Lec. 3: Architectures, Hyperparameters

📌 **关键词**

- Transformer Architectures
- Hyper-parameters

1. **Pre-norm**
   几乎所有的现代模型使用pre-norm，BERT除外，它是post-norm。
   post-norm训练没有那么稳定，需要learning rate warm up。
   推测的原因：pre-norm使梯度大小恒定，post-norm会使梯度爆炸。
   最初的优点在于不需要warm up，在大规模网络的今天，优点在于**pre-norm更稳定，能够使用更大的学习率**。

2. **Double Norm**

   把Layer Norm放在残差流中不好，那么为什么不把LN放在Attention和FFN的后面？
   Grok和Gemma2不仅用了pre-norm，还在FFN和Multi-head Attention后面加了一层LN。

   Layer Norm在Residual中不好的原因：Residual Stream是一个从上到下连通的结构，能很好地回传梯度，如果把LN放在这中间，可能会干扰这种梯度回传。
   GPT：在 Post-Norm（把 LayerNorm 放在残差相加之后）里，每一层的残差分支都要经过一个 LayerNorm，这就意味着原本直接的恒等映射（identity path，也就
   是“output=Sublayer(x)+x”里的“+x”，能使梯度直通）不再是“纯粹”的恒等映射，梯度回传时要穿过 LayerNorm 的雅可比（所有偏导数组成的矩阵，当是恒等映射时，其为单位矩阵，如果有LN，其特征值通常 ⩽ 1），长层网络里多次累积就容易造成梯度缩放或衰减，从而不够稳定。
   而 Pre-Norm 架构恰好把 LayerNorm 提到子层计算之前，只对子层输入做归一化，残差分支保持了纯粹的 identity，这样做能大幅提升深层网络的训练稳定性，也就允许用更大的学习率、更深的堆叠。

3. **Layer Norm vs RMSNorm**

LayerNorm vs RMSNorm

Original transformer: **LayerNorm** – normalizes the mean and variance across d_{model}

$$y = \frac{x - \mathrm{E}[x]}{\sqrt{\mathrm{Var}[x] + \epsilon}} * \gamma + \beta$$

Notable models:

GPT3/2/1, OPT, GPT-J, BLOOM

Many modern LMs: **RMSNorm** – does not subtract mean or add a bias term

$$y = \frac{x}{\sqrt{||x||_2^2 + \varepsilon}} * \gamma$$

Notable models:

LLaMA-family, PaLM, Chinchilla, T5

RMSNorm 更快（更少的operation，不用减去平均值；更少的参数量）且效果同样好

4. **Drop the bias**

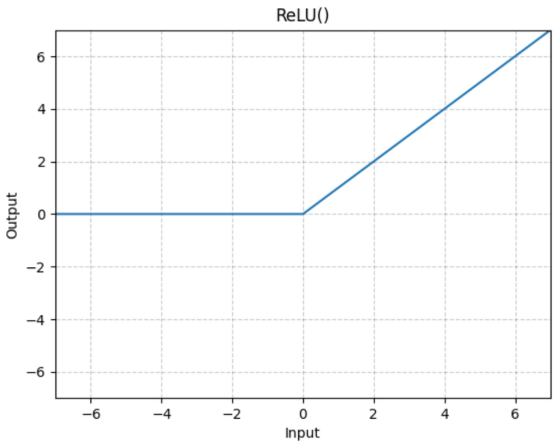
   FFN架构去掉bias -> 更少的memory，优化更稳定

5. **Activations**

   ReLU, GeLU, Swish, ELU, GLU, GeGLU, ReGLU, SeLU, SwiGLU, LiGLU

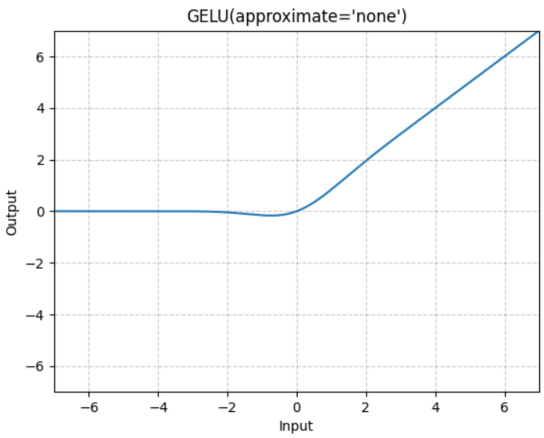
A **ReLU** $ReLU(x) = max(0, x)$

- **非线性**：x<0时将所有输入截断为0，使网络能学习复杂的非线性映射。
- **稀疏激活**：输入为负时，输出恒为0，即关闭了一部分神经元，产生了稀疏性，有助于缓解过拟合并提高计算效率。
- **梯度简洁**：x<0时梯度为0，x>0时梯度为1，边界点 x=0处可取任意常数或0。简单的梯度计算既加快了反向传播的速度，也减少了梯度消失的风险。
- **优点**：收敛更快、计算高效、抗梯度消失
- **缺点**：死亡ReLU问题（始终接收负输入，对应梯度恒为0，永远不活跃）、忽略负值信息（可能丢失一部分有用特征）
- **模型**：Original transformer, T5, Gopher, Chinchilla, OPT



B **GeLU** $GeLU(x) = x\Phi(x)$

- 其中 $\Phi(x) = \int_{-\infty}^{x} \frac{1}{\sqrt{2\pi}} e^{-t^2/2}\, dt$ 是标准正态分布的累积分布函数（随机变量X小于等于x的概率）
- 用tanh近似时，
  $GeLU(x) = 0.5x\Big(1 + Tanh[\sqrt{\frac{2}{\pi}}(x + 0.044715x^3)]\Big)$
- **平滑非线性**：整个实数域上都具有连续的一阶导数，优化过程更平滑
- **输入加权**：通过乘 $\Phi(x)$，对正输入几乎保留原值，负输入近似置零，中间区间按照高斯概率平滑过渡
- **随机特性**：对x以 $\Phi(x)$ 的概率进行保留，否则置零，有一定类似Dropout的随机正则化效果



- **优点**：训练更稳定——平滑曲线和连续导数减少了梯度的噪声，尤其在超深和大规模模型中；性能提升；正则化效果，一定程度上缓解过拟合
- **缺点**：计算开销更大（现有库已高效实现）；可解释性不如ReLU直观
- **模型**：GPT1/2/3, GPTJ, GPT-Neox, BLOOM

🌟 **GLU**：$max(0, xW_1) => max(0, xW_1) \otimes (xV)$ xV 作门控作用 $\otimes$ 是逐元素相乘

ReGLU：$FF_{ReGLU}(x) = (max(0, xW_1) \otimes xV)W_2$

GeGLU：$FFN_{GEGLU}(x, W, V, W_2) = (GELU(xW) \otimes xV)W_2$
模型：T5 v1.1, mT5, LaMDA, Phi3, Gemma 2, Gemma 3

SwiGLU：（swish is x * sigmoid(x)）
$FFN_{SwiGLU}(x, W, V, W_2) = (Swish_1(xW) \otimes xV)W_2$
模型：LLaMa 1/2/3, PaLM, Mistral, OlMo, most models post 2023

带门控的模型，d_{ff} 的维度是原来的2/3
普通FFN的参数量是 $2 * d_{model} * d_{ff}$
带门控的模型参数量是 $xW_1 : d_{model} * d_{ff}, xV : d_{model} * d_{ff}, (...)W_2 : d_{model} * d_{ff}$
加起来是原来的3/2倍，所以维度是原来的2/3

优点：动态信息筛选——用门动态控制，有选择地放大或抑制特征；增强非线性建模；正则化效果——门控随机关闭部分通道，类似Dropout；性能提升——能够略微提升下游任务性能和收敛速度；
捕捉长期依赖关系：GLU 引入了「线性变换 × 门控」的结构，它既保留了一条近似恒等（或线性）信息通道，又让网络能够动态地选择哪些信息要“放行”到下一层，从而更好地捕捉和维护跨层、跨距离的信息。

缺点：参数和计算开销——增加了约50%的点乘/投影开销；逐元素相乘会改变梯度分布，对学习率和优化器更敏感

6. **Serial vs Parallel**
   传统Transformer是serial的，一些模型尝试了并行——同时算MLP和Attention，然后把它们加起来（可以share参数）。（更多的还是选择串行）

