Advanced Deep Learning

Section 3:

Natural Language Processing

Rényi Alfréd Matematikai Kutatóintézet

ELTE EÖTVÖS LORÁND TUDOMÁNYEGYETEM

Schedule

Lectures on NLP:

- **Lecture 1: Introduction and Foundations**
  - Characteristics of the domain
  - Classical methods
  - Character encodings
  - Tokenization
  - Embeddings
- **Lecture 2: Language Modeling**
  - Objective functions
  - Sequential modeling
  - Decoding strategies
  - Models: Transformers (BERT, GPT)
  - Training: pre-training, fine-tuning
  - Evaluation
- **Lecture 3: Large Language Models (LLMs)**
  - Emergent properties
  - Scaling laws
  - GPT-series
  - Instruction tuning
  - Reinforcement Learning with Human Feedback (RLHF)
- **Lecture 4: Research**
  - Prompt engineering
  - Multimodality: CLIP
  - Problems: hallucination
  - Retrieval Augmented Generation (RAG)
  - Security

Schedule

Lectures on NLP:

- **Lecture 1: Introduction and Foundations**
  - Characteristics of the domain
  - Classical methods
  - Character encodings
  - Tokenization
  - Embeddings
- **Lecture 2: Language Modeling**
  - Objective functions
  - Sequential modeling
  - Decoding strategies
  - Models: Transformers (BERT, GPT)
  - Training: pre-training, fine-tuning
  - Evaluation
- **Lecture 3: Large Language Models (LLMs)**
  - Emergent properties
  - Scaling laws
  - GPT-series
  - Instruction tuning
  - Reinforcement Learning with Human Feedback (RLHF)
- **Lecture 4: Research**
  - Prompt engineering
  - Multimodality: CLIP
  - Problems: hallucination
  - Retrieval Augmented Generation (RAG)
  - Security

DATA: how to transform

TRAINING: how they work

PROPERTIES: what makes powerful

HOT-TOPICS: what can it be used for

gabar92 addig content to overview		514baa7 · 2 months ago 154 Commits
lectures	addig content to lecture 1	2 months ago
LICENSE	Initial commit	2 months ago
README.md	addig content to overview	2 months ago

README Apache-2.0 license

Advanced Deep Learning course on ELTE (2024)

Lecture contents for Advanced Deep Learning lecture in ELTE

Lecture 1: Introduction and Foundations

What is this lecture about?

This lecture tries to answer the following questions:

1. What is the current state of Natural Language Processing (NLP) across various applications?
2. How do we define Natural Language Processing (NLP)?
3. What has led to the rapid adoption of NLP-based applications?
4. What are the key tasks and associated applications in NLP?
5. What are the challenges and advantages of NLP, along with common solutions?
6. How Computer Vision (CV) and NLP inspired each other's best practices?
7. What is the foundational principle underlying nearly every Language Model?
8. How textual data is represented for Language Model processing?

Content of the lecture:

- Motivation: this part is intended to give some motivation and interest in the topic of Natural Language Processing and Large Language Models.
  - **Demo of recent NLP-related applications**:
    - showcasing a couple of recent applications and products which use NLP-related cutting-edge AI advancements
  - **Definition of Natural Language Processing**:
    - giving an informal definition of NLP
    - setting NLP among related disciplines and domains
  - **Reasons behind the quick adoption of NLP-based applications**:
    - focusing on the background factors that made recent NLP-related AI products such successful and

Introduction and Foundation

Lecture 1

What is this lecture about?

This lecture tries to answer the following questions:

- What is the current state of Natural Language Processing (NLP) across various applications?
- How do we define Natural Language Processing (NLP)?
- What has led to the rapid adoption of NLP-based applications?
- What are the key tasks and associated applications in NLP?
- What are the challenges and advantages of NLP, along with common solutions?
- What is the foundational principle underlying nearly every Language Model?
- How textual data is represented for Language Model processing?

Motivation

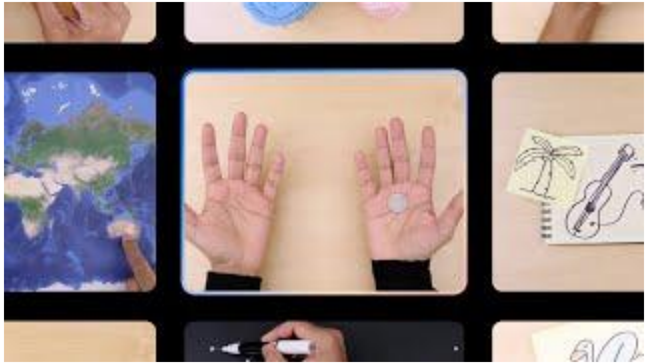
Cutting-edge applications

Applications: state-of-the-art

Text-to-Image

Applications: state-of-the-art

Text-to-Video
Video-to-Text



Applications: state-of-the-art

Voice-to-Text
Text-to-Voice

Text-to-Code

Introduction

AI: Artificial Intelligence
NLP: Natural Language Processing

Definition of NLP

NLP: Natural Language Processing

- Fields:
    - Computer Science
    - Artificial Intelligence
    - Linguistics
- Goal:
    - AI: machines make human intelligible tasks
    - NLP: Enabling computers to **understand**, **interpret**, and **generate** human language
        - **Understand**: grasping the meaning of the words, phrases, or larger units of text
        - **Interpret**: extracting deeper meaning, context, or intent from the text
        - **Generate**: producing human-like text
- Modalities:
    - Textual data: written language
    - Visual data: images and videos
    - Auditory data: speech and audio signals

Reasons behind the success of NLP

Quick adaptation of the field:

- achieving great success
- ChatGPT reached 100 million users faster than any other application to date

Key factors:

- Natural and Convenient Interaction
- Elimination of Technical Barriers
- Unleashing Creativity
- Versatility across Domains
- Global Accessibility
- Human-level Performance
- Ease of Customization

NLP-specific challenges

Variable input length:

- Normalization:
  - Image:
    - Size: image can be resized to the same size of 1024 x 1024
    - Range: pixels / variables can be moved into the [0, 1] range
  - Text:
    - Length: arbitrary

Lack of standard representation:

- (previous point)

Lack of inherent structure:

- Interpolation:
  - Image:
    - spatial dimension: linear, quadratic
    - pixel level: adding two images with 0.5, 0.5 alphas
  - Text:
    - ?

Discrete vs. Continuous:

- Perturbation:
  - Image: small perturbation of an image usually has no effect
  - Text: what is a perturbation

Long sequences:

- how to find relationship between words / tokens distant to each other

Expensive / costly labeling for downstream tasks:

- for a lot of every-day task it is very hard to provide supervised learning setting (labels, ground truth)

NLP-specific challenges

Variable input length:

- Normalization:
  - Image:
    - Size: image can be resized to the same size of 1024 x 1024
    - Range: pixels / variables can be moved into the [0, 1] range
  - Text:
    - Length: arbitrary

Lack of standard representation:

- (previous point)

Lack of inherent structure:

- Interpolation:
  - Image:
    - spatial dimension: linear, quadratic
    - pixel level: adding two images with 0.5, 0.5 alphas
  - Text:
    - ?

Discrete vs. Continuous:

- Perturbation:
  - Image: small perturbation of an image usually has no effect
  - Text: what is a perturbation

Long sequences:

- how to find relationship between words / tokens distant to each other

Expensive / costly labeling for downstream tasks:

- for a lot of every-day task it is very hard to provide supervised learning setting (labels, ground truth)

Solution

Mapping discrete tokens into a vector space:

- **Meaning depends on the context:**
  - mapping words into continuous space
  - a word can have different meanings

NLP-specific advantages

Abundant data:

- on the internet there

Unsupervised and self-supervised training:

- creating supervised dataset is challenging and expensive
- Language Modeling: Next Word Prediction
  - strong learning signal

Efficacy of Transfer Learning:

- learning the basics of the language
- then fine-tuning to downstream tasks

Emergent abilities:

- surprising abilities
  - abilities that are key the performance on complex tasks
- emerging with scaling Language Models
- abilities were not explicitly trained for

Effects on other domains

CV: Computer Vision
NLP: Natural Language Processing
ViT: Vision Transformer
SSL: Self-Supervised Learning
CL: Contrastive Learning

<u>CV → NLP:</u>

- Pre-training → Fine-tuning:
  - CV:
    - supervised fashion
    - pre-training models on large ImageNet (supervised)
    - fine-tuning on the downstream task to deploy model
  - NLP:
    - pre-trained models: large resource need (unsupervised / self-supervised)
    - fine-tuning models: moderate resource need

<u>NLP → CV</u>:

- Unsupervised Learning / Self-Supervised Learning (SSL)
  - NLP:
    - best practice
    - abundant unlabeled data
  - CV:
    - Contrastive Learning (CL)
- Transformer architecture
  - CV: Vision Transformer (ViT)

Tasks

BERT: Bidirectional Encoder Representations from Transformer
GPT: Generative Pre-trained Transformer
RAG: Retrieval-Augmented Generation

Tasks:

- **Text Classification** (spam detection, sentiment analysis)
- **Representation Learning**
- **Text Generation**
- **Question Answering**
- **Machine Translation** (Google Translate)
- **Summarization**
- **Dialogue / Chatbot** (ChatGPT)
- **Information Retrieval** (Search Engines, RAG)
- **Reading Comprehension**

Multimodal:

- **Visual Question Answering**
- **Text-to-Image**; **Image-to-Text**
- **Optical Character Recognition**
- **Text-to-Speech, Text-to-Voice**; **Speech-to-Text, Voice-to-Text**

Tasks

BERT: Bidirectional Encoder Representations from Transformer
GPT: Generative Pre-trained Transformer
RAG: Retrieval-Augmented Generation

Tasks:

- **Text Classification** (spam detection, sentiment analysis)
- **Representation Learning**
- **Text Generation**
- **Question Answering**
- **Machine Translation** (Google Translate)
- **Summarization**
- **Dialogue / Chatbot** (ChatGPT)
- **Information Retrieval** (Search Engines, RAG)
- **Reading Comprehension**

Multimodal:

- **Visual Question Answering**
- **Text-to-Image**; **Image-to-Text**
- **Optical Character Recognition**
- **Text-to-Speech, Text-to-Voice**; **Speech-to-Text, Voice-to-Text**

Hugging Face

Papers With Code

Preprocessing methods

Classical methods

Classical methods:

- **Tokenization**
- **Embedding**
- **Stemming**
- **Lemmatization**
- **Part of Speech (PoS) tagging**
- **Named Entity Recognition (NER)**
- **Chunking**
- **Parsing**
- **Stop Word removal**

Classical methods

Classical methods:

- **Tokenization**:
  - the process of breaking down text into its basic ‘atomic’ units called **tokens**
    - such as words, phrases, or symbols, or other elements
- **Embedding**:
  - words or tokens from a vocabulary are mapped to vectors of real numbers
  - creating a dense and continuous vector space
  - each word / token is represented by a point in this space
  - semantically similar words are located closer to each other
- **Stemming**
- **Lemmatization**
- **Part of Speech (PoS) tagging**
- **Named Entity Recognition (NER)**
- **Chunking**
- **Parsing**
- **Stop Word removal**

Classical methods

Classical methods:

- **Tokenization**
- **Embedding**
- **Stemming**:
  - the process of reducing words to their base or root form by chopping off the ends of words
  - often leading incomplete or incorrect words forms
  - computationally less expensive
  - Examples: better → bet
- **Lemmatization**:
  - similar to stemming
  - lemmatization also reduces words to their base form
  - utilizing a vocabulary and morphological analysis
  - Examples: better → good; am, is, are → be
- **Part of Speech (PoS) tagging**
- **Named Entity Recognition (NER)**
- **Chunking**
- **Parsing**
- **Stop Word removal**

Classical methods

Classical methods:

