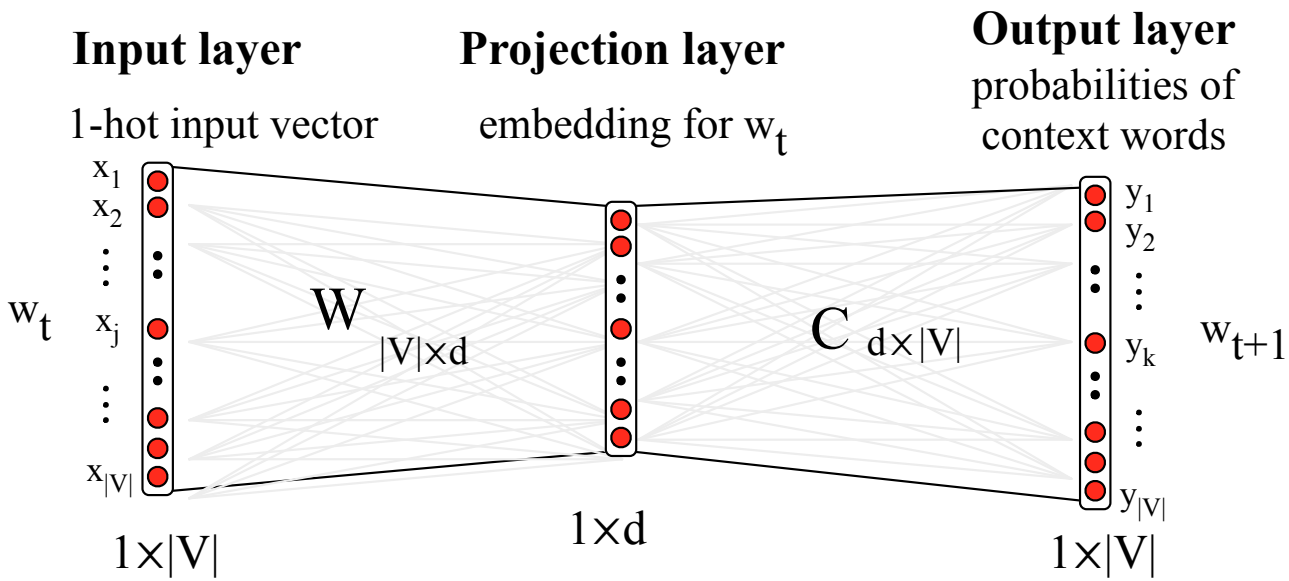
- For each position $t = 1, \ldots T$, predict context words within a window

$$\text{Likelihood} = L(\theta) = \prod_{t=1}^{T} \prod_{m \leq j \leq m} P(w_{t+j} \mid w_t; \theta)$$

- Objective function $J(\theta)$ is the average negative log likelihood

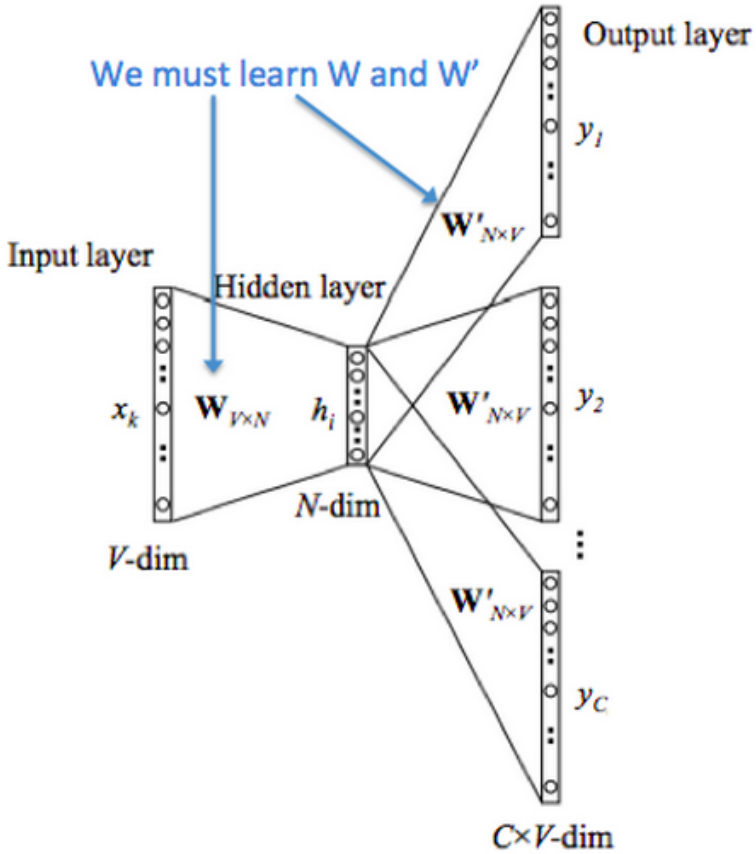
$$J(\theta) = -\frac{1}{T} \log L(\theta) = -\frac{1}{T} \sum_{t=1}^{T} \sum_{m \leq j \leq m} \log P(w_{t+j} \mid w_t; \theta)$$

