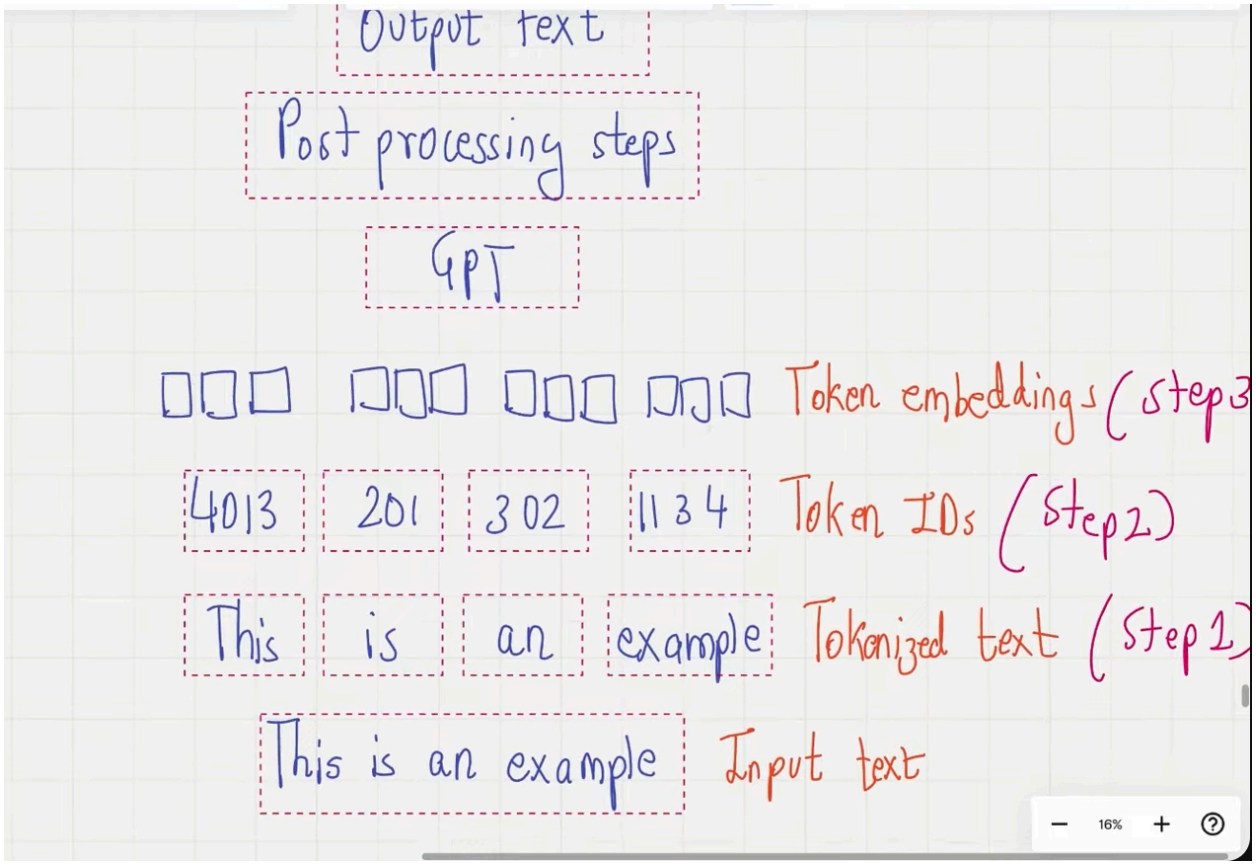
Chapter 6: Tokenization

Tokenization: Process of breaking down sentence to tokens

How to prepare input text for training LLMs (Steps)

1. Tokenizing text: Splitting text into individual word and subword tokens
2. Convert tokens into token IDs
3. Encode token IDs into vector representations



https://github.com/zhzhhyzh/learn-tokenizer

- In 'concept' folder will explain the concept

Chapter 7: Byte Pair Encoding (BPE)