- **Tokenization**
- **Embedding**
- **Stemming**
- **Lemmatization**
- **Part of Speech (PoS) tagging**:
  - assigning a part of speech to each word in a text
    - e.g., noun, verb, adjective
  - helping in understanding the grammatical structure of sentences and the roles of words in sentences
- **Named Entity Recognition (NER)**:
  - identifying and classifying named entities in text into predefined categories
    - names of persons, organizations, locations
- **Chunking**
- **Parsing**
- **Stop Word removal**

Classical methods

Classical methods:

- **Tokenization**
- **Embedding**
- **Stemming**
- **Lemmatization**
- **Part of Speech (PoS) tagging**
- **Named Entity Recognition (NER)**
- **Chunking:**
    - aka. shallow parsing
    - extracting phrases from unstructured text and grouping words into chunks based on their parts of speech
    - working on top of POS tagging: grouping words / tokens into chunks
    - Example: "A diligent student studied late in the quiet library."
        - Subject: "A diligent student";
        - Action: "studied late";
        - Location: "in the quiet library".
- **Parsing:**
    - analyzing text, conforming to the rules of a formal grammar
    - involving the syntactic analysis of text
    - the goal is to understand the grammatical structure of sentences
        - identifying subjects, predicates, and objects and how they relate to each other
    - constructing a parse tree that represents the syntactic structure of the sentence
- **Stop Word removal**

Classical methods

Classical methods:

- **Tokenization**
- **Embedding**
- **Stemming**
- **Lemmatization**
- **Part of Speech (PoS) tagging**
- **Named Entity Recognition (NER)**
- **Chunking**
- **Parsing**
- **Stop Word removal:**
  - eliminating common word, such as “that”, “is”, or “at” from text data
  - occurring frequently in the language
  - do not contribute significant information to the meaning of ta text
  - Example: “The dog sits in the door.” → “dog sits door”

Character Encodings

Character encodings: terms

Character set
Character encoding
Character encoding standard
Fixed-length vs. Variable-length encodings

Character encodings: terms

Character set:

- a defined collection of:
  - characters (‘a’, ‘b’, …)
  - symbols (‘$’, ‘♣’, ‘§”, …)
  - control codes (NUL ‘\0’, TAB ‘\t’, LF ‘\n’, …)
- it defines the characters that are recognized and utilized by a computer system or network
- Examples:
  - ASCII character set
  - Unicode character set

Character encoding

Character encoding standard

Fixed-length vs. Variable-length encodings

NUL	DLE	SP	0	@	P	`	p
SOH	DC1	!	1	A	Q	a	q
STX	DC2	"	2	B	R	b	r
ETX	DC3	#	3	C	S	c	s
EOT	DC4	$	4	D	T	d	t
ENQ	NAK	%	5	E	U	e	u
ACK	SYN	&	6	F	V	f	v
BEL	ETB	'	7	G	W	g	w
BS	CAN	(	8	H	X	h	x
HT	EM	)	9	I	Y	i	y
LF	SUB	*	:	J	Z	j	z
VT	ESC	+	;	K	[	k	{
FF	FS	,	<	L	\	l	|
CR	GS	—	=	M	]	m	}
SO	RS	.	>	N	^	n	~
SI	US	/	?	O	_	o	DEL

Character encodings: terms

Character set

Character encoding:

- the process of assigning numbers (**Code points**) to a character set
- allowing characters to be stored, transmitted, and transformed using digital computers
- establishing the rules for converting characters into binary code and back

Character encoding standard:

- a specific character encoding
- Examples:
  - ASCII, Unicode
  - think of character encoding standard as a table that pairs characters with their corresponding unique numbers (Code points)

Fixed-length vs. Variable-length encodings

Samples from the ASCII code table:

Unique Code Point (Decimal)	Character	Description
0	NUL	Null char
1	SOH	Start of Heading
...	...	...
10	\n	Line Feed
11	\v	Vertical Tab
...	...	...
48	0	Digit Zero
49	1	Digit One
...	...	...
65	A	Uppercase A
66	B	Uppercase B
...	...	...
97	a	Lowercase a
98	b	Lowercase b
...	...	...
126	~	Tilde
127	DEL	Delete

Character encodings: terms

Character set

Character encoding

Character encoding standard

Fixed-length vs. Variable-length encodings:

- **Fixed-length encoding**:
    - each character is represented by the same number of bytes
    - Example: UTF-32
- **Variable-length encoding**:
    - different characters may have different byte lengths
    - Example: UTF-8, UTF-16

```
U:      85
n:     110
i:     105
c:      99
o:     111
d:     100
e:     101
 :      32
π:     960
†:    8224
😄: 128516
```

- UTF-32:

```
U: 00000000 00000000 00000000 01010101
n: 00000000 00000000 00000000 01101110
i: 00000000 00000000 00000000 01101001
c: 00000000 00000000 00000000 01100011
o: 00000000 00000000 00000000 01101111
d: 00000000 00000000 00000000 01100100
e: 00000000 00000000 00000000 01100101
 : 00000000 00000000 00000000 00100000
π: 00000000 00000000 00000011 11000000
†: 00000000 00000000 00100000 00100000
😄: 00000000 00000001 11110110 00000100
```

- UTF-8:

```
U: 01010101
n: 01101110
i: 01101001
c: 01100011
o: 01101111
d: 01100100
e: 01100101
 : 00100000
π: 11001111 10000000
†: 11100010 10000000 10100000
😄: 11110000 10011111 10011000 10000100
```

Character encodings: standards

Character encoding standards:

- **ASCII**
- **extended ASCII**
- **ISO/IEC 8859**
- **Windows-1252**
- **Unicode**

Character encodings: standards

Character encoding standards:

- **ASCII**:
    - American Standard Code for Information Interchange
        - designed for represent the English alphabet
    - introduced in 1963
    - utilizes 7-bit code points
        - 128 different characters
- **extended ASCII**:
    - in 1970s
    - utilizes 8-bit code points
        - 256 different characters
    - the characters 128-255 were not standardized universally
- **ISO/IEC 8859**
- **Windows-1252**
- **Unicode**

Character encodings: standards

Character encoding standards:

- **ASCII**
- **extended ASCII**
- **ISO/IEC 8859**:
    - introduced in 1987
    - 8-bit code points
        - 256 different characters
    - accommodating the needs of most other languages that use the Latin alphabet
    - **ISO/IEC 8859-1 (Latin-1)**:
        - covers most Western European languages
            - Hungarian is missing a few specific letters
    - **ISO/IEC 8859-2 (Latin-2)**:
        - covers most Central or Eastern European languages
            - provides complete coverage for Hungarian
- **Windows-1252**
- **Unicode**

Character encodings: standards

Character encoding standards:

- **ASCII**
- **extended ASCII**
- **ISO/IEC 8859**
- **Windows-1252**:
  - introduced in 1985
  - the most-used 8-bit single byte character encoding in the world
  - Hungarian is not supported completely:
    - Windows-1250 supports Hungarian completely
- **Unicode**

Character encodings: standards

Character encoding standards:

- **ASCII**
- **extended ASCII**
- **ISO/IEC 8859**
- **Windows-1252**
- **Unicode**:
    - introduced in the late 1980s
    - designed to support written text in all the world's major writing systems
        - including symbols and emojis
    - can represent 1,114,112 characters
    - the current version, as of the last update, is 15.1, which includes 149,813 characters
    - compatible with ASCII
        - the first 128 code points in Unicode are identical to ASCII
    - Unicode text is processed and stored as binary data using one of the several encodings:
        - UTF-32, UTF-16, UTF-8

Character encodings

Unicode Transformation Format (UTF):

- **UTF-32**:
  - fixed-length format
  - each code point is represented using 4 bytes (32 bits)
  - simplifying certain computing operations
    - easier to calculate the position of a particular character within a sequence
  - wasteful in terms of storage and bandwidth
    - especially for Latin characters
- **UTF-16**
- **UTF-8**

```
 U:     85
 n:    110
 i:    105
 c:     99
 o:    111
 d:    100
 e:    101
  :     32
 π:    960
 †:   8224
😄: 128516
```

■ UTF-32:

```
 U: 00000000 00000000 00000000 01010101
 n: 00000000 00000000 00000000 01101110
 i: 00000000 00000000 00000000 01101001
 c: 00000000 00000000 00000000 01100011
 o: 00000000 00000000 00000000 01101111
 d: 00000000 00000000 00000000 01100100
 e: 00000000 00000000 00000000 01100101
  : 00000000 00000000 00000000 00100000
 π: 00000000 00000000 00000011 11000000
 †: 00000000 00000000 00100000 00100000
😄: 00000000 00000001 11110110 00000100
```

Character encodings

Unicode Transformation Format (UTF):

- **UTF-32**
- **UTF-16**:
  - variable-length encoding scheme
    - can use 1 or 2 units (each unit is 2 bytes) for encoding a code point
- **UTF-8**

```
U:     85
n:    110
i:    105
c:     99
o:    111
d:    100
e:    101
 :     32
π:    960
†:   8224
😄: 128516
```

■ UTF-16:

```
U: 00000000 01010101
n: 00000000 01101110
i: 00000000 01101001
c: 00000000 01100011
o: 00000000 01101111
d: 00000000 01100100
e: 00000000 01100101
 : 00000000 00100000
π: 00000011 11000000
†: 00100000 00100000
😄: 11011000 00111101 11011110 00000100
```

Character encodings

Unicode Transformation Format (UTF):

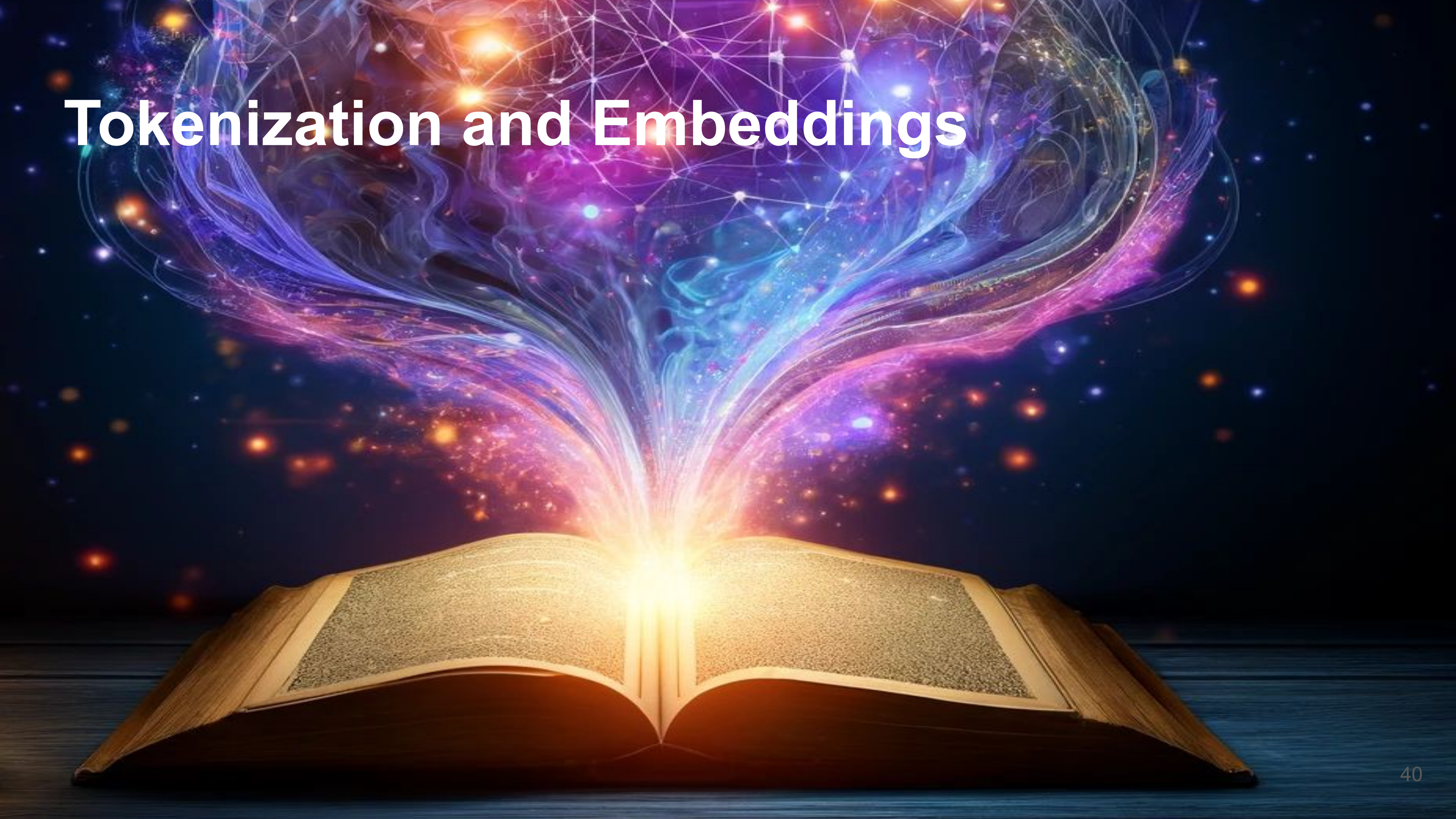
- **UTF-32**
- **UTF-16**
- **UTF-8**:
    - variable-length encoding scheme
        - can use 1 to 4 bytes for encoding a code point
    - designed to be backward compatible with ASCII
        - ASCII text is also valid UTF-8 encoded text
    - the most flexible and space-efficient encoding for a wide range of languages
    - the world’s most frequently used character encoding

```
 U:     85
 n:    110
 i:    105
 c:     99
 o:    111
 d:    100
 e:    101
  :     32
 π:    960
 †:   8224
😄: 128516
```

