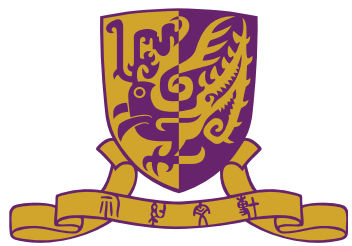
DDA5001 Machine Learning

Transformer (Part II)

Xiao Li

School of Data Science
The Chinese University of Hong Kong, Shenzhen



Information about Final

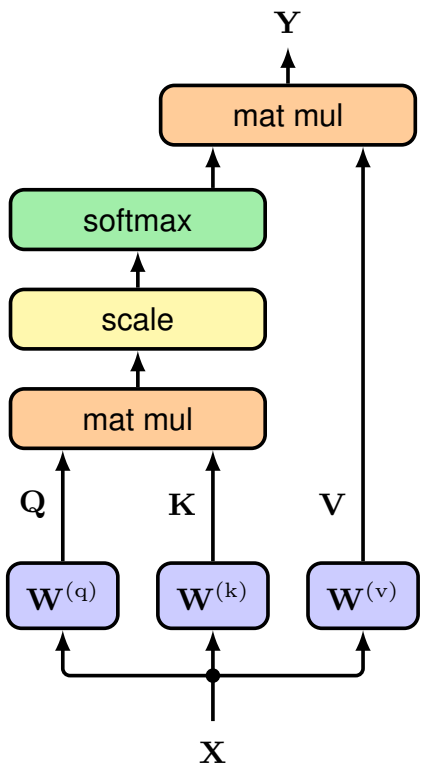
- Time: December 13 (Saturday), 3:00pm - 5:00pm (2 hours).
- Venue: Liwen Hall.
- A closed-book, closed-notes examination. No cheat sheet (this leads to an easier exam). No other things will be allowed, such as calculators.
- A sample exam will be available later to aid you in your review. Note that do not go ovefitting, you need a little generalization. Hence, you are highly recommended to go through our lecture slides and Homework questions.

Recap: Self-Attention

- Attention is to consider the corrections between different tokens in a sentence.
- We use self-attention to capture the semantic relationship:

$$\boldsymbol{Y} = \text{attention}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = \text{softmax}\left(\frac{\boldsymbol{Q}\boldsymbol{K}^\top}{\sqrt{d_k}}\right)\boldsymbol{V}.$$

Illustration:



Recap: Multi-Head Attention, Attention Masks

Multi-head attention: To capture different semantic structure by different attentions:

$$\boldsymbol{Y}(\boldsymbol{X}) = [\boldsymbol{H}_1, \ldots, \boldsymbol{H}_H]\boldsymbol{W}^{(o)}.$$

Each head $\boldsymbol{H}_i$ is generated by attention.

Attention masks:

- We need attention masks to block attention to padding tokens and future tokens (for causality during training):

$$\boldsymbol{Y} = \mathrm{softmax}\left(\frac{\boldsymbol{Q}\boldsymbol{K}^\top}{\sqrt{d_k}} + \boldsymbol{M}^{\text{causal}} + \boldsymbol{M}^{\text{pad}}\right)\boldsymbol{V},$$

where

$$\boldsymbol{M}_{ij} = \begin{cases} 0, & \text{allowed} \\ -\infty, & \text{blocked} \end{cases}$$

Complexity of Attention

Consider masked attention with n being sequence length.

- Embedding dimension d is often fixed, such as $d = 4096, 8192$.
- During pre-training, or reasoning modeling training, we often have long-sequence training, e.g., n can scale up to 128k. Therefore, we often focus on the scaling with n.

Complexity:

- Score matrix: $\boldsymbol{S} = \boldsymbol{Q}\boldsymbol{K}^\top \in \mathbb{R}^{n \times n}$ with computation $O(n^2 d)$.
- Masking + softmax on $\boldsymbol{S}$: Store and softmax $n \times n$ scores with computation $O(n^2)$ (softmax) and storing memory $O(n^2)$ (score matrix and mask).
- Weighted sum: $\boldsymbol{Y} = \mathrm{softmax}(\boldsymbol{S})\boldsymbol{V}$ with computation $O(n^2 d)$.

Observations:

- Overall, computation and memory are both quadratic in sequence length n.
- Attention is not in linear complexity. One of the major drawback of Transformers.

