Transformer

引入

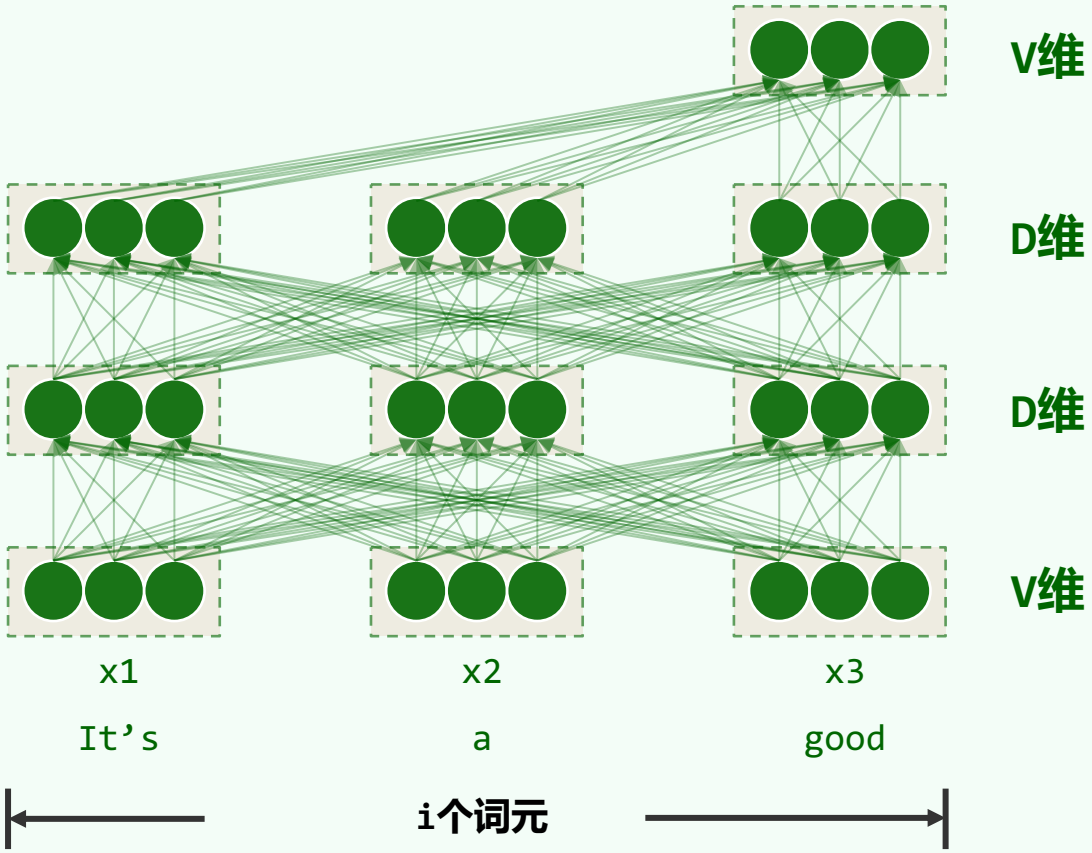
清华大学计算机系 陈键飞

《大模型计算》课程团队

如何处理可变长度的输入？

- **建模** $P(x_{i+1}|x_{\leq i}) = \text{Categorical}(\text{softmax}(g(x_{\leq i})))$
- **输入：i个1～V间的整数，即[i，V]的独热矩阵，或i*V维向量**
- **输出：V个logits，即V维向量**

