Deep Learning for NLP
Lecture 4: Text Representations I

Dr. Mohsen Mesgar

Ubiquitous Knowledge Processing Lab (UKP Lab)

This lecture

- common features used for converting textual data into numerical vectors
- basics of word embeddings
  - how to get them?
  - where to use them?
  - how to use them?
- limitations

Recall

- the input to a supervised learning algorithm is a training set $(x_{1:n}, y_{1:n})$, where

Recall

- the input to a supervised learning algorithm is a training set $(x_{1:n}, y_{1:n})$, where
  - $x_{1:n} = x_1, x_2, ..., x_n$ shows input examples

Recall

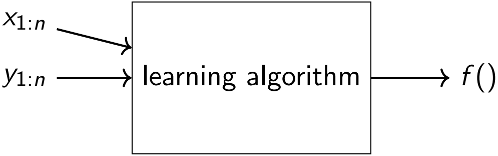
- the input to a supervised learning algorithm is a training set $(x_{1:n}, y_{1:n})$, where
  - $x_{1:n} = x_1, x_2, ..., x_n$ shows input examples
  - $y_{a:n} = y_1, y_2, ..., y_n$ shows corresponding labels

Recall

- the input to a supervised learning algorithm is a training set $(x_{1:n}, y_{1:n})$, where
  - $x_{1:n} = x_1, x_2, ..., x_n$ shows input examples
  - $y_{a:n} = y_1, y_2, ..., y_n$ shows corresponding labels
- the goal of a learning algorithm is to return a function $f()$ that accurately maps input examples to their desired labels

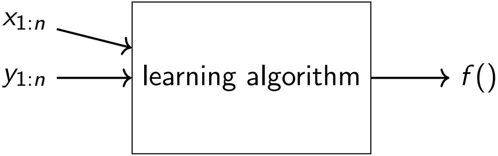
Recall

- the input to a supervised learning algorithm is a training set $(x_{1:n}, y_{1:n})$, where
  - $x_{1:n} = x_1, x_2, ..., x_n$ shows input examples
  - $y_{a:n} = y_1, y_2, ..., y_n$ shows corresponding labels
- the goal of a learning algorithm is to return a function $f()$ that accurately maps input examples to their desired labels



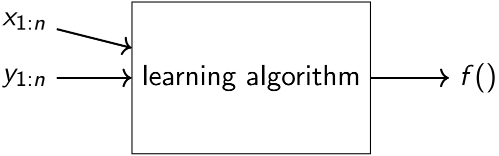
Recall

- the input to a supervised learning algorithm is a training set $(x_{1:n}, y_{1:n})$, where
  - $x_{1:n} = x_1, x_2, ..., x_n$ shows input examples
  - $y_{a:n} = y_1, y_2, ..., y_n$ shows corresponding labels
- the goal of a learning algorithm is to return a function $f()$ that accurately maps input examples to their desired labels



Recall

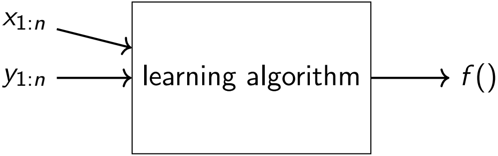
- the input to a supervised learning algorithm is a training set $(x_{1:n}, y_{1:n})$, where
  - $x_{1:n} = x_1, x_2, ..., x_n$ shows input examples
  - $y_{a:n} = y_1, y_2, ..., y_n$ shows corresponding labels
- the goal of a learning algorithm is to return a function $f()$ that accurately maps input examples to their desired labels



- the input to neural models $f()$ should be a numerical vector but NLP tasks are defined on texts

Recall

- the input to a supervised learning algorithm is a training set $(x_{1:n}, y_{1:n})$, where
  - $x_{1:n} = x_1, x_2, \ldots, x_n$ shows input examples
  - $y_{a:n} = y_1, y_2, \ldots, y_n$ shows corresponding labels
- the goal of a learning algorithm is to return a function $f()$ that accurately maps input examples to their desired labels



- the input to neural models $f()$ should be a numerical vector but NLP tasks are defined on texts
- how can we represent texts via numerical vectors?

Text Representation (a.k.a., Feature Extraction)

- vector representation of a text should ideally reflect various linguistic properties of the text

Text Representation (a.k.a., Feature Extraction)

- vector representation of a text should ideally reflect various linguistic properties of the text

- text $\longrightarrow$ $\boxed{\text{feature function}}$ $\longrightarrow x \longrightarrow f()$

Text Representation (a.k.a., Feature Extraction)

- vector representation of a text should ideally reflect various linguistic properties of the text
- text $\longrightarrow$ $\boxed{\text{feature function}}$ $\longrightarrow x \longrightarrow f()$
- in this lecture we focus on feature functions rather than learning algorithms

Some Terminologies

- letters: smallest units in a language $\rightarrow$ a,b,c,...

Some Terminologies

- letters: smallest units in a language $\rightarrow$ a,b,c,...
- tokens and words: tokens are outputs of a tokenizer and words are meaning-bearing units
  - note: in this course we use the terms “word” and “token” interchangeably

Some Terminologies

- letters: smallest units in a language → a,b,c,...
- tokens and words: tokens are outputs of a tokenizer and words are meaning-bearing units
  - note: in this course we use the terms "word" and "token" interchangeably
- lemma: the dictionary entry of the word → "book" is the lemma of "booking", "booked", and "books"
  - how to obtain lemmas? use morphological analyzers
  - are available for many languages
  - lemmatization may not work for any sequence of letters e.g., for mis-spelling

Some Terminologies

- letters: smallest units in a language → a,b,c,...
- tokens and words: tokens are outputs of a tokenizer and words are meaning-bearing units
  - note: in this course we use the terms "word" and "token" interchangeably
- lemma: the dictionary entry of the word → "book" is the lemma of "booking", "booked", and "books"
  - how to obtain lemmas? use morphological analyzers
  - are available for many languages
  - lemmatization may not work for any sequence of letters e.g., for mis-spelling
- stems: a shorter venison of a word defined based on some language-specific heuristic → "pictur" is the stem of "pictures", "pictured", and "picture"
  - the output of a stemmer need not be a valid word

Some Terminologies

- lexical resources: provide some information about words and their relations
  - dictionaries

Some Terminologies

- lexical resources: provide some information about words and their relations
  - dictionaries
  - WordNet for English

Some Terminologies

- lexical resources: provide some information about words and their relations
  - dictionaries
  - WordNet for English
    - capture semantic knowledge about words

Some Terminologies

- lexical resources: provide some information about words and their relations
  - dictionaries
  - WordNet for English
    - capture semantic knowledge about words
    - each word is associated with one or several synsets

Some Terminologies

- lexical resources: provide some information about words and their relations
  - dictionaries
  - WordNet for English
    - capture semantic knowledge about words
    - each word is associated with one or several synsets
    - synsets are linked to each other according to semantic relations between their words

Some Terminologies

- lexical resources: provide some information about words and their relations
  - dictionaries
  - WordNet for English
    - capture semantic knowledge about words
    - each word is associated with one or several synsets
    - synsets are linked to each other according to semantic relations between their words
    - semantic relations are: hypernym (more specific), hyponym (less specific), antonyms, holonyms (part-whole), and meronyms (whole-part)

