CS 784: Computational Linguistics
Lecture 5: Edit Distance and Distributional (Lexical) Semantics

Freda Shi

School of Computer Science, University of Waterloo
fhs@uwaterloo.ca

January 23, 2025

This Lecture

- Edit Distance
- Distributional (Lexical) Semantics

Edit Distance: Problem Definition

- **Definition:** The edit distance between two strings is the **minimum cost of operations** required to transform one string into the other.
- **Operations:**
  - **Insertion** of a character
  - **Deletion** of a character
  - **Substitution** of a character
- Can be applied to real-life problems such as spell checking, DNA sequence alignment, as well as linguistics-oriented tasks such as morphological analysis.

Example: Calculating Edit Distance

Example

Suppose the costs of insertion, deletion, and substitution are all 1. Calculate the edit distance between the strings `"kitten"` and `"sitting"`.

Answer:

- **Step 1:** Substitute `'k'` with `'s'` → `"sitten"`
- **Step 2:** Substitute `'e'` with `'i'` → `"sittin"`
- **Step 3:** Insert `'g'` at the end → `"sitting"`

Edit Distance: 3

Dynamic Programming Approach

- **Problem:** Given two strings X and Y and the constant cost of each operation, find the minimum cost of operations to convert X to Y.
- **Solution:** Use a dynamic programming table `D` where `D[i, j]` represents the edit distance between the first `i` characters of X and the first `j` characters of Y.
  Key idea of Dynamic Programming: Break down the problem into smaller subproblems (in the same form) and solve them first.
- The Wagner and Fischer (1974) algorithm, which was also independently discovered by many people:

$$\texttt{D[i, j]} = \begin{cases} \max(i, j) & \text{if } \min(i, j) = 0, \textit{(edge cases)} \\ \min \begin{cases} \texttt{D[i-1, j]} + Cost_{del}(X[i]), \\ \texttt{D[i, j-1]} + Cost_{ins}(Y[j]), \\ \texttt{D[i-1, j-1]} + Cost_{sub}(X[i], Y[j]) \end{cases} & \text{o.w.} \end{cases}$$

Edit Distance between "kitten" and "sitting"

		k	i	t	t	e	n
	0	1	2	3	4	5	6
s	1						
i	2						
t	3						
t	4						
i	5						
n	6						
g	7						

$$\texttt{D[i, j]} = max(i, j)$$

Edit Distance between "kitten" and "sitting"

		k	i	t	t	e	n
	0	1	2	3	4	5	6
s	1	1	2	3	4	5	6
i	2	2	1	2	3	4	5
t	3	3	2	1	2	3	4
t	4	4	3	2	1	2	3
i	5	5	4	3	2	2	3
n	6	6	5	4	3	3	2
g	7	7	6	5	4	4	3

$$C_d = 1, C_i = 1$$

$$C_s = 1[x \neq y]$$

$$\texttt{D[i, j]} = \min \begin{cases} \texttt{D[i-1, j]} + C_d, \\ \texttt{D[i, j-1]} + C_i, \\ \texttt{D[i-1, j-1]} + C_s \end{cases}$$

Variants of Edit Distance

The Levenshtein distance:

- All equal cost (usually 1) for insertion, deletion, and substitution.
- Only insertion and deletion allowed (with cost 1), and no substitution.
  Substitution is essentially a deletion followed by an insertion!
  This is equivalent to the having insertion and deletion costs of 1, and substitution cost of 2.

Longest Common Subsequence (LCS): find the longest (possibly discontinuous) subsequence that is common to both strings.

LCS(`kitten` and `sitting`) → `ittn`

- Insertion and deletion cost 1, no substitution.
- $\text{ED}(X, Y) = |X| + |Y| - 2 \times \text{LCS}(X, Y)$

Digital Representations

How does a computer see and read?

Everything needs to be represented in a digital form (or more specifically, binary sequences).

Computers never work with “raw” text.

- Tokenization: breaking text into tokens, which are represented by indexes.
  These indices do not capture meanings of words.
  inter (3849) vs. **interface** (50256)
- Vector representations: represent tokens/words as vectors in a high-dimensional space, and use vector similarity as a proxy for semantic similarity.

Word Vectors

Until the 2010s, in NLP, words meant *atomic symbols*.

Nowadays, in NLP and many CL tasks, words are represented as **vectors**.

Key idea: Similar words are nearby in the vector space.

