Natural Language Processing

Representing Text with Vectors

Junyeong Kim
Dept. of AI, Chung-Ang University

Notation

- We assume:
  - A token is the basic unit of discrete data, defined to be an item from a vocabulary indexed by 1, …, V.
  - A document is a sequence of N words denoted by $d = (w_1, w_2, \dots, w_N)$, where w_N is the the N-th word in the sequence.
  - A corpus is a collection of M documents denoted by $D = (d_1, d_2, \dots, d_M)$

Notation

- We assume:
  - A token is the basic unit of discrete data, defined to be an item from a vocabulary indexed by 1, …, V.
  - A document is a sequence of N words denoted by $d = (w_1, w_2, \dots, w_N)$, where w_N is the the N-th word in the sequence.
  - A corpus is a collection of M documents denoted by $D = (d_1, d_2, \dots, d_M)$
- In this lecture, a token will be **a word**

What is a word?

- There are many ways to define a word based on what aspect of language we consider (typography, syntax, semantics…)

- Definition (Semantic):
  - Words are the smallest linguistic expressions that are conventionally associated with a non-compositional meaning and can be articulated in isolation to convey semantic content.

Objective

- Given a vocabulary $w_1, \dots, w_V$ and a corpus D, our goal is to associate each word with some representation.

- What do we want from this representation?

  - Identify a word (bijection)
  - Capture the similarities of words (based on morphology, syntax, semantics, …)
  - Help us solve downstream tasks

- Vector-based representations of text are called embedding

One-hot Embedding

- Traditional way to represent words as atomic symbols with a unique integer associated with each word:
  - {1 = Movie, 2 = Hotel, 3 = Apple, 4 = Movies, 5 = art}
- Equivalent to represent words as one-hot vectors:
  - Movie = [1, 0, 0, 0, 0]
  - Hotel = [0, 1, 0, 0, 0]
  - …
  - Art = [0, 0, 0, 0, 1]

One-hot Encoding

- Most basic representation of any textual unit of NLP. Always start with it.

- Implicit assumption: word vectors are an orthonormal basis
  - Orthogonal
  - Normalized

- Problem 1 : Not very informative

  → Weird to consider “movie” and “movies” as independent entities or to consider all words equidistant: $||house - home|| = ||house - car||$

- Problem 2 : Polysemy

  → Should the Mouse of a computer get the same vector of the mouse animal?

Hand-Crafted Feature Representation

- Example of potential features:
  - Morphology: prefix, suffix, stem…
  - Grammar: Part of speech, gender, number, …
  - Shape: Capitalization, digit, hyphen

- Those features can be defined based on relations to other words
  - Synonyms of …
  - Hypernyms of …
  - Antonyms of …

- We present one popular hand-crafted semantically based representation of words → WordNet

WordNet

- Definition: a (word) sense is a discrete representation of one aspect of the meaning of a word

- WordNet is a large lexical database of word senses for English and other languages

WordNet

- Word types are grouped into (cognitive) synonym sets: **synsets**
  - S09293800 = {Earth, earth, world, globe}
- Polysemous words: assigned to different synsets
  - S14867162 = {earth, ground}
- Contains explanations for synsets:
  - The 3rd planet from the sun; the planet we live on
- Noun/verb synsets: organized in hierarchy, capturing IS-A relation
  - apple IS-A fruit

WordNet

- X is a hyponym of Y if X is an instance of Y:
  - Cat is a hyponym of animal
- X is a hypernym of Y if Y is an instance of X:
  - Animal is a hypernym of cat
- X and Y are co-hyponyms if they have the same hypernym:
  - Cat and dog are co-hyponyms
- X is a meronym of Y if X is a part of Y:
  - Wheel is a meronym of car
- X is a holonym of Y if Y is a part of X:
  - Car is a holonym of wheel

WordNet

- Similarity

$$\mathrm{sim}(S_1,\ S_2) = \frac{1}{\mathrm{length}(\mathrm{path}(S_1,\ S_2))}$$

- Idea: The shorter the hypernym/hyponym path from one synset to another the higher is the similarity

- Similarity between words

$$\mathrm{sim}(w_1,\ w_2) = \max_{\substack{S_1, S_2 \\ w_1 \in S_1 \\ w_2 \in S_2}} \mathrm{sim}(S_1,\ S_2)$$

Hand-crafted Representations: Limits

- Requires a lot of human annotations
- Subjectivity of the annotators
- Does not adapt to new words (languages are not stationary!):
  - Mocktail, Guac, Fave, Biohacking were added to the Merriam-Webster Dictionary in 2018

→ It does not scale easily to new languages, new concepts, new words…

How to Infer "Good" Representation with Data?

- Distributional Hypothesis
  - You shall know a word by the company it keeps – Firth (1957)
  - Idea: Model the context of a word to build its vectorial representation

Example: What is the meaning of “Bardiwac”?

