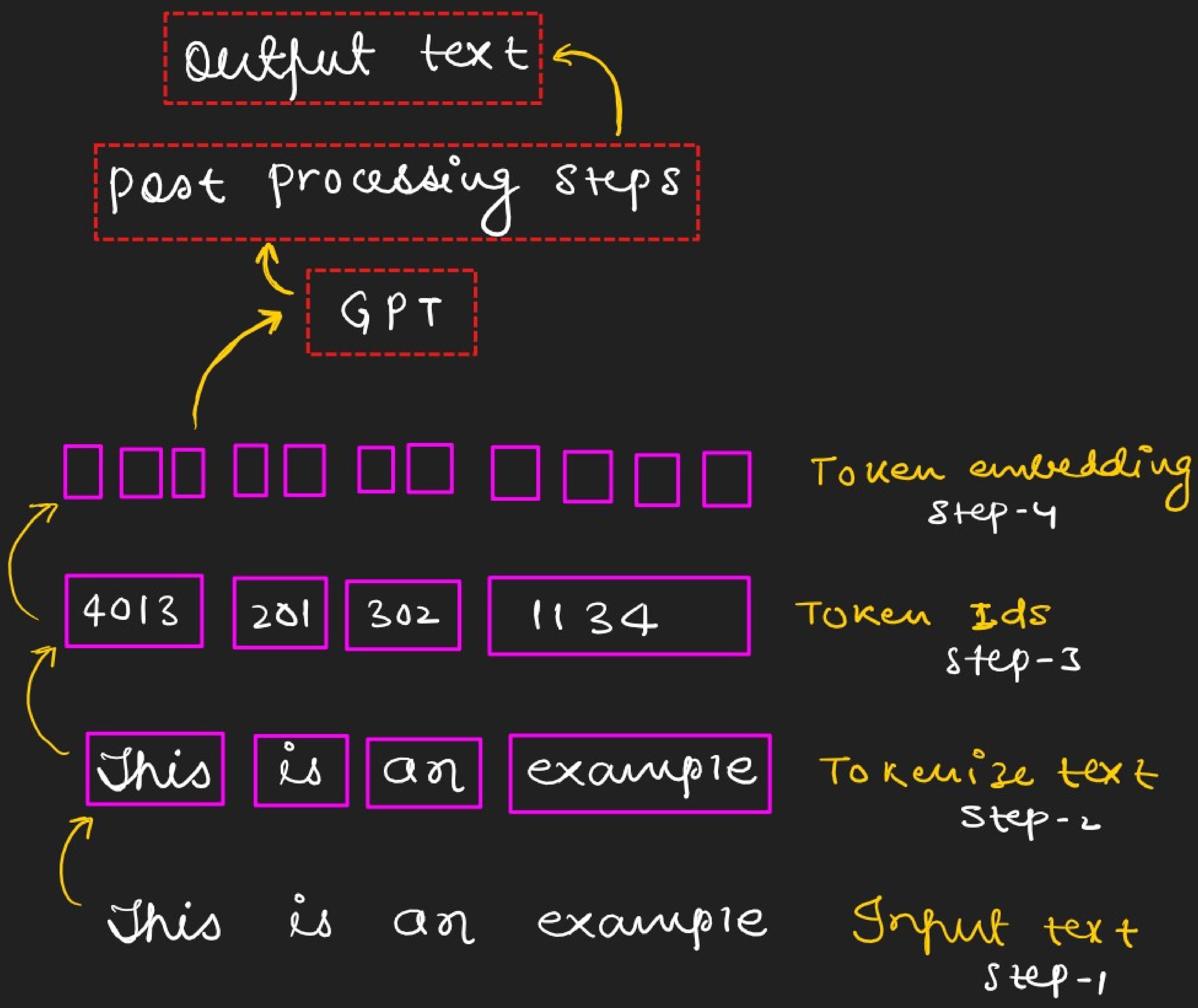
Lecture-10: Token Embedding

vector - embedding



Today, we goint to learn about

Step 3: creating token embeddings

1. conceptual understanding of why token embeddings are needed?

2. small hands on demo: playing with token embeddings.

3. How are token embeddings created for LLM ?

Build

LLMs from Scratch

What are token embeddings and why we need them?

Representing Words Numerically

- Computers need numerical representation of words
- How can we represent words in numbers?

Can we assign random numbers to each word?

“cat” → 34

“book” → 2.9

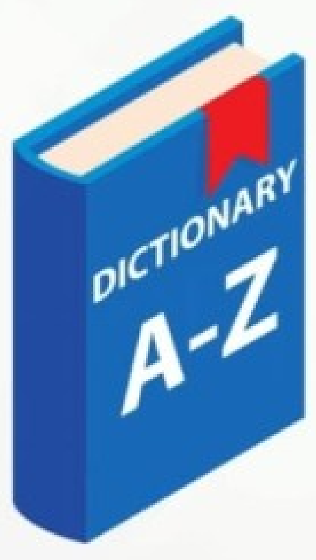
“tablet” → -20

“kitten” → -13

What About One-Hot Encoding?

