Lab 2. Word vectors

Jan 22, 2026

Outline

Outline

1 Review

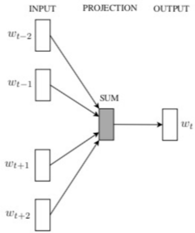
2 Lab2

3 Preview

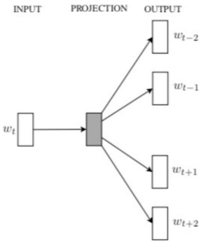
Word2vec

- **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 word o (window, n-gram)
  - Use the similarity of the word vectors for c and o to **calculate the probability** of o given c (or vice versa) $\leftarrow$ Neural network-based model

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 ($P(w_{t+i}|w_t), i \in \{-2, -1, 1, 2\}$)

Focus on **Skip-gram**.

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)

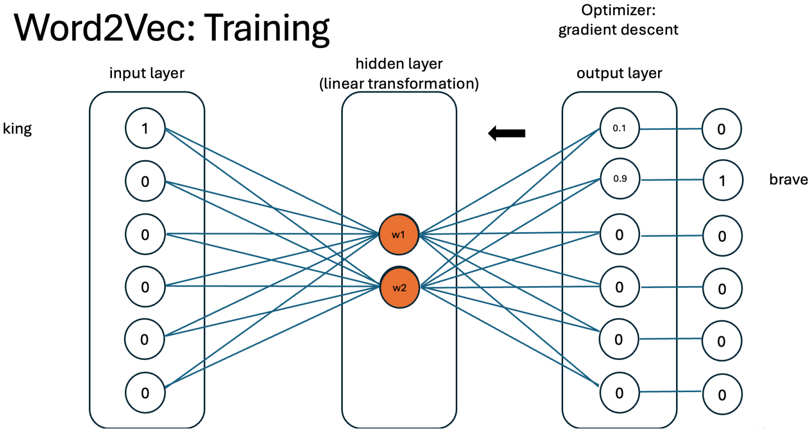
word	one-hot encoding	neighbor	one-hot encoding
king	[1, 0, 0, 0, 0, 0]	brave	[0, 1, 0, 0, 0, 0]
king	[1, 0, 0, 0, 0, 0]	man	[0, 0, 1, 0, 0, 0]
brave	[0, 1, 0, 0, 0, 0]	man	[0, 0, 1, 0, 0, 0]
brave	[0, 1, 0, 0, 0, 0]	king	[1, 0, 0, 0, 0, 0]
man	[0, 0, 1, 0, 0, 0]	king	[1, 0, 0, 0, 0, 0]
man	[0, 0, 1,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

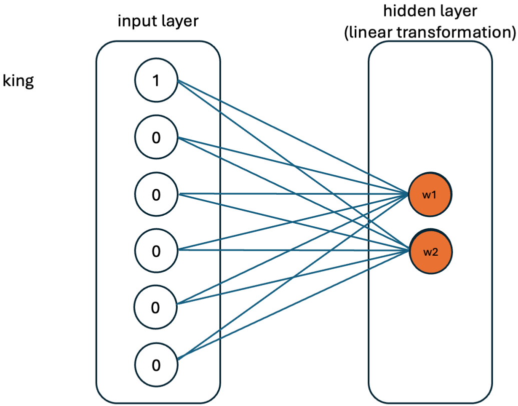
Word2Vec: Input and output

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]

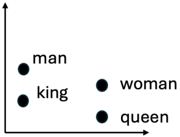
output
[0, 1, 0, 0, 0, 0]
[0, 0, 1, 0, 0, 0]
[0, 0, 1, 0, 0, 0]
[1, 0, 0, 0, 0, 0]
[1, 0, 0, 0, 0, 0]
[0, 1, 0, 0, 0, 0]
[0, 0, 0, 0, 1, 0]
[0, 0, 0, 0, 0, 1]
[0, 0, 0, 1, 0, 0]
[0, 0, 0, 0, 0, 1]
[0, 0, 0, 1, 0, 0]
[0, 0, 0, 0, 1, 0]

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 encoding and dense embedding

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

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

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 (See the appendix in last class's slides.)

