3. Word vectors

LING-581-Natural Language Processing1

Instructor: Hakyung Sung

September 2, 2025

*Acknowledgment: These course slides are based on materials from CS224N: NLP with Deep Learning @ Stanford University.

Table of contents

Review

How do we represent the meanings *in computer*

Can computers understand meanings of the words as we do?

How do we represent the meanings *in computer*

Can computers understand meanings of the words as we do?

NO

How do we represent the meanings *in computer*

Can computers understand meanings of the words as we do?

NO

Traditional NLP method: Use the sets of synonyms and hypernyms of word by querying some databases (e.g., *WordNet*)

Problems with the traditional method (like WordNet)

- Missing nuances

Problems with the traditional method (like WordNet)

- Missing nuances
- Missing new meanings of words

Problems with the traditional method (like WordNet)

- Missing nuances
- Missing new meanings of words
    - Word meanings constantly change and adapt based on how people really use the language in the world

Problems with the traditional method (like WordNet)

- Missing nuances
- Missing new meanings of words
    - Word meanings constantly change and adapt based on how people really use the language in the world
    - Practically, building/updating a database is expensive and inefficient.

Problems with the traditional method (like WordNet)

- Missing nuances
- Missing new meanings of words
  - Word meanings constantly change and adapt based on how people really use the language in the world
  - Practically, building/updating a database is expensive and inefficient.
- Can’t compute accurate word similarity

Lesson plan

Lesson plan

- ~~Review~~

Lesson plan

- ~~Review~~
- Encoding and embedding

Lesson plan

- ~~Review~~
- Encoding and embedding
- Word2vec

Lesson plan

- ~~Review~~
- Encoding and embedding
- Word2vec
- Evaluation

Lesson plan

- ~~Review~~
- Encoding and embedding
- Word2vec
- Evaluation

Lesson plan

- ~~Review~~
- Encoding and embedding
- Word2vec
- Evaluation

Key idea: Word meanings can be represented well by a high-dimensional vector of real numbers

Encoding and embedding

Encoding

- Words themselves cannot be given as inputs to computers

Encoding

- Words themselves cannot be given as inputs to computers
- BUT, numbers can be given as inputs to computers

Encoding

- Words themselves cannot be given as inputs to computers
- BUT, numbers can be given as inputs to computers
- Encoding = converting words to vectors

Encoding

- Words themselves cannot be given as inputs to computers
- BUT, numbers can be given as inputs to computers
- Encoding = converting words to vectors
    - *vector*: an ordered list of numbers (e.g., [0.1, 0.3, -0.5])

One-hot encoding

- *The cat sat*

- Only the entry for the word is set to 1 (others = 0)

word	encoding
the	[1, 0, 0]
cat	[0, 1, 0]
sat	[0, 0, 1]

One-hot encoding

- *The cat sat*

word	encoding
the	[1, 0, 0]
cat	[0, 1, 0]
sat	[0, 0, 1]

- Only the entry for the word is set to 1 (others = 0)
- Each vector is in $\mathbb{R}^{|v| \times 1}$, where $|v| =$ vocabulary size

One-hot encoding

- *The cat sat*

word	encoding
the	[1, 0, 0]
cat	[0, 1, 0]
sat	[0, 0, 1]

- Only the entry for the word is set to 1 (others = 0)
- Each vector is in $\mathbb{R}^{|v| \times 1}$, where $|v| =$ vocabulary size
- For simplicity, we wrote them as row vectors in the - should be transposed; turning a row into a column vector)

One-hot encoding

- *The cat sat*

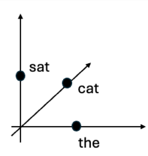
word	encoding
the	[1, 0, 0]
cat	[0, 1, 0]
sat	[0, 0, 1]

- Only the entry for the word is set to 1 (others = 0)
- Each vector is in $\mathbb{R}^{|v| \times 1}$, where $|v| =$ vocabulary size
- For simplicity, we wrote them as row vectors in the - should be transposed; turning a row into a column vector)
- **Localist, sparse** representation

One-hot encoding (problem)

- *The cat sat*

word	encoding
the	[1, 0, 0]
cat	[0, 1, 0]
sat	[0, 0, 1]



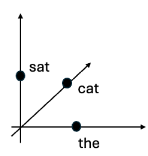
- Problems with one-hot encoding:
  - High dimensional vectors (size = vocab size)

- Solution:

One-hot encoding (problem)

- *The cat sat*

word	encoding
the	[1, 0, 0]
cat	[0, 1, 0]
sat	[0, 0, 1]

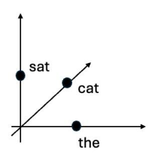


- Problems with one-hot encoding:
    - High dimensional vectors (size = vocab size)
    - No sense of similarity between words
- Solution:

One-hot encoding (problem)

- *The cat sat*

word	encoding
the	[1, 0, 0]
cat	[0, 1, 0]
sat	[0, 0, 1]



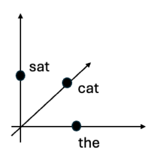
- Problems with one-hot encoding:
  - High dimensional vectors (size = vocab size)
  - No sense of similarity between words
  - All one-hot vectors are orthogonal (see graph)

- Solution:

One-hot encoding (problem)

- *The cat sat*

word	encoding
the	[1, 0, 0]
cat	[0, 1, 0]
sat	[0, 0, 1]

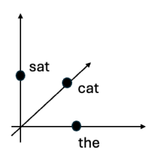


- Problems with one-hot encoding:
  - High dimensional vectors (size = vocab size)
  - No sense of similarity between words
  - All one-hot vectors are orthogonal (see graph)
  - Cosine similarity:
- Solution:

One-hot encoding (problem)

- *The cat sat*

word	encoding
the	[1, 0, 0]
cat	[0, 1, 0]
sat	[0, 0, 1]

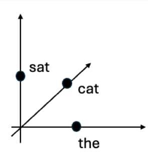


- Problems with one-hot encoding:
  - High dimensional vectors (size = vocab size)
  - No sense of similarity between words
  - All one-hot vectors are orthogonal (see graph)
  - Cosine similarity:
    - $\cos(\theta) = \frac{\vec{A} \cdot \vec{B}}{\|\vec{A}\| \|\vec{B}\|}$ (code)
- Solution:

One-hot encoding (problem)

- *The cat sat*

word	encoding
the	[1, 0, 0]
cat	[0, 1, 0]
sat	[0, 0, 1]

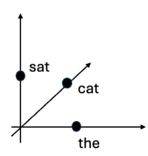


- Problems with one-hot encoding:
  - High dimensional vectors (size = vocab size)
  - No sense of similarity between words
  - All one-hot vectors are orthogonal (see graph)
  - **Cosine similarity:**
    - $\cos(\theta) = \frac{\vec{A} \cdot \vec{B}}{\|\vec{A}\| \|\vec{B}\|}$ (code)
- Solution:
  - Move from **sparse** to **distributed** representation

One-hot encoding (problem)

- *The cat sat*

word	encoding
the	[1, 0, 0]
cat	[0, 1, 0]
sat	[0, 0, 1]

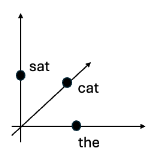


- Problems with one-hot encoding:
  - High dimensional vectors (size = vocab size)
  - No sense of similarity between words
  - All one-hot vectors are orthogonal (see graph)
  - **Cosine similarity:**
    - $\cos(\theta) = \frac{\vec{A} \cdot \vec{B}}{\|\vec{A}\| \|\vec{B}\|}$ (code)
- Solution:
  - Move from **sparse** to **distributed** representation
  - *Learn to encode similarity in the vectors themselves*

One-hot encoding (problem)

- *The cat sat*

word	encoding
the	[1, 0, 0]
cat	[0, 1, 0]
sat	[0, 0, 1]



- **Problems with one-hot encoding:**
  - High dimensional vectors (size = vocab size)
  - No sense of similarity between words
  - All one-hot vectors are orthogonal (see graph)
  - **Cosine similarity:**
    - $\cos(\theta) = \frac{\vec{A} \cdot \vec{B}}{\|\vec{A}\| \|\vec{B}\|}$ (code)
- **Solution:**
  - Move from **sparse** to **distributed** representation
  - *Learn to encode similarity in the vectors themselves*
  - Word embeddings (e.g., *Word2Vec, GloVe*)

Representing words by their *context*

- **Distributional semantics**: A word's meaning is given by the words that frequently appear close-by

Representing words by their *context*

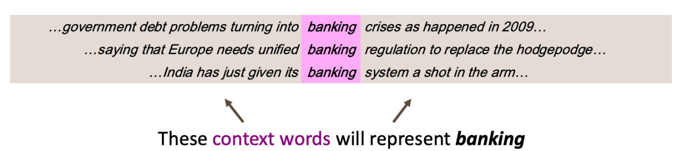
- **Distributional semantics**: A word's meaning is given by the words that frequently appear close-by
- "You shall know a word by the company it keeps" (Firth, 1957) - One of the most successful ideas of modern statistical NLP.

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" (Firth, 1957) - 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)

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" (Firth, 1957) - 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 context of *w* to build up a representation of *w*



Word vector representations: Two ways

1. **Count-based models**: Build a co-occurrence matrix and apply SVD

Word vector representations: Two ways

1. **Count-based models**: Build a co-occurrence matrix and apply SVD
2. **Neural network–based models**: Learn embeddings by predicting context words (e.g., Word2Vec, GloVe)

Word vectors: Count-based Models

- Start with a Bag-of-Words (BoW) representation

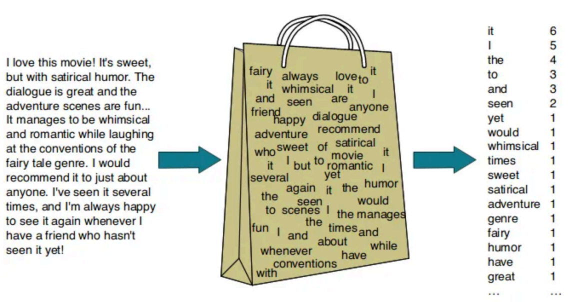
Word vectors: Count-based Models

- Start with a Bag-of-Words (BoW) representation
- Extend to a co-occurrence matrix: count how often words appear together in a context window

Word vectors: Count-based Models

- Start with a Bag-of-Words (BoW) representation
- Extend to a co-occurrence matrix: count how often words appear together in a context window
- Apply Singular Value Decomposition (SVD) to reduce dimensions (a way of breaking a big matrix into a smaller pieces)

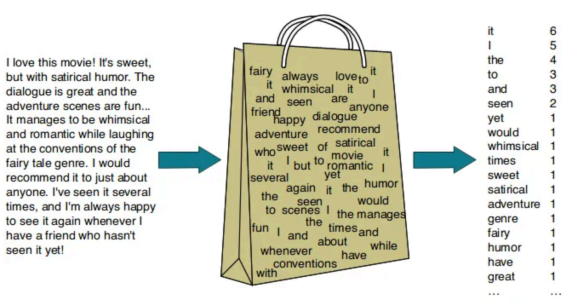
cf. *Bag of Words*



(source: `https://nachi-keta.medium.com/nlp-explain-bag-of-words-3b9fc4f211e8`)

- **Bag-of-Words assumption:** Context words are treated as **unordered and independent**.

cf. *Bag of Words*



(source: `https://nachi-keta.medium.com/nlp-explain-bag-of-words-3b9fc4f211e8`)

- **Bag-of-Words assumption:** Context words are treated as **unordered and independent**.
- In other words, the **position** of a context word relative to the target is ignored.

Word vectors: Count-based models

Example sentences:

- I like apples.
- You like bananas.
- They eat bananas.
- We enjoy apples.
- They like fruit.

Word vectors: Count-based models

Example sentences:

- I like apples.
- You like bananas.
- They eat bananas.
- We enjoy apples.
- They like fruit.