1) Create a dictionary of words
2) Assign sequential one-hot encoding to each word

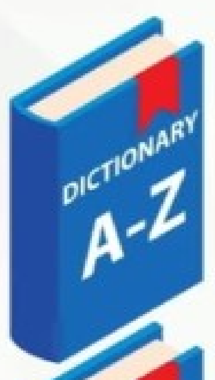
“dog” →



→ [0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,..........]

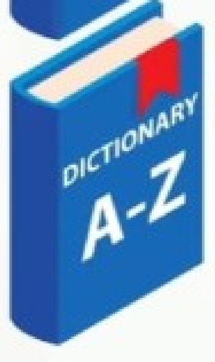
The Problem With One-Hot Encoding

“dog” →



→ [0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,..........]

“puppy” →



→ [0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,..........]

One-hot encoding also fails to capture semantic relationship

Semantically Similar Words Should Have Similar Vectors

	“dog”	“cat”	“apple”	“banana”
has_a_tail	23	31	1	2
is_eatable	2	3	22	38
has_4_legs	19	21	0	0
makes_sound	12	18	0.5	0.2
is_a_pet	35	31	5	7

Vectors can capture semantic meaning

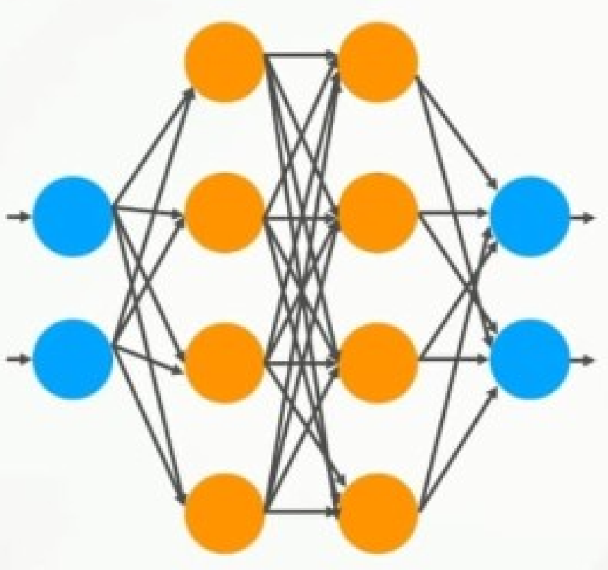
We Can Train a Neural Network To Create Vector Embedding

“dog”

“cat”

“apple”

“banana”



[23, 2, 19, 12, 35]

[31, 3, 21, 18, 31]

[1, 22, 0, 0.5, 5]

[2, 38, 0, 0.2, 7]

③ How are token embeddings created for LLM?

(a.) Initialize embedding weights with random values.

(b.) This initialization serves as the starting point for the LLM learning process.

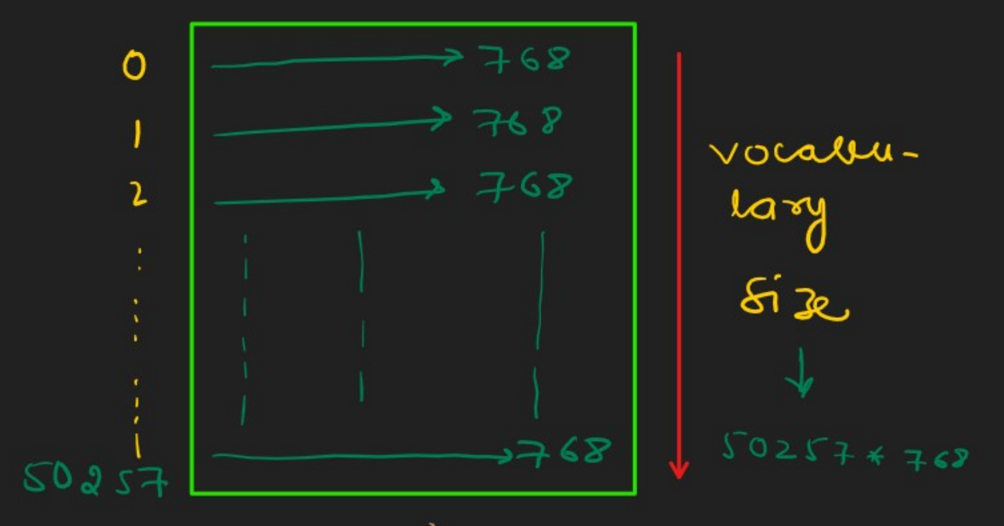
(c.) The embedding weights are optimised as part of the LLM training process.