- He handed her a glass of **bardiwac**.
- Beef dishes are made to complement the **bardiwacs**.
- Nigel staggered to his feet, face flushed from too much **bardiwac**.
- Malbec, one of the lesser-known **bardiwac** grapes, responds well to Australia’s sunshine.
- I denied off bread and cheese and this excellent **bardiwac**
- The drinks were delicious: blood-red **bardiwac** as well as light, sweet Rhenish.

→ **Bardiwac** is a heavy red alcoholic beverage made from grapes

Distributional word representation in a nutshell

1. Define what is **the context** of a word

2. Count how many times each target word occurs in this context

3. Build vectors out of (a function of) these context occurrence counts

$$x_w = f(w, Context(w))$$

How to define "**the context**" of a word?

- It can be defined as
  - The surrounding words (left and right words)
  - All the other words of the sentence / the paragraph
  - All the words after preprocessing and filtering-out some words

How to Model the Context to get

$$x_w = f(w, Context(w))$$

- Approach 1: Count-Based
  1. Measure frequency of words in the context for each word in the vocabulary
  2. Define vector representations based on those frequency
- Approach 2: Prediction-Based

Counting the Occurrences of the words in the context of **dog**

The dog barked in the park.
The owner of the dog put him on the leash since he barked.

co-occurence	# dog
barked	++
park	+
owner	+
leash	+

Co-Occurrence Matrix

	leash	walk	run	owner	pet	barked
dog	3	5	2	5	3	2
cat	0	3	3	2	3	0
lion	0	3	2	0	1	0
light	0	0	0	0	0	0
bark	1	0	0	2	1	0
car	0	0	1	3	0	0

Define vector representation based on the Co-Occurrence

	leash	walk	run	owner	pet	barked	the
dog	3	5	2	5	3	2	8
lion	0	3	2	0	1	0	6
light	0	0	0	0	0	0	5
bark	1	0	0	2	1	0	0
car	0	0	1	3	0	0	3

- Naïve Approach: Take the row of the co-occurrence matrix

Define vector representation based on the Co-Occurrence

	leash	walk	run	owner	pet	barked	the
dog	3	5	2	5	3	2	8
lion	0	3	2	0	1	0	6
light	0	0	0	0	0	0	5
bark	1	0	0	2	1	0	0
car	0	0	1	3	0	0	3

- Limits:
  - Representations depends on the size of the corpus
  - Frequent words impacts a lot the representations
  - Representations very sensitive to change in very infrequence words

Solution: Pointwise Mutual Information (PMI)

- Idea: Instead of absolute co-occurrence statistics, use probability (relative) of co-occurrences

$$PMI(w_1, w_2) = log \frac{P(w_1, w_2)}{P(w_1)P(w_2)}$$

- Intuition
  - The more dependent dog and cat the closer P(dog, cat) is from P(dog)P(cat), the larger the PMI

Solution: Pointwise Mutual Information (PMI)

- Idea: Instead of absolute co-occurrence statistics, use probability (relative) of co-occurrences

$$PMI(w_1, w_2) = log \frac{P(w_1, w_2)}{P(w_1)P(w_2)}$$

- Intuition
  - The more dependent dog and cat the closer P(dog, cat) is from P(dog)P(cat), the larger the PMI

$$PMI(w_1, w_2) = log \frac{\frac{1}{n_{pairs}}\#\{(w_1, w_2)\}}{\frac{1}{n_{word}}\#\{w_1\}\frac{1}{n_{word}}\#\{w_2\})}$$

Pointwise Mutual Information (PMI)

	leash	walk	run	owner	pet	barked	the
dog	2.75	2.24	3.16	2.24	2.75	3.16	1.77
lion	0	2.75	3.16	0	3.85	0	2.06
car	0	0	3.85	2.75	0	0	2.75

- Word embedding vectors are the row of the PMI matrix
  - We usually take the Positive PMI (assigned to 0 when negative) + Smooth unobserved pairs (Laplace smoothing: add 1)
  - Does not depend on size of the corpus (PMI is **normalized**)
  - Much less sensitive to change in frequent words (**log**)

Pointwise Mutual Information (PMI)

- Limits
  - Very large matrix $O(V^2)$! Very large word vectors
  - Hard to use large vectors in practice (i.e., 1M word vocabulary)
  - Cannot compare word vectors estimated on two different corpora unless they have exactly the same vocabulary!

- Idea: Build vectors with predefined size based on the PMI matrix

  → Dimensionality Reduction Technique

Singular Value Decomposition (SVD)

- We can decompose the PMI matrix with SVD
  1. We build a symmetric definite matrix based on the PPMI
  2. We decompose it using SVD algorithm

$$P = \mathbf{U}_d \Sigma_d \mathbf{V}_d^T$$

  3. U is of size (V, d) give us the representation of each word in a latent/embedding space

- Properties of SVD:
  - U is a orthonormal matrix
  - U aggregates the highest variance of the original word embedding

Limits of Dimensionality Reduction Approach

- Need to store a matrix of size $O(V^2)$
- SVD is $O(V * d^2)$

→ It is inefficient to build a very large matrix for reducing:
  Can we do both simultaneously?

