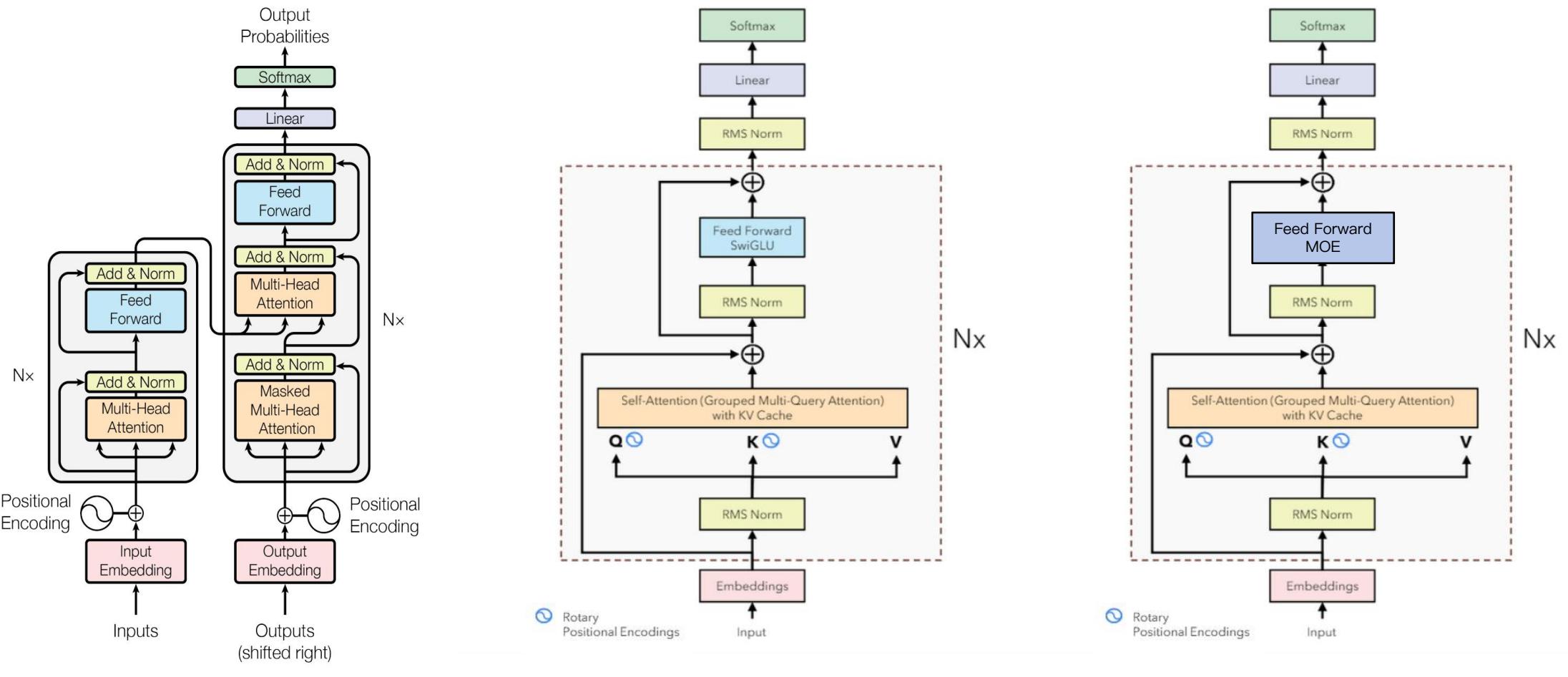


通过文心4.5学习大语言模型

Latest Upgrades of LLM

yang

大语言模型的最新进展

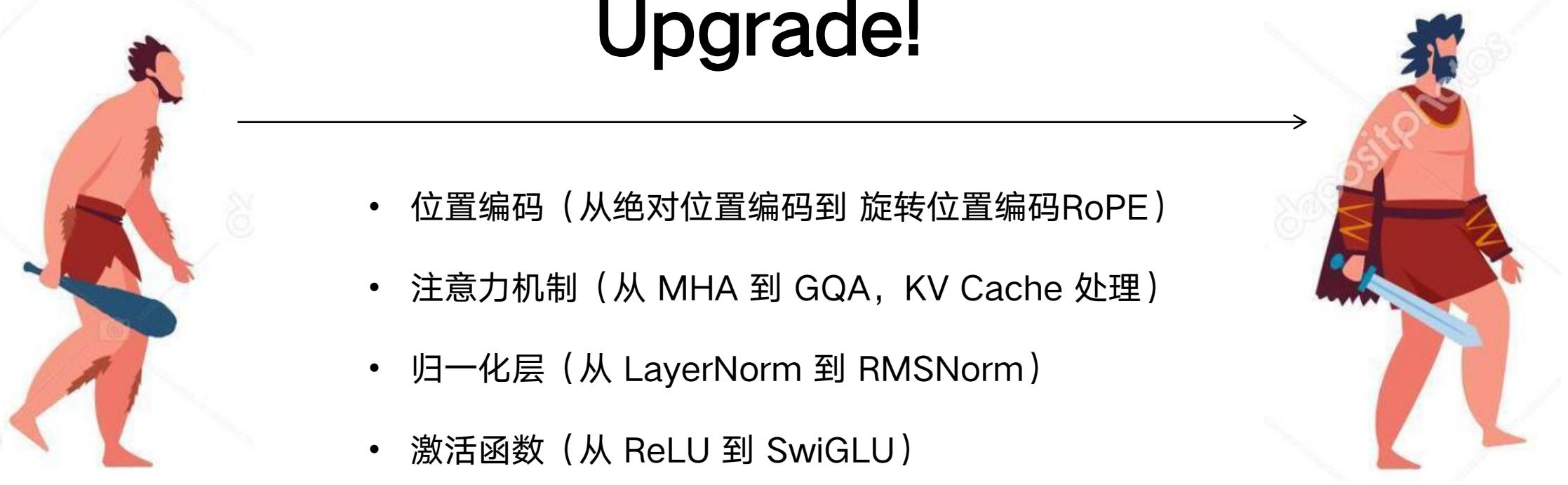


Transformer

LLama 3

ERNIE 4.5

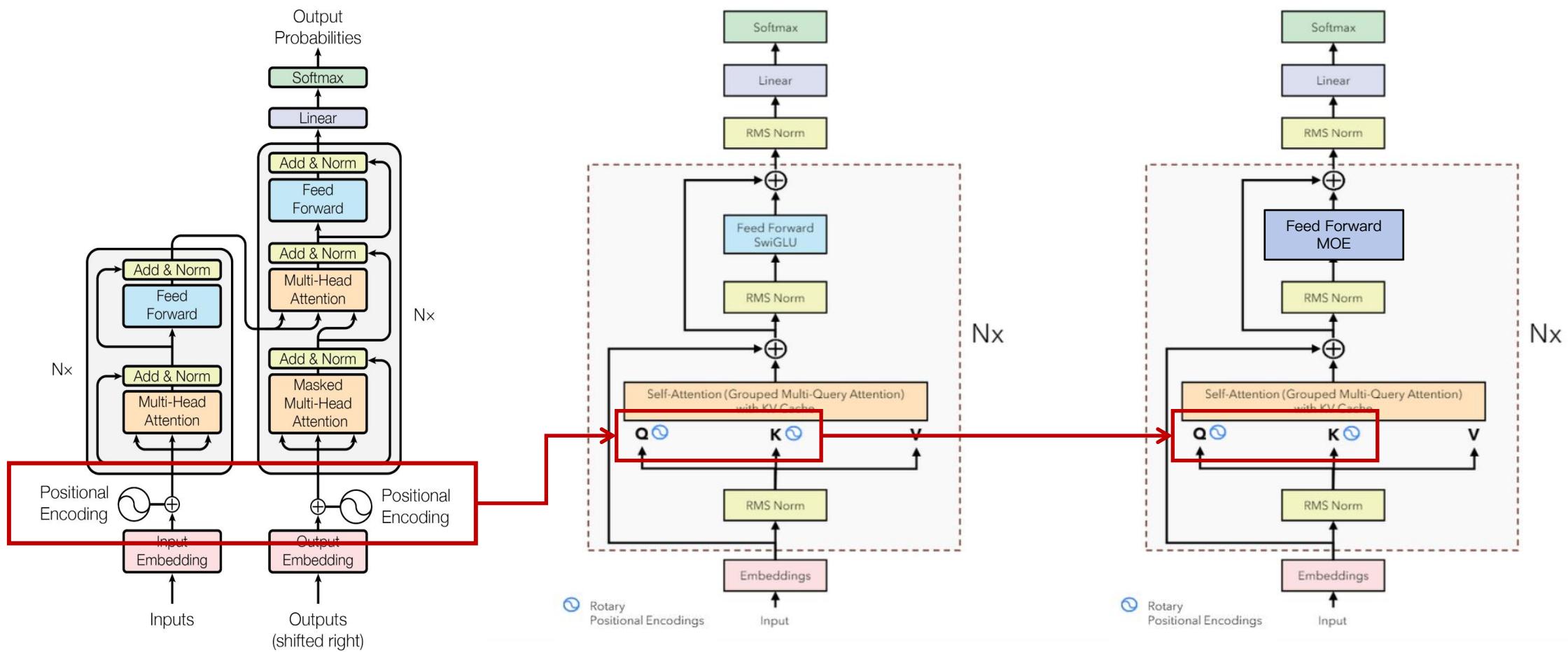
Upgrade!

- 
- 位置编码（从绝对位置编码到 旋转位置编码RoPE）
 - 注意力机制（从 MHA 到 GQA, KV Cache 处理）
 - 归一化层（从 LayerNorm 到 RMSNorm）
 - 激活函数（从 ReLU 到 SwiGLU）

- 位置编码



• 位置编码（绝对位置编码 → 旋转位置编码RoPE）

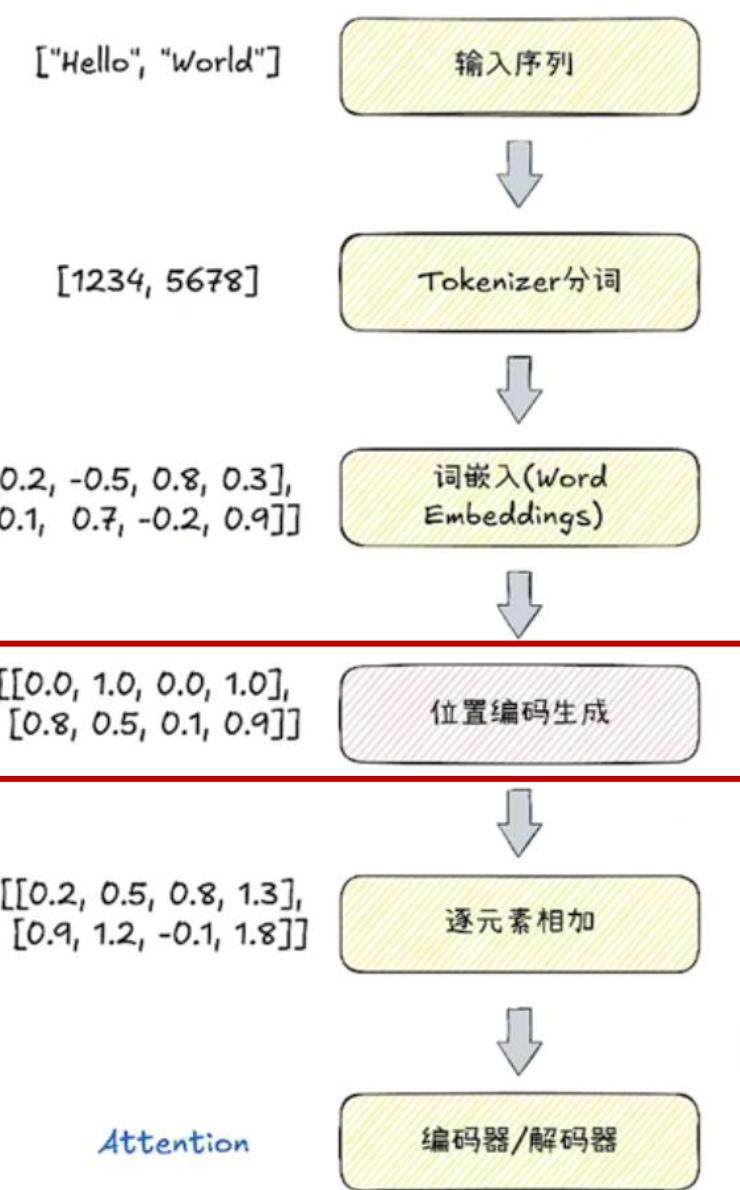
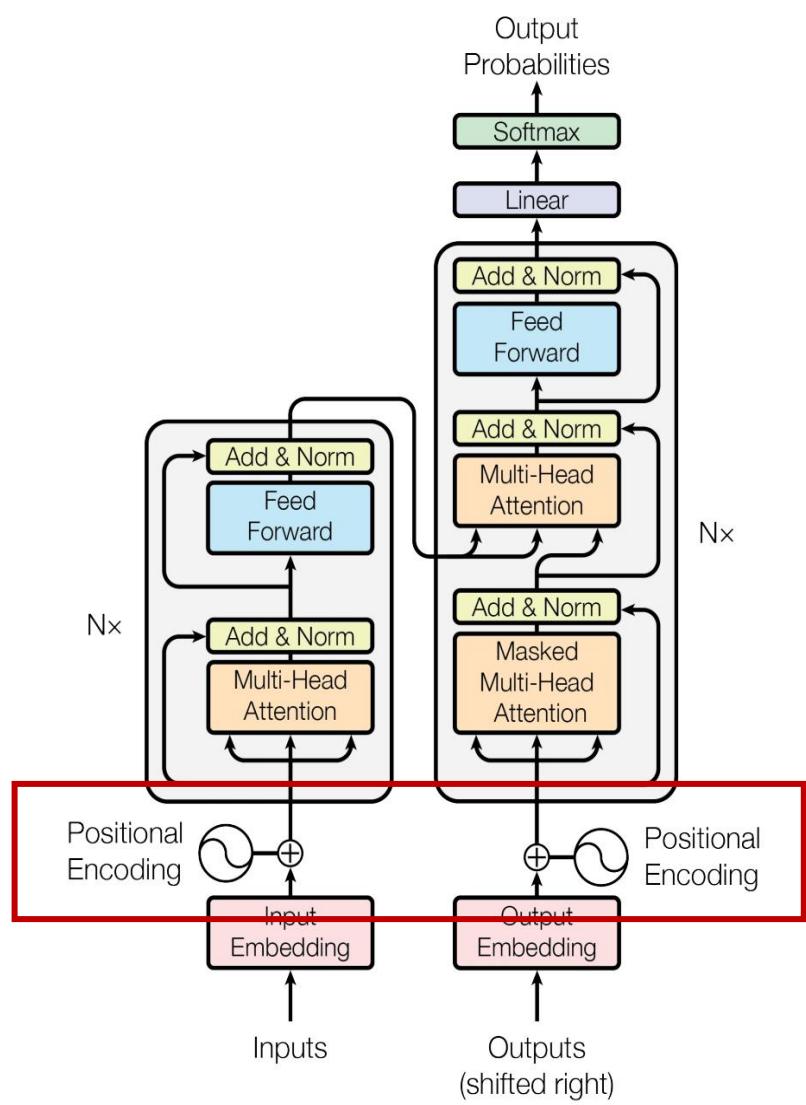


