INTERACTION WITH LARGE SCALE MODELS

LIACD/1

University of Beira Interior,
Department of Informatics

Hugo Pedro Proença,
hugomcp@di.ubi.pt, 2024/2025

INTERACTION WITH LARGE SCALE MODELS

[10]

- Hands-On Transformers;

Transformers Hands-On

Consider a simple sentence: "***The cat sat on the mat.***"

- The query, key, and value vectors for the tokens are given as follows:

Token	Query	Key	Value
The	[1, 0]	[1, 0]	[0, 1]
cat	[0, 1]	[0, 1]	[1, 0]
sat	[1, 1]	[1, 1]	[0, 0]
on	[0, 0]	[1, 1]	[1, 1]
the	[1, 0]	[0, 0]	[0, 0]
mat	[0, 1]	[1, 0]	[1, 1]

- **Step 1:** Calculate the attention score between the tokens **"The"** and **"cat"**.
- **Step 2:** Given the attention formula:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

where dk=2d_k = 2dk=2 (the dimension of the query and key vectors), calculate the attention weight for the token **"cat"**.

- **Step 3:** What would be the output for the token **"cat"** after applying attention?

Transformers Hands-On – Positional Encoding

- Understand the role of **Positional Encoding** in Transformers and how it differs from RNNs and LSTMs. Then, for a sequence of length 3 and d_{model}=4, calculate the positional encodings for each token using the following formula:

$$\mathrm{PE}(pos, 2i) = \sin\left(\frac{pos}{10000^{2i/d_{\text{model}}}}\right)$$

$$\mathrm{PE}(pos, 2i+1) = \cos\left(\frac{pos}{10000^{2i/d_{\text{model}}}}\right)$$

where pos is the position of the token (0-based), and i is the dimension index.

1. For position 0, 1, and 2, compute the positional encoding vectors.
2. Why are sinusoidal functions used in this formula? Discuss their properties and benefits in representing positions.

The sequence has 3 positions: **pos = 0, 1, 2**.
We need to calculate the positional encoding for each token.
For d_{model}=4d, we have two dimensions (i=0, 1) for each position.

- $\mathrm{PE}(0,0) = \sin\left(\frac{0}{10000^{0/4}}\right) = 0$
- $\mathrm{PE}(0,1) = \cos\left(\frac{0}{10000^{1/4}}\right) = 1$
- **Positional Encoding** = [0, 1, 0, 1]

Pos 0

- $\mathrm{PE}(1,0) = \sin\left(\frac{1}{10000^{0/4}}\right) = \sin(1) \approx 0.841$
- $\mathrm{PE}(1,1) = \cos\left(\frac{1}{10000^{1/4}}\right) = \cos(1) \approx 0.540$
- **Positional Encoding** = [0.841, 0.540, 0.841, 0.540]

Pos 1

- $\mathrm{PE}(2,0) = \sin\left(\frac{2}{10000^{0/4}}\right) = \sin(2) \approx 0.909$
- $\mathrm{PE}(2,1) = \cos\left(\frac{2}{10000^{1/4}}\right) = \cos(2) \approx -0.416$
- **Positional Encoding** = [0.909, -0.416, 0.909, -0.416]

Pos 2

Transformers Hands-On – Self-Attention

In a multi-head attention mechanism, we use multiple sets of *query*, *key*, and *value* matrices to compute attention over different subspaces.

Step 1: Imagine you have a sequence of tokens with the following query and key vectors:

1. Q1: [1, 0], Q2: [0, 1], Q3: [1, 1]
2. K1: [0, 1], K2: [1, 0], K3: [1, 1]

Compute the attention score between Q1 and all keys (K1, K2, K3).

