Lecture 6 Activity: A Language Model by Hand

E190BO Names: _____

Work in groups of 2–3. In this activity you will push one phrase, **the fluffy blue creature**, through a tiny language model by hand, in the same order a real GPT does it: tokens → embeddings → attention → MLP → next-token prediction. At each **Checkpoint**, compare your result with the answer on the last page (answers only, not how to get them). If yours doesn't match, sort it out before moving on.

Part 1: Tokens and embeddings

Tokenization. The first step is to break the input sequence of characters into **tokens**, which are typically pieces of words. In our simple example, we will simply assume that each word is a token. Our model's entire vocabulary has six tokens. **BOS** is a special "beginning of sequence" token that is placed at the start of every sequence.

token	BOS	blue	creature	fluffy	is	the
token ID	0	1	2	3	4	5

(a) Write the token IDs for the sequence **BOS the fluffy blue creature**.

Training. A GPT is trained to **predict the next token**. Suppose our training text begins "the fluffy blue creature is . . . ". At *every* position, the model sees the tokens so far and must predict the one that comes next. BOS gives the model something to look at when it predicts the very first word.

(b) Fill in the target (the correct next token) for each input position.

position	0	1	2	3	4
input token	BOS	the	fluffy	blue	creature
target token					

Embeddings. Each token ID is used to look up a row in the **word embedding table** (left). Our embeddings have 5 features: [*fuzzy*, *blue*, *animal*, *verb*, *position*]. Each position also has a row in the **position embedding table** (right). A token's embedding is the **sum** of its word embedding and its position embedding.

Word embeddings

	fuzzy	blue	animal	verb	pos.
BOS	0	0	0	0	0
blue	0	1	0	0	0
creature	0	0	1	0	0
fluffy	1	0	0	0	0
is	0	0	0	1	0
the	0	0	0	0	0

Position embeddings

	fuzzy	blue	animal	verb	pos.
position 0	0	0	0	0	0
position 1	0	0	0	0	1
position 2	0	0	0	0	2
position 3	0	0	0	0	3
position 4	0	0	0	0	4

(c) Compute the embedding e of each token in our sequence.

	word embedding	position embedding	e = sum
BOS (pos. 0)			
the (pos. 1)			
fluffy (pos. 2)			
blue (pos. 3)			
creature (pos. 4)			

Checkpoint 1: check the last page.

In a real model, both tables are learned, the embeddings have hundreds of dimensions, and the position embeddings are dense vectors (no single feature means "position"). Our simplified version illustrates the idea: a token's position is encoded into the embedding itself, in a way the model can later use to figure out where the token is.

Part 2a: Draw the attention pattern (no numbers)

In each grid, a **row** is a token *being updated* (it asks the question: its **query**). A **column** is a token *being looked at* (it answers: its **key**). The 3Blue1Brown video draws the same grid flipped (queries along the top).

(a) In **both** grids, first put an **X** in every square that the causal mask blocks.

(b) Then **shade** the squares where each row's token puts most of its attention, according to the head's job.

Head A: "Look at the token right before me." (BOS has nothing before it, so it looks at itself.)

	BOS	the	fluffy	blue	creature
BOS					
the					
fluffy					
blue					
creature					

Head B: "If I'm a noun, look at the adjectives that describe me." (If not a noun, spread attention evenly over every token it can see.)

	BOS	the	fluffy	blue	creature
BOS					
the					
fluffy					
blue					
creature					

Checkpoint 2: check the last page.

(c) Why is a causal mask needed? Note that during training, a transformer processes all tokens at the same time. This means that each token sees all other tokens as an unordered set, which is why we need to encode the token positions into the embeddings themselves and prevent each token from seeing future tokens. What would be the problem if tokens could see future tokens? *(Hint: look at your table in Part 1(b).)*

Part 2b: One attention head, by hand

Now let's compute **Head B** with numbers. The model multiplies each embedding by learned matrices W_Q, W_K, W_V to get a query, a key, and a value for every token. We'll skip those multiplications and give you the results. Queries and keys have dimension $d_k = 4$, with features [*adjective*, *noun*, *article*, *start*].

	key k ("what am I?")	query q ("what am I looking for?")	value v ("what I pass along")
BOS	[0, 0, 0, 1]	[0, 0, 0, 0]	[0, 0, 0, 0, 0]
the	[0, 0, 1, 0]	[0, 0, 0, 0]	[0, 0, 0, 0, 0]
fluffy	[1, 0, 0, 0]	[−6, 6, −6, −6]	[1, 0, 0, 0, 0]
blue	[1, 0, 0, 0]	[−6, 6, −6, −6]	[0, 1, 0, 0, 0]
creature	[0, 1, 0, 0]	[6, −6, −6, −6]	[0, 0, 1, 0, 0]

(a) In words: what is *creature*'s query looking for? What are *fluffy* and *blue* looking for? What about BOS and *the*?

Inner (dot) products are a way of computing **similarity**. We first compute scores indicating the similarity between each query and all keys up to and including the current position.

(b) Scores. Fill in each score $= \frac{q_{\text{row}} \cdot k_{\text{column}}}{\sqrt{d_k}}$. **(c) Mask.** Put $-\infty$ in all masked squares. (Why?)

	k_{BOS}	k_{the}	k_{fluffy}	k_{blue}	k_{creature}
q_{BOS}					
q_{the}					
q_{fluffy}					
q_{blue}					
q_{creature}					

Checkpoint 3: check the last page.

(d) Weights. Apply softmax to each row. Softmax: replace each score s with e^s, then divide by the row total. Round to 2 decimals.