Transformer

LLama 3

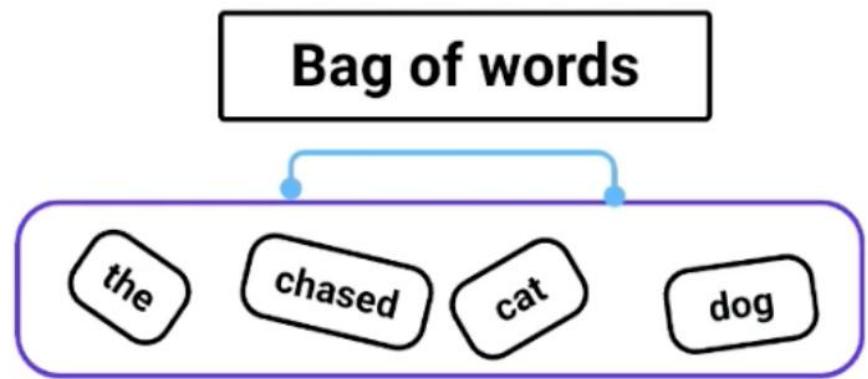
ERNIE 4.5

• 位置编码在哪里出现？



• 位置编码是用来干嘛的？

- The dog chased the cat
- The cat chased the dog



—Self-attention并行处理很快
—但Self-attention本质上不关心元素在序列里的绝对/相对位置
—文本问题trouble

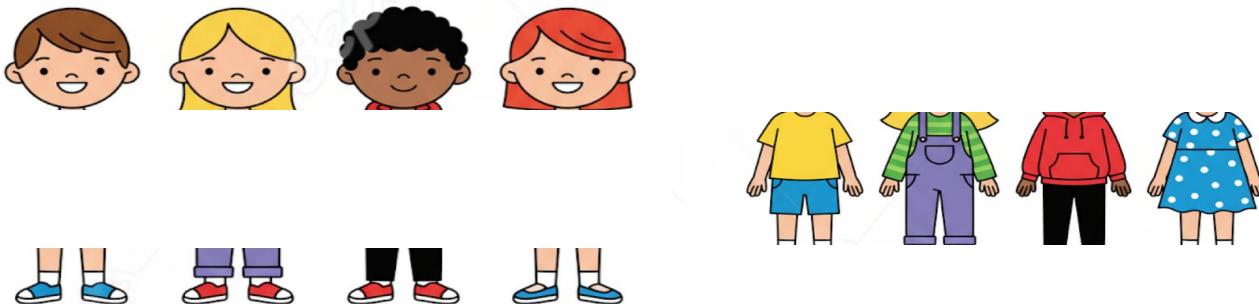
• example

$$\begin{array}{c}
 \text{维度0} \quad \text{维度1} \quad \text{维度2} \quad \text{维度3} \\
 \text{Are} \quad [0.4002 \quad 0.9882 \quad 0.6261 \quad 0.1502] \\
 \text{you} \quad [0.3413 \quad 0.9284 \quad 0.9349 \quad 0.8958] \\
 \text{OK} \quad [0.5347 \quad 0.2969 \quad 0.7019 \quad 0.6293] \\
 ? \quad [0.8704 \quad 0.4956 \quad 0.3794 \quad 0.5646]
 \end{array}
 +
 \begin{array}{c}
 \text{维度0} \quad \text{维度1} \quad \text{维度2} \quad \text{维度3} \\
 \left[\begin{array}{cccc}
 \sin\left(\frac{0}{10000^{0/4}}\right) & \cos\left(\frac{0}{10000^{0/4}}\right) & \sin\left(\frac{0}{10000^{2/4}}\right) & \cos\left(\frac{0}{10000^{2/4}}\right) \\
 \sin\left(\frac{1}{10000^{0/4}}\right) & \cos\left(\frac{1}{10000^{0/4}}\right) & \sin\left(\frac{1}{10000^{2/4}}\right) & \cos\left(\frac{1}{10000^{2/4}}\right) \\
 \sin\left(\frac{2}{10000^{0/4}}\right) & \cos\left(\frac{2}{10000^{0/4}}\right) & \sin\left(\frac{2}{10000^{2/4}}\right) & \cos\left(\frac{2}{10000^{2/4}}\right) \\
 \sin\left(\frac{3}{10000^{0/4}}\right) & \cos\left(\frac{3}{10000^{0/4}}\right) & \sin\left(\frac{3}{10000^{2/4}}\right) & \cos\left(\frac{3}{10000^{2/4}}\right)
 \end{array} \right]
 \end{array}$$

词向量矩阵

4*4的位置编码矩阵

小问题：为什么可以直接相加？



• Transformer的绝对位置编码设计

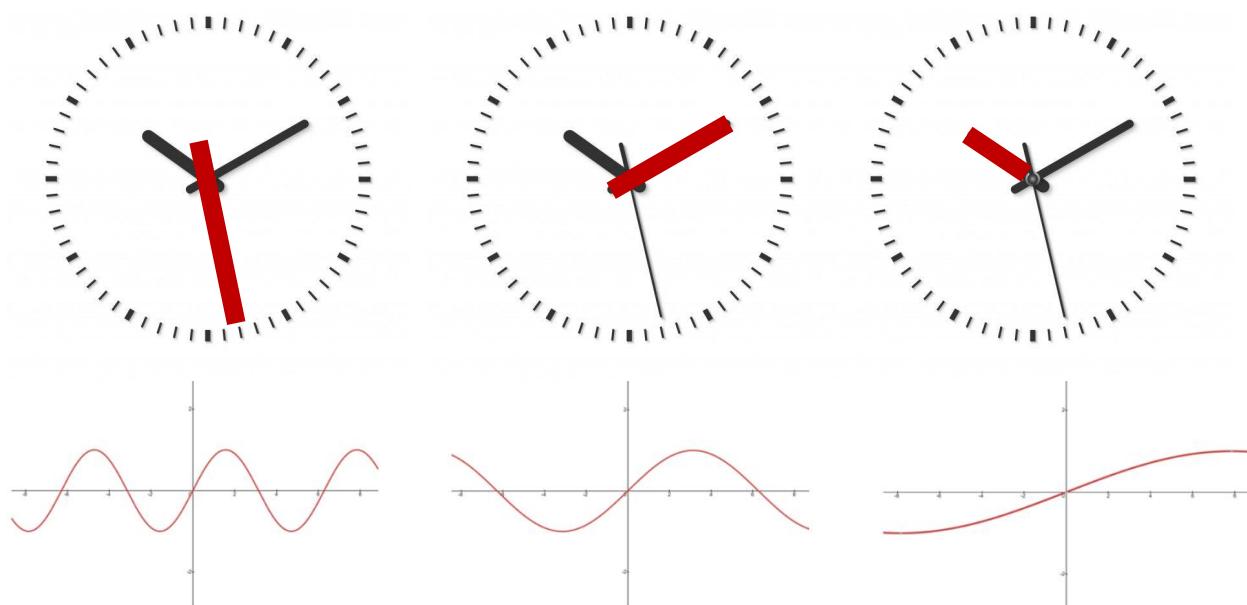
1. 直接编码

$$f_{t:t \in \{q,k,v\}}(\mathbf{x}_i, i) := \mathbf{W}_{t:t \in \{q,k,v\}}(\mathbf{x}_i + \mathbf{p}_i)$$

←绝对位置

这样的方法有长文本外推问题，比如训练数据长度都在500以内，若想让模型最终生成第501编码位置的token就会出问题。

2. 正余弦编码



(a) $y = \sin(x)$

(b) $y = \sin(0.5x)$

(c) $y = \sin(0.2x)$

$$PE_{(pos,2i)} = \sin\left(\frac{pos}{10000^{2i/d_{model}}}\right)$$

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

利用不同频率的正弦和余弦函数，为每个位置生成一个独特的位置向量。这样就能生成。

但是：没有相对位置信息

• LLaMA和ERNIE的相对位置编码设计

3. 旋转位置编码

$$q = [q_1, q_2]$$

$$\text{attention}(q, k) = q \cdot k = qk^T$$

$$k = [k_1, k_2]$$

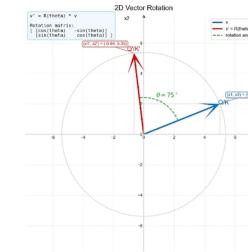
没有位置信息

天才苏剑林发明了旋转矩阵法：设计了一个旋转矩阵。使得向量旋转一个与它的位置 m 成比例的角度。

$$\text{Rotary}(x_0, x_1) = \begin{pmatrix} \cos(m\theta) & -\sin(m\theta) \\ \sin(m\theta) & \cos(m\theta) \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \end{pmatrix}$$

θ 是可学习的参数， m 是位置索引

$$R(\theta) = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix}$$

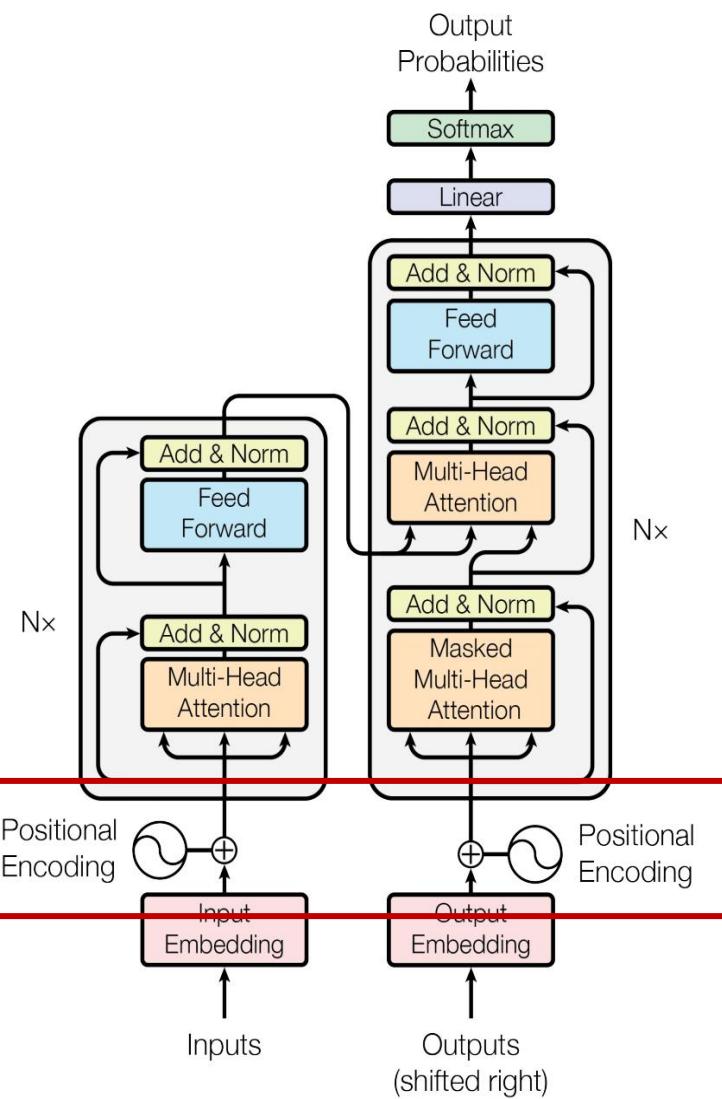


旋转后再点乘：

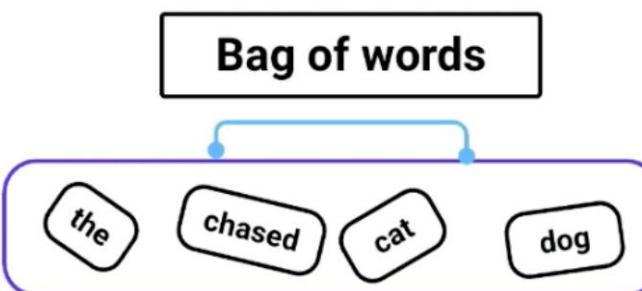
带上了相对位置信息：第M个和第N个这样的绝对信息转化为了(M-N)这样的相对位置信息

$$\begin{aligned} qR(m) \cdot kR(n) &= qR(m)R(n)^T k^T \\ &= qR(m)R(-n)k^T \\ &= qR(m-n)k^T \end{aligned}$$

• 小结：位置编码的进化



- The dog chased the cat
- The cat chased the dog



1. 直接编码

长文本外推 (✗)
相对位置信息 (✗)

2. 正余弦编码

长文本外推 (✓)
相对位置信息 (✗)

3. 旋转位置编码

长文本外推 (✓)
相对位置信息 (✓)



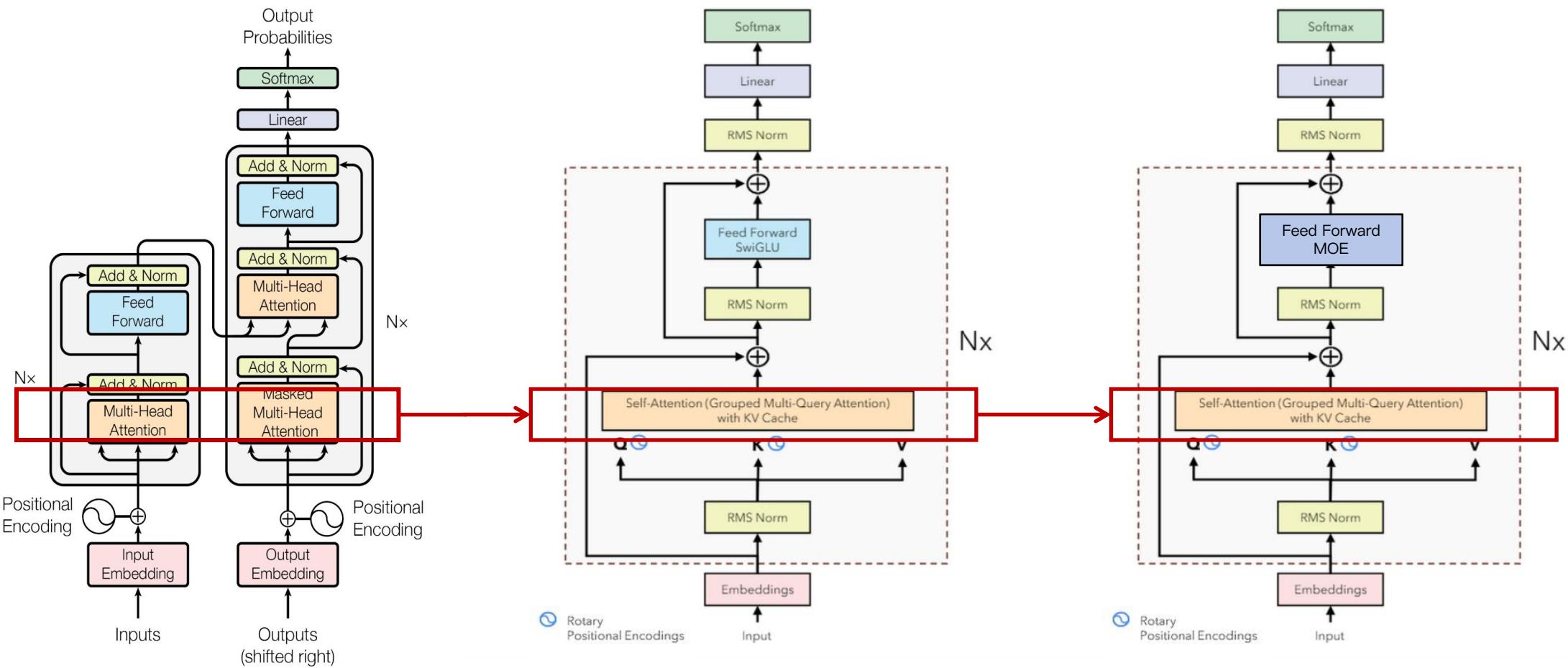
LLama 3
ERNIE 4.5



• 注意力机制



• 注意力机制 (MHA → GQA)

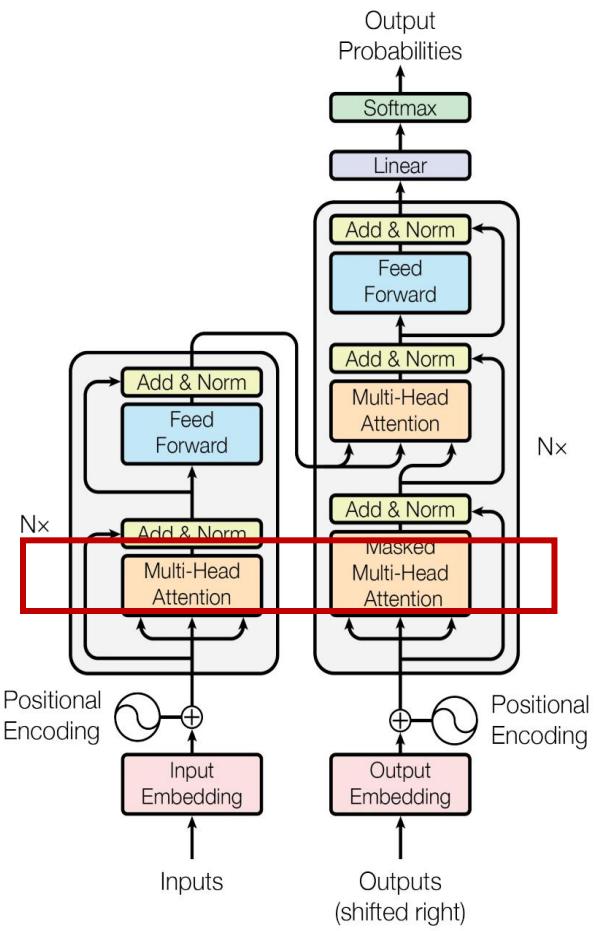


Transformer

LLama 3

ERNIE 4.5

- 注意力机制 (MHA)
- 为什么我们需要多头?