Tokenization algorithms types

1. **Word based.**
    - Each word in the sentence is one token
    - Eg. “My hobby is playing cricket”. Then in here is 5 tokens
    - Adv: Every words is a token
    - Problem 1: What do we do with out of vocabulary (OOV) words, different meanings of similar words [boy, boys]. Eg, when “football” come out, then it will having error
    - Problem 2: English having 170,000 to 200,000. The tokens will be large to include all
2. **Sub-word based**
    - Rule 1: Don’t split frequently used words into smaller subwords.
    - Rule 2: Split the rare words into smaller, meaningful subwords
    - Eg: “boy” should not be split, “boys” should be split into “boy” and “s”.
    - The subword splitting helps the model learn that different words with same root words as “token” like “tokens” and “tokenizing” are similar in meaning
    - It also helps the model learn that “tokenization” and “modernization” are made up of different root words but have the same suffix “ization” and are used in the same syntactic situations.
3. **Character based**
    - Each character in the sentence is one token
    - Eg. “My hobby is playing cricket”. Then in here is 18 tokens
    - Lead to very small vocabulary. Every language has fixed number of characters (English around 256 char) to solve the OOV problem
    - Adv: Can make everything specific, and solve OOV problem
    - Problem 1: The meaning associated with words is completely lost.
    - Problem 2: The tokenized sequence is much longer than the initial raw text.

***Byte Pair Encoding (BPS) is a subword tokenization algorithm**

BPE algorithm is most common pair of consecutive bytes of data is replaced with a byte that does not occur in data

For example, original data: aaabdaaabac
Identify the most common byte pair, the byte pair ‘aa’ occurs the most. We will replace it with Z and Z does not occur in the data
Compressed data: ZabdZabac

(Keep on repeating it sequentially)
The next common byte pair is “ab”, we will replace this with Y
Compressed data: ZYdZYac

Only a byte pair ‘ac’ left. But it appears only once, so we do not encode it.
It can be continue by identify the ZY as pair and replace with W
Compressed data: WdWac

Importance of BPE for LLMs

1. BPE ensures that the most common words in the vocabulary are represented as a single token, while rare words are broken down into two or more subword tokens
2. Practical example:
    a. {“old”: 7, “older”: 3, “finest”: 9, “lowest”: 4}
    b. Preprocessing: We need to add an end token “</w>” at the end of each word. {“old</w>”: 7, “older</w>”: 3, “finest</w>”: 9, “lowest</w>”: 4}
    c. Let us now split words into characters and count their frequency by using frequency table

Number	**Token**	**Frequency**
1	</w>	23
2	o	14
3	l	14
4	d	10
5	e	16
6	r	3
7	f	9
8	i	9
9	n	9
10	s	13
11	t	13
12	w	4

3. The next step in the BPE algorithm is to look for the most frequent pairing.
    - By merging them and performing the same iteration again and again until we reach the token limit or iteration limit.
    a. Iteration 1: Start with second most common token ‘e’, most common token btte pair start with e: “es”

Number	**Token**	**Frequency**
1	</w>	23
2	o	14
3	l	14
4	d	10
5	e	16 - 13 = 3
6	r	3
7	f	9
8	i	9
9	n	9
10	s	13 - 13 = 0
11	t	13
12	w	4
13	es	9 + 4 = 13

b. Iteration 2: Merge the tokens “es” and “t” as they have appeared 13 times in our dataset

Number	**Token**	**Frequency**
1	</w>	23
2	o	14
3	l	14
4	d	10
5	e	16 - 13 = 3
6	r	3
7	f	9
8	i	9
9	n	9
10	s	13 - 13 = 0
11	t	13 - 13 = 0
12	w	4
13	es	9 + 4 = 13 - 13 = 0
14	est	13

c. Iteration 3: Now let us loot at the </w> token, we see that “est</w>” has appeared 13 times.
    - It is to help algorithms understand the difference between estimate and highest.