- Solution: Prediction-based word embedding approaches

Prediction-based Model

- Idea
  - Learn directly dense word vectors
  - Using the distributional hypothesis
  - Implicitly, by parameterizing words as dense vectors and learning to predict context using this parametrization
- Many word embedding methods use these ideas successfully
- We present the **word2vec skip-gram model** (one of the most popular)

Word2Vec Skip-Gram model

- For each Sentence
  1. Sample a target word
  2. Predict context words defined as words in a fixed window from the target word

my dog is barking and chasing its tail

Word2Vec Skip-Gram model

- For each Sentence
  1. Sample a target word
  2. Predict context words defined as words in a fixed window from the target word

Word2Vec Skip-Gram model

- Given $d \in \mathbb{N}$, let $W \in \mathbb{R}^{(V,d)}$ and $C \in \mathbb{R}^{(V,d)}$ two word representations (or word embedding) matrices. For each sequence $(w_1, \cdots, w_T)$:
  - Pick a focus word w, associated to the vector $w \in \mathbb{R}^d$ (vector w is the row associated to word w in W)
  - Pick a context word c, associated to the vector $c \in \mathbb{R}^d$ (vector c is the row associated to word c in C)
  - Maximize $\max\limits_{W \in \mathbb{R}^{(V,d)}, C \in \mathbb{R}^{(V,d)}} \log p(c|w)$ (maximum likelihood estimator)

Word2Vec Skip-Gram Model

1. How to define $\log p(c|w)$?

2. How to optimize $\log p(c|w)$?

Word2Vec Skip-Gram Model

1. How to define $\log p(c|w)$?

2. How to optimize $\log p(c|w)$?

- Intuition
  - This is a classification problem
  - The labels we want to predict are the context words
  - Classification with a very large number of labels (V ~ 100K)

- Ideas:
  - → **Softmax**
  - → Simplify the softmax with **Negative Sampling** for Efficienty

Word2Vec Skip-Gram Model

- Softmax of dot-products of context vs. focus word vectors

$$p(c|w) = \frac{e^{w \cdot c}}{\sum_v e^{w \cdot v}}$$

- We compute the log-likelihood, our object function, as:

$$\log p(c|w) = w \cdot c \; - \log \sum_v e^{w \cdot v}$$

- Limits: O(V) to compute the loss (at every iteration)
  → Negative Sampling

Word2Vec Skip-Gram Model: Negative Sampling

- Idea: Instead of computing the probability objective over the entire vocabulary (all the V-1 non-context words)
  - → We sample K words that are not in the context of w, $v \in N_K \;\; (K \ll V)$

  - New objective function:

$$\sigma(w,c) + \frac{1}{K}\sum_{v \in N_K} \log \sigma(-w,v) \;\; with \; \sigma(x,y) = \frac{1}{1+e^{-x \cdot y}}$$

  - Complexity?

  → O(K) to compute with K independent of V

Word2Vec Model: Optimization

Algorithm 1 Skip-Gram Word2vec Training

Given a corpus C, made of a set of unique tokens V. Hyperparameters: number of negative samples K, a window size l, dimension of word vectors d, learning rate (α_t)

Initalize Randomly: $\mathbf{W} \in \mathbb{R}^{(V,d)}$ and $\mathbf{C} \in \mathbb{R}^{(V,d)}$

for `step` t `in` $0..T$ **do**

  ### Step 1: Sampling

  Sample $s = (w_1, .., w_n) \in C$ # a sequence in your corpus (e.g. sentence)

  Sample a pair $(i, j) \in [1, .., n]$ with $|i - j| \leq l$

  we note $w = w_i$, $c = w_j$ represented by vectors $\mathbf{w}$ in $\mathbf{W}$ and $\mathbf{c}$ in $\mathbf{C}$

  Sample $N_K = \{v_1, .., v_K\} \subset V$ represented by $\{\mathbf{v}_1, .., \mathbf{v}_K\}$ in $\mathbf{C}$ # Negative samples

  ### Step 2: Compute loss

  $l(\mathbf{W}, \mathbf{C}) = -\sigma(\mathbf{w}, \mathbf{c}) - \frac{1}{K} \sum_{v \in N_K} log\, \sigma(\text{-}\mathbf{w}, \mathbf{v})$

  ### Step 3: Parameter update with SGD

  $\mathbf{W}_t = \mathbf{W}_{t-1} - \alpha_t . \nabla\, l(\mathbf{W}_{t-1}, \mathbf{C}_{t-1})$

  $\mathbf{C}_t = \mathbf{C}_{t-1} - \alpha_t . \nabla\, l(\mathbf{W}_{t-1}, \mathbf{C}_{t-1})$

end

Word2Vec Model: Optimization

Algorithm 1 Skip-Gram Word2vec Training

Given a corpus C, made of a set of unique tokens V. Hyperparameters: number of negative samples K, a window size l, dimension of word vectors d, learning rate (α_t)
Initalize Randomly: $\mathbf{W} \in \mathbb{R}^{(V,d)}$ and $\mathbf{C} \in \mathbb{R}^{(V,d)}$
for *step* t *in* $0..T$ **do**

