Introduction to Natural Language Processing

Machine Learning Methods – Lecture 24

June 2021

fixelalgorithms.gitlab.io

Or Yair

Natural Language Processing – Introduction

- Consider the following IMDb reviews:

  "This movie is terrible but it has some good effects." positive

  "I wouldn't rent this one even on dollar rental night." negative

  "Ming The Merciless does a little Bardwork and a movie most foul!"

  "You'd better choose Paul Verhoeven's even if you have watched it."

  "Adrian Pasdar is excellent in this film. He makes a fascinating woman."

  "Long, boring, blasphemous. Never have I been so glad to see ending credits roll."

  "I don't know why I like this movie so well, but I never get tired of watching it."

  "no comment - stupid movie, acting average or worse... screenplay - no sense at all... SKIP IT!"

  "A rating of "1" does not begin to express how dull, depressing and relentlessly bad this movie is."

  "This is the definitive movie version of Hamlet.
  Branagh cuts nothing, but there are no wasted moments."

- A classic NLP task is to predict the sentiment of a given review.

Bag of Words – Introduction

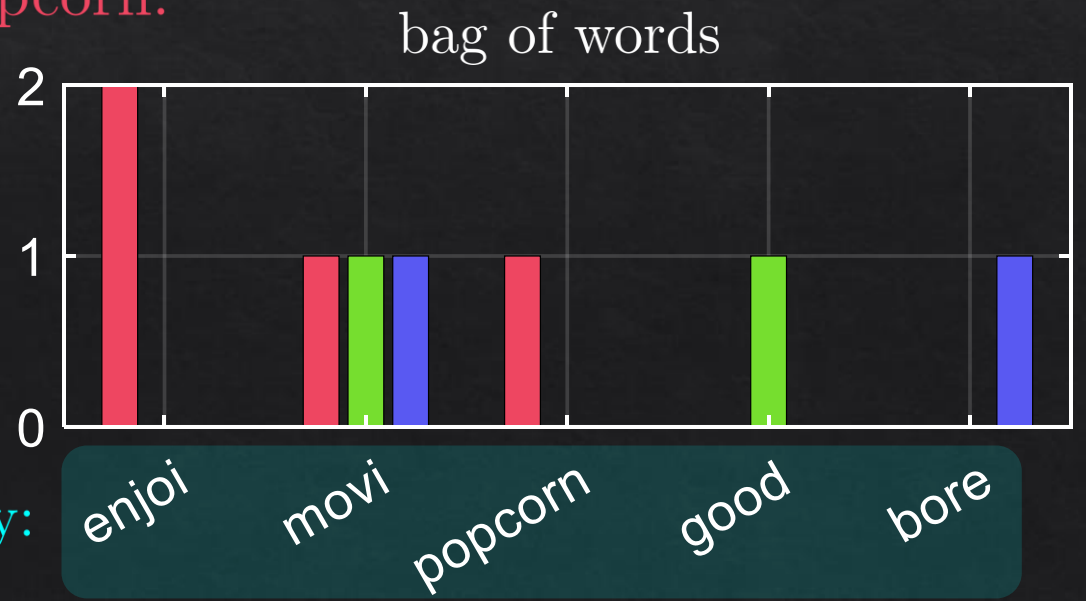
- Usually, the first task is to make the data numeric.
- In Bag of Words we:
  1. Apply some pre-processing.
  2. Build a vocabulary (a.k.a. dictionary).
  3. Replace each text with a corresponding histogram.

Example

"I enjoyed the movie, but I did not enjoy the popcorn."

"This is a good movie."

"The movie was boring."



Bag of Words – Pre-processing

- The most common words among $N = 2,000$ IMDb reviews:

Capital letters

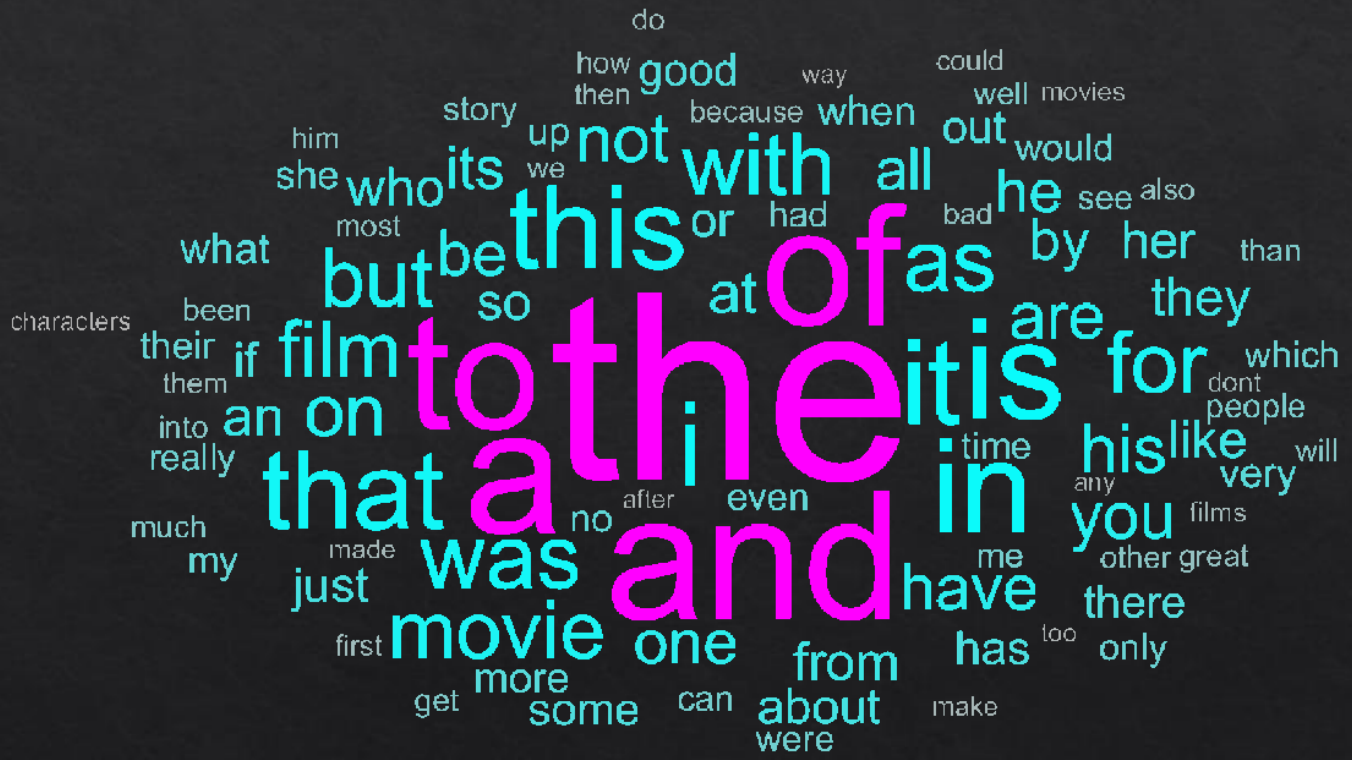
Punctuation: “””, “.”, “(“, “\”, etc.

- First step is to remove spaces, punctuation, and change all characters to lower case.

Bag of Words – Pre-processing

- The most common words among $N = 2,000$ IMDb reviews:

- First step is to remove spaces, punctuation, and change all characters to lower case.



- Second, we remove stop words: “the”, “of”, “to”, “and”, “a”, etc.

Bag of Words – Pre-processing

- The most common words among $N = 2,000$ IMDb reviews:
- First step is to remove spaces, punctuation, and change all characters to lower case.
- Second, we remove stop words: “the”, “of”, “to”, “and”, “a”, etc.

- Last step is stemming the words: movie(s) $\rightarrow$ movi, watch(ing) $\rightarrow$ watch.

Bag of Words – Dictionary

- The most common words among $N = 2,000$ IMDb reviews:
- First step is to remove spaces, punctuation, and change all characters to lower case.
- Second, we remove stop words: "the", "of", "to", "and", "a", etc.
- Last step is stemming the words: movie(s) $\rightarrow$ movi, watch(ing) $\rightarrow$ watch.

after these steps,
we remain with $19,910$ unique words

let $\mathcal{V} = \{w_j\}_{j=1}^{19,910}$ be the set (vocabulary) of all unique words.