Some Terminologies

- lexical resources: provide some information about words and their relations
  - dictionaries
  - WordNet for English
    - capture semantic knowledge about words
    - each word is associated with one or several synsets
    - synsets are linked to each other according to semantic relations between their words
    - semantic relations are: hypernym (more specific), hyponym (less specific), antonyms, holonyms (part-whole), and meronyms (whole-part)
    - contains information about nouns, verbs, adjectives, and adverbs

Some Terminologies

- lexical resources: provide some information about words and their relations
  - dictionaries
  - WordNet for English
    - capture semantic knowledge about words
    - each word is associated with one or several synsets
    - synsets are linked to each other according to semantic relations between their words
    - semantic relations are: hypernym (more specific), hyponym (less specific), antonyms, holonyms (part-whole), and meronyms (whole-part)
    - contains information about nouns, verbs, adjectives, and adverbs
    - manually created

Some Terminologies

- lexical resources: provide some information about words and their relations
  - dictionaries
  - WordNet for English
    - capture semantic knowledge about words
    - each word is associated with one or several synsets
    - synsets are linked to each other according to semantic relations between their words
    - semantic relations are: hypernym (more specific), hyponym (less specific), antonyms, holonyms (part-whole), and meronyms (whole-part)
    - contains information about nouns, verbs, adjectives, and adverbs
    - manually created
  - FrameNet and VerbNet for English

Some Terminologies

- lexical resources: provide some information about words and their relations
  - dictionaries
  - WordNet for English
    - capture semantic knowledge about words
    - each word is associated with one or several synsets
    - synsets are linked to each other according to semantic relations between their words
    - semantic relations are: hypernym (more specific), hyponym (less specific), antonyms, holonyms (part-whole), and meronyms (whole-part)
    - contains information about nouns, verbs, adjectives, and adverbs
    - manually created
  - FrameNet and VerbNet for English
    - focus around verbs

Some Terminologies

- lexical resources: provide some information about words and their relations
  - dictionaries
  - WordNet for English
    - capture semantic knowledge about words
    - each word is associated with one or several synsets
    - synsets are linked to each other according to semantic relations between their words
    - semantic relations are: hypernym (more specific), hyponym (less specific), antonyms, holonyms (part-whole), and meronyms (whole-part)
    - contains information about nouns, verbs, adjectives, and adverbs
    - manually created
  - FrameNet and VerbNet for English
    - focus around verbs
    - manually created

Some Terminologies

- lexical resources: provide some information about words and their relations
  - dictionaries
  - WordNet for English
    - capture semantic knowledge about words
    - each word is associated with one or several synsets
    - synsets are linked to each other according to semantic relations between their words
    - semantic relations are: hypernym (more specific), hyponym (less specific), antonyms, holonyms (part-whole), and meronyms (whole-part)
    - contains information about nouns, verbs, adjectives, and adverbs
    - manually created
  - FrameNet and VerbNet for English
    - focus around verbs
    - manually created
  - Paraphrase Database (PPDB)

Some Terminologies

- lexical resources: provide some information about words and their relations
  - dictionaries
  - WordNet for English
    - capture semantic knowledge about words
    - each word is associated with one or several synsets
    - synsets are linked to each other according to semantic relations between their words
    - semantic relations are: hypernym (more specific), hyponym (less specific), antonyms, holonyms (part-whole), and meronyms (whole-part)
    - contains information about nouns, verbs, adjectives, and adverbs
    - manually created
  - FrameNet and VerbNet for English
    - focus around verbs
    - manually created
  - Paraphrase Database (PPDB)
    - contains paraphrases

Some Terminologies

- lexical resources: provide some information about words and their relations
  - dictionaries
  - WordNet for English
    - capture semantic knowledge about words
    - each word is associated with one or several synsets
    - synsets are linked to each other according to semantic relations between their words
    - semantic relations are: hypernym (more specific), hyponym (less specific), antonyms, holonyms (part-whole), and meronyms (whole-part)
    - contains information about nouns, verbs, adjectives, and adverbs
    - manually created
  - FrameNet and VerbNet for English
    - focus around verbs
    - manually created
  - Paraphrase Database (PPDB)
    - contains paraphrases
    - automatically created

Common Features for Textual Data

what information can we extract directly from a word

- letters comprising a word and their order

Common Features for Textual Data

what information can we extract directly from a word

- letters comprising a word and their order
- length of word

Common Features for Textual Data

what information can we extract directly from a word

- letters comprising a word and their order
- length of word
- is the first letter capitalized?

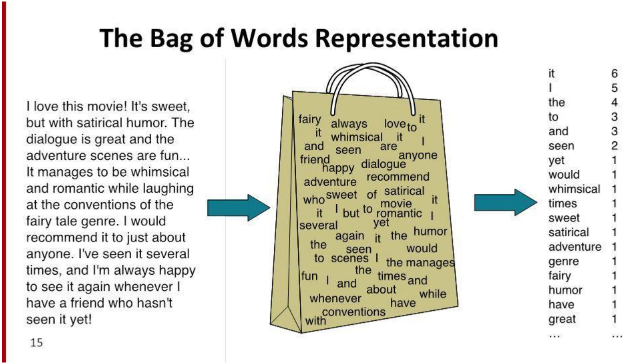
Common Features for Textual Data

what information can we extract directly from a word

- letters comprising a word and their order
- length of word
- is the first letter capitalized?
- does word include hyphen?

Common Features for Textual Data

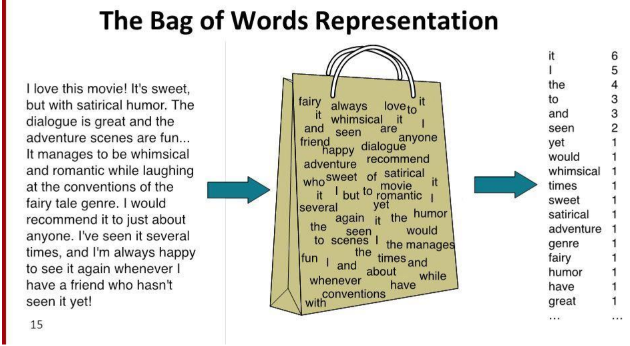
- Bag-of-Words (BoW): the count of each word in a text is taken as a feature



(Taken from:https://www.programmersought.com/article/4304366575/)

Common Features for Textual Data

- Bag-of-Words (BoW): the count of each word in a text is taken as a feature



- BOW does not care about the order of words

(Taken from: https://www.programmersought.com/article/4304366575/)

Common Features for Textual Data

- TF-IDF: Term Frequency - Inverse Document Frequency

Common Features for Textual Data

- TF-IDF: Term Frequency - Inverse Document Frequency
- let d be a document from a given corpus D

Common Features for Textual Data

- TF-IDF: Term Frequency - Inverse Document Frequency
- let d be a document from a given corpus D
- we map word w of d to a number as follows:

$$\frac{\#(w,d)}{\sum_{w'} \#(w',d)} \times \log \frac{|D|}{|\{d \in D : w \in d\}|}$$