7. **Position Embedding**

Sine embeddings: add sines and cosines that enable localization

$$Embed(x,i) = v_x + PE_{pos}$$

$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{\text{model}}})$$
$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{\text{model}}})$$

Notable models:

Original transformer

Absolute embeddings: add a position vector to the embedding

$$Embed(x,i) = v_x + u_i$$

Notable models:

GPT1/2/3, OPT

Relative embeddings: add a vector to the *attention computation*

$$e_{ij} = \frac{x_i W^Q (x_j W^K + a_{ij}^K)^T}{\sqrt{d_z}}$$

Notable models:

T5, Gopher, Chinchilla

Rope embeddings (next slides..)

Notable models:
GPTJ, PaLM, LLaMA
Most 2024+ models

RoPE数学推导

作者讲解：https://zhuanlan.zhihu.com/p/359502624

🌟 Transformer的输入大小是 [batch_size, sequence_length, d_model]

忽略batch_size，sequence中的每一个词都对应一个位置下标，下面用m、n代替，d_model指embedding的维度，用i指代是embedding中的第几维，θ 是和i相关的函数，指基础角度步长。

背景：欧拉公式 $e_{ix} = cosx + isinx$ 复数：$z = x + iy = re^{i\phi}$ ϕ 是相位

假设d_model为2，那么 $q_m = \begin{pmatrix} q_m^0 \\ q_m^1 \end{pmatrix}$，可以表示成 $q_m = q_m^0 + iq_m^1$，旋转后 $q'_m = q_m e^{im\theta} = (q_m^0 + iq_m^1)(cosm\theta + isinm\theta)$，展开后写成矩阵乘法就是：

$$\begin{pmatrix} q_m^{0'} \\ q_m^{1'} \end{pmatrix} = \begin{pmatrix} cosm\theta & -sinm\theta \\ sinm\theta & cosm\theta \end{pmatrix} \begin{pmatrix} q_m^0 \\ q_m^1 \end{pmatrix}$$

Transformer中用第m个词的 $\vec{q}$ 乘其它词的 $\vec{k}$，假设现在乘的是第n个词的key，为了乘出来是标量，用 $\vec{q}_m^{T'}$ 乘 $\vec{k}'_n$。

$$q_m'^T k'_n = \begin{pmatrix} q_m^0 & q_m^1 \end{pmatrix} \begin{pmatrix} cosm\theta & sinm\theta \\ -sinm\theta & cosm\theta \end{pmatrix} \begin{pmatrix} cosn\theta & -sinn\theta \\ sinn\theta & cosn\theta \end{pmatrix} \begin{pmatrix} k_n^0 \\ k_n^1 \end{pmatrix} =$$
$$\begin{pmatrix} q_m^0 & q_m^1 \end{pmatrix} \begin{pmatrix} cos(n-m)\theta & -sin(n-m)\theta \\ sin(n-m)\theta & cos(n-m)\theta \end{pmatrix} \begin{pmatrix} k_n^0 \\ k_n^1 \end{pmatrix} = (k_0 q_0 + k_1 q_1)cos(n-m)\theta - (k_1 q_0 - k_0 q_1)sin(n-m)\theta$$

可以看到乘积和相对位置n-m有关。

核心思想：通过某种方式修改 query (查询) 和 key (键) 向量，使得它们的点积结果不仅包含词本身的信息，还包含它们之间的相对位置信息。

希望找到一个函数 f，它接收原始词向量 x 和它的绝对位置 m，然后输出一个新的向量。这个函数需要满足以下性质：

$$\langle f(q,m), f(k,n)\rangle = g(q,k,m-n)$$

所以这个f在二维的时候就是：

$$f(q,m) = \begin{pmatrix} cosm\theta & -sinm\theta \\ sinm\theta & cosm\theta \end{pmatrix} \begin{pmatrix} q_m^0 \\ q_m^1 \end{pmatrix}$$

