Natural Language Processing (NLP) and Large Language Models (LLMs)
Lecture 7-2: The transformer

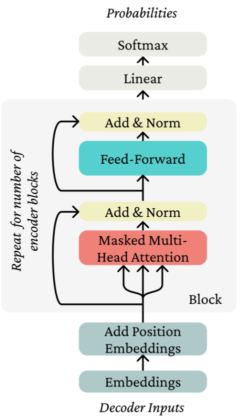
Chendi Wang (王晨笛)
chendi.wang@xmu.edu.cn

WISE @ XMU

2025 年 4 月 17 日

Adapted in part from CS 224n and Dive into deep learning

Recap: a self-attention layer



Transformer Decoder

Figure is from Stanford cs224n

Transformer

- Attention Is All You Need, Vaswani et al., 2017

Transformer-based LLMs

Model Name	Org / Team	Year	Type	Params	Primary Use Cases
BERT	Google AI	2018	Encoder	110M (base)	Text classification, QA, NER
RoBERTa	Facebook AI	2019	Encoder	125M (base)	Robust BERT with more training
ALBERT	Google / TTIC	2019	Encoder	12M–235M	Efficiency-focused BERT variant
DeBERTa	Microsoft	2021	Encoder	140M+	Enhanced BERT with disentangled attention
GPT-2	OpenAI	2019	Decoder	117M–1.5B	Text generation, completion
GPT-3	OpenAI	2020	Decoder	175B	General-purpose language generation
GPT-4	OpenAI	2023	Decoder	Undisclosed	Multimodal (text & image), general-purpose
LLaMA	Meta AI	2023	Decoder	7B–65B	Open-source alternative to GPT
Gemma	Google DeepMind	2024	Decoder	2B–7B	Lightweight open LLMs

Figure is generated by GPT-4o

Transformer-based LLMs

模型名称	机构 / 团队	发布时间	架构类型	参数规模	主要用途
DeepSeek	DeepSeek（深度求索）	2023	Decoder	7B / 33B / 671B	代码生成、推理任务、通用文本生成
文心一言 (ERNIE Bot)	百度	2023	Decoder	~百亿到千亿级	中文问答、多模态、搜索、写作
通义千问 (Qwen)	阿里达摩院 / 阿里云	2023	Decoder	7B / 14B / 72B+	多语言问答、文本生成、代码生成
ChatGLM 系列	清华 KEG / 智谱AI	2022–2024	Decoder	6B / 10B / 130B+	中文对话、多轮问答、知识增强
百川 (Baichuan)	百川智能	2023	Decoder	7B / 13B / 53B+	通用生成、多语言、代码、对话
悟道·天鹰 / 悟道2.0	智源研究院 (Beijing Academy of AI)	2021–2023	Decoder	百亿到千亿级	通用大模型、多模态、科研平台
盘古α (PanGu-α)	华为诺亚方舟实验室	2021	Decoder	200B	中文文本生成、金融、工业应用
昇腾MindGPT	华为昇腾团队	2023	Decoder	~百亿级	轻量级本地部署

Figure is generated by GPT-4o

Vision transformer (ViT) achieves the SOTA of computer vision

Published as a conference paper at ICLR 2021

AN IMAGE IS WORTH 16X16 WORDS: TRANSFORMERS FOR IMAGE RECOGNITION AT SCALE

Alexey Dosovitskiy[*,†], **Lucas Beyer**[*], **Alexander Kolesnikov**[*], **Dirk Weissenborn**[*], **Xiaohua Zhai**[*], **Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, Neil Houlsby**[*,†]
[*]equal technical contribution, [†]equal advising
Google Research, Brain Team
{adosovitskiy, neilhoulsby}@google.com

Why is (vision) transformer so powerful?

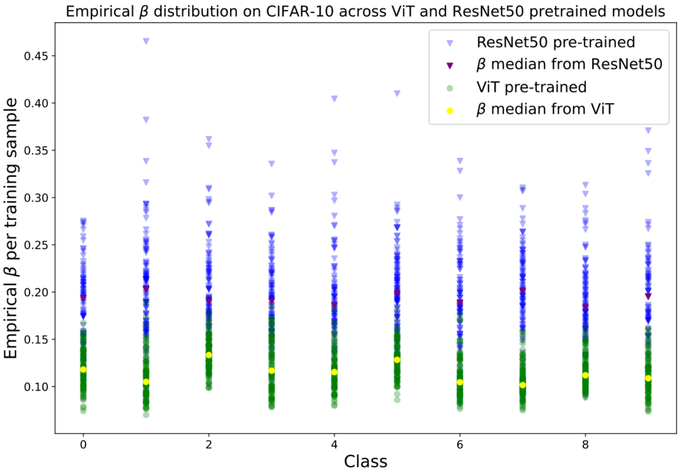
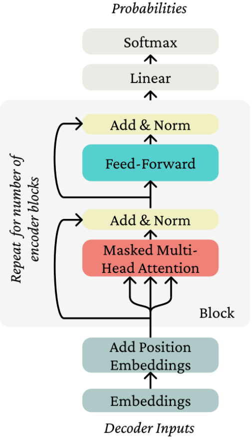


Figure is from Wang et al., 2024

What we have done so far?