Common Features for Textual Data

- TF-IDF: Term Frequency - Inverse Document Frequency
- let d be a document from a given corpus D
- we map word w of d to a number as follows:

$$\frac{\#(w, d)}{\sum_{w'} \#(w', d)} \times \log \frac{|D|}{|\{d \in D : w \in d\}|}$$

- N-grams: instead of using the frequency of a word, we use the frequency of N sequence of words

Common Features for Textual Data

- TF-IDF: Term Frequency - Inverse Document Frequency
- let d be a document from a given corpus D
- we map word w of d to a number as follows:

$$\frac{\#(w,d)}{\sum_{w'} \#(w',d)} \times \log \frac{|D|}{|\{d \in D : w \in d\}|}$$

- N-grams: instead of using the frequency of a word, we use the frequency of N sequence of words
- N=2 $\rightarrow$ bi-gram

Common Features for Textual Data

- TF-IDF: Term Frequency - Inverse Document Frequency
- let d be a document from a given corpus D
- we map word w of d to a number as follows:

$$\frac{\#(w,d)}{\sum_{w'} \#(w',d)} \times \log \frac{|D|}{|\{d \in D : w \in d\}|}$$

- N-grams: instead of using the frequency of a word, we use the frequency of N sequence of words
- N=2 $\rightarrow$ bi-gram
- N=3 $\rightarrow$ tri-gram

Common Features for Textual Data

- context: each piece of a text occurs within a larger text which is known as context

Common Features for Textual Data

- context: each piece of a text occurs within a larger text which is known as context
- how to encode contextual information?

Common Features for Textual Data

TECHNISCHE UNIVERSITÄT DARMSTADT

- context: each piece of a text occurs within a larger text which is known as context
- how to encode contextual information?
- using position of a word within a sentence or a document

Common Features for Textual Data

- context: each piece of a text occurs within a larger text which is known as context
- how to encode contextual information?
- using position of a word within a sentence or a document
- using the words that appear in an immediate context of a word

Common Features for Textual Data

- context: each piece of a text occurs within a larger text which is known as context
- how to encode contextual information?
- using position of a word within a sentence or a document
- using the words that appear in an immediate context of a word
  - immediate context: a window of k words surrounding the target word

Common Features for Textual Data

- context: each piece of a text occurs within a larger text which is known as context
- how to encode contextual information?
- using position of a word within a sentence or a document
- using the words that appear in an immediate context of a word
  - immediate context: a window of k words surrounding the target word
  - "the cat sat on the mat" , k=2 $\rightarrow$ { word-minus-2=the, word-minus-1=cat, word-plus-1=on, word-plus-2=the }

Common Features for Textual Data

what information can we extract from the relation of text with external source of information?

- is a word in a list of common person names in Germany?

Common Features for Textual Data

what information can we extract from the relation of text with external source of information?

- is a word in a list of common person names in Germany?
- is a word a female or male person name?

Common Features for Textual Data

what information can we extract from the relation of text with external source of information?

- is a word in a list of common person names in Germany?
- is a word a female or male person name?
- what is the lemma of the word?

Common Features for Textual Data

what information can we extract from the relation of text with external source of information?

- is a word in a list of common person names in Germany?
- is a word a female or male person name?
- what is the lemma of the word?
- what is the stem of the word?

Common Features for Textual Data

what information can we extract from the relation of text with external source of information?

- is a word in a list of common person names in Germany?
- is a word a female or male person name?
- what is the lemma of the word?
- what is the stem of the word?
- what information do lexical resources give us about the word?

Feature Vectors

- the output of a learning algorithm is a $f()$

Feature Vectors

- the output of a learning algorithm is a $f()$
- f takes as input a vector x with dimension d_{in}

Feature Vectors

- the output of a learning algorithm is a $f()$
- f takes as input a vector x with dimension d_{in}
- f returns as output a vector $\hat{y}$ with dimension d_{out}

Feature Vectors

- the output of a learning algorithm is a $f()$
- f takes as input a vector x with dimension d_{in}
- f returns as output a vector $\hat{y}$ with dimension d_{out}
- how can we map textual features to a vector?

Different Types of Features

- numerical features
  - the value of a feature is a number

Different Types of Features

- numerical features
  - the value of a feature is a number
  - e.g., the frequency of a word in a text

Different Types of Features

- numerical features
  - the value of a feature is a number
  - e.g., the frequency of a word in a text
- categorical features
  - the value of a feature is from a set of values

Different Types of Features

- numerical features
  - the value of a feature is a number
  - e.g., the frequency of a word in a text
- categorical features
  - the value of a feature is from a set of values
  - what is the POS of a word?

Different Types of Features

- numerical features
  - the value of a feature is a number
  - e.g., the frequency of a word in a text
- categorical features
  - the value of a feature is from a set of values
  - what is the POS of a word?
- how to encode categorical features?

Different Types of Features

- numerical features
  - the value of a feature is a number
  - e.g., the frequency of a word in a text
- categorical features
  - the value of a feature is from a set of values
  - what is the POS of a word?
- how to encode categorical features?
  - one-hot encodings

Different Types of Features

- numerical features
  - the value of a feature is a number
  - e.g., the frequency of a word in a text
- categorical features
  - the value of a feature is from a set of values
  - what is the POS of a word?
- how to encode categorical features?
  - one-hot encodings
  - dense embeddings

One-hot Encodings

- let's assume that values of a feature is in categories $\{v_0, v_1, v_2\}$

One-hot Encodings

- let's assume that values of a feature is in categories $\{v_0, v_1, v_2\}$
- we encode the space of feature values via vectors in which

One-hot Encodings

- let's assume that values of a feature is in categories $\{v_0, v_1, v_2\}$
- we encode the space of feature values via vectors in which
  - each entry is either 0 or 1

One-hot Encodings

- let's assume that values of a feature is in categories $\{v_0, v_1, v_2\}$
- we encode the space of feature values via vectors in which
  - each entry is either 0 or 1
  - each entry is associated with one of the categories

One-hot Encodings

TECHNISCHE UNIVERSITÄT DARMSTADT

- let's assume that values of a feature is in categories $\{v_0, v_1, v_2\}$
- we encode the space of feature values via vectors in which
  - each entry is either 0 or 1
  - each entry is associated with one of the categories
  - only one item in a vector can be 1, the rest should be 0

One-hot Encodings

TECHNISCHE UNIVERSITÄT DARMSTADT

- let's assume that values of a feature is in categories $\{v_0, v_1, v_2\}$
- we encode the space of feature values via vectors in which
  - each entry is either 0 or 1
  - each entry is associated with one of the categories
  - only one item in a vector can be 1, the rest should be 0
- for example:
  - $v_0 = [1, 0, 0]$
  - $v_1 = [0, 1, 0]$
  - $v_2 = [0, 0, 1]$

Example: Categorical Labels

- let $f()$ take a vector x which encodes a text and also return a vector $\hat{y}$ representing the topic of the text which can be from $\{$news, sport, science, politic$\}$

Example: Categorical Labels

- let $f()$ take a vector x which encodes a text and also return a vector $\hat{y}$ representing the topic of the text which can be from $\{$news, sport, science, politic$\}$
- how can we represent topic labels using one-hot encoding?