由于内积满足线性叠加性，因此任意偶数维的RoPE，我们都可以表示为二维情形的拼接，即

$$\underbrace{\begin{pmatrix} \cos m\theta_0 & -\sin m\theta_0 & 0 & 0 & \cdots & 0 & 0 \\ \sin m\theta_0 & \cos m\theta_0 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & \cos m\theta_1 & -\sin m\theta_1 & \cdots & 0 & 0 \\ 0 & 0 & \sin m\theta_1 & \cos m\theta_1 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & \cos m\theta_{d/2-1} & -\sin m\theta_{d/2-1} \\ 0 & 0 & 0 & 0 & \cdots & \sin m\theta_{d/2-1} & \cos m\theta_{d/2-1} \end{pmatrix}}_{\boldsymbol{W}_m} \begin{pmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \\ \vdots \\ q_{d-2} \\ q_{d-1} \end{pmatrix}$$

由于 W_m 的稀疏性，所以直接用矩阵乘法来实现会很浪费算力，对矩阵进行分解：

$$\begin{pmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \\ \vdots \\ q_{d-2} \\ q_{d-1} \end{pmatrix} \otimes \begin{pmatrix} \cos m\theta_0 \\ \cos m\theta_0 \\ \cos m\theta_1 \\ \cos m\theta_1 \\ \vdots \\ \cos m\theta_{d/2-1} \\ \cos m\theta_{d/2-1} \end{pmatrix} + \begin{pmatrix} -q_1 \\ q_0 \\ -q_3 \\ q_2 \\ \vdots \\ -q_{d-1} \\ q_{d-2} \end{pmatrix} \otimes \begin{pmatrix} \sin m\theta_0 \\ \sin m\theta_0 \\ \sin m\theta_1 \\ \sin m\theta_1 \\ \vdots \\ \sin m\theta_{d/2-1} \\ \sin m\theta_{d/2-1} \end{pmatrix}$$

其中 $\otimes$ 是按位相乘。

🌟 为了更清楚地理解为什么要这样拼接 -> 尝试计算四维的情况

此时m=1，n=4

$q = [q_0, q_1, q_2, q_3], k = [k_0, k_1, k_2, k_3]$

第一组 (维度0, 1) 使用的旋转频率是 θ_0 ，第二组 (维度2, 3) 使用的旋转频率是 θ_1 ， $\theta_0 \neq \theta_1$

分别进行旋转：

对q在位置m=1进行旋转

第一组 (θ_0):

- $q'_0 = q_0 cos(1 \cdot \theta_0) - q_1 sin(1 \cdot \theta_0)$
- $q'_1 = q_0 sin(1 \cdot \theta_0) + q_1 cos(1 \cdot \theta_0)$

第二组 (θ_1):

- $q'_2 = q_2 cos(1 \cdot \theta_1) - q_3 sin(1 \cdot \theta_1)$
- $q'_3 = q_2 sin(1 \cdot \theta_1) + q_3 cos(1 \cdot \theta_1)$

对k在位置n=4进行旋转

第一组 (θ_0):

- $k'_0 = k_0 cos(4 \cdot \theta_0) - k_1 sin(4 \cdot \theta_0)$
- $k'_1 = k_0 sin(4 \cdot \theta_0) + k_1 cos(4 \cdot \theta_0)$

第二组 (θ_1):

- $k'_2 = k_2 cos(4 \cdot \theta_1) - k_3 sin(4 \cdot \theta_1)$
- $k'_3 = k_2 sin(4 \cdot \theta_1) + k_3 cos(4 \cdot \theta_1)$

计算 q'和k' 的内积

$$q'_0 k'_0 = q_0 k_0 [\frac{cos3\theta_0}{2} + \frac{cos5\theta_0}{2}] - q_1 k_0 [\frac{sin5\theta_0}{2} - \frac{sin3\theta_0}{2}] - q_0 k_1 [\frac{sin5\theta_0}{2} + \frac{sin3\theta_0}{2}] + q_1 k_1 [\frac{cos3\theta_0}{2} - \frac{cos5\theta_0}{2}]$$

$q'_1 k'_1$ 同理，和上式相加抵消掉了所有的 $5\theta_0$ 。

最后得到 $(q_0 k_0 + q_1 k_1)cos(3\theta_0) - (q_0 k_1 - q_1 k_0)sin(3\theta_0)$ ，和2维的时候算出来的一样。