(d.) vocabulary (usually sorted alphabetically)

- brown → 0
- dog → 1
- fox → 2
- jumps → 3
- lazy → 4
- over → 5
- quick → 6
- the → 7

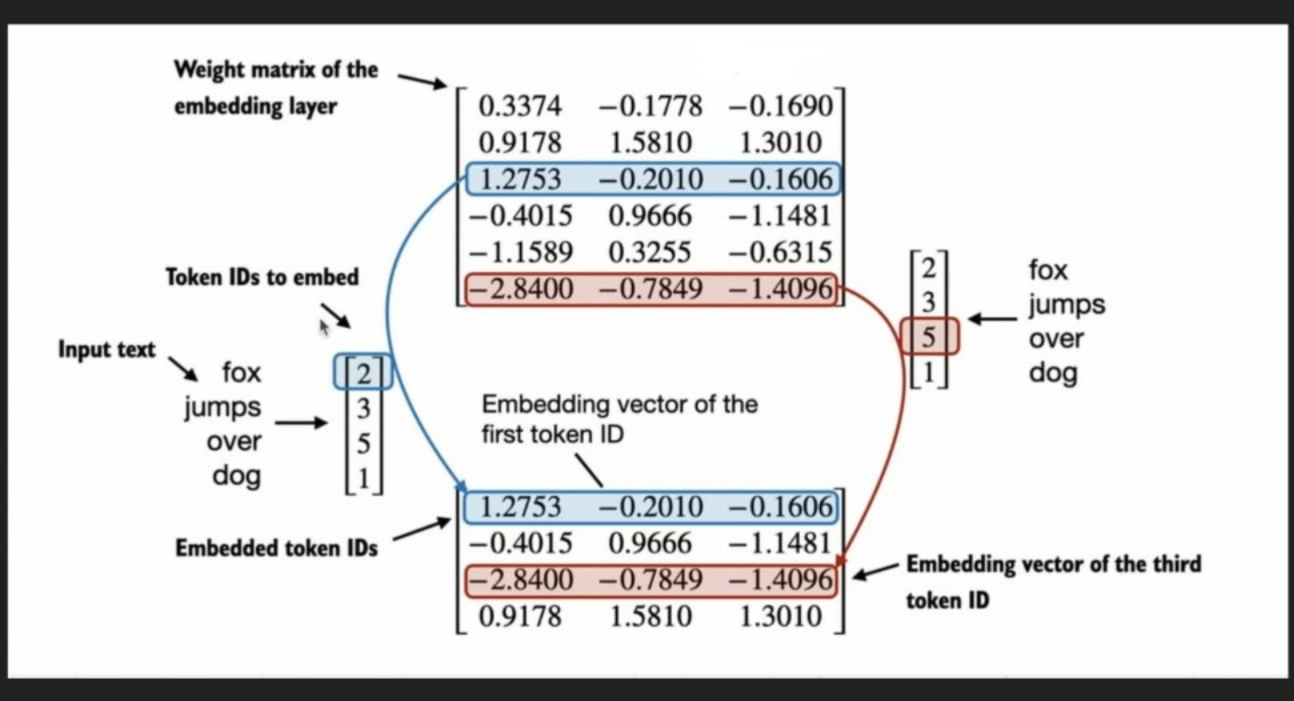
token IDs are converted into embedding vectors

vector dimension



Embedding layer weight matrix

(e.) The embedding layer is a lookup operation that retrieves rows from the embedding layer weight matrix using a token IDs.



(f) what an embedding layer actually does?

* suppose we have the following 3 training example

idx = torch.tensor([2,3,1])

* Embedding dimension = 5

* embedding = torch.nn.embedding(4,5)

* embedding.weight

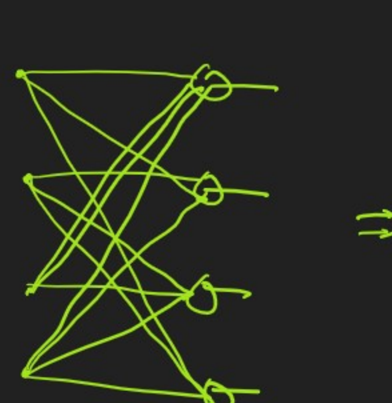
4x5

* embedding(idx)

extracted matrix

This is the same as neural network linear layer.

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



⇒ output

$X \cdot W^T$

transpose

$$W^T = \begin{bmatrix} w_{11} & w_{21} & w_{31} & w_{41} & w_{51} \\ w_{12} & w_{22} & w_{32} & w_{42} & w_{52} \\ w_{13} & w_{23} & w_{33} & w_{43} & w_{53} \\ w_{14} & w_{24} & w_{34} & w_{44} & w_{54} \end{bmatrix}$$

Neuron Neuron

* Both embedding layer and NN linear layer lead to same output.

* Embedding layer is much more computationally efficient, since NN layer has many unnecessary multiplication with zero.