Flash-Attention

- Flash-attention is a technique to reduce storage memory from $O(n^2)$ to $O(nd)$, i.e., nearly linear in n when n is very large.

The main idea:

- Rewrites attention as a sequence of smaller block matrix operations.
- Never materializes the full $n \times n$ score matrix in GPU memory.
- Uses an online softmax algorithm to keep numerical stability while processing tiles.

Complexity of flash-attention:

- Compute: Still $\mathcal{O}(n^2 d)$ (same as standard attention).
- Memory: Reduced from $\mathcal{O}(n^2)$ to roughly $\mathcal{O}(nd)$.

For short sequences (i.e., small n), the benefit is modest. Flash-attention is very useful to reduce GPU memory when n is very large.

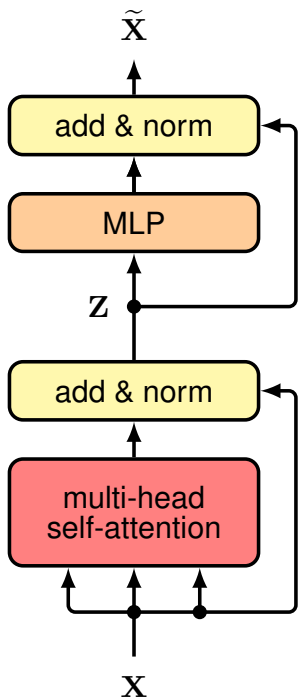
Transformer

Transformer Layer

Transformer Layer

- We now need to form a transformer layer.
- We want to add some residual connection to improve efficiency.
- We want to add normalization for efficiency and stability as well.
- We also want to add several MLP layer (feed-forward layer) to have more flexibility.

Overall, a transformer layer has the form:



- The mathematical modeling is:

$$\boldsymbol{Z} = \text{Norm}(\boldsymbol{Y}(\boldsymbol{X}) + \boldsymbol{X}).$$

$$\widetilde{\boldsymbol{X}} = \text{Norm}(\text{MLP}(\boldsymbol{Z}) + \boldsymbol{Z}).$$

- The dimension of the output $\widetilde{\boldsymbol{X}}$ need to be the same as that of the input $\boldsymbol{X}$.

Norm Method I: Layer Normalization (LayerNorm)

Consider $\boldsymbol{x} = (x_1, \ldots, x_d)$ as one data sample, like one row of $\boldsymbol{X}$ (i.e., one token). LayerNorm first does the following,

$$\hat{x}_i = \frac{x_i - \mu}{\sigma}, \quad \text{where} \quad \mu = \frac{1}{d}\sum_{i=1}^{d} x_i, \quad \sigma = \sqrt{\frac{1}{d}\sum_{i=1}^{d}(x_i - \mu)^2}$$

- The above process is to make the data zero mean and unit variance.

It then adds trainable parameters $\boldsymbol{\gamma} = (\gamma_1, \ldots, \gamma_d)$ and $\boldsymbol{\beta} = (\beta_1, \ldots, \beta_d)$ to scale ($\boldsymbol{\gamma}$) and shift ($\boldsymbol{\beta}$) the obtained $\hat{x}_i$:

$$y_i = \gamma_i \hat{x}_i + \beta_i, \quad i = 1, \ldots, d.$$

Finally, $\boldsymbol{y} = (y_1, \ldots, y_d)$ will be the layer normalized data sample.

- LayerNorm can be regarded as giving a new learned mean and variance for each data.
- LayerNorm is important to improving stability of the training process.

Norm Method II: RMSNorm

Another popular norm is Root Mean Squares Normalization (RMSNorm).

- First compute the root mean square (RMS)

$$\text{RMS}(\boldsymbol{x}) = \sqrt{\frac{1}{d}\sum_{i=1}^{d} x_i^2 + \varepsilon}.$$

  where ε is a small number for numerical stability.

- Then, apply normalization using RMS

$$\hat{x}_i = \frac{x_i}{\text{RMS}(\boldsymbol{x})}, \quad y_i = \gamma_i \hat{x}_i.$$

  γ_i is the trainable scaling factor.

Comparison: LayerNorm both scale and shift, while RMSNorm only scales.

- Older / classic LLMs (GPT-2, GPT-3) uses LayerNorm.
- More recent very large LLMs uses RMSNorm.

