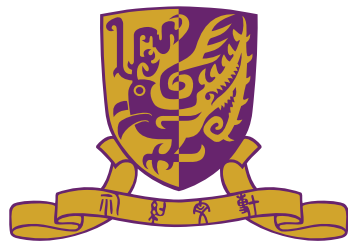
DDA5001 Machine Learning

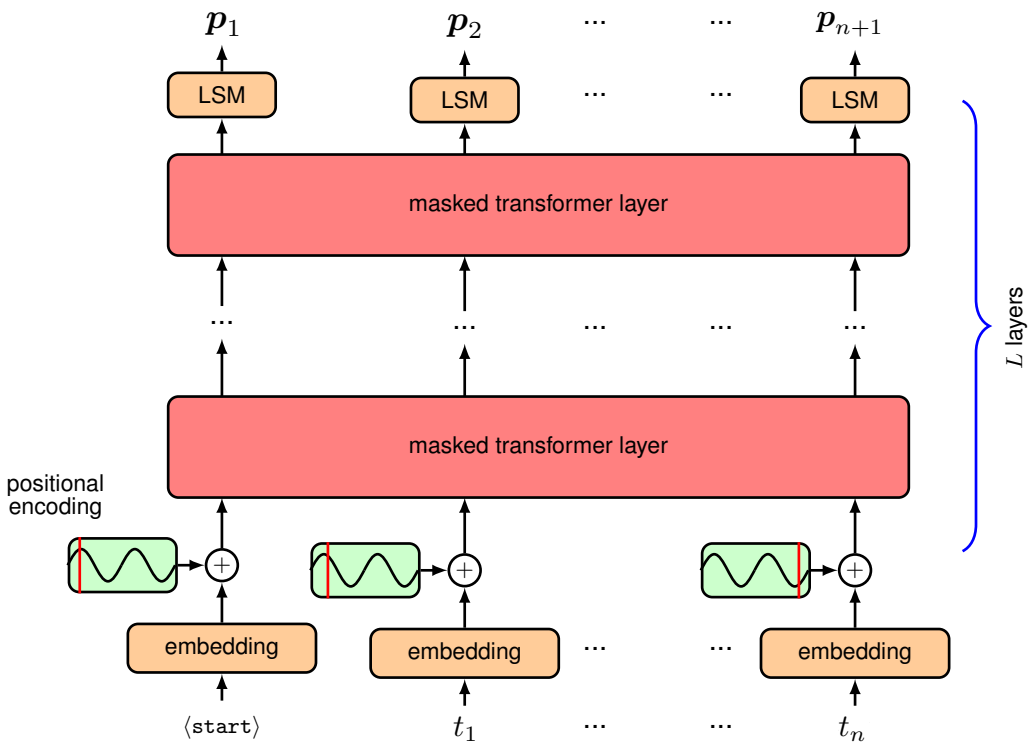
Large Language Models (Part I)

Xiao Li

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



Recap: Decoder Transformer



- Contemporary large language models (LLMs) are usually built on (decoder) transformer architecture.

Transformer — Continued

Autoregressive Modeling

Pre-training

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.