	I	you	we	they	like	eat	enjoy	apples	bananas	fruit
I	0	0	0	0	1	0	0	0	0	0
you	0	0	0	0	1	0	0	0	0	0
we	0	0	0	0	0	0	1	0	0	0
they	0	0	0	0	1	1	0	0	0	0
like	1	1	0	1	0	0	0	1	1	1
eat	0	0	0	1	0	0	0	0	1	0
enjoy	0	0	1	0	0	0	0	1	0	0
apples	0	0	0	0	1	0	1	0	0	0
bananas	0	0	0	0	1	1	0	0	0	0
fruit	0	0	0	0	1	0	0	0	0	0

Co-occurrence Matrix (window size = 1)

Count-based models (Limitations)

- High computational cost

Count-based models (Limitations)

- High computational cost
  - Co-occurrence matrix size: $|V| \times |V|$ (vocabulary squared)

Count-based models (Limitations)

- High computational cost
  - Co-occurrence matrix size: $|V| \times |V|$ (vocabulary squared)
  - Too large to store/compute for big corpora

Count-based models (Limitations)

- High computational cost
  - Co-occurrence matrix size: $|V| \times |V|$ (vocabulary squared)
  - Too large to store/compute for big corpora
- Sparse and noisy

Count-based models (Limitations)

- High computational cost
    - Co-occurrence matrix size: $|V| \times |V|$ (vocabulary squared)
    - Too large to store/compute for big corpora
- Sparse and noisy
    - Most cells are 0 $\Rightarrow$ sparse matrix

Count-based models (Limitations)

- High computational cost
  - Co-occurrence matrix size: $|V| \times |V|$ (vocabulary squared)
  - Too large to store/compute for big corpora
- Sparse and noisy
  - Most cells are 0 $\Rightarrow$ sparse matrix
  - Rare words/contexts yield unreliable statistics

Count-based models (Limitations)

- High computational cost
  - Co-occurrence matrix size: $|V| \times |V|$ (vocabulary squared)
  - Too large to store/compute for big corpora
- Sparse and noisy
  - Most cells are 0 $\Rightarrow$ sparse matrix
  - Rare words/contexts yield unreliable statistics
- Poor scalability / update issues

Count-based models (Limitations)

- High computational cost
  - Co-occurrence matrix size: $|V| \times |V|$ (vocabulary squared)
  - Too large to store/compute for big corpora
- Sparse and noisy
  - Most cells are 0 $\Rightarrow$ sparse matrix
  - Rare words/contexts yield unreliable statistics
- Poor scalability / update issues
  - Adding new words requires recomputing the entire matrix and SVD

Word vectors: Neural Network–Based Models

How are they different from count-based models?

Word vectors: Neural Network–Based Models

How are they different from count-based models?

- **Count-based models**: build large co-occurrence matrices, then compress (e.g., SVD)

Word vectors: Neural Network–Based Models

How are they different from count-based models?

- **Count-based models**: build large co-occurrence matrices, then compress (e.g., SVD)
- **Neural models**: learn vectors *directly* by predicting context words

Word vectors: Neural Network–Based Models

How are they different from count-based models?

- **Count-based models**: build large co-occurrence matrices, then compress (e.g., SVD)
- **Neural models**: learn vectors *directly* by predicting context words

Word vectors: Neural Network–Based Models

How are they different from count-based models?

- **Count-based models**: build large co-occurrence matrices, then compress (e.g., SVD)
- **Neural models**: learn vectors *directly* by predicting context words

Consistent progress

- 1986: *Learning representations by back propagting errors* (Rumelhart et al., 1986)

Word vectors: Neural Network–Based Models

How are they different from count-based models?

- **Count-based models**: build large co-occurrence matrices, then compress (e.g., SVD)
- **Neural models**: learn vectors *directly* by predicting context words

Consistent progress

- 1986: *Learning representations by back propagting errors* (Rumelhart et al., 1986)
- 2003: *A neural probabilistic language model* (Bengio et al., 2003)

Word vectors: Neural Network–Based Models

How are they different from count-based models?

- **Count-based models**: build large co-occurrence matrices, then compress (e.g., SVD)
- **Neural models**: learn vectors *directly* by predicting context words

Consistent progress

- 1986: *Learning representations by back propagting errors* (Rumelhart et al., 1986)
- 2003: *A neural probabilistic language model* (Bengio et al., 2003)
- 2013: Word2Vec (Skip-gram, CBOW)

Word vectors: Neural Network–Based Models

How are they different from count-based models?

- **Count-based models**: build large co-occurrence matrices, then compress (e.g., SVD)
- **Neural models**: learn vectors *directly* by predicting context words

Consistent progress

- 1986: *Learning representations by back propagting errors* (Rumelhart et al., 1986)
- 2003: *A neural probabilistic language model* (Bengio et al., 2003)
- 2013: Word2Vec (Skip-gram, CBOW)
- 2014–2015: GloVe, fastText

Word vectors: Neural Network–Based Models

How are they different from count-based models?

- **Count-based models**: build large co-occurrence matrices, then compress (e.g., SVD)
- **Neural models**: learn vectors *directly* by predicting context words

Consistent progress

- 1986: *Learning representations by back propagting errors* (Rumelhart et al., 1986)
- 2003: *A neural probabilistic language model* (Bengio et al., 2003)
- 2013: Word2Vec (Skip-gram, CBOW)
- 2014–2015: GloVe, fastText
- 2018– : Contextual embeddings (ELMo, BERT, GPT)

Word2vec

Word2vec: Overview

- Word2vec (Mikolov et al., 2013) is a framework for learning word vectors
- Idea:
  - Start with a large corpus ("body") of text

Word2vec: Overview

- **Word2vec** (Mikolov et al., 2013) is a framework for learning word vectors
- Idea:
  - Start with a large corpus ("body") of text
  - Every word in a fixed vocabulary is represented by a **vector**

Word2vec: Overview

- **Word2vec** (Mikolov et al., 2013) is a framework for learning word vectors
- Idea:
  - Start with a large corpus (“body”) of text
  - 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) word o

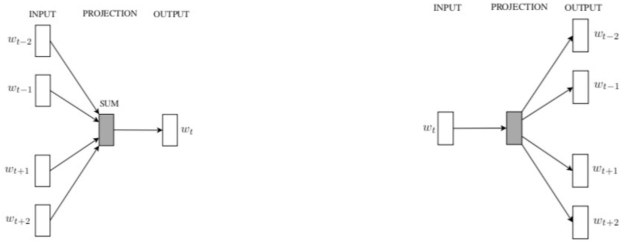
Word2vec: Overview

- **Word2vec** (Mikolov et al., 2013) is a framework for learning word vectors
- Idea:
  - Start with a large corpus ("body") of text
  - 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) word o
  - Use the **similarity of the word vectors** for c and o to **calculate the probability** of o given c (or vice versa)

Word2vec: Overview

- Word2vec (Mikolov et al., 2013) is a framework for learning word vectors
- Idea:
  - Start with a large corpus (“body”) of text
  - 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) word 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** the probability

Word2vec: Two models



Continuous Bag of Words (CBOW):
predicting the center words using
the context words ($P(w_t|w_{t-2}, w_{t-1}, w_{t+1}, w_{t+2})$)

Skip-grams (SG):
predicting the context words using
the center word ($P(w_{t+i}|w_t), i \in \{-2, -1, 1, 2\}$)

In practice, we focus on Skip-gram.