Post-Norm and Pre-Norm

The way we outlined of applying normalization is called post-norm:

$$\begin{aligned} \boldsymbol{Z} &= \text{Norm}(\boldsymbol{Y}(\boldsymbol{X}) + \boldsymbol{X}). \\ \widetilde{\boldsymbol{X}} &= \text{Norm}(\text{MLP}(\boldsymbol{Z}) + \boldsymbol{Z}). \end{aligned}$$

- It post-normalizes output after residual connection.
- This is the Norm used in the "Attention is All You Need" paper.

However, modern LLMs mainly use pre-norm which pre-normalize the input:

$$\begin{aligned} \boldsymbol{Z} &= \boldsymbol{Y}(\text{Norm}(\boldsymbol{X})) + \boldsymbol{X}. \\ \widetilde{\boldsymbol{X}} &= \text{MLP}(\text{Norm}(\boldsymbol{Z})) + \boldsymbol{Z}. \end{aligned}$$

As it is found that pre-norm provides a more stable training process.

Complexity of One Transformer Layer

Consider post-norm transformer layer:

$$\boldsymbol{Z} = \text{Norm}(\boldsymbol{Y}(\boldsymbol{X}) + \boldsymbol{X}), \qquad \widetilde{\boldsymbol{X}} = \text{Norm}(\text{MLP}(\boldsymbol{Z}) + \boldsymbol{Z}).$$

Notations. n: sequence length, d: model dimension, d_{MLP}: hidden size of MLP (often $d_{\text{MLP}} \approx 4d$).

- As we studied, the attention module has computation complexity $O(n^2 d)$ and memory complexity $O(n^2)$.

In MLP module:

- Computation complexity:

$$O\big(ndd_{\text{MLP}}\big) \quad \text{(two linear layers)}$$

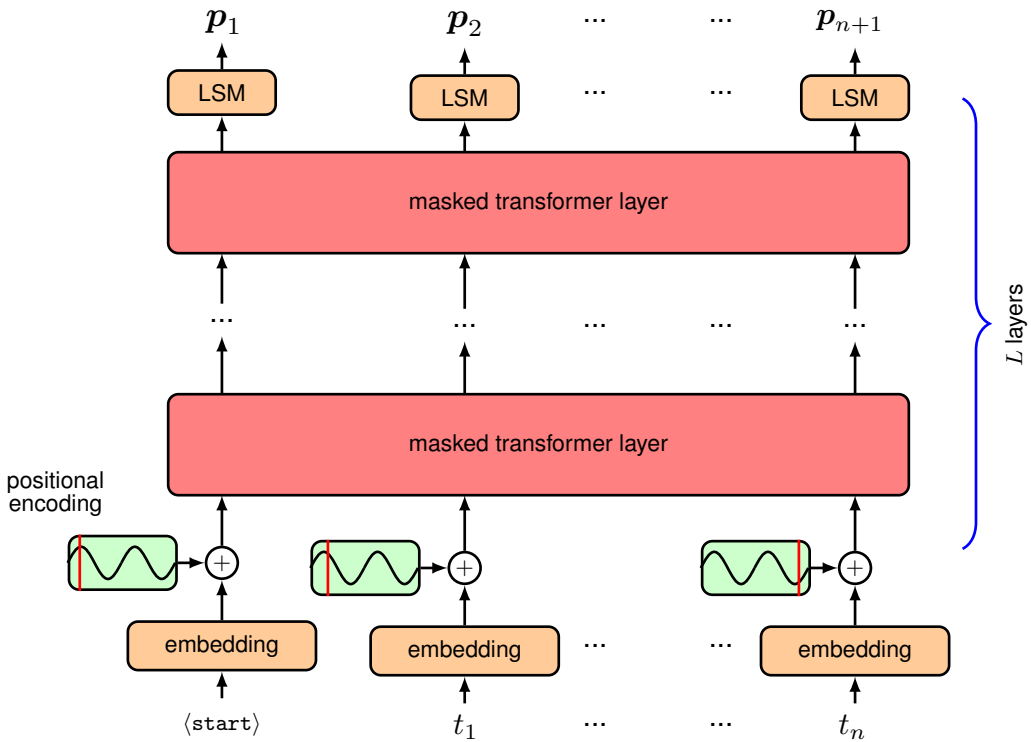
- Memory for activation values:

$$\mathcal{O}\big(nd_{\text{MLP}}\big) \quad \text{(hidden activations)}$$

So the overall order of complexity of the full Transformer layer is quadratic in n.

