

Natural Language Processing with Deep Learning

CS224N/Ling284



Christopher Manning

Lecture 1: Introduction and Word Vectors

ChatGPT, GPT-4, and more



A Hey please draft a polite mail to explain my boss Jeremy that I would not be able to come to office for next 2 days because my 9 year son Peter is angry with me that I am not giving him much time. So I have plans to take him out for a trip. I will return and complete my work by meeting deadline.

Dear Jeremy,

I hope this email finds you well. I'm writing to let you know that I will not be able to come into the office for the next two days.

My 9-year-old son Peter is feeling upset with me because he feels like I haven't been giving him enough attention lately. In order to make it up to him, I've decided to take him on a little trip to help us reconnect.

I understand that this may cause some inconvenience, but I promise to make up for any lost time by working extra hard to meet any deadlines when I return.

Thank you for your understanding.

Best regards,
[Your Name]

User What is unusual about this image?

Source: Bloomberg

GPT-4 The unusual thing about this image is that a man is ironing clothes on an ironing board attached to the roof of a moving taxi.

Representing words as discrete symbols

In traditional NLP, we regard words as discrete symbols:
hotel, conference, motel – a **localist** representation

Means one 1, the rest 0s

Such symbols for words can be represented by **one-hot vectors**:

motel = [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]

Vector dimension = number of words in vocabulary (e.g., 500,000+)

Problem with words as discrete symbols

Example: in web search, if a user searches for "Seattle motel", we would like to match documents containing "Seattle hotel"

But:

motel = [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]

These two vectors are **orthogonal**

There is no natural notion of **similarity** for one-hot vectors!

Solution:

- Could try to rely on WordNet's list of synonyms to get similarity?
 - But it is well-known to fail badly: incompleteness, etc.
- **Instead: learn to encode similarity in the vectors themselves**

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*" (J. R. Firth 1957: 11)
 - 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).
- We use the many contexts of w to build up a representation of w



...government debt problems turning into **banking** crises as happened in 2009...
 ...saying that Europe needs unified **banking** regulation to replace the hodgepodge...
 ...India has just given its **banking** system a shot in the arm...

These **context words** will represent **banking**

22

Word vectors

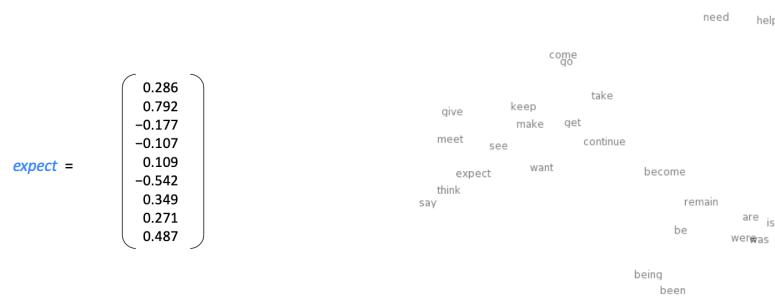
We will build a dense vector for each word, chosen so that it is similar to vectors of words that appear in similar contexts, measuring similarity as the vector dot (scalar) product

$$\text{banking} = \begin{pmatrix} 0.286 \\ 0.792 \\ -0.177 \\ -0.107 \\ 0.109 \\ -0.542 \\ 0.349 \\ 0.271 \end{pmatrix} \quad \text{monetary} = \begin{pmatrix} 0.413 \\ 0.582 \\ -0.007 \\ 0.247 \\ 0.216 \\ -0.718 \\ 0.147 \\ 0.051 \end{pmatrix}$$

Note: **word vectors** are also called **(word) embeddings** or **(neural) word representations**
 They are a **distributed** representation

23

Word meaning as a neural word vector – visualization



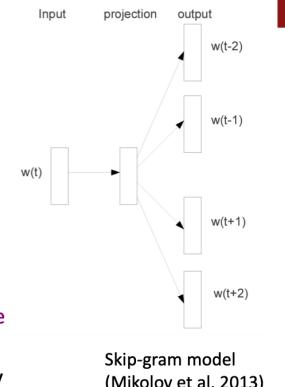
24

3. Word2vec: Overview

Word2vec is a framework for learning word vectors
 (Mikolov et al. 2013)

Idea:

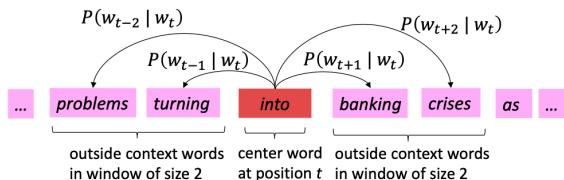
- We have a large corpus ("body") of text: a long list of words
- Every word in a fixed vocabulary is represented by a **vector**
- Go through each position t in the text, which has a center word c and context ("outside") words o
- Use the **similarity of the word vectors** for c and o to **calculate the probability** of o given c (or vice versa)
- Keep adjusting the word vectors to maximize this probability



