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.

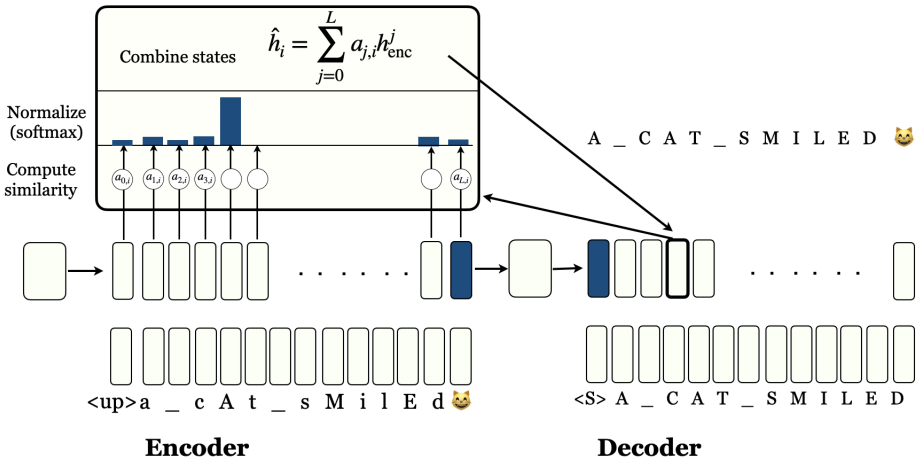

Figure 1: Attention Mechanism for Sequence Modelling with RNNs.

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?**

Solution: We can add cross-attention between the encoder and decoder by taking the hidden states from the encoder as keys and values, and the hidden states from the decoder as queries.

(a) The encoder weights need to learn a representation of the position of a particular token in the input-sequence.

(b) The attention scores computes similarity between the decoder "query" vector and the hidden-states of the encoder. As long as it scores the token at the same index as the query vector, it can be used to perform the task.

(c) The bottleneck activation no-longer needs to store information about the entire input sequence, since we are allowed to perform a look-up with the attention scores.

(d) The decoder weights learn to count, that is used to identify which token in the output-sequence we are decoding.

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.**

Solution: Position encoding is used to ensure that word position is known. Because attention is applied symmetrically to all input vectors from the layer below, there is no way for the network to know which positions were filtered through to the output of the attention block.

Position encoding also allows the network to compare words (nearby position encodings have high inner product) and find nearby words. It is necessary in language translation tasks, where the order of the words affects the meaning. For example, "the man chased the dog" and "the dog chased the man" have very different meanings.

Note: Early transformer implementations used absolute PE: for each position in the context length (C), a unique positional encoding vector was either learned or generated using a fixed function (like sine/cosine). Absolute PE is not used much anymore because relative PE (like RoPE) tends to better allow for context length generalization.

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

Solution: Relative positional embeddings are designed such that the embedding values only depend on the relative position of where the element that you are attending to (rather than the absolute position as ussed in the absollute PE). The easiest way to implement such embeddings is to simply shift the absolute embedding for the elements you attend to such that your current position always encodes a fixed position. Another very clever way to do relative positional embedding is Rotary Positional Encoding (RoPE), which we will talk about in the next subpart.

(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

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

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

$$\text{RoPE}\,(z_t, t) = e^{jt\omega} z_t$$

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

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

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.

Solution: Given $2n$-dimensional vector $(x^{(1)}, x^{(2)}, ..., x^{(2n)})$, represent it as a n-dimensional complex vector

$$(z^{(1)}, z^{(2)}, ..., z^{(n)}) := (x^{(1)} + jx^{(2)}, ..., x^{(2n-1)} + jx^{(2n)})$$