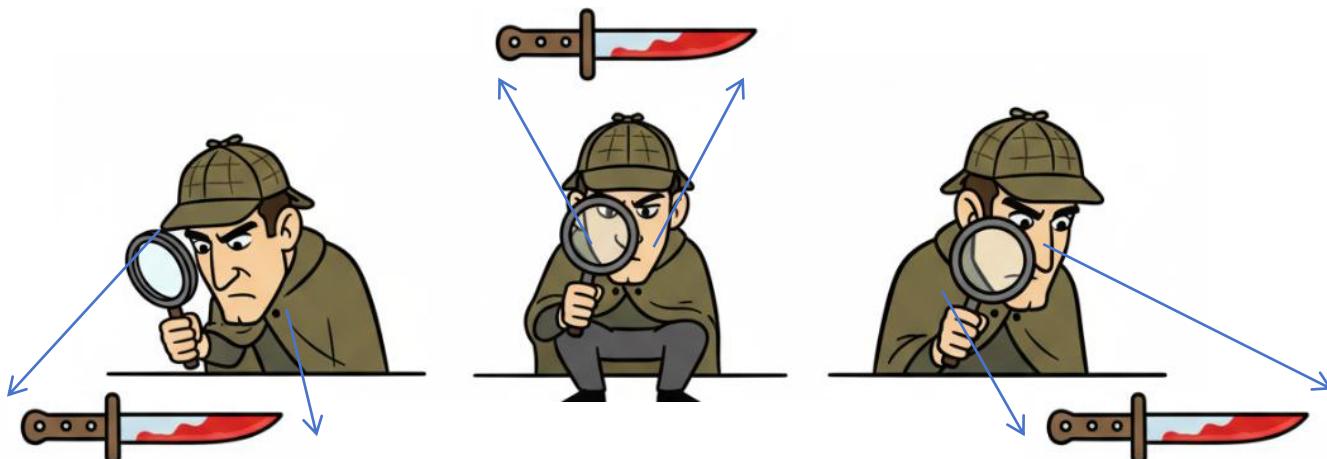


Transformer

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \dots, \text{head}_h)W^O$$

where $\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$

- **多头multi-head**让模型可以并行捕捉不同类型的依赖关系:
 - 语法结构?
 - 语义关系?
 - 长距离依赖?
 - 类似于：“多个观察视角”