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

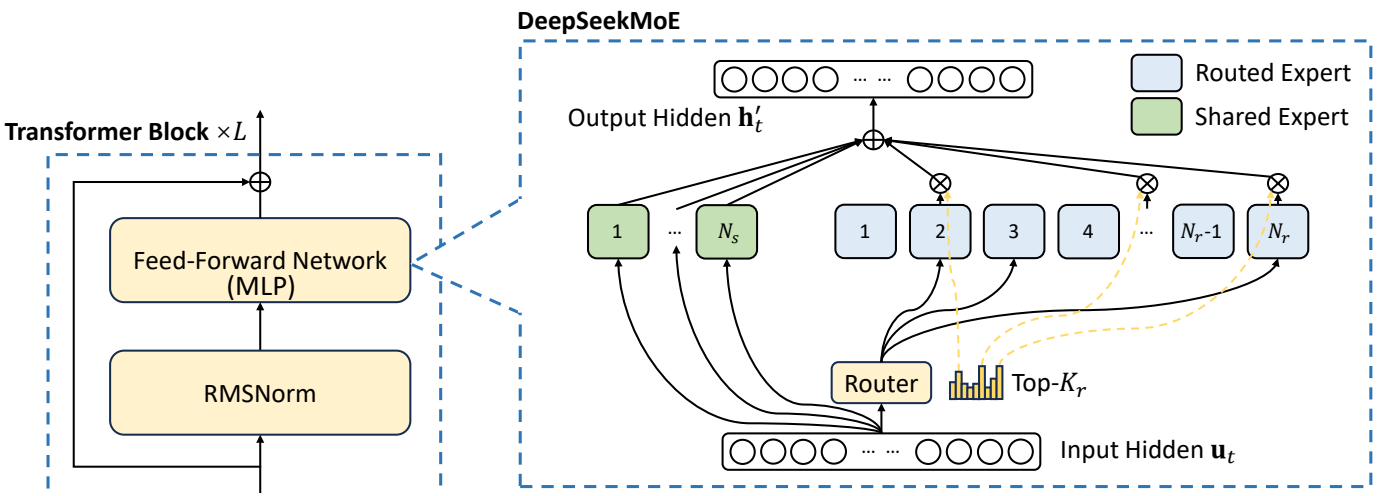
- This is called Tied LM head.
- it reduces parameters by $V \times d$. This is often a very large matrix.

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

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

Transformer — Continued

Autoregressive Modeling

Pre-training

Conditional Probability is All We Need

- Recall that classification can be transferred to finding conditional probability. This is the foundation of logistic regression.
- Language models are trained to predict the next token, which can also be modeled as maximizing conditional probability.

Suppose that our input prompt (i.e., input tokens) is $\boldsymbol{x}$ and the language model will generate the response $\boldsymbol{y} \in \mathbb{R}^m$ with probability

$$\mathbb{P}[\boldsymbol{y}|\boldsymbol{x}].$$

In LLMs, it models the conditional probability using a transformer with autoregressive probability decomposition:

$$\mathbb{P}_{\boldsymbol{\theta}}[\boldsymbol{y}|\boldsymbol{x}].$$

- $\mathbb{P}_{\boldsymbol{\theta}}$ is the transformer model, which predicts the conditional probability using the logistic regression in the final layer.
- $\boldsymbol{\theta}$ contains transformer's trainable parameters.

Autoregressive Modeling of Conditional Probability

In LLM, it models the conditional probability using a transformer with autoregressive probability decomposition:

$$\mathbb{P}_{\boldsymbol{\theta}}[\boldsymbol{y}|\boldsymbol{x}] = \prod_{j=1}^{m} \mathbb{P}_{\boldsymbol{\theta}}[y_j|\boldsymbol{x}, \boldsymbol{y}_{1:j-1}].$$

y_j is the j-th token in the output $\boldsymbol{y}$.

- The above decomposition is always true. It is a foundation of LLM.
- Generating the whole response $\boldsymbol{y}$ can be decomposed into autoregressively predicting the next token.

Self-Labeling During Training

Each $\boldsymbol{p}_i$ is for sampling the next token.

During training: The input is <start> plus $\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\}$ ($n+1$ tokens). The output is $\{\boldsymbol{p}_1, \ldots, \boldsymbol{p}_n, \boldsymbol{p}_{n+1}\}$, i.e., $n+1$ distributions.

- Distribution $\boldsymbol{p}_1$ should predict label $\boldsymbol{e}_{t_1}$ of token $\boldsymbol{x}_1$.
- ...
- Distribution $\boldsymbol{p}_n$ should predict label $\boldsymbol{e}_{t_n}$ of token $\boldsymbol{x}_n$.
- Distribution $\boldsymbol{p}_{n+1}$ should predict label $\boldsymbol{e}_{t_{n+1}}$ of the end of sequence <EOS> special token.