■ UTF-8:

```
 U: 01010101
 n: 01101110
 i: 01101001
 c: 01100011
 o: 01101111
 d: 01100100
 e: 01100101
  : 00100000
 π: 11001111 10000000
 †: 11100010 10000000 10100000
😄: 11110000 10011111 10011000 10000100
```

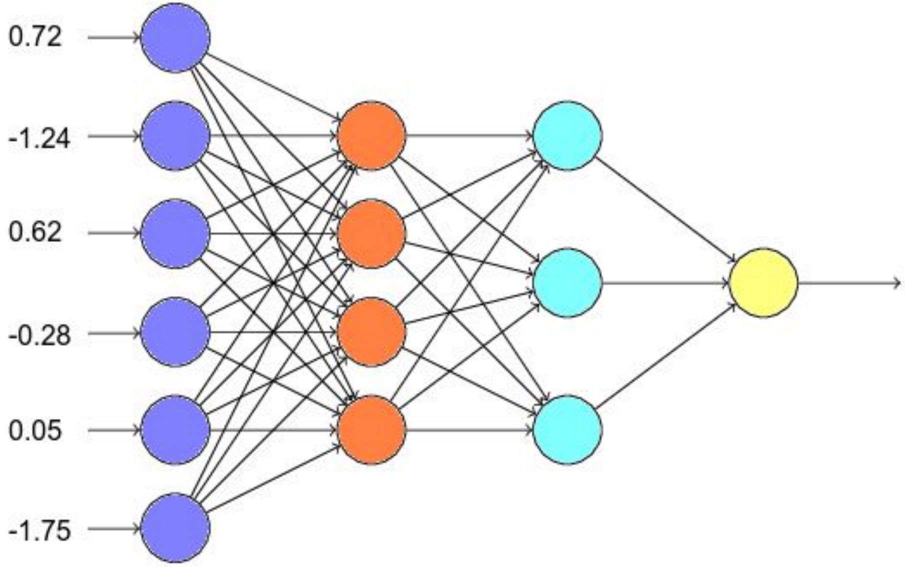
Tokenization and Embeddings



Tokenization and Embeddings

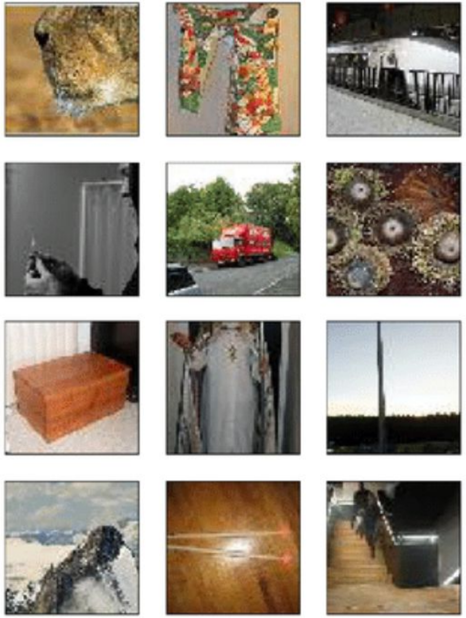
Input representation: Tabular data + MLP

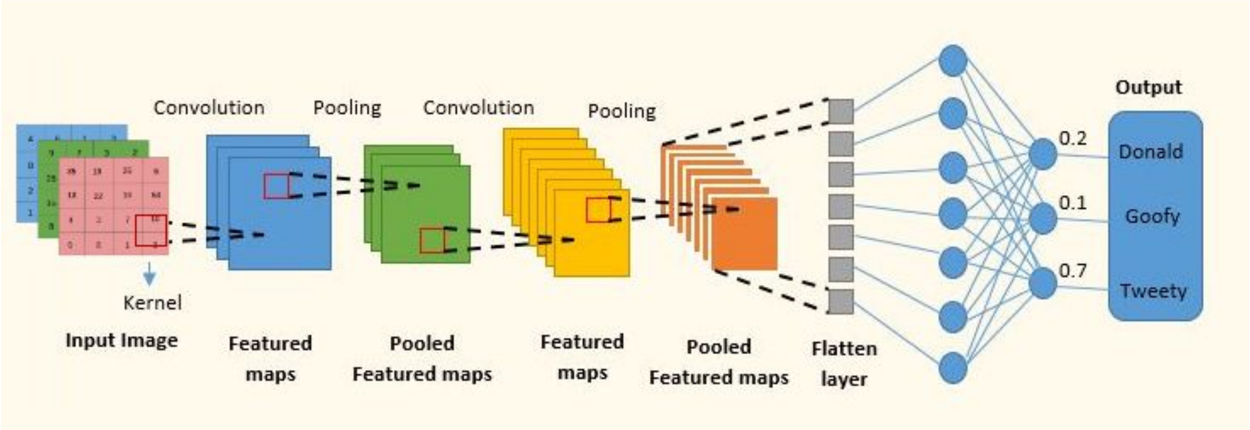
	0	1	2
id	7129300520	6414100192	5631500400
date	10/13/2014	12/9/2014	2/25/2015
price	221900	538000	180000
bedrooms	3	3	2
bathrooms	1	2.25	1
sqft_living	1180	2570	770
sqft_lot	5650	7242	10000
floors	1	2	1
waterfront	0	0	0
view	0	0	0
condition	3	3	3
grade	7	7	6
sqft_above	1180	2170	770
sqft_basement	0	400	0
yr_built	1955	1951	1933
yr_renovated	0	1991	0
zipcode	98178	98125	98028



Tokenization and Embeddings

Input representation: Image data + ConvNet





[Source]

Tokenization and Embeddings

Input representation:

Text data → ?

Tokenization is the process of breaking down text into smaller, manageable units such as words, characters, or subwords for analysis or processing.

Embedding is a technique in machine learning where words or tokens are converted into numerical vectors, representing them in a continuous, multidimensional space to capture their meanings, relationships, and context.

Text

Tokenization and Embeddings

Input representation:

Text data → ?

Tokenization is the process of breaking down text into smaller, manageable units such as words, characters, or subwords for analysis or processing.

Embedding is a technique in machine learning where words or tokens are converted into numerical vectors, representing them in a continuous, multidimensional space to capture their meanings, relationships, and context.

Text

Code points

84 111 107 101 110 105 122 97 116 105
111 110 32 105 115 32 116 104 101 32
112 114 111 99 101 115 115 32 111 102
32 98 114 101 97 107 105 110 103 32 100
111 119 110 32 116 101 120 116 32 105
110 116 111 32 115 109 97 108 108 101
114 44 32 109 97 110 97 103 101 97 98
108 101 32 117 110 105 116 115 32 115
117 99 104 32 97 115 32 119 111 114 100
115 44 32 99 104 97 114 97 99 116

01010100 01101111 01101011 01100101
01101110 01101001 01111010 01100001
01110100 01101001 01101111 01101110
00100000 01101001 01110011 00100000
01110100 01101000 01100101 00100000
01110000 01110010 01101111 01100011
01100101 01110011 01110011 00100000
01101111 01100110 00100000 01100010
01110010 01100101 01100001 01101011
01101001 01101110 01100111 00100000

UTF-8 encoding

Tokenization and Embeddings

Computer Science

Input representation:

Text data → ?

Tokenization is the process of breaking down text into smaller, manageable units such as words, characters, or subwords for analysis or processing.

Embedding is a technique in machine learning where words or tokens are converted into numerical vectors, representing them in a continuous, multidimensional space to capture their meanings, relationships, and context.

Text

Code points

84 111 107 101 110 105 122 97 116 105 111 110 32 105 115 32 116 104 101 32 112 114 111 99 101 115 115 32 111 102 32 98 114 101 97 107 105 110 103 32 100 111 119 110 32 116 101 120 116 32 105 110 116 111 32 115 109 97 108 108 101 114 44 32 109 97 110 97 103 101 97 98 108 101 32 117 110 105 116 115 32 115 117 99 104 32 97 115 32 119 111 114 100 115 44 32 99 104 97 114 97 99 116

01010100 01101111 01101011 01100101 01101110 01101001 01111010 01100001 01110100 01101001 01101111 01101110 00100000 01101001 01110011 00100000 01110100 01101000 01100101 00100000 01110000 01110010 01101111 01100011 01100101 01110011 01110011 00100000 01101111 01100110 00100000 01100010 01110010 01100101 01100001 01101011 01101001 01101110 01100111 00100000

UTF-8 encoding

Tokenization and Embeddings

Computer Science = Machine Learning

Input representation:

Text data → ?

Tokenization is the process of breaking down text into smaller, manageable units such as words, characters, or subwords for analysis or processing.

Embedding is a technique in machine learning where words or tokens are converted into numerical vectors, representing them in a continuous, multidimensional space to capture their meanings, relationships, and context.

Text

Code points

84 111 107 101 110 105 122 97 116 105 111 110 32 105 115 32 116 104 101 32 112 114 111 99 101 115 115 32 111 102 32 98 114 101 97 107 105 110 103 32 100 111 119 110 32 116 101 120 116 32 105 110 116 111 32 115 109 97 108 108 101 114 44 32 109 97 110 97 103 101 97 98 108 101 32 117 110 105 116 115 32 115 117 99 104 32 97 115 32 119 111 114 100 115 44 32 99 104 97 114 97 99 116

01010100 01101111 01101011 01100101 01101110 01101001 01111010 01100001 01110100 01101001 01101111 01101110 00100000 01101001 01110011 00100000 01110100 01101000 01100101 00100000 01110000 01110010 01101111 01100011 01100101 01110011 01110011 00100000 01101111 01100110 00100000 01100010 01110010 01100101 01100001 01101011 01101001 01101110 01100111 00100000

UTF-8 encoding

Vector

0.36
1.23
-2.01
0.97
7.68
3.21
-2.91
0.78
-3.85
2.94

3.16	-1.36	0.16	1.36
1.83	-3.23	-8.93	2.23
3.81	5.01	-1.01	-3.01
-0.97	-0.87	0.87	4.97
1.68	3.68	-9.68	6.68
5.21	0.21	1.21	5.71
-1.01	0.91	1.91	-9.91
0.08	5.78	8.78	8.78
-2.15	2.85	7.85	-9.85
1.91	9.94	2.34	0.94

Vector sequence

Tokenization and Embeddings

Text

```
Tokenization is the process of breaking down text into smaller,
manageable units such as words, characters, or subwords for analysis or
processing.
```

Tokenization and Embeddings

Text

```
Tokenization is the process of breaking down text into smaller,
manageable units such as words, characters, or subwords for analysis or
processing.
```

Tokenization

```
Tokenization is the process of breaking down text into smaller,
 manageable units such as words, characters, or subwords for analysis or
 processing.
```

Tokenization and Embeddings

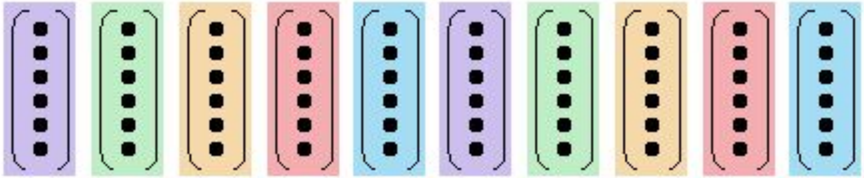
Text

Tokenization is the process of breaking down text into smaller, manageable units such as words, characters, or subwords for analysis or processing.