	BOS	the	fluffy	blue	creature
BOS					
the					
fluffy					
blue					
creature					

Checkpoint 4: check the last page.

(e) Update. The update for a token is $\Delta e = \sum(\text{weight}) \times (\text{value})$, and its new embedding is $x = e + \Delta e$ (a residual connection: the update is *added* to the embedding from Part 1). Compute x_{creature}. In words, what does *creature* "know" now that it didn't before?

(f) Compute x_{fluffy} too. Did *fluffy* learn anything about *creature*? Why or why not?

Checkpoint 5: check the last page.

What we skipped: a real GPT runs several heads in parallel. Their results are concatenated and passed through one more linear transformation before being added to the embedding. There are also normalization layers throughout. So don't be surprised when you see these extra pieces in Lab 3!

Part 3: The MLP

After attention, each token goes through an **MLP** (a small dense network, like the one you built in Lab 2). The two blocks do different jobs: **attention shares context between tokens**, while **the MLP processes each token in isolation**, using the same weights for every token. Like attention, the MLP has a residual connection: its output is *added* to its input.

Our MLP has one layer with a ReLU (a real one has a second linear layer after the ReLU, which we skip):

$$h = \text{ReLU}(x\,W + b), \qquad x_{\text{out}} = x + h$$

$$W = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}, \qquad b = \begin{bmatrix} 0 & 0 & 0 & -1 & 0 \end{bmatrix}$$

(a) Only one entry of $x\,W + b$ can be non-zero. Which feature is it, and which input features feed into it? In words, when does this MLP "fire"?

(b) Compute h and x_{out} for **creature**, using x_{creature} from Part 2b.

(c) Now do the same for **the**, whose vector after attention is $x_{\text{the}} = [0, 0, 0, 0, 1]$. What did the ReLU do here, and why is that the right behavior?

Checkpoint 6: check the last page.

Part 4: Predicting the next token

A real GPT repeats attention + MLP many times. We'll stop after one round. To predict the token that follows *creature*, the model compares *creature*'s final vector with the embedding of **every token in the vocabulary**. Recall that a dot product measures similarity. The matrix used for this comparison is called the **LM head**. Sometimes the LM head and the word embedding table are **tied** (they are the same matrix), and sometimes they are different. Here we assume they are tied, so we reuse the word embedding table from Part 1.

(a) Logits. Compute the dot product of x_{out} for *creature* with each row of the word embedding table. These six numbers are called the **logits**.

(b) Probabilities. Apply softmax to the logits. Round to 2 decimals.

	BOS	blue	creature	fluffy	is	the
logit						
probability						

Checkpoint 7: check the last page.

(c) Which token does the model think is most likely to come next? Trace back through Parts 2b and 3: which step made that token likely? Did the *position* feature affect the logits?

During training, we compare these probabilities with the correct target from Part 1(b) using the **cross-entropy loss**: $\text{loss} = -\ln(p_{\text{target}})$, where p_{target} is the probability the model gave to the correct next token.

(d) Compute the loss for this one prediction. What would the loss be if the model had given the target a probability of 1? A probability near 0?

Checkpoint 8: check the last page.

The total training loss averages this over **every position** in the sequence (each position has its own target) and over all the sequences in the dataset. Gradient descent then adjusts every table and matrix you saw today to make the loss smaller.

During inference (generating text), there is no target. Instead, we **sample** a token from the probability distribution. That token is appended to the sequence, and the whole process repeats to generate the token after it. We stop when a special end-of-sequence token is drawn.

(e) If you generated from this model many times starting from "the fluffy blue creature", would you always get the same next word? About how often would you get "is"?

Checkpoint answers

Answers only. If yours doesn't match, go back and find out why before moving on.

Checkpoint 1. $e_{\text{BOS}} = [0, 0, 0, 0, 0]$, $e_{\text{the}} = [0, 0, 0, 0, 1]$, $e_{\text{fluffy}} = [1, 0, 0, 0, 2]$, $e_{\text{blue}} = [0, 1, 0, 0, 3]$, $e_{\text{creature}} = [0, 0, 1, 0, 4]$.

Checkpoint 2. Each grid has 10 X's, all above the diagonal. Head A shades 5 squares; Head B shades 12 (rows of 1, 2, 3, 4, and 2 squares).

Checkpoint 3. The *creature* row is $[-3,\ -3,\ 3,\ 3,\ -3]$. Before masking, *fluffy* and *blue* each score 3 with k_{creature}, but that square is masked ($-\infty$).

Checkpoint 4. Every row sums to 1. The *blue* row is $[0.25,\ 0.25,\ 0.25,\ 0.25,\ 0]$. The *creature* row is $[0.00,\ 0.00,\ 0.50,\ 0.50,\ 0.00]$.

Checkpoint 5. $x_{\text{creature}} = [0.5,\ 0.5,\ 1,\ 0,\ 4]$ and $x_{\text{fluffy}} = [1.33,\ 0,\ 0,\ 0,\ 2]$.

Checkpoint 6. creature: $h = [0, 0, 0, 2, 0]$ and $x_{\text{out}} = [0.5,\ 0.5,\ 1,\ 2,\ 4]$. the: $h = [0, 0, 0, 0, 0]$, so $x_{\text{out}} = x_{\text{the}}$ (unchanged).

Checkpoint 7. Logits: $[0,\ 0.5,\ 1,\ 0.5,\ 2,\ 0]$. Probabilities: $[0.06,\ 0.11,\ 0.18,\ 0.11,\ 0.48,\ 0.06]$.

Checkpoint 8. loss ≈ 0.73.