Introduction to Word Embeddings

Wei Zhao

School of Natural and Computing Sciences
University of Aberdeen, UK

wei.zhao@abdn.ac.uk

May 14, 2025

Acknowledgement

- Many of the slides are adapted from the following lectures:
  - Katja Markert, Heidelberg University, CL students
    - https://www.cl.uni-heidelberg.de/courses/ss19/emb/
  - Richard Socher, Stanford University, CS students
    - https://cs224d.stanford.edu/lectures/
  - Yannick Couzinié, LMU Munich, MATH students
    - https://www.mathematik.uni-muenchen.de/ deckert/teaching/SS17/ATML/media/Word2Vec_slides.pdf

What is word embedding

- Representing a word as a vector of real numbers.
- Embedding is a function from a Vocabulary V to $\mathbb{R}^n$.
  - Corpus: Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna, ...
  - Vocabulary V: [Lorem, ipsum, dolor, ...]
  - Embedding $f : V \rightarrow \mathbb{R}^n$
    - $f(lorem) = (0.1, 0.4, ..., 0.3)$

Why we study word embeddings

- Computer understands human vocabulary as vectors
  - All vector and matrix operations from linear algebra at our disposal
  - word similarity:

```
word_vectors.similarity("love", "adore") = 0.681
```

  - text classification $f : \mathbb{R}^n \rightarrow \{1, 2, \ldots, K\}$
- Building block of ChatGPT

Learning objectives

- how to construct word embeddings
- how to evaluate word embeddings

Outline

Some concepts about counting

- Word co-occurrence: the occurrence of two or more words in a text
- Collocation (statistical association): two words that often occur together in a corpus
  - How often do two words co-occur?
  - Do they co-occur more often than chance (significance test: t-test):
    - Null hypothesis (H0): co-occurrance of w_1 and w_2 is due to chance (no collocation): $P(w_1, w_2) = P(w_1) \cdot P(w_2)$
    - In case we reject H0, w1 and w2 are a collocation.
  - How much mutual information do two words contribute (information theory): $\text{PMI}(w_1, w_2) = \log_2 \left(\frac{P(w_1, w_2)}{P(w_1) \cdot P(w_2)} \right)$

Bigram	freq(w1)	freq(w2)	freq(w1, w2)	t-test	PMI
unsalted butter	24	320	20	4.47	15.19
over many	13484	10570	20	2.24	1.01

Count-based embeddings

Co-occurrence matrix: entry = freq($w1, w2$)

	astronaut	cosmonaut	tomato	...
NASA	4	0	1	...
Roscosmos	0	4	0	...

- f(NASA) = (4,0,1, ...)
- f(Roscosmos) = (0,4,0, ...)
- **Problems**:
  - Long vectors. Length = $|V|$.
  - very high dimensional: require a lot of storage
  - Sparse matrix due to Zipfs law.
  - (near)-synonyms are in different dimensions: astronaut/cosmonaut

```
word_vectors.similarity("NASA", "Roscosmos") = 0
```

Singular value decomposition (SVD)

$$\underset{m \times n}{\hat{X}} \approx \underset{m \times r}{U} \; \underset{r \times r}{S} \; \underset{r \times n}{V^{\mathsf{T}}}$$

$$\hat{X} = \begin{pmatrix} x_{11} & x_{12} & \cdots & x_{1n} \\ x_{21} & x_{22} & \cdots & \\ \vdots & \vdots & \ddots & \\ x_{m1} & & & x_{mn} \end{pmatrix}$$

$$U = \begin{pmatrix} u_{11} & \cdots & u_{1r} \\ \vdots & \ddots & \\ u_{m1} & & u_{mr} \end{pmatrix}$$

$$S = \begin{pmatrix} s_{11} & 0 & \cdots \\ 0 & \ddots & \\ \vdots & & s_{rr} \end{pmatrix}$$

$$V^{\mathsf{T}} = \begin{pmatrix} v_{11} & \cdots & v_{1n} \\ \vdots & \ddots & \\ v_{r1} & & v_{rn} \end{pmatrix}$$

Properties:

- Low-dimensional approximation. r $\ll$ n
- Most important hidden dimensions captured
- Computationally expensive; for *nxm* matrix: $O(mn^2)$
- Difficult to update when we want to assign vectors to new words.

Outline

Next word prediction

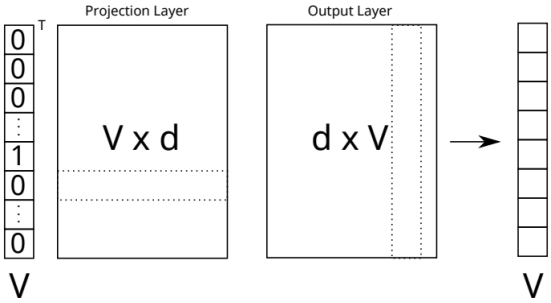
Leipzig is a nice city.

=> (Leipzig, is), (is, a), (a, nice), (nice, city)

```
for word in sentence:
    take the current_word_vector
    predict the next word
    if prediction vector not next_word_vector:
        (i.e. if the prediction wrong)
        do gradient descent
repeat epoch times
```

A simple NN method for next word prediction

- input: Leipzig
- next word: is



- $\mathbf{v}_I(\text{Leipzig})^\top \cdot \mathbf{v}_O(w) = \text{score for word } w \in V$
- If $\arg\max_{w \in V} \left(\mathbf{v}_I(w_t)^\top \cdot \mathbf{v}_O(w)\right) \neq w_{t+1}$, then do gradient descent.