Step 1: Sampling
Sample $s = (w_1, .., w_n) \in C$ # a sequence in your corpus (e.g. sentence)
Sample a pair $(i, j) \in [1, .., n]$ with $|i - j| \leq l$
we note $w = w_i, c = w_j$ represented by vectors $\mathbf{w}$ in $\mathbf{W}$ and $\mathbf{c}$ in $\mathbf{C}$
Sample $N_K = \{v_1, .., v_K\} \subset V$ represented by $\{\mathbf{v}_1, .., \mathbf{v}_K\}$ in $\mathbf{C}$ # Negative samples

Step 2: Compute loss
$l(\mathbf{W}, \mathbf{C}) = -\sigma(\mathbf{w}, \mathbf{c}) - \frac{1}{K} \sum_{v \in N_K} log\, \sigma(\mathbf{-w, v})$

Step 3: Parameter update with SGD
$\mathbf{W}_t = \mathbf{W}_{t-1} - \alpha_t . \nabla\, l(\mathbf{W}_{t-1}, \mathbf{C}_{t-1})$
$\mathbf{C}_t = \mathbf{C}_{t-1} - \alpha_t . \nabla\, l(\mathbf{W}_{t-1}, \mathbf{C}_{t-1})$

end

Word2Vec Model: Optimization

Algorithm 1 Skip-Gram Word2vec Training

Given a corpus C, made of a set of unique tokens V. Hyperparameters: number of negative samples K, a window size l, dimension of word vectors d, learning rate (α_t)

Initalize Randomly: $\mathbf{W} \in \mathbb{R}^{(V,d)}$ and $\mathbf{C} \in \mathbb{R}^{(V,d)}$

for *step* t *in* $0, T$ **do**

Step 1: Sampling

Sample $s = (w_1, .., w_n) \in C$ # a sequence in your corpus (e.g. sentence)

Sample a pair $(i, j) \in [1, .., n]$ with $|i - j| \leq l$

we note $w = w_i, c = w_j$ represented by vectors $\mathbf{w}$ in $\mathbf{W}$ and $\mathbf{c}$ in $\mathbf{C}$

Sample $N_K = \{v_1, .., v_K\} \subset V$ represented by $\{\mathbf{v}_1, .., \mathbf{v}_K\}$ in $\mathbf{C}$ # Negative samples

Step 2: Compute loss

$l(\mathbf{W}, \mathbf{C}) = -\sigma(\mathbf{w}, \mathbf{c}) - \frac{1}{K} \sum_{v \in N_K} \mathit{log}\, \sigma(\text{-}\mathbf{w}, \mathbf{v})$

Step 3: Parameter update with SGD

$\mathbf{W}_t = \mathbf{W}_{t-1} - \alpha_t . \nabla\, l(\mathbf{W}_{t-1}, \mathbf{C}_{t-1})$

$\mathbf{C}_t = \mathbf{C}_{t-1} - \alpha_t . \nabla\, l(\mathbf{W}_{t-1}, \mathbf{C}_{t-1})$

end

Word2Vec Model: Optimization

Algorithm 1 Skip-Gram Word2vec Training

Given a corpus C, made of a set of unique tokens V. Hyperparameters: number of negative samples K, a window size l, dimension of word vectors d, learning rate (α_t)

Initalize Randomly: $\mathbf{W} \in \mathbb{R}^{(V,d)}$ and $\mathbf{C} \in \mathbb{R}^{(V,d)}$

for `step` t `in` $0..T$ **do**

Step 1: Sampling

Sample $s = (w_1, .., w_n) \in C$ # a sequence in your corpus (e.g. sentence)

Sample a pair $(i, j) \in [1, .., n]$ with $|i - j| \leq l$

we note $w = w_i$, $c = w_j$ represented by vectors $\mathbf{w}$ in $\mathbf{W}$ and $\mathbf{c}$ in $\mathbf{C}$

Sample $N_K = \{v_1, .., v_K\} \subset V$ represented by $\{\mathbf{v}_1, .., \mathbf{v}_K\}$ in $\mathbf{C}$ # Negative samples

Step 2: Compute loss

$l(\mathbf{W}, \mathbf{C}) = -\sigma(\mathbf{w}, \mathbf{c}) - \frac{1}{K} \sum_{v \in N_K} log\, \sigma(\text{-}\mathbf{w}, \mathbf{v})$

Step 3: Parameter update with SGD

$\mathbf{W}_t = \mathbf{W}_{t-1} - \alpha_t . \nabla\, l(\mathbf{W}_{t-1}, \mathbf{C}_{t-1})$