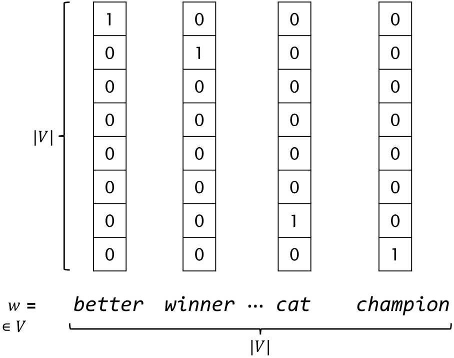
These vectors are also called **word embeddings**.

Word vectors offer a way for account for the variability of natural language: if multiple forms are similar in meaning, their vectors should be close in the vector space.

$$\begin{aligned} \text{really}: &\quad [2.1 \quad -7.9 \quad 8.4 \quad -1.3] \\ \text{reallly}: &\quad [2.0 \quad -6.1 \quad 7.8 \quad -0.8] \\ \text{rly}: &\quad [1.8 \quad -6.8 \quad 7.9 \quad -1.0] \end{aligned}$$

How to represent a word?

One-hot representation: a binary vector with a 1 at the index of the word and 0s elsewhere.



$|V|$ can be very large!
If $|V| = 50K$,
then $|V|^2 = 2.5B$

If stored in 32-bit floats, it would take 10 GB of memory!

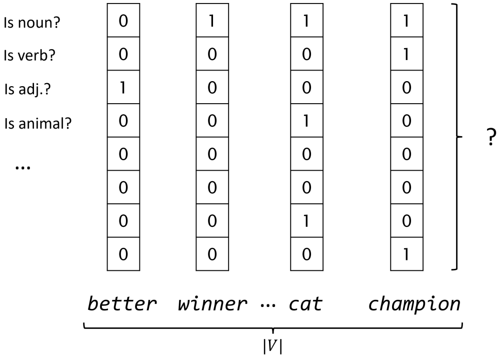
Word Representations

What is an ideal word representation?

It should (probably) capture information about usage and meaning:

- Part-of-speech tags (noun, verb, etc.)
- The intended sense
- Semantic similarities (e.g., winner vs. champion)
- Semantic relationships (antonyms, hypernyms, etc.)
- ...

Feature-Based Representation?



How many features should we design?

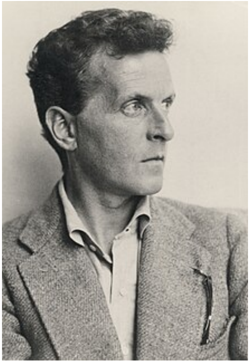
Are there features that we might miss?

Do some features weigh more than others?

Distributional Semantics

The meaning of a word is its use in the language.

—Ludwig Wittgenstein (1943)



The use of a word is defined by its contexts (i.e., the words that appear around it).

Key Idea of Distributional Semantics

Consider this word: **tezgüino**, which appears in the following sentences:

1. A bottle of **tezgüino** is on the table.
2. Everybody likes **tezgüino**.
3. Don't have too much **tezgüino** before you drive.
4. **Tezgüino** is made out of corn.

What do you think **tezgüino** means?

skiing loud motor oil wine

Word	1	2	3	4
tezgüino	1	1	1	1
skiing	0	1	0	0
loud	0	0	0	0
motor oil	1	0	0	1
wine	1	1	1	0

The Distributional Hypothesis in Linguistics

> *The meaning of a word is its use in the language.*
>
> —*Ludwig Wittgenstein (1943)*

> *You shall know a word by the company it keeps.*
>
> —*J.R. Firth (1957)*

> *If A and B have almost identical environments we say that they are synonymous.*
>
> —*Zellig Harris (1954)*

The **distributional hypothesis**: words with similar meaning are used in similar contexts, and vice versa.

How to represent words based on their contexts?

Co-Occurrence

Let $\mathbf{C} \in \mathbb{Z}_+^{|V| \times |V|}$ denote the co-occurrence matrix of a corpus.

V is the vocabulary.

C_{ij} is the number of times word j appears in the context of word i.

We will need to define a context window size w: if two word tokens are within w tokens of each other, they are considered to be in each other's context.

Use $\mathbf{C}_i$ as the word vector for word $v_i (v_i \in |V|)$, and use dot-product or cosine similarity to measure word similarity.

$$\text{dot-product}(\boldsymbol{\alpha}, \boldsymbol{\beta}) = \boldsymbol{\alpha} \cdot \boldsymbol{\beta} = \langle \boldsymbol{\alpha}, \boldsymbol{\beta} \rangle = \boldsymbol{\alpha}^T \boldsymbol{\beta} = \sum_i \alpha_i \beta_i$$

$$\text{cosine}(\boldsymbol{\alpha}, \boldsymbol{\beta}) = \frac{\boldsymbol{\alpha}^T \boldsymbol{\beta}}{\|\boldsymbol{\alpha}\| \|\boldsymbol{\beta}\|} = \frac{\sum_i \alpha_i \beta_i}{\sqrt{\sum_i \alpha_i^2} \sqrt{\sum_i \beta_i^2}}$$

Any issues?