25

Word2Vec Overview

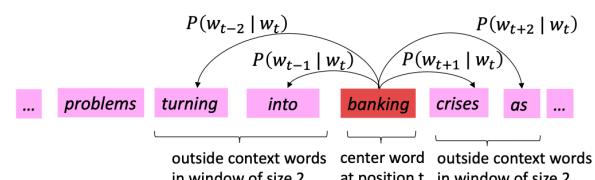
Example windows and process for computing $P(w_{t+j} | w_t)$



26

Word2Vec Overview

Example windows and process for computing $P(w_{t+j} | w_t)$



27

Word2vec: objective function

For each position $t = 1, \dots, T$, predict context words within a window of fixed size m , given center word w_t . Data likelihood:

$$\text{Likelihood} = L(\theta) = \prod_{t=1}^T \prod_{\substack{-m \leq j \leq m \\ j \neq 0}} P(w_{t+j} | w_t; \theta)$$

θ is all variables to be optimized

sometimes called a *cost* or *loss* function

The objective function $J(\theta)$ is the (average) negative log likelihood:

$$J(\theta) = -\frac{1}{T} \log L(\theta) = -\frac{1}{T} \sum_{t=1}^T \sum_{\substack{-m \leq j \leq m \\ j \neq 0}} \log P(w_{t+j} | w_t; \theta)$$

Minimizing objective function \Leftrightarrow Maximizing predictive accuracy

28

Word2vec: objective function

- We want to minimize the objective function:

$$J(\theta) = -\frac{1}{T} \sum_{t=1}^T \sum_{\substack{-m \leq j \leq m \\ j \neq 0}} \log P(w_{t+j} | w_t; \theta)$$

- Question: How to calculate $P(w_{t+j} | w_t; \theta)$?

- Answer: We will use two vectors per word w :

- v_w when w is a center word
- u_w when w is a context word

} These word vectors are subparts of the big vector of all parameters θ

- Then for a center word c and a context word o :

$$P(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{w \in V} \exp(u_w^T v_c)}$$

29

Word2vec: prediction function

$$P(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{w \in V} \exp(u_w^T v_c)}$$

② Exponentiation makes anything positive
 ① Dot product compares similarity of o and c .
 $u^T v = u \cdot v = \sum_{i=1}^n u_i v_i$
 Larger dot product = larger probability
 ③ Normalize over entire vocabulary to give probability distribution

- This is an example of the **softmax function** $\mathbb{R}^n \rightarrow (0,1)^n$
- $$\text{softmax}(x_i) = \frac{\exp(x_i)}{\sum_{j=1}^n \exp(x_j)} = p_i$$
- The softmax function maps arbitrary values x_i to a probability distribution p_i
 - "max" because amplifies probability of largest x_i
 - "soft" because still assigns some probability to smaller x_i
 - Frequently used in Deep Learning

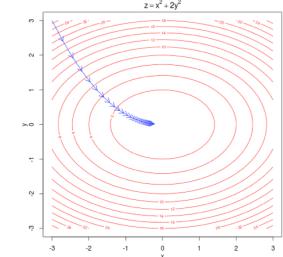
30

To train the model: Optimize value of parameters to minimize loss

To train a model, we gradually adjust parameters to minimize a loss

- Recall: θ represents **all** the model parameters, in one long vector
- In our case, with d -dimensional vectors and V -many words, we have $\theta \in \mathbb{R}^{2dV}$
- Remember: every word has two vectors

$$\theta = \begin{bmatrix} v_{aardvark} \\ v_a \\ \vdots \\ v_{zebra} \\ u_{aardvark} \\ u_a \\ \vdots \\ u_{zebra} \end{bmatrix} \in \mathbb{R}^{2dV}$$



- We optimize these parameters by walking down the gradient (see right figure)
- We compute **all** vector gradients!

31

Natural Language Processing with Deep Learning

CS224N/Ling284

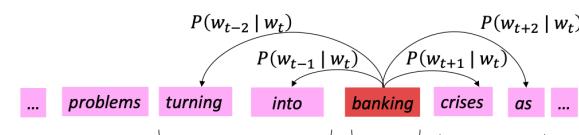


Christopher Manning

Lecture 2: Word Vectors, Word Senses, and Neural Classifiers

3. Review: Main idea of word2vec

- Start with random word vectors
- Iterate through each word position in the whole corpus
- Try to predict surrounding words using word vectors: $P(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{w \in V} \exp(u_w^T v_c)}$



- Learning:** Update vectors so they can predict actual surrounding words better
- Doing no more than this, this algorithm learns word vectors that capture well word similarity and meaningful directions in a word space!



Magic!