Example: Categorical Labels

- let $f()$ take a vector x which encodes a text and also return a vector $\hat{y}$ representing the topic of the text which can be from $\{$news, sport, science, politic$\}$
- how can we represent topic labels using one-hot encoding?
  - news $= [1, 0, 0, 0]$

Example: Categorical Labels

- let $f()$ take a vector x which encodes a text and also return a vector $\hat{y}$ representing the topic of the text which can be from $\{$news, sport, science, politic$\}$
- how can we represent topic labels using one-hot encoding?
  - news $= [1, 0, 0, 0]$
  - sport $= [0, 1, 0, 0]$

Example: Categorical Labels

TECHNISCHE UNIVERSITÄT DARMSTADT

- let $f()$ take a vector x which encodes a text and also return a vector $\hat{y}$ representing the topic of the text which can be from $\{$news, sport, science, politic$\}$
- how can we represent topic labels using one-hot encoding?
  - news $= [1, 0, 0, 0]$
  - sport $= [0, 1, 0, 0]$
  - science $= [0, 0, 1, 0]$

Example: Categorical Labels

- let $f()$ take a vector x which encodes a text and also return a vector $\hat{y}$ representing the topic of the text which can be from $\{$news, sport, science, politic$\}$
- how can we represent topic labels using one-hot encoding?
  - news $= [1, 0, 0, 0]$
  - sport $= [0, 1, 0, 0]$
  - science $= [0, 0, 1, 0]$
  - politic $= [0, 0, 0, 1]$

Example: Categorical Labels

- let $f()$ take a vector x which encodes a text and also return a vector $\hat{y}$ representing the topic of the text which can be from $\{$news, sport, science, politic$\}$
- how can we represent topic labels using one-hot encoding?
  - news $= [1, 0, 0, 0]$
  - sport $= [0, 1, 0, 0]$
  - science $= [0, 0, 1, 0]$
  - politic $= [0, 0, 0, 1]$
- what is the length of one-hot vectors for a feature with k categories?

Example: Word Encodings

- let V be vocabulary of a language

Example: Word Encodings

- let V be vocabulary of a language
- we take V as categories a word can take in a text

Example: Word Encodings

- let V be vocabulary of a language
- we take V as categories a word can take in a text
- so words can be encoded by one-hot vectors

Example: Word Encodings

- let V be vocabulary of a language
- we take V as categories a word can take in a text
- so words can be encoded by one-hot vectors
- let $V = \{v_0, v_1, v_2, ..., v_{|V|-1}\}$

Example: Word Encodings

TECHNISCHE UNIVERSITÄT DARMSTADT

- let V be vocabulary of a language
- we take V as categories a word can take in a text
- so words can be encoded by one-hot vectors
- let $V = \{v_0, v_1, v_2, ..., v_{|V|-1}\}$
  - $v_0 = [1, 0, 0, ..., 0]$

Example: Word Encodings

TECHNISCHE UNIVERSITÄT DARMSTADT

- let V be vocabulary of a language
- we take V as categories a word can take in a text
- so words can be encoded by one-hot vectors
- let $V = \{v_0, v_1, v_2, ..., v_{|V|-1}\}$
  - $v_0 = [1, 0, 0, ..., 0]$
  - $v_1 = [0, 1, 0, ..., 0]$

Example: Word Encodings

- let V be vocabulary of a language
- we take V as categories a word can take in a text
- so words can be encoded by one-hot vectors
- let $V = \{v_0, v_1, v_2, ..., v_{|V|-1}\}$
  - $v_0 = [1, 0, 0, ..., 0]$
  - $v_1 = [0, 1, 0, ..., 0]$
  - $v_2 = [0, 0, 1, ..., 0]$
  - ...
  - $v_{|V|-1} = [0, 0, 0, ..., 1]$

Example: Word Encodings

- let V be vocabulary of a language
- we take V as categories a word can take in a text
- so words can be encoded by one-hot vectors
- let $V = \{v_0, v_1, v_2, ..., v_{|V|-1}\}$
  - $v_0 = [1, 0, 0, ..., 0]$
  - $v_1 = [0, 1, 0, ..., 0]$
  - $v_2 = [0, 0, 1, ..., 0]$
  - ...
  - $v_{|V|-1} = [0, 0, 0, ..., 1]$
- we can easily represent a text via BOW and then represent each word in BOW with its one-hot vector

Example: Word Encodings

TECHNISCHE UNIVERSITÄT DARMSTADT

- let V be vocabulary of a language
- we take V as categories a word can take in a text
- so words can be encoded by one-hot vectors
- let $V = \{v_0, v_1, v_2, ..., v_{|V|-1}\}$
  - $v_0 = [1, 0, 0, ..., 0]$
  - $v_1 = [0, 1, 0, ..., 0]$
  - $v_2 = [0, 0, 1, ..., 0]$
  - ...
  - $v_{|V|-1} = [0, 0, 0, ..., 1]$
- we can easily represent a text via BOW and then represent each word in BOW with its one-hot vector
- how can we define vocabulary V given a corpus D?

Comment

pause

One-hot Encodings

- word vectors are very sparse

One-hot Encodings

TECHNISCHE UNIVERSITÄT DARMSTADT

- word vectors are very sparse
- semantic relations between words are not encoded in word vectors

One-hot Encodings

- word vectors are very sparse
- semantic relations between words are not encoded in word vectors
- it's better to use one-hot representations for a few distinct features where we expect no correlation between features

Dense Encodings

- a categorical feature is embedded as a vector in a d dimensional space

Dense Encodings

- a categorical feature is embedded as a vector in a d dimensional space
- assuming categories $C = \{c_0, c_1, ..., c_{|C|-1}\}$, $d = 4$

Dense Encodings

- a categorical feature is embedded as a vector in a d dimensional space
- assuming categories $C = \{c_0, c_1, ..., c_{|C|-1}\}$, $d = 4$
  - $c_0 = [+0.1, -0.2, +0.3, +0.5]$

Dense Encodings

- a categorical feature is embedded as a vector in a d dimensional space
- assuming categories $C = \{c_0, c_1, ..., c_{|C|-1}\}$, $d = 4$
    - $c_0 = [+0.1, -0.2, +0.3, +0.5]$
    - $c_1 = [-0.2 - 0.1, +0.1, +0.2]$
    - ...
    - $c_{|C|-1} = [+0.2, -0.2, -0.1, +0.3]$

Example: Word Encodings

TECHNISCHE UNIVERSITÄT DARMSTADT

- assuming vocabulary $V = \{v_0, v_1, ..., v_{|V|-1}\}$, $d = 4$

Example: Word Encodings

- assuming vocabulary $V = \{v_0, v_1, ..., v_{|V|-1}\}$, $d = 4$
  - $v_0 = [+0.1, -0.2, +0.3, +0.5]$
  - $v_1 = [-0.2 - 0.1, +0.1, +0.2]$
  - ...
  - $v_{|V|-1} = [+0.2, -0.2, -0.1, +0.3]$