Transformer Architecture

Decoder Transformer Architecture



- This is decoder transformer, which is the architecture of GPT, Llama, and many other popular language models.
- Transformer is to stack transformer layers multiple times (just like one hidden layer to multiple), and add some other modules.

Tokenization

Goal: Map raw text to a sequence of discrete tokens

$$\text{“It is raining.”} \quad \longrightarrow \quad (t_1, t_2, \ldots, t_n).$$

- Is token in the unit of a character or a whole word? None of them.
  - Characters: very long sequences $\Rightarrow$ inefficient.
  - Words: huge vocabulary, many rare/unknown words.
- Modern LLMs use subword tokenization:
  - Methods: BPE (GPT2), SentencePiece (Qwen3), WordPiece, etc.
  - Common words are single tokens; rare words are split:

$$\text{“unbelievable”} \rightarrow \texttt{``un''}, \texttt{``believ''}, \texttt{``able''}$$

$$\text{“raining”} \rightarrow \text{“rain”}, \text{“ing”}$$

- Each token is represented internally by an integer id

$$t_i \in \{1, \ldots, V\},$$

where V is the vocabulary size (e.g., $V \approx 30\text{k} \sim 128\text{k}$).

Embedding Layer

- The model cannot work directly with token ids t_i; it needs vectors.
- Embedding matrix shared by all embedding positions:

$$\boldsymbol{E} \in \mathbb{R}^{V \times d}, \quad \text{row } \boldsymbol{E}_{t_i:} = \boldsymbol{x}_i^\top \in \mathbb{R}^{1 \times d}.$$

  Each token id t_i is mapped to an embedding vector $\boldsymbol{x}_i$:

$$t_i \longrightarrow \boldsymbol{x}_i = \boldsymbol{E}^\top \boldsymbol{e}_{t_i},$$

  where $\boldsymbol{e}_{t_i}$ is the one-hot vector of t_i, i.e., taking out the corresponding row of $\boldsymbol{E}$.
- Embedding matrix $\boldsymbol{E}$ is learned, rather than designed.
- After adding positional encodings (represent order of words in your sentence), the sequence

$$\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\}$$

  is fed into the stack of masked transformer layers shown earlier.

The Last Layer: Logistic Regression

- In the last LSM layer, it means linear transformation + softmax, i.e., multi-class logistic regression on the final learned feature $\boldsymbol{Z} \in \mathbb{R}^{(n+1)\times d}$, the added token is <start>.
- The number of class is V, i.e., the vocabulary size. Using classification to choose the next token from the vocabulary.
- The classification is done using logistic regression with lm_head $\boldsymbol{W}_{\text{lm}} \in \mathbb{R}^{d\times V}$, i.e., linear classification weight matrix.

Mathematically,

$$\boldsymbol{P} = \left[\boldsymbol{p}_1, \boldsymbol{p}_2, \ldots, \boldsymbol{p}_{n+1}\right]^\top = \text{softmax}(\boldsymbol{Z}\boldsymbol{W}_{\text{lm}}) = \text{softmax}(\boldsymbol{L}).$$

- $\boldsymbol{L} = \boldsymbol{Z}\boldsymbol{W}_{\text{lm}} = [\boldsymbol{l}_1, \ldots, \boldsymbol{l}_{n+1}]^\top \in \mathbb{R}^{(n+1)\times V}$ are called logits corresponding to each input token $\boldsymbol{x}_i$.
- $\boldsymbol{p}_i \in \mathbb{R}^V$ is the probability distribution over the vocabulary, for sampling the next token i, as we have shifted the position by adding the starting token <start>.

Interpretation: Transformer uses the new attention mechanism. Its overall idea is similar to other neural networks. Trying to extract useful features for linear classification.