Number	**Token**	**Frequency**
1	</w>	23 - 13 = 10
2	o	14
3	l	14
4	d	10
5	e	16 - 13 = 3
6	r	3
7	f	9
8	i	9
9	n	9
10	s	13 - 13 = 0
11	t	13 - 13 = 0
12	w	4
13	es	9 + 4 = 13 - 13 = 0
14	est	13 - 13 = 0
15	est</w>	13

d. Iteration 4: “o” and “l” has appeared 10 times

Number	Token	Frequency
1	</w>	23 - 13 = 10
2	o	14
3	l	14
4	d	10
5	e	16 - 13 = 3
6	r	3
7	f	9
8	i	9
9	n	9
10	s	13 - 13 = 0
11	t	13 - 13 = 0
12	w	4
13	es	9 + 4 = 13 - 13 = 0
14	est	13 - 13 = 0
15	est</w>	13
16	ol	7 + 3 = 10

e. Iteration 5: “ol” and “d” has appeared 10 times

Number	Token	Frequency
1	</w>	23 - 13 = 10
2	o	14
3	l	14
4	d	10 - 10 = 0
5	e	16 - 13 = 3
6	r	3
7	f	9
8	i	9
9	n	9
10	s	13 - 13 = 0
11	t	13 - 13 = 0
12	w	4
13	es	9 + 4 = 13 - 13 = 0
14	est	13 - 13 = 0
15	est</w>	13
16	ol	7 + 3 = 10 - 10 = 0
17	old	7 + 3 = 10

f. Iteration 6: “f” ,”i”,”n” appear 9 times. But we just have one word with these characters. Therefore we are not merging them.

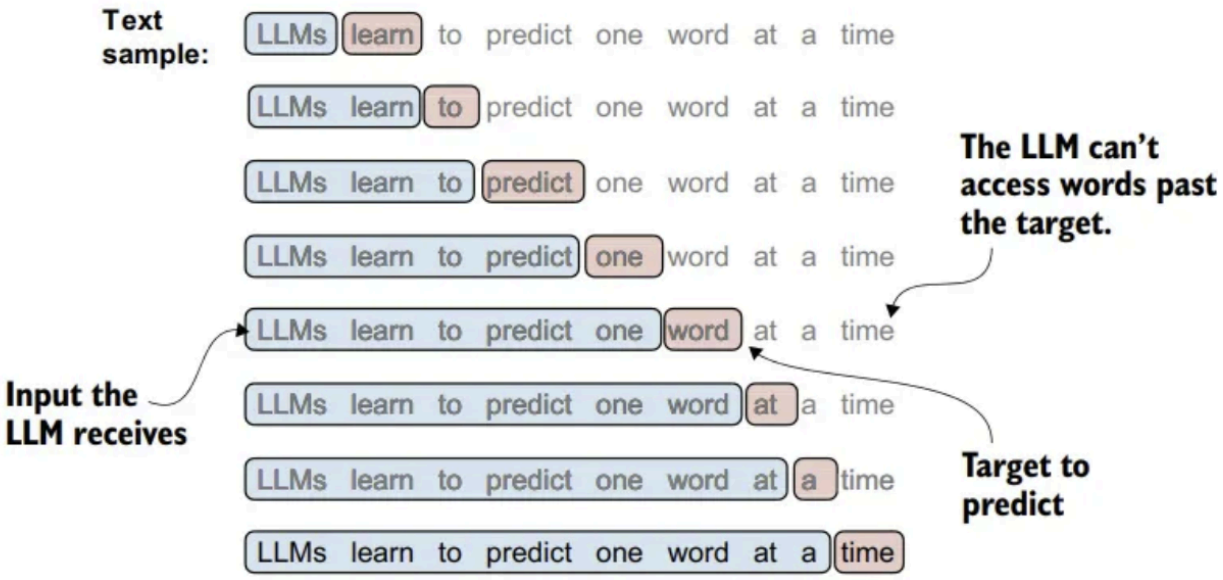
Remove the token with 0 count.
This list of 11 tokens will serve as our vocabulary
The stopping criteria can either be the token count or the number of iterations

