Word Embedding: Word2Vec

Yu Meng
University of Virginia
yumeng5@virginia.edu

Sep 18, 2024

University of Virginia

Join at slido.com #2580 145

Reminder

- Project proposal is due Friday 11:59pm!
- We have set up Rivanna access (GPU compute) for everyone; an instruction will be released

Overview of Course Contents

- Week 1: Logistics & Overview
- Week 2: N-gram Language Models
- Week 3: Word Senses, Semantics & Classic Word Representations
- Week 4: Word Embeddings
- Week 5: Sequence Modeling and Transformers
- Week 6-7: Language Modeling with Transformers (Pretraining + Fine-tuning)
- Week 8: Large Language Models (LLMs) & In-context Learning
- Week 9-10: Knowledge in LLMs and Retrieval-Augmented Generation (RAG)
- Week 11: LLM Alignment
- Week 12: Language Agents
- Week 13: Recap + Future of NLP
- Week 15 (after Thanksgiving): Project Presentations

(Recap) Pointwise Mutual Information (PMI)

- PMI compares the probability of two words co-occurring with the probabilities of the words occurring independently

$$\mathrm{PMI} = \log_2 \frac{p(w_1, w_2)}{p(w_1)p(w_2)} = \log_2 \frac{\#(w_1, w_2)}{\#(w_1)\#(w_2)}$$

- PMI = 0: Two words co-occur as expected by chance => no particular association
- PMI > 0: Two words co-occur more often than by chance => the higher the PMI, the stronger the association between the words
- PMI < 0: Two words co-occur less often than expected by chance => negative associations; not much actionable insight
- Positive PMI (PPMI): replaces all negative PMI values with zero

$$\mathrm{PPMI} = \max\left(\log_2 \frac{p(w_1, w_2)}{p(w_1)p(w_2)}, 0\right)$$

UNIVERSITY of VIRGINIA

Join at slido.com #2580 145

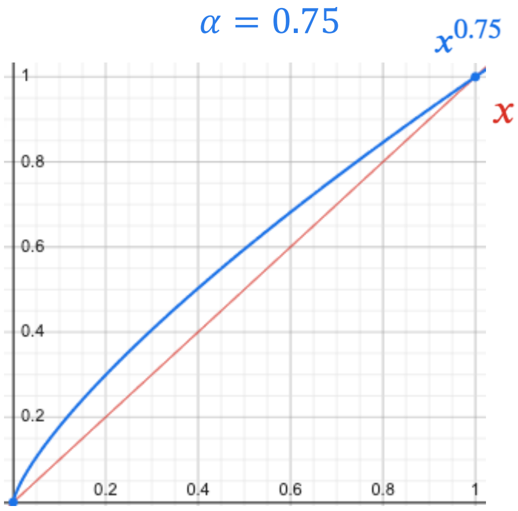
(Recap) PPMI with Power Smoothing

Power smoothing: Manually boost low probabilities by raising to a power α

$$\mathrm{PPMI} = \max\left(\log_2 \frac{p(w_1, w_2)}{p(w_1)p(w_2)}, 0\right)$$

Original: $$p(w) = \frac{\#(w)}{\sum_{w' \in \mathcal{V}} \#(w')}$$

Power smoothed: ($\alpha < 1$) $$p_\alpha(w) = \frac{\#(w)^\alpha}{\sum_{w' \in \mathcal{V}} \#(w')^\alpha}$$



(Recap) PPMI with Add-*k* Smoothing

- Another way of increasing the counts of rare occurrences is to apply add-*k* smoothing

	computer	data	result	pie	sugar
cherry	2	8	9	442	25
strawberry	0	0	1	60	19
digital	1670	1683	85	5	4
information	3325	3982	378	5	13

Add a constant *k* to all counts

- The larger the *k* (*k* can be larger than 1), the more we boost the probability of rare occurrences