对于第二组，同理： $q'_2 k'_2 + q'_3 k'_3 = (q_2 k_2 + q_3 k_3)cos(3\theta_1) - (q_2 k_3 - q_3 k_2)sin(3\theta_1)$

$\theta_i = base^{-\frac{2i}{d}}, base = 10000$ （随着相对距离的变大，内积结果有衰减趋势的出现）

当i很小（即embedding前面部分的维度）时， θ_i 接近1，频率较大；
当i很大（即embedding后面部分的维度）时， θ_i 接近1/10000，频率较小。
不同频率捕捉不同距离的关系，高频部分对近距离的关系变化敏感，能精确捕捉词与词之间的精细、短程的相对位置关系。
低频部分对远距离的位置变化更敏感，能有效编码长程的、宏观的相对位置关系。

数据流：

Token A (位置 m) -> 原始 q 向量 -> **用位置 m 生成旋转矩阵** -> 旋转 q 得到 q'

Token B (位置 n) -> 原始 k 向量 -> **用位置 n 生成旋转矩阵** -> 旋转 k 得到 k'

计算注意力分数： $AttentionScore(A, B) = q' \cdot k'$

因为 q' 和 k' 都被各自位置的旋转矩阵“注入”了位置信息，所以它们的点积结果就自然而然地包含了 m-n 这个相对位置信息。

8. **Hyper-parameters**

FFN中，**隐藏层的大小** $d_{ff} = 4d_{model}$ ，GLU中是8/3倍的d_model，约为2.66

num_heads, head_dim：num_heads * head_dim / d_model = 1

Aspect ratios：宽度和深度的选择，d_model/n_layer=128

增加深度会使模型无法在多设备上并行，因为需要等待前一个layer计算完成后才能开始当前layer，增加宽度可以轻松地并行。

Vocabulary sizes：单语模型——35-50K，多语言——100-250K

Dropout & Weight Decay：在pre-training中，很难过拟合，除qwen外，大多选择只用weight decay

GPT：即便在训练大规模语言模型时我们仍然会使用诸如权重衰减（weight decay）、Dropout、LayerNorm、梯度裁剪（gradient clipping）等“正则化”手段，但它们对模型**泛化能力**（防止过拟合）的提升作用已经相对有限。相反，这些正则化技术现在主要是用来**调节训练过程**本身的动态特性，比如：

- **平滑梯度**，减少突变
- **稳定学习率**，防止学习率过大导致发散
- **控制参数更新幅度**，避免训练初期的数值不稳定
- **改善收敛速度**，在整体 loss 曲面更平滑、噪声更小的环境下更容易到达低点

总之，它们如今更多地被看作是“优化技巧”——让大模型在巨量数据和超大参数下的训练更加稳定、高效——而不是像在小模型／小数据集里那样，主要用来对抗过拟合、提升泛化。

Summary: hyperparameters

Feedforward

- Factor-of-4 rule of thumb (8/3 for GLUs) is standard (with some evidence)

Head dim

- Head dim*Num head = D model is standard – but low to no validation

Aspect ratio

- Wide range of 'good' values (100-200). Systems concerns dictate the value

Regularization

- You still 'regularize' LMs but its effects are primarily on optimization dynamics