使用MLP



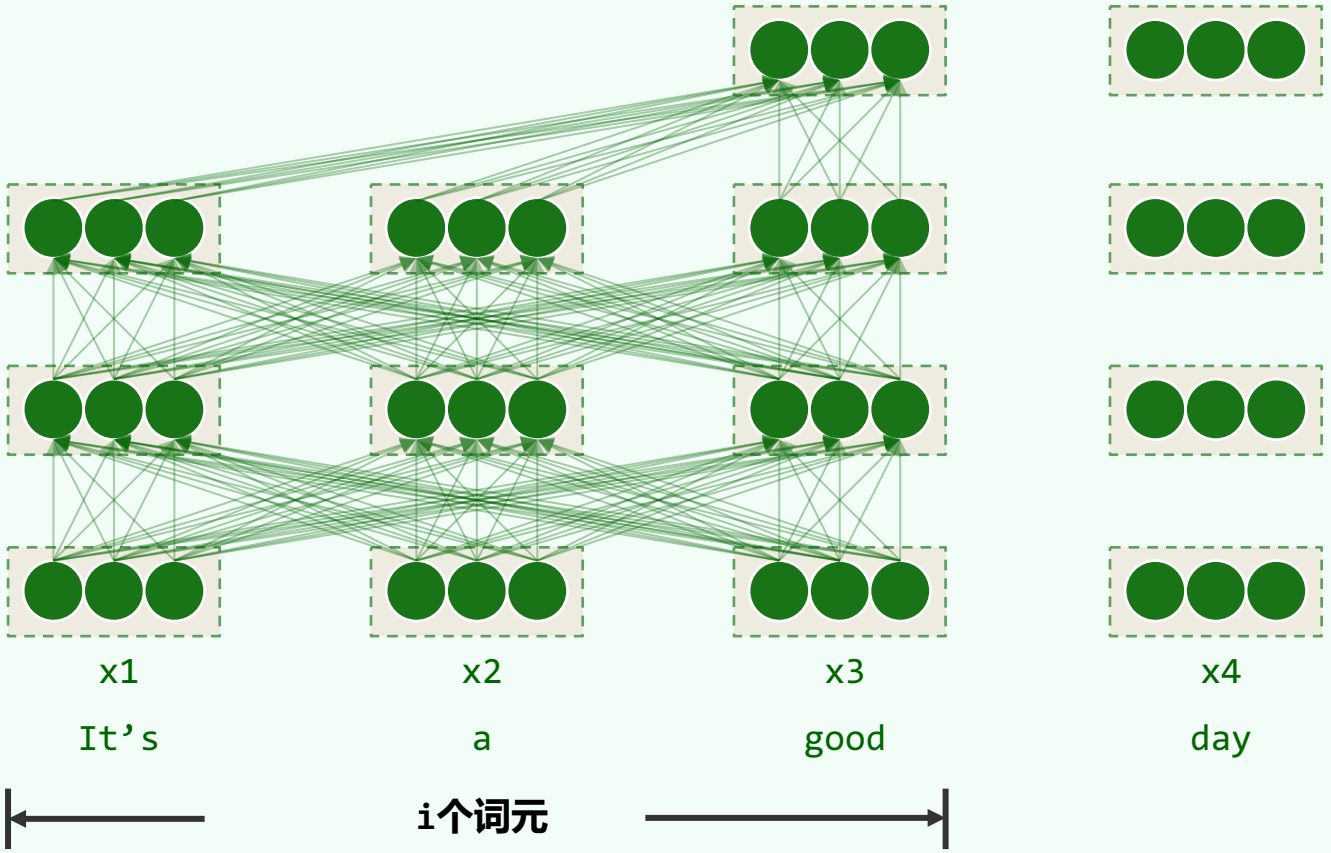
参数：i VD个

参数：i^2 D^2个

参数：i^2 VD个

- ❖ 参数量竟与序列长度相关
- ❖ 问题1：参数爆炸
- ❖ 8K序列长度
- ❖ D=2048，100层
- ❖ 需26843545 B个参数

使用MLP

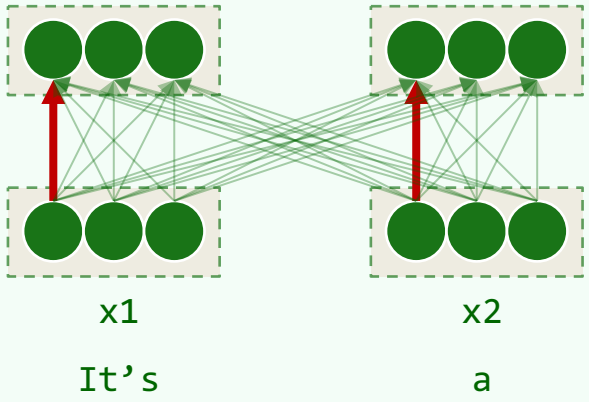


- 参数量竟与序列长度相关
- 问题1:参数爆炸