(Recap) TF-IDF vs. PMI Weighting

- TF-IDF
  - Measures the importance of a word in a document relative to other documents (corpus)
  - Context granularity: document level
  - Based on heuristics
  - High TF-IDF = frequent in a document but infrequent across the corpus
- PMI:
  - Measures the strength of association between two words
  - Context granularity: word pair level (usually based on local context windows)
  - Based on probability assumptions
  - High PMI = words co-occur more often than expected by chance, a strong association

(Recap) Count-based Vector Limitations

- Count-based vectors are **sparse** (lots of zeros)
  - Zero values in the vectors do not carry any semantics
- Count-based vectors are **long** (many dimensions)
  - Vector dimension = vocabulary size (usually > 10K)
  - “Curse of dimensionality”: metrics (e.g. cosine) become less meaningful in high dimensions

	aardvark	...	**computer**	**data**	**result**	**pie**	**sugar**	...
cherry	0	...	2	8	9	442	25	...
strawberry	0	...	0	0	1	60	19	...
digital	0	...	1670	1683	85	5	4	...
information	0	...	3325	3982	378	5	13	...

Many more words!

(Recap) Dense Vectors

- More efficient & effective vector representations?
- **Dense** vectors!
  - Most/all dimensions in the vectors are non-zero
  - Usually floating-point numbers; each dimension could be either positive or negative
  - Dimension much smaller than sparse vectors (i.e., << 10K)
- Also called "**distributed** representations"
  - The information is **distributed** across multiple units/dimensions
  - Each unit/dimension participates in representing multiple pieces of information
  - Analogous to human brains: the brain stores and processes information in a distributed manner: instead of having a single neuron/region represent a concept, information is represented across a network of neurons

(Recap) Dense Vector Example

- One dimension might (partly) contribute to distinguishing animals (“cat” “dog”) from vehicles (“car” “truck”)
- One dimension might (partly) capture some aspect of size
- Another might (partly) represent formality or emotional tone
- …
- Each of these dimensions is not exclusively responsible for any single concept, but together, they combine to form a rich and nuanced representation of words!

$$\boldsymbol{v}_{\text{good}} = [-1.34, 2.58, 0.37, 4.32, -3.21, \ldots]$$
$$\boldsymbol{v}_{\text{nice}} = [-0.58, 1.97, 0.20, 3.13, -2.58, \ldots]$$

Only showing two decimal places
(typically they are floating point numbers!)

(Recap) Dense Vectors Pros & Cons

- **(+) Compactness**: Represent a large number of concepts using fewer resources (richer semantic information per dimension); easier to use as features to neural networks
- **(+) Robustness**: Information is spread across many dimensions => more robust to the randomness/noise in individual units
- **(+) Scalability & Generalization**: Efficiently handle large-scale data and generalize to various applications
- **(-) Lack of Interpretability**: (Unlike sparse vectors) difficult to assign a clear meaning to individual dimensions, making model interpretation challenging

Agenda

- Sparse vs. Dense Vectors
- Word Embeddings: Overview
- Word2Vec Training
- Word Embedding Properties & Evaluation

Distributional Hypothesis

- Words that occur in similar contexts tend to have similar meanings
- A word's meaning is largely defined by the company it keeps (its context)
- Example: suppose we don’t know the meaning of “Ong choy” but see the following:
  - Ong choy is delicious **sautéed with garlic**
  - Ong choy is superb **over rice**
  - … ong choy **leaves** with **salty** sauces
- And we’ve seen the following contexts:
  - … spinach **sautéed with garlic over rice**
  - … chard stems and **leaves** are **delicious**
  - … collard greens and other **salty** leafy greens
- Ong choy = water spinach!