The idea of the above prediction is because $\boldsymbol{x}_{i+1}$ (i.e., $\boldsymbol{e}_{t_{i+1}}$) should be the true label at any input position $\boldsymbol{x}_{\leq i}$. Namely, the generated distribution $\boldsymbol{p}_{i+1}$ using input $\boldsymbol{x}_{\leq i}$ should predict the true label of $\boldsymbol{x}_{i+1}$.

- The true label $\boldsymbol{e}_{i+1}$ of $\boldsymbol{x}_{i+1}$ is represented as a one-hot encoding $(0, 0, \ldots, 1, \ldots, 0) \in \mathbb{R}^V$ with 1 at the token position $\boldsymbol{x}_{i+1}$ in the vocabulary.
- The model generated $\boldsymbol{p}_{i+1}$ is encouraged by training to be as close as possible to the above one-hot label of $\boldsymbol{x}_{i+1}$.
- This is self-labeling.

Pre-training Problem Formulation

- The learning problem of LLMs is just a maximum likelihood estimation (MLE).

That is, given a set of training data pairs $\mathcal{D} = \{\boldsymbol{x}_i\}$, the MLE learning problem of LLMs can be formulated as minimizing the negative log-likelihood of the generation probability:

$$\widehat{\boldsymbol{\theta}} \leftarrow \underset{\boldsymbol{\theta}}{\operatorname{argmin}} \ \mathcal{L}(\boldsymbol{\theta}) = \sum_{i \in \mathcal{D}} \sum_{j=1}^{m} -\log \left(\mathbb{P}_{\boldsymbol{\theta}}[x_{i,j} | \boldsymbol{x}_{i,1:j-1}] \right). \tag{P}$$

- This training formulation of autoregressive model is called self-supervised learning, using the later tokens in a sentence as supervision.
- At each position of data $\boldsymbol{x}_i$, the previous token $\boldsymbol{x}_{i,\leq j-1}$ is trained to predict the next j-th token $x_{i,j}$.
- Namely, the final layer's logistic regression forces the generated $\boldsymbol{p}_{j+1}$ is trained being close to $\boldsymbol{e}_{j+1}$.

Inference / Generating Answers

- After trained an LLM, the generation task is called inference.
- Inference is the forward passes of an LLM, i.e., generating the response $\boldsymbol{y}$ given input prompt $\boldsymbol{x}$.

Inference process of LLMs: We will only use the last classification to predict next token:

$$\text{softmax}(\boldsymbol{z}_{end}^{\top}\boldsymbol{W}^{L}),$$

where $\boldsymbol{z}_{end}$ is the learned feature of the last token of the input.

- We need to input all the tokens of the prompt even though we only use the last feature for classification, due to attention mechanism.
- After predicting the next token, we need to append it to the last prompt and then input to transformer to make prediction, until we finish generation.
- The generation is finished when we reach the preset maximum number of generation length, or it automatically generates the special end of sequence <EOS> token.

Sampling Strategies in Inference

- The generated probability distribution $\text{softmax}(\boldsymbol{z}_{end}^{\top}\boldsymbol{W}^{L})$ fully determines the next token to choose from the vocabulary.

Sampling strategies:

- Greedy: Choose the highest probability to be the next token. But this is not equivalent to max $\mathbb{P}_{\boldsymbol{\theta}}[\boldsymbol{y}|\boldsymbol{x}]$.
- Beam search: Predict B alternative sequences, and then choose the best sequence. This is different from single next token prediction, and is more accurate.

Transformer — Continued

Autoregressive Modeling

Pre-training

Pre-training Overview

Overview:

- Pre-training is to train an LLM from scratch by optimizing the learning objective (P) on a large corpus (i.e., training dataset).
- This looks like a standard training procedure. However, it contains so many non-trivial engineering steps, huge computing resources, and man power resources due to truly large-scale feature.

We quickly go through: data, model architecture, scaling law, training setting for pre-training.

We will take Llama 3 developed by Meta as example.

Pre-training Dataset

Llama 3's pre-training uses 15T carefully crafted data from the internet.

The processing of data includes:

- Safety filtering.
- Text extraction and cleaning: extract high quality text from HTML.
- De-duplication.
- Quality filtering.
- Code and math reasoning data.
- Multilingual data: 176 languages.