Bag of Words – Data

- The numeric data set is $\boldsymbol{X} \in \mathbb{N}_0^{2{,}000 \times 19{,}910}$.
- The ith row of $\boldsymbol{X}$ is the histogram of the ith document.
- Since most of the entries in $\boldsymbol{X}$ are zero, it is efficient to use sparse representation.
- For example:

$$\boldsymbol{X} = \begin{bmatrix} 2 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 \end{bmatrix}$$

$$X_{ij} = \sum_{w \in d_i} \mathbb{I}\{w_j = w\}$$

d_i is the ith review (document)

$w_j \in \mathcal{V}$ is the jth word (term)

$N = 2{,}000$ IMDb reviews:

after these steps,

we remain with $19{,}910$ unique words

let $\mathcal{V} = \{w_j\}_{j=1}^{19{,}910}$ be the set (vocabulary) of all unique words.

Term Frequency Inverse Document Frequency (TF-IDF)

- TF-IDF is a weighted version of bag of words. $X_{ij} = \sum_{w \in d_i} \mathbb{I}\{w_j = w\}$
- Term Frequency: counts the occurrences of the word $w \in \mathcal{V}$ in a document d:

$$\text{tf}(w, d) = \sum_{w_j \in d} \mathbb{I}\{w_j = w\}$$

tf is the bag of words: $X_{ij} = \text{tf}(w_j, d_i)$

- Inverse Document Frequency: measures whether the word $w \in \mathcal{V}$ is common or rare across all documents $\mathcal{D} = \{d_1, d_2, \ldots, d_N\}$:

$\mathcal{D}$ is known as corpus

$$\text{idf}(w, \mathcal{D}) = \log\left(\frac{N}{\sum_{d_i \in \mathcal{D}} \mathbb{I}\{w \in d_i\}}\right)$$

$N = |\mathcal{D}|$

There are other possible definitions for both TF and IDF.

- TD-IDF is given by:

$$\text{tf-idf}(w, d, \mathcal{D}) = \text{tf}(w, d) \cdot \text{idf}(w, \mathcal{D})$$

$$\boldsymbol{X}_{\text{tf-idf}}[i, j] = \text{tf}(w_j, d_i) \cdot \text{idf}(w_j, \mathcal{D})$$

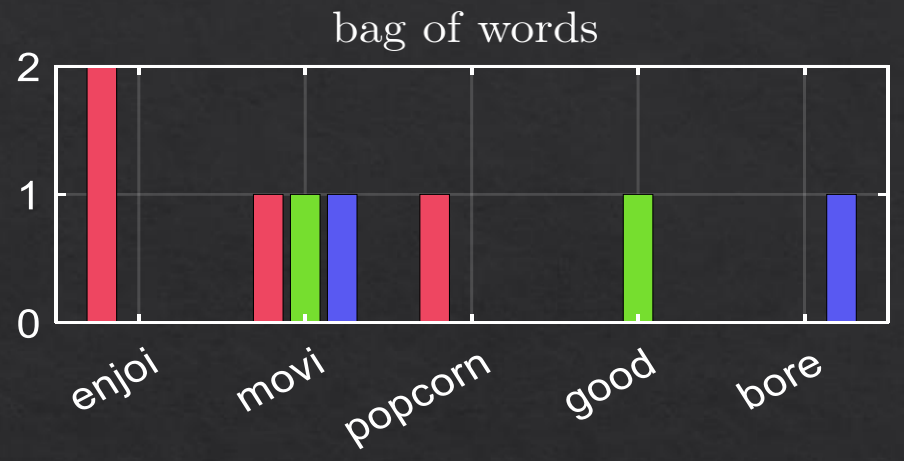
Bag of Words – IMDb Example

$$\text{tf}(w, d) = \sum_{w_j \in d} \mathbb{I}\{w_j = w\}$$

$$\text{idf}(w, \mathcal{D}) = \log\left(\frac{N}{\sum_{d_i \in \mathcal{D}} \mathbb{I}\{w \in d_i\}}\right)$$

$$\text{tf-idf}(w, d, \mathcal{D}) = \text{tf}(w, d) \cdot \text{idf}(w, \mathcal{D})$$

$$\boldsymbol{X}_{\text{tf-idf}}[i, j] = \text{tf}(w_j, d_i) \cdot \text{idf}(w_j, \mathcal{D})$$



let $\mathcal{V} = \{w_j\}_{j=1}^{19,910}$ be the set (vocabulary) of all unique words.