Tokenization

Tokenization is the process of breaking down text into smaller, manageable units such as words, characters, or subwords for analysis or processing.

Embedding



Tokenization and Embeddings

Terms:

- **Text**:
  - the unprocessed, original form of written content, consisting of a long sequence of characters
  - has a complex web of meaning through its structure and the intricate relationships
- **Vectors**:
  - Neural Networks (NN) interpret textual data through scalar values
    - vectors, matrices, tensors
  - different dimensions of the vector can represent different meanings
- **Vector space**:
  - the space where vectors live
  - the structure of the space can represent the meaning of the text
    - text with similar meaning are positioned closely to each other (clustering)
- **Tokenization**:
  - breaking down the input text into its basic (atomic) units (aka. tokens)
  - Design decisions: how to split?
- **Token**:
  - the fundamental, atomic element processed by the Language Model
  - can be letters, words, subwords
- **Embedding**:
  - a token's vector representation
  - each token has a unique embedding vector
  - encoding meaning in the structure of the vector space
  - Design decisions: dimension of the vectors
- **Vocabulary**:
  - the outcome of the tokenization and embedding processes
  - a collection of all the tokens and the belonging embedding vectors

Tokenization

<u>Considerations</u>:

- **Vocabulary size**
- **Tokenization level**
- **Open Vocabulary Problem and Out of Vocabulary (OOV) words**
- **Special tokens**

Tokenization

Considerations:

- **Vocabulary size**:
  - how many different tokens we have
- **Tokenization level**
- **Open Vocabulary Problem and Out of Vocabulary (OOV) words**
- **Special tokens**

Tokenization

Considerations:

- **Vocabulary size**
- **Tokenization level**:
  - **Word-based tokenization**:
    - Example: “I am batman.” → ‘I’, ‘am’, ‘batman’, ‘.’
    - Advantages:
      - tokens have semantic meaning
      - short token sequence length for a text
        - models limit the length of the token sequence they can digest
    - Disadvantages:
      - large vocabulary size
      - Out-of-Vocabulary (OOV) problem
        - “I am batamn.”; “I am batmna.”; “I am battman."
  - **Character-based tokenization**
  - **Subword-based tokenization**
- **Open Vocabulary Problem and Out of Vocabulary (OOV) words**
- **Special tokens**

Tokenization

Considerations:

- **Vocabulary size**
- **Tokenization level**:
  - **Word-based tokenization**
  - **Character-based tokenization**:
    - Example: “I am batman.” → ‘I’, ‘ ‘, ‘a’, ‘m’, ‘b’, ‘a’, ‘t’, ‘m’, ‘a’, ‘n’, ‘.’
    - Advantages:
      - small vocabulary size
      - handling Out-of-Vocabulary problem
    - Disadvantages:
      - tokens don’t have meaning
      - very long token sequence for a text
  - **Subword-based tokenization**
- **Open Vocabulary Problem and Out of Vocabulary (OOV) words**
- **Special tokens**

Tokenization

Considerations:

- **Vocabulary size**
- **Tokenization level**:
    - **Word-based tokenization**:
    - **Character-based tokenization**
    - **Subword-based tokenization**:
        - Example: “I am batman.” → ‘I’, ‘am’, ‘bat’, ‘man’, ‘.’
        - Advantages:
            - best traits from word and character-based versions
                - medium length for token sequence length
                - medium vocabulary size
                - handling Out-of-Vocabulary problem
                - more frequent words or subwords are included in the vocabulary
                - tokens have semantic meaning
        - Disadvantages:
            - we have to define (learn) the tokens
            - there are hyperparameters we have to explore
- **Open Vocabulary Problem and Out of Vocabulary (OOV) words**
- **Special tokens**

Tokenization

Considerations:

- **Vocabulary size**
- **Tokenization level**
- **Open Vocabulary Problem and Out of Vocabulary (OOV) words**:
  - typos
  - the constant evolution of the languages
  - issue of rare words
    - long-tail distribution of text
    - no data for everything
- **Special tokens**

Tokenization

Considerations:

- **Vocabulary size**
- **Tokenization level**
- **Open Vocabulary Problem and Out of Vocabulary (OOV) words**
- **Special tokens**:
    - <UNK>: representing unknown tokens (e.g., OOV words)
    - <SEP>: separating different parts of the sequence
    - <BOS>: indicating the beginning of the sequence
    - <EOS>: indicating the end of the sequence

Tokenization

Tokenization methods:

- **Byte Pair Encoding (BPE):**
- **WordPiece**
- **Unigram**
- **SentencePiece**

Tokenization

Tokenization methods:

- **Byte Pair Encoding (BPE)**:
  - fundamentally a data compression technique
  - paper introduced: Neural Machine Translation of Rare Words with Subword Units
    - primarily addresses the challenge of tokenizing rare words
  - a subword-based tokenization method
  - **Core concept**:
    - in a bottom-up fashion, merging frequent token pairs and adding as a new token
  - **Process of BPE**:
    1. Initialize the vocabulary with the characters as individual tokens occurring in the dataset
    2. Calculating the frequency of each adjacent character (or byte) within the dataset
    3. The most frequent adjacent pair is merged into a new token (merged tokens are retained)
    4. Repeat step 2. and 3. until the vocabulary reaches a specified size
  - 2 versions os BPE:
    - Character-based implementation of BPE
    - Byte-based implementation of BPE
- **WordPiece**
- **Unigram**
- **SentencePiece**

Byte Pair Encoding (BPE) - Example

Dataset: "she sells seashells by the seashore"

Byte Pair Encoding (BPE) - Example

Dataset: "she sells seashells by the seashore"

Initial vocabulary:

- {'s': 0, 'h': 1, 'e': 2, ' ': 3, 'l': 4, 'a': 5, 'b': 6, 'y': 7, 't': 8, 'o': 9, 'r': 10}

Byte Pair Encoding (BPE) - Example

Dataset: "she sells seashells by the seashore"

Initial vocabulary:

- {'s': 0, 'h': 1, 'e': 2, ' ': 3, 'l': 4, 'a': 5, 'b': 6, 'y': 7, 't': 8, 'o': 9, 'r': 10}

Step 1:

Byte Pair Encoding (BPE) - Example

Dataset: "she sells seashells by the seashore"

Initial vocabulary:

- {'s': 0, 'h': 1, 'e': 2, ' ': 3, 'l': 4, 'a': 5, 'b': 6, 'y': 7, 't': 8, 'o': 9, 'r': 10}

Step 1:

- Tokenized text:
    - [0, 1, 2, 3, 0, 2, 4, 4, 0, 3, 0, 2, 5, 0, 1, 2, 4, 4, 0, 3, 6, 7, 3, 8, 1, 2, 3, 0, 2, 5, 0, 1, 9, 10, 2]

Byte Pair Encoding (BPE) - Example

Dataset: "she sells seashells by the seashore"

Initial vocabulary:

- {'s': 0, 'h': 1, 'e': 2, ' ': 3, 'l': 4, 'a': 5, 'b': 6, 'y': 7, 't': 8, 'o': 9, 'r': 10}

Step 1:

- Tokenized text:
  - [0, 1, 2, 3, 0, 2, 4, 4, 0, 3, 0, 2, 5, 0, 1, 2, 4, 4, 0, 3, 6, 7, 3, 8, 1, 2, 3, 0, 2, 5, 0, 1, 9, 10, 2]
- Frequencies:
  - {(0, 1): 3, (1, 2): 3, (3, 0): 3, (0, 2): 3, (2, 3): 2, (2, 4): 2, (4, 4): 2, (4, 0): 2, (0, 3): 2, (2, 5): 2, (5, 0): 2, (3, 6): 1, (6, 7): 1, (7, 3): 1, (3, 8): 1, (8, 1): 1, (1, 9): 1, (9, 10): 1, (10, 2): 1}

Byte Pair Encoding (BPE) - Example

Dataset: "she sells seashells by the seashore"

Initial vocabulary:

- {'s': 0, 'h': 1, 'e': 2, ' ': 3, 'l': 4, 'a': 5, 'b': 6, 'y': 7, 't': 8, 'o': 9, 'r': 10}

Step 1:

- Tokenized text:
  - [0, 1, 2, 3, 0, 2, 4, 4, 0, 3, 0, 2, 5, 0, 1, 2, 4, 4, 0, 3, 6, 7, 3, 8, 1, 2, 3, 0, 2, 5, 0, 1, 9, 10, 2]
- Frequencies:
  - {(0, 1): 3, (1, 2): 3, (3, 0): 3, (0, 2): 3, (2, 3): 2, (2, 4): 2, (4, 4): 2, (4, 0): 2, (0, 3): 2, (2, 5): 2, (5, 0): 2, (3, 6): 1, (6, 7): 1, (7, 3): 1, (3, 8): 1, (8, 1): 1, (1, 9): 1, (9, 10): 1, (10, 2): 1}

Byte Pair Encoding (BPE) - Example

Dataset: "she sells seashells by the seashore"

Initial vocabulary:

- {'s': 0, 'h': 1, 'e': 2, ' ': 3, 'l': 4, 'a': 5, 'b': 6, 'y': 7, 't': 8, 'o': 9, 'r': 10}

Step 1:

- Tokenized text:
  - [0, 1, 2, 3, 0, 2, 4, 4, 0, 3, 0, 2, 5, 0, 1, 2, 4, 4, 0, 3, 6, 7, 3, 8, 1, 2, 3, 0, 2, 5, 0, 1, 9, 10, 2]
- Frequencies:
  - {(0, 1): 3, (1, 2): 3, (3, 0): 3, (0, 2): 3, (2, 3): 2, (2, 4): 2, (4, 4): 2, (4, 0): 2, (0, 3): 2, (2, 5): 2, (5, 0): 2, (3, 6): 1, (6, 7): 1, (7, 3): 1, (3, 8): 1, (8, 1): 1, (1, 9): 1, (9, 10): 1, (10, 2): 1}

Byte Pair Encoding (BPE) - Example

Dataset: "she sells seashells by the seashore"

Initial vocabulary:

- {'s': 0, 'h': 1, 'e': 2, ' ': 3, 'l': 4, 'a': 5, 'b': 6, 'y': 7, 't': 8, 'o': 9, 'r': 10}

Step 1:

- Tokenized text:
    - [0, 1, 2, 3, 0, 2, 4, 4, 0, 3, 0, 2, 5, 0, 1, 2, 4, 4, 0, 3, 6, 7, 3, 8, 1, 2, 3, 0, 2, 5, 0, 1, 9, 10, 2]
- Frequencies:
    - {(0, 1): 3, (1, 2): 3, (3, 0): 3, (0, 2): 3, (2, 3): 2, (2, 4): 2, (4, 4): 2, (4, 0): 2, (0, 3): 2, (2, 5): 2, (5, 0): 2, (3, 6): 1, (6, 7): 1, (7, 3): 1, (3, 8): 1, (8, 1): 1, (1, 9): 1, (9, 10): 1, (10, 2): 1}