The attention score between token "The" and "cat" is calculated as the **dot product** of the query vector for "The" and the key vector for "cat":

$$Q_{\text{The}} \cdot K_{\text{cat}} = [1, 0] \cdot [0, 1] = 1 \times 0 + 0 \times 1 = 0$$

Transformers Hands-On – Self-Attention

The attention weight for each token is given by the formula:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

calculate the attention scores for all pairs of query and key vectors:
- **"The" and "cat"**: As computed, score = **0**.
- **"The" and "sat"**: Score=[1,0]·[1,1]=1×1+0×1=1
- **"The" and "on"**: Score=[1,0]·[0,0]=0
- **"The" and "the"**: Score=[1,0]·[1,0]=1
- **"The" and "mat"**: Score=[1,0]·[0,1]=0

Now compute the **scaled scores** (divide by dk=2≈1.414\sqrt{d_k} = \sqrt{2} \approx 1.414dk=2≈1.414) and apply **softmax** to these scores.
Finally, multiply the softmax result by the value vectors to get the output for "The".

Output for "The"
The output for "The" will be a weighted sum of the value vectors based on the attention weights computed in step 2.

Transformers Hands-On – Transformer Block

The transformer block consists of two main components:

Multi-Head Self-Attention

Feedforward Neural Network (FFN)

Explain the process step-by-step for the sentence **"The quick brown fox"**. In your explanation, describe how the multi-head self-attention mechanism works in the transformer block:

- How does each token attend to other tokens in the sentence?
- What role does the feedforward network play in processing the attention output?

Multi-Head Attention:
Each token attends to every other token in the sequence. For example, "quick" will attend to "the", "brown", and "fox".
Multiple attention heads are used to capture different relationships.

Feedforward Neural Network (FFN):
After self-attention, each token's output is passed through a feedforward network (typically two linear layers with ReLU activation). This helps capture more complex relationships.

Transformers Hands-On – Decoder in Transformer Architectures

The transformer architecture is divided into an **encoder** and a **decoder**. Explain the role of the decoder in tasks like machine translation.

- What information does the decoder take as input?
- How does the self-attention in the decoder differ from that in the encoder?
- In what way does the decoder generate the output sequence one token at a time? Explain the mechanism used for this.

Input to Decoder:
The decoder receives the encoder's output and previously generated tokens. It uses this information to generate the next token in the sequence.

Self-Attention in Decoder:
The decoder's self-attention is masked to prevent attending to future tokens. This ensures the model generates tokens one-by-one.

Generating Tokens:
The decoder generates tokens autoregressively, meaning it generates one token at a time based on previous tokens and the encoder's output.

Transformers Hands-On – Transformers vs RNNs vs LSTMs

In your own words, explain the key difference between Transformers and traditional RNNs/LSTMs. Discuss the advantages of the transformer architecture in handling long-range dependencies.

Explain why transformers are more parallelizable than RNNs/LSTMs. How does this impact training efficiency?

Key Difference:
Transformers can process all tokens in parallel, while RNNs/LSTMs process tokens sequentially. This makes transformers much faster to train, especially for long sequences.

Parallelism in Transformers:
Since transformers use attention mechanisms, they can attend to all tokens in a sequence simultaneously, unlike RNNs/LSTMs which process one token at a time. This allows for better parallelization, speeding up training.

Transformers Hands-On –Masked Language Models

Consider a masked language model (MLM) task where a token in a sentence is randomly masked, and the model is tasked with predicting the masked word. For the sentence:
"The cat sat on the [MASK]."

- What are the advantages of using a transformer model like BERT for this task instead of an RNN or CNN?
- How does BERT utilize the bidirectional context to predict the masked word? Describe the architecture of BERT in relation to masked language modeling.
- How does the attention mechanism help BERT predict the masked token accurately? What role does the positional encoding play in this process?

BERT's Bidirectional Nature:
BERT uses both left and right context to predict the masked token. It uses the **encoder-only transformer** architecture, meaning it can look at the whole sentence at once, unlike a unidirectional model like GPT.
How Attention Helps:
The self-attention mechanism allows each token to pay attention to all other tokens in the sentence. For the masked word "[MASK]", the model uses the surrounding context (e.g., "The cat sat on the") to predict the correct word ("mat").