Example source: https://web.stanford.edu/~jurafsky/slp3/slides/vectorsemantics2024.pdf

Word Embeddings: General Idea

- Learn dense vector representations of words based on distributional hypothesis
- Semantically similar words (based on context similarity) will have similar vector representations
- **Embedding**: a mapping that takes elements from one space and represents them in a different space

$$
\begin{aligned}
\boldsymbol{v}_{\text{to}} &= [1,0,0,0,0,0,\dots] \\
\boldsymbol{v}_{\text{by}} &= [0,1,0,0,0,0,\dots] \\
\boldsymbol{v}_{\text{that}} &= [0,0,1,0,0,0,\dots] \\
\boldsymbol{v}_{\text{good}} &= [0,0,0,1,0,0,\dots] \\
\boldsymbol{v}_{\text{nice}} &= [0,0,0,0,1,0,\dots] \\
\boldsymbol{v}_{\text{bad}} &= [0,0,0,0,0,1,\dots]
\end{aligned}
$$

2D visualization of a word embedding space

Figure source: https://web.stanford.edu/~jurafsky/slp3/6.pdf

Learning Word Embeddings

- Assume a large text collection (e.g., Wikipedia)
- Hope to learn similar word embeddings for words occurring in similar contexts
- Construct a prediction task: use a center word's embedding to predict its contexts!
- Intuition: If two words have similar embeddings, they will predict similar contexts, thus being semantically similar!

Predicted contexts

$\boldsymbol{v}_{\text{ong choy}}$ → sautéed, garlic, rice, salty, leaves, …

Predicted contexts

$\boldsymbol{v}_{\text{spinach}}$ → sautéed, garlic, rice, salty, leaves, …

Word Embedding Is Self-Supervised Learning

- **Self-supervised learning**: a model learns to predict parts of its input from other parts of the same input

Input: *Ong choy is superb over rice*

Prediction task: Ong choy → is, superb, over, rice

- Self-supervised learning vs. supervised learning:
  - Self-supervised learning: **no human-labeled data** – the model learns from unlabeled data by generating supervision through the structure of the data itself
  - Supervised learning: **use human-labeled data** – the model learns from human annotated input-label pairs

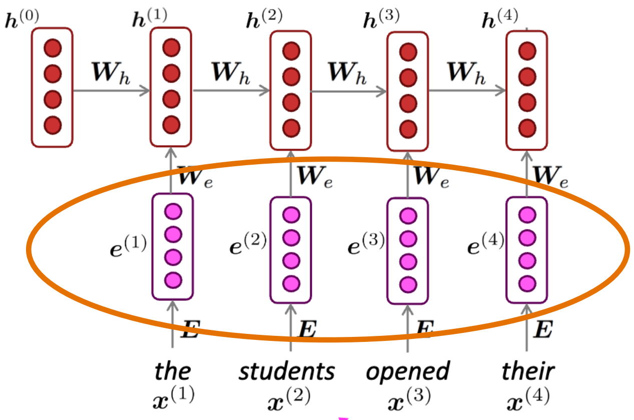
Word Embedding as Input Features

Word embeddings are commonly used as input features to language models

hidden states
$\boldsymbol{h}^{(t)} = \sigma\left(\boldsymbol{W}_h \boldsymbol{h}^{(t-1)} + \boldsymbol{W}_e \boldsymbol{e}^{(t)} + \boldsymbol{b}_1\right)$
$\boldsymbol{h}^{(0)}$ is the initial hidden state

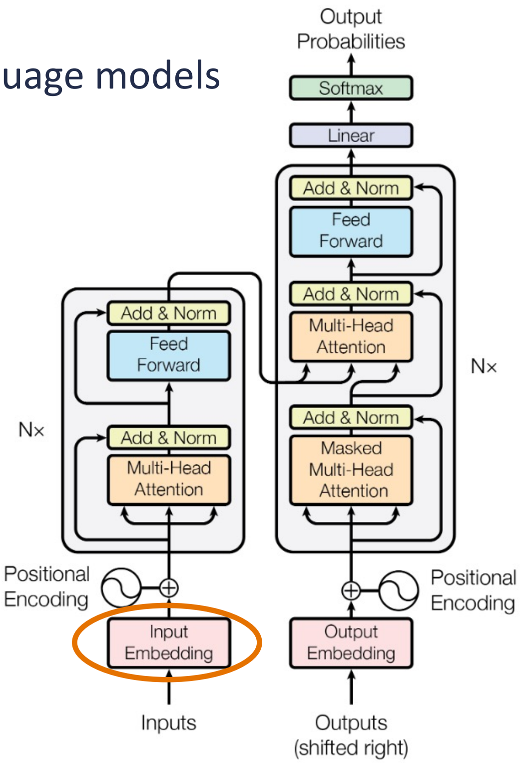
word embeddings
$\boldsymbol{e}^{(t)} = \boldsymbol{E}\boldsymbol{x}^{(t)}$