Byte Pair Encoding (BPE) - Example

Dataset: "she sells seashells by the seashore"

Initial vocabulary:

- {'s': 0, 'h': 1, 'e': 2, ' ': 3, 'l': 4, 'a': 5, 'b': 6, 'y': 7, 't': 8, 'o': 9, 'r': 10}

Step 1:

- Tokenized text:
  - [0, 1, 2, 3, 0, 2, 4, 4, 0, 3, 0, 2, 5, 0, 1, 2, 4, 4, 0, 3, 6, 7, 3, 8, 1, 2, 3, 0, 2, 5, 0, 1, 9, 10, 2]
- Frequencies:
  - {(0, 1): 3, (1, 2): 3, (3, 0): 3, (0, 2): 3, (2, 3): 2, (2, 4): 2, (4, 4): 2, (4, 0): 2, (0, 3): 2, (2, 5): 2, (5, 0): 2, (3, 6): 1, (6, 7): 1, (7, 3): 1, (3, 8): 1, (8, 1): 1, (1, 9): 1, (9, 10): 1, (10, 2): 1}
- Merge tokens: new vocabulary:

Byte Pair Encoding (BPE) - Example

Dataset: "she sells seashells by the seashore"

Initial vocabulary:

- {'s': 0, 'h': 1, 'e': 2, ' ': 3, 'l': 4, 'a': 5, 'b': 6, 'y': 7, 't': 8, 'o': 9, 'r': 10}

Step 1:

- Tokenized text:
    - [0, 1, 2, 3, 0, 2, 4, 4, 0, 3, 0, 2, 5, 0, 1, 2, 4, 4, 0, 3, 6, 7, 3, 8, 1, 2, 3, 0, 2, 5, 0, 1, 9, 10, 2]
- Frequencies:
    - {(0, 1): 3, (1, 2): 3, (3, 0): 3, (0, 2): 3, (2, 3): 2, (2, 4): 2, (4, 4): 2, (4, 0): 2, (0, 3): 2, (2, 5): 2, (5, 0): 2, (3, 6): 1, (6, 7): 1, (7, 3): 1, (3, 8): 1, (8, 1): 1, (1, 9): 1, (9, 10): 1, (10, 2): 1}
- Merge tokens: new vocabulary:

Byte Pair Encoding (BPE) - Example

Dataset: "she sells seashells by the seashore"

Initial vocabulary:

- {'s': 0, 'h': 1, 'e': 2, ' ': 3, 'l': 4, 'a': 5, 'b': 6, 'y': 7, 't': 8, 'o': 9, 'r': 10}

Step 1:

- Tokenized text:
  - [0, 1, 2, 3, 0, 2, 4, 4, 0, 3, 0, 2, 5, 0, 1, 2, 4, 4, 0, 3, 6, 7, 3, 8, 1, 2, 3, 0, 2, 5, 0, 1, 9, 10, 2]
- Frequencies:
  - {(0, 1): 3, (1, 2): 3, (3, 0): 3, (0, 2): 3, (2, 3): 2, (2, 4): 2, (4, 4): 2, (4, 0): 2, (0, 3): 2, (2, 5): 2, (5, 0): 2, (3, 6): 1, (6, 7): 1, (7, 3): 1, (3, 8): 1, (8, 1): 1, (1, 9): 1, (9, 10): 1, (10, 2): 1}
- Merge tokens: new vocabulary:

Byte Pair Encoding (BPE) - Example

Dataset: "she sells seashells by the seashore"

Initial vocabulary:

- {'s': 0, 'h': 1, 'e': 2, ' ': 3, 'l': 4, 'a': 5, 'b': 6, 'y': 7, 't': 8, 'o': 9, 'r': 10}

Step 1:

- Tokenized text:
  - [0, 1, 2, 3, 0, 2, 4, 4, 0, 3, 0, 2, 5, 0, 1, 2, 4, 4, 0, 3, 6, 7, 3, 8, 1, 2, 3, 0, 2, 5, 0, 1, 9, 10, 2]
- Frequencies:
  - {(0, 1): 3, (1, 2): 3, (3, 0): 3, (0, 2): 3, (2, 3): 2, (2, 4): 2, (4, 4): 2, (4, 0): 2, (0, 3): 2, (2, 5): 2, (5, 0): 2, (3, 6): 1, (6, 7): 1, (7, 3): 1, (3, 8): 1, (8, 1): 1, (1, 9): 1, (9, 10): 1, (10, 2): 1}
- Merge tokens: new vocabulary:
  - {'s': 0, 'h': 1, 'e': 2, ' ': 3, 'l': 4, 'a': 5, 'b': 6, 'y': 7, 't': 8, 'o': 9, 'r': 10, 'sh': 11}

Byte Pair Encoding (BPE) - Example

Dataset: "she sells seashells by the seashore"

Step 1 vocabulary:

- {'s': 0, 'h': 1, 'e': 2, ' ': 3, 'l': 4, 'a': 5, 'b': 6, 'y': 7, 't': 8, 'o': 9, 'r': 10, 'sh': 11}

Byte Pair Encoding (BPE) - Example

Dataset: "she sells seashells by the seashore"

Step 1 vocabulary:

- {'s': 0, 'h': 1, 'e': 2, ' ': 3, 'l': 4, 'a': 5, 'b': 6, 'y': 7, 't': 8, 'o': 9, 'r': 10, 'sh': 11}

Step 2:

Byte Pair Encoding (BPE) - Example

Dataset: "she sells seashells by the seashore"

Step 1 vocabulary:

- {'s': 0, 'h': 1, 'e': 2, ' ': 3, 'l': 4, 'a': 5, 'b': 6, 'y': 7, 't': 8, 'o': 9, 'r': 10, 'sh': 11}

Step 2:

- Tokenized text:
    - [11, 2, 3, 0, 2, 4, 4, 0, 3, 0, 2, 5, 11, 2, 4, 4, 0, 3, 6, 7, 3, 8, 1, 2, 3, 0, 2, 5, 11, 9, 10, 2]

Byte Pair Encoding (BPE) - Example

Dataset: "she sells seashells by the seashore"

Step 1 vocabulary:

- {'s': 0, 'h': 1, 'e': 2, ' ': 3, 'l': 4, 'a': 5, 'b': 6, 'y': 7, 't': 8, 'o': 9, 'r': 10, 'sh': 11}

Step 2:

- Tokenized text:
  - [11, 2, 3, 0, 2, 4, 4, 0, 3, 0, 2, 5, 11, 2, 4, 4, 0, 3, 6, 7, 3, 8, 1, 2, 3, 0, 2, 5, 11, 9, 10, 2]

Byte Pair Encoding (BPE) - Example

Dataset: "she sells seashells by the seashore"

Step 1 vocabulary:

- {'s': 0, 'h': 1, 'e': 2, ' ': 3, 'l': 4, 'a': 5, 'b': 6, 'y': 7, 't': 8, 'o': 9, 'r': 10, 'sh': 11}

Step 2:

- Tokenized text:
    - [11, 2, 3, 0, 2, 4, 4, 0, 3, 0, 2, 5, 11, 2, 4, 4, 0, 3, 6, 7, 3, 8, 1, 2, 3, 0, 2, 5, 11, 9, 10, 2]
- Frequencies:
    - {(3, 0): 3, (0, 2): 3, (11, 2): 2, (2, 3): 2, (2, 4): 2, (4, 4): 2, (4, 0): 2, (0, 3): 2, (2, 5): 2, (5, 11): 2, (3, 6): 1, (6, 7): 1, (7, 3): 1, (3, 8): 1, (8, 1): 1, (1, 2): 1, (11, 9): 1, (9, 10): 1, (10, 2): 1}

Byte Pair Encoding (BPE) - Example

Dataset: "she sells seashells by the seashore"

Step 1 vocabulary:

- {'s': 0, 'h': 1, 'e': 2, ' ': 3, 'l': 4, 'a': 5, 'b': 6, 'y': 7, 't': 8, 'o': 9, 'r': 10, 'sh': 11}

Step 2:

- Tokenized text:
  - [11, 2, 3, 0, 2, 4, 4, 0, 3, 0, 2, 5, 11, 2, 4, 4, 0, 3, 6, 7, 3, 8, 1, 2, 3, 0, 2, 5, 11, 9, 10, 2]
- Frequencies:
  - {(3, 0): 3, (0, 2): 3, (11, 2): 2, (2, 3): 2, (2, 4): 2, (4, 4): 2, (4, 0): 2, (0, 3): 2, (2, 5): 2, (5, 11): 2, (3, 6): 1, (6, 7): 1, (7, 3): 1, (3, 8): 1, (8, 1): 1, (1, 2): 1, (11, 9): 1, (9, 10): 1, (10, 2): 1}

Byte Pair Encoding (BPE) - Example

Dataset: "she sells seashells by the seashore"

Step 1 vocabulary:

- {'s': 0, 'h': 1, 'e': 2, ' ': 3, 'l': 4, 'a': 5, 'b': 6, 'y': 7, 't': 8, 'o': 9, 'r': 10, 'sh': 11}

Step 2:

- Tokenized text:
  - [11, 2, 3, 0, 2, 4, 4, 0, 3, 0, 2, 5, 11, 2, 4, 4, 0, 3, 6, 7, 3, 8, 1, 2, 3, 0, 2, 5, 11, 9, 10, 2]
- Frequencies:
  - {(3, 0): 3, (0, 2): 3, (11, 2): 2, (2, 3): 2, (2, 4): 2, (4, 4): 2, (4, 0): 2, (0, 3): 2, (2, 5): 2, (5, 11): 2, (3, 6): 1, (6, 7): 1, (7, 3): 1, (3, 8): 1, (8, 1): 1, (1, 2): 1, (11, 9): 1, (9, 10): 1, (10, 2): 1}

Byte Pair Encoding (BPE) - Example

Dataset: "she sells seashells by the seashore"

Step 1 vocabulary:

- {'s': 0, 'h': 1, 'e': 2, ' ': 3, 'l': 4, 'a': 5, 'b': 6, 'y': 7, 't': 8, 'o': 9, 'r': 10, 'sh': 11}

Step 2:

- Tokenized text:
  - [11, 2, 3, 0, 2, 4, 4, 0, 3, 0, 2, 5, 11, 2, 4, 4, 0, 3, 6, 7, 3, 8, 1, 2, 3, 0, 2, 5, 11, 9, 10, 2]
- Frequencies:
  - {(3, 0): 3, (0, 2): 3, (11, 2): 2, (2, 3): 2, (2, 4): 2, (4, 4): 2, (4, 0): 2, (0, 3): 2, (2, 5): 2, (5, 11): 2, (3, 6): 1, (6, 7): 1, (7, 3): 1, (3, 8): 1, (8, 1): 1, (1, 2): 1, (11, 9): 1, (9, 10): 1, (10, 2): 1}

Byte Pair Encoding (BPE) - Example

Dataset: "she sells seashells by the seashore"

Step 1 vocabulary:

- {'s': 0, 'h': 1, 'e': 2, ' ': 3, 'l': 4, 'a': 5, 'b': 6, 'y': 7, 't': 8, 'o': 9, 'r': 10, 'sh': 11}

Step 2:

- Tokenized text:
  - [11, 2, 3, 0, 2, 4, 4, 0, 3, 0, 2, 5, 11, 2, 4, 4, 0, 3, 6, 7, 3, 8, 1, 2, 3, 0, 2, 5, 11, 9, 10, 2]
- Frequencies:
  - {(3, 0): 3, (0, 2): 3, (11, 2): 2, (2, 3): 2, (2, 4): 2, (4, 4): 2, (4, 0): 2, (0, 3): 2, (2, 5): 2, (5, 11): 2, (3, 6): 1, (6, 7): 1, (7, 3): 1, (3, 8): 1, (8, 1): 1, (1, 2): 1, (11, 9): 1, (9, 10): 1, (10, 2): 1}
- Merge tokens: new vocabulary
  - {'s': 0, 'h': 1, 'e': 2, ' ': 3, 'l': 4, 'a': 5, 'b': 6, 'y': 7, 't': 8, 'o': 9, 'r': 10, 'sh': 11, ' s': 12}

Byte Pair Encoding (BPE) - Example

Dataset: "she sells seashells by the seashore"

Step 2 vocabulary:

- {'s': 0, 'h': 1, 'e': 2, ' ': 3, 'l': 4, 'a': 5, 'b': 6, 'y': 7, 't': 8, 'o': 9, 'r': 10, 'sh': 11, ' s': 12}

Byte Pair Encoding (BPE) - Example

Dataset: "she sells seashells by the seashore"

Step 2 vocabulary:

- {'s': 0, 'h': 1, 'e': 2, ' ': 3, 'l': 4, 'a': 5, 'b': 6, 'y': 7, 't': 8, 'o': 9, 'r': 10, 'sh': 11, ' s': 12}

Step 3:

Byte Pair Encoding (BPE) - Example

Dataset: "she sells seashells by the seashore"

Step 2 vocabulary:

- {'s': 0, 'h': 1, 'e': 2, ' ': 3, 'l': 4, 'a': 5, 'b': 6, 'y': 7, 't': 8, 'o': 9, 'r': 10, 'sh': 11, ' s': 12}

Step 3:

- Tokenized text:
    - [11, 2, 12, 2, 4, 4, 0, 12, 2, 5, 11, 2, 4, 4, 0, 3, 6, 7, 3, 8, 1, 2, 12, 2, 5, 11, 9, 10, 2]

Byte Pair Encoding (BPE) - Example

Dataset: "she sells seashells by the seashore"

Step 2 vocabulary:

- {'s': 0, 'h': 1, 'e': 2, ' ': 3, 'l': 4, 'a': 5, 'b': 6, 'y': 7, 't': 8, 'o': 9, 'r': 10, 'sh': 11, ' s': 12}

Step 3:

- Tokenized text:
    - [11, 2, 12, 2, 4, 4, 0, 12, 2, 5, 11, 2, 4, 4, 0, 3, 6, 7, 3, 8, 1, 2, 12, 2, 5, 11, 9, 10, 2]
- Frequencies:
    - {(12, 2): 3, (11, 2): 2, (2, 12): 2, (2, 4): 2, (4, 4): 2, (4, 0): 2, (2, 5): 2, (5, 11): 2, (0, 12): 1, (0, 3): 1, (3, 6): 1, (6, 7): 1, (7, 3): 1, (3, 8): 1, (8, 1): 1, (1, 2): 1, (11, 9): 1, (9, 10): 1, (10, 2): 1}

Byte Pair Encoding (BPE) - Example

Dataset: "she sells seashells by the seashore"

Step 2 vocabulary:

- {'s': 0, 'h': 1, 'e': 2, ' ': 3, 'l': 4, 'a': 5, 'b': 6, 'y': 7, 't': 8, 'o': 9, 'r': 10, 'sh': 11, ' s': 12}

Step 3:

- Tokenized text:
    - [11, 2, 12, 2, 4, 4, 0, 12, 2, 5, 11, 2, 4, 4, 0, 3, 6, 7, 3, 8, 1, 2, 12, 2, 5, 11, 9, 10, 2]
- Frequencies:
    - {(12, 2): 3, (11, 2): 2, (2, 12): 2, (2, 4): 2, (4, 4): 2, (4, 0): 2, (2, 5): 2, (5, 11): 2, (0, 12): 1, (0, 3): 1, (3, 6): 1, (6, 7): 1, (7, 3): 1, (3, 8): 1, (8, 1): 1, (1, 2): 1, (11, 9): 1, (9, 10): 1, (10, 2): 1}
- Merge tokens: new vocabulary
    - {'s': 0, 'h': 1, 'e': 2, ' ': 3, 'l': 4, 'a': 5, 'b': 6, 'y': 7, 't': 8, 'o': 9, 'r': 10, 'sh': 11, ' s': 12, ' se': 13}

Byte Pair Encoding (BPE) - Example

Dataset: "she sells seashells by the seashore"

Step 0:

- {'s': 0, 'h': 1, 'e': 2, ' ': 3, 'l': 4, 'a': 5, 'b': 6, 'y': 7, 't': 8, 'o': 9, 'r': 10}
- [0, 1, 2, 3, 0, 2, 4, 4, 0, 3, 0, 2, 5, 0, 1, 2, 4, 4, 0, 3, 6, 7, 3, 8, 1, 2, 3, 0, 2, 5, 0, 1, 9, 10, 2]

Byte Pair Encoding (BPE) - Example

Dataset: "she sells seashells by the seashore"

Step 0:

- {'s': 0, 'h': 1, 'e': 2, ' ': 3, 'l': 4, 'a': 5, 'b': 6, 'y': 7, 't': 8, 'o': 9, 'r': 10}
- [0, 1, 2, 3, 0, 2, 4, 4, 0, 3, 0, 2, 5, 0, 1, 2, 4, 4, 0, 3, 6, 7, 3, 8, 1, 2, 3, 0, 2, 5, 0, 1, 9, 10, 2]

Step 1:

- {'s': 0, 'h': 1, 'e': 2, ' ': 3, 'l': 4, 'a': 5, 'b': 6, 'y': 7, 't': 8, 'o': 9, 'r': 10, 'sh': 11}
- [11, 2, 3, 0, 2, 4, 4, 0, 3, 0, 2, 5, 11, 2, 4, 4, 0, 3, 6, 7, 3, 8, 1, 2, 3, 0, 2, 5, 11, 9, 10, 2]

Byte Pair Encoding (BPE) - Example

Dataset: "she sells seashells by the seashore"

Step 0:

- {'s': 0, 'h': 1, 'e': 2, ' ': 3, 'l': 4, 'a': 5, 'b': 6, 'y': 7, 't': 8, 'o': 9, 'r': 10}
- [0, 1, 2, 3, 0, 2, 4, 4, 0, 3, 0, 2, 5, 0, 1, 2, 4, 4, 0, 3, 6, 7, 3, 8, 1, 2, 3, 0, 2, 5, 0, 1, 9, 10, 2]

Step 1:

- {'s': 0, 'h': 1, 'e': 2, ' ': 3, 'l': 4, 'a': 5, 'b': 6, 'y': 7, 't': 8, 'o': 9, 'r': 10, 'sh': 11}
- [11, 2, 3, 0, 2, 4, 4, 0, 3, 0, 2, 5, 11, 2, 4, 4, 0, 3, 6, 7, 3, 8, 1, 2, 3, 0, 2, 5, 11, 9, 10, 2]

Step 2:

- {'s': 0, 'h': 1, 'e': 2, ' ': 3, 'l': 4, 'a': 5, 'b': 6, 'y': 7, 't': 8, 'o': 9, 'r': 10, 'sh': 11, ' s': 12}
- [11, 2, 12, 2, 4, 4, 0, 12, 2, 5, 11, 2, 4, 4, 0, 3, 6, 7, 3, 8, 1, 2, 12, 2, 5, 11, 9, 10, 2]

Byte Pair Encoding (BPE) - Example

Dataset: "she sells seashells by the seashore"

Step 0:

- {'s': 0, 'h': 1, 'e': 2, ' ': 3, 'l': 4, 'a': 5, 'b': 6, 'y': 7, 't': 8, 'o': 9, 'r': 10}
- [0, 1, 2, 3, 0, 2, 4, 4, 0, 3, 0, 2, 5, 0, 1, 2, 4, 4, 0, 3, 6, 7, 3, 8, 1, 2, 3, 0, 2, 5, 0, 1, 9, 10, 2]

Step 1:

- {'s': 0, 'h': 1, 'e': 2, ' ': 3, 'l': 4, 'a': 5, 'b': 6, 'y': 7, 't': 8, 'o': 9, 'r': 10, 'sh': 11}
- [11, 2, 3, 0, 2, 4, 4, 0, 3, 0, 2, 5, 11, 2, 4, 4, 0, 3, 6, 7, 3, 8, 1, 2, 3, 0, 2, 5, 11, 9, 10, 2]

Step 2:

- {'s': 0, 'h': 1, 'e': 2, ' ': 3, 'l': 4, 'a': 5, 'b': 6, 'y': 7, 't': 8, 'o': 9, 'r': 10, 'sh': 11, ' s': 12}
- [11, 2, 12, 2, 4, 4, 0, 12, 2, 5, 11, 2, 4, 4, 0, 3, 6, 7, 3, 8, 1, 2, 12, 2, 5, 11, 9, 10, 2]

Step 3:

- {'s': 0, 'h': 1, 'e': 2, ' ': 3, 'l': 4, 'a': 5, 'b': 6, 'y': 7, 't': 8, 'o': 9, 'r': 10, 'sh': 11, ' s': 12, ' se': 13}
- [11, 2, 13, 4, 4, 0, 13, 5, 11, 2, 4, 4, 0, 3, 6, 7, 3, 8, 1, 2, 13, 5, 11, 9, 10, 2]

Byte Pair Encoding (BPE) - Example

Dataset: "she sells seashells by the seashore"

Step 0:

- {'s': 0, 'h': 1, 'e': 2, ' ': 3, 'l': 4, 'a': 5, 'b': 6, 'y': 7, 't': 8, 'o': 9, 'r': 10}
- [0, 1, 2, 3, 0, 2, 4, 4, 0, 3, 0, 2, 5, 0, 1, 2, 4, 4, 0, 3, 6, 7, 3, 8, 1, 2, 3, 0, 2, 5, 0, 1, 9, 10, 2]

Step 1:

- {'s': 0, 'h': 1, 'e': 2, ' ': 3, 'l': 4, 'a': 5, 'b': 6, 'y': 7, 't': 8, 'o': 9, 'r': 10, 'sh': 11}
- [11, 2, 3, 0, 2, 4, 4, 0, 3, 0, 2, 5, 11, 2, 4, 4, 0, 3, 6, 7, 3, 8, 1, 2, 3, 0, 2, 5, 11, 9, 10, 2]

Step 2:

- {'s': 0, 'h': 1, 'e': 2, ' ': 3, 'l': 4, 'a': 5, 'b': 6, 'y': 7, 't': 8, 'o': 9, 'r': 10, 'sh': 11, ' s': 12}
- [11, 2, 12, 2, 4, 4, 0, 12, 2, 5, 11, 2, 4, 4, 0, 3, 6, 7, 3, 8, 1, 2, 12, 2, 5, 11, 9, 10, 2]

Step 3:

- {'s': 0, 'h': 1, 'e': 2, ' ': 3, 'l': 4, 'a': 5, 'b': 6, 'y': 7, 't': 8, 'o': 9, 'r': 10, 'sh': 11, ' s': 12, ' se': 13}
- [11, 2, 13, 4, 4, 0, 13, 5, 11, 2, 4, 4, 0, 3, 6, 7, 3, 8, 1, 2, 13, 5, 11, 9, 10, 2]

Step 4:

- {'s': 0, 'h': 1, 'e': 2, ' ': 3, 'l': 4, 'a': 5, 'b': 6, 'y': 7, 't': 8, 'o': 9, 'r': 10, 'sh': 11, ' s': 12, ' se': 13, 'she': 14}
- [14, 13, 4, 4, 0, 13, 5, 14, 4, 4, 0, 3, 6, 7, 3, 8, 1, 2, 13, 5, 11, 9, 10, 2]

Byte Pair Encoding (BPE) - Example

Dataset: "she sells seashells by the seashore"

Step 0:

- {'s': 0, 'h': 1, 'e': 2, ' ': 3, 'l': 4, 'a': 5, 'b': 6, 'y': 7, 't': 8, 'o': 9, 'r': 10}
- [0, 1, 2, 3, 0, 2, 4, 4, 0, 3, 0, 2, 5, 0, 1, 2, 4, 4, 0, 3, 6, 7, 3, 8, 1, 2, 3, 0, 2, 5, 0, 1, 9, 10, 2]

Step 1:

- {'s': 0, 'h': 1, 'e': 2, ' ': 3, 'l': 4, 'a': 5, 'b': 6, 'y': 7, 't': 8, 'o': 9, 'r': 10, 'sh': 11}
- [11, 2, 3, 0, 2, 4, 4, 0, 3, 0, 2, 5, 11, 2, 4, 4, 0, 3, 6, 7, 3, 8, 1, 2, 3, 0, 2, 5, 11, 9, 10, 2]

Step 2:

- {'s': 0, 'h': 1, 'e': 2, ' ': 3, 'l': 4, 'a': 5, 'b': 6, 'y': 7, 't': 8, 'o': 9, 'r': 10, 'sh': 11, ' s': 12}
- [11, 2, 12, 2, 4, 4, 0, 12, 2, 5, 11, 2, 4, 4, 0, 3, 6, 7, 3, 8, 1, 2, 12, 2, 5, 11, 9, 10, 2]

Step 3:

- {'s': 0, 'h': 1, 'e': 2, ' ': 3, 'l': 4, 'a': 5, 'b': 6, 'y': 7, 't': 8, 'o': 9, 'r': 10, 'sh': 11, ' s': 12, ' se': 13}
- [11, 2, 13, 4, 4, 0, 13, 5, 11, 2, 4, 4, 0, 3, 6, 7, 3, 8, 1, 2, 13, 5, 11, 9, 10, 2]

Step 4:

- {'s': 0, 'h': 1, 'e': 2, ' ': 3, 'l': 4, 'a': 5, 'b': 6, 'y': 7, 't': 8, 'o': 9, 'r': 10, 'sh': 11, ' s': 12, ' se': 13, 'she': 14}
- [14, 13, 4, 4, 0, 13, 5, 14, 4, 4, 0, 3, 6, 7, 3, 8, 1, 2, 13, 5, 11, 9, 10, 2]

Step 5:

- {'s': 0, 'h': 1, 'e': 2, ' ': 3, 'l': 4, 'a': 5, 'b': 6, 'y': 7, 't': 8, 'o': 9, 'r': 10, 'sh': 11, ' s': 12, ' se': 13, 'she': 14, 'll': 15}
- [14, 13, 15, 0, 13, 5, 14, 15, 0, 3, 6, 7, 3, 8, 1, 2, 13, 5, 11, 9, 10, 2]

Byte Pair Encoding (BPE) - Example

Dataset: "she sells seashells by the seashore"

Step 0:

- {'s': 0, 'h': 1, 'e': 2, ' ': 3, 'l': 4, 'a': 5, 'b': 6, 'y': 7, 't': 8, 'o': 9, 'r': 10}
- [0, 1, 2, 3, 0, 2, 4, 4, 0, 3, 0, 2, 5, 0, 1, 2, 4, 4, 0, 3, 6, 7, 3, 8, 1, 2, 3, 0, 2, 5, 0, 1, 9, 10, 2]

Step 1:

- {'s': 0, 'h': 1, 'e': 2, ' ': 3, 'l': 4, 'a': 5, 'b': 6, 'y': 7, 't': 8, 'o': 9, 'r': 10, 'sh': 11}
- [11, 2, 3, 0, 2, 4, 4, 0, 3, 0, 2, 5, 11, 2, 4, 4, 0, 3, 6, 7, 3, 8, 1, 2, 3, 0, 2, 5, 11, 9, 10, 2]

Step 2:

- {'s': 0, 'h': 1, 'e': 2, ' ': 3, 'l': 4, 'a': 5, 'b': 6, 'y': 7, 't': 8, 'o': 9, 'r': 10, 'sh': 11, ' s': 12}
- [11, 2, 12, 2, 4, 4, 0, 12, 2, 5, 11, 2, 4, 4, 0, 3, 6, 7, 3, 8, 1, 2, 12, 2, 5, 11, 9, 10, 2]

Step 3:

- {'s': 0, 'h': 1, 'e': 2, ' ': 3, 'l': 4, 'a': 5, 'b': 6, 'y': 7, 't': 8, 'o': 9, 'r': 10, 'sh': 11, ' s': 12, ' se': 13}
- [11, 2, 13, 4, 4, 0, 13, 5, 11, 2, 4, 4, 0, 3, 6, 7, 3, 8, 1, 2, 13, 5, 11, 9, 10, 2]

Step 4:

- {'s': 0, 'h': 1, 'e': 2, ' ': 3, 'l': 4, 'a': 5, 'b': 6, 'y': 7, 't': 8, 'o': 9, 'r': 10, 'sh': 11, ' s': 12, ' se': 13, 'she': 14}
- [14, 13, 4, 4, 0, 13, 5, 14, 4, 4, 0, 3, 6, 7, 3, 8, 1, 2, 13, 5, 11, 9, 10, 2]

Step 5:

- {'s': 0, 'h': 1, 'e': 2, ' ': 3, 'l': 4, 'a': 5, 'b': 6, 'y': 7, 't': 8, 'o': 9, 'r': 10, 'sh': 11, ' s': 12, ' se': 13, 'she': 14, 'll': 15}
- [14, 13, 15, 0, 13, 5, 14, 15, 0, 3, 6, 7, 3, 8, 1, 2, 13, 5, 11, 9, 10, 2]

Step 6:

- {'s': 0, 'h': 1, 'e': 2, ' ': 3, 'l': 4, 'a': 5, 'b': 6, 'y': 7, 't': 8, 'o': 9, 'r': 10, 'sh': 11, ' s': 12, ' se': 13, 'she': 14, 'll': 15, 'lls': 16}
- [14, 13, 16, 13, 5, 14, 16, 3, 6, 7, 3, 8, 1, 2, 13, 5, 11, 9, 10, 2]

Byte Pair Encoding (BPE) - Example

Dataset: "she sells seashells by the seashore"

Step 0:

- {'s': 0, 'h': 1, 'e': 2, ' ': 3, 'l': 4, 'a': 5, 'b': 6, 'y': 7, 't': 8, 'o': 9, 'r': 10}
- [0, 1, 2, 3, 0, 2, 4, 4, 0, 3, 0, 2, 5, 0, 1, 2, 4, 4, 0, 3, 6, 7, 3, 8, 1, 2, 3, 0, 2, 5, 0, 1, 9, 10, 2]

Step 1:

- {'s': 0, 'h': 1, 'e': 2, ' ': 3, 'l': 4, 'a': 5, 'b': 6, 'y': 7, 't': 8, 'o': 9, 'r': 10, 'sh': 11}
- [11, 2, 3, 0, 2, 4, 4, 0, 3, 0, 2, 5, 11, 2, 4, 4, 0, 3, 6, 7, 3, 8, 1, 2, 3, 0, 2, 5, 11, 9, 10, 2]

Step 2:

- {'s': 0, 'h': 1, 'e': 2, ' ': 3, 'l': 4, 'a': 5, 'b': 6, 'y': 7, 't': 8, 'o': 9, 'r': 10, 'sh': 11, ' s': 12}
- [11, 2, 12, 2, 4, 4, 0, 12, 2, 5, 11, 2, 4, 4, 0, 3, 6, 7, 3, 8, 1, 2, 12, 2, 5, 11, 9, 10, 2]

Step 3:

- {'s': 0, 'h': 1, 'e': 2, ' ': 3, 'l': 4, 'a': 5, 'b': 6, 'y': 7, 't': 8, 'o': 9, 'r': 10, 'sh': 11, ' s': 12, ' se': 13}
- [11, 2, 13, 4, 4, 0, 13, 5, 11, 2, 4, 4, 0, 3, 6, 7, 3, 8, 1, 2, 13, 5, 11, 9, 10, 2]

Step 4:

- {'s': 0, 'h': 1, 'e': 2, ' ': 3, 'l': 4, 'a': 5, 'b': 6, 'y': 7, 't': 8, 'o': 9, 'r': 10, 'sh': 11, ' s': 12, ' se': 13, 'she': 14}
- [14, 13, 4, 4, 0, 13, 5, 14, 4, 4, 0, 3, 6, 7, 3, 8, 1, 2, 13, 5, 11, 9, 10, 2]

Step 5:

- {'s': 0, 'h': 1, 'e': 2, ' ': 3, 'l': 4, 'a': 5, 'b': 6, 'y': 7, 't': 8, 'o': 9, 'r': 10, 'sh': 11, ' s': 12, ' se': 13, 'she': 14, 'll': 15}
- [14, 13, 15, 0, 13, 5, 14, 15, 0, 3, 6, 7, 3, 8, 1, 2, 13, 5, 11, 9, 10, 2]

Step 6:

- {'s': 0, 'h': 1, 'e': 2, ' ': 3, 'l': 4, 'a': 5, 'b': 6, 'y': 7, 't': 8, 'o': 9, 'r': 10, 'sh': 11, ' s': 12, ' se': 13, 'she': 14, 'll': 15, 'lls': 16}
- [14, 13, 16, 13, 5, 14, 16, 3, 6, 7, 3, 8, 1, 2, 13, 5, 11, 9, 10, 2]

Step 7:

- {'s': 0, 'h': 1, 'e': 2, ' ': 3, 'l': 4, 'a': 5, 'b': 6, 'y': 7, 't': 8, 'o': 9, 'r': 10, 'sh': 11, ' s': 12, ' se': 13, 'she': 14, 'll': 15, 'lls': 16, ' sea': 17}
- [14, 13, 16, 17, 14, 16, 3, 6, 7, 3, 8, 1, 2, 17, 11, 9, 10, 2]

Tokenization

Tokenization methods:

- **Byte Pair Encoding (BPE)**
- **WordPiece**:
    - a subword-tokenization method
    - incorporating special word boundary symbols
    - employing a data-driven approach to develop the vocabulary
        - training a language model
        - adding new tokens based on the the model's probability
    - **Core concept**:
        - in a bottom-up fashion, merging token pairs with high model probability and adding as a new token
    - **Process of WordPiece**:
        1. Initialize the vocabulary with the characters as individual tokens occurring in the dataset
        2. Merging symbol pairs whose inclusion in the vocabulary maximizes the likelihood of the training data
        3. Repeat step 2. until the vocabulary reaches a specified size
    - ensuring that the merge contributes positively to the model's performance
- **Unigram**
- **SentencePiece**

Tokenization

Tokenization methods:

- **Byte Pair Encoding (BPE)**
- **WordPiece**
- **Unigram**:
    - a subword tokenization method
    - **Core concept**:
        - in a top-down fashion, progressively removing tokens tokens based on how the loss varies for a Unigram Language Model
    - **Process of Unigram**:
        1. Initialize the vocabulary with a large base vocabulary that might include all pre-tokenized words along with the most common substrings
        2. evaluating a loss function (often: log-likelihood) over the training data given the current vocabulary and a unigram language model
        3. symbols that result in the smallest increase in loss (least affecting the model's performance) are pruned
        4. Repeat step 2. and 3. until the vocabulary reaches a specified size
    - unlike BPE and WordPiece, Unigram is not constrained by merge rules
        - allowing for multiple potential tokenizations of new text after training
        - Unigram maintains the probability of each token occurring in the training corpus, which helps in determining the most probable tokenization
    - calculating the potential increase in overall loss for each symbol if it were to be removed from the vocabulary
    - not directly used in models
        - integral part of the SentencePiece library
- **SentencePiece**

Tokenization

Tokenization methods:

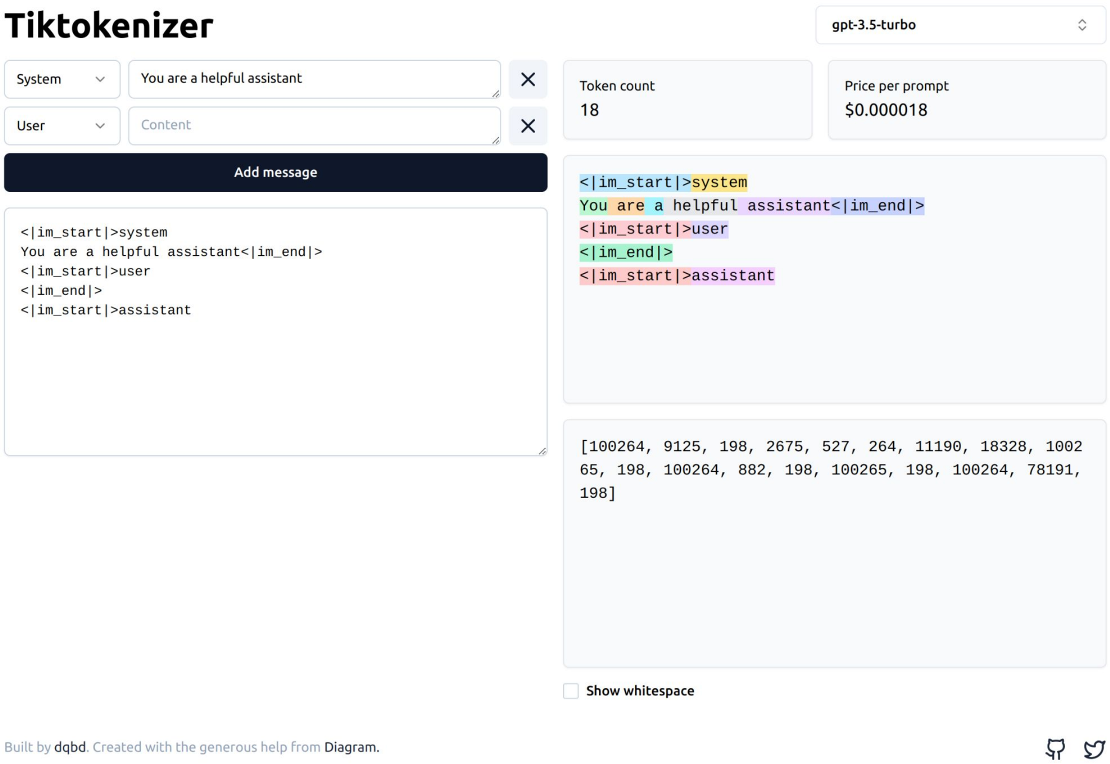
- **Byte Pair Encoding (BPE)**
- **WordPiece**
- **Unigram**
- **SentencePiece**:
    - a subword tokenization method
    - language-independent
        - applying a language-agnostic approach to tokenization
            - no assumption that input text relies on spaces to delineate words
        - good for training multilanguage models
    - using either BPE or Unigram
    - Architecture with 4 components:
        - Normalizer:
            - preparing the text for tokenization by standardizing it
            - removing possible variations that do not affect the meaning
        - Trainer:
            - applying BPE or Unigram to learn the optimal subword vocabulary from the raw input stream
        - Encoder:
            - converting input text into a sequence of tokens using the generated subword vocabulary
        - Decoder:
            - reconstructing the original text from the sequence of tokens
            - ensuring that the tokenization process is reversible

Tokenization

Tokenization methods of LLMs:

BPE	WordPiece	SentencePiece
BART	BERT	ALBERT
CLIP	BLIP	Chinchilla
DALL-E	Flamingo	LaMDA
DeBERTa	Layout-LM	Layout-XLM
GPT, GPT-2, GPT-3, GPT-4	StructBERT	LLama, LLama-2
InstructGPT	UniLM	PaLM
RoBERTa		T5
		XLM-RoBERTa

Tokenization



Embeddings

What are embeddings?

- Words / Tokens are mapped into a Vector Space
- Clustering in the Vector Space
    - semantic similarity - distance
- Vector arithmetic

Embeddings

Classical methods: Features / Descriptors

- **One-hot encoding**
- **Bag-of-Words (BoW)**
- **Term Frequency - Inverse Document Frequency (TF-IDF)**

Embeddings

Classical methods: Features / Descriptors

- **One-hot encoding**:
    - transforming categorical data, such as tokens, into numerical form
    - each word / token in the vocabulary is assigned a unique ID
    - each one is represented as a sparse vector
    - Example:
        - first token: [1, 0, 0, …]
        - second token: [0, 1, 0, …]
        - last token: [0, 0, 0, …, 1]
    - Advantages:
        - Simplicity
    - Disadvantages:
        - Sparsity
        - Lack of semantic information
        - Scalability issues
- **Bag-of-Words (BoW)**
- **Term Frequency - Inverse Document Frequency (TF-IDF)**

Embeddings

Classical methods: Features / Descriptors

- **One-hot encoding**
- **Bag-of-Words (BoW):**
  - BoW represents text as a collection of the counts of words that appear in the document
    - words order is omitted
    - words cardinality is kept
  - Vector representation
  - each unique word in the entire dataset becomes a feature
    - another version: words of interest become features
  - Advantages:
    - Simplicity and efficiency
    - Scalability
  - Disadvantages:
    - Loss of context
    - Sparsity
    - High dimensionality
    - Fixed vocabulary
- **Term Frequency - Inverse Document Frequency (TF-IDF)**

Embeddings

Classical methods: Features / Descriptors

- **One-hot encoding**
- **Bag-of-Words (BoW)**
- **Term Frequency - Inverse Document Frequency (TF-IDF):**
    - improving BoW by considering not just the frequency of words within a single document, but also how unique these words are across all documents in the corpus
    - Vector representation
    - Term Frequency (TF):
        - measure how frequently a term occurs in a document
        - words that appear in many documents have higher TF score
    - Inverse Document Frequency (IDF):
        - measures how important a term is within the corpus
        - words that appear in many documents have lower IDF score
    - TF-IDF score:
        - the product of TF and IDF
    - Advantages:
        - Relevance; Dimensionality Reduction
    - Disadvantages:
        - Complexity; Context Ignorance; Fixed Vocabulary

Embeddings

Deep Learning-based methods: Embeddings

- **Word2Vec**
- **GloVe**
- **CoVe**
- **ELMo**
- **BERT**

Embeddings

Deep Learning-based methods: Embeddings

- **Word2Vec**: [2013 - Google]
    - Neural Network based model produces learned distributed vector representations
        - shallow NN: 2-layer NN
            - Continuous Bag of Words (CBOW):
                - Objective: predicting the target words based on the context surrounding it
            - Skip-Gram:
                - Objective: predicting surrounding context words for a given target word
    - Distributed representations (inherent structure)
    - capturing syntactic and semantic word relationships by **vector arithmetic**
        - first method to achieve this
- **GloVe**
- **CoVe**
- **ELMo**
- **BERT**

Embeddings

Deep Learning-based methods: Embeddings

- **Word2Vec**
- **GloVe**: [2014 - Stanford]
  - GloVE: Global Vectors for Word Representations
  - matrix statistics-based models
    - gradient-based learning to better fit co-occurrences
  - unlike Word2Vec (which uses local context information), Glove constructs an explicit word-context matrix of statistics across the whole text corpus
    - global information is used
  - Matrix Factorization is applied:
    - Global Matrix Factorization
    - Local Context Window-based method
  - utilizing global information (context)
- **CoVe**
- **ELMo**
- **BERT**

Embeddings

Deep Learning-based methods: Embeddings

- **Word2Vec**
- **GloVe**
- **CoVe**: [2018]
  - CoVe: Contextualized word Vectors
  - Deep Neural Network-based models used
    - Machine Translation model
      - Seq2Seq model: LSTM, GRU
  - providing word representations that are sensitive to the context in which a words appears
    - Polysemy
  - dynamic representation vs. static representations
- **ELMo**
- **BERT**

Embeddings

Deep Learning-based methods: Embeddings

- **Word2Vec**
- **GloVe**
- **CoVe**
- **ELMo**: [2018]
  - ELMo: Embeddings from Language Models
  - Deep Neural Networks:
    - stacked bidirectional LSTM
    - Objective: Language Modeling
  - contextualized representations
  - shallow bidirectional representations
    - concatenation of unidirectional representations
    - capturing complex characteristics
- **BERT**

Embeddings

Deep Learning-based methods: Embeddings

- **Word2Vec**
- **GloVe**
- **CoVe**
- **ELMo**
- **BERT**: [2018 -Google]
    - BERT: Bidirectional Encoder Representations from Transformers
    - Deep Neural Network model
        - Transformer architecture
    - contextualized representations
    - deeply bidirectional representations

Embeddings

Distributed representations

Adding contextual information (static)

Using Deep Neural Networks (DNNs) - Representation Learning

Shallow Bidirectional Representations

Deep Bidirectional Representations

Text Embeddings

A vector representation for an arbitrary text:

- a sentence
- a document
- etc.

How to create?

Applications of Text Embeddings:

- Semantic Search
- Classification
- Recommendations
- Question Answering
- Retrieval Augmented Generation (RAG)

Additional Resources

- Videos:
  - Google: Introduction to Large Language Models (16 min)
  - Andrej Karpathy: Intro to Large Language Models (60 min)
  - Google: Introduction to Generative AI (22 min)
  - Harvard University: Large Language Models and The End of Programming (67 min)
  - Andrej Karpathy: Let's build the GPT Tokenizer (134 min)
- Papers:
  - A survey of Large Language Models (2023)

Homework

Byte-Pair Encoding (BPE) implementation:

- Implement the BPE algorithm in Python
- Process of BPE:
    1. Initialize the vocabulary with the characters as individual tokens occurring in the dataset
    2. Calculating the frequency of each adjacent character (or byte) within the dataset
    3. The most frequent adjacent pair is merged into a new token (merged tokens are retained)
    4. Repeat step 2. and 3. until the vocabulary reaches a specified size
- Details:
    - there is a *bpe* function that takes 2 inputs (text, max_vocabulary_size) and 2 outputs (vocabulary, tokenized_text):
        - text: a string, the text we want to use to build the vocabulary, then tokenize
        - max_vocabulary_size: an integer, which defines the size of the final vocabulary
        - vocabulary: dictionary, where keys are the tokens (str) while values are the code point (int)
        - tokenized_text: list, where entries are the code points from vocabulary

Homework

Clarifications:

- When we have more tokens during tokenization which should be used?
  - we opt for the longest token
  - we tokenized from left to right
  - Example:
    - we want to tokenize the ‘apple’ text
    - we have the tokens: ‘a’, ‘p’, ‘l’, ‘e’, ‘ap’, ‘app’
    - **tokenization: ‘app’, ‘l’, ‘e’**
- If we have more token pairs with max frequency, which one to merge?
  - that one which occurs first in the sequence (from left to right)
  - Example:
    - text: ‘aaabbb’
    - tokens: ‘a’, ‘b’
    - tokenized_text: ‘a’, ‘a’, ‘a’, ‘b, ‘b’, ‘b’
    - token pairs frequencies: (‘a’, ‘a’): 2, (‘a’, ‘b’): 1, (‘b’, ‘b’): 2
    - **(‘a’,’a’) token pair is merged to ‘aa’**
- the vocabulary retains the order of insertion

Homework

Example:

- Input:
  - text = “she sells seashells by the seashore”
  - max_vocabulary_size = 15
- Output:
  - vocabulary:
    - {'s': 0, 'h': 1, 'e': 2, ' ': 3, 'l': 4, 'a': 5, 'b': 6, 'y': 7, 't': 8, 'o': 9, 'r': 10, 'sh': 11, ' s': 12, ' se': 13, 'she': 14}
  - tokenized_text:
    - [14, 13, 4, 4, 0, 13, 5, 14, 4, 4, 0, 3, 6, 7, 3, 8, 1, 2, 13, 5, 11, 9, 10, 2]