Build training pairs

- Take a large text corpus

Build training pairs

- Take a large text corpus
- For each word, collect nearby words within a fixed window size

Build training pairs

- Take a large text corpus
- For each word, collect nearby words within a fixed window size
- These become training pairs: (center word, context word)

Word2Vec: Skip-grams (*window size* = 1)

- “king brave man”
- “queen beautiful woman”

word	**neighbor**
king	brave
brave	king
brave	man
man	brave
queen	beautiful
beautiful	queen
beautiful	woman
woman	beautiful

Word2Vec: Skip-grams (*window size* = 2)

- "king brave man"
- "queen beautiful woman"

word	**neighbor**
king	brave
king	man
brave	man
brave	king
man	king
man	brave
queen	beautiful
queen	woman
beautiful	queen
beautiful	woman
woman	queen
woman	beautiful

Word2Vec: Skip-grams (window size = 2)

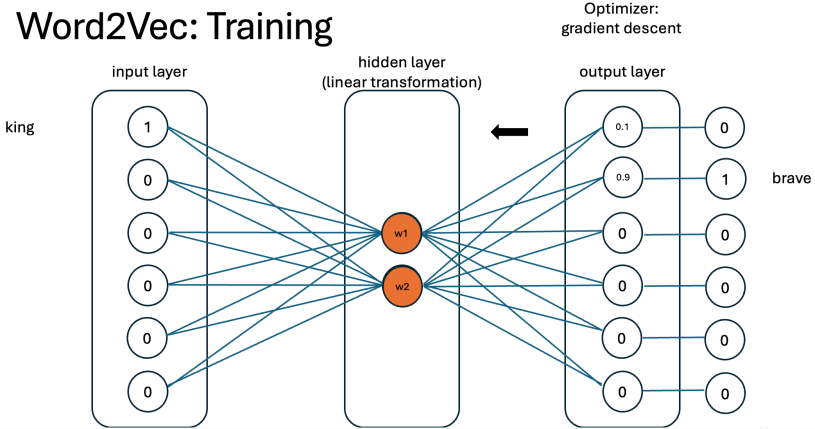
word	one-hot encoding	neighbor	one-hot encoding
king	[1, 0, 0, 0, 0, 0]	brave	[0, 1, 0, 0, 0, 0]
king	[1, 0, 0, 0, 0, 0]	man	[0, 0, 1, 0, 0, 0]
brave	[0, 1, 0, 0, 0, 0]	man	[0, 0, 1, 0, 0, 0]
brave	[0, 1, 0, 0, 0, 0]	king	[1, 0, 0, 0, 0, 0]
man	[0, 0, 1, 0, 0, 0]	king	[1, 0, 0, 0, 0, 0]
man	[0, 0, 1, 0, 0, 0]	brave	[0, 1, 0, 0, 0, 0]
queen	[0, 0, 0, 1, 0, 0]	beautiful	[0, 0, 0, 0, 1, 0]
queen	[0, 0, 0, 1, 0, 0]	woman	[0, 0, 0, 0, 0, 1]
beautiful	[0, 0, 0, 0, 1, 0]	queen	[0, 0, 0, 1, 0, 0]
beautiful	[0, 0, 0, 0, 1, 0]	woman	[0, 0, 0, 0, 0, 1]
woman	[0, 0, 0, 0, 0, 1]	queen	[0, 0, 0, 1, 0, 0]
woman	[0, 0, 0, 0, 0, 1]	beautiful	[0, 0, 0, 0, 1, 0]

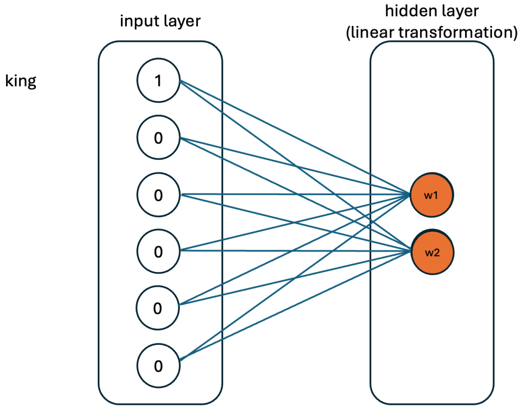
Word2Vec: Input and output

input
[1, 0, 0, 0, 0, 0]
[1, 0, 0, 0, 0, 0]
[0, 1, 0, 0, 0, 0]
[0, 1, 0, 0, 0, 0]
[0, 0, 1, 0, 0, 0]
[0, 0, 1, 0, 0, 0]
[0, 0, 0, 1, 0, 0]
[0, 0, 0, 1, 0, 0]
[0, 0, 0, 0, 1, 0]
[0, 0, 0, 0, 1, 0]
[0, 0, 0, 0, 0, 1]
[0, 0, 0, 0, 0, 1]

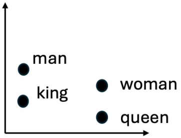
output
[0, 1, 0, 0, 0, 0]
[0, 0, 1, 0, 0, 0]
[0, 0, 1, 0, 0, 0]
[1, 0, 0, 0, 0, 0]
[1, 0, 0, 0, 0, 0]
[0, 1, 0, 0, 0, 0]
[0, 0, 0, 0, 1, 0]
[0, 0, 0, 0, 0, 1]
[0, 0, 0, 1, 0, 0]
[0, 0, 0, 0, 0, 1]
[0, 0, 0, 1, 0, 0]
[0, 0, 0, 0, 1, 0]

Word2Vec: Training





word	embedding
king	[1, 1]
brave	[1, 2]
man	[1, 3]
queen	[5, 1]
beautiful	[5, 2]
woman	[5, 3]



1. One-hot and embedding lookup

- Each word in the vocabulary is represented as a **dense vector**.