Issues with the Co-Occurrence Matrix

- It's very large: $|V|^2$ entries.
  Solution: Use a small set of words as the context—make $\mathbf{C} \in \mathbb{Z}_+^{|V| \times |C|}$, where $|C| \ll |V|$.
- Some common words (e.g., **the**, **a**, **of**) will have high counts with many words, dominating the similarity calculation, but they may not be very informative.
  Solution: Exclude them from the context vocabulary (**stop words**).

Or use better quantities to substitute the raw counts (options below)!

TF-IDF

Before getting into word vectors, let's talk about a common technique used to represent document in conventional information retrieval.

- **Term Frequency** $tf(d, w)$:number of times a word w appears in a document d.
- **Inverse Document Frequency (IDF)** $idf(w)$: inverse of the number of documents that contain the term

$$idf(w) = \log \frac{N}{N_w}$$

N :total number of documents

N_w :number of documents containing word w

- **TF-IDF**: the product of TF and IDF.

$$\text{tf-idf}(d, w) = \text{tf}(d, w) \times \text{idf}(w)$$

Alternative 1 of Co-Occurence: (PMI)

Recall: The mutual information between variables X and Y is

$$I(X;Y) = \sum_{x \in X} \sum_{y \in Y} P(x,y) \log \frac{P(x,y)}{P(x)P(y)}$$

Pointwise mutual information (PMI; Fano, 1961) measures the association between two words w_i and w_j by

$$\mathrm{PMI}(w_i, w_j) = \log \frac{P(w_i, w_j)}{P(w_i)P(w_j)} = \log \frac{P(w_i|w_j)}{P(w_i)} = \log \frac{P(w_j|w_i)}{P(w_j)}$$

$P(w_i, w_j)$: probability of observing w_i and w_j together.

$P(w_i)$ and $P(w_j)$: probabilities of observing w_i and w_j independently.

PMI is a measure of how much more likely the two words co-occur than if they were independent.

PMI: Implementation

Using frequentist estimation of probability:

$$P(w_i, w_j) \approx \frac{C_{ij}}{\ell C} \qquad P(w_i) = \frac{C_i}{C} \qquad P(w_j) = \frac{C_j}{C}$$

ℓ: context window length.

C_i, C_j: word token counts of w_i and w_j.

C_{ij}: co-occurrence count of w_i (left) and w_j (right).

C: total word token count.

$$PMI(w_i, w_j) = \log \frac{P(w_i, w_j)}{P(w_i)P(w_j)} = \log \frac{C_{ij} \cdot C^2}{C_i \cdot C_j \cdot \ell(C-1)}$$

$$\approx \log \frac{C_{ij} \cdot C}{C_i \cdot \ell}$$

PMI with Laplace Smoothing

$$PMI(w_i, w_j) \approx \log \frac{C_{ij} \cdot C}{C_i \cdot \ell}$$

If we enumerate all possible word pairs, we will have many $C_{ij} = 0$ in the co-occurrence matrix, which makes the above formula ill-defined.

Solution: Laplace smoothing—add a small constant α (usually $\alpha \in [0.1, 3]$) to all counts.

$$P(w_i, w_j) \approx \frac{C_{ij} + \alpha}{\ell C + \alpha |V|^2}$$

Highest PMI Pairs on Wikipedia Oct 2015 Dump

w_i	w_j	C_i	C_j	C_{ij}	$PMI_e(w_i, w_j)$
puerto	rico	1938	1311	1159	10.03
hong	kong	2438	2694	2205	9.73
los	angeles	3501	2808	2791	9.56
carbon	dioxide	4265	1353	1032	9.10
prize	laureate	5131	1676	1210	8.86
san	francisco	5237	2477	1779	8.83
nobel	prize	4098	5131	2498	8.69
ice	hockey	5607	3002	1933	8.66
star	trek	8264	1594	1489	8.64
car	driver	5578	2749	1384	8.41

[Source: Wikipedia]

Lowest PMI Pairs on Wikipedia Oct 2015 Dump

w_i	w_j	C_i	C_j	C_{ij}	$PMI_e(w_i, w_j)$
it	the	283891	3293296	3347	-1.72
are	of	234458	1761436	1019	-2.09
this	the	199882	3293296	1211	-2.39
is	of	565679	1761436	1562	-2.55
and	of	1375396	1761436	2949	-2.80
a	and	984442	1375396	1457	-2.92
in	and	1187652	1375396	1537	-3.06
to	and	1025659	1375396	1286	-3.09
to	in	1025659	1187652	1066	-3.13
of	and	1761436	1375396	1190	-3.71

[Source: Wikipedia]

Positive PMI

The PMI matrix still suffers from the large ($|V|^2$) size.