- The Transformer is an architecture based on self-attention that consists of *stacked Blocks*.
- Each block contains self-attention and feedforward layers.
- We still need multi-head self-attention, layer normalization, residual connections, and attention scaling.



Transformer Decoder

Figure is from Stanford CS 224n

1. Section 1: Multi-head attention

2. Section 2: Add & Norm Layers

3. Section 3: Encoder & Decoder

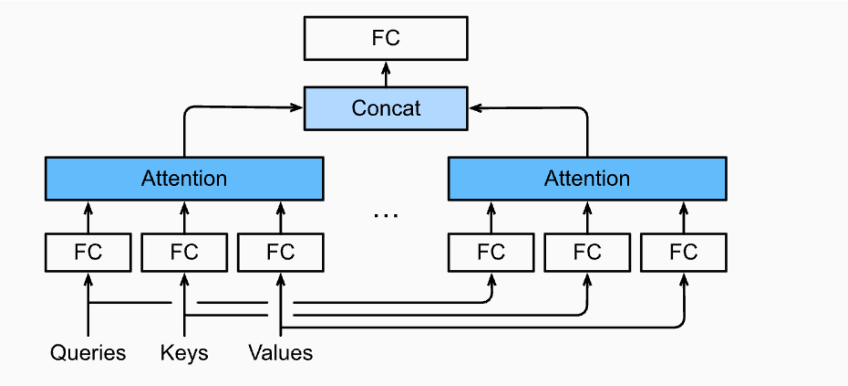
Single Head vs. Multi-Head Attention

- A single call of self-attention selects one value from a set of values (a single attention head).
- It softly achieves one objective by computing attention weights $\boldsymbol{\alpha} = (\alpha_1, \cdots, \alpha_m)$ based on queries, keys, and values.
- Ideally, we want to apply self-attention multiple times in parallel to capture different types of information —this is the motivation behind multi-head attention.

An example of multi-head attention

Figure if from Stanford CS 224n

Multi-head attention



Recap: single head scaled dot-product self-attention

- $\mathbf{q}_t = \mathbf{Q}\mathbf{e}_t, \mathbf{k}_t = \mathbf{K}\mathbf{e}_t, \mathbf{v}_t = \mathbf{V}\mathbf{e}_t$ with embeddings $\mathbf{e}_t \in \mathbb{R}^d$ and $\mathbf{Q}, \mathbf{K}, \mathbf{V} \in \mathbb{R}^{d\times d}$
- $\mathbf{a}_t = (a_{t1}, \cdots, a_{tL})$ with $a_{ts} = \mathbf{q}_t^T \mathbf{k}_s / \sqrt{d}$.
- $\boldsymbol{\alpha}_t = (\alpha_{t1}, \cdots, \alpha_{tL}) = \mathrm{SoftMax}(\mathbf{a}_t)$.
- $h_t = \sum_{s=1}^{L} \alpha_{ts}\mathbf{v}_s \in \mathbb{R}^d$.
- Concisely, we write: $h_t = \mathrm{Attention}(\mathbf{e}_t; \mathbf{Q}, \mathbf{K}, \mathbf{V})$.

Multi-Head Attention

- Suppose we use K attention heads, each producing an output $h_t^{(k)} \in \mathbb{R}^d$, $k = 1, 2, \cdots, K$.
- Here each $h_t^{(k)} = \mathrm{Attention}(\mathbf{e}_t; \mathbf{Q}^{(k)}, \mathbf{K}^{(k)}, \mathbf{V}^{(k)}) \in \mathbb{R}^d$.
- Then the concatenated output is

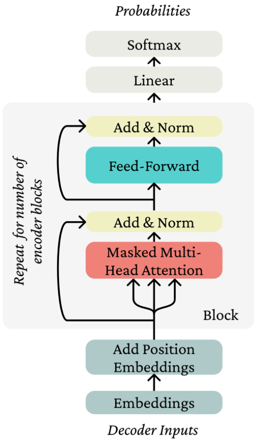
$$h_t = \mathrm{Concat}(h_t^{(1)}; \cdots; h_t^{(K)}) \in \mathbb{R}^{Kd}.$$

- But what if we still want $h_t \in \mathbb{R}^d$?
  (Why? reduce computational cost)
- Solution: reduce the dimensionality of each head by setting projection matrices $\mathbf{Q}^{(l)}, \mathbf{K}^{(l)}, \mathbf{V}^{(l)} \in \mathbb{R}^{d_k \times d}$, where $d_k = d/K$.
- Then, the attention score becomes

$$a_{ts} = \frac{\mathbf{q}_t^\top \mathbf{k}_s}{\sqrt{d_k}}.$$

What's next?

