

# Understanding and Training Language Models: **Text preprocessing**

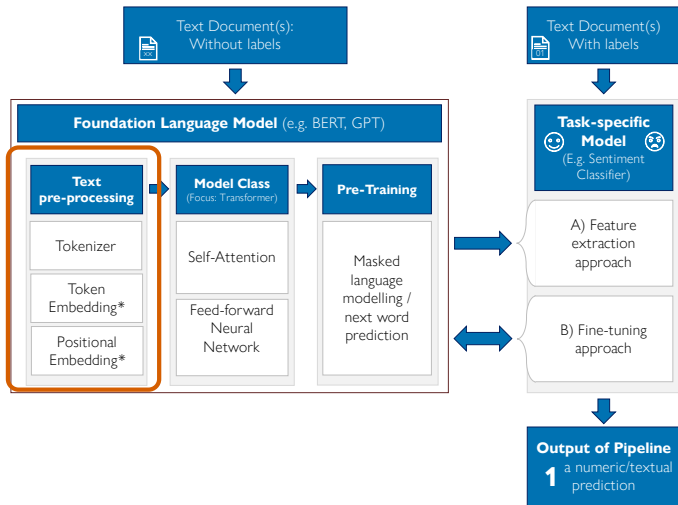
Erik-Jan Senn

Faculty of Mathematics and Statistics, University of St. Gallen

---

CSH Autumn School at University of Hohenheim  
September/October 2024

# Where are we?



Language Modelling Pipeline

# Roadmap

Tokenization

Input Embeddings

# Overview

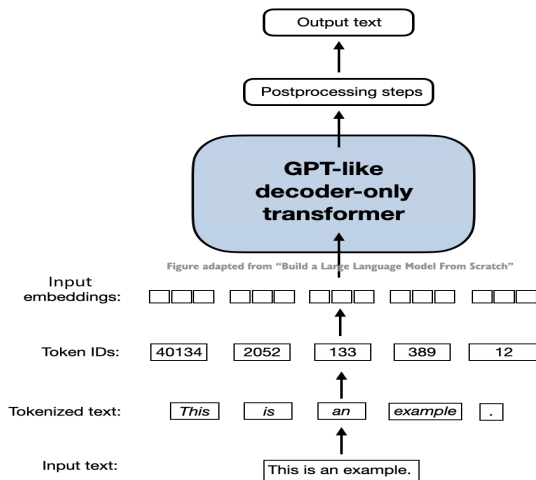
## ► Tokenizer

- Cuts **input text** to smallest components, so called **tokens** (string vector).
- For each **token**, assigns a **token ID** (integer vector) using the vocabulary.

## ► Input embedding: For **token ID**, assign an **input embedding** (float matrix).

- *Embedding means latent (vector) representation.*
- This will boil down to simple matrix multiplication:

$$\mathbf{E}_{l \times d} = \mathbf{E}_{one-hot, l \times v} \cdot \mathbf{E}_{T, v \times d}(\theta_T) + \mathbf{E}_{P, l \times d}(\theta_P)$$



Source

# Text Preprocessing

## **Tokenization**

# Tokenization

We want to represent a (sequence of) text as integers.

1. Split the text in smallest indivisible units called **tokens**.
2. Assign each unique word its own integer: **token ID**

The **vocabulary** is a list of tokens and their corresponding token IDs.

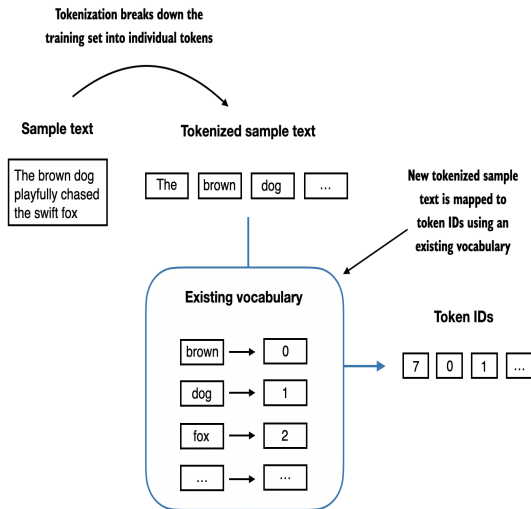


Figure adapted from "Build a Large Language Model From Scratch"

## Building a Tokenizer: How large should a Token be?

There is a **tradeoff**:

	Small tokens	Large tokens
Example	Characters (a, b)	Sentences (I love Apples)
Semantic meaning of T.	Ambiguous	Precise
Interaction effects between T's	Complicated	Simple
Occurrences of T. in real text	Frequent	Infrequent
Number of T's.	Low	High

# Building a Tokenizer: Common Approaches

- ▶ Word-level tokenization
- ▶ Character-level tokenization
- ▶ **Subword tokenization** is the State-of-the-art (SOTA):
  - ▶ General idea:
    1. Start with single characters as tokens:  
a,b, ...
    2. Iteratively merge frequently occurring character combinations to own tokens:  
{t} and {h} merged to {th},  
then {th} and {e} merged to {the}, ...
    3. Stop when a target vocabulary size (number of tokens) is reached.
  - ▶ E.g. *WordPiece* (variants used e.g. in *BERT*), *Byte-Pair-Encoding* (variants used e.g. by *GPT2*, *LLama*)



# Some Notes on Tokens

- ▶ **Special context tokens** are tokens that carry a special meaning.