$\mathbf{C}_t = \mathbf{C}_{t-1} - \alpha_t . \nabla\, l(\mathbf{W}_{t-1}, \mathbf{C}_{t-1})$

end

Word2Vec Model: Optimization

Algorithm 1 Skip-Gram Word2vec Training

Given a corpus C, made of a set of unique tokens V. Hyperparameters: number of negative samples K, a window size l, dimension of word vectors d, learning rate (α_t)

Initalize Randomly: $\mathbf{W} \in \mathbb{R}^{(V,d)}$ and $\mathbf{C} \in \mathbb{R}^{(V,d)}$

for `step` t `in` $0..T$ **do**

    ### Step 1: Sampling
    Sample $s = (w_1, .., w_n) \in C$ # a sequence in your corpus (e.g. sentence)
    Sample a pair $(i, j) \in [1, .., n]$ with $|i - j| \leq l$
    we note $w = w_i$, $c = w_j$ represented by vectors $\mathbf{w}$ in $\mathbf{W}$ and $\mathbf{c}$ in $\mathbf{C}$
    Sample $N_K = \{v_1, .., v_K\} \subset V$ represented by $\{\mathbf{v}_1, .., \mathbf{v}_K\}$ in $\mathbf{C}$ # Negative samples
    ### Step 2: Compute loss
    $l(\mathbf{W}, \mathbf{C}) = -\sigma(\mathbf{w}, \mathbf{c}) - \frac{1}{K} \sum_{v \in N_K} \log \sigma(\text{-}\mathbf{w}, \mathbf{v})$
    ### Step 3: Parameter update with SGD
    $\mathbf{W}_t = \mathbf{W}_{t-1} - \alpha_t . \nabla\, l(\mathbf{W}_{t-1}, \mathbf{C}_{t-1})$
    $\mathbf{C}_t = \mathbf{C}_{t-1} - \alpha_t . \nabla\, l(\mathbf{W}_{t-1}, \mathbf{C}_{t-1})$

end

Word2Vec Model: Optimization

- Loop over the dataset E times (number of epochs)

- Complexity: O(d * K * T)
  - → No memory bottleneck
  - → Scale to Billion-tokens datasets

Algorithm 1 Skip-Gram Word2vec Training

Given a corpus C, made of a set of unique tokens V. Hyperparameters: number of negative samples K, a window size l, dimension of word vectors d, learning rate (α_t)

Initalize Randomly: $\mathbf{W} \in \mathbb{R}^{(V,d)}$ and $\mathbf{C} \in \mathbb{R}^{(V,d)}$

for $\texttt{step}\ t\ \texttt{in}\ 0..T$ **do**

    ### Step 1: Sampling

    Sample $s = (w_1, .., w_n) \in C$ # a sequence in your corpus (e.g. sentence)

    Sample a pair $(i, j) \in [1, .., n]$ with $|i - j| \leq l$

    we note $w = w_i$, $c = w_j$ represented by vectors **w** in **W** and **c** in **C**

    Sample $N_K = \{v_1, .., v_K\} \subset V$ represented by $\{\mathbf{v}_1, .., \mathbf{v}_K\}$ in **C** # Negative samples

    ### Step 2: Compute loss

    $l(\mathbf{W}, \mathbf{C}) = -\sigma(\mathbf{w}, \mathbf{c}) - \frac{1}{K} \sum_{v \in N_K} log\, \sigma(\textbf{-w}, \mathbf{v})$

    ### Step 3: Parameter update with SGD

    $\mathbf{W}_t = \mathbf{W}_{t-1} - \alpha_t . \nabla\, l(\mathbf{W}_{t-1}, \mathbf{C}_{t-1})$

    $\mathbf{C}_t = \mathbf{C}_{t-1} - \alpha_t . \nabla\, l(\mathbf{W}_{t-1}, \mathbf{C}_{t-1})$

end

Word2Vec Skip-Gram Model & the PMI

- (Levy & Goldberg 2014) showed that
  - Estimating the embedding matrix with Skip-Gram and Negative Sample (SGNS)…
  - … is equivalent to computing the shifted-PMI matrix

$$M_{ij}^{SGNS} = W_i \cdot C_j = \vec{w_i} \cdot \vec{c_j} = PMI(w_i, c_j) - \log k$$

Word2Vec

- Still very popular in practice
- Works very well with Deep Learning architecture (e.g., LSTM models) to model specific tasks (e.g., NER)
- Recently "beaten" by contextualized approaches (BERT)

- Extensions
  - Lots of variant of the Skip-Gram exists (CBOW, Glove…)
  - Multilingual setting: build shared representations across languages (fastext)

- Limits
  - Doesn't model morphology
  - Fixed Vocabulary: What if we add new tokens in the vocabulary?
  - Polysemy: each token has a unique representation (e.g., cherry)

Evaluation of Word embeddings

- How to evaluate the quality of word embeddings?