Name	Year	MLP factor	Aspect ratio (d/layer)	weight decay	drop_rate
Original transformer	2017	4	85	0	0.1
GPT	2018	4	64	0.1	0.1
T5 (11B)	2019	64	43	0	0.1
GPT2	2019	4	33	0.1	0.1
T5 (XXL 11B) v1.1	2020	2.5	171	0	0
mT5	2020	2.5	171	0	0
GPT3 (175B)	2020	4	128	0.1	0.1
GPTJ	2021		146	0.1	0
LaMDA	2021	8	128		
Anthropic LM (not claude)	2021	4	128		
Gopher (280B)	2021	4	205		
GPT-NeoX	2022	4	140	0.01	0
BLOOM (175B)	2022	4	205	0.1	0
OPT (175B)	2022	4	128	0.1	0.1
PaLM (540B)	2022	4	156		0
Chinchilla	2022	4	102		
Baichuan 2	2023	2.68	128	0.1	0
Mistral (7B)	2023	3.5	128	0.1	0
LLaMA2 (70B)	2023	3.5	102	0.1	0
LLaMA (65B)	2023	2.6875	102	0.1	0
GPT4	2023		0		
Olmo 2	2024	2.6875	128		
Gemma 2 (27B)	2024	8	100		
Nemotron-4 (340B)	2024	4	192		0
Qwen 2 (72b) - same for 2.5	2024	3.609	102		
Falcon 2 11B	2024	4	68	0.1	
Phi3 (small) - same for phi4	2024	3.5	128		
Llama 3 (70B)	2024	3.5	102		0
Reka Flash	2024		0		
Command R+	2024	2.75	192		
OLMo	2024	2.6875	128	0.1	0
Qwen (14B)	2024	2.675	128	0.1	0.1
DeepSeek (67B)	2024	2.6875	86	0.1	0
Yi (34B)	2024	2.857142	119	0.1	0
Mixtral of Experts	2024		0		
Command A	2025		0		
Gemma 3	2025	4	87		
SmolLM2 (1.7B)	2025	4	85		

9. **Stability Tricks**

   Transformer中有两个Softmax，分别在attention中和最后输出的地方。Softmax的不稳定性主要来自于分母的计算，容易overflow导致整个计算结果无效，使梯度计算崩溃。

   Z-loss (for output softmax) 引入一个辅助损失项，假设Softmax：`P(x) = e^U(x) / Z(x)`，`Z(x) = Σ e^U(x)`，这个损失项的目标是惩罚过大的Z(x)。$z_{loss} = 10^{-4} * log^2(Z)$，在 $log(Z) = 0$ 时取最小值，也就是Z=1，通过这个惩罚项，模型被“鼓励”去调整其输出的 logits `U(x)`，使得它们的归一化项 `Z(x) = Σ e^U(x)` 的值尽可能地接近 1。z-loss 就像一个正则化项，专门作用于 logits 的大小。

   使用该loss的模型：Baichuan 2 (2023), DCLM (2024), OLMo 2 (2025)

   QK norm (for softmax in attention)：把 每个 query 向量和 key 向量都单独做 L2 归一化，然后再算attention score，送入softmax。这能固定score的范围，使softmax更平滑，训练更稳定。

   模型： DCLM, OLMo2, Gemma 2, Qwen3

10. **Reducing attention head cost**

    GQA/MQA

    KV-Cache

    Sparse/Sliding window attention

    interleave ‘full’ and ‘LR’ attention

💡 总结栏

本节课详细介绍了transformer中架构上的改进，学习了激活函数以及相对位置编码。感觉大家有很认真地思考每一个模块到底选择哪一种，优缺点是什么，和小模型完全不一样。

GeLU推导：

$\Phi(x)=\int_{-\infty}^{x}\frac{1}{\sqrt{2\pi}}e^{-t^2/2}\,\mathrm{d}t$ 误差函数 $erf(z)=\frac{2}{\sqrt{\pi}}\int_0^z e^{-u^2}du$

令 $u=\frac{t}{\sqrt{2}}=>t=\sqrt{2}u$

$\Phi(x)=\int_{-\infty}^{\frac{x}{\sqrt{2}}}\frac{1}{\sqrt{\pi}}e^{-u^2}du=\int_{-\infty}^{0}\frac{1}{\sqrt{\pi}}e^{-u^2}+\int_0^{\frac{x}{\sqrt{2}}}\frac{1}{\sqrt{\pi}}e^{-u^2}=\frac{1}{2}+\int_0^{\frac{x}{\sqrt{2}}}\frac{1}{\sqrt{\pi}}e^{-u^2}=\frac{1}{2}[1+erf(\frac{x}{\sqrt{2}})]$

可以用一个三次多项式加权的tanh逼近erf(y)

$erf(y)\approx tanh(ay+by^3)=>\Phi(x)\approx\frac{1}{2}[1+tanh(a\frac{x}{\sqrt{2}}+b\frac{x^3}{2\sqrt{2}})]$

$a=\sqrt{\frac{2}{\pi}},b=0.044715*2a$