Examples:

- ▶ A text component that cannot be found/split up in known vocabulary: [UNK] in *BERT*.
  - ▶ Mark separate text documents (e.g. multiple tweets): <endoftext> in *GPT-2*, [CLS] & [SEP] in *BERT*.
  - ▶ Some models require every sequence to have the same length /: e.g. in [PAD] tokens are used in *BERT* to fill shorter sequences up to that length.
- 
- ▶ The token  $##\{text\}$  denotes a continuation of a word in subword tokenization: e.g. working could consist of two tokens, work and ##ing.
- 
- ▶ **Adding new tokens** to a tokenizer:
    - ▶ Useful when your text documents differ from the text used to train the tokenizer.
    - ▶ E.g. add tokens to handle domain-specific vocabulary (e.g. *EBIT* in accounting, *MRI* in medicine) or identify entities in your data (e.g. company names).
    - ▶ New tokens require resizing and training of the new token embeddings and fine-tuning of entire LLM.

# One-Hot-Encoding

- ▶ After tokenization, input text sequences can be represented using **one-hot-encoding**: Each token is assigned a sparse vector with many 0s and a 1 for the corresponding token index in the vocabulary.
- ▶ The **matrix**  $\mathbf{E}_{one-hot} \in \{0, 1\}^{l \times v}$  has  $l$  rows for the tokens in the text input (of maximum size  $l$ ) and  $v$  columns for the vocabulary (of size  $v$ ).

Input (length $l$ )	Tokens	Tokens Token IDs	Vocabulary (size $v$ )									
			you	.	fruit	apples	is	are	healthy	nutritious	...	...
Fruit is healthy.	fruit	2	0	0	1	0	0	0	0	0	...	...
	is	4	0	0	0	0	1	0	0	0	...	...
	healthy	6	0	0	0	0	0	0	1	0	...	...
	.	1	0	1	0	0	0	0	0	0	...	...
Apples are nutritious.	apples	3	0	0	0	1	0	0	0	0	...	...
	are	5	0	0	0	0	0	1	0	0	...	...
	nutritious	7	0	0	0	0	0	0	0	1	...	...
	.	1	0	1	0	0	0	0	0	0	...	...

# Text Preprocessing

## **Input Embeddings**

# Input embeddings

**Input embeddings**  $\mathbf{E} \in \mathbb{R}^{l \times d}$  are **low-dimensional dense representations** of the text before it is passed to the *heart* of the LMs, the transformer.

$$\mathbf{E}_{l \times d} = \mathbf{E}_{one-hot, l \times v} \cdot \mathbf{E}_{T, v \times d}(\theta_T) + \mathbf{E}_{P, l \times d}(\theta_P)$$

- ▶  $\mathbf{E}_{one-hot} \in \{0, 1\}^{l \times v}$  : one-hot-encoded token ids of sequence
- ▶  $\mathbf{E}_T(\theta_T) \in \mathbb{R}^{v \times d}$  : token embedding matrix with learnable parameters  $\theta_T$
- ▶  $\mathbf{E}_P(\theta_P) \in \mathbb{R}^{l \times d}$  : absolute position embedding matrix with learnable parameters  $\theta_P$
- ▶  $l$  : sequence length in tokens (e.g. max 512 in *BERT*)
- ▶  $d$  : output dimension (decision, e.g. 768 in *BERT*)
- ▶  $v$  : vocabulary size (depends on tokenizer, e.g. 30522 in *BERT*)

# Token Embeddings

Motivating example (from before):

1. Fruit is healthy.  $(\mathbf{E}^{(1)})_{one-hot, l \times v}$
2. Apples are nutritious.  $(\mathbf{E}^{(2)})_{one-hot, l \times v}$

**Problem 1:**  $\mathbf{E}_{one-hot, l \times v}$  is **high-dimensional and sparse**: E.g. for  $l = 512$  and  $v = 30522$  as in *BERT*, the matrix  $\mathbf{E}_{one-hot, l \times v}$  has 15,627,264 entries.

**Solution 1:** The **token embedding** matrix  $\mathbf{E}_T(\theta_T) \in \mathbb{R}^{v \times d}$  transforms input text into a **dense continuous low-dimensional** token representation of dimension  $l \times d$  with  $d \ll v$

# Token Embeddings

Motivating example (from before):

1. Fruit is healthy. ( $\mathbf{E}^{(1)}_{one-hot, l \times v}$ )
2. Apples are nutritious. ( $\mathbf{E}^{(2)}_{one-hot, l \times v}$ )

**Problem 2:**  $\mathbf{E}_{one-hot, l \times v}$  ignores semantic similarity: E.g. the tokens apple and apple have similar meanings, but the representations are not similar.