- Extrinsic Evaluation
  - Use them in a task-specific model and measure the performance on your task
- Intrinsic Evaluation
  - Idea: “Similar” words should have similar vectors

- What do we mean by “similar” words?
  - Morphologically similar: e.g., computer, computers
  - Syntactically similar: e.g., determiners
  - Semantically similar: e.g., animal, cat

Intrinsic Evaluation of Word Embeddings

- How to evaluate the quality of word embeddings?

- Qualitative Evaluation
  - Visualize word embedding space
  - Case by case: look at nearest neighbors of given words

- Quantitative Evaluation
  - Is word embedding similarity related with human judgement?

Intrinsic Evaluation of Word Embeddings

- Visualization

- Word vectors are high dimensions (usually >100)

  → Project the word embedding vectors using PCA or T-SNE
  → Visualize in 2D or 3D
  → Analyze the clusters

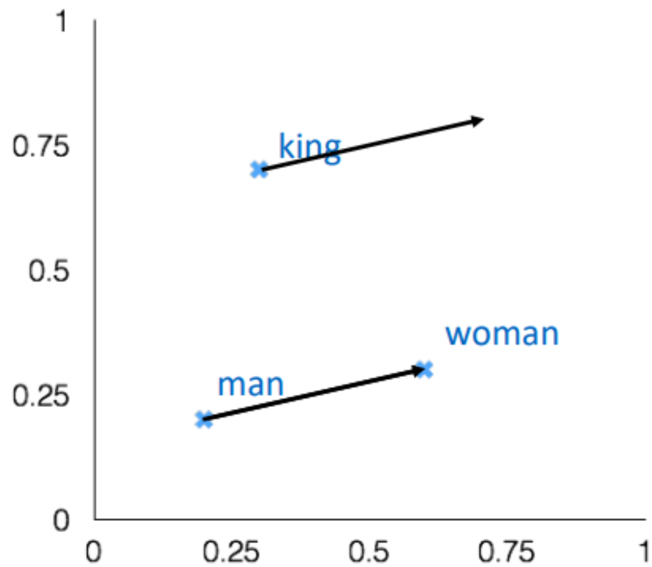
Intrinsic Evaluation of Word Embeddings

- Word Vector Analogies

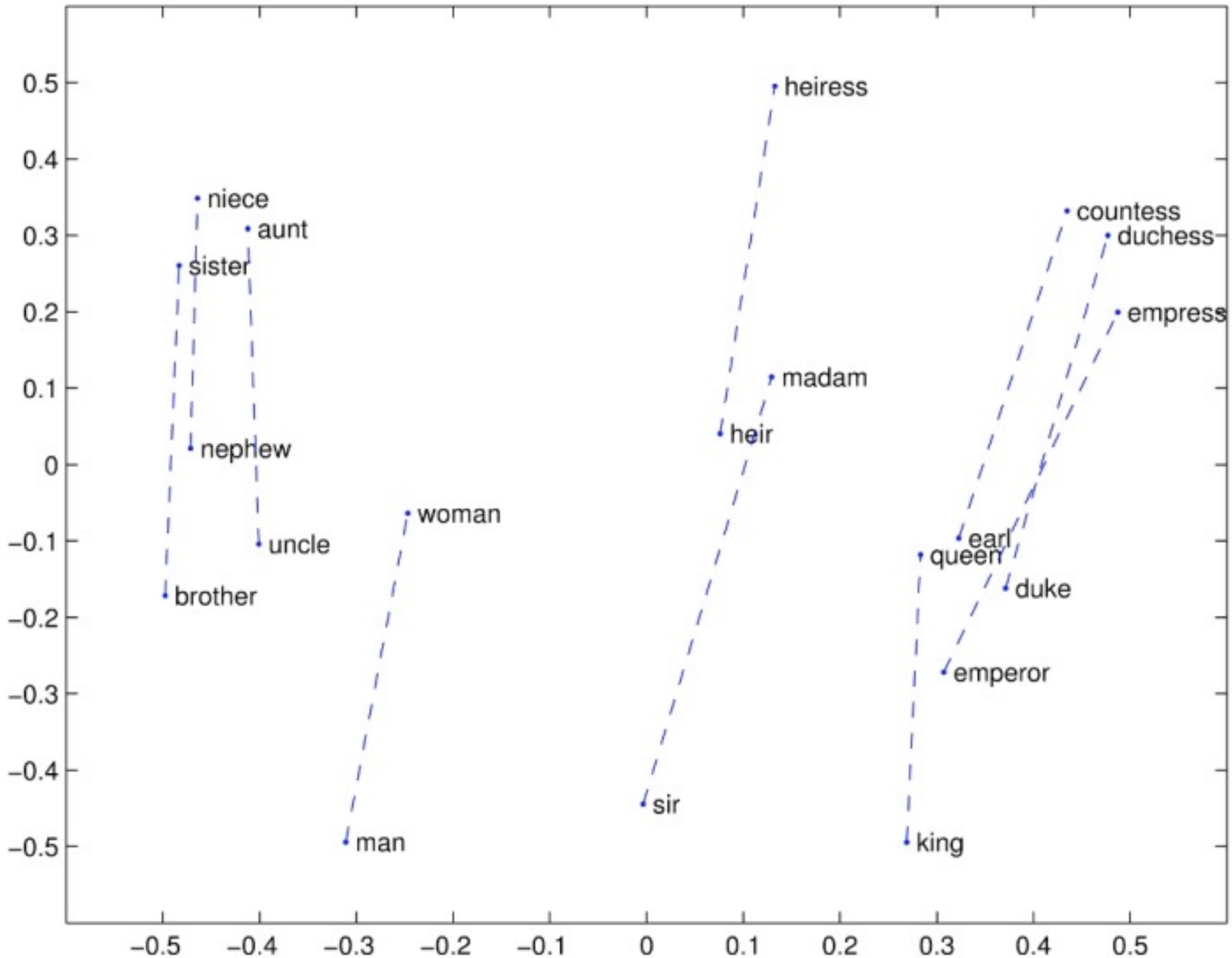
a:b :: c:?