- The Transformer is an architecture based on self-attention that consists of *stacked Blocks*.
- Each block contains multi-head self-attention and feedforward layers.
- We still need layer normalization, residual connections.
- Layer normalization and residual connections are written together as a "Add & Norm" layer in most Transformer diagrams.



Transformer Decoder

Layer normalization (Ba et al., 2016)

- The output of an attention layer $h_t = \mathrm{Attention}(\mathbf{e}_t; \mathbf{Q}, \mathbf{K}, \mathbf{V}) \in \mathbb{R}^d$.
- Write $h_t = (h_{t1}, \cdots, h_{td})$.
- Mean: $\widehat{\mu}_t = \frac{1}{d}\sum_{j=1}^{d} h_{tj}$
- Variance: $\widehat{\sigma}_t^2 = \frac{1}{d}\sum_{j=1}^{d}(h_{tj} - \widehat{\mu}_j)^2$
- Layer Normalization:

$$\mathrm{LN}(h_t) = \frac{h_t - \widehat{\mu}_t \cdot \mathbf{1}}{\widehat{\sigma}_t}.$$

Why layer normalization?

- Originally, Ba et al., (2016) use layer normalization since batch normalization can not be applied to RNN.
- Layer Xu et al., (2019) found that LN may be most useful not in normalizing the forward pass, but actually in improving gradients in the backward pass.
- Reading materials: Understanding and improving layer normalization. Xu et al., 2019

Residual connection (He et al., 2016)

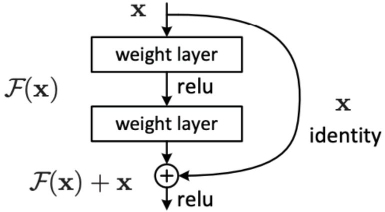


Figure 2. Residual learning: a building block.

Residual Connection (He et al., 2016)

- Given an input representation h, the goal is to predict a target output $\widehat{y}$.
- The residual connection focuses on modeling the difference: $\widehat{y} - h$.
- A neural network $f(h) = \mathbf{W}_2\sigma(\mathbf{W}_1 h + b)$ is used to approximate the residual.
- This leads to the formulation:

$$\widehat{y} \approx f(h) + h.$$

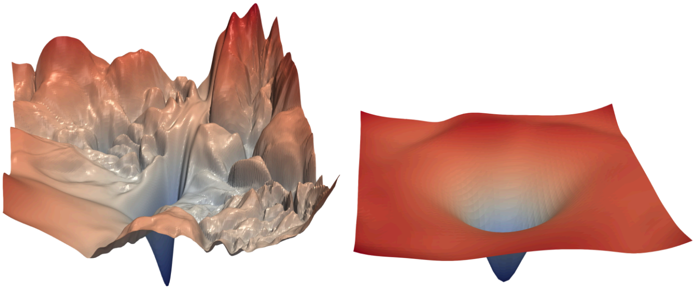
- But what if $f(h)$ and h have different dimensions?
- Solution: apply a linear transformation to h, i.e.,

$$\widehat{y} \approx f(h) + \mathbf{W}' h.$$

Residual Connection (He et al., 2016)

- $f_{\mathrm{res}} = f(h_{1:L}) + h_{1:L}$ with $h_{1:L} = (h_1, \cdots, h_L) \in \mathbb{R}^{Ld}$.
- Here f is a fully connected neural network.

Why Use Residual Connections?



(a) without skip connections　　　(b) with skip connections

Figure 1: The loss surfaces of ResNet-56 with/without skip connections. The proposed filter normalization scheme is used to enable comparisons of sharpness/flatness between the two figures.

Figure is from https://arxiv.org/abs/1712.09913

Why Use Residual Connections?

- The identity function provides a smooth gradient flow, which helps mitigate the vanishing gradient problem in deep networks.
- It is often easier to learn the residual —that is, the difference between a function and the identity —than to learn the full function from scratch.