$$X_{ij} = \sum_{w \in d_i} \mathbb{I}\{w_j = w\}$$

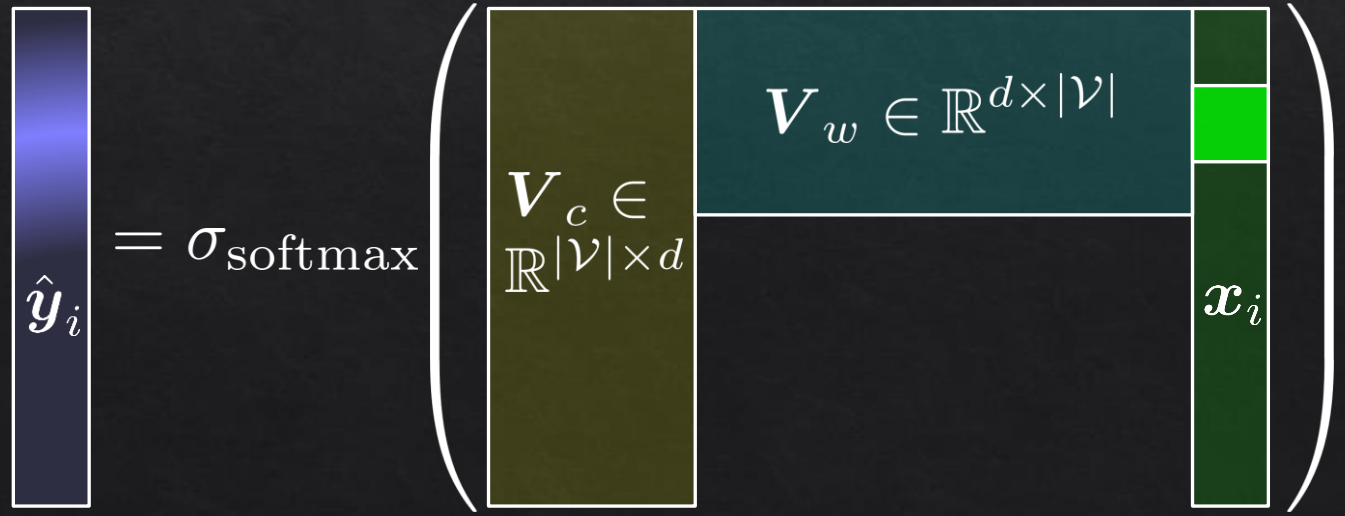
Bag of Words – IMDb

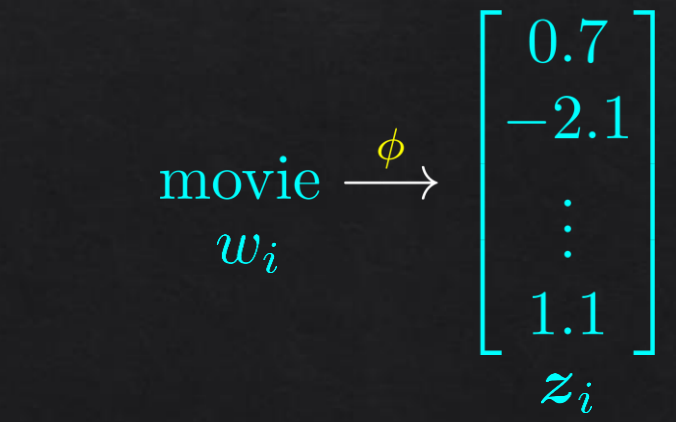
Word2Vec – Introduction

- Word2Vec is a method to embed words into a vector (feature) space $\mathbb{R}^d$.
- Let $\boldsymbol{z}_i = \phi(w_i)$ be the mapping of the word $w_i \in \mathcal{V}$ to the vector $\boldsymbol{z}_i \in \mathbb{R}^d$, where $\mathcal{V}$ is the vocabulary (set of all words).
- The feature space has a useful structure, for example:

$$\phi(\text{actor}) - \phi(\text{man}) + \phi(\text{woman}) \approx \phi(\text{actress})$$

- The word embedding is obtained by optimizing a linear transformation followed by a softmax activation:





Word2Vec – Context Window

- Consider the following text:

  "... an exciting summer blockbuster that was visually fantastic but also curiously ..."
  context context word context context

- From the text we build pairs (word, context).