Example: Word Encodings

- assuming vocabulary $V = \{v_0, v_1, ..., v_{|V|-1}\}$, $d = 4$
  - $v_0 = [+0.1, -0.2, +0.3, +0.5]$
  - $v_1 = [-0.2 - 0.1, +0.1, +0.2]$
  - ...
  - $v_{|V|-1} = [+0.2, -0.2, -0.1, +0.3]$
- in the one-hot method we represent each word with a sparse vector of size $|V|$

Example: Word Encodings

TECHNISCHE UNIVERSITÄT DARMSTADT

- assuming vocabulary $V = \{v_0, v_1, ..., v_{|V|-1}\}$, $d = 4$
  - $v_0 = [+0.1, -0.2, +0.3, +0.5]$
  - $v_1 = [-0.2 - 0.1, +0.1, +0.2]$
  - ...
  - $v_{|V|-1} = [+0.2, -0.2, -0.1, +0.3]$
- in the one-hot method we represent each word with a sparse vector of size $|V|$
- in dense encoding method we represent each word with a dense vector with a small size d

Dense Encodings

- dimensionality of vectors is d

Dense Encodings

- dimensionality of vectors is d
- model training will cause similar features to have similar vectors

Dense Encodings

- dimensionality of vectors is d
- model training will cause similar features to have similar vectors
- it's mainly useful when we expect some correlations between features $\rightarrow$ "cat" and "dog" are semantically related

Dense Encodings

- dimensionality of vectors is d
- model training will cause similar features to have similar vectors
- it's mainly useful when we expect some correlations between features $\rightarrow$ "cat" and "dog" are semantically related
- is also useful when we have a large number of features $\rightarrow$ for example vocabulary

Embeddings

- by embeddings we mean representing each textual feature as a dense vector in a low dimensional space

Embeddings

- by embeddings we mean representing each textual feature as a dense vector in a low dimensional space
- how should we define these dense vectors or embeddings?

Word Embeddings

- dense representations of words in an embedding space is known as word embeddings
- how to obtain word embeddings?

Random Initizalization

- we initialize the embedding vectors to random values

Random Initizalization

- we initialize the embedding vectors to random values
- values are uniformly sampled from numbers in the range

Random Initizalization

- we initialize the embedding vectors to random values
- values are uniformly sampled from numbers in the range
  - $[-\frac{1}{2d}, +\frac{1}{2d}]$ where d is the number of dimensions (Mikolov et al., 2013)

Random Initizalization

- we initialize the embedding vectors to random values
- values are uniformly sampled from numbers in the range
  - $[-\frac{1}{2d}, +\frac{1}{2d}]$ where d is the number of dimensions (Mikolov et al., 2013)
  - $[-\frac{\sqrt{6}}{\sqrt{d}}, +\frac{\sqrt{6}}{\sqrt{d}}]$ (xavier initialization)

Word Embeddings

- dense representations of words in an embedding space is known as word embeddings

Word Embeddings

- dense representations of words in an embedding space is known as word embeddings
- these vectors can be obtained by random initialization and tuned for any NLP task

Word Embeddings

- dense representations of words in an embedding space is known as word embeddings
- these vectors can be obtained by random initialization and tuned for any NLP task
- however, we as human beings can define semantic relations between words independent of any NLP task
  - e.g., “blue” and ’black” are colors
  - e.g., “dog” is more similar to “cat” than to “chair”
  - e.g., “easy” is the opposite of “difficult”

Word Embeddings

- dense representations of words in an embedding space is known as word embeddings
- these vectors can be obtained by random initialization and tuned for any NLP task
- however, we as human beings can define semantic relations between words independent of any NLP task
  - e.g., “blue” and ’black” are colors
  - e.g., “dog” is more similar to “cat” than to “chair”
  - e.g., “easy” is the opposite of “difficult”
- how can we find word embeddings such that vectors of words with similar meaning be close to each other in the embedding space?

Word Meaning

- "you should know a word by the company it keeps" (Frith, 1957)

Distributional Hypothesis

TECHNISCHE UNIVERSITÄT DARMSTADT

- words that occur in the same contexts tend to have similar meanings (Harris, 1945)

Distributional Hypothesis

- words that occur in the same contexts tend to have similar meanings (Harris, 1945)
- how can we model the distributional hypothesis?

Word-Context Matrix

- we count how often a word has occurred with a context in texts from a large corpus

Word-Context Matrix

- we count how often a word has occurred with a context in texts from a large corpus
- context is defined by a window over words

Word-Context Matrix

- we count how often a word has occurred with a context in texts from a large corpus
- context is defined by a window over words
- let V be the set of words in vocabulary and C be the set of possible contexts

Word-Context Matrix

- we count how often a word has occurred with a context in texts from a large corpus
- context is defined by a window over words
- let V be the set of words in vocabulary and C be the set of possible contexts
- word-context matrix M is a two dimensional matrix whose rows are associated with V and columns with C

Word-Context Matrix

- we count how often a word has occurred with a context in texts from a large corpus
- context is defined by a window over words
- let V be the set of words in vocabulary and C be the set of possible contexts
- word-context matrix M is a two dimensional matrix whose rows are associated with V and columns with C
- each entry of the matrix indicates how often a word co-occurs with a context in the given corpus

Word-Context Matrix

- we count how often a word has occurred with a context in texts from a large corpus
- context is defined by a window over words
- let V be the set of words in vocabulary and C be the set of possible contexts
- word-context matrix M is a two dimensional matrix whose rows are associated with V and columns with C
- each entry of the matrix indicates how often a word co-occurs with a context in the given corpus
- this matrix is also known as co-occurrence matrix

Word-Context Matrix

- corpus:
  - I like DL
  - I like NLP
  - I love ML
  - I love NLP
- window size $= 1$

Word-Context Matrix

- corpus:
  - I like DL
  - I like NLP
  - I love ML
  - I love NLP
- window size = 1

	I	like	love	DL	NLP	ML
I	0	2	2	0	0	0
like	2	0	0	1	1	0
love	2	0	0	0	1	1
DL	0	1	0	0	0	0
NLP	0	1	1	0	0	0
ML	0	0	1	0	0	0

Word-Context Matrix

TECHNISCHE UNIVERSITÄT DARMSTADT

- corpus:
  - I like DL
  - I like NLP
  - I love ML
  - I love NLP
- window size = 1

	I	like	love	DL	NLP	ML
I	0	2	2	0	0	0
like	2	0	0	1	1	0
love	2	0	0	0	1	1
DL	0	1	0	0	0	0
NLP	0	1	1	0	0	0
ML	0	0	1	0	0	0

Count-based Embeddings

- for a large corpus, the size of matrix is very large

Count-based Embeddings

- for a large corpus, the size of matrix is very large
- the matrix could be sparse too

Count-based Embeddings

- for a large corpus, the size of matrix is very large
- the matrix could be sparse too
- to encounter this, we may use dimensionality reduction techniques

Count-based Embeddings

- for a large corpus, the size of matrix is very large
- the matrix could be sparse too
- to encounter this, we may use dimensionality reduction techniques
- however, for adding a new word we need to enlarge the matrix and apply dimensionality reduction again