Visualising skip-gram as a network

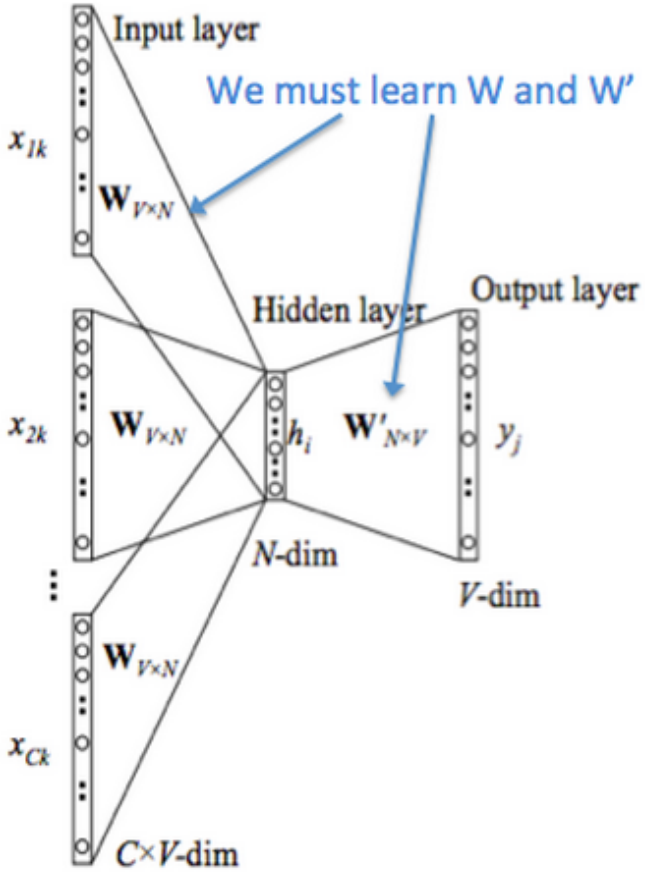


Slide credit: Dan Jurafsky

Network schematic: Skip-gram



Network schematic: CBOW



Skip-gram with negative sampling

- Problem with softmax : expensive to compute the denominator for the whole vocabulary

$$p(w_k \mid w_j) = \frac{e^{c_k \cdot v_j}}{\sum_{c_i \in V} e^{c_i \cdot v_j}}$$

- Solution: approximate the denominator by **negative sampling**
  - at training time, walk through the corpus
  - for each target word and positive context
  - sample k noise samples (negative samples), i.e. other words
  - Goal: to move the embeddings towards the positive context words, and away from the noise words

Skip-gram with negative sampling

Objective in training:

- Make the word like the context words
  lemon, a [tablespoon of apricot preserves or] jam.

  c_1 c_2 w c_3 c_4
- And not like the k negative examples

[cement idle dear coaxial apricot attendant whence forever puddle]

n_1 n_2 n_3 n_4 w n_5 n_6 n_7 n_8

Skip-gram with negative sampling: training data

Convert the dataset into word pairs:

- **Positive (+)**

  (apricot, tablespoon)
  (apricot, of)
  (apricot, jam)
  (apricot, or)

- **Negative (-)**

  (apricot, cement)
  (apricot, idle)
  (apricot, attendant)
  (apricot, dear)
  ...

Skip-gram with negative sampling

- instead of treating it as a **multi-class problem** (and returning a probability distribution over the whole vocabulary)
- **return a probability** that word w_k is a valid context for word w_j

$$P(+|w_j, w_k)$$

$$P(-|w_j, w_k) = 1 - P(+|w_j, w_k)$$

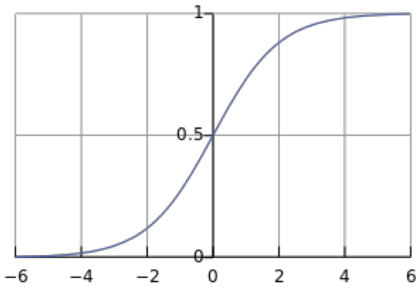
Skip-gram with negative sampling

- model similarity as a dot product

$$sim(c_k, v_j) \propto c_k \cdot v_j$$

- turn this into a probability using the **sigmoid function**

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



$$P(+|w_j, w_k) = \frac{1}{1 + e^{-c_k \cdot v_j}}$$

$$P(-|w_j, w_k) = 1 - P(+|w_j, w_k) = 1 - \frac{1}{1 + e^{-c_k \cdot v_j}} = \frac{1}{1 + e^{c_k \cdot v_j}}$$