**Solution 2:** The token embedding matrix  $\mathbf{E}_T(\theta_T) \in \mathbb{R}^{v \times d}$  is trained to capture semantic meanings of tokens - similar tokens have similar token embedding vectors (e.g. measured by cosine similarity). Training is conducted jointly with the rest of the LM (see later lectures), or using e.g. word2vec.

# Token Embeddings Computation

Token embeddings for an input text can be computed in two equivalent ways:

- ▶ Matrix multiplication:

$$\mathbf{E}_{one-hot, l \times v} \cdot \mathbf{E}_{T, v \times d}(\theta_T)$$

- ▶ Index Look-up (see figure):

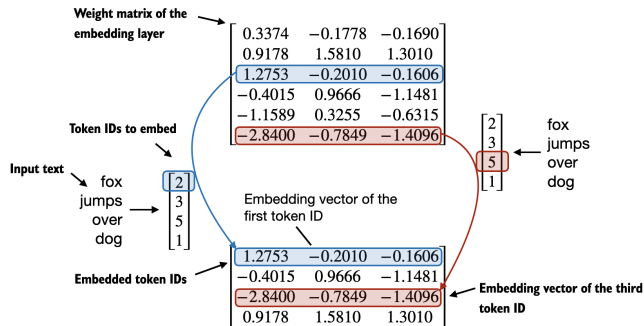


Figure adapted from "Build a Large Language Model From Scratch"

The top matrix is the token embedding matrix of the entire vocabulary, the bottom matrix is the token embedding example text. [Source](#)

# Position embeddings

Motivating example

1. I ate the pizza.
2. The pizza ate me.

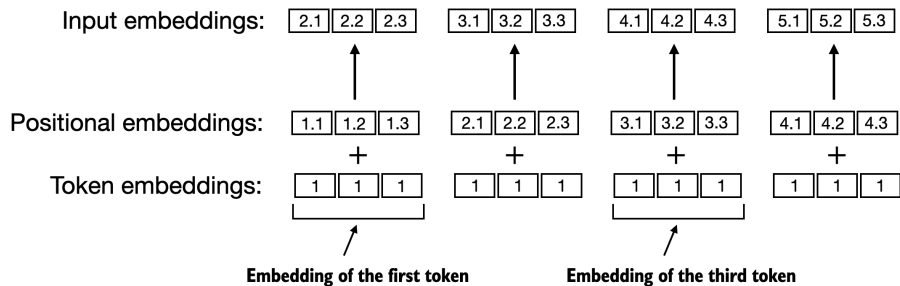
**Problem:** Same tokens, but different sequence changes the entire meaning.  
However, **token embeddings are independent of the token position** in a sequence.

**Solution:** The **positional embedding** matrix  $\mathbf{P}(\theta_P)_{l \times d}$  captures the sequential structure of text by modifying the **embedding differ depending on the token position** in a sequence.

**Result:** Two sentences with the same words but different order have different input embeddings and therefore be differentiated.



# Input and position embeddings (figure)



Source

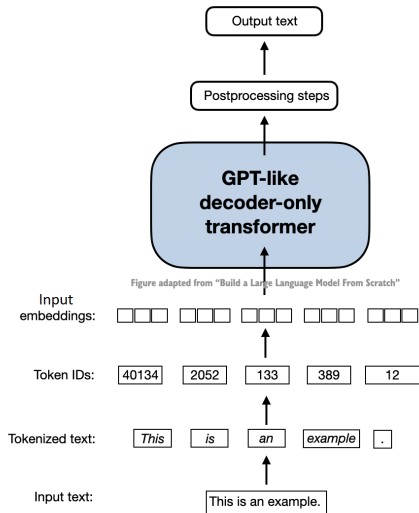
# Input embeddings

**Input embeddings**  $\mathbf{E} \in \mathbb{R}^{l \times d}$  are **low-dimensional dense representations** of the text before it is passed to the *heart* of the LMs, the transformer.

$$\mathbf{E}_{l \times d} = \mathbf{E}_{one-hot, l \times v} \cdot \mathbf{E}_{T, v \times d}(\theta_T) + \mathbf{E}_{P, l \times d}(\theta_P)$$

- ▶  $\mathbf{E}_{one-hot} \in \{0, 1\}^{l \times v}$  : one-hot-encoded token ids of sequence
- ▶  $\mathbf{E}_T(\theta_T) \in \mathbb{R}^{v \times d}$  : token embedding matrix with learnable parameters  $\theta_T$
- ▶  $\mathbf{E}_P(\theta_P) \in \mathbb{R}^{l \times d}$  : absolute position embedding matrix with learnable parameters  $\theta_P$
- ▶  $l$  : sequence length in tokens (e.g. max 512 in *BERT*)
- ▶  $d$  : output dimension (decision, e.g. 768 in *BERT*)
- ▶  $v$  : vocabulary size (depends on tokenizer, e.g. 30522 in *BERT*)

# Recap



Source

# Strawberries are a Problem for LMs



How many 'R's are in the word "Strawberry"?



The word "Strawberry" contains **3** 'R's.

# Strawberries are a Problem for LMs



How many 'R's are in the word "Strawberry"?



The word "Strawberry" contains 3 'R's.

Why could this task be so difficult for *GPT4o*?

# Questions ?

# References