EECS 182 Deep Neural Networks
Fall 2025 Anant Sahai and Gireeja Ranade

Discussion 10

1. Attention Mechanisms for Sequence Modelling

Recall the sentence capitalization problem from last week's discussion. Given an input which is a mixed case sequence like "<U> I am a student", the model should identify this as an upper-case task based on token <U>, and convert it to "I AM A STUDENT". Similarly, given "<L> I am a student", the lower-case task is to convert it to "i am a student".

This character-level task is particularly difficult for RNNs. Last week, we saw that in an encoder-decoder architecture using vanilla RNNs, all information must be encoded in the network hidden states. In particular, this results in a *information bottleneck*: the final hidden state of the encoder needs to store all the information about the input sequence, which can result in issues for long sequences.

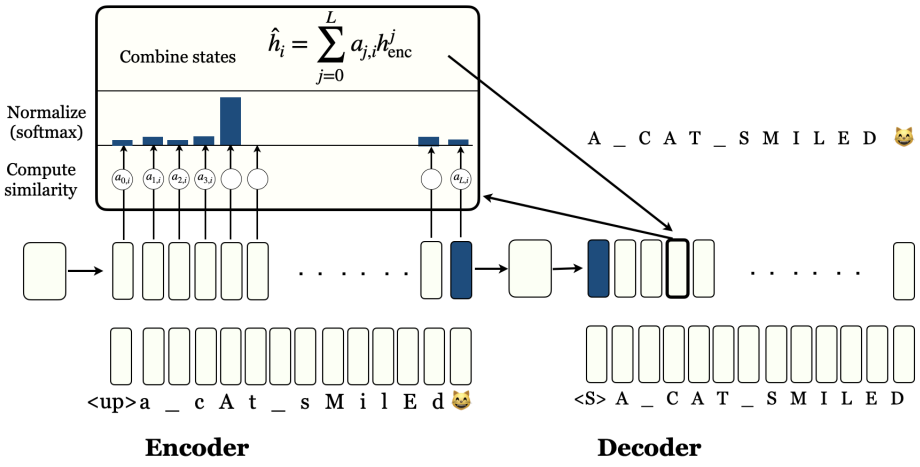

Figure 1: Attention Mechanism for Sequence Modelling with RNNs.

Instead of storing all the information in the hidden state, we can use attention to selectively store information. **How would you modify the encoder-decoder architecture to incorporate attention, and how does this help overcome the information bottleneck?**

2. Relative Positional Embedding

Year	Model Name	Type of Positional Embedding (PE)
2018, 2019, 2020	GPT-1, 2, and 3	Learned absolute PE
2021	GOPHER	Learned relative PE
2022	PaLM	RoPE
2023, 2024	LLaMA, LLaMA-2, and LLaMA-3	RoPE
2024	Grok-1, 2	RoPE
2024, 2025	Deepseek-v2, v3	decoupled RoPE
2025	LLaMA-4	iRoPE (interleaving RoPE and NoPE)
2025	Qwen-3	RoPE
2025	Gemma 3	RoPE

Table 1: Types of positional embeddings used in large language models over the years.

(a) **Why do we need positional encoding? Describe a situation where word order information is necessary for the task performed.**

(b) **How does relative positional embedding work? How is it different from the absolute positional embedding?**

(c) Consider a list of 2-dimensional vectors $\left[\left(x_1^{(1)}, x_1^{(2)}\right), \left(x_2^{(1)}, x_2^{(2)}\right), \left(x_3^{(1)}, x_3^{(2)}\right), \ldots\right]$. Now for any choice of parameter ω, the corresponding RoPE encoding is

$$\mathrm{RoPE}\left(x_t^{(1)}, x_t^{(2)}, t\right) = \begin{pmatrix} \cos t\omega & -\sin t\omega \\ \sin t\omega & \cos t\omega \end{pmatrix} \begin{pmatrix} x_t^{(1)} \\ x_t^{(2)} \end{pmatrix} = \begin{pmatrix} x_t^{(1)} \cos t\omega - x_t^{(2)} \sin t\omega \\ x_t^{(2)} \cos t\omega + x_t^{(1)} \sin t\omega \end{pmatrix}$$

Equivalently, if we write the 2-dimensional vectors as complex numbers $z_t := x_t^{(1)} + j x_t^{(2)}$, then RoPE encoding is just multiplication by a (real-valued!) parameter ω:

$$\mathrm{RoPE}\left(z_t, t\right) = e^{jt\omega} z_t$$

The benefit of RoPE is that the dot-product between two vectors depends on their relative location only:

$$\mathrm{RoPE}(x, m)^T \,\mathrm{RoPE}(y, n) = \mathrm{RoPE}(x, m+k)^T \,\mathrm{RoPE}(y, n+k)$$

for any integer k. This is obvious if we look at RoPE in the complex number form.

Extend RoPE to work for $2n$-dimensional vectors.

(d) **Does adding RoPE change the number of learnable parameters in the model? If so, how many new parameters are added (compared to a similar model with no positional encoding)?**

3. NoPE (No Positional Encoding)

In models with a decoder-only architecture (such as GPT) using causal attention, it might be possible for the attention layer itself to learn to encode positional information.