7

Word2vec parameters

$$\begin{bmatrix} \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \end{bmatrix} \quad \begin{bmatrix} \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \end{bmatrix}$$

U outside V center

... and computations

$$\begin{array}{ll} \begin{array}{c} \cdot \\ \cdot \\ \cdot \\ \cdot \\ \cdot \end{array} & \begin{array}{c} \cdot \\ \cdot \\ \cdot \\ \cdot \\ \cdot \end{array} \\ U.v_4^T & \text{softmax}(U.v_4^T) \\ \text{dot product} & \text{probabilities} \end{array}$$

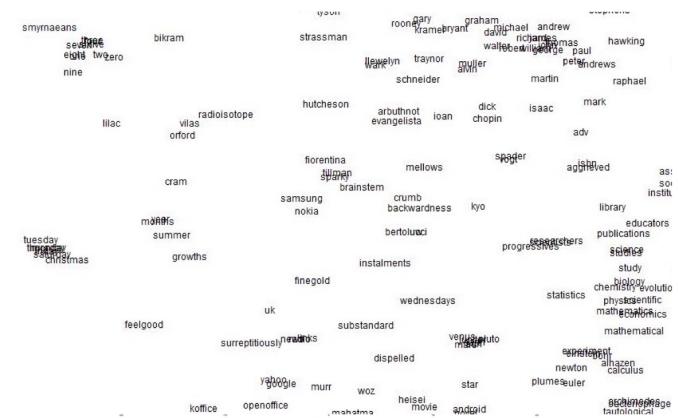
"Bag of words" model!

The model makes the same predictions at each position

We want a model that gives a reasonably high probability estimate to *all* words that occur in the context (at all often)

8

Word2vec maximizes objective by putting similar words nearby in space



9

4. Word2vec algorithm family: More details

[Mikolov et al. 2013: "Distributed Representations of Words and Phrases and their Compositionalities"]

Why two vectors? → Easier optimization. Average both at the end

- But can implement the algorithm with just one vector per word ... and it helps a bit

Two model variants:

1. Skip-grams (SG)

Predict context ("outside") words (position independent) given center word

2. Continuous Bag of Words (CBOW)

Predict center word from (bag of) context words

We presented: Skip-gram model

Loss functions for training:

1. Naïve softmax (simple but expensive loss function, when many output classes)
2. More optimized variants like hierarchical softmax
3. Negative sampling

So far, we explained naïve softmax

10

The skip-gram model with negative sampling

- The normalization term is computationally expensive (when many output classes):

$$P(o|c) = \frac{\exp(u_o^T v_c)}{\sum_{w \in V} \exp(u_w^T v_c)}$$

A big sum over many words

- Hence, in standard word2vec, you implement the skip-gram model with **negative sampling**

- Idea: train binary logistic regressions to differentiate a true pair (center word and a word in its context window) versus several "noise" pairs (the center word paired with a random word)

11

7. Word senses and word sense ambiguity

- Most words have lots of meanings!
 - Especially common words
 - Especially words that have existed for a long time
- Example: **pike**
- Does one vector capture all these meanings or do we have a mess?

30

pike

- A sharp point or staff
- A type of elongated fish
- A railroad line or system
- A type of road
- The future (coming down the pike)
- A type of body position (as in diving)
- To kill or pierce with a pike
- To make one's way (pike along)
- In Australian English, pike means to pull out from doing something:
 - *I reckon he could have climbed that cliff, but he piked!*

31

8. Deep Learning Classification: Named Entity Recognition (NER)

- The task: **find** and **classify** names in text, by labeling word tokens, for example:

Last night , Paris Hilton wowed in a sequin gown .

PER PER

Samuel Quinn was arrested in the Hilton Hotel in Paris in April 1989 .

PER PER LOC LOC LOC DATE DATE

- Possible uses:
 - Tracking mentions of particular entities in documents
 - For question answering, answers are usually named entities
 - Relating sentiment analysis to the entity under discussion
 - Often followed by Entity Linking/Canonicalization into a Knowledge Base such as Wikidata

34

Simple NER: Window classification using binary logistic classifier

- **Idea:** **classify each word in its context window** of neighboring words
- Train logistic classifier on hand-labeled data to classify center word {yes/no} for each class based on a **concatenation of word vectors** in a window
 - Really, we usually use multi-class softmax, but we're trying to keep it simple ☺
- **Example:** Classify "Paris" as +/- location in context of sentence with window length 2:

the museums in Paris are amazing to see .

$$X_{\text{window}} = [x_{\text{museums}} \quad x_{\text{in}} \quad x_{\text{Paris}} \quad x_{\text{are}} \quad x_{\text{amazing}}]^T$$

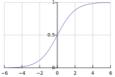
- Resulting vector $x_{\text{window}} = x \in \mathbb{R}^{5d}$
- To classify all words: run classifier for each class on the vector centered on each word in the sentence

35

NER: Binary classification for center word being location

- We do supervised training and want high score if it's a location

$$J_t(\theta) = \sigma(s) = \frac{1}{1 + e^{-s}}$$



predicted model probability of class

$$s = \mathbf{u}^T \mathbf{h}$$

$$\mathbf{h} = f(\mathbf{W}\mathbf{x} + \mathbf{b})$$

f = Some element-wise non-linear function, e.g., logistic, tanh, ReLU

$$\mathbf{x} \text{ (input)} \in \mathbb{R}^{5d}$$

$$\mathbf{x} = [x_{\text{museums}} \quad x_{\text{in}} \quad x_{\text{Paris}} \quad x_{\text{are}} \quad x_{\text{amazing}}]$$

Embedding of 1-hot words