Token Embeddings

Tokenized → token embeddings → GPT → post processing → output

↑ tokenized input

---

(1) conceptual understanding of why token embeddings are needed

(2) Demo

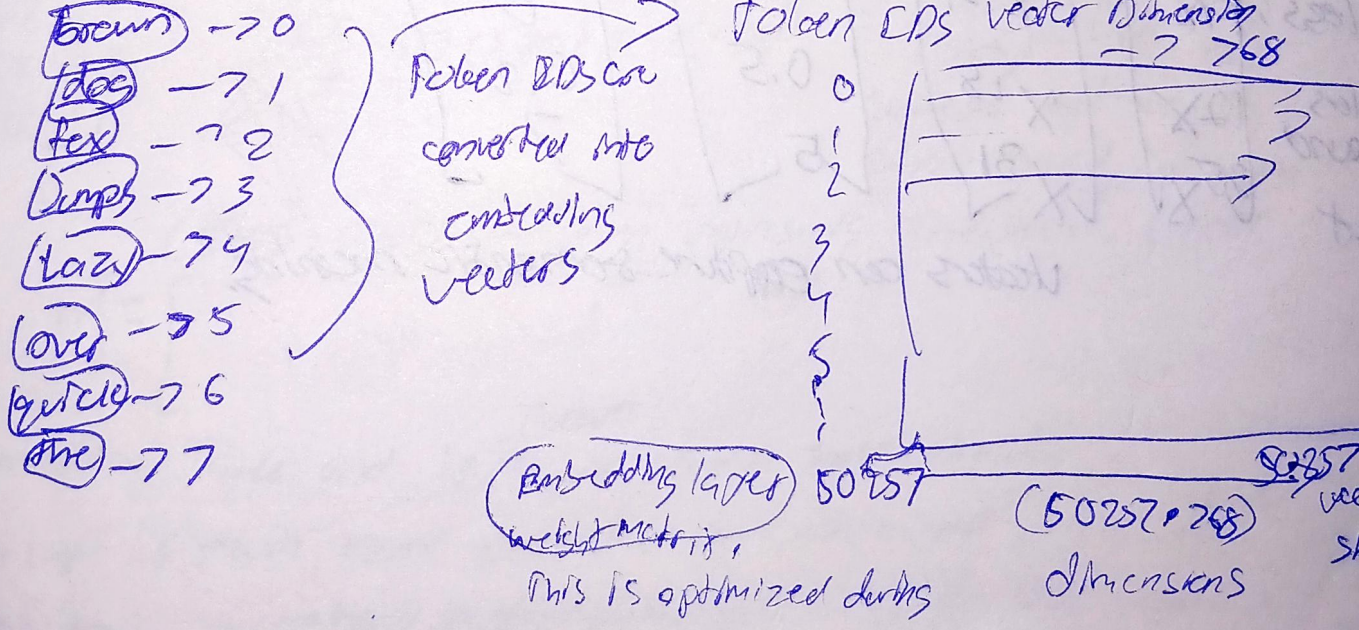
(3) How are token embeddings created for LLMs

(a) Initialize embedding weights w/ random values

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

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

(d) Vocabulary (usually sorted alphabetically)



What are token embeddings?

Often to represent words as #'s?

"cat" -> 34

"dog" -> 3, 2, etc

"hatred" -> -20

"letter" -> -15

"cat" and "letter" are semantically related, however assigned numbers 34 and -15 cannot capture this relation

What about one-hot encodings?

1 create dictionary of words

2 assign sequential one-hot encodings to each word

"dog" -> dict -> [0,0,1,0,0,0,...]

"puppy" -> dict -> [0,0,...,1,0,...]

Also fails to capture semantic relationship.

Semantically similar words should have similar vectors

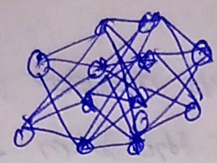
	"dog"	"cat"	"apple"	"banana"
hatred	X23	X31	1	2
edible	2	3	X22	X 38
4 legs?	X18	X 21	0	0
makes sound	12X	X 18	0.5	0.2
pet	35 X	X 31	5	7

apple + banana ≈ dog + cat

Vectors can capture semantic meaning.

we can train a Neural Network to create vector embeddings

"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]

(7) Embedding layer is a lookup operation that retrieves rows from the embedding layer weight matrix using a token ID

(8) What an embedding layer actually does?

* suppose we have following 3 training examples

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

[0, 1, 2, 3] ---- (5)

* Embedding dimension = 5

* Embedding = torch.embedding(4, 5)

* Embedding weights ------

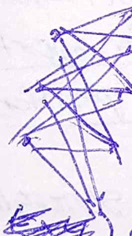
* Embedding (idx) (only 2, 3, 1)

(2) -----
(3) -----
(1) -----

=> Same as neural network linear layer

one-hot

[0 0 1 0] - 2
[0 0 0 1] - 3
[0 1 0 0] - 1



=> output = $X \cdot W^T$ = (3×5)

Each input(3) has 5 dimensions, shape input = 3×5

Same as what we get when we used embedding layer

(4×5) each column represents a neuron

W^T = [ ]

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

* Embedding layer is much more computationally efficient, since NN layer has many unnecessary multiplications by 0 b/c of one-hot encoding