- 多头的问题

如果生成一个长度为1000的token序列，当我们生成第1000个token时，第一个token的k和v已经被计算了999次



- KV Cache的引入

这造成了巨大的重复计算浪费。解决办法就是把以前计算好的k和v缓存起来，以供后续调用

这就是**KV Cache**

小问题：为什么不缓存Q？

- KV Cache的局限性：内存瓶颈

LLM的推理都是在GPU上进行，但是GPU的显存有上限。



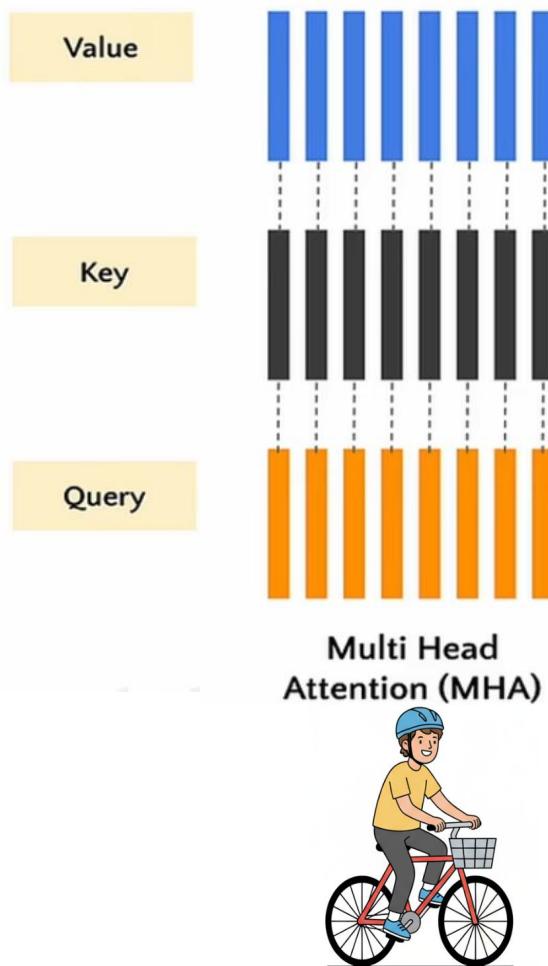
显存

KV Cache ↑↑↑↑↑↑↑

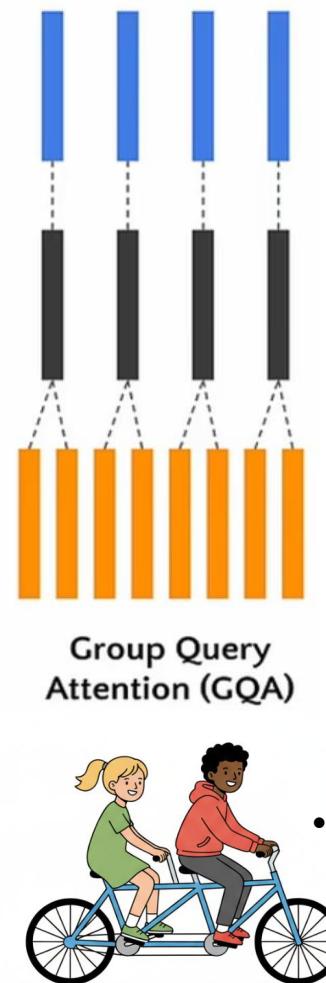
模型的参数+前向计算的激活值

——如何减少KV Cache同时尽可能地保证效果？

- Group Query Attention (GQA)的引入

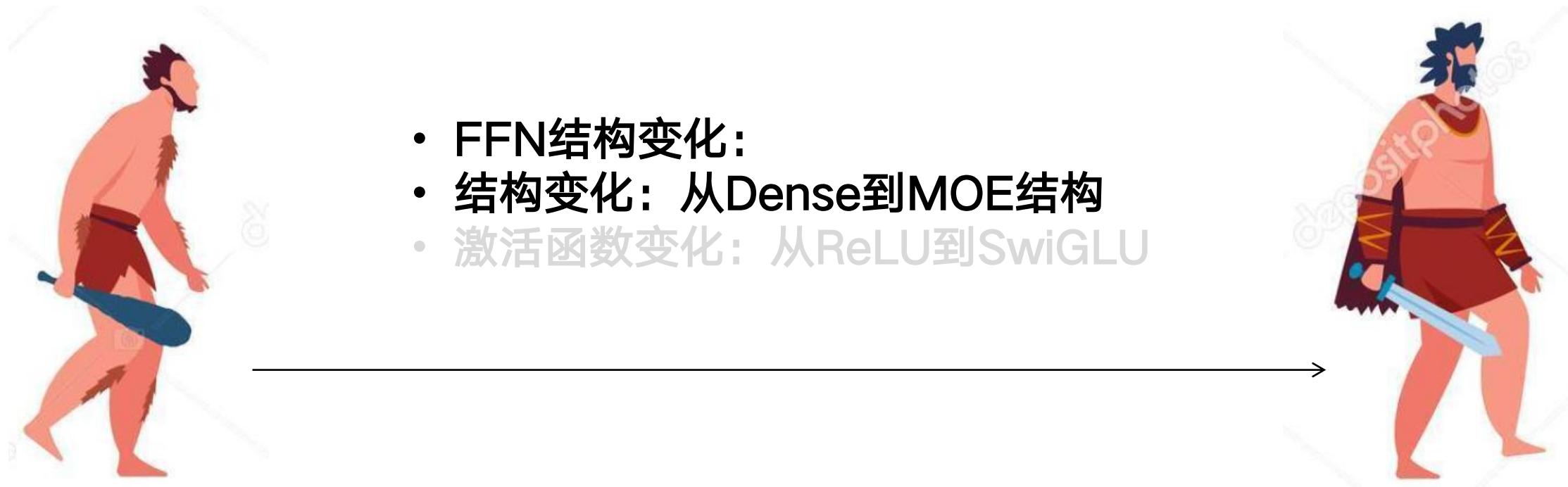


Transformer

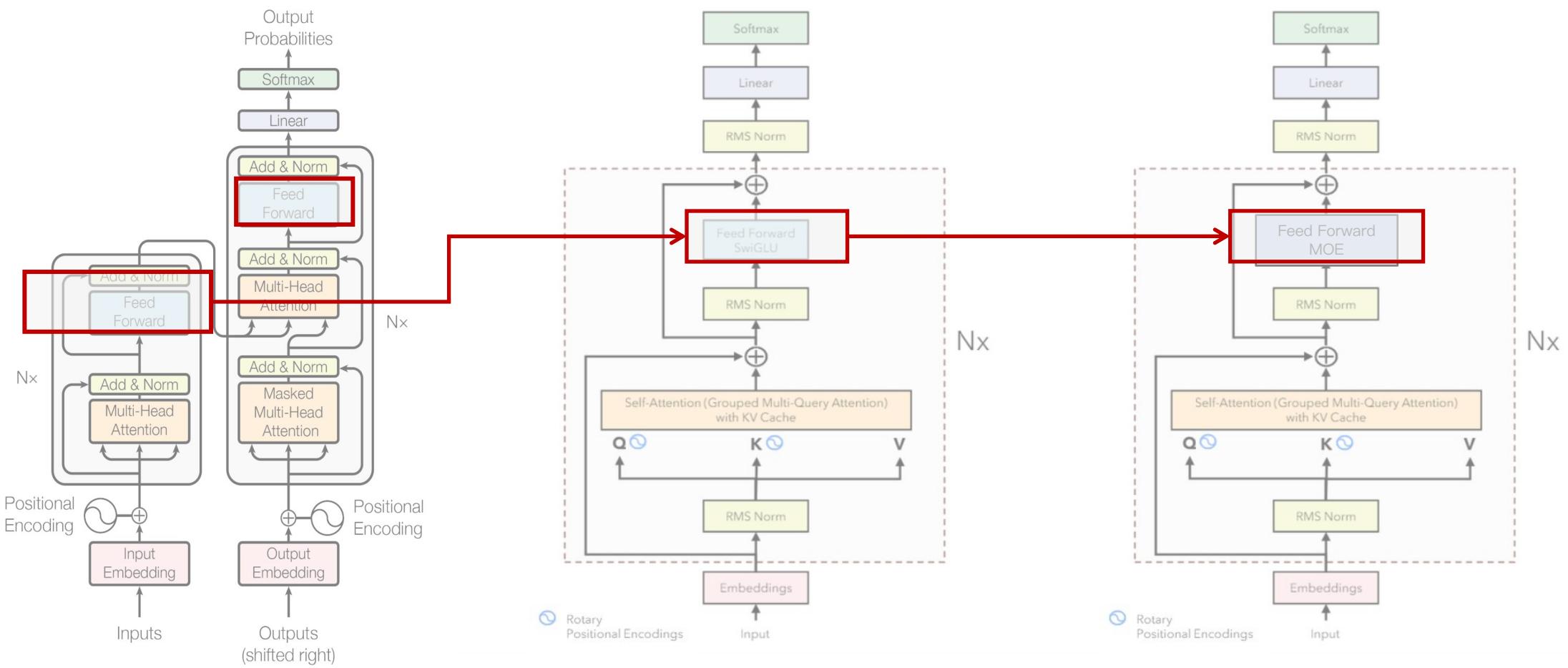


LLama 3
ERNIE 4.5

- ERNIE 4.5系列模型也集成了GQA机制
- ERNIE4.5 300B采用了64头/8组的分组方案，以支持高吞吐量解码和更长上下文推理

- 
- FFN结构变化:
 - 结构变化: 从Dense到MOE结构
 - 激活函数变化: 从ReLU到SwiGLU

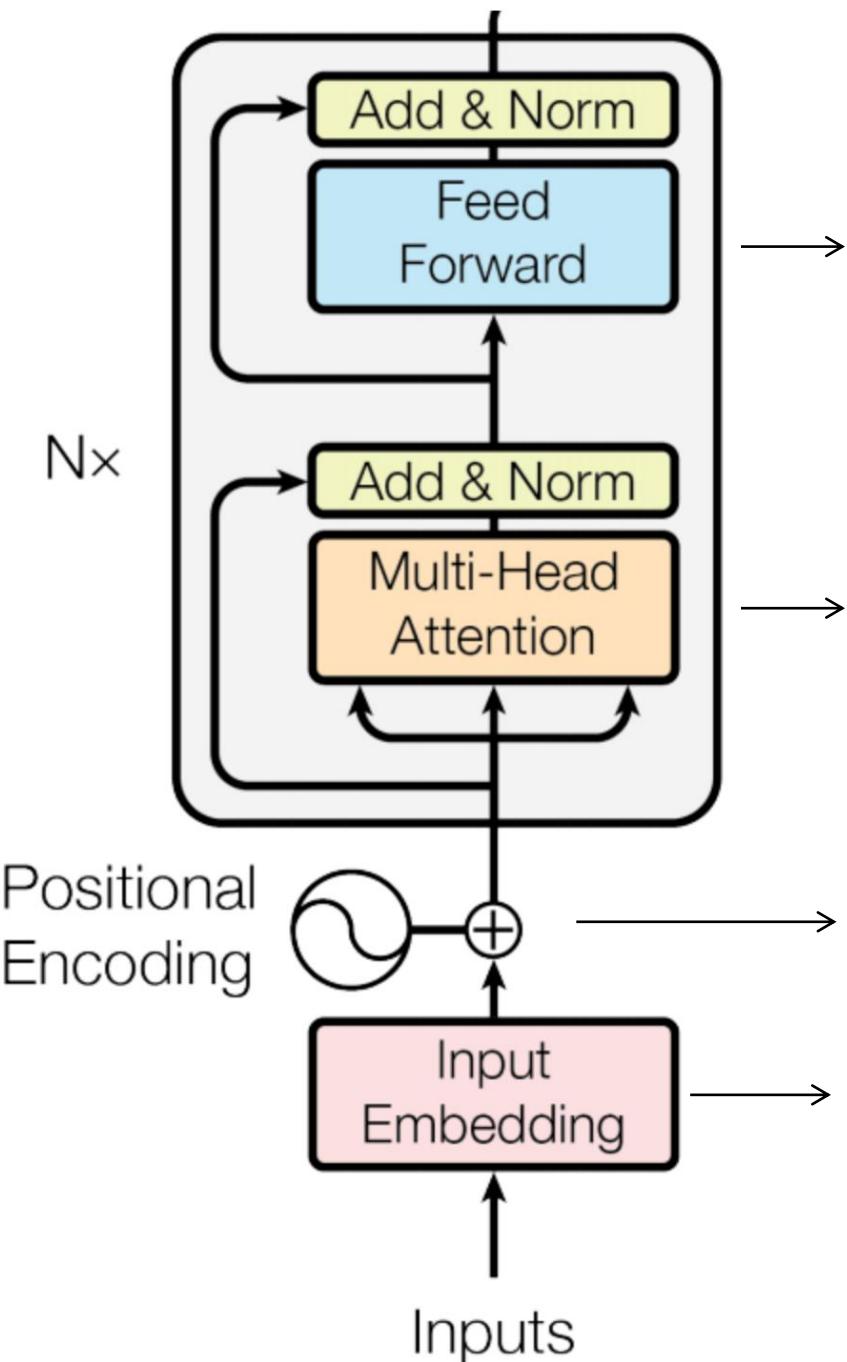
• MOE



Transformer

LLama 3

ERNIE 4.5



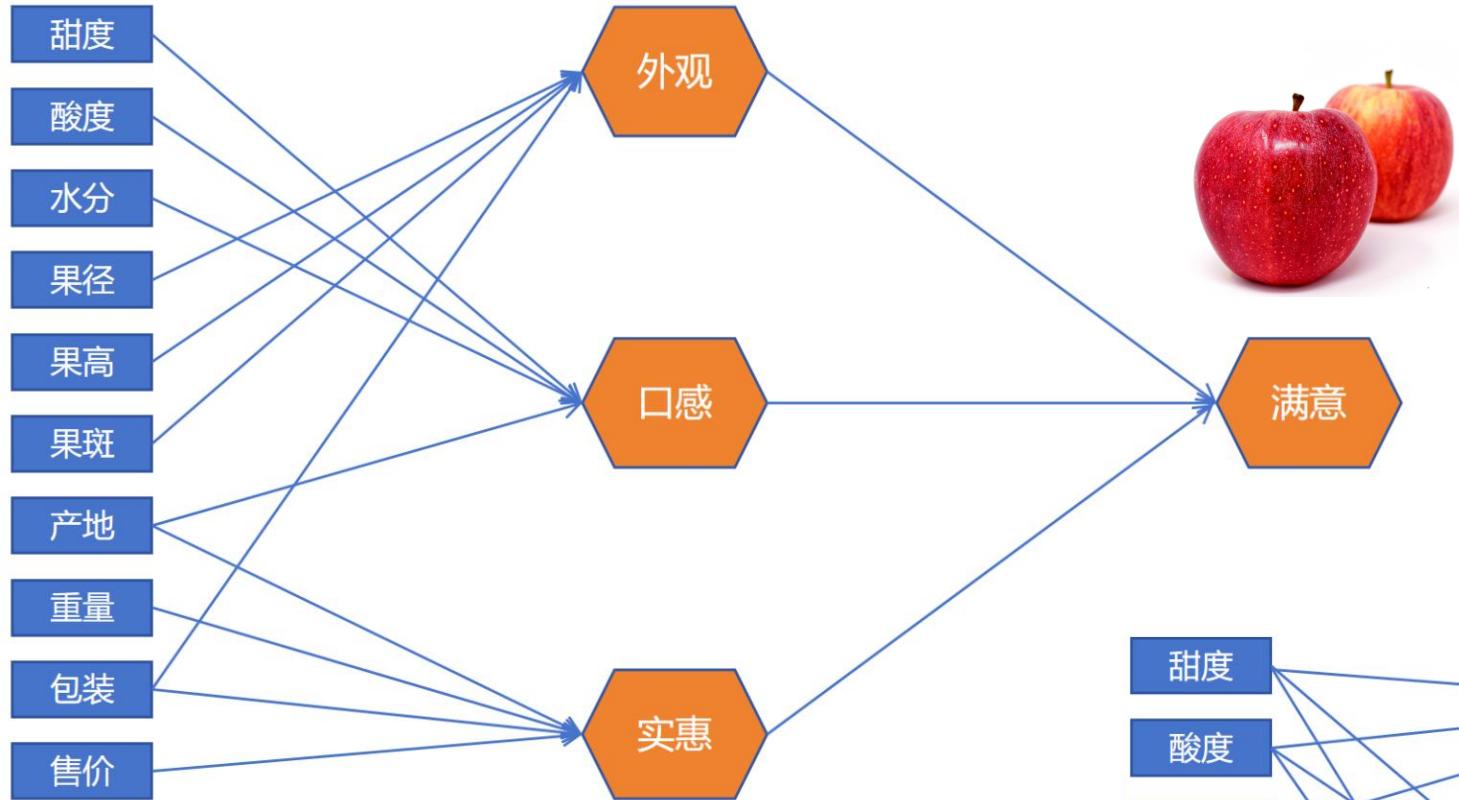
04 前馈层——处理单token的深度特征信息：
对经过注意力层提炼后的每个词的向量进行独立的、深度的非线性加工，以提取更抽象、更精细的特征。

03 注意力——处理token与token之间的关系：
处理当前词时，同时关注（查阅）序列中的所有其他词，并根据它们的关联程度分配不同的重要性。

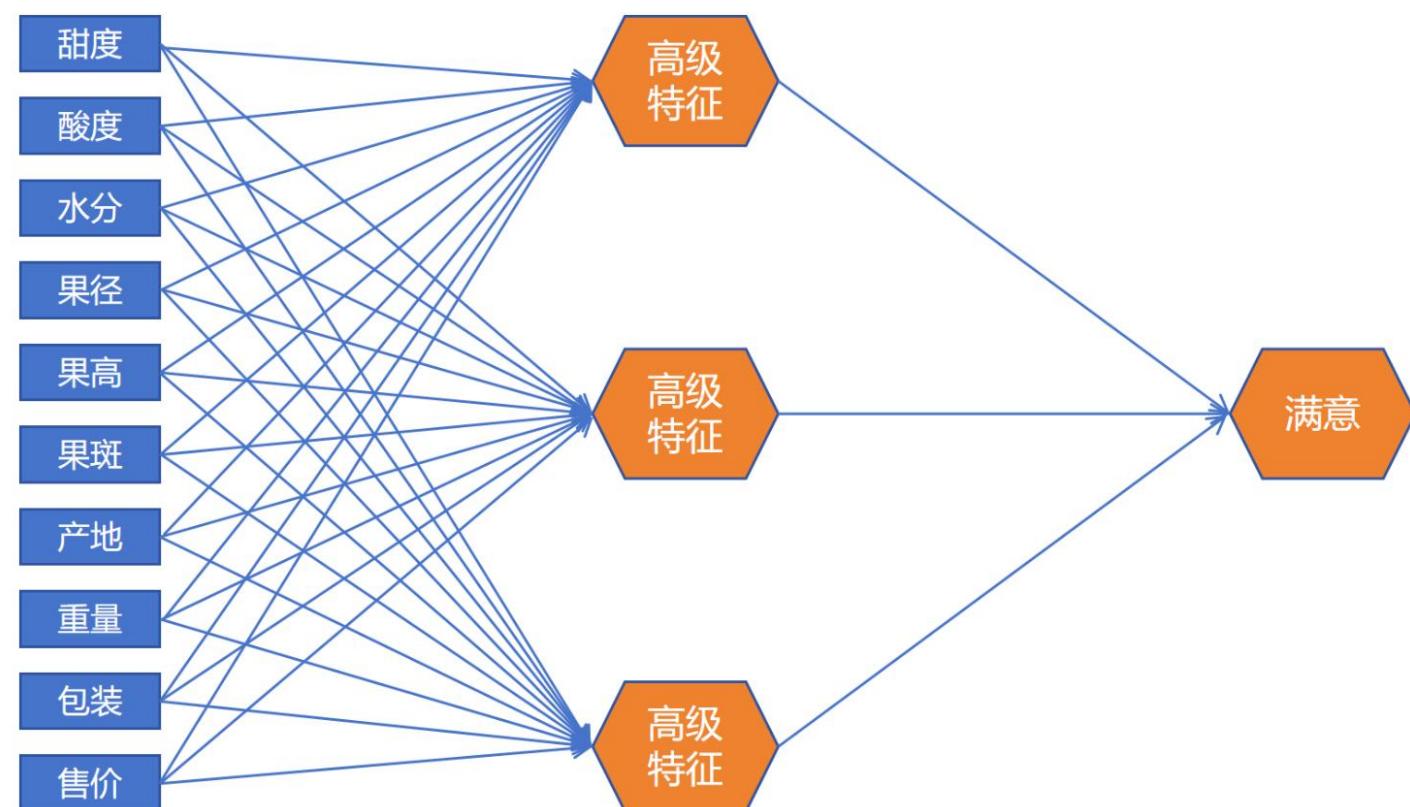
02 位置编码——添加词在句中位置信息：
为每个向量添加一个表示其在序列中位置的信息。

01 嵌入——词转化成机器语言：
将输入的原始数据（如文字）转化为模型能理解的数值向量。

• 从逻辑回归到神经网络



1. 不人为抽象高级特征，让模型在训练过程中自己抽象高级特征。
2. 将所有的原始特征接入到每个提取高级特征的逻辑回归模型，让模型学习使用哪些特征，模型可以通过训练将不重要的特征权重调整为0。
3. 训练时，给模型只提供原始的特征，和最终顾客是否满意的label。虽然整个系统由多个逻辑回归构成，但训练时当做一个完整的模型。



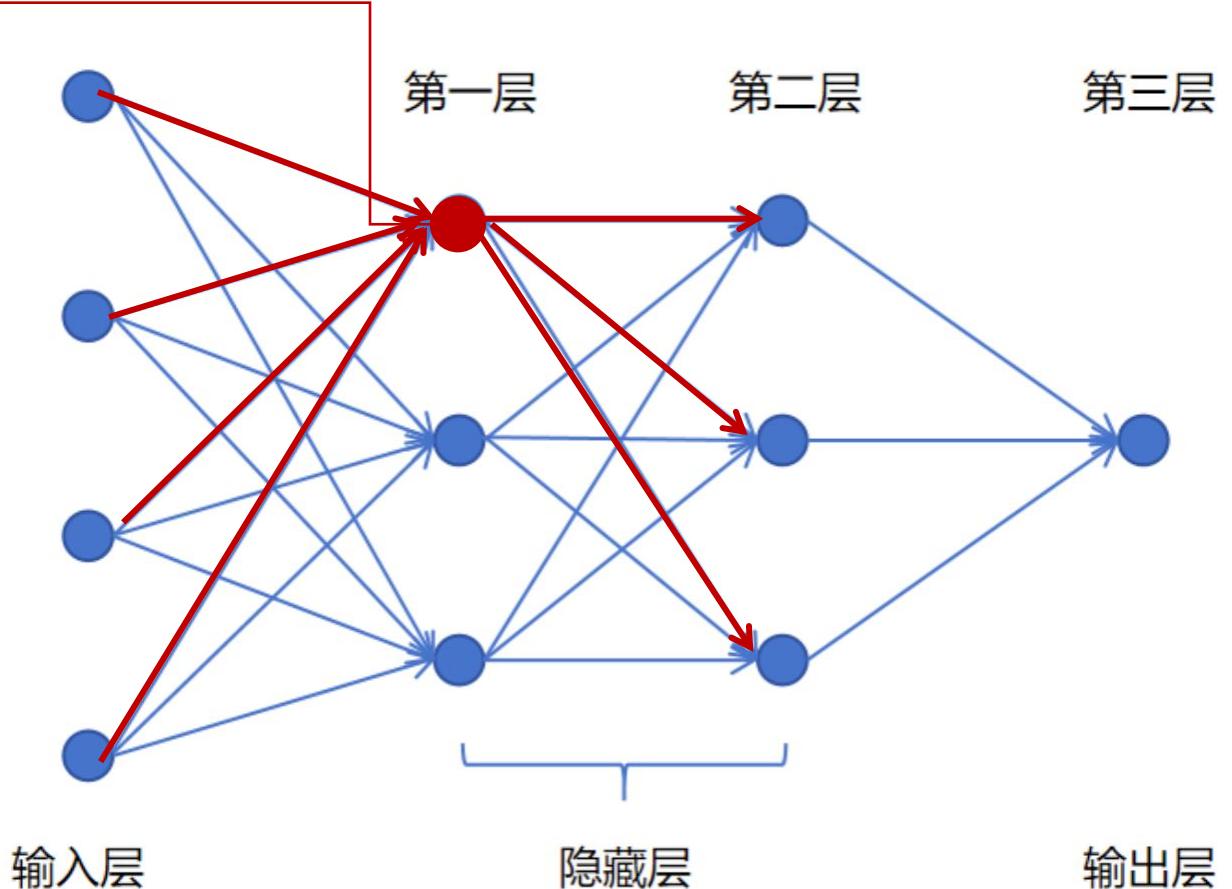
- 前馈神经网络是怎么发挥作用的？

$$a = \text{Activation}(z) \dots \dots \quad (2)$$

一个神经元里有啥？

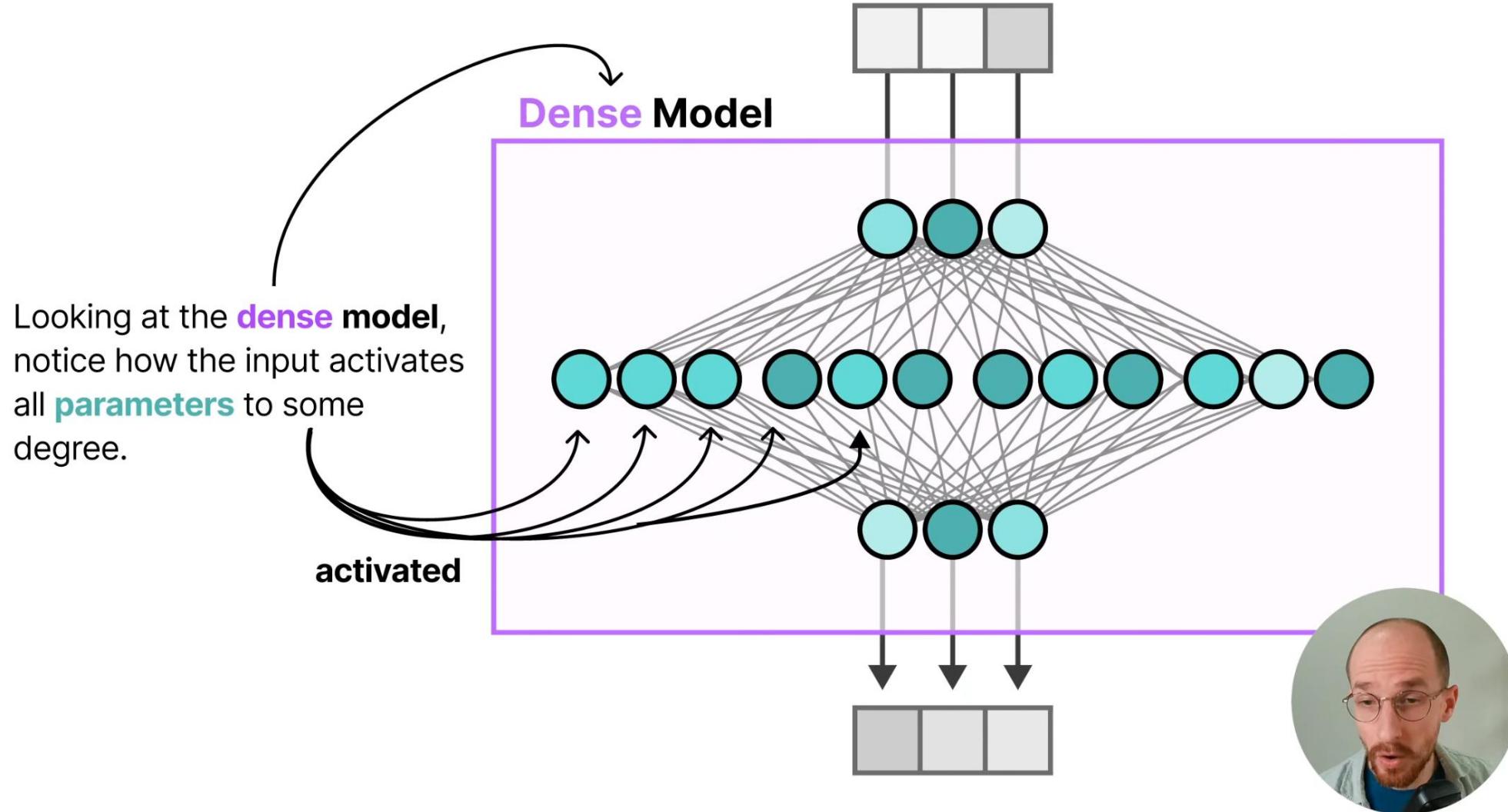
- (1) 1个线性变换函数
 - (2) 1个非线性激活函数
如ReLU、Sigmoid、Tanh

多层多个神经元的网络，能够拟合复杂特征



- 神经网络中的一层我们叫做线性层（Linear Layer），因为其中每个神经元都是一个线性回归+激活函数。
 - 或者叫做全连接层、稠密层（Dense Layer），因为每一层的每个神经元都和前一层的每个神经元进行连接。

