Transformer and Large Language Models

Instructor: Lei Wu [1]

Mathematical Introduction to Machine Learning

Peking University, Fall 2024

[1]School of Mathematical Sciences; Center for Machine Learning Research

Transformer

Transformers

- were introduced in *Attention is all you need* (Vaswani et al., NeurIPS 2017);
- have revolutionized NLP, CV, robotics and many applications;
- have enabled the creation of powerful LLMs such as GPT-4;
- hold the promise of unlocking the potential for AGI (artificial general intelligence).

Sequence Modeling

Consider a simple block for sequence modeling:

$$X := (\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n) \overset{\mathcal{T}}{\mapsto} Y := (\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_n).$$

In practical models, we may compose the $\mathcal{T}$-type transforms for many times (aka layers).

Sequence Modeling

Consider a simple block for sequence modeling:

$$X := (\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n) \overset{\mathcal{T}}{\mapsto} Y := (\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_n).$$

In practical models, we may compose the $\mathcal{T}$-type transforms for many times (aka layers).

- Recurrence

$$\mathbf{y}_i = f(\mathbf{x}_i, \mathbf{y}_{i-1}).$$

Sequence Modeling

Consider a simple block for sequence modeling:

$$X := (\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n) \overset{\mathcal{T}}{\mapsto} Y := (\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_n).$$

In practical models, we may compose the $\mathcal{T}$-type transforms for many times (aka layers).

- Recurrence

$$\mathbf{y}_i = f(\mathbf{x}_i, \mathbf{y}_{i-1}).$$

- Convolution

$$\mathbf{y}_i = f(\mathbf{x}_{i-1}, \mathbf{x}_i, \mathbf{x}_{i+1}).$$

Sequence Modeling

Consider a simple block for sequence modeling:

$$X := (\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n) \overset{\mathcal{T}}{\mapsto} Y := (\mathbf{y}_1, \mathbf{y}_2, \dots, \mathbf{y}_n).$$

In practical models, we may compose the $\mathcal{T}$-type transforms for many times (aka layers).

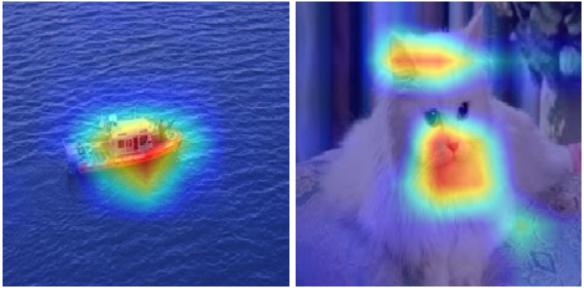
- Recurrence
$$\mathbf{y}_i = f(\mathbf{x}_i, \mathbf{y}_{i-1}).$$
- Convolution
$$\mathbf{y}_i = f(\mathbf{x}_{i-1}, \mathbf{x}_i, \mathbf{x}_{i+1}).$$
- Attention (simplified, **adaptive/selective weighted average**):
$$\mathbf{y}_i = f\left(\sum_{j=1}^{n} w_{i,j}(X)\mathbf{x}_j\right),$$
where $W(X) = (w_{i,j}(X)) \in \mathbb{R}^{n\times n}$ satisfies $\sum_{j=1}^{n} w_{i,j}(X) = 1$.

Attention Mechanism (Cont'd)

We often call $w_{i,j}(X)$'s the **attention score** and we want

the attention scores $(w_{i,1}(X), w_{i,2}(X), \cdots, w_{i,n}(X))$ to be **sparse** (i.e., **selective**).

- Attention in vision modeling:



Attention Mechanism (Cont'd)

Attention in machine translation (cross attention) [2]:

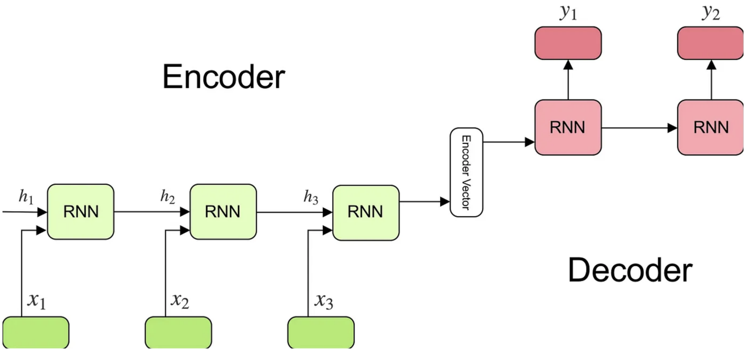


Figure 1: See a better animation in this link.

