Self-attention is a key mechanism in transformer models that allows each token to "look at" other tokens to collect relevant information. This activity will help you understand how it works step-by-step.

Overview

In this activity, we'll work with a simple sentence: *"the cat chases the"* (and we'd expect it to predict *"mouse"*).

We want to understand how the last token "*the*" could gather information to predict that "*mouse*" should come next. It does this by sending out a query that essentially asks "what's typically chased in this context?"

<div align="center">

the   cat chases the
pos 0 pos 1 pos 2 pos 3

</div>

Figure 1: Our sentence with position indices

Given Information

We'll work in 2D with pre-computed key, query, and value vectors. (In a real Transformer these are computed by matrix multiplications.) **For the query vector for "*the*" (position 3), use [2, 3]**.

Token	Key Vector	Value Vector
"the" (pos 0)	[1, 0]	[0, 0]
"cat" (pos 1)	[3, 1]	[4, 1]
"chases" (pos 2)	[0, 3]	[2, 2]
"the" (pos 3)	[2, 0]	[0, 0]

Attention Calculation

First compute the attention scores by taking the dot product of the last token's query vectorwith each key vector. Then normalize by sum (most real Transformers use softmax), then multiply by the corresponding value vector.

Token	Attention Score (Q·K dot product)	Normalized Weight (Score ÷ Sum)	Weighted Value (Weight × Value)
"the" (pos 0)	2*1 + 3*0 = 2		[0, 0]
"cat" (pos 1)			
"chases" (pos 2)			
"the" (pos 3)			
column sum:		*should equal 1.0*	

The bottom-right corner is the *output* of this attention calculation. This output vector represents what information the token "*the*" has gathered from all other tokens. It would be used by the next layers to predict "*mouse*".

Note: this a simplified version of the attention calculation. In a real transformer:

• There are many of these attention "heads", each with its own set of key, query, and value vectors.
• The key, query, and value vectors are computed by multiplying the input embeddings by learned weight matrices.
• The attention scores are divided by a constant factor (to prevent large dot products).
• The softmax function is applied to the attention scores to ensure they sum to 1.
• There's one final step where the output vectors from all heads are concatenated and passed through a linear layer.
• There's also a residual connection and layer normalization.
• These attention layers alternate with MLP layers.

Discussion Questions

1. Which token(s) received the most attention from "the"? What information did it gather?

2. How would the attention pattern change if the query was [1, 3] instead? How would this affect the output?

3. If we wanted "the" to pay more attention to "chases", should we change its query vector or the key vector of "chases"? How?

4. If we wanted the first number in the output vector to increase a bit, how could we change (a) the query vector, (b) the key vectors, and (c) the value vectors?

Before you leave, pick a couple of these questions to react to:

1. What was the most important concept from today for you?
2. What was the muddiest concept today?
3. How does what we did today connect with what you've learned before?
4. What would you like to review or clarify next time we meet?
5. What are you curious, hopeful, or excited about?