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** $\rightarrow$ 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** $\rightarrow$ label $= 1$
  - **Negative (random) words** $\rightarrow$ 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** $\rightarrow$ label $= 1$
  - **Negative (random) words** $\rightarrow$ 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** $\rightarrow$ label $= 1$
  - **Negative (random) words** $\rightarrow$ 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

RIT | Rochester Institute of Technology

4. Compute loss

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

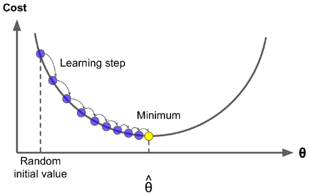
$$\mathcal{L} = -\left(\log \sigma(\vec{v}_c \cdot \vec{u}_{w^+}) + \sum_{i=1}^{k} \log\left(1 - \sigma(\vec{v}_c \cdot \vec{u}_{w_i^-})\right)\right)$$

- 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

RIT Rochester Institute of Technology

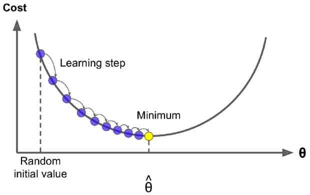
5. Update word vectors

- Optimizer updates parameters based on gradients



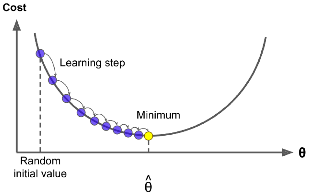
5. Update word vectors

- Optimizer updates parameters based on gradients
- Parameters updated:



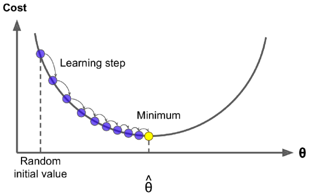
5. Update word vectors

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



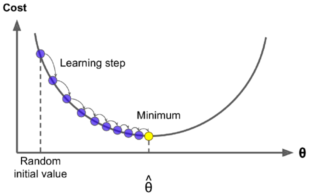
5. Update word vectors

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



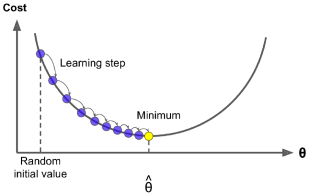
5. Update word vectors

- 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



GloVe: Encoding meaning via co-occurrence ratios

Example: ice vs. steam

- x = a context word (e.g., *solid*, *gas*, *water*, *random*)
- Compare $P(x \mid \text{ice})$ and $P(x \mid \text{steam})$

	x = solid	**x = gas**	**x = water**	**x = random**
$P(x \mid \text{ice})$	large	small	large	small
$P(x \mid \text{steam})$	small	large	large	small
$\frac{P(x\mid\text{ice})}{P(x\mid\text{steam})}$	large	small	≈ 1	≈ 1

These ratio patterns can encode **semantic differences**.

GloVe: Goal

Find word vectors $\vec{w}_{\text{ice}}$, $\vec{w}_{\text{steam}}$ such that:

$$(\vec{w}_{\text{ice}} - \vec{w}_{\text{steam}}) \cdot \vec{w}_x \approx \log \frac{P(x \mid \text{ice})}{P(x \mid \text{steam})}$$

tl;dr

- Neural network models generally outperform count-based models in representing word meaning.
- (We will return to *neural network architectures* next week.)
- **Key question**: **Is this performance gain due solely to neural networks, or to other factors?**
- Today's paper (Levy et al., 2015; Presenter: Leona) examines how *hyperparameters*, which control how algorithms process text, contribute to this improvement.

Outline

1 Review

2 Lab2

3 Preview

Lab Overview

To-do list

- In this lab, we will practice concepts related to *word vectors*, which we covered on Tuesday:
  - Section 1: Count-based model
  - Section 2: Word2Vec
  - Section 3: GloVe
- Please read the guidelines and the provided code carefully.

Evaluation Criteria

- Each section is worth **2 points**.

Section	Credit (2)	Partial (1)	No Credit (0)
1	Complete	Partial	None
2	Complete	Partial	None
3	Complete	Partial	None

Outline

1. Review
2. Lab2
3. Preview

Next Week

- **Tuesday**: Neural network — Presenter: Jacob
- **Thursday**: Lab 3 — PyTorch, Project guide