Data mix:

- 50% general knowledge tokens.
- 25% mathematical and reasoning tokens.
- 17% code tokens.
- 8% multilingual tokens.

Model Architecture

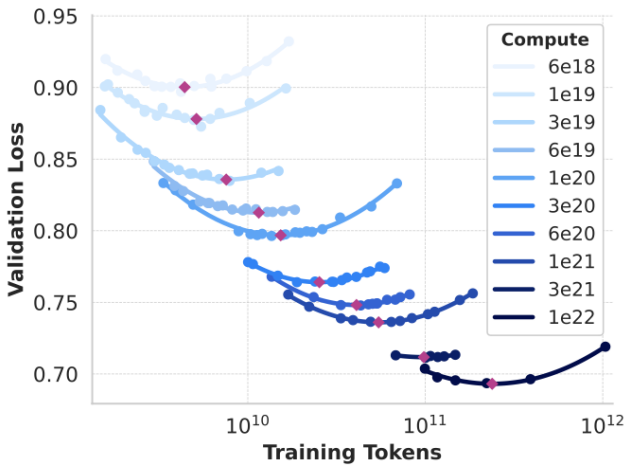
- Llama 3 uses a standard, dense (decoder) Transformer architecture.
- Use group query attention with 8 key-value heads to accelerate inference speed.
- Vocabulary is of size 128K, i.e., V in the final layer logistic regression classifier has dimension 128K.
- Llama 3 has 8B, 70B, 405B models (trainable parameters of the transformer).

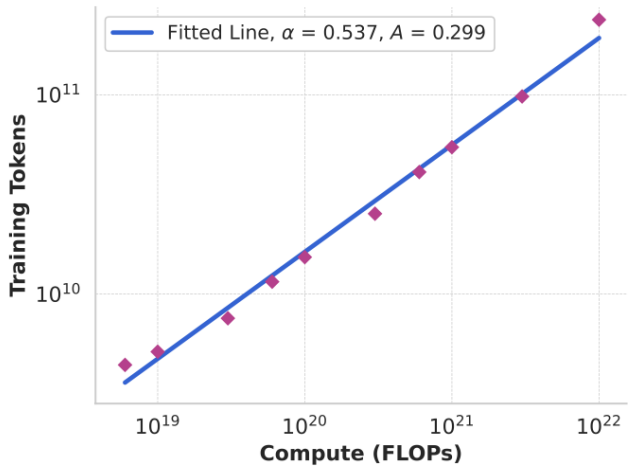
Overview of the key hyperparameters of Llama 3:

	8B	70B	405B
Layers	32	80	126
Model Dimension	4,096	8192	16,384
FFN Dimension	6,144	12,288	20,480
Attention Heads	32	64	128
Key/Value Heads	8	8	8
Peak Learning Rate	3×10^{-4}	1.5×10^{-4}	8×10^{-5}
Activation Function	SwiGLU		
Vocabulary Size	128,000		
Positional Embeddings	RoPE ($\theta = 500,000$)		

Scaling Law

- Scaling law is to predict the training loss of (P) based on training tokens and model size trade-off, given a fixed compute budget (measured by FLOPS).





- The left figure uses quadratic function to do fitting.
- The right figure takes the red compute-optimal points in left to fit a power law relationship:

$$N^*(C) = A \cdot C^{\alpha}.$$

Here, C is compute FLOPS (x-axis) and N^* is the optimal training tokens (y-axis). This also indicates the model size.

Training Recipe

Llama 3-405B is pre-trained using

- AdamW optimizer.
- 8×10^{-5} initial lr, and use cosine decay to 8×10^{-7}.
- It uses an increasing batch size to stabilize training process. The batch size for computing the stochastic gradient is starting from 4M, to 8M, and finally to 16M.
- It also increases the sequence length n, starting from 4096 to final 128K sequence length / context window, in order to learn long context understanding.

$\rightsquigarrow$ Next lecture: Post-training and reasoning models.