[2] Bahdanau et al., Neural Machine Translation by Jointly Learning to Align and Translate, ICLR 2015.

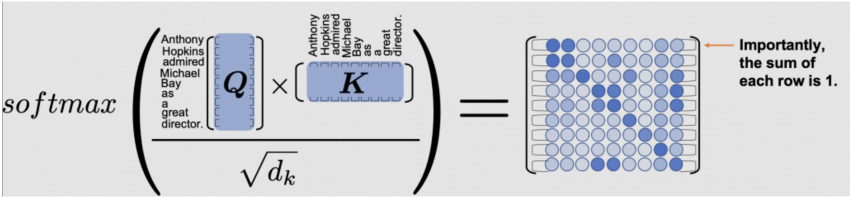
Self-Attention via Dot-Product

- Let $X = (\mathbf{x}_1, \dots, \mathbf{x}_n) \in \mathbb{R}^{d \times n}$ be our input sequence. We often call $\{\mathbf{x}_i\}$ **tokens**.
- A self-attention $\mathbb{A} : \mathbb{R}^{d \times n} \mapsto \mathbb{R}^{n \times n}$ outputs an attention-score map $P = \mathbb{A}(X)$. The most popular choice is

$$\mathbb{A}_{W_K, W_Q}(X) = \sigma\left(\frac{1}{\sqrt{d}}(W_K X)^\top (W_Q X)\right) \in \mathbb{R}^{n \times n},$$

where

  - $W_K, W_Q \in \mathbb{R}^{d_{\text{key}} \times d}$ are the key and query weight matrices, which are learned from data.
  - σ denotes the softmax normalization performed in a column-wise manner, ensuring the column represent a selective average.



Self-Attention via Dot-Product (Cont'd)

- The dot-products are implemented in a **token-wise manner** (can be naively paralleled):

$$\mathbf{k}_i = W_K \mathbf{x}_i, \mathbf{q}_j = W_Q \mathbf{x}_j \quad \text{for } i, j \in [n]$$

$$(\mathbb{A}_{W_K, W_Q}(X))_{i,j} = \frac{e^{\mathbf{k}_i^\top \mathbf{q}_j}}{\sum_{i'=1}^{n} e^{\mathbf{k}_{i'}^\top \mathbf{q}_j}}$$

- The attention scores are determined by the **dot-product correlation** among tokens. In principle, one can also propose other alternatives.
- A single-head attention layer $\texttt{SA} : \mathbb{R}^{d \times n} \mapsto \mathbb{R}^{d \times n}$ is given as follows

$$\texttt{SA}_{W_K, W_Q, W_V}(X) = V\sigma(QK),$$

where Q, K, V are called the query, key, value matrices, respectively and given by

$$Q = W_Q X, \quad K = W_K X, \quad V = W_V X.$$

A Transformer Block

- A transformer block defines a sequence-to-sequence map

$$X = (\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n) \in \mathbb{R}^{d \times n} \mapsto Y = (\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_n) \in \mathbb{R}^{d \times n}.$$

- This maps consists of two blocks:

$$Y = \mathbb{F}(X + \mathtt{MHA}(X)),$$

where

  - **Multi-head attention** (MHA)

$$\mathtt{MHA}(X) := \sum_{h=1}^{H} W_O^h \mathtt{SA}^h(X).$$

  - **Tokenwise** **feed-forward networks** (FFN):

$$\mathbb{F}(Z) := (h(\mathbf{z}_1), h(\mathbf{z}_2), \ldots, h(\mathbf{z}_n)) \in \mathbb{R}^{d \times n}.$$

In practice, $h : \mathbb{R}^d \mapsto \mathbb{R}^d$ is often chosen to be a two-layer MLP with hidden size d_{FF}.

$$h(\mathbf{z}) = W_1^\top \mathrm{ReLU}(W_2 \mathbf{z} + \mathbf{b}),$$

where $W_1, W_2 \in \mathbb{R}^{d_{\mathrm{FF}} \times d}$ and $\mathbf{b} \in \mathbb{R}^d$.