Two Add & Norm Variants

- **Pre-Norm:** Apply Layer Normalization before the residual block:

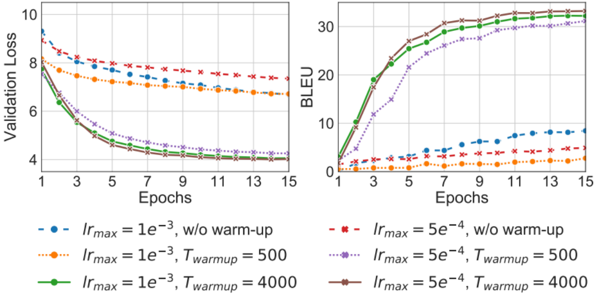
$$h_{\text{pre-norm}} = f_{\text{res}}(\mathrm{LN}(h_{1:n})) + h_{1:n}$$

- **Post-Norm:** Apply Layer Normalization after the residual addition:

$$h_{\text{post-norm}} = \mathrm{LN}(f_{\text{res}}(h_{1:n}) + h_{1:n})$$

- Pre-norm has been shown to yield more stable gradients at initialization, resulting in significantly faster training (Xiong et al., 2020).
- Most modern language models use pre-norm except for BERT.

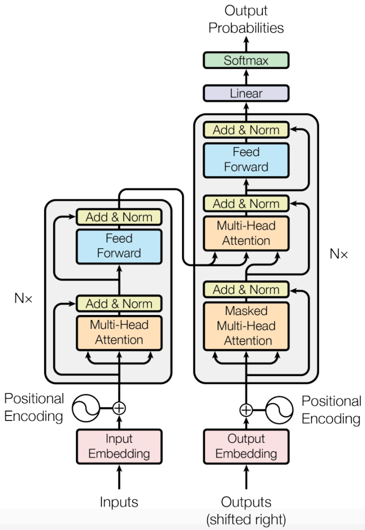
Pre-norm v.s. Post-norm



(a) Loss/BLEU on the IWSLT14 De-En task (Adam)

1. Section 1: Multi-head attention
2. Section 2: Add & Norm Layers
3. **Section 3: Encoder & Decoder**

What is the difference between encoder and decoder? Why?



The Transformer Decoder

- The Transformer decoder is composed of a stack of identical Transformer Decoder Blocks.
- Each block consists of the following components:
  - Masked self-attention
  - Add & Layer Normalization
  - The encoder–decoder attention (keys and values are from encoder, queries are from decoder)
  - Add & Layer Normalization
  - Position-wise feed-forward network
  - Add & Layer Normalization
- The decoder operates in a uni-directional (causal) manner to ensure proper autoregressive language modeling.

The Transformer Encoder

- The Transformer encoder is composed of a stack of identical **Encoder Blocks**.
- Unlike the decoder, the encoder is **bi-directional** (no masking is applied).
- Each block consists of the following components:
  - Self-attention
  - Add & Layer Normalization
  - Position-wise feed-forward network
  - Add & Layer Normalization

Implementation with PyTorch

- Transformer.ipynb

Advantage: Parallelizable Self-Attention

- Computes attention scores and weighted sums through efficient matrix operations
- Processes all positions simultaneously, unlike sequential RNNs
- Maintains parallelizability with future-masking for autoregressive tasks

Disadvantage: Quadratic Computational Complexity

- **Scaling issue**: Computing $\mathbf{q}_t^\top \mathbf{k}_s$ for all (t, s) pairs requires $O(L^2 d)$ operations
- **Practical limitation**: Typical maximum length constraints $(L \leq 512)$ restrict applications
- **Challenge for LLMs**: Sequence generation becomes computationally expensive for long contexts

Return of the RNNs for long sequences

- RNN structures such as RWKV or Mamba may have better performance for very long sequences

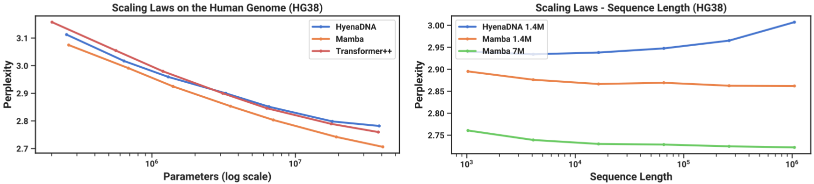
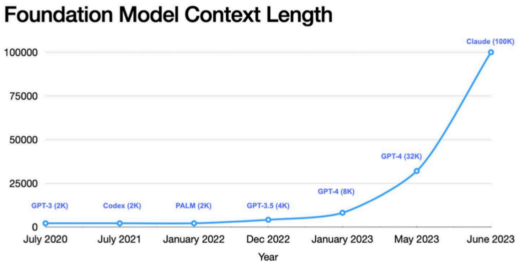


Figure: Mamba architecture

Figure is from Mamba: Linear-Time Sequence Modeling with Selective State Spaces. Gu and Dao, 2023

Why do we still use transformers for long sequences?

- Despite the quadratic cost of self-attention, an increasingly large portion of compute is spent outside the attention mechanism.
- Modern optimizations (e.g., FlashAttention, sparse attention) mitigate the memory bottleneck.
- Transformers benefit from parallel training and strong pretraining scalability.



What's next?

- Pretraining/fine-tuning
- Model evaluation (a suggested research field)