  For example, given the word $w_i =$ blockbuster and a context radius of 2 we have:

  $(x_1, y_1) = (\text{blockbuster}, \text{exciting})$
  $(x_2, y_2) = (\text{blockbuster}, \text{summer})$
  $(x_3, y_3) = (\text{blockbuster}, \text{that})$
  $(x_4, y_4) = (\text{blockbuster}, \text{was})$
  $\cdots$
  $(x_{17}, y_{17}) = (\text{fantastic}, \text{was})$
  $(x_{18}, y_{18}) = (\text{fantastic}, \text{visually})$
  $(x_{19}, y_{19}) = (\text{fantastic}, \text{but})$
  $(x_{20}, y_{20}) = (\text{fantastic}, \text{also})$
  $\cdots$

- We iterate the entire text and construct all (word, context) pairs $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{N}$.
- Word2Vec tries to predict the correct context from the given word.

Word2Vec – One-hot Encoding

- We encode each word using a one-hot encoding.
  For example, if $x_i = \text{blockbuster}$ is the third word in the vocabulary $\mathcal{V}$, then:

$$\boldsymbol{x}_i = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \in \mathbb{R}^{|\mathcal{V}|} \implies \mathcal{D} = \{ \underbrace{(\boldsymbol{x}_1, \boldsymbol{y}_1)}, \underbrace{(\boldsymbol{x}_2, \boldsymbol{y}_2)}, \cdots, \underbrace{(\boldsymbol{x}_N, \boldsymbol{y}_N)} \}$$

$|\mathcal{V}|$ is the size of the vocabulary.

$$= \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

$(x_1, y_1) = (\text{blockbuster}, \text{exciting})$

$(x_2, y_2) = (\text{blockbuster}, \text{summer})$

$(x_3, y_3) = (\text{blockbuster}, \text{that})$

$(x_4, y_4) = (\text{blockbuster}, \text{was})$

$$\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{N}$$

Architecture and Loss Function

- The prediction function of $\boldsymbol{y}_i$ given $\boldsymbol{x}_i$:

$$\hat{\boldsymbol{y}}_i = \sigma_{\text{softmax}}\left(\boldsymbol{V}_c \boldsymbol{V}_w \boldsymbol{x}\right)$$

$$\hat{\boldsymbol{y}}_i = \sigma_{\text{softmax}}\left(\boldsymbol{V}_c \in \mathbb{R}^{|\mathcal{V}| \times d} \quad \boldsymbol{V}_w \in \mathbb{R}^{d \times |\mathcal{V}|} \quad \boldsymbol{x}_i \right)$$

$$\mathcal{D} = \{ \underbrace{(,)}_{(\boldsymbol{x}_1, \boldsymbol{y}_1)}, \underbrace{(,)}_{(\boldsymbol{x}_2, \boldsymbol{y}_2)}, \cdots, \underbrace{(,)}_{(\boldsymbol{x}_N, \boldsymbol{y}_N)} \}$$

$$\boldsymbol{p} = \sigma_{\text{softmax}}\left(\boldsymbol{v}\right) = \frac{\exp\left(\boldsymbol{v}\right)}{\mathbf{1}^T \exp\left(\boldsymbol{v}\right)}$$

$\boldsymbol{p}$ is a probabilistic vector

- As in logistic regression, the MLE of $\boldsymbol{V}_w$ and $\boldsymbol{V}_c$ are the minimizeres of the cross-entropy loss:

$$L = -\frac{1}{N} \sum_{i=1}^{N} \boldsymbol{y}_i^T \log\left(\hat{\boldsymbol{y}}_i\right) \quad \Longrightarrow \quad \arg\min_{\boldsymbol{V}_c, \boldsymbol{V}_w} -\frac{1}{N} \sum_{i=1}^{N} \boldsymbol{y}_i^T \log\left(\hat{\boldsymbol{y}}_i\right)$$