Transformer

- **Input:** Linear embedding to change the dimension of each token.

$$X^{(0)} = VX \text{ with } V \in \mathbb{R}^{d_{\text{model}} \times d}.$$

- **Main block**:

$$X^{\ell} = \mathbb{F}^{(\ell)}(X^{(\ell-1)} + \mathtt{MHA}^{(\ell)}(X^{(\ell-1)})), \quad 1 \leq \ell \leq L.$$

- **Ouput:** The output format depends on the tasks. In classification, we may

$$f(X) = p(\mathbf{x}_1^{(L)}),$$

  where p can be either a linear layer or small MLP.

- Architecure hyperparameters: d_{model}, H, L, d_{key} , d_{FF}. In practice, a common choice $d_{\text{FF}} = 4d_{\text{model}}, d_{\text{key}} = d_{\text{model}}/H$.

Absolute Positional Embedding (APE)

Transformers are still inherently **permutationally invariant** and we need to modify transformers by injecting position information.

The most natural way of injecting position information is using **absolute positional embedding** (APE): let $\mathbf{r}_i \in \mathbb{R}^d$ denote the information for token i:

$$\mathbf{x}_i \to \mathbf{x}_i + \mathbf{r}_i,$$

- Learnable APE: $\mathbf{r}_i$ are parameters to be learned.
- One-hot APE: $\mathbf{r}_i = \mathbf{e}_i$ where $\mathbf{e}_i$ is the one-hot label with 1 in the i-th coordinates and zero else.
- Sinusoidal APE:

$$\mathbf{r}_i = \Big(\sin(i), \cos(i), \sin(i/c), \cos(i/c), \ldots, \sin(i/c^{2i/d}), \cos(i/c^{2i/d})\Big) \in \mathbb{R}^d,$$

  where c is constant, e.g. 1000.

Absolute Positional Embedding (APE)

Transformers are still inherently **permutationally invariant** and we need to modify transformers by injecting position information.

The most natural way of injecting position information is using **absolute positional embedding** (APE): let $\mathbf{r}_i \in \mathbb{R}^d$ denote the information for token i:

$$\mathbf{x}_i \to \mathbf{x}_i + \mathbf{r}_i,$$

- Learnable APE: $\mathbf{r}_i$ are parameters to be learned.
- One-hot APE: $\mathbf{r}_i = \mathbf{e}_i$ where $\mathbf{e}_i$ is the one-hot label with 1 in the i-th coordinates and zero else.
- Sinusoidal APE:

$$\mathbf{r}_i = \left(\sin(i), \cos(i), \sin(i/c), \cos(i/c), \ldots, \sin(i/c^{2i/d}), \cos(i/c^{2i/d})\right) \in \mathbb{R}^d,$$

  where c is constant, e.g. 1000.

APE is rarely used in practice anymore due to:

- APE can not handle input sequence longer than that used in training.
- In many real problems, it is "relative distance" matters.

Relative Positional Embedding (RPE)

- **Additive RPE**: Let $E = (W_K X)^\top (W_Q X) \in \mathbb{R}^{n\times n}$ be the pre-softmax attention weights. Then, we inject relative position information by

$$\mathbb{A}(X) = \sigma(E - P),$$

where $P = (h(j-i))_{i,j} \in \mathbb{R}^{n\times n}$.

- In T5 RPE chooses

$$h(t) = \begin{cases} |t| & \text{if } |t| \le B/2 \\ \frac{B}{2} + \frac{B}{2}\left\lfloor \frac{\log(\frac{|t|}{B/2})}{\log(\frac{D}{B/2})} \right\rfloor & \text{if } \frac{B}{2} \le |t| \le D \\ B-1 & \text{if } |t| \ge D \end{cases}$$

In Alibi RPE, $h(t) = -\alpha|t| + \beta$. Where the α and β can be either learnable or fixed.

- Currently, the most popular one is the rotary positional embedding (RoPE), which has been adopted in nearly all LLMs.

Tokenization in NLP for Transformers

Before we do one-hot embedding, we need to tokenize natural language.

- **Definition:** Converting text into tokens (small units) before feeding it into a model.
- **Purpose:** Makes the text interpretable for the model, facilitating further processing like embedding and sequence modeling.

Tokenization in NLP for Transformers

Before we do one-hot embedding, we need to tokenize natural language.

- **Definition:** Converting text into tokens (small units) before feeding it into a model.
- **Purpose:** Makes the text interpretable for the model, facilitating further processing like embedding and sequence modeling.