Suppose we have a sequence of tokens $\{< beg >, x_2, \ldots, x_n\}$ where $< beg >$ is a special token indicating the beginning of the sequence. Additionally, suppose we learn an embedding where $< beg >$ is represented by d-dimensional vector $[1, 1, e_{b,3}, \ldots, e_{b,d}]^T$ and each other token x_i is represented by $[1, 0, e_{i,3}, \ldots, e_{i,d}]^T$ for $i = 1, 2, \ldots, n$. Here, $e_{b,j}$ and $e_{i,j}$ can be arbitrary when $j \geq 3$.

Additionally, suppose we have an attention head with query, key, value, and output matrices $W^Q \in \mathbb{R}^{d\times d}$, $W^K \in \mathbb{R}^{d\times d}$, $W^V \in \mathbb{R}^{d\times d}$ with the following learned values:

$$W^K = \begin{bmatrix} 1 & 1 & \cdots & 1 \\ 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 \end{bmatrix}, \quad W^V = \begin{bmatrix} 0 & 0 & \cdots & 0 \\ 1 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 \end{bmatrix}$$

and W^Q is arbitrary.

Using the convention that weight matrices are multiplied on the right of the token embeddings, **show that the output of the attention head for token x_t encodes the position t, and explain how this can pass positional information to subsequent layers in the model.**

4. Transformer Architecture

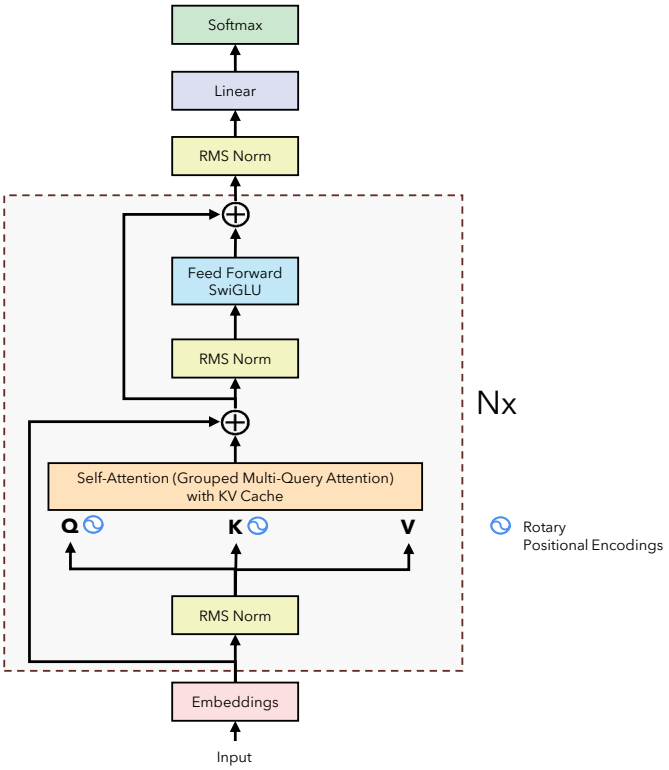

Figure 2: LLaMA architecture diagram. Many modern transformers use rotary positional embeddings (RoPE) and SwiGLU activation function (source: https://github.com/hkproj/pytorch-llama-notes/).

(a) **What is the advantage of multi-headed attention?** Give some examples of linguistic structures that can be found using multi-headed attention (e.g. subject-verb agreement, translation between languages).

(b) Let's say we're using argmax attention, which uses argmax rather than softmax, like we saw previously. **What is the size of the receptive field of a node at level n...**
If we have only a single head?
If we have two heads?
If we have h heads?

(c) For input sequences of length M and output sequences of length N, **what are the complexities of (1) Non-Causal Self-Attention[1] (on the input sequence), (2) Non-Causal Self-Attention (on the input sequence), but with multi-query attention, (3) Cross Attention (from the output sequence to the input sequence), (4) Causal Self-Attention (on the output sequence).** Let d be the hidden dimension of the network (both the encoder and the decoder part), and h be the number of heads.

Recall: Normal multi-headed attention is:

$$\text{MultiHeadedAttention}(Q,K,V) = \text{Concat}_{i\in[\#\text{ heads}]}\left(\text{Attention}\left(XW_i^Q, XW_i^K, XW_i^V\right)\right)W^O$$

For this question, we assume W_i^Q, W_i^K, W_i^V are of shape $d \times d$ and X is shape $M \times d$ where each row of X is a token embedding. We also assume that we use the naive matrix multiplication, that is, if A, B are of shape $m \times n, n \times k$, then AB takes mnk multiply-accumulate operations.

With multi-query attention, there is just one W^K, W^V, thus:

$$\text{MultiQueryAttention}(Q,K,V) = \text{Concat}_{i\in[\#\text{ heads}]}\left(\text{Attention}\left(XW_i^Q, XW^K, XW^V\right)\right)W^O$$

(d) **How many operations does the multi-query attention save compared to multi-head attention? What other some other advantages?**

Contributors:

- Kumar Krishna Agrawal.
- Kevin Li.
- Anant Sahai.
- Olivia Watkins.
- Jerome Quenum.
- Saagar Sanghavi.
- Yuxi Liu.
- Qiyang Li.
- CS 182/282A Staff from previous semesters.
- Lance Mathias.

[1] This is the "normal" self-attention from lecture.