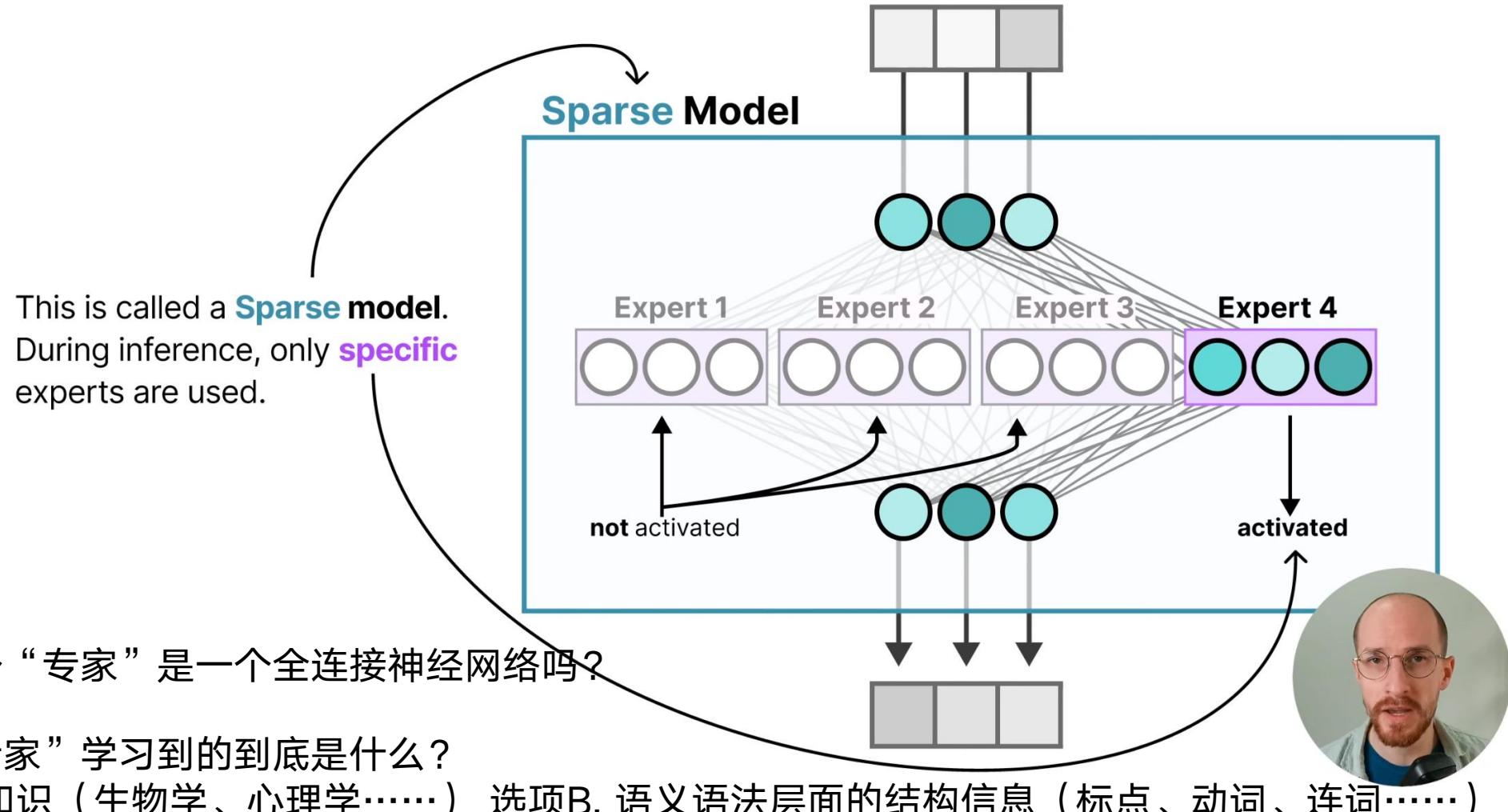
- 全连接结构前馈神经网络的问题



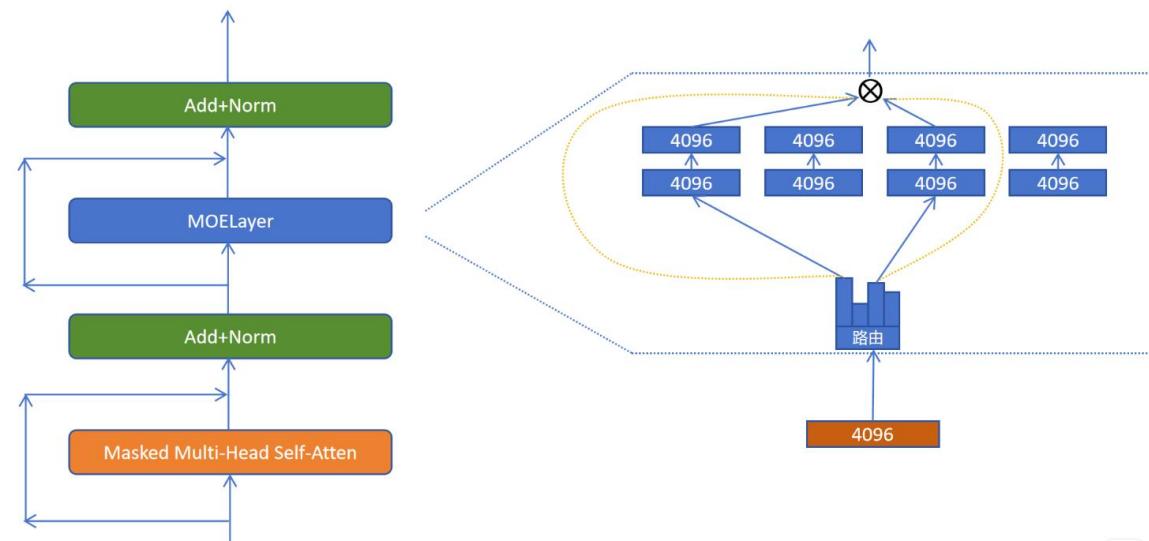
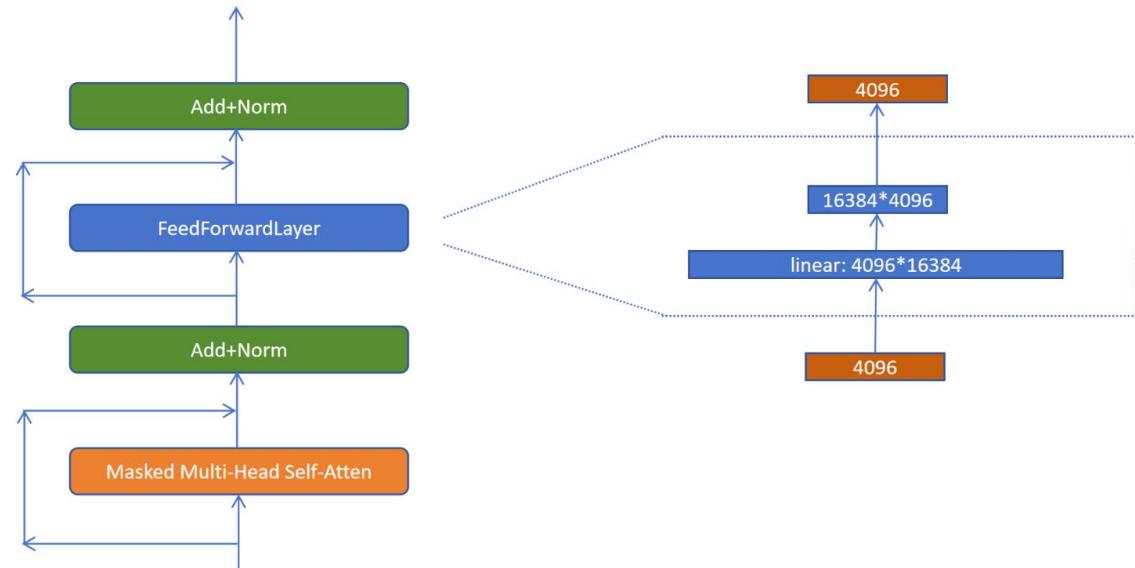
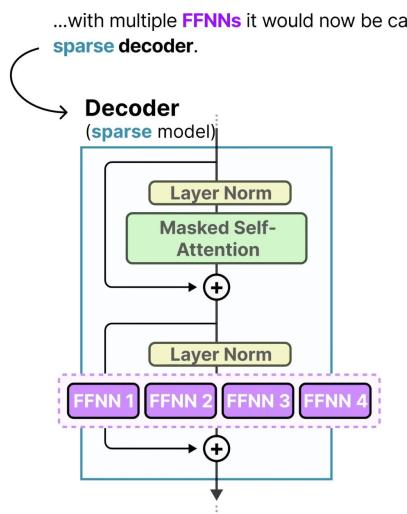
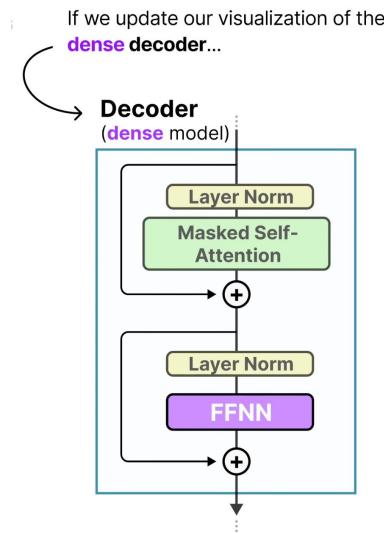
小问题1：ERNIE 0.3b模型和ERNIE 424b模型中「假设他们都是稠密模型（dense结构）」，
Self-attention部分和Feed Forward层（FFN）和的计算量哪部分更大？

• MOE结构前馈神经网络的引入

- 每层包含 若干专家网络（例如 64 个专家 FFN）， 推理时通过门控机制（Gating Network）选择其中 少量（如 2 个）专家执行。
- 这样参数规模极大（数百亿/千亿级）， 但每个 token 实际只激活部分专家， 计算量大大减少。



- 对比Dense和MOE结构：直观的计算量减少



- 
- 
- FFN结构变化:
 - 结构变化: 从Dense到MOE结构
 - 激活函数变化: 从ReLU到SwiGLU

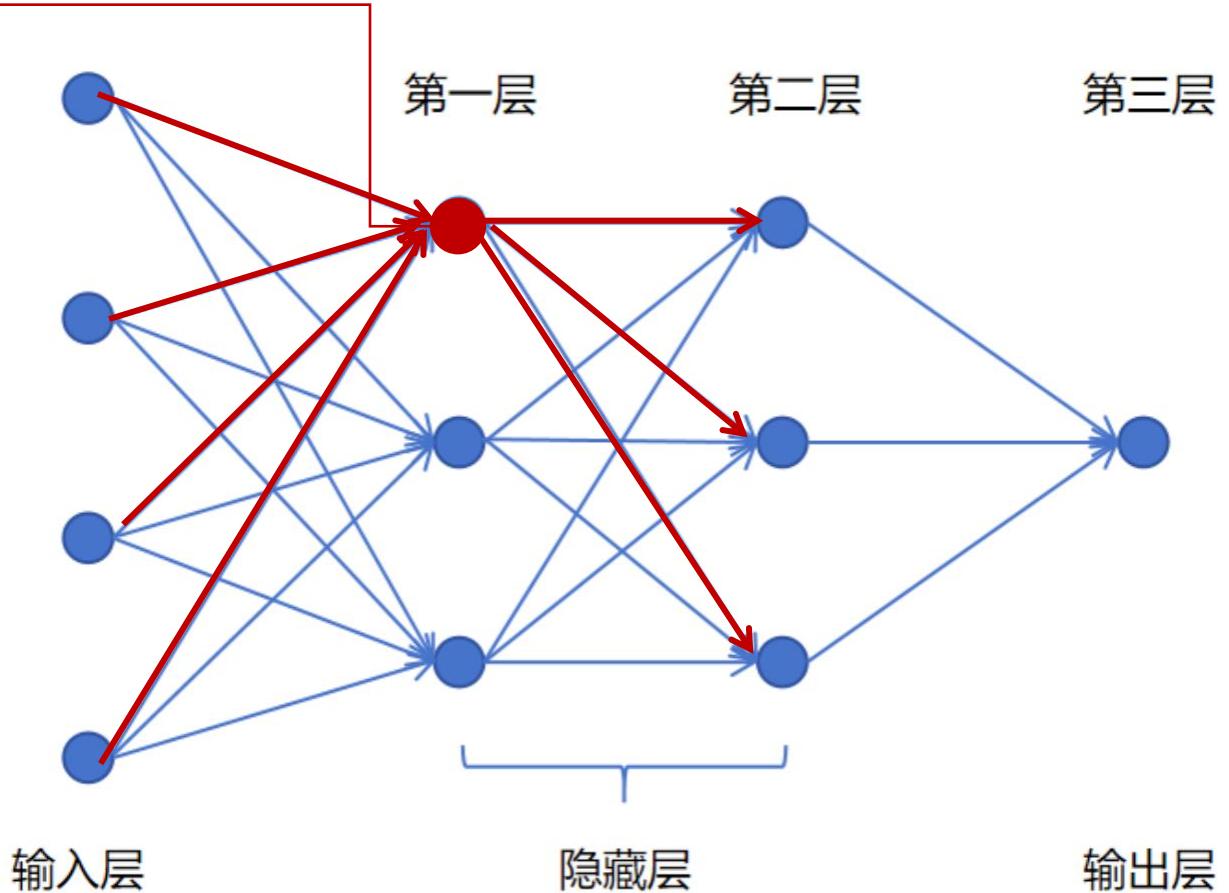
- 前馈神经网络是怎么发挥作用的？

$$a = \text{Activation}(z) \dots \dots \quad (2)$$

一个神经元里有啥？

- (1) 1个线性变换函数
 - (2) 1个非线性激活函数
如ReLU、Sigmoid、Tanh

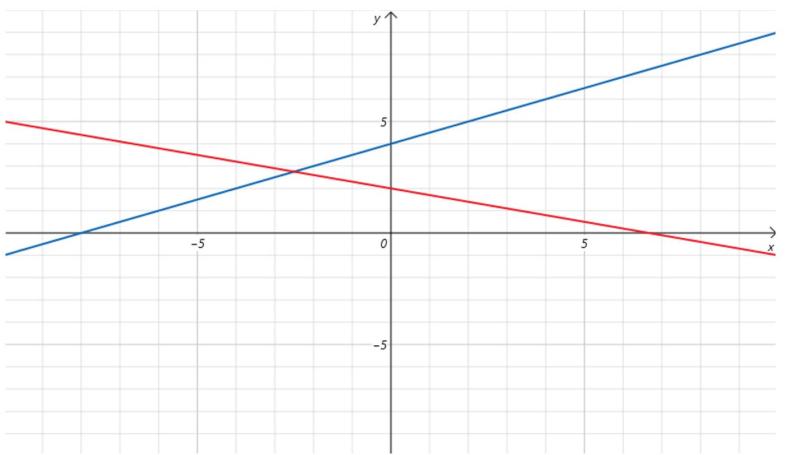
多层多个神经元的网络，能够拟合复杂的特征



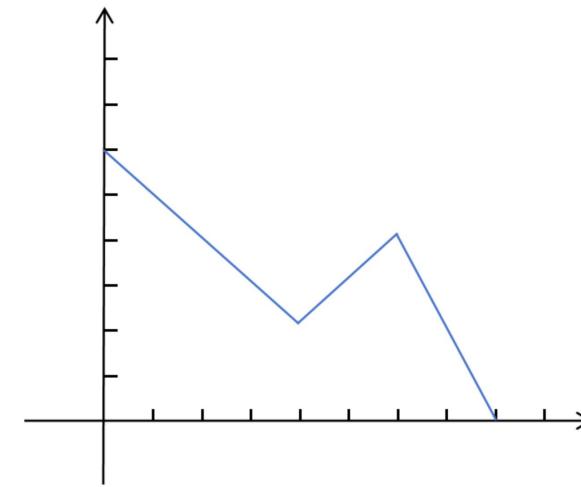
- 为什么需要激活函数？

- 引入**非线性**的能力，让网络能够学会复杂的任务。
 - ——打破“**线性变换叠加仍是线性**”的局限，让神经网络能拟合复杂的非线性关系（比如图像、语言中的规律）

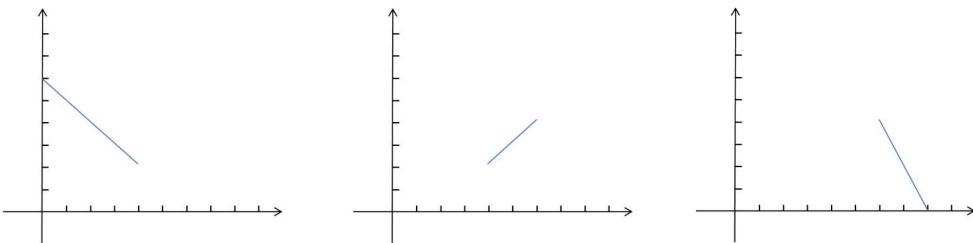
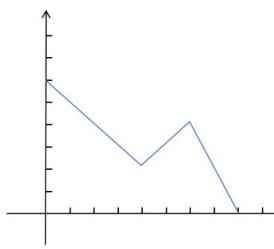
- 为什么神经网络引入非线性激活函数后，可以拟合任意函数？