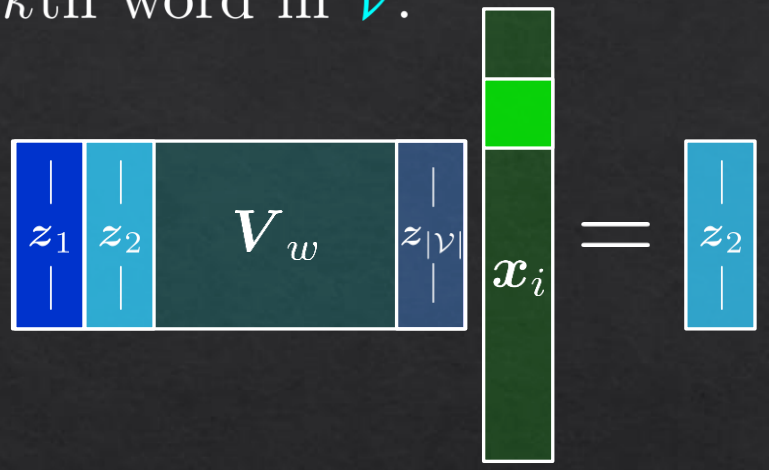
the objective is not convex

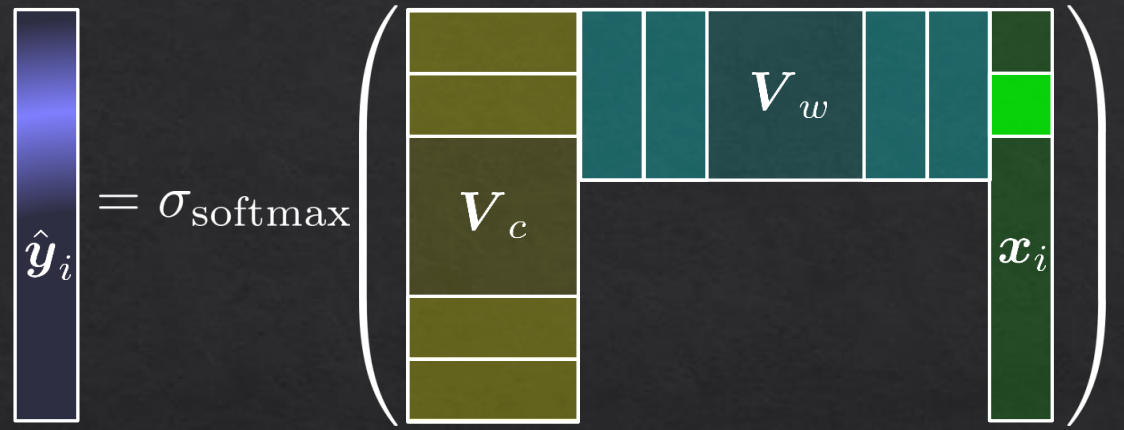
- The final words embedding are given by the columns of $\boldsymbol{V}_w^\star$.

Word2Vec – Interpretation

$$\arg \min_{\boldsymbol{V}_c, \boldsymbol{V}_w} -\frac{1}{N} \sum_{i=1}^{N} \boldsymbol{y}_i^T \log(\hat{\boldsymbol{y}}_i)$$

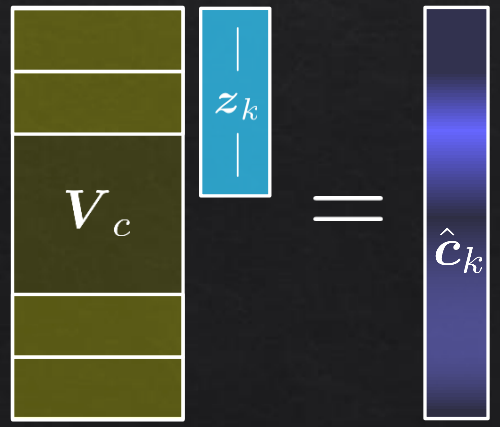
- Consider $\boldsymbol{x}_i$ that corresponds to the kth word in $\mathcal{V}$:





$\implies \boldsymbol{z}_k \in \mathbb{R}^d$ is a vector representation of $w_k \in \mathcal{V}$.