words / one-hot vectors
$\boldsymbol{x}^{(t)} \in \mathbb{R}^{|V|}$



***Note**: this input sequence could be much longer now!*

RNN Language Model: https://web.stanford.edu/class/cs224n/slides/cs224n-spr2024-lecture05-rnnlm.pdf



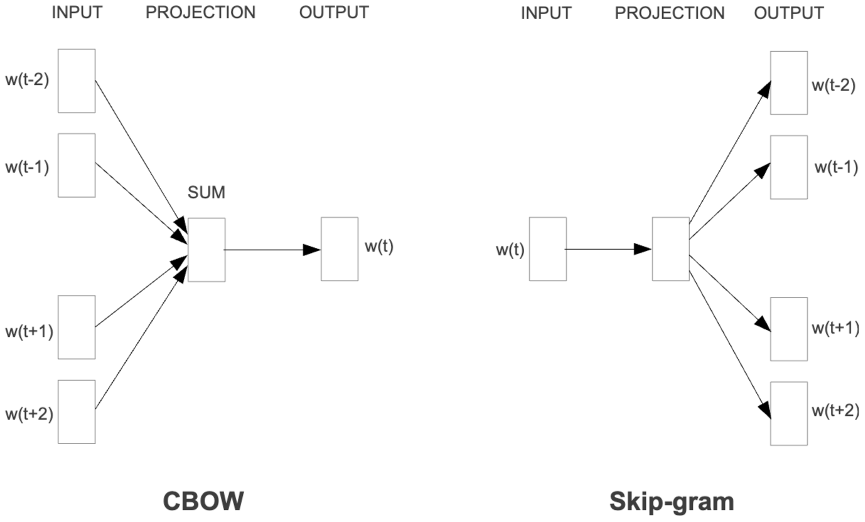
Transformer: https://arxiv.org/pdf/1706.03762

Agenda

- Sparse vs. Dense Vectors
- Word Embeddings: Overview
- Word2Vec Training
- Word Embedding Properties & Evaluation

Word2Vec Overview

- The earliest & most well-known word embedding learning method (published in 2013)
- Two variants: Skip-gram and CBOW (Continuous Bag-of-Words)
- We will mainly cover Skip-gram in this lecture



Word2Vec paper: https://arxiv.org/pdf/1301.3781

Word2Vec Setting

- Input: a corpus D – the larger, the better!
- Training data: word-context pairs (w, c) where w is a center word, and c is a context word
  - Each word in the corpus can act as center word
  - Context words = neighboring words of the center word in a local context window ($\pm l$ words)
- Parameters to learn: $\boldsymbol{\theta} = \{\boldsymbol{v}_w, \boldsymbol{v}_c\}$ – each word has two vectors (center word representation & context word representation)
- The center word representations $\boldsymbol{v}_w$ are usually used as the final word embeddings
- Number of parameters to store: $d \times |V|$
  - d is the embedding dimension; usually 100-300
  - $|V|$ is the vocabulary size; usually > 10K
  - Sparse vector representations will have $|V|^2$ parameters!

Word2Vec Training Data Example

- Input sentence: “there is a cat on the mat”
- Suppose context window size = 2
- Word-context pairs as training data:
  - (there, is), (there, a)
  - (is, there), (is, a), (is, cat)
  - (a, there), (a, is), (a, cat), (a, on)
  - (cat, is), (cat, a), (cat, on), (cat, the)
  - (on, a), (on, cat), (on, the), (on, mat)
  - (the, cat), (the, on), (the, mat)
  - (mat, on), (mat, the)

there is a cat on the mat
there is a cat on the mat
there is a cat on the mat
there is a cat on the mat
there is a cat on the mat
there is a cat on the mat
there is a cat on the mat

- “Skip-gram”: skipping over some context words to predict the others!
- Training data completely derived from the raw corpus (no human labels!)