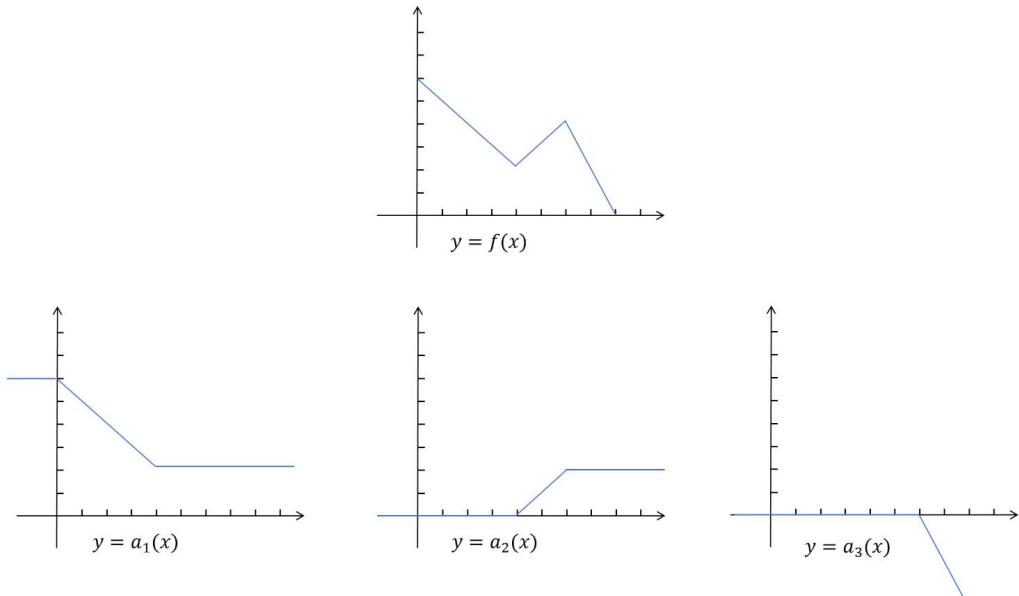
01 $y=wx+b$ 可以表达二维平面上的任何一条直线。



02 那如果是这样的呢？如何拟合？

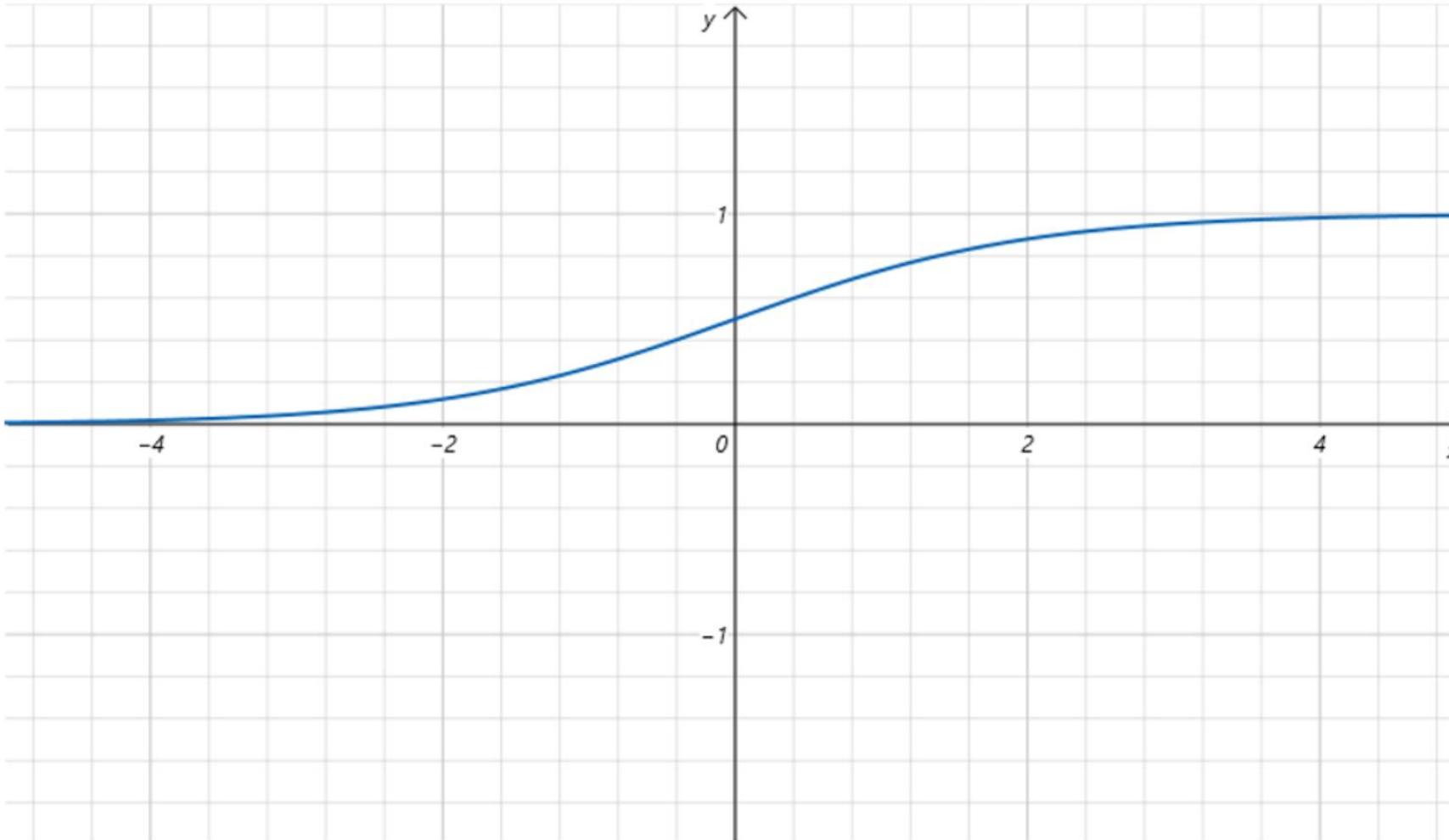


03 很明显上边的折线分为3段，我们可以将它拆成3段分别拟合。



04 用直线补齐折线的其他定义域，然使起始值为0。

- 为什么神经网络引入非线性激活函数后，可以拟合任意函数？



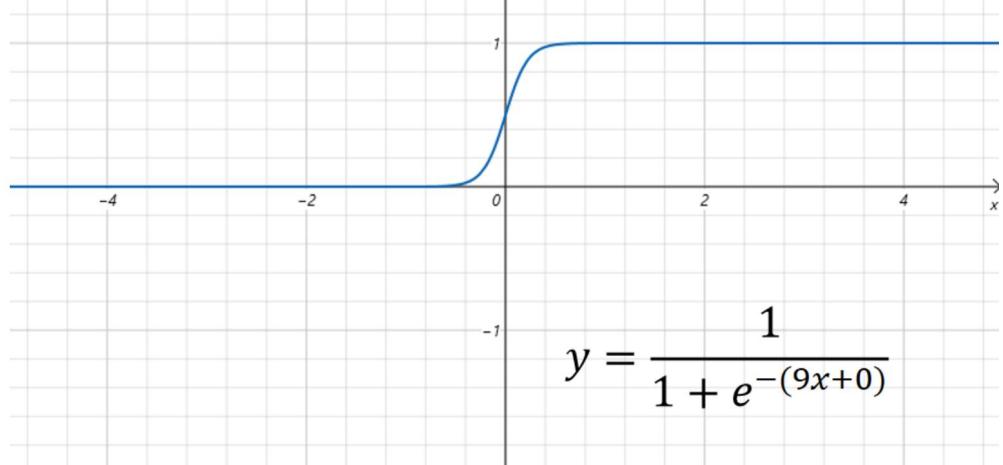
$$Sigmoid(x) = \frac{1}{1+e^{-x}}$$

我们来拿sigmoid函数跟它比划比划。

• 为什么神经网络引入非线性激活函数后，可以拟合任意函数？

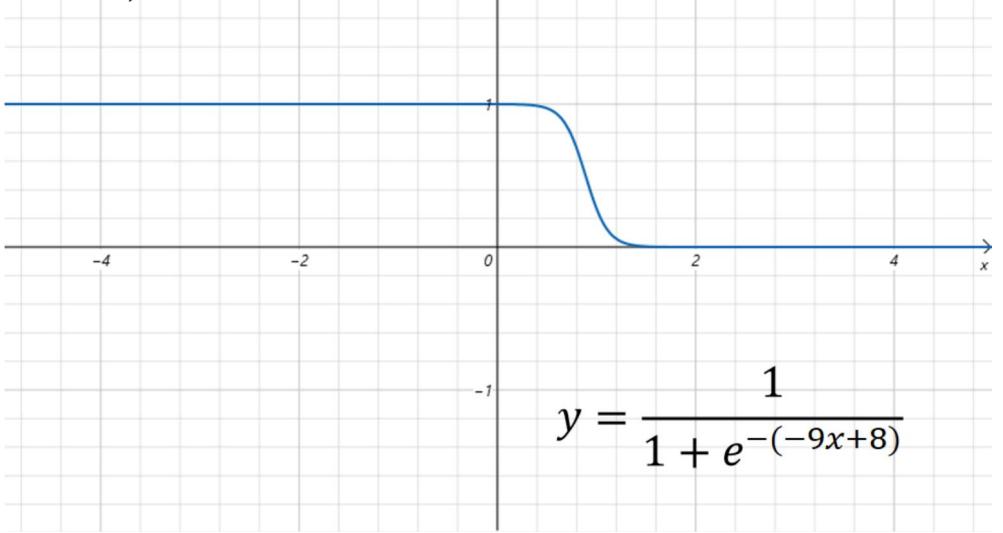
01 比如我们设置 $w=9, b=0$, 图像为↓

它对sigmoid函数沿着x轴方向进行了压缩，看着更像一个阶梯函数了。



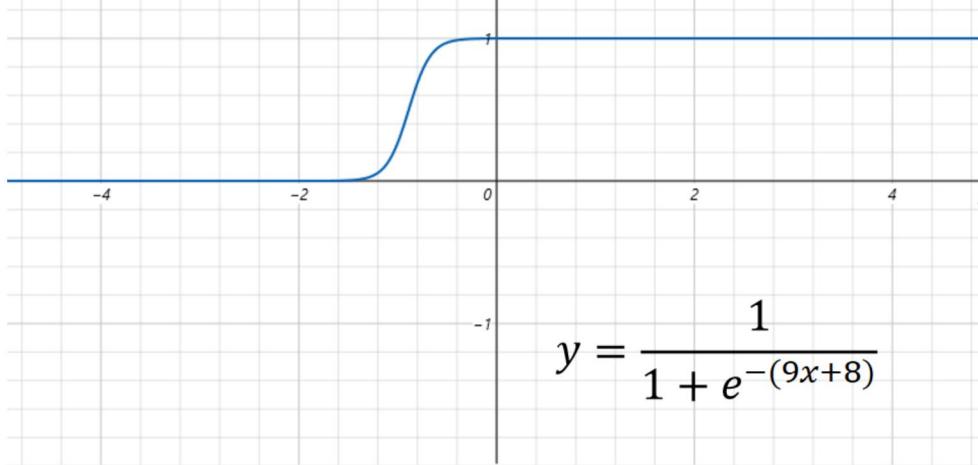
02 接着我们设置 $w=-9, b=8$, 图像为↓

卧槽，图像在上一步基础上沿x轴进行了翻转。



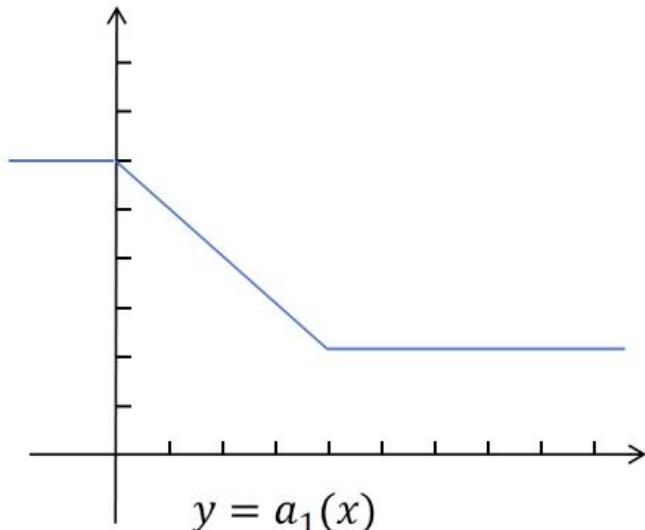
03 我们设置 $w=-9, b=8$, 图像为↓

可以看到函数图像在之前基础上沿x轴方向进行了平移。

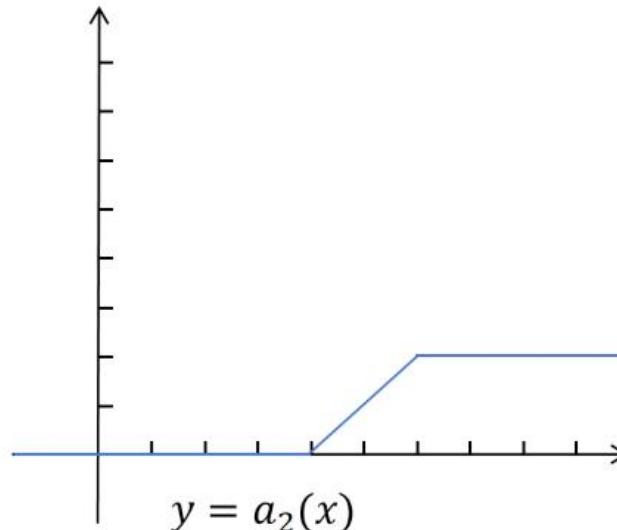


04 经过上边的变化，我们发现通过**调整w,b**可以让一个Sigmoid的阶梯图像沿着x轴进行**平移，翻转，伸缩！**

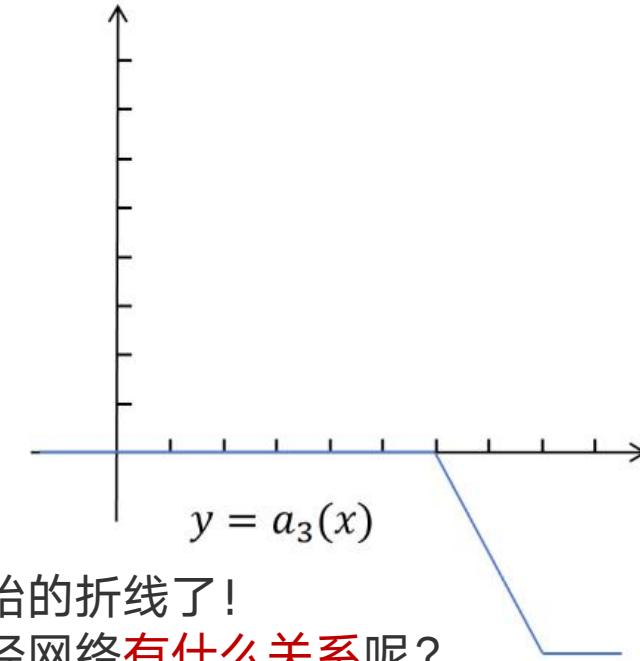
- 为什么神经网络引入非线性激活函数后，可以拟合任意函数？



$$y = a_1(x)$$



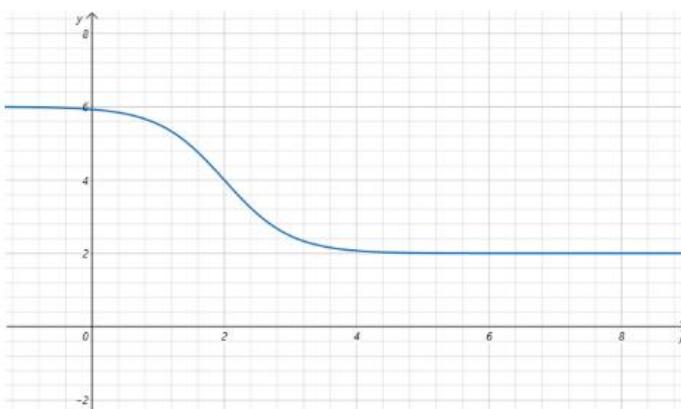
$$y = a_2(x)$$



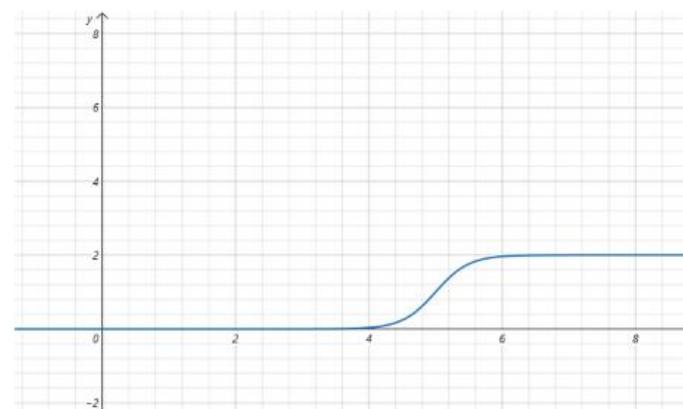
$$y = a_3(x)$$

这样我们就可以用这三个Sigmoid拟合的阶梯函数的加和来拟合原始的折线了！

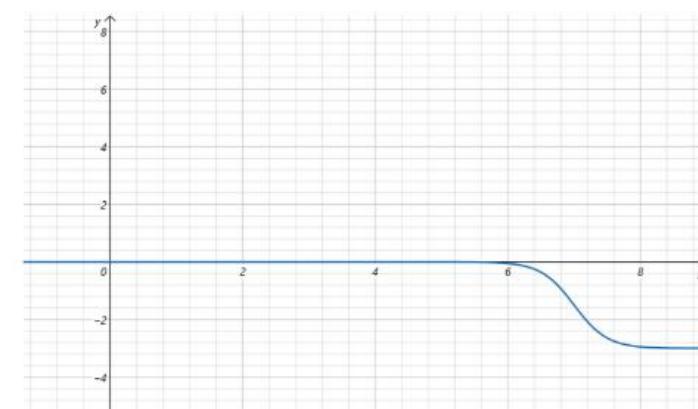
我们是用**多个带参数的Sigmoid函数拟合了一个折线**，但是这和神经网络**有什么关系呢**？