- 8K序列长度
- D=2048, 100层
- 需26843545 B个参数

- 问题2:难以推广到其他长度
- 如果出现了第8001个词元
- 模型竟没有参数可以处理

使用MLP

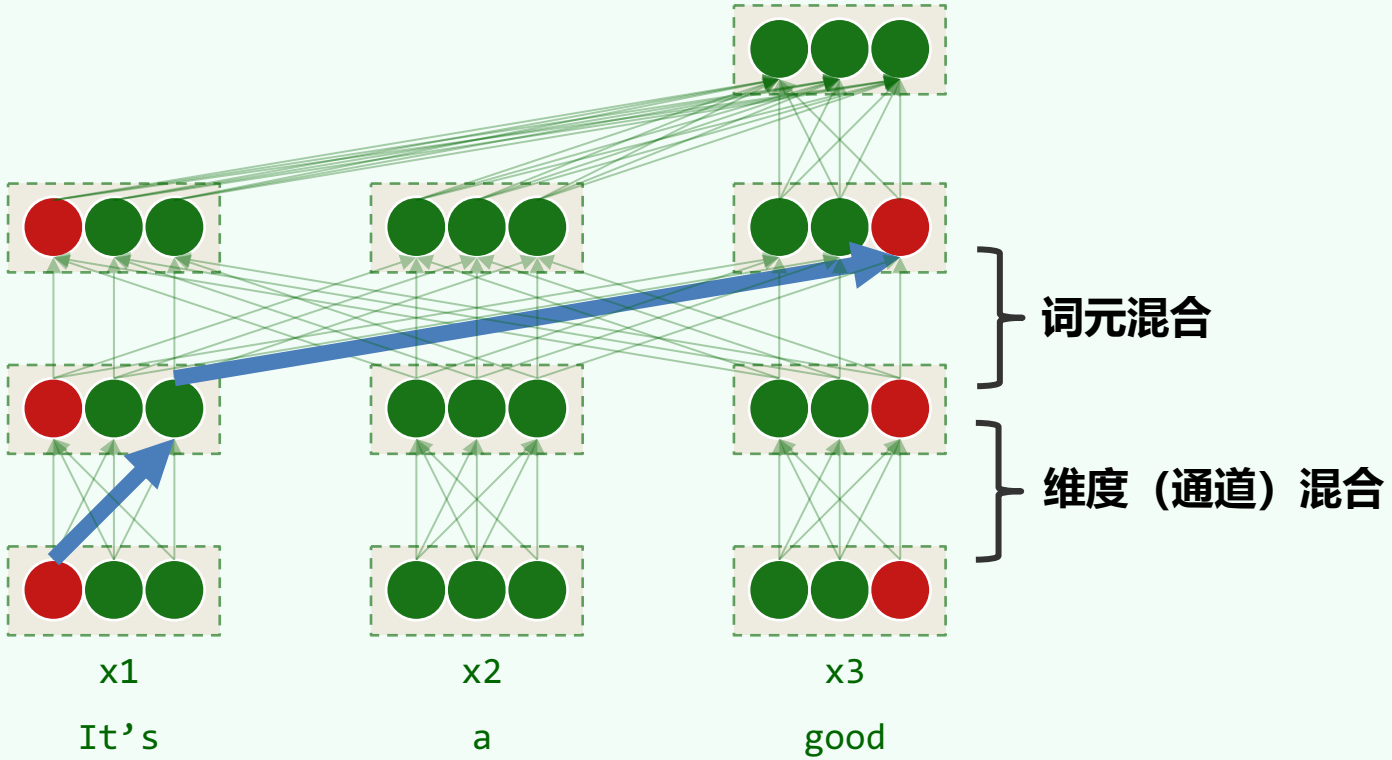
- 问题3:样本复杂性高
- 一个参数至少需要一个数据来学习
- 26843545B个参数需要26843545B=26844T个数据
- 全部互联网数据约10～20T
- 为什么会出现这样的问题?