1. One-hot and embedding lookup

- Each word in the vocabulary is represented as a **dense vector**.
- All these word vectors are stored in a single matrix:

$$\textbf{Embedding matrix} \quad E \in \mathbb{R}^{V \times d}$$

1. One-hot and embedding lookup

- Each word in the vocabulary is represented as a **dense vector**.
- All these word vectors are stored in a single matrix:

$$\textbf{Embedding matrix} \quad E \in \mathbb{R}^{V \times d}$$

- Why do we store all vectors in one matrix?

1. One-hot and embedding lookup

- Each word in the vocabulary is represented as a **dense vector**.
- All these word vectors are stored in a single matrix:

$$\textbf{Embedding matrix} \quad E \in \mathbb{R}^{V \times d}$$

- Why do we store all vectors in one matrix?
    - Each word has a unique ID, so we can quickly select its row from the matrix.

1. One-hot and embedding lookup

- Each word in the vocabulary is represented as a **dense vector**.
- All these word vectors are stored in a single matrix:

$$\textbf{Embedding matrix} \quad E \in \mathbb{R}^{V \times d}$$

- Why do we store all vectors in one matrix?
  - Each word has a unique ID, so we can quickly select its row from the matrix.
  - This operation is very efficient — it's just a lookup.

2. Predicting context words

- Take the center word’s embedding

2. Predicting context words

- Take the center word's embedding
- Compare it with each candidate context word's output vector

2. Predicting context words

- Take the center word's embedding
- Compare it with each candidate context word's output vector
- Compute a **dot product** as a similarity score

Note. Dot product as similarity score

- **Algebraic definition:** For two vectors $a = (a_1, \ldots, a_n)$ and $b = (b_1, \ldots, b_n)$,

$$a \cdot b = \sum_{i=1}^{n} a_i b_i$$

  (multiply each coordinate and add them up)

- **Geometric interpretation:** The same dot product can also be written as

$$a \cdot b = \|a\| \, \|b\| \cos \theta$$

  where θ is the angle between a and b. Larger values $\Rightarrow$ vectors point in a similar direction (more related).

Note. Dot product as similarity score

- In Word2Vec:

$$s(w|c) = v_c \cdot u_w = \sum_{i=1}^{d} v_{c,i}\, u_{w,i}$$

  where v_c is the center word vector, u_w is a candidate context vector.

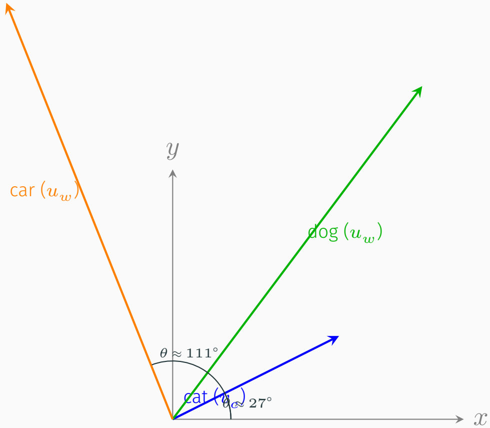
- **Toy example:** $v_c = [2, 1]$ ("cat"), $u_w = [3, 4]$ ("dog")

$$v_c \cdot u_w = (2 \times 3) + (1 \times 4) = 10$$

- **Comparison:** $u_w = [-2, 5]$ ("car")

$$v_c \cdot u_w = (2 \times -2) + (1 \times 5) = 1$$

Note. Dot Product as Geometry (Examples)



- $v_c = [2, 1]$ ("cat"), $u_w = [3, 4]$ ("dog") $\quad v_c \cdot u_w = 10 \quad \Rightarrow$ large positive (similar direction).
- $v_c = [2, 1]$ ("cat"), $u_w = [-2, 5]$ ("car") $\quad v_c \cdot u_w = 1 \quad \Rightarrow$ small (weak relation).

3. From similarity scores to probabilities

- After retrieving the center word and a context word's vectors, we compute their **dot product**:

$$\text{score} = \vec{v}_c \cdot \vec{u}_w$$

3. From similarity scores to probabilities

- After retrieving the center word and a context word's vectors, we compute their **dot product**:

$$\text{score} = \vec{v}_c \cdot \vec{u}_w$$

- To interpret this score as a probability, we apply the **sigmoid function**:

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

3. From similarity scores to probabilities

- After retrieving the center word and a context word's vectors, we compute their **dot product**:

$$\text{score} = \vec{v}_c \cdot \vec{u}_w$$

- To interpret this score as a probability, we apply the **sigmoid function**:

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

- The output is a number between 0 and 1 — representing how likely this word is to appear in the context.

4. Compute loss

- We compare predicted probabilities with actual labels:

4. Compute loss

- We compare predicted probabilities with actual labels:
    - **True context words** → label = 1

4. Compute loss

- We compare predicted probabilities with actual labels:
  - **True context words** → label = 1
  - **Negative (random) words** → label = 0

4. Compute loss

- We compare predicted probabilities with actual labels:
  - **True context words** → label = 1
  - **Negative (random) words** → label = 0
- We apply the **binary cross-entropy loss**:

$$\mathcal{L} = -\left(\log \sigma(\vec{v}_c \cdot \vec{u}_{w^+}) + \sum_{i=1}^{k} \log \left(1 - \sigma(\vec{v}_c \cdot \vec{u}_{w_i^-}) \right) \right)$$

4. Compute loss

- We compare predicted probabilities with actual labels:
  - **True context words** → label = 1
  - **Negative (random) words** → label = 0
- We apply the **binary cross-entropy loss**:

$$\mathcal{L} = -\left(\log \sigma(\vec{v}_c \cdot \vec{u}_{w^+}) + \sum_{i=1}^{k} \log \left(1 - \sigma(\vec{v}_c \cdot \vec{u}_{w_i^-})\right) \right)$$

- The model is rewarded when:

4. Compute loss

- We compare predicted probabilities with actual labels:
  - **True context words** → label = 1
  - **Negative (random) words** → label = 0
- We apply the **binary cross-entropy loss**:

$$\mathcal{L} = -\left(\log \sigma(\vec{v}_c \cdot \vec{u}_{w^+}) + \sum_{i=1}^{k} \log\left(1 - \sigma(\vec{v}_c \cdot \vec{u}_{w_i^-})\right)\right)$$