Word2Vec Objective (Skip-gram)

- Intuition: predict the contexts words using the center word (semantically similar center words will predict similar contexts words)
- Objective: using the parameters $\boldsymbol{\theta} = \{\boldsymbol{v_w}, \boldsymbol{v_c}\}$ to maximize the probability of predicting the context word c using the center word w

$$\max_{\boldsymbol{\theta}} \prod_{(w,c)\in\mathcal{D}} \boxed{p_{\boldsymbol{\theta}}(c|w)}$$

Probability expressed as a function of the model parameters

- How to parametrize the probability?

Word2Vec Probability Parametrization

- Word2Vec objective: $\max_{\boldsymbol{\theta}} \prod_{(w,c)\in\mathcal{D}} p_{\boldsymbol{\theta}}(c|w)$
- Assume the log probability (i.e., logit) is proportional to vector dot product

$$\log p_{\boldsymbol{\theta}}(c|w) \propto \boldsymbol{v}_c \cdot \boldsymbol{v}_w$$

- Rationale: a larger vector dot product *can* indicate a higher vector similarity
- Why not use cosine similarity?
  - Cosine similarity is a non-linear function; more complicated to optimize than dot product
  - With advanced optimization techniques, optimizing cosine similarity is more beneficial (Meng et al.)

Spherical Text Embedding: https://arxiv.org/pdf/1911.01196

Word2Vec Parameterized Objective

- Word2Vec objective: $$\max_{\boldsymbol{\theta}} \prod_{(w,c)\in\mathcal{D}} p_{\boldsymbol{\theta}}(c|w)$$
- Assume the log probability (i.e., logit) is proportional to vector dot product
$$\log p_{\boldsymbol{\theta}}(c|w) \propto \boldsymbol{v}_c \cdot \boldsymbol{v}_w$$
- The final probability distribution is given by the softmax function:
$$p_{\boldsymbol{\theta}}(c|w) = \frac{\exp(\boldsymbol{v}_c \cdot \boldsymbol{v}_w)}{\sum_{c'\in|\mathcal{V}|} \exp(\boldsymbol{v}_{c'} \cdot \boldsymbol{v}_w)} \Rightarrow \sum_{c'\in|\mathcal{V}|} p_{\boldsymbol{\theta}}(c'|w) = 1$$
- Word2Vec objective (log-scale):
$$\max_{\boldsymbol{\theta}} \sum_{(w,c)\in\mathcal{D}} \log p_{\boldsymbol{\theta}}(c|w) = \sum_{(w,c)\in\mathcal{D}} \left(\boldsymbol{v}_c \cdot \boldsymbol{v}_w - \log \sum_{c'\in|\mathcal{V}|} \exp(\boldsymbol{v}_{c'} \cdot \boldsymbol{v}_w) \right)$$

Word2Vec Negative Sampling

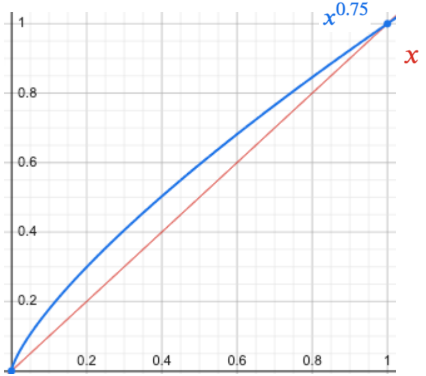
- Challenges with the original objective: Sum over the entire vocabulary – expensive!

$$\max_{\boldsymbol{\theta}} \sum_{(w,c)\in\mathcal{D}} \log p_{\boldsymbol{\theta}}(c|w) = \sum_{(w,c)\in\mathcal{D}} \left(\boldsymbol{v}_c \cdot \boldsymbol{v}_w - \log \boxed{\sum_{c'\in|\mathcal{V}|}} \exp(\boldsymbol{v}_{c'} \cdot \boldsymbol{v}_w) \right)$$