man:woman :: king:?

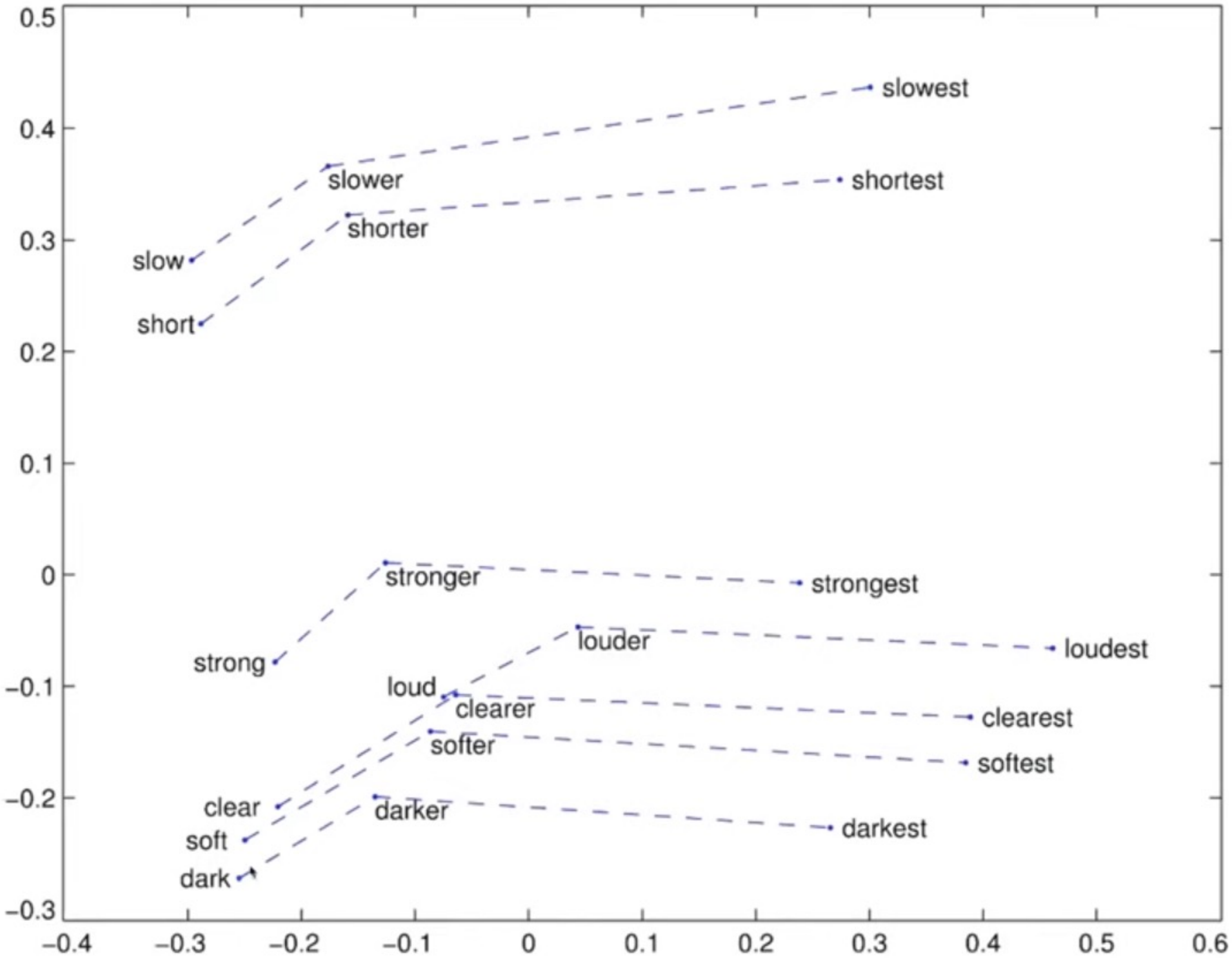
$$d = \arg\max_i \frac{(x_b - x_a + x_c)^T x_i}{||x_b - x_a + x_c||}$$



Intrinsic Evaluation of Word Embeddings



Intrinsic Evaluation of Word Embeddings



Intrinsic Evaluation of Word Embeddings

- How to measure similarity in the word embedding space?