- The model is rewarded when:
  - It assigns high probability to true context words

4. Compute loss

- We compare predicted probabilities with actual labels:
    - **True context words** → label = 1
    - **Negative (random) words** → label = 0
- We apply the **binary cross-entropy loss**:

$$\mathcal{L} = -\left(\log \sigma(\vec{v}_c \cdot \vec{u}_{w^+}) + \sum_{i=1}^{k} \log \left(1 - \sigma(\vec{v}_c \cdot \vec{u}_{w_i^-}) \right) \right)$$

- The model is rewarded when:
    - It assigns high probability to true context words
    - It assigns low probability to negative (random) words

4. Compute loss

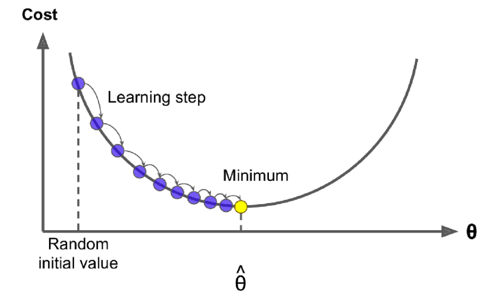
- We compare predicted probabilities with actual labels:
    - **True context words** → label = 1
    - **Negative (random) words** → label = 0
- We apply the **binary cross-entropy loss**:

$$\mathcal{L} = -\left(\log \sigma(\vec{v}_c \cdot \vec{u}_{w^+}) + \sum_{i=1}^{k} \log \left(1 - \sigma(\vec{v}_c \cdot \vec{u}_{w_i^-})\right) \right)$$

- The model is rewarded when:
    - It assigns high probability to true context words
    - It assigns low probability to negative (random) words
- The model adjusts vectors to maximize the probability of real words and minimize that of negatives

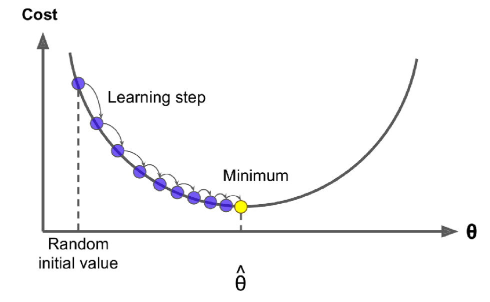
5. Update word vectors

- Optimizer updates parameters based on gradients



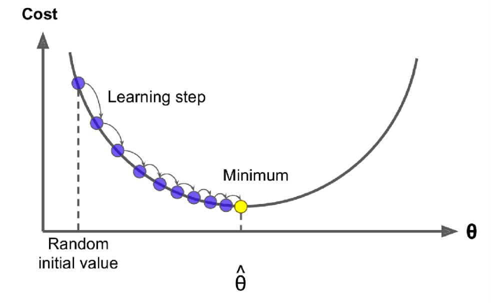
5. Update word vectors

- Optimizer updates parameters based on gradients
- Parameters updated:



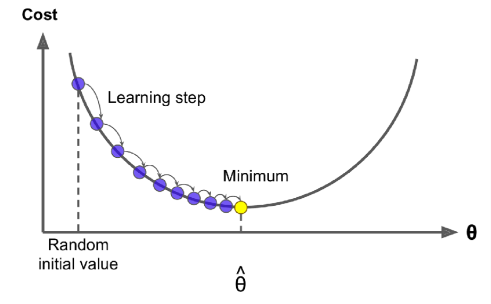
5. Update word vectors

- Optimizer updates parameters based on gradients
- Parameters updated:
  - The center word's vector



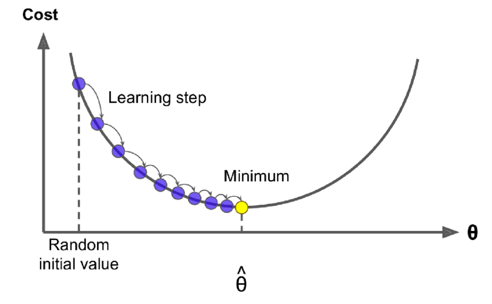
5. Update word vectors

- Optimizer updates parameters based on gradients
- Parameters updated:
    - The center word's vector
    - The true context word's vector



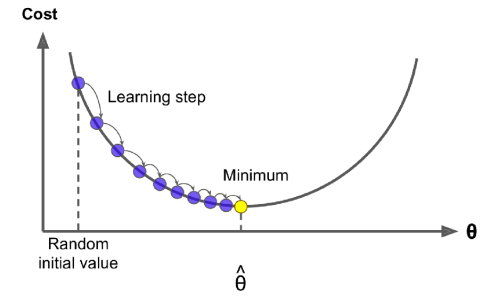
5. Update word vectors

- Optimizer updates parameters based on gradients
- Parameters updated:
    - The center word's vector
    - The true context word's vector
    - The negative samples' vectors



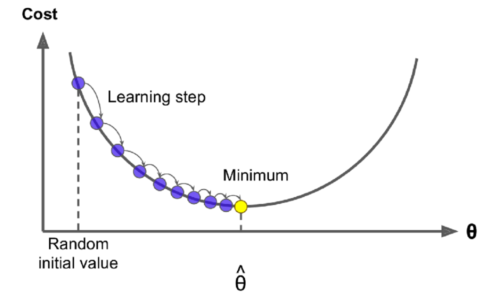
5. Update word vectors

- Optimizer updates parameters based on gradients
- Parameters updated:
  - The center word's vector
  - The true context word's vector
  - The negative samples' vectors
- Over time, words with similar contexts move closer in vector space



5. Update word vectors

- Optimizer updates parameters based on gradients
- Parameters updated:
  - The center word's vector
  - The true context word's vector
  - The negative samples' vectors
- Over time, words with similar contexts move closer in vector space
- We'll look at the optimization more closely in the following slides.