$$y = 4 \frac{1}{1 + e^{-(2x+4)}} - 2$$



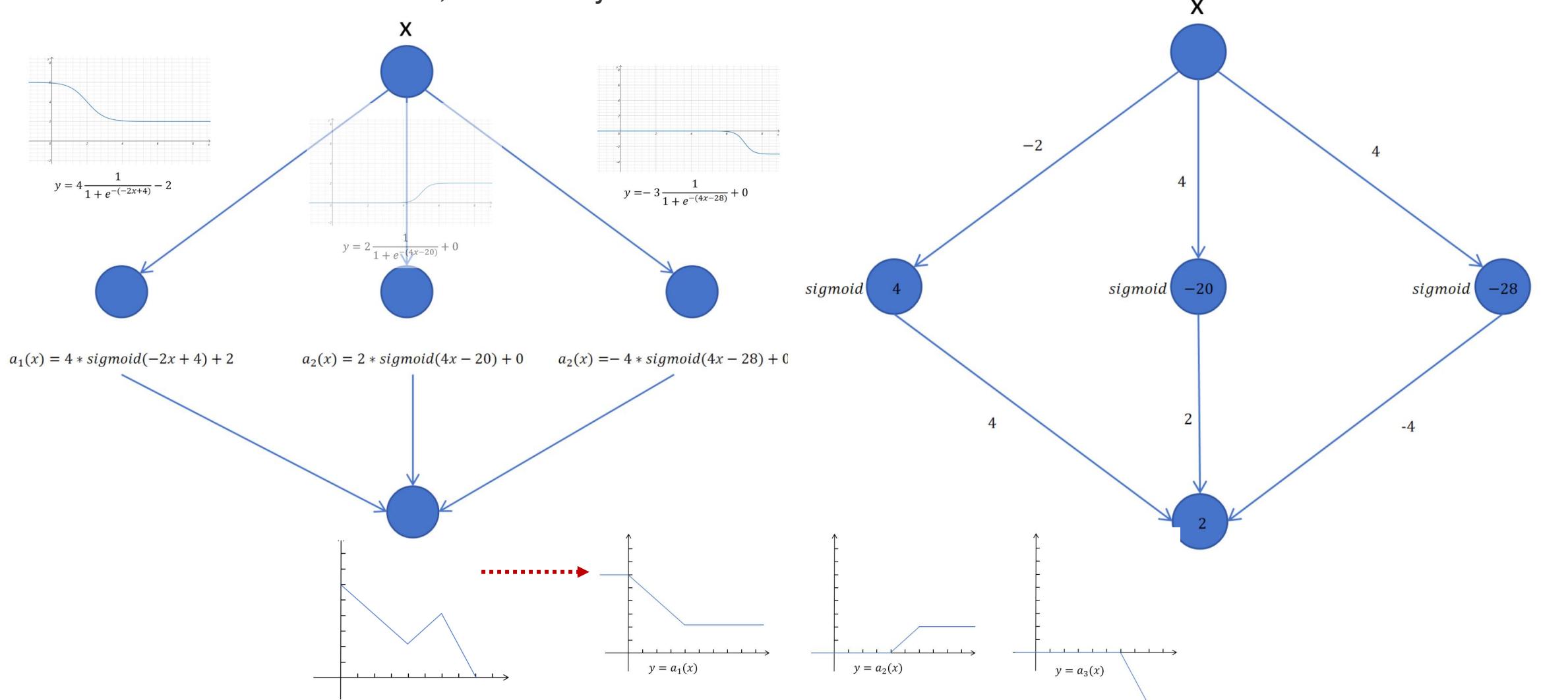
$$y = 2 \frac{1}{1 + e^{-(4x-20)}} + 0$$



$$y = -3 \frac{1}{1 + e^{-(4x-28)}} + 0$$

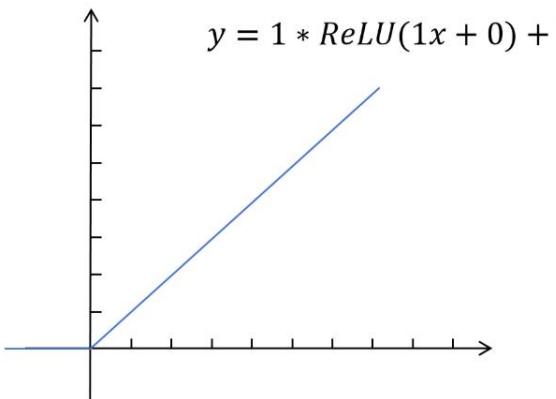
• 为什么神经网络引入非线性激活函数后，可以拟合任意函数？

- 实际上3个阶梯函数就代表3个隐藏层的神经元。
- 小写的w,b是隐藏层的权重和偏置。（大写的W,B是输出层的权重和偏置。）
- Sigmoid就是每个神经元的激活函数。
- 这个神经网络只有一个输入x，一个输出y。

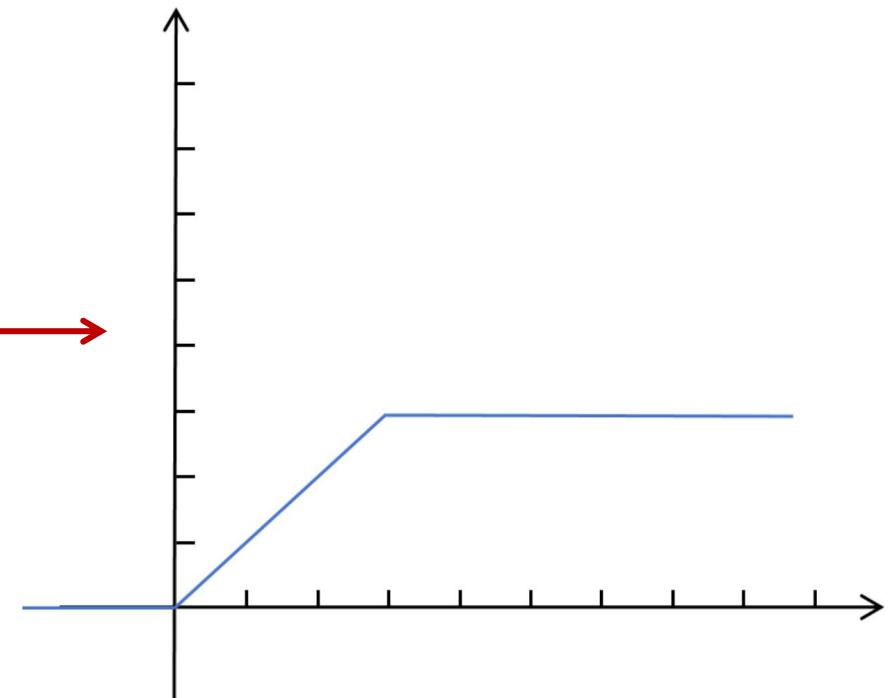
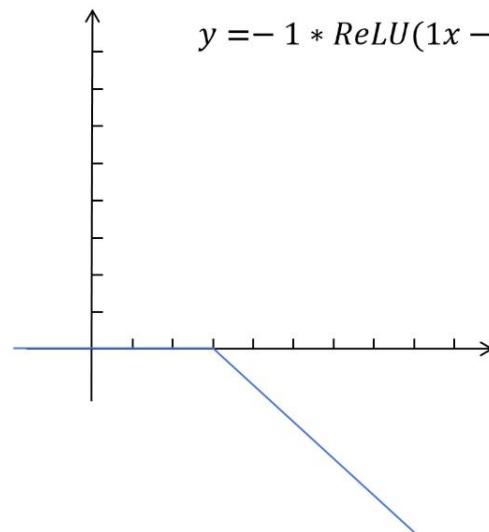


- 为什么神经网络引入非线性激活函数后，可以拟合任意函数？
- 如果激活函数选择ReLU该怎么拟合呢？Sigmoid函数或者tanh函数本来长得就像阶梯函数，但是ReLU怎么拟合阶梯函数呢？
- ——答案是用两个ReLU函数的和来构成一个阶梯函数。

$$y = W * \text{ReLU}(wx + b) + B$$



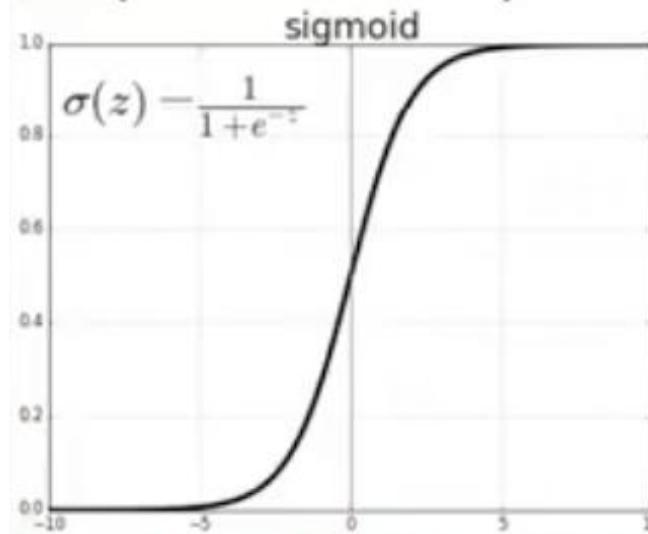
+



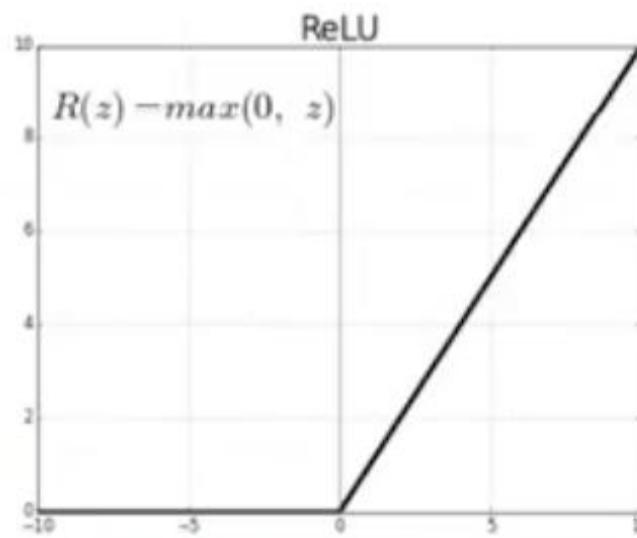
- 用ReLU作为激活函数，也是可以拟合任意函数的。

- 古早的激活函数

ReLU(Rectified Linear Unit)



sigmoid



ReLU

ReLU是目前使用最广泛的激活函数之一。其计算公式如下: $\text{ReLU}(x) = \max(0, x)$

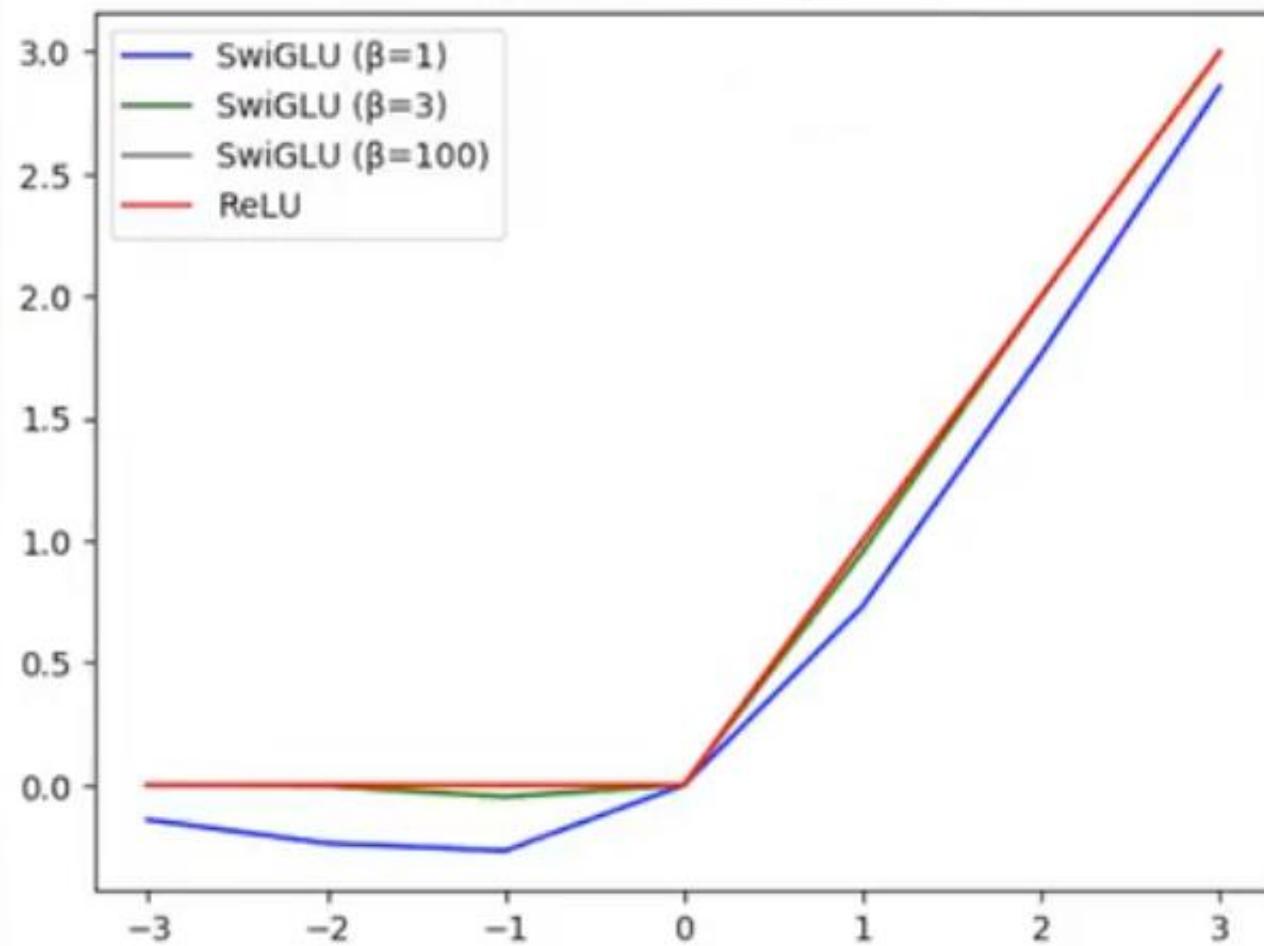
- sigmoid 是早期神经网络的“主力激活函数”（比如 BP 神经网络），但它存在**梯度消失问题**（输入过大或过小时，梯度接近 0，导致深层网络难以训练）。
- ReLU 的出现正是为了缓解这一问题——它在正区间的梯度恒为 1，能有效传递梯度，更适合深层网络（如 CNN、Transformer）。
- 但ReLU存在“**神经元死亡**”问题。

• SwiGLU激活函数

$$\text{SwiGLU}(x) = (\text{Linear}_1(x)) \odot \text{Swish}(\text{Linear}_2(x))$$

SwiGLU 并非一个简单的函数（如 ReLU 或 GELU），而是一种结构。它的核心思想是使用乘法门控（Multiplicative Gating）来动态地控制信息流。