同一个词的计算应与词出现的位置无关

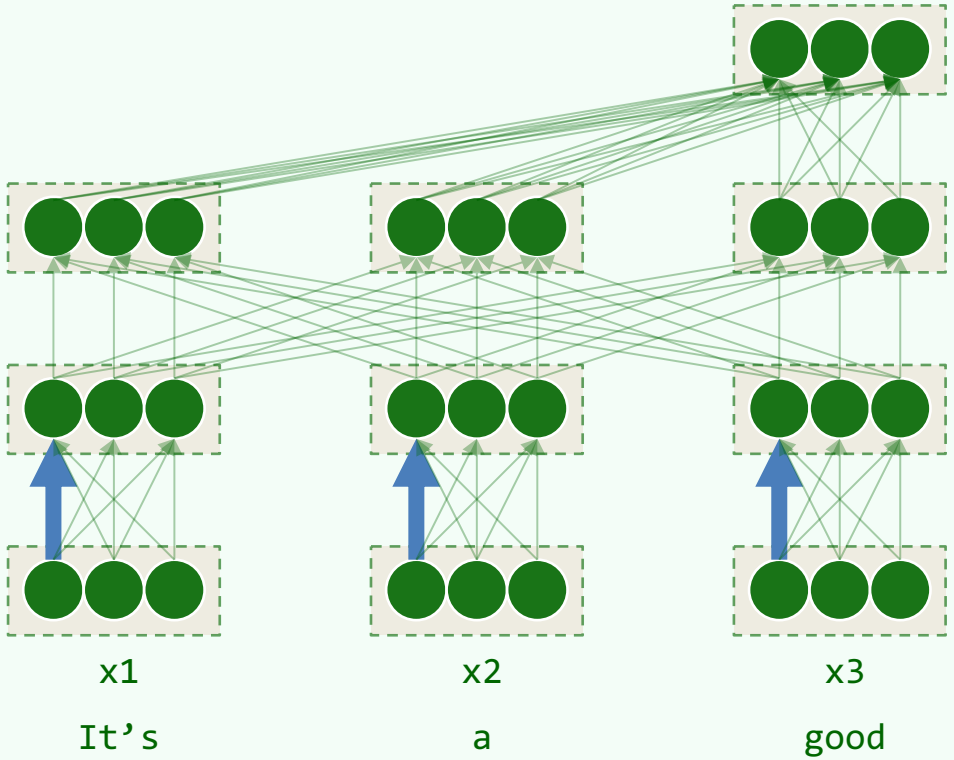
改进1：MLP-mixer

- 核心思路1：拆分，将（词元,维度)x(词元,维度）的混合拆分为 词元x词元 和 维度x维度 的混合
- 任意一对神经元仍然可以通过至多两跳发生交互



改进1：MLP-mixer

- 核心思路2:权重共享
- 词元混合层的参数在不同位置共享



参数应相同

- 参数量

  通道混合：D^2个

  词元混合：T^2D个

- 8K序列长度

  D=2048，100层

  需13107 B个参数，**大大降低**

- 存在的问题：

1. 参数量还是与序列长度相关
2. 词元之间的交互方式是固定的

   与文本内容无关

Transformer

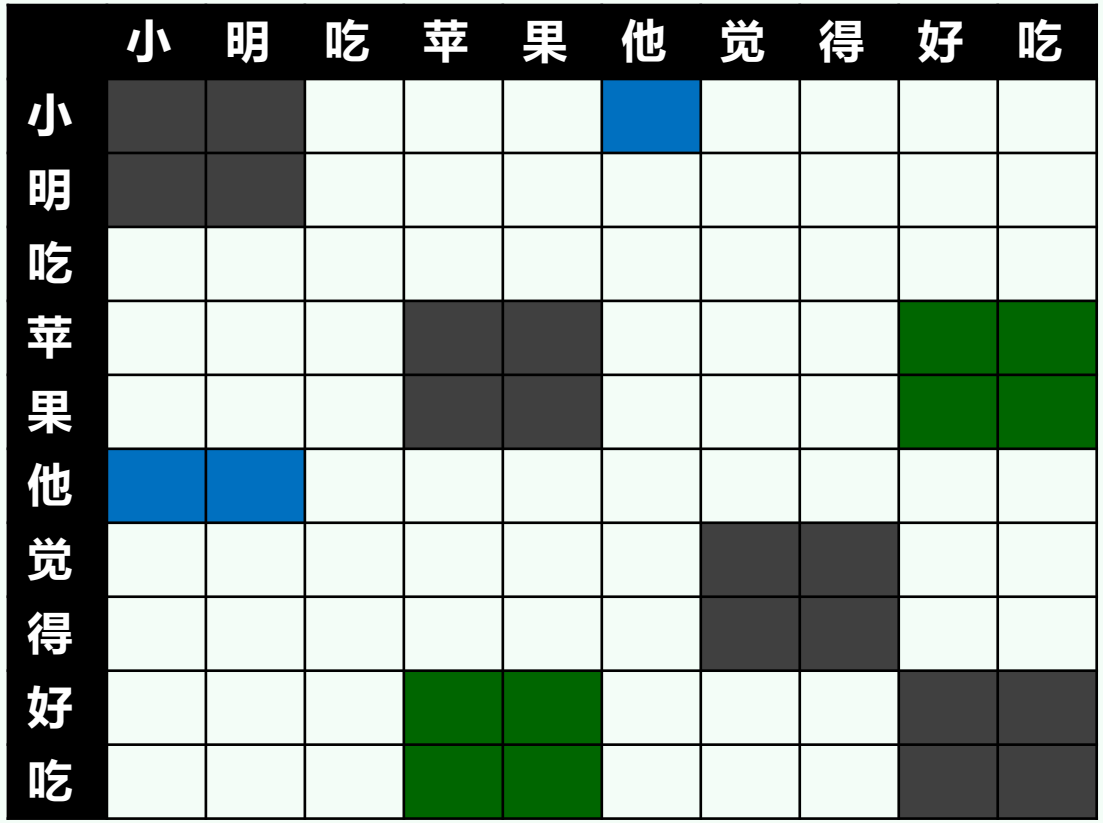
注意力

清华大学计算机系 陈键飞

《大模型计算》课程团队

思路

- 将所有词元（$x_{1\sim T}$）的信息存储到一个数据库中
- 每个词元x_i根据自身语义有选择地从数据库中查询其他词元的信息



if (dict[“名词”]=“苹果” and dict[“形容词”]=“好吃”) then activate

基于数据库的文本理解

小明吃苹果他觉得好吃

- 小：seq[1].save(“小”)
- 明：seq.load(1)　　dict[“主语”].save(“小明”)
- 吃：dict.load(“主语”)　　dict[“谓语”].save(“吃”)
- 苹：seq[4].save(“苹”)
- 果：seq.load(4)　　dict[“宾语”].save(“苹果”)　dict[“食物”].save(“苹果”)
- 他：dict.load(“主语”)
- 需求1：可微—能训练
- 需求2：查询是模糊的

小明告诉小华他考了第一名

可微查询

❖ 输入：键值对

键 Key	值 value
🍎	v1
🍌	v2
🍊	v3
🍑	v4
🍐	v5
🍋	v6
🌶️	v7

d_k维向量

❖ 输入：查询q

查询 Query
🍊

计算相似度

键 Key	相似度 P
🍎	0.02
🍌	0.01
🍊	0.8
🍑	0.03
🍐	0.03
🍋	0.1
🌶️	0.01

$$P = \mathrm{softmax}\big((\langle q, k_1\rangle, \langle q, k_2\rangle, \ldots, \langle q, k_N\rangle)\big)$