Note1. Embedding matrix

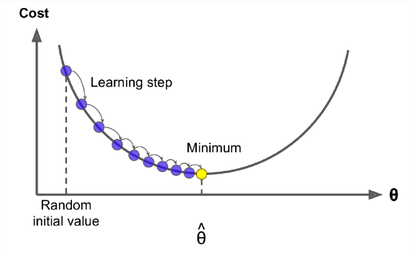
- E is the embedding matrix: each row corresponds to one word
- Its size:

$$E \in \mathbb{R}^{V \times N}$$

  - V = vocabulary size (number of unique words)
  - N = embedding dimension (hyperparameter)
- Example: $V = 10{,}000$, $N = 300 \Rightarrow 3$ million parameters
- Larger N = more expressive vectors, but higher cost

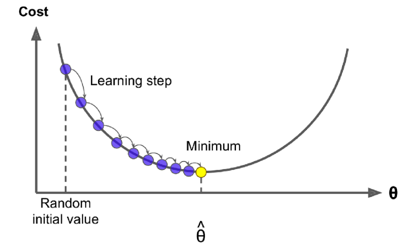
Note2. Optimization: Gradient Descent

- **Goal:** Learn good word vectors by minimizing a loss function $J(\theta)$ (measures how wrong predictions are).



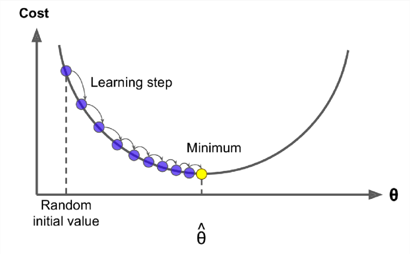
Note2. Optimization: Gradient Descent

- **Goal:** Learn good word vectors by minimizing a loss function $J(\theta)$ (measures how wrong predictions are).
- **Idea:**



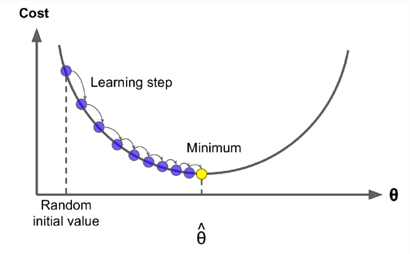
Note2. Optimization: Gradient Descent

- **Goal:** Learn good word vectors by minimizing a loss function $J(\theta)$ (measures how wrong predictions are).
- **Idea:**
  - Start from random initial values



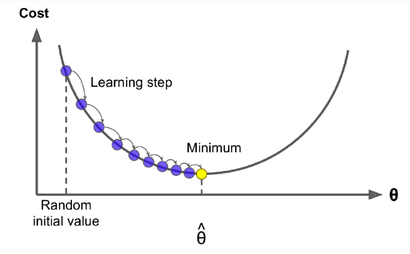
Note2. Optimization: Gradient Descent

- **Goal:** Learn good word vectors by minimizing a loss function $J(\theta)$ (measures how wrong predictions are).
- **Idea:**
  - Start from random initial values
  - Compute the gradient of $J(\theta)$ (which tells us the slope)



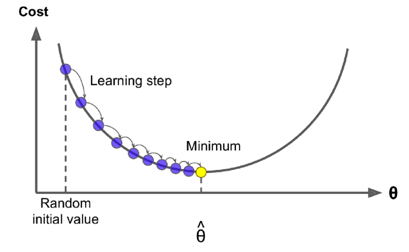
Note2. Optimization: Gradient Descent

- **Goal:** Learn good word vectors by minimizing a loss function $J(\theta)$ (measures how wrong predictions are).
- **Idea:**
  - Start from random initial values
  - Compute the gradient of $J(\theta)$ (which tells us the slope)
  - Move a small step in the **opposite direction** of the gradient



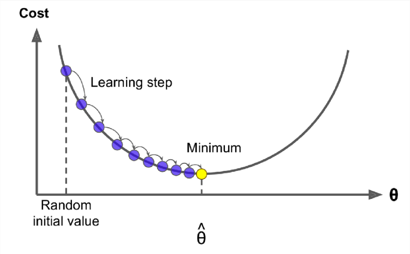
Note2. Optimization: Gradient Descent

- **Goal:** Learn good word vectors by minimizing a loss function $J(\theta)$ (measures how wrong predictions are).
- **Idea:**
  - Start from random initial values
  - Compute the gradient of $J(\theta)$ (which tells us the slope)
  - Move a small step in the **opposite direction** of the gradient
  - Repeat many times until the loss becomes small



Note2. Optimization: Gradient Descent

- **Goal:** Learn good word vectors by minimizing a loss function $J(\theta)$ (measures how wrong predictions are).
- **Idea:**
  - Start from random initial values
  - Compute the gradient of $J(\theta)$ (which tells us the slope)
  - Move a small step in the **opposite direction** of the gradient
  - Repeat many times until the loss becomes small
- Loss functions may not always convex.



(Batch) Gradient Descent Algorithm: Issues

- Compute the gradient of $J(\theta)$ using **all data**, then update θ.
- Because all data is considered, the update direction is accurate.
- However, when the dataset is large, computation becomes very slow.

(Batch) Gradient Descent Algorithm: Issues

- Compute the gradient of $J(\theta)$ using **all data**, then update θ.
- Because all data is considered, the update direction is accurate.
- However, when the dataset is large, computation becomes very slow.

Stochastic Gradient Descent (SGD)

- Randomly sample **one data point** from the training set, compute its gradient, then update θ.
- Because only one sample is used, the path may fluctuate a lot.
- Despite the noise, it is much faster than batch gradient descent.

Note3. Optimization: Stochastic Gradient Descent (SGD)

(Batch) Gradient Descent Algorithm: Issues

- Compute the gradient of $J(\theta)$ using **all data**, then update θ.
- Because all data is considered, the update direction is accurate.
- However, when the dataset is large, computation becomes very slow.

Stochastic Gradient Descent (SGD)

- Randomly sample **one data point** from the training set, compute its gradient, then update θ.
- Because only one sample is used, the path may fluctuate a lot.
- Despite the noise, it is much faster than batch gradient descent.

Mini-Batch Gradient Descent

- Compute the gradient using a **mini-batch** of data, then update θ.
- This balances the pros and cons of batch and stochastic gradient descent, making it the most practical method.

GloVe

Revisit: Count-based & Neural-based models

- Count-based
  - Fast training
  - Efficient usage of statistics
  - Primarily used to capture word similarity
- Neural-based
- Scales with corpus size
- Inefficient usage of statistics (e.g., random sampling)

Motivation: Encoding meaning via co-occurrence ratios

- Idea: Meaning differences between words can be reflected in the **ratios** of their co-occurrence probabilities with other words.
- **GloVe leverages these ratios to learn word vectors** where vector differences encode semantic components.
- Next lecture (on Tuesday), we'll start from here.