Types of Tokenization

- **Word-level Tokenization:** Splits text into individual words.

  `"Transformers are amazing!"" -> ["Transformers", "are", "amazing", "!"]`

Tokenization in NLP for Transformers

Before we do one-hot embedding, we need to tokenize natural language.

- **Definition:** Converting text into tokens (small units) before feeding it into a model.
- **Purpose:** Makes the text interpretable for the model, facilitating further processing like embedding and sequence modeling.

Types of Tokenization

- **Word-level Tokenization:** Splits text into individual words.

  ```
  "Transformers are amazing!"" -> ["Transformers", "are", "amazing", "!"]
  ```

- **Subword-level Tokenization:** Breaks words into smaller pieces (subwords) for efficient handling of unknown words and morphemes. Popular in transformers.

  ```
  "Transformers are amazing!" ->
               ["Trans", "##form", "##ers", "are", "amaz", "##ing", "!"]
  ```

Tokenization in NLP for Transformers

Before we do one-hot embedding, we need to tokenize natural language.

- **Definition:** Converting text into tokens (small units) before feeding it into a model.
- **Purpose:** Makes the text interpretable for the model, facilitating further processing like embedding and sequence modeling.

Types of Tokenization

- **Word-level Tokenization:** Splits text into individual words.

```
"Transformers are amazing!"" -> ["Transformers", "are", "amazing", "!"]
```

- **Subword-level Tokenization:** Breaks words into smaller pieces (subwords) for efficient handling of unknown words and morphemes. Popular in transformers.

```
"Transformers are amazing!" ->
            ["Trans", "##form", "##ers", "are", "amaz", "##ing", "!"]
```

- Many other tokenizations. Libraries like NLTK, spaCy provide basic tokenization. `transformers` library by Hugging Face for transformer-specific tokenization.

Cost Analysis

$$\texttt{MHA}(X) = X + \sum_{h=1}^{H} W_O^h W_V^h X \texttt{SA}^h(X),$$
$$\mathbb{F}(\mathbf{x}) = W_1^\top \mathrm{ReLU}(W_2\mathbf{x} + \mathbf{b}).$$

In practice, it is often choose

$$d_{\text{key}} = d_{\text{model}}/H, \quad d_{\text{FF}} = 4d_{\text{model}}.$$

- **Storage:** $4d_{\text{model}}^2 + 8d_{\text{model}}^2$
- **Computation:**
  - MHA: $\underbrace{4nd_{\text{model}}^2}_{\text{Token-fea. extr.}} + \underbrace{d_{\text{model}}n^2}_{\textbf{attention}}$
  - FF: $8d_{\text{model}}^2 n$.
- Tokenwise operations can be parallelized. The total cost depends on the sequence length **qudratically**. This is especially bad for inference!!

Training Deep Transformers Need Many Tricks

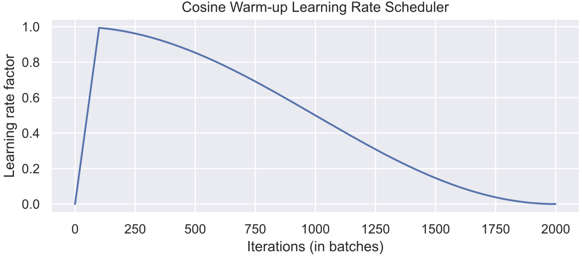
- Scaled dot-product attention

$$\mathbb{A}_{W_K,W_Q}(X) = \sigma\left(\frac{1}{\sqrt{d_k}}(W_K X)^\top (W_Q X)\right) \in \mathbb{R}^{n\times n},$$

- Layer normalization + residual connection:

$$\tilde{X}^{(\ell-1)} = \mathrm{LN}(X^{(\ell-1)})$$
$$X^{\ell} = \mathbb{F}\left(\tilde{X}^{(\ell-1)} + \mathtt{MHA}(\tilde{X}^{(\ell-1)})\right)$$

- AdamW optimizer with $(\beta_1 = 0.9, \beta_2 = 0.98)$ and gradient clipping.
- Learning rate warmup + Cosine decay.



Readings

- The original paper https://arxiv.org/abs/1706.03762
- Annotated Transformer https://jalammar.github.io/illustrated-transformer/
- Illustrated Transformer https://poloclub.github.io/transformer-explainer/

BERT

- Developed by Google.
- Bidirectional: Unlike traditional models that read text unidirectionally, BERT reads the entire sequence of words at once.
- Layers: Typically 12 layers (BERT Base) or 24 layers (BERT Large).