Skip-gram with negative sampling

New objective function:

- maximise the probability of a word and context being in the corpus data if it indeed is
- maximise the probability of a word and context *not* being in the corpus data if it indeed is not

$$\theta = \underset{\theta}{\text{argmax}} \prod_{(w,c)\in D} P(D=1|w,c,\theta) \prod_{(w,c)\in \tilde{D}} P(D=0|w,c,\theta)$$

- Maximising the likelihood $=$ minimising the negative log likelihood

$$J = -\sum_{(w,c)\in D} \log \frac{1}{1+\exp(-u_w^T v_c)} - \sum_{(w,c)\in \tilde{D}} \log\left(\frac{1}{1+\exp(u_w^T v_c)}\right)$$

Skip-gram with negative sampling: new objective function

$$
\begin{aligned}
\theta &= \underset{\theta}{\text{argmax}} \prod_{(w,c)\in D} P(D=1|w,c,\theta) \prod_{(w,c)\in \tilde{D}} P(D=0|w,c,\theta) \\
&= \underset{\theta}{\text{argmax}} \prod_{(w,c)\in D} P(D=1|w,c,\theta) \prod_{(w,c)\in \tilde{D}} (1-P(D=1|w,c,\theta)) \\
&= \underset{\theta}{\text{argmax}} \sum_{(w,c)\in D} \log P(D=1|w,c,\theta) + \sum_{(w,c)\in \tilde{D}} \log(1-P(D=1|w,c,\theta)) \\
&= \underset{\theta}{\text{argmax}} \sum_{(w,c)\in D} \log \frac{1}{1+\exp(-u_w^T v_c)} + \sum_{(w,c)\in \tilde{D}} \log(1-\frac{1}{1+\exp(-u_w^T v_c)}) \\
&= \underset{\theta}{\text{argmax}} \sum_{(w,c)\in D} \log \frac{1}{1+\exp(-u_w^T v_c)} + \sum_{(w,c)\in \tilde{D}} \log(\frac{1}{1+\exp(u_w^T v_c)})
\end{aligned}
$$

Word embeddings in practise

Word2vec is often used for pretraining in other tasks.

- It will help your models start from an **informed** position
- Requires only **plain text** - which we have a lot of
- Is very **fast** and easy to use
- Already **pretrained** vectors also available (trained on 100B words)

However, for best performance it is important to continue training, fine-tuning the embeddings for a specific task.

GloVe: Global Vectors for Word Representation

[Pennington, Socher, and Manning, 2014]

Two main classes of methods for word embeddings

- Count-based + matrix factorization (e.g. SVD) to get dense embeddings
  - capture global statistical information
  - capture word similarity, but poor on tasks like word-analogy
- Shallow window-based, learn by making predictions in local context windows
  - fail to make use of global statistics
  - capture complex linguistic patterns beyond similarity

GloVe: Trains on global word co-occurrence counts

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

- Very fast training
- Scalable to huge corpora, but good performance even with small corpus and small vectors
- State-of-the-art results on many semantic tasks (esp. analogy)

Encoding meaning in vector differences [Pennington, Socher, and Manning, 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, 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

Ratio is better able to distinguish relevant words from irrelevant ones, as compared to raw probabilities!

Encoding meaning in vector differences [Pennington, Socher, and Manning, 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 x 10^{-4}	6.6 x 10^{-5}	3.0 x 10^{-3}	1.7 x 10^{-5}
$P(x\|\text{steam})$	2.2 x 10^{-5}	7.8 x 10^{-4}	2.2 x 10^{-3}	1.8 x 10^{-5}
$\frac{P(x\|\text{ice})}{P(x\|\text{steam})}$	8.9	8.5 x 10^{-2}	1.36	0.96

Co-occurrence matrix

- X : Co-occurrence matrix
- X_{ij} : the number of times word j occurs in the context of word i

Co-occurrence matrix

- X : Co-occurrence matrix
- X_{ij} : the number of times word j occurs in the context of word i
- X_i : number of times a word k appears in the context of word i

$$X_i = \sum_k X_{ik}$$

Co-occurrence matrix

- X : Co-occurrence matrix
- X_{ij} : the number of times word j occurs in the context of word i
- X_i : number of times a word k appears in the context of word i

$$X_i = \sum_k X_{ik}$$

- Probability of j appearing in the context of i

$$P_{ij} = P(w_j \mid w_i) = \frac{X_{ij}}{X_i}$$

- Populating this matrix needs a single pass through the corpus to collect the statistics: can be large, but is a one-time cost.