  - Cosine Similarity

$$sim(w_1, w_2) = \cos\left(x_{w_1}, x_{w_2}\right) = x_{w_1}^T \cdot \frac{x_{w_2}}{\left|\left|x_{w_1}\right|\right| \cdot \left|\left|x_{w_2}\right|\right|}$$

  - L2 Distance

$$sim(w_1, w_2) = L2\left(x_{w_1}, x_{w_2}\right) = \left|\left|x_{w_1} - x_{w_2}\right|\right|$$

Intrinsic Evaluation of Word Embeddings

- Nearest-Neighbor with the cosine similarity (skip-gram trained on Wikipedia (1B tokens))

moon	score
mars	0.615
moons	0.611
lunar	0.602
sun	0.602
venus	0.583

talking	score
discussing	0.663
telling	0.657
joking	0.632
thinking	0.627
talked	0.624

blue	score
red	0.704
yellow	0.677
purple	0.676
green	0.655
pink	0.612

Intrinsic Evaluation of Word Embeddings

- We can compare the similarity between words in the embedding space with human judgement
  1. Collect human judgement on a list of pairs of words
  2. Compute similarity of the word vectors of those pairs
  3. Measure correlation between both

Word 1	Word 2	Word2vec Cosine Similarity	Human Judgment
tiger	tiger	1.0	10
dollar	buck	0.3065	9.22
dollar	profit	0.3420	7.38
smart	stupid	0.4128	5.81

Representing Documents with Vectors

- Similarity to what we saw for word-level representation, we can represent documents into vectors

1. Using word vectors
2. Count-based representations
3. Generative Probabilistic Graphical Model (e.g., LDA)
4. Using language models

Count-based Representation of Documents

- Given a Corpus made of novels of Shakespeare (Macbeth, Hamlet…), each document is a novel here:

1. Get the vocabulary of the Corpus

2. Compute the Count-based Matrix at the document-level

3. Build the term-frequency matrix

$tf_{t,d} = |\{t \in d\}|$: frequency of word t in document d

Count-based Representation of Documents

- Given a Corpus made of novels of Shakespeare (Macbeth, Hamlet…), each document is a novel here:

1. Get the vocabulary of the Corpus

2. Compute the Count-based Matrix at the document-level

	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
Antony	157	73	0	0	0	0
Brutus	4	157	0	1	0	0
Caesar	232	227	0	2	1	1
Calpurnia	0	10	0	0	0	0
Cleopatra	57	0	0	0	0	0
mercy	2	0	3	5	5	1
worser	2	0	1	1	1	0

Count-based Representation of Documents

	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
Antony	157	73	0	0	0	0
Brutus	4	157	0	1	0	0
Caesar	232	227	0	2	1	1
Calpurnia	0	10	0	0	0	0
Cleopatra	57	0	0	0	0	0
mercy	2	0	3	5	5	1
worser	2	0	1	1	1	0

- We get a vector representation for each document of the corpus
- Such a model is called a bag-of-word (BoW) model because the ordering of the words in each document does not matter

Count-based Representation of Documents

- Limits: high sensitivity to frequent words OR to very infrequent words

- How to improve?
  - A word that is in all documents of the corpus (e.g., “the”) is **not informative** at all for the document representation, still it impacts the document vector
  - A word that is in only 1 document is likely to be **very informative** of the document

- Solution:
  - Weight the count with → **Inverse Document Frequency**

Count-based Representation of Documents

- Weighting the importance of each term with the document frequency

- Definition: given N the total number of documents, a term t (token),

$$idf_{t,C} = \log\left(\frac{|C|}{|\{d \in C, s.t. t \in d\}|}\right)$$

- Compute the log to smooth the impact of words that are in only a few documents

TF-IDF Representation of Documents

- Matrix becomes: $tf * idf(t, d, C)$

	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
Antony	5.25	3.18	0	0	0	0.35
Brutus	1.21	6.1	0	1	0	0
Caesar	8.59	2.54	0	1.51	0.25	0
Calpurnia	0	1.54	0	0	0	0
Cleopatra	2.85	0	0	0	0	0
mercy	1.51	0	1.9	0.12	5.25	0.88
worser	1.37	0	0.11	4.15	0.25	1.95

TF-IDF Representation of Documents

- We can then apply dimension reduction technique to get dense vectors

  → e.g., we can apply SVD: Latent Semantic Analysis

Summary

- Word as one-hot vectors (using indexes)

- Hand-crafted approach (e.g., WordNet)

Word vectors inferred with data using the distributional hypothesis:
- Word Vectors with count-based approach

- Prediction-based approach with the skip-gram model

- Document representation: Bag-of-Words model and TF-IDF