Number	**Token**	**Frequency**
1	</w>	10
2	o	14
3	l	14
4	e	3
5	r	3
6	f	9
7	i	9
8	n	9
9	w	4
10	est</w>	13
11	old	10

https://github.com/zhzhhyzh/learn-tokenizer
In ‘bpe_concept’ folder will explain the BPE algorithm how it works. It is still in the STEP 2 which is tokenized text (Step 1) then token IDs generated. The last step is vector embedding (Step 3).

Chapter 8: Create Input-Target Pairs

Prerequisite to create vector embeddings is to create input-target pairs
Input-target pairs:



Here is given the text sample, then extract input blocks as subsamples that serve as input to the LLM. The LLMs prediction task during training is to predict the next word that follows the input block. During training, we mask out all words that are past the target.

Data Loader - It will fetch input-target pairs using sliding windows
To implement efficient data loaders, we collect inputs in a tensor x, where each row of y tensor is the corresponding predicted target (next words), which are created by shifting the input by one position.

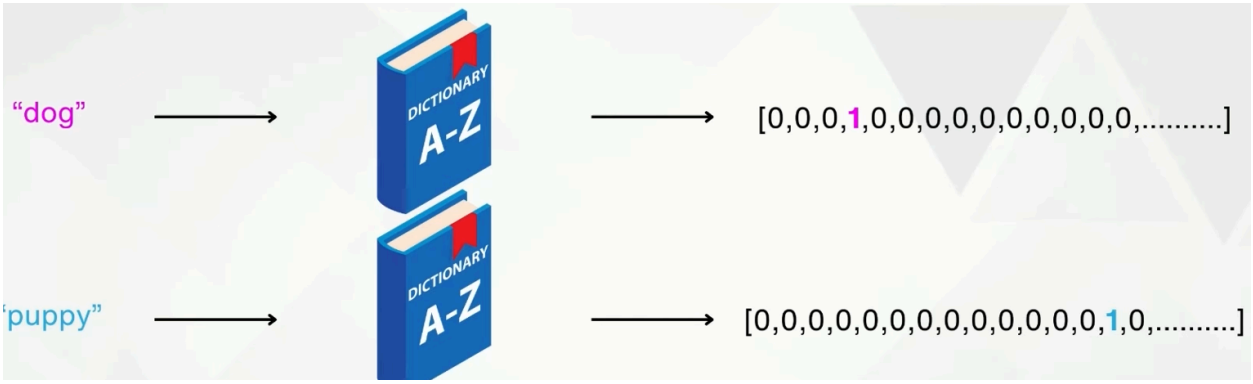
https://github.com/zhzhhyzh/learn-tokenizer
In 'input-target-pairs' folder will explain the input-target pairs, and how it is working.