Then, we have

$$\text{RoPE}\left((z^{(1)}, z^{(2)}, ..., z^{(n)}), t\right) = \left(e^{jt\omega^{(1)}} z^{(1)}, e^{jt\omega^{(2)}} z^{(2)}, ..., e^{jt\omega^{(n)}} z^{(n)}\right)^T$$

or equivalently,

$$\text{RoPE}\left((x^{(1)}, x^{(2)}, ..., x^{(2n)}), t\right) = \begin{pmatrix} x^{(1)}\cos t\omega^{(1)} - x^{(2)}\sin t\omega^{(1)} \\ x^{(2)}\cos t\omega^{(1)} + x^{(1)}\sin t\omega^{(1)} \\ x^{(3)}\cos t\omega^{(2)} - x^{(4)}\sin t\omega^{(2)} \\ x^{(4)}\cos t\omega^{(2)} + x^{(3)}\sin t\omega^{(2)} \\ \vdots \\ x^{(2n-1)}\cos t\omega^{(n)} - x^{(2n)}\sin t\omega^{(n)} \\ x^{(2n)}\cos t\omega^{(n)} + x^{(2n-1)}\sin t\omega^{(n)} \end{pmatrix}$$

(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)? Solution:** RoPE does not add any learnable parameters to the model since the parameters ω are fixed. One rule of thumb is to set $\omega^{(i)} = 10000^{-2(i-1)/d}$ where d is the dimension of the input vector.

More modern models will employ additional techniques such as dynamically scaling ω depending on context length or interleaving RoPE with NoPE (as in LLaMA-4). This has the advantage of improving model performance on sequences longer than the training context length. Note that once again, these techniques do not add any additional learnable parameters.

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.**

Solution: Given the token sequence $\{< beg >, x_2, \ldots, x_n\}$, the learned embeddings are given by $X \in \mathbb{R}^{n\times d}$ where each row corresponds to the embedding of a token:

$$X = \begin{bmatrix} 1 & 1 & e_{b,3} & \cdots & e_{b,d} \\ 1 & 0 & e_{2,3} & \cdots & e_{2,d} \\ 1 & 0 & e_{3,3} & \cdots & e_{3,d} \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & 0 & e_{n,3} & \cdots & e_{n,d} \end{bmatrix}.$$

Applying the key, query, and value matrices yields $K \in \mathbb{R}^{n\times d}$, $Q \in \mathbb{R}^{n\times d}$, and $V \in \mathbb{R}^{n\times d}$ where

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

Then our (unnormalized) causal attention scores are given by

$$\alpha = \text{Mask}(QK^T) = \text{Mask}\left(\begin{bmatrix} s_1 & s_1 & s_1 & \cdots & s_1 \\ s_2 & s_2 & s_2 & \cdots & s_2 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ s_n & s_n & s_n & \cdots & s_n \end{bmatrix}\right) = \begin{bmatrix} s_1 & -\infty & -\infty & \cdots & -\infty \\ s_2 & s_2 & -\infty & \cdots & -\infty \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ s_n & s_n & s_n & \cdots & s_n \end{bmatrix},$$

where $s_t = \sum_{i=1}^{d} Q_{t,i}$ and $\text{Mask}(\cdot)$ applies the causal mask.

after normalization and applying the softmax to each row, we get the attention weights

$$\text{Softmax}(\alpha) = \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ \frac{1}{2} & \frac{1}{2} & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \frac{1}{n} & \frac{1}{n} & \frac{1}{n} & \cdots & \frac{1}{n} \end{bmatrix}.$$

Finally, the output of the attention head is given by

$$\begin{aligned}\text{Attention}(Q,K,V) &= \text{Softmax}(\alpha)V \\ &= \begin{bmatrix} 1 & 0 & 0 & \cdots & 0 \\ \frac{1}{2} & \frac{1}{2} & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \frac{1}{n} & \frac{1}{n} & \frac{1}{n} & \cdots & \frac{1}{n} \end{bmatrix} \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 \end{bmatrix} \\ &= \begin{bmatrix} 1 & 0 & \cdots & 0 \\ \frac{1}{2} & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ \frac{1}{n} & 0 & \cdots & 0 \end{bmatrix}.\end{aligned}$$

Each row of this output corresponds to one token; the output for token x_t is given by the vector $[\frac{1}{t}, 0, \ldots, 0]^T$, which encodes the token's position t. The output of this attention head can be passed to subsequent layers in the model, allowing us to learn arbitrary positional encodings as a function of the position t. This can be advantageous because strategies like RoPE can fail on context lengths if the parameters ω are not set appropriately.