Gradient descent

- $p(is|Leipzig) = \frac{\exp(\mathbf{v}(\text{Leipzig})^\top \mathbf{v}(\text{is}))}{\sum_{w \in V} \exp(\mathbf{v}(\text{Leipzig})^\top \mathbf{v}(\text{w}))}$
- $\mathbf{v}_{\text{is}}$ starting with a random vector
- Take the derivative with respect to $\mathbf{v}_{\text{is}}$

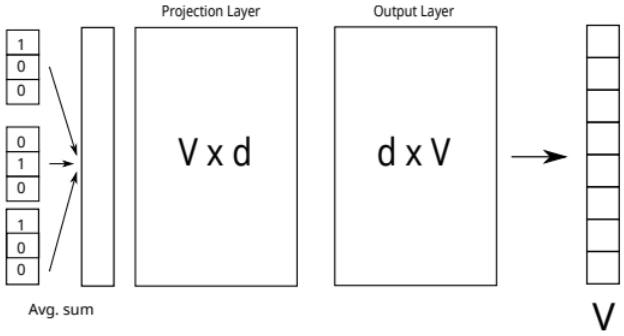
$$\nabla_\theta J_t(\theta) = \begin{bmatrix} 0 \\ \vdots \\ \nabla_{\mathbf{v}_{\text{is}}} \\ \vdots \\ 0 \end{bmatrix} \in \mathbb{R}^{dV}$$

- $\nabla_\theta J_t(\theta)$ is very sparse

Two popular methods: CBOW and Skip-gram

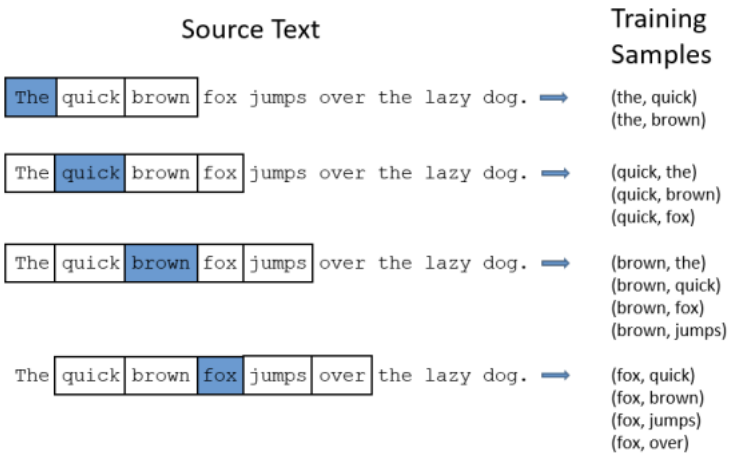
- Previous idea: predict next word based on last word (Leipzig, is)
- Predict center word from context (CBOW)
- Predict context from center word (Skip-gram)

Continuous Bag-of-words (CBOW)



- Predict center word from context words as input.
- Leipzig is a nice
- Change in pipeline: method input from last word into context words
- $\mathbf{v}_I = \frac{1}{C} \sum_{c \in C} \mathbf{v}_c$, where $C = \{\text{Leipzig}, \text{is}, \text{nice}\}$.

Skip-gram



- Predict context words from center word as input

Comparison of CBOW and Skip-gram

- Computational costs:
  - CBOW: $O \propto (|C| \times d + d \times V)$
  - Skip-gram: $O \propto |C| \times (d + d \times V)$
  - CBOW is faster
  - Skip-gram good with smaller corpus and rarer words
  - Why? Rare words as center words are less often, so their vectors receive fewer updates.
    - Example: relics from the latterday pioneers
    - Skip-gram: (latterday, from), (latterday, the), (latterday, pioneers)
    - CBOW: (the, latterday)

Outline

Evaluation of word embeddings

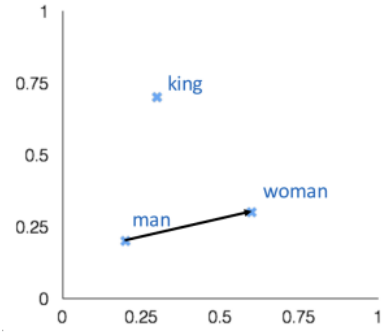
Intrinsic evaluation:

- evaluation on word similarity and analogy tasks
- not clear how meaningful for real-world tasks

Extrinsic evaluation:

- plug-in pretrained embeddings as features for different NLP tasks

Intrinsic evaluation



Word vector analogies:

- A is to B as what C is to?
- e.g., man is to women as what king is to?
- Task: v(woman) - v(man) + v(king) = X
- Look for a word whose vector is closest to the resulting vector X
- $w = \arg\max_{w \in V} \frac{(v(man) - v(woman) + v(king))^T v(w)}{\|v(man) - v(woman) + v(king)\| \cdot \|v(w)\|}$
- Correct if w = 'queen', otherwise wrong.

Conclusion Remarks

- Open-source LLMs
- Embeddings are numbers in a blackbox, how can we interpret them?
- Biases in embeddings: low-resource languages, gender stereotypes

References I