Tied Embedding and LM Head

Tied parameters in LLMs:

- In decoder-only LMs, the same matrix (or its transpose) is often used for the input embedding and for the LM head matrix $\boldsymbol{W}_{\text{lm}}$ in the final logistic regression layer.

$$\boldsymbol{W}_{\text{lm}} = \boldsymbol{E} \quad (\text{or } \boldsymbol{W}_{\text{lm}} = \boldsymbol{E}^{\top}, \text{ depending on convention}).$$

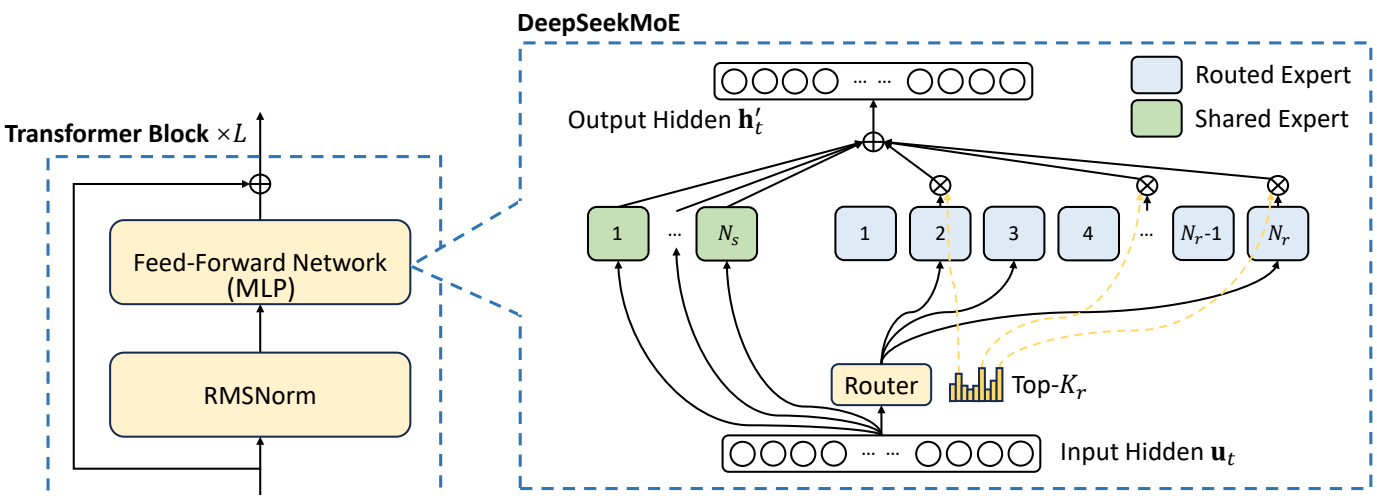
- This is called Tied LM head.
- it reduces parameters by $V \times d$. This is often a very large matrix.

This is reasonable as we use the same embedding to transfer tokens to embedding vectors, and then use the same matrix to transfer embedding back to tokens in the vocabulary.

Mixture of Experts (MoE)

Mixture-of-Experts (MoE) Feed-Forward Layer

- MoE is to expand the MLP / feed-forward network (FFN), not attention.
- In standard Transformers, each token goes through the same MLP layer.
- In a MoE MLP layer, we have multiple MLPs (experts) and a router that chooses a small subset of experts for each token.
- MoE is a simple architecture scaling approach that can expand the representation power of transformer (said by OpenAI people).



Mathematical Definition of MoE

- Let $\boldsymbol{u}_t \in \mathbb{R}^d$ be the hidden state / activation value of token $\boldsymbol{x}_t$ entering a MoE MLP. The router produces a sparse weighting over N_r experts:

$$\boldsymbol{g}_t = \text{Top-}K_r\big(\text{softmax}(\boldsymbol{R}\boldsymbol{u}_t)\big) \in \mathbb{R}^{N_r},$$

  where $\boldsymbol{R} \in \mathbb{R}^{N_r \times d}$ is the router matrix and K_r is the number of activated experts per token (e.g., $K_r = 2$).
- Only the top-K_r entries of $\boldsymbol{g}_t$ are kept, others are 0, i.e., only the top-K_r routed experts are chosen.

Overall, the MoE layer per token is mathematically defined as

$$\boldsymbol{h}'_t = \underbrace{\boldsymbol{u}_t}_{\text{skip connection}} + \underbrace{\sum_{i=1}^{N_s} \text{MLP}_i^{(s)}(\boldsymbol{u}_t)}_{\text{shared MLP}} + \underbrace{\sum_{i=1}^{N_r} \boldsymbol{g}_t[i] \cdot \text{MLP}_i^{(r)}(\boldsymbol{u}_t)}_{\text{routed expert}}.$$

$\rightsquigarrow$ Next lecture: Large language models (LLMs).