BERT

- Developed by Google.
- Bidirectional: Unlike traditional models that read text unidirectionally, BERT reads the entire sequence of words at once.
- Layers: Typically 12 layers (BERT Base) or 24 layers (BERT Large).

Pre-training Tasks:

- Masked Language Model (MLM): Randomly masks words in the sentence and predicts them.
- Next Sentence Prediction (NSP): Predicts if a given sentence logically follows another.

BERT

- Developed by Google.
- Bidirectional: Unlike traditional models that read text unidirectionally, BERT reads the entire sequence of words at once.
- Layers: Typically 12 layers (BERT Base) or 24 layers (BERT Large).

Pre-training Tasks:

- Masked Language Model (MLM): Randomly masks words in the sentence and predicts them.
- Next Sentence Prediction (NSP): Predicts if a given sentence logically follows another.

Fine-tuning: Adapts pre-trained BERT for various downstream tasks like question answering, sentiment analysis, etc.

GPT (Generative Pre-trained Transformer)

- Next-token prediction (autoregressive model):

$$\max_{\theta} \sum_{i=1}^{n} \log P_{\theta}(x_i | x_1, \ldots, x_{i-1}).$$

GPT (Generative Pre-trained Transformer)

- Next-token prediction (autoregressive model):

$$\max_{\theta} \sum_{i=1}^{n} \log P_{\theta}(x_i | x_1, \ldots, x_{i-1}).$$

- Text Generation:

text = $[\langle bos \rangle]$ or [some context]
while True:
  logit = decoder(embed(text))
  index = top(logit$[-1]$)
  token = vocabulary(index)
  if token == $\langle eos \rangle$:
    break
  text.append(token)
return text

Practice

- Pre-train models in large dataset. Fine-tune models on down-stream tasks.
- Fine-tuning needs to retrain our model, which is not user-friendly.
- Next-token prediction enables capability of doing **in-context learning**.

```
In the following lines, the symbol -> represents a simple mathematical operation.
100 + 200 -> 301
838 + 520 -> 1359
343 + 128 -> 472
647 + 471 -> 1119
64 + 138 -> 203
498 + 592 ->
```

Answer:

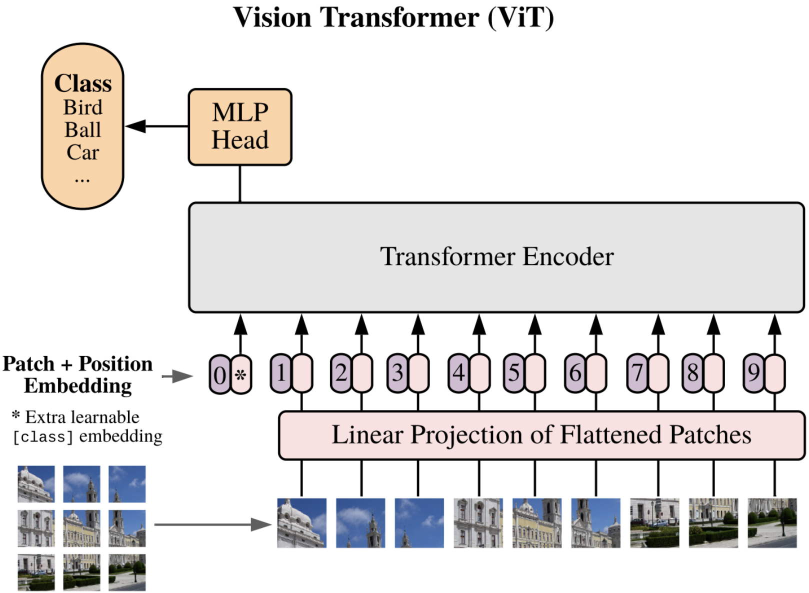
```
1091
```

- Prompt!

Remark

GPT and its focus on next-token prediction have fundamentally transformed how pre-trained models are utilized, marking a significant step toward AGI. The transition from BERT to GPT represents a **major breakthrough** in this evolution.

Vision Transformer (ViT)



Summary

- Transformers or attention-based models are versitile in many applications.
- Next-token prediction is powerful and **it implicitly performs multi-task learning**. The latter might be the major reason of why GPT is so successful.