Encoding meaning in vector differences

- Starting point of learning word representations: ratio of co-occurrence probabilities

$$F(w_i, w_j, w_k) = \frac{P_{ik}}{P_{jk}}$$

- How can we capture ratios of co-occurrence probabilities as linear meaning components in a word vector space?

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

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

$$F(w_i - w_j \ , \ w_k) = \frac{P_{ik}}{P_{jk}}$$

GloVe

- Fast training
- Scalable to huge corpora
- Good performance even with small corpus and small vectors
- State-of-the-art results on similarity, analogy and other relations.

Glove Results

Nearest words to frog:

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

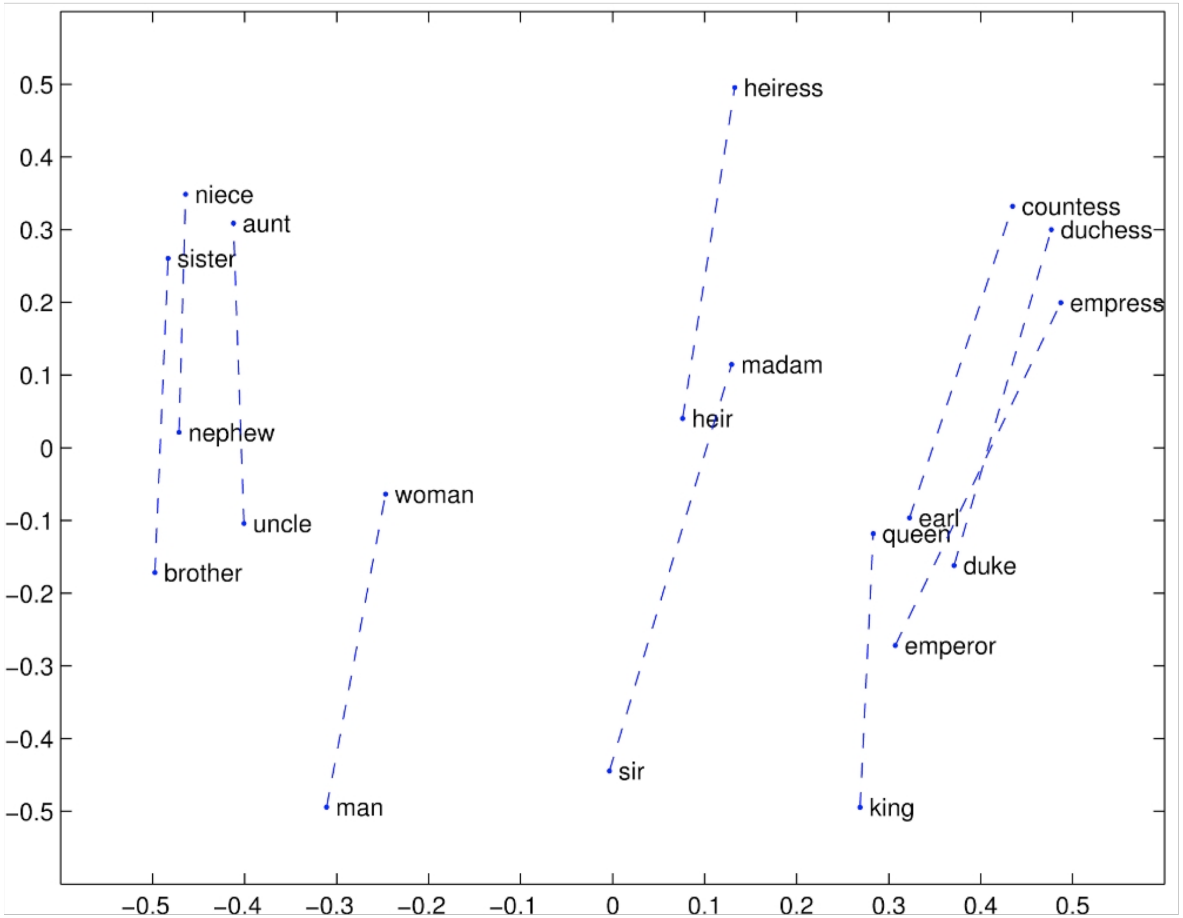
litoria

leptodactylidae

rana

eleutherodactylus

Glove Visualizations



Glove Visualizations : Superlatives