- Randomly sample a few negative terms from the vocabulary to form a negative set N
- How to sample negatives? Based on the (power-smoothed) unigram distribution

$$p_{\text{neg}}(w) \propto \left(\frac{\#(w)}{\sum_{w'\in\mathcal{V}} \#(w')} \right)^{0.75}$$

Rare words get a bit boost in sampling probability

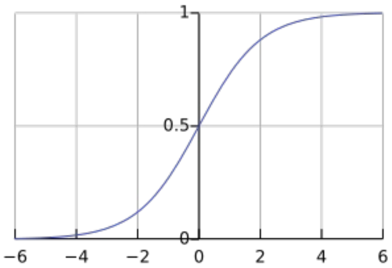


Word2Vec Negative Sampling

Join at
slido.com
#2580 145

- Formulate a binary classification task; predict whether (w, c) is a real context pair:

$$p_{\boldsymbol{\theta}}(\text{True}|c, w) = \sigma(\boldsymbol{v}_c \cdot \boldsymbol{v}_w) = \frac{1}{1 + \exp(-\boldsymbol{v}_c \cdot \boldsymbol{v}_w)}$$



$y = \sigma(x)$

- Maximize the binary classification probability for real context pairs, and minimize for negative (random) pairs

$$\max_{\boldsymbol{\theta}} \log \sigma(\underbrace{\boldsymbol{v}_c \cdot \boldsymbol{v}_w}_{\text{Real context pair}}) - \sum_{c' \in \mathcal{N}} \log \sigma(\underbrace{\boldsymbol{v}_{c'} \cdot \boldsymbol{v}_w}_{\text{Negative context pair}})$$

Word2Vec Optimization

- How to optimize the following objective?

$$\max_{\boldsymbol{\theta}} \log \sigma(\boldsymbol{v}_c \cdot \boldsymbol{v}_w) - \sum_{c' \in \mathcal{N}} \log \sigma(\boldsymbol{v}_{c'} \cdot \boldsymbol{v}_w)$$

- Stochastic gradient descent (SGD)!
- First, initialize parameters $\boldsymbol{\theta} = \{\boldsymbol{v}_w, \boldsymbol{v}_c\}$ with random d-dimensional vectors
- In each step: update parameters in the direction of the gradient of the objective (weighted by the learning rate)

$$\boldsymbol{\theta}^{(t+1)} \leftarrow \boldsymbol{\theta}^{(t)} - \eta \nabla_{\boldsymbol{\theta}} \mathcal{L}\big|_{\boldsymbol{\theta}=\boldsymbol{\theta}^{(t)}}$$

Learning rate — Loss function

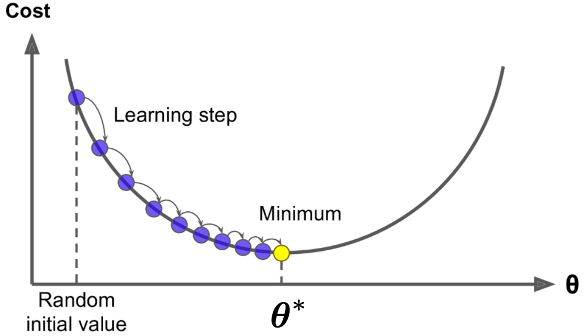


Figure source: https://web.stanford.edu/class/cs224n/slides/cs224n-spr2024-lecture02-wordvecs2.pdf

Word2Vec Hyperparameters

- Word embedding dimension d (usually 100-300)
  - Larger d provides richer vector semantics
  - Extremely large d suffers from inefficiency and curse of dimensionality
- Local context window size l (usually 5-10)
  - Smaller l learns from immediately nearby words – more syntactic information
  - Bigger l learns from longer-ranged contexts – more semantic/topical information
- Number of negative samples k (usually 5-10)
  - Larger k usually makes training more stable but also more costly
- Learning rate η (usually 0.02-0.05)

Thank You!

Yu Meng
University of Virginia
yumeng5@virginia.edu