- In addition, the vector $\boldsymbol{c}_k = \boldsymbol{V}_c \boldsymbol{z}_k$ provide a context score between w_k and all other words (as context words).



Word2Vec – Negative Sampling

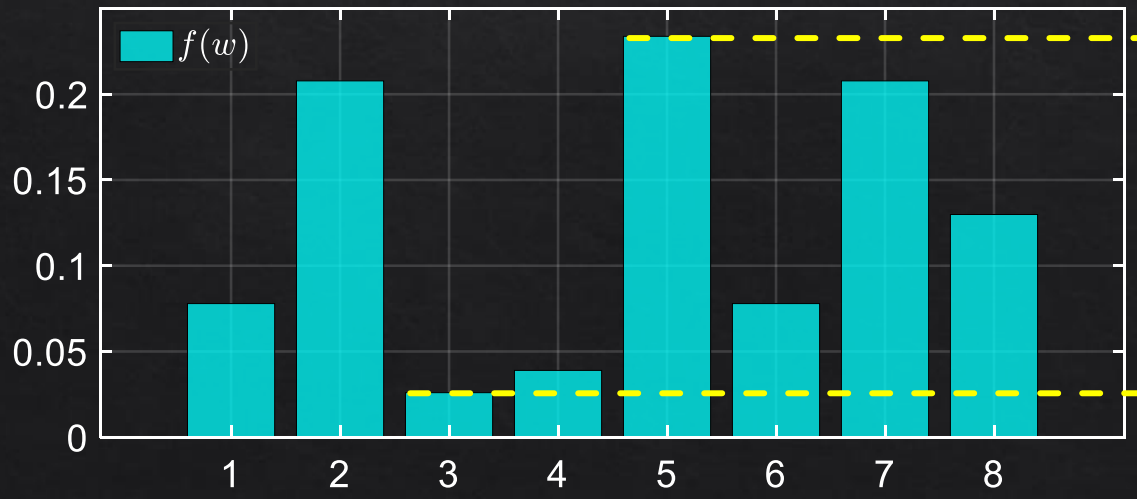
- When the size of the vocabulary is large $|\mathcal{V}|$, the optimization is not trivial.
- Let $P_{\mathcal{V}}$ be a probability of the word w:

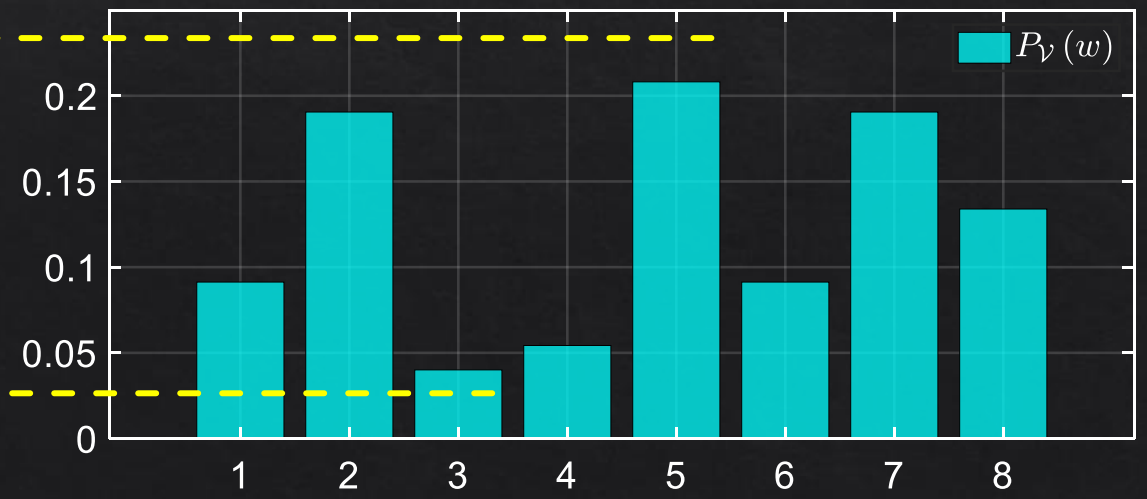
$$P_{\mathcal{V}}(w) = \frac{f^{\frac{3}{4}}(w)}{\sum_{w_i \in \mathcal{V}} f^{\frac{3}{4}}(w_i)}$$

where $f(w) \in [0, 1]$ is the frequency of w appears in the corpus.