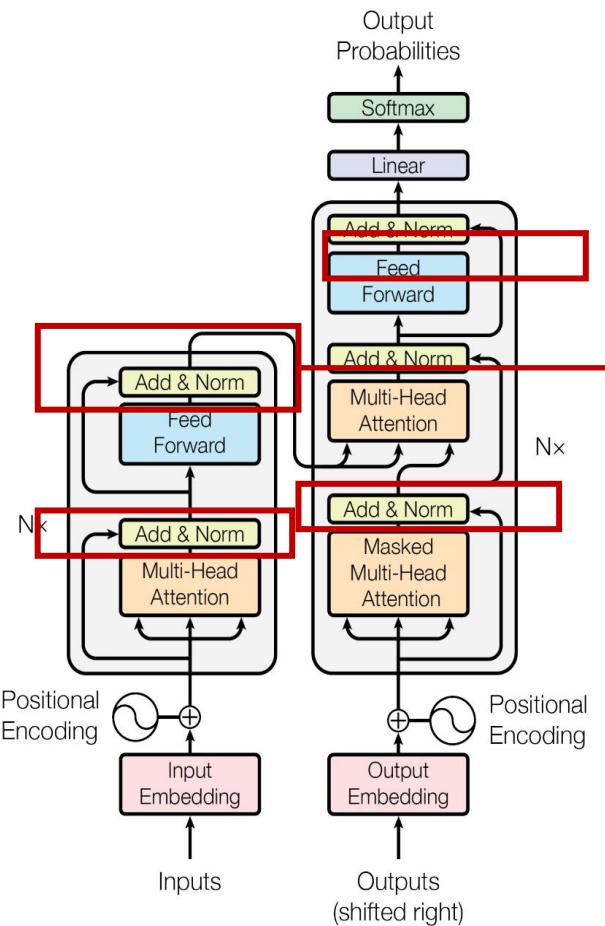
1. 线性路径 (Linear Path): 负责主要的数据转换。
2. 门控路径 (Gating Path): 负责决定多少信息应该通过，它使用 Swish 函数作为非线性激活。
 - 对于一个输入 Token，如果某个特征对当前任务不重要，门控路径就可以输出接近 0 的值将其“关闭”；如果某个特征很重要，门控路径就可以输出接近 1 的值将其“放大”。
 - 一股是“要传递的数据”，另一股是“一个软开关”。
 - 这个“软开关”根据数据本身的内容动态调整，决定了有多少“数据”会被允许通过。这种机制让网络可以**动态地过滤和强调特征**。



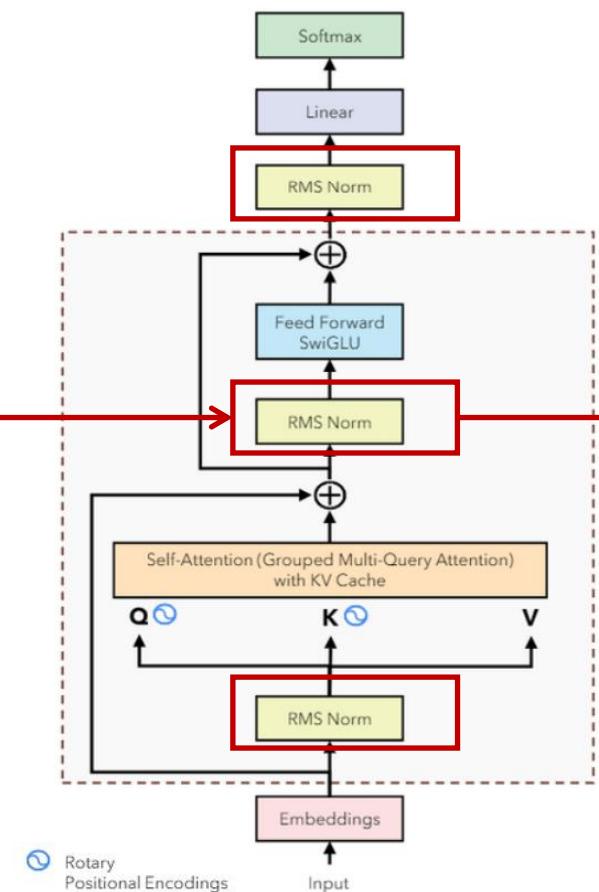
• 归一化层



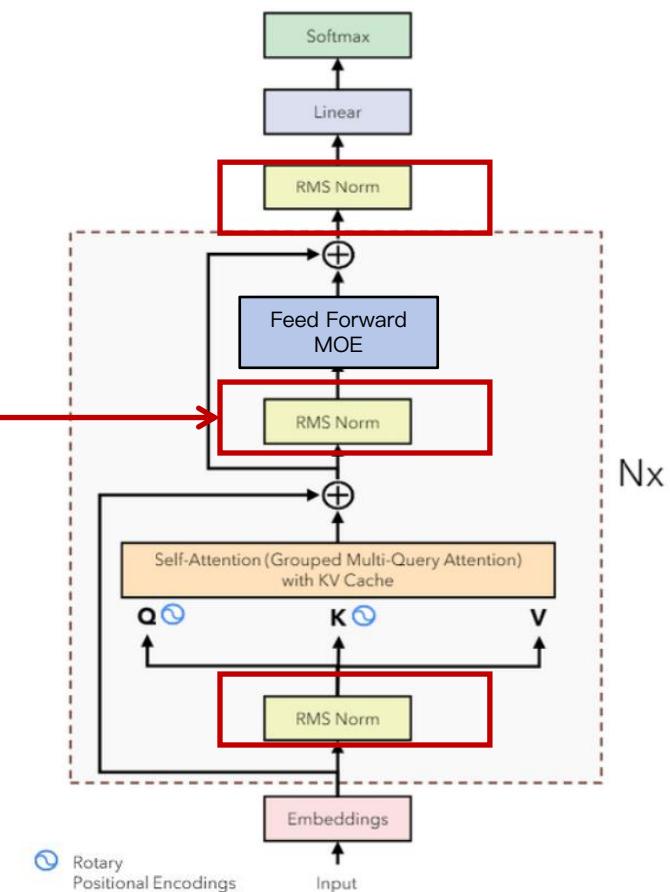
• 归一化层



Transformer



LLama 3



ERNIE 4.5

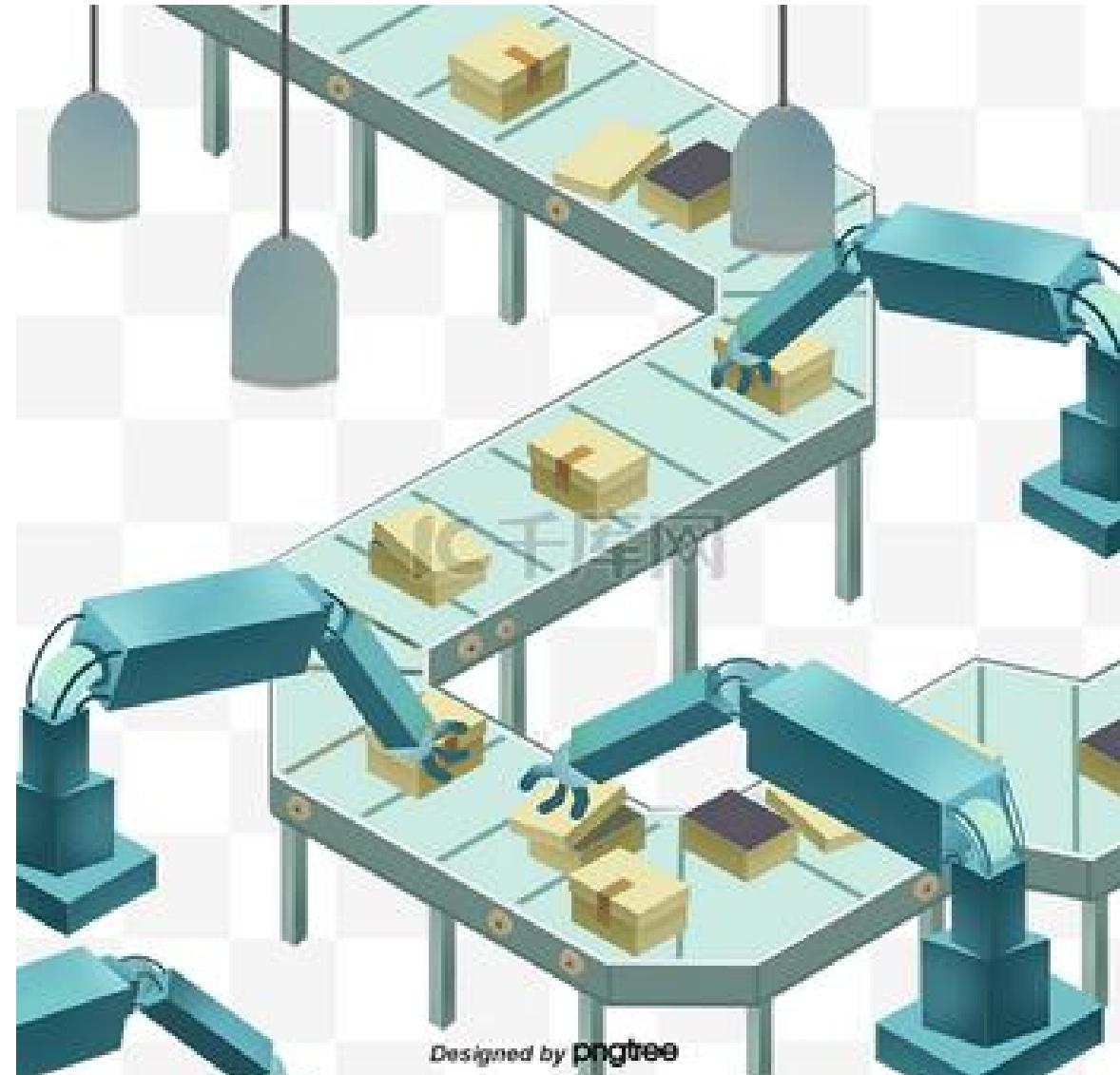
• 为什么我们需要归一化？深层网络的稳定性问题

1. 想象一个生产流水线上，有一批产品形状大小各异，正在传送带上移动。
2. 为了方便后续的加工或包装，每个产品需要调整到统一的尺寸和位置，比如把**大小调整到一致、中心对齐等**。**方便后续处理**，这就是归一化的作用。
3. 训练时每层神经网络输出会随参数更新不断变化，这其中波动的传递会使得训练很不稳定。因此需要在模块与模块间做标准化处理。

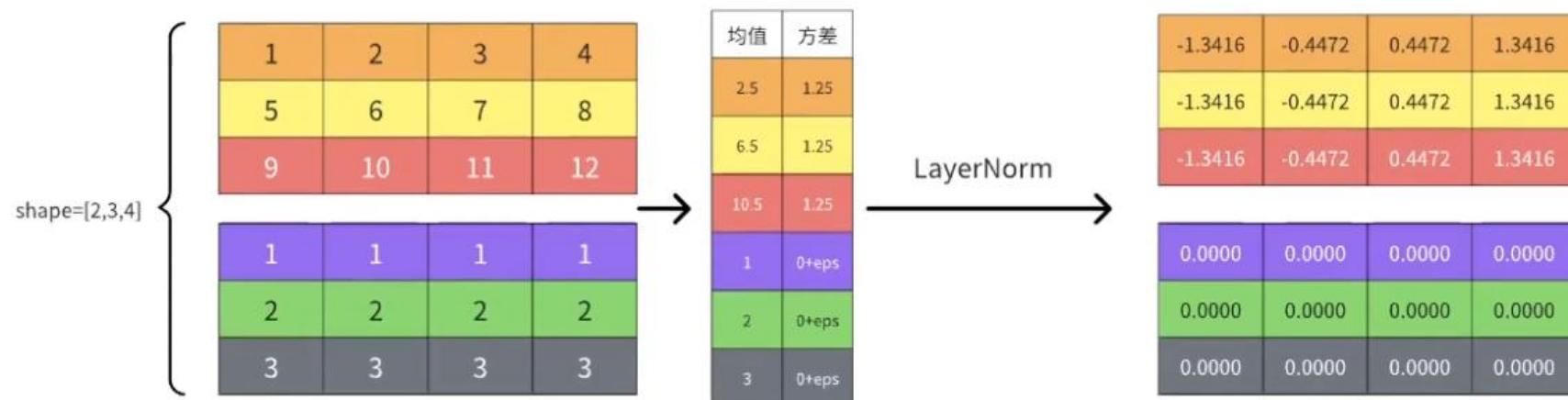
• 归一化的数学原理

$$\hat{x}^{(k)} = \frac{x^{(k)} - \text{E}[x^{(k)}]}{\sqrt{\text{Var}[x^{(k)}]}}$$

- 对于数据如何"标准化"？
- 用一个操作，使数据分布的**均值为0，方差(标准差)为1**。



• 怎么做归一化? LayerNorm和BatchNorm



↑ LayerNorm 在文本任务中常用，只关注输入文本自身进行计算。
(对于一个输入，没必要和语义不相关的句子去一起计算均值和方差！)

↓ BatchNorm是把同一个batch不同实例中相同位置的数据一起计算均值和方差来做归一化，在图像任务中比较常用。



• RMSNorm: 减少计算量开销

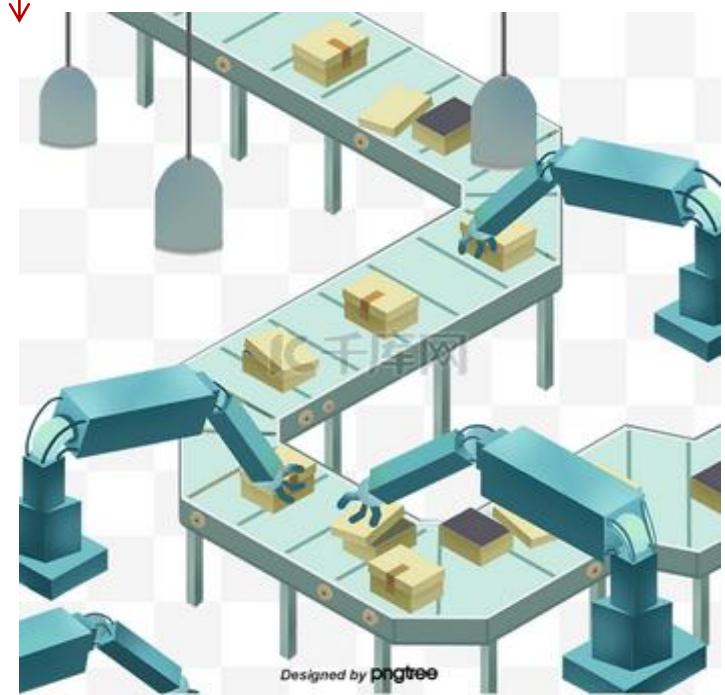
- RMSNorm沿用LayerNorm，相比LayerNorm减少计算开销(不需要均值)。
- 计算量少了，效果基本不变。

$$\hat{x}^{(k)} = \frac{x^{(k)} - \mathbb{E}[x^{(k)}]}{\sqrt{\text{Var}[x^{(k)}]}}$$



$$\bar{a}_i = \frac{a_i}{\text{RMS}(\mathbf{a})} g_i, \quad \text{where } \text{RMS}(\mathbf{a}) = \sqrt{\frac{1}{n} \sum_{i=1}^n a_i^2}.$$

- RMS即对应位置项平方和，这样的话无需计算所有项的均值了。



- 小结

	原始Transformer (2017)	Llama 3 (2024)	文心4.5 (2025)
前馈层 -整体结构	Dense结构	Dense结构	MOE结构
前馈层 -激活函数	ReLU	SwiGLU	SwiGLU
注意力机制	多头注意力 (MHA)	分组查询注意力 (GQA)	分组查询注意力 (GQA)
位置编码	绝对正弦编码	旋转位置编码 (RoPE)	旋转位置编码 (RoPE)
归一化层	层归一化 (LayerNorm)	均方根归一化 (RMSNorm)	均方根归一化 (RMSNorm)

- 从**dense**到**MOE**，通过专家网络和门控机制，显著提升了大参数规模下的模型容量与表达力，同时显著降低推理成本。
- 从**ReLU**到**SwiGLU**，通过门控机制提升了模型的表达能力和训练稳定性。
- 从**多头注意力 (MHA)** 到**分组查询注意力 (GQA)**，解决了KV缓存的内存瓶颈，极大地提升了推理速度。
- 从**绝对位置编码**到**旋转位置编码 (RoPE)**，解决了长度外推+相对位置问题，使模型能够处理更长的上下文。
- 从**层归一化 (LayerNorm)** 到**均方根归一化 (RMSNorm)**，通过移除冗余计算降低了每一步的开销。

你征服了大语言模型的基本原理！ ERNIE与你同在！

请作者喝一杯咖啡
交个朋友！



杨半仙的柚子

江苏 南京



扫一扫上面的二维码图案，加我为朋友。