CBoW Method

- CBoW stands for Continuous Bag of Words
- task: given a context $\rightarrow$ predict a missing word from the context

  input: "I _ NLP" $\rightarrow$ output: "like"

CBoW Method

- CBoW stands for Continuous Bag of Words
- task: given a context → predict a missing word from the context

  input: "I _ NLP" → output: "like"

  input: ("I _ NLP", "like") → output: 1 and
  input: ("I _ NLP", "apple") → output: 0

CBoW Method

- given a corpus, we create two sets:

CBoW Method

- given a corpus, we create two sets:
  - positive set (D): consisting of pairs (c, w) where c is a context and w is the correct value for the missing word

CBoW Method

- given a corpus, we create two sets:
  - positive set (D): consisting of pairs (c, w) where c is a context and w is the correct value for the missing word
  - negative set (D'): consisting of pairs (c, w) where c is a context and w is a random value for the missing word

CBoW Method

- given a corpus, we create two sets:
  - positive set (D): consisting of pairs (c, w) where c is a context and w is the correct value for the missing word
  - negative set (D'): consisting of pairs (c, w) where c is a context and w is a random value for the missing word
- we compute the score estimating similarity between context c and word w given context-word pair (c, w)

$$s(c, w) = e(w) \sum_{w_i \in c} e(w_i)$$

where e is a function that maps each word to its embeddings

CBoW Method

- given a corpus, we create two sets:
  - positive set (D): consisting of pairs (c, w) where c is a context and w is the correct value for the missing word
  - negative set (D'): consisting of pairs (c, w) where c is a context and w is a random value for the missing word
- we compute the score estimating similarity between context c and word w given context-word pair (c, w)

$$s(c, w) = e(w) \sum_{w_i \in c} e(w_i)$$

  where e is a function that maps each word to its embeddings
- example

$$s(\mathrm{I\ \_\ NLP}, \mathrm{like}) = e(\mathrm{like})(e(\mathrm{I}) + e(\mathrm{NLP}))$$

CBoW Method

TECHNISCHE UNIVERSITÄT DARMSTADT

- we use the sigmoid function to map the score to a probability

$$P(y = 1|(c, w)) = \frac{1}{1 + e^{-s(c,w)}}$$

CBoW Method

- we use the sigmoid function to map the score to a probability

$$P(y = 1|(c, w)) = \frac{1}{1 + e^{-s(c,w)}}$$

- we should maximize:

$$\frac{1}{|D|} \sum_{(c,w) \in D} P(y = 1|(c, w)) + \frac{1}{|D'|} \sum_{(c,w) \in D'} P(y = 0|(c, w))$$

CBoW Method

- we use the sigmoid function to map the score to a probability

$$P(y = 1|(c, w)) = \frac{1}{1 + e^{-s(c,w)}}$$

- we should maximize:

$$\frac{1}{|D|} \sum_{(c,w) \in D} P(y = 1|(c, w)) + \frac{1}{|D'|} \sum_{(c,w) \in D'} P(y = 0|(c, w))$$

or minimize the following loss

$$L(\Theta) = -\frac{1}{|D|} \sum_{(c,w) \in D} \log P(y = 1|(c, w)) - \frac{1}{|D'|} \sum_{(c,w) \in D'} \log P(y = 0|(c, w))$$

Skip-Gram Method

- we treat words of a context independent from each other

$$P(y = 1 | (c, w)) = \prod_{c_i \in c} P(y = 1 | (w, c_i)) = \prod_{c_i \in c} \frac{1}{1 + e^{-e(w)e(c_i)}}$$

Skip-Gram Method

- we treat words of a context independent from each other

$$P(y=1|(c,w)) = \prod_{c_i \in c} P(y=1|(w,c_i)) = \prod_{c_i \in c} \frac{1}{1+e^{-e(w)e(c_i)}}$$

- loss in Skip-Gram is identical to that in CBoW

Skip-Gram Method

- we treat words of a context independent from each other

$$P(y=1|(c,w)) = \prod_{c_i \in c} P(y=1|(w,c_i)) = \prod_{c_i \in c} \frac{1}{1+e^{-e(w)e(c_i)}}$$

- loss in Skip-Gram is identical to that in CBoW
- we fine-tune parameters (word embeddings) using SGD

CBoW and Skip-Gram

- CBoW loses the order information between context's elements

CBoW and Skip-Gram

- CBoW loses the order information between context's elements
- CBoW allows the use of variable-length contexts

CBoW and Skip-Gram

- CBoW loses the order information between context's elements
- CBoW allows the use of variable-length contexts
- Skip-Gram decouples the dependence between context elements even further than CBoW

CBoW and Skip-Gram

- CBoW loses the order information between context's elements
- CBoW allows the use of variable-length contexts
- Skip-Gram decouples the dependence between context elements even further than CBoW
- Skip-Gram treats each context's element as an independent context

Context Matters!

- context of a word translates to its surrounding words in a sentence or paragraph

Context Matters!

TECHNISCHE UNIVERSITÄT DARMSTADT

- context of a word translates to its surrounding words in a sentence or paragraph
- window-based context: for a word we consider all words in a window of length $2m + 1$ centred on the word as context

Context Matters!

- context of a word translates to its surrounding words in a sentence or paragraph
- window-based context: for a word we consider all words in a window of length $2m + 1$ centred on the word as context
- the size of window: large windows tend to capture more topical similarities and small windows capture syntactic similarities

Context Matters!

- context of a word translates to its surrounding words in a sentence or paragraph
- window-based context: for a word we consider all words in a window of length $2m + 1$ centred on the word as context
- the size of window: large windows tend to capture more topical similarities and small windows capture syntactic similarities
- contexts can be defined based on text units, for example, words that occur in the same sentence, paragraph, or text

Context Matters!

- context of a word translates to its surrounding words in a sentence or paragraph
- window-based context: for a word we consider all words in a window of length $2m + 1$ centred on the word as context
- the size of window: large windows tend to capture more topical similarities and small windows capture syntactic similarities
- contexts can be defined based on text units, for example, words that occur in the same sentence, paragraph, or text
- contexts can also be defined based on syntax of a sentence for example using parse tree or dependency tree of a sentence

Context Matters!