Note that this is just an *existence* argument showing that it is theoretically possible for the attention layer to learn positional information; in practice, models actually tend to learn some variant of relative positional encoding.

4. Transformer Architecture

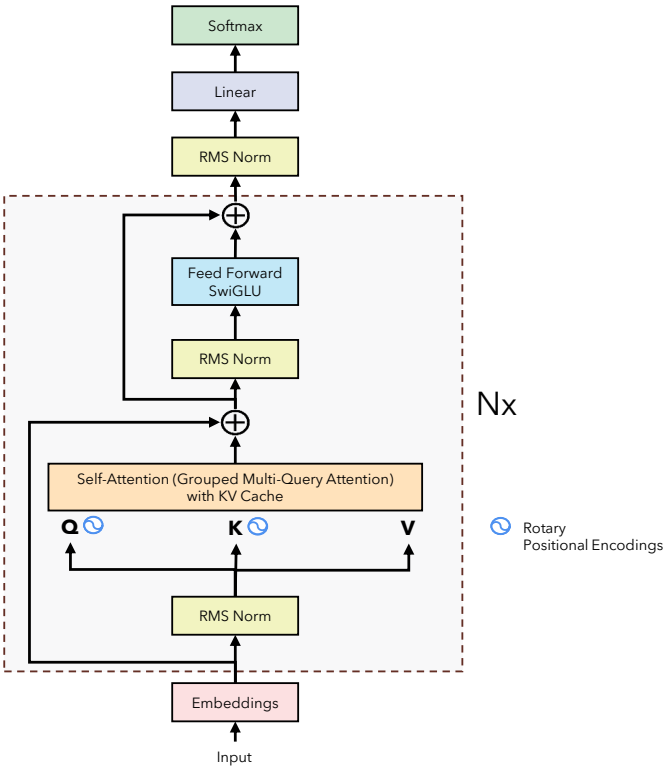

Figure 2: LLaMA architecture diagram. Many modern transformers use rotary positional embeddings (RoPE) and SwiGLU activation function (source: https://github.com/hkproj/pytorch-llama-notes/).

(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).

Solution: Multi-Head attention allows for a single attention module to attend to multiple parts of an input sequence. This is useful when the output is dependent on multiple inputs (such as in the case of the tense of a verb in translation). Attention heads find features like start of sentence and paragraph, subject/object relations, pronouns, etc.

In particular, when translating between languages where the tense and plurality of verbs and nouns can depend on multiple other words in the sentence. To maintain grammatical correctness, the model must be able to attend to all of these words in the input sequence. Another example is translating from English to a language like Chinese where several English words may map to a single Chinese character. Here, the model must attend to multiple input words to generate the correct output character.

(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?

Solution: With only a single head, we only have attention with one other time step (ie. the key vector), so with the residual connection in the transformer block, a branching factor of 2 at each level. Hence total size is 2^n.

With two heads, each hidden state can pay attention to itself and two other hidden states, so we have a branching factor of 3. Total size of receptive field is 3^n.

Similarly, with h heads, size of the receptive field is $(h+1)^n$

(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:

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

Solution:

i. $\mathcal{O}((Md + M^2)dh)$
ii. $\mathcal{O}((Md + M^2)dh)$
iii. $\mathcal{O}((Md + Nd + MN)hd)$
iv. $\mathcal{O}((Nd + N^2)dh)$

As one example, the encoder-self-attention takes $3Md^2$ operations for each XW_i^Q, XW_i^K, XW_i^V. Computing the attention matrix takes M^2d operations, and $O(M^2)$ for computing its softmax. Multiplying the attention with value matrix takes M^2d operations. This is repeated for h heads. Finally the outputs of all the heads are linearly projected by another hd^2 operations. Together:

$$\mathcal{O}((3Md^2 + M^2d + M^2 + M^2d)h + hd^2) = \mathcal{O}(Md^2h + M^2dh)$$

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

Solution: In multi-query attention, XW^K and XW^V are computed only once instead of h times. For each of these computations, we compute only one matrix multiplication instead of h, saving $2Md^2(h-1)$ operations in total.

Another advantage during inference time is that we only need to store one set of key and value projections instead of h sets, which reduces memory usage.

Contributors:

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

- 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.