Evaluation

How to evaluate word vectors (based on the GloVe)

Extrinsic evaluation

- Evaluate performance when word vectors are used in a real downstream task.

Intrinsic evaluation

How to evaluate word vectors (based on the GloVe)

Extrinsic evaluation

- Evaluate performance when word vectors are used in a real downstream task.
- Requires evaluation at every epoch while solving the real task $\Rightarrow$ time-consuming.

Intrinsic evaluation

How to evaluate word vectors (based on the GloVe)

Extrinsic evaluation

- Evaluate performance when word vectors are used in a real downstream task.
- Requires evaluation at every epoch while solving the real task $\Rightarrow$ time-consuming.
- Hard to tell whether performance issues come from the model structure itself or from the embeddings.

Intrinsic evaluation

How to evaluate word vectors (based on the GloVe)

Extrinsic evaluation

- Evaluate performance when word vectors are used in a real downstream task.
- Requires evaluation at every epoch while solving the real task $\Rightarrow$ time-consuming.
- Hard to tell whether performance issues come from the model structure itself or from the embeddings.
- e.g., Name Entity Recognition Task

Intrinsic evaluation

How to evaluate word vectors (based on the GloVe)

Extrinsic evaluation

- Evaluate performance when word vectors are used in a real downstream task.
- Requires evaluation at every epoch while solving the real task $\Rightarrow$ time-consuming.
- Hard to tell whether performance issues come from the model structure itself or from the embeddings.
- e.g., Name Entity Recognition Task

Intrinsic evaluation

- Evaluate performance through concrete subtasks at intermediate stages (e.g., word similarity, analogy).

How to evaluate word vectors (based on the GloVe)

Extrinsic evaluation

- Evaluate performance when word vectors are used in a real downstream task.
- Requires evaluation at every epoch while solving the real task $\Rightarrow$ time-consuming.
- Hard to tell whether performance issues come from the model structure itself or from the embeddings.
- e.g., Name Entity Recognition Task

Intrinsic evaluation

- Evaluate performance through concrete subtasks at intermediate stages (e.g., word similarity, analogy).
- Faster evaluation speed.

How to evaluate word vectors (based on the GloVe)

Extrinsic evaluation

- Evaluate performance when word vectors are used in a real downstream task.
- Requires evaluation at every epoch while solving the real task $\Rightarrow$ time-consuming.
- Hard to tell whether performance issues come from the model structure itself or from the embeddings.
- e.g., Name Entity Recognition Task

Intrinsic evaluation

- Evaluate performance through concrete subtasks at intermediate stages (e.g., word similarity, analogy).
- Faster evaluation speed.
- Difficult to judge whether improvements actually transfer to real tasks.

Extrinsic evaluation

- e.g., Name Entity Recognition Task

Table 4: F1 score on NER task with 50d vectors. *Discrete* is the baseline without word vectors. We use publicly-available vectors for HPCA, HSMN, and CW. See text for details.

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**

Figure 1: Pennington et al. (2014)

How to evaluate word vectors

Intrinsic evaluation

- e.g., Word Analogies: Syntactic, Semantic

Word analogy task: ”a is to b as c is to ?”

- **Semantic example:** Athens : Greece :: Berlin : ___
- **Syntactic example:** dance : dancing :: fly : ___

How to evaluate word vectors

Intrinsic evaluation

- e.g., Word Analogies: Syntactic, Semantic

Table 2: Results on the word analogy task, given as percent accuracy. Underlined scores are best within groups of similarly-sized models; bold scores are best overall. HPCA vectors are publicly available[2]; (i)vLBL results are from (Mnih et al., 2013); skip-gram (SG) and CBOW results are from (Mikolov et al., 2013a,b); we trained SG† and CBOW† using the word2vec tool[3]. See text for details and a description of the SVD models.

Model	Dim.	Size	Sem.	Syn.	Tot.
ivLBL	100	1.5B	55.9	50.1	53.2
HPCA	100	1.6B	4.2	16.4	10.8
GloVe	100	1.6B	<u>67.5</u>	<u>54.3</u>	<u>60.3</u>
SG	300	1B	61	61	61
CBOW	300	1.6B	16.1	52.6	36.1
vLBL	300	1.5B	54.2	<u>64.8</u>	60.0
ivLBL	300	1.5B	65.2	63.0	64.0
GloVe	300	1.6B	<u>80.8</u>	61.5	<u>70.3</u>
SVD	300	6B	6.3	8.1	7.3
SVD-S	300	6B	36.7	46.6	42.1
SVD-L	300	6B	56.6	63.0	60.1
CBOW†	300	6B	63.6	<u>67.4</u>	65.7
SG†	300	6B	73.0	66.0	69.1
GloVe	300	6B	<u>77.4</u>	67.0	<u>71.7</u>
CBOW	1000	6B	57.3	68.9	63.7
SG	1000	6B	66.1	65.1	65.6
SVD-L	300	42B	38.4	58.2	49.2
GloVe	300	42B	**<u>81.9</u>**	**<u>69.3</u>**	**<u>75.0</u>**

Figure 2: Pennington et al. (2014)

How to evaluate word vectors

Intrinsic evaluation

- e.g., Correlation evaluation: calculate the relationship between word vector and human judgments
- Dataset: wordsim353

(`https://aclweb.org/aclwiki/WordSimilarity-353_Test_Collection_(State_of_the_art)`)

Table 3: Spearman rank correlation on word similarity tasks. All vectors are 300-dimensional. The CBOW* vectors are from the `word2vec` website and differ in that they contain phrase vectors.

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

Figure 3: Pennington et al. (2014)

Wrap-up

Conclusion

- Encoding and embedding

Key idea: Word meanings can be represented well by a high-dimensional vector of real numbers

Conclusion

- Encoding and embedding
- Word2vec

Key idea: Word meanings can be represented well by a high-dimensional vector of real numbers

Conclusion

- Encoding and embedding
- Word2vec
- Evaluation

Key idea: Word meanings can be represented well by a high-dimensional vector of real numbers