- context of a word translates to its surrounding words in a sentence or paragraph
- window-based context: for a word we consider all words in a window of length $2m + 1$ centred on the word as context
- the size of window: large windows tend to capture more topical similarities and small windows capture syntactic similarities
- contexts can be defined based on text units, for example, words that occur in the same sentence, paragraph, or text
- contexts can also be defined based on syntax of a sentence for example using parse tree or dependency tree of a sentence
- contexts of a word can be foreign words that are aligned to the word in multilingual corpora

Word2Vec Software Package

- the implementations of CBoW and Skip-Gram methods exist in Word2Vec
- `https://code.google.com/archive/p/word2vec/`

Word2Vec Software Package

- the implementations of CBoW and Skip-Gram methods exist in Word2Vec
- `https://code.google.com/archive/p/word2vec/`
- Skip-Gram is more effective in practice

GLoVe (Pennington et al., 2014)

- GloVe: Global Vectors for Word Representation
- GloVe is an unsupervised method for obtaining word embeddings
- GloVe aims at reconciling the advantages of corpus-wide co-occurrence counts and local context windows
- `https://nlp.stanford.edu/projects/glove/`

Evaluating Word Embeddings

- intrinsic

Evaluating Word Embeddings

- intrinsic
  - word similarity tasks
  - word analogy tasks

Evaluating Word Embeddings

- intrinsic
  - word similarity tasks
  - word analogy tasks
- extrinsic

Evaluating Word Embeddings

TECHNISCHE UNIVERSITÄT DARMSTADT

- intrinsic
  - word similarity tasks
  - word analogy tasks
- extrinsic
  - on a downstream NLP task, we compare the performance of two models that differ only in the word embeddings they use

Evaluating Word Embeddings

- intrinsic
  - word similarity tasks
  - word analogy tasks
- extrinsic
  - on a downstream NLP task, we compare the performance of two models that differ only in the word embeddings they use
  - named entity recognition (NER): accuracy

Evaluating Word Embeddings

- intrinsic
  - word similarity tasks
  - word analogy tasks
- extrinsic
  - on a downstream NLP task, we compare the performance of two models that differ only in the word embeddings they use
  - named entity recognition (NER): accuracy
  - machine translation (MT): BLEU score

Evaluating Word Embeddings

- intrinsic
  - word similarity tasks
  - word analogy tasks
- extrinsic
  - on a downstream NLP task, we compare the performance of two models that differ only in the word embeddings they use
  - named entity recognition (NER): accuracy
  - machine translation (MT): BLEU score
  - summarization: ROUGE score

Evaluating Word Embeddings

- intrinsic
  - word similarity tasks
  - word analogy tasks
- extrinsic
  - on a downstream NLP task, we compare the performance of two models that differ only in the word embeddings they use
  - named entity recognition (NER): accuracy
  - machine translation (MT): BLEU score
  - summarization: ROUGE score
  - information retrieval (IR): precision - recall - F1 score

Word Similarity Tasks

- similar words should have similar representations
- dataset: `http://alfonseca.org/eng/research/wordsim353.html`

Word Similarity Tasks

- similar words should have similar representations
- dataset: `http://alfonseca.org/eng/research/wordsim353.html`
- word-1 and word-2 $\rightarrow$ similarity score $\in [0, 10]$

Word Similarity Tasks

- similar words should have similar representations
- dataset:
  `http://alfonseca.org/eng/research/wordsim353.html`
- word-1 and word-2 $\rightarrow$ similarity score $\in [0, 10]$
- our function $f()$ should map two words to a similarity score using the distance between the word vectors of the words

Word Similarity Tasks

- similar words should have similar representations
- dataset:
  `http://alfonseca.org/eng/research/wordsim353.html`
- word-1 and word-2 $\rightarrow$ similarity score $\in [0, 10]$
- our function $f()$ should map two words to a similarity score using the distance between the word vectors of the words
- cosine similarity

$$\text{sim}(w_i, w_j) = \frac{e(w_i)e(w_j)}{||e(w_i)||^2||e(w_j)||^2}$$

Word Analogy Tasks

- A is to B as C to ?

Word Analogy Tasks

- A is to B as C to ?
- Germany is to Berlin as France is to ?
- dataset: `http://download.tensorflow.org/data/questions-words.txt`

Word Analogy Tasks

- A is to B as C to ?
- Germany is to Berlin as France is to ?
- dataset: `http://download.tensorflow.org/data/questions-words.txt`
- capital-common-countries
  - Athens Greece Baghdad $\rightarrow$ Iraq
  - Athens Greece Berlin $\rightarrow$ Germany

Word Analogy Tasks

- A is to B as C to ?
- Germany is to Berlin as France is to ?
- dataset: `http://download.tensorflow.org/data/questions-words.txt`
- capital-common-countries
  - Athens Greece Baghdad $\rightarrow$ Iraq
  - Athens Greece Berlin $\rightarrow$ Germany
- family
  - boy girl brother $\rightarrow$ sister
  - brother sister dad $\rightarrow$ mom

Word Analogy Tasks

- A is to B as C to ?
- Germany is to Berlin as France is to ?
- dataset: `http://download.tensorflow.org/data/questions-words.txt`
- capital-common-countries
  - Athens Greece Baghdad $\rightarrow$ Iraq
  - Athens Greece Berlin $\rightarrow$ Germany
- family
  - boy girl brother $\rightarrow$ sister
  - brother sister dad $\rightarrow$ mom
- currency, adj-to-adverb, comparative, ...

Finding the Prototype of a Group of Words

- if we have a group of words $g = \{w_1, w_2, ..., w_k\}$

Finding the Prototype of a Group of Words

- if we have a group of words $g = \{w_1, w_2, ..., w_k\}$
- the prototype of this group can be computed as follows

$$\text{proto}(g) = \frac{1}{k} \sum_{i \in 1..k} e(w_i)$$

where e maps a word to its embeddings

Finding Similar Words

- words found by embedding-based similarities can be filtered with other types of word similarities

Finding Similar Words

- words found by embedding-based similarities can be filtered with other types of word similarities

Finding Similar Words

- words found by embedding-based similarities can be filtered with other types of word similarities

Finding Similar Words

- cosine similarity

$$\text{sim}(w_i, w_j) = \frac{e(w_i)e(w_j)}{||e(w_i)||^2||e(w_j)||^2}$$

- words found by embedding-based similarities can be filtered with other types of word similarities

Short Text Similarities

- $d_1 = \{w_1^1, w_2^1, ..., w_m^1\}$ and $d_2 = \{w_1^2, w_2^2, ..., w_n^2\}$

Short Text Similarities

- $d_1 = \{w_1^1, w_2^1, ..., w_m^1\}$ and $d_2 = \{w_1^2, w_2^2, ..., w_n^2\}$
- $$\text{sim}(d_1, d_2) = \frac{1}{m.n} \sum_{i=1}^{m} \sum_{j=1}^{n} \cos\left(e(w_i^1), e(w_j^2)\right)$$

Pre-Trained Embeddings

- Word2Vec
  - trained on Google News (100 billion tokens)
- GloVe
  - trained on Wikipedia (6 billion tokens)
  - trained on CommonCrawl (42 and 840 billion tokens)
  - trained on Twitter (27 million tokens)
- many other pre-trained embeddings for different languages
  - `https://fasttext.cc/docs/en/crawl-vectors.html`

Practical Hints

- always try out different word embeddings (consider them as a hyperparameter)

Practical Hints

- always try out different word embeddings (consider them as a hyperparameter)
- results may vary drastically with different embeddings

Practical Hints

- always try out different word embeddings (consider them as a hyperparameter)
- results may vary drastically with different embeddings
- consider source of corpora used to train word embeddings (larger is not always better, a smaller but more domain-focused can be more effective)

Practical Hints

- always try out different word embeddings (consider them as a hyperparameter)
- results may vary drastically with different embeddings
- consider source of corpora used to train word embeddings (larger is not always better, a smaller but more domain-focused can be more effective)
- consider what contexts were used to define similarities