- The power $(\cdot)^{\frac{3}{4}}$ makes less frequent words to be sampled more often:

$$\hat{\boldsymbol{y}}_i = \sigma_{\text{softmax}}\left(\boldsymbol{V}_c \boldsymbol{V}_w \boldsymbol{x}_i \right)$$





Word2Vec – Negative Sampling

$$P_{\mathcal{V}}(w) = \frac{f^{\frac{3}{4}}(w)}{\sum_{w_i \in \mathcal{V}} f^{\frac{3}{4}}(w_i)}$$

- Given a true pair $(\boldsymbol{x}_i, \boldsymbol{y}_i)$, we create a fake (negative) pair $(\boldsymbol{x}_i, \tilde{\boldsymbol{y}}_i)$, where $\tilde{\boldsymbol{y}}_i$ is one-hot vector of a random word $\tilde{w} \sim P_{\mathcal{V}}$.
- Instead of consider the entire vocabulary $\mathcal{V}$, we reduce the problem to the words correspond to $\boldsymbol{x}_i$, $\boldsymbol{y}_i$, and $\tilde{\boldsymbol{y}}_i$:

$$\tilde{\boldsymbol{y}}_i \; \boldsymbol{y}_i \;, \quad \hat{\boldsymbol{y}}_i = \sigma_{\text{softmax}} \left(\begin{bmatrix} \boldsymbol{v}_c \\ \boldsymbol{V}_c \\ \tilde{\boldsymbol{v}}_c \end{bmatrix} \boldsymbol{V}_w \boldsymbol{z}_k \; \boldsymbol{x}_i \right) \Longrightarrow \tilde{\boldsymbol{p}} = \sigma_{\text{softmax}} \left(\begin{bmatrix} \boldsymbol{v}_c \\ \tilde{\boldsymbol{v}}_c \end{bmatrix} \boldsymbol{z}_k \right)$$

- We optimize only w.r.t. $\boldsymbol{z}_k$,$\boldsymbol{v}_c$, and $\tilde{\boldsymbol{v}}_c$:

$$\arg \min_{\boldsymbol{z}_k, \boldsymbol{v}_c, \tilde{\boldsymbol{v}}_c} - \left\langle \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \log(\tilde{\boldsymbol{p}}) \right\rangle \longleftarrow \text{cross-entropy loss}$$

$$\arg \min_{\boldsymbol{V}_c, \boldsymbol{V}_w} -\frac{1}{N} \sum_{i=1}^{N} \boldsymbol{y}_i^T \log(\hat{\boldsymbol{y}}_i)$$

- At each step (of the optimization), we draw a new negative sample.
- One can take more than one negative pair.

Word2Vec – Example

$$\hat{\boldsymbol{y}}_i = \sigma_{\text{softmax}}\left(\boldsymbol{V}_c \boldsymbol{V}_w \boldsymbol{x}_i\right)$$

$$\boldsymbol{z}_1 \; \boldsymbol{z}_2 \; \boldsymbol{V}_w \; \boldsymbol{z}_{|\mathcal{V}|} \; \boldsymbol{x}_i = \boldsymbol{z}_2$$

$$\boldsymbol{V}_c \; \boldsymbol{z}_k = \hat{\boldsymbol{c}}_k$$

[Word2Vec – IMDb]

Questions

Word2Vec:

$$\hat{\boldsymbol{y}}_i = \sigma_{\text{softmax}}\left(\boldsymbol{V}_c \, \boldsymbol{V}_w \, \boldsymbol{x}_i \right)$$

TF-IDF:

$$\text{tf}\left(w, d\right) = \sum_{w_j \in d} \mathbb{I}\left\{w_j = w\right\}$$

$$\text{idf}\left(w, \mathcal{D}\right) = \log\left(\frac{N}{\sum_{d_i \in \mathcal{D}} \mathbb{I}\left\{w \in d_i\right\}}\right)$$

$$\text{tf-idf}\left(w, d, \mathcal{D}\right) = \text{tf}\left(w, d\right) \cdot \text{idf}\left(w, \mathcal{D}\right)$$

fixelalgorithms.gitlab.io