Chapter 9: Token Embeddings

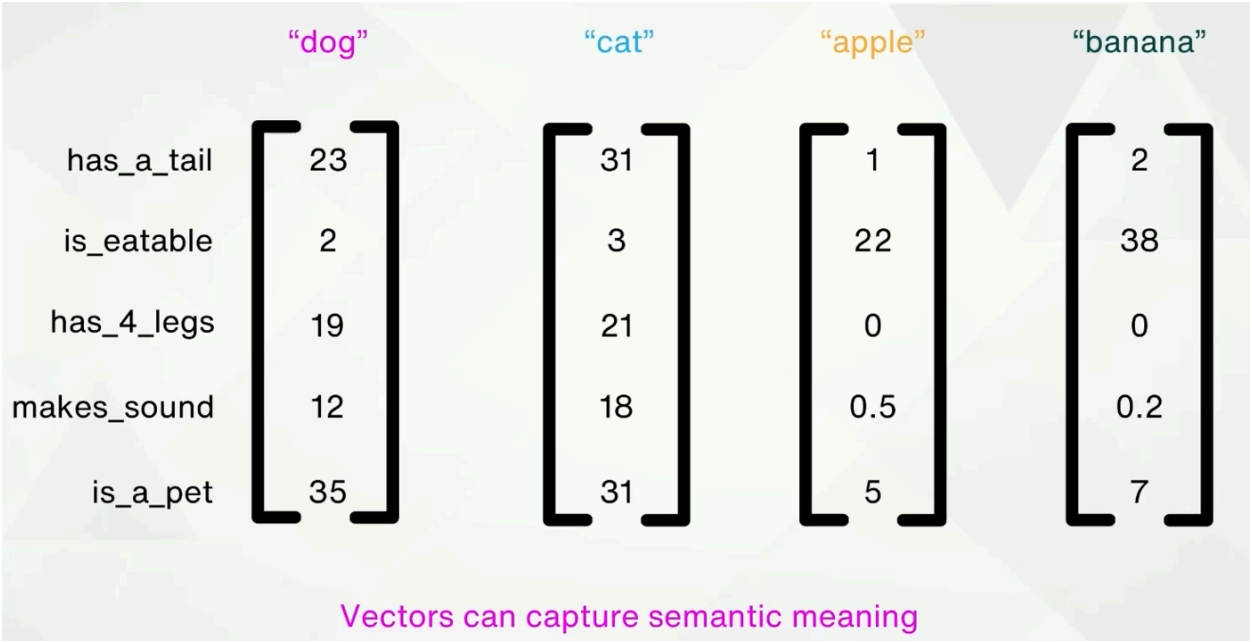
Ways of computer to recognize the word

1. **Assigning random token ID**
    - Computers need to represent the words in numerical value.
    - Problem: Random numbers using. Eg, the ‘cat’ and ‘kitten’ is semantic related, however the associated numbers cannot capture this relation

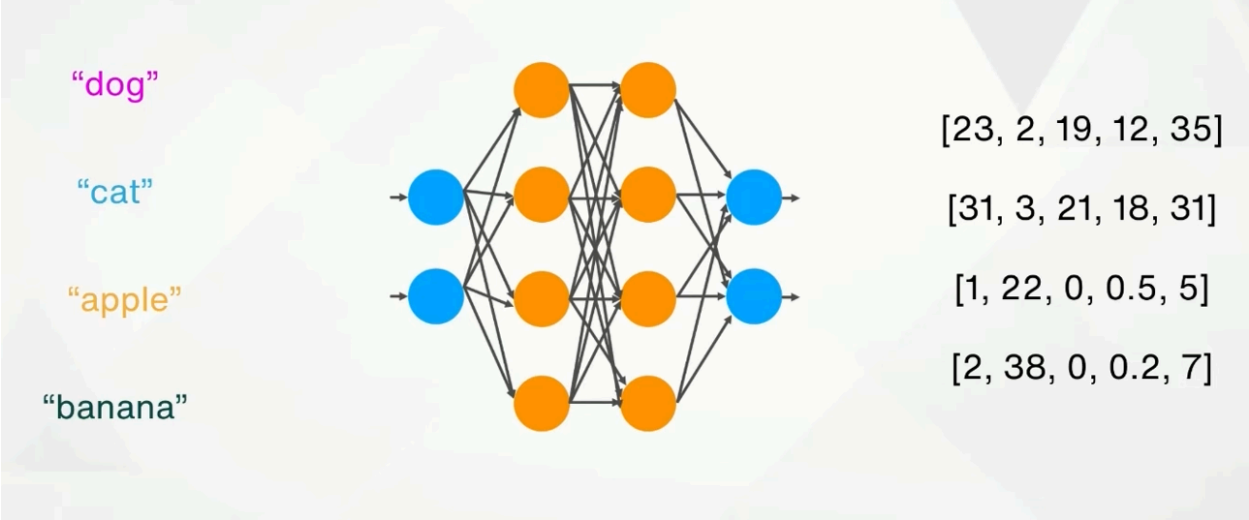
2. **One-Hot Encoding**
    A. Create a dictionary of words
    B. Assign sequential one-hot encoding toe each words
    - Problems: Still fails to capture semantic relationship



3. **Vector Embedding**
    - Illustrate that there is a comparison table for different words, where semantically similar words should have similar vectors.