Practical Hints

- always try out different word embeddings (consider them as a hyperparameter)
- results may vary drastically with different embeddings
- consider source of corpora used to train word embeddings (larger is not always better, a smaller but more domain-focused can be more effective)
- consider what contexts were used to define similarities
- it's better to use the same tokenization and text normalization methods that were used for creating word embeddings

Embedding Layers in PyTorch

- an embedding layer (a.k.a lookup table) maps a sequence of word IDs to a sequence of embedding vectors

Embedding Layers in PyTorch

- an embedding layer (a.k.a lookup table) maps a sequence of word IDs to a sequence of embedding vectors
- it uses a lookup table, which is a matrix with size $|V| \times d_{emb}$

Embedding Layers in PyTorch

- an embedding layer (a.k.a lookup table) maps a sequence of word IDs to a sequence of embedding vectors
- it uses a lookup table, which is a matrix with size $|V| \times d_{emb}$
- the i'th row of this matrix contains embeddings of i'th word in vocabulary V

Embedding Layers in PyTorch

- an embedding layer (a.k.a lookup table) maps a sequence of word IDs to a sequence of embedding vectors
- it uses a lookup table, which is a matrix with size $|V| \times d_{emb}$
- the i'th row of this matrix contains embeddings of i'th word in vocabulary V
- so the only thing we need to do is to replace a word with its index in vocabulary $\rightarrow$ a dictionary does this easily (word_to_ix)

Embedding Layers in PyTorch


```python
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim

torch.manual_seed(1)

word_to_ix = {"hello": 0, "world": 1}

embeds = nn.Embedding(2, 5)

lookup_tensor = torch.tensor([word_to_ix["hello"]], dtype=torch.long)

hello_embed = embeds(lookup_tensor)

print(hello_embed)
```

Embedding Layers in PyTorch

```
import torch
import torch.nn as nn

class EmbeddingLayer(nn.Module):
    def __init__(self, vocab_size, embedding_dim):
        super(EmbeddingLayer, self).__init__()
        self.embeddings = nn.Embedding(vocab_size, embedding_dim)

    def forward(self, inputs):
        embeds = self.embeddings(inputs)
        return embeds
```

Embedding Layers in PyTorch


```python
import torch
import torch.nn as nn

class EmbeddingLayer(nn.Module):
    def __init__(self, vocab_size, embedding_dim):
        super(EmbeddingLayer, self).__init__()
        self.embeddings = nn.Embedding(vocab_size, embedding_dim)

    def forward(self, inputs):
        embeds = self.embeddings(inputs)
        return embeds

if __name__ == '__main__':
    model = EmbeddingLayer(voc_size=10, emb_size=3)
    inputs = [1,5]
    inputs_tenseor = torch.tensor(inputs, dtype=torch.long)

    emb_vectors = model(inputs_tenseor)

    print(f"the shape of emb vectors is {emb_vectors.shape}")
```

Using Pretrained Embedding Layers in PyTorch

TECHNISCHE UNIVERSITÄT DARMSTADT

```
import torch
import torch.nn as nn

class EmbeddingLayer(nn.Module):
    def __init__(self, vocab_size, embedding_dim, weights_matrix):
        super(EmbeddingLayer, self).__init__()
        self.embeddings = nn.Embedding(vocab_size, embedding_dim)

        self.embeddings.load_state_dict({'weight': weights_matrix})


    def forward(self, inputs):
        embeds = self.embeddings(inputs)
        return embeds
```

Freezing Embedding Layers in PyTorch


```
import torch
import torch.nn as nn

class EmbeddingLayer(nn.Module):
    def __init__(self, vocab_size, embedding_dim, weights_matrix, freeze=False):
        super(EmbeddingLayer, self).__init__()
        self.embeddings = nn.Embedding(vocab_size, embedding_dim)

        self.embeddings.load_state_dict({'weight': weights_matrix})


        if freeze:
            self.embeddings.weight.requires_grad = False


    def forward(self, inputs):
        embeds = self.embeddings(inputs)
        return embeds
```

Limitations

- the algorithms discussed provide very little control over the kind of similarity they include

Limitations

- the algorithms discussed provide very little control over the kind of similarity they include
  - "cat" is more similar to "dog" than to "tiger" as they both are pets

Limitations

- the algorithms discussed provide very little control over the kind of similarity they include
  - "cat" is more similar to "dog" than to "tiger" as they both are pets
  - "cat" is more similar to "tiger" than to "dog" as they both as felines

Limitations

- the algorithms discussed provide very little control over the kind of similarity they include
  - "cat" is more similar to "dog" than to "tiger" as they both are pets
  - "cat" is more similar to "tiger" than to "dog" as they both as felines
- many of the trivial properties of words are ignored because people are less likely to mention known information than they are to mention novel one

Limitations

- the algorithms discussed provide very little control over the kind of similarity they include
  - "cat" is more similar to "dog" than to "tiger" as they both are pets
  - "cat" is more similar to "tiger" than to "dog" as they both as felines
- many of the trivial properties of words are ignored because people are less likely to mention known information than they are to mention novel one
  - when people talk about "white sheep", they will likely prefer to use only "sheep"

Limitations

- the algorithms discussed provide very little control over the kind of similarity they include
  - “cat” is more similar to “dog” than to “tiger” as they both are pets
  - “cat” is more similar to “tiger” than to “dog” as they both as felines
- many of the trivial properties of words are ignored because people are less likely to mention known information than they are to mention novel one
  - when people talk about “white sheep”, they will likely prefer to use only “sheep”
  - for “black sheep”, they are very likely to use color information “black sheep”

Limitations

- the algorithms discussed provide very little control over the kind of similarity they include
  - “cat” is more similar to “dog” than to “tiger” as they both are pets
  - “cat” is more similar to “tiger” than to “dog” as they both as felines
- many of the trivial properties of words are ignored because people are less likely to mention known information than they are to mention novel one
  - when people talk about “white sheep”, they will likely prefer to use only “sheep”
  - for “black sheep”, they are very likely to use color information “black sheep”
  - a model trained on texts only might easily misled by this

Limitations

- text corpora can easily be biased for better or worse $\rightarrow$ so word embeddings can become biased too

Limitations

- text corpora can easily be biased for better or worse $\rightarrow$ so word embeddings can become biased too
  - gender and racial biases are very common

Limitations

- text corpora can easily be biased for better or worse $\rightarrow$ so word embeddings can become biased too
  - gender and racial biases are very common
- these word vectors are context independent

Limitations

TECHNISCHE UNIVERSITÄT DARMSTADT

- text corpora can easily be biased for better or worse $\rightarrow$ so word embeddings can become biased too
  - gender and racial biases are very common
- these word vectors are context independent
  - in reality, there is no such a thing to have context independent word meaning

Limitations

- text corpora can easily be biased for better or worse $\rightarrow$ so word embeddings can become biased too
  - gender and racial biases are very common
- these word vectors are context independent
  - in reality, there is no such a thing to have context independent word meaning
  - some words have multiple sense e.g., “bank” may refer to a financial institution or to the side of a river

Summary

TECHNISCHE UNIVERSITÄT DARMSTADT

- common features used for converting textual data into numerical vectors
- basics of word embeddings
  - how to get them?
  - where to use them?
  - how to use them?
- limitations

Thank You!