Negative PMIs: how words are **not** related, which may not be very informative to define context, especially when the absolute values are close to 0.

Church and Hanks (1989) and others:

$$\text{PPMI}(w_i, w_j) = \max\left(\text{PMI}(w_i, w_j), 0\right)$$

Enables a range of algorithms that requires sparsity!

Before `word2vec` (Mikolov et al., 2013), SVD of the PPMI matrix was a popular method to obtain word vectors.

See an example of PPMI word vectors and its application here: [Turney et al. EMNLP 2011]

Alternative 2 of Co-Occurence: Neural Word Vectors

Recall the **distributional hypothesis**: words with similar meanings are used in similar contexts.

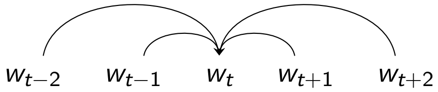
Translate to neural network approach: word embeddings for a word should be learned (from random initialization) such that they can well-predict (or can be well-predicted by) the surrounding words in the context.

Trainable parameters $\boldsymbol{\Theta} = \mathbf{W} \in \mathbb{R}^{|V| \times d}$: word vectors.

d: dimensionality of the word vectors.

Word2Vec (Mikolov et al., 2013)

Continuous bags of words (CBOW): predict the target word from the context words, or predict one from many.



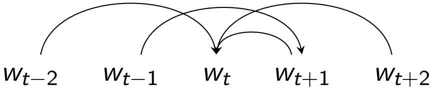
$$\begin{aligned}
\mathbf{W}^* &= \max_{\mathbf{W}} \mathbb{E}_{w_t \sim Pop} \left[P_{\mathbf{W}}(w_t | w_{t-\ell}, \ldots, w_{t-1}, w_{t+1}, \ldots, w_{t+\ell}) \right] \\
&= \max_{\mathbf{W}} \mathbb{E}_{w_t} \left[\frac{\exp\left(\mathbf{w}_t \cdot \text{avg}(\mathbf{w}_{t-\ell}, \ldots, \mathbf{w}_{t-1}, \mathbf{w}_{t+1}, \ldots, \mathbf{w}_{t+\ell})\right)}{\sum_{v \in V} \exp\left(\mathbf{w}_v \cdot \text{avg}(\mathbf{w}_{t-\ell}, \ldots, \mathbf{w}_{t-1}, \mathbf{w}_{t+1}, \ldots, \mathbf{w}_{t+\ell})\right)} \right]
\end{aligned}$$

Pop: the population distribution of words in the corpus.

This formulation of $\frac{\exp(\cdot)}{\sum \exp(\cdot)}$ is called the **softmax** function.

Word2Vec (Mikolov et al., 2013)

Skip-gram (SG): predict the context words from the target word, or predict one from one, by learning to distinguish between true pair $\langle w, c \rangle$ and negative samples $\langle w, v \rangle$.



$$\mathbf{W}^* = \max_{\mathbf{W}} \mathbb{E}_{w_t, w_c, w_v \in Pop} \left[\log P(\langle w_t, w_c \rangle^+) + \log P(\langle w_t, w_v \rangle^-) \right]$$

$$P(\langle w_t, w_c \rangle^+) = \sigma(\mathbf{w}_t \cdot \mathbf{w}_c)$$

$$P(\langle w_t, w_v \rangle^-) = \sigma(-\mathbf{w}_t \cdot \mathbf{w}_v)$$

$$\sigma(x) = \frac{1}{1 + \exp(-x)}$$ is the sigmoid function.

Empirically, each positive pair is coupled with K negative samples.

Lightweight Libraries/Resources for Word2Vec

- **GenSim** (For training word vectors from your corpus)
  `https://radimrehurek.com/gensim/models/word2vec.html`
- **Glove** (Pennington et al., 2014)
  `https://nlp.stanford.edu/projects/glove/`
- **FastText** (Bojanowski et al., 2017; for a state-of-the-art word embeddings with awareness of subword information)
  `https://fasttext.cc/`
- The 0-th layer of pretrained language models such as BERT (Devlin et al., 2019) and GPT-2 (Radford et al., 2019).

Questions

Think about the following questions:

- What are the possible issues of neural word2vec models?
  For example, are there linguistic features that cannot be captured by the model?
- Do the issues exist with subword tokenization?

Next

Dataset and Data Curation

P.S. A random picture (distantly) relevant to this lecture

Ludwig the Cat