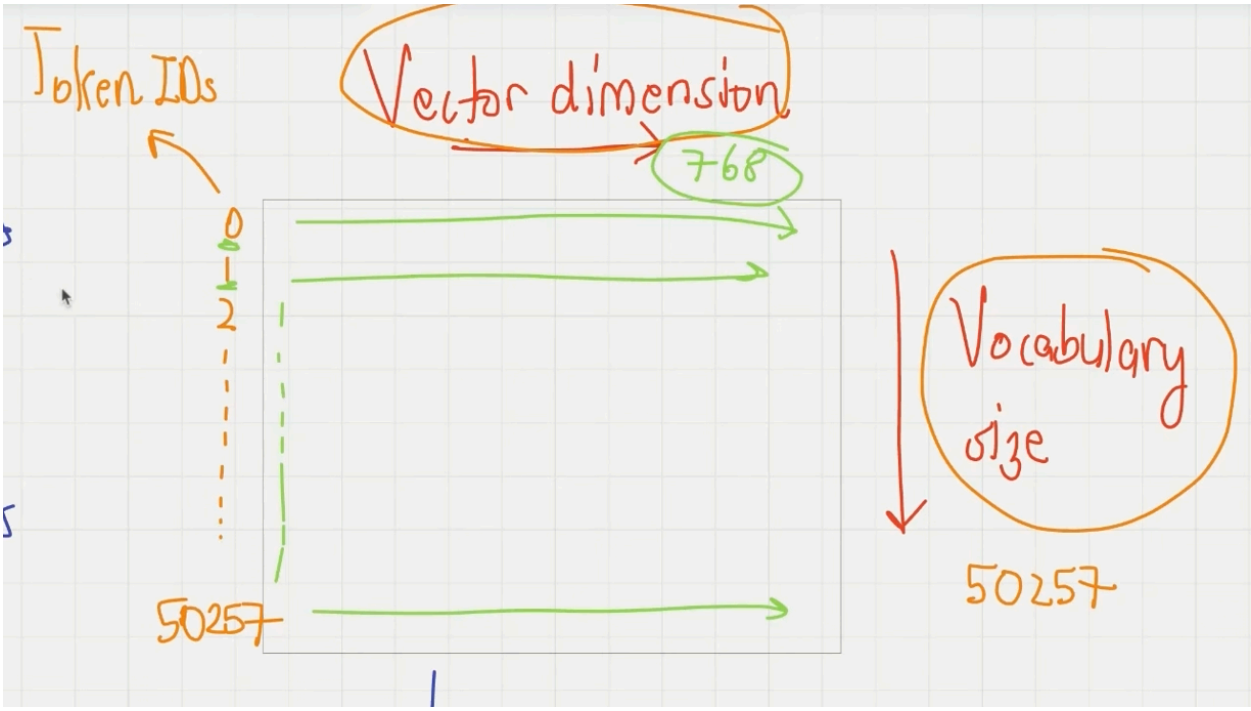


The solution for vector embedding is to train a Neural Network



How are token embeddings created for LLMs?

a. Initialize embedding weights with random values
b. This initialization serves as the starting point for the LLM learning process
c. The embedding weights are optimized as part of LLM training.
d. Vocabulary (Sorted in alphabetical order) [Vector dimension * Vocabulary Size]. {GPT-2 using 768 Vector dimension and 50257 Vocabulary Size}



https://github.com/zhzhhyzh/learn-tokenizer
In 'vector-embedding' folder will explain the vector embeddings working.
In 'input-target-pairs' folder main.py, having partial usage of vector-embedding.

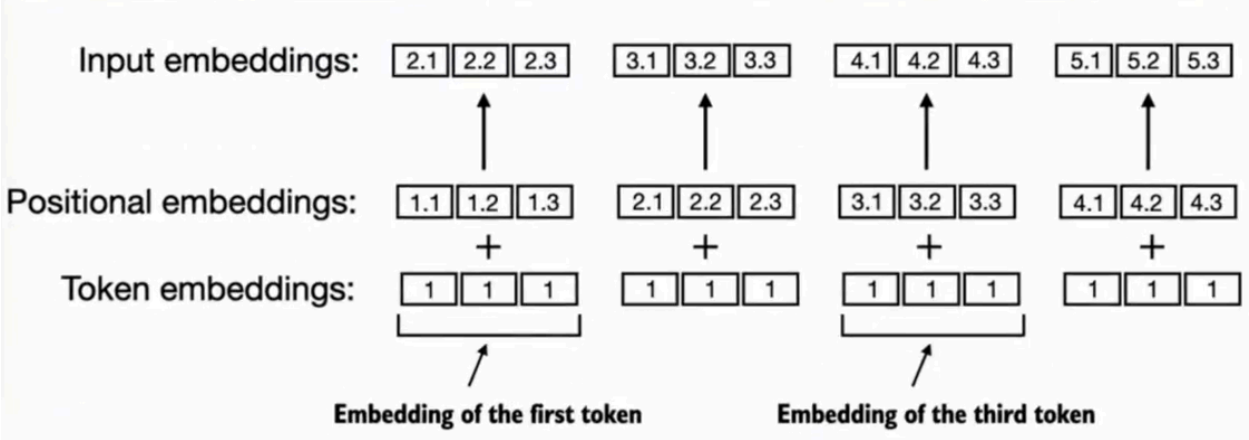
Chapter 10: Positional Encoding

In the embedding layer, the same token ID gets mapped to the same vector representation regardless of where the token ID is positioned in the input sequence
Eg. “The cat sat on the map” and “On the map the cat sat”

It is helpful to inject additional position information to the LLM

There are **two types of positional embeddings**

1. **Absolute**
    - For each position in input sequence, a unique embedding is added to the tokens’ embedding to convey its’ exact location
    - Eg. “The cat sat on the map” will get x + y ,while “On the map the cat sat” will get x+z
    - The positional vectors have the same dimension as the original token embeddings



2. **Relative**
    - The emphasis is on the relative position or distance between tokens. The model learns the relationships in terms of “how far apart” rather than at which exact position
    - Advantage: Model can generalize better to sequence of varying lengths, even if it has not seen such lengths during training.

Both typetypes of positional encoding enable LLMs to understand the order and relationship between tokens. Ensuring more accurate and context aware predictions.

The choice between the two depends on specific and nature of data being processed

- Absolute: Suitable when fixed order token is crucial, such as sequence generation. GPT and original transformer paper are trained based on absolute. (It is used more commonly)
- Relative: Suitable for tasks like language modeling over long sequences, where the same phrase can appear in different parts of the sequence. Useful when analyzing long sequence

Open AI's GPT models use absolute positional embedding that are optimized using the training process. THis optimisation is part of the model training itself

Hands on experience of positional embedding in
https://github.com/zhzhhyzh/learn-tokenizer
In 'input-target-pairs' folder will explain the positional embedding at the bottom of main.py.

Summary

4 Steps of LLM Data Processing

1. Tokenization (Word-based, subword based [BPE tokenizer], or character based)
2. Token embeddings (To convert token IDs to vectors)
3. Positional Embeddings(Encoding information about position)
4. Input Embeddings = Token Embedding + Positional Embedding

References

Gage, P. (1994). A new algorithm for data compression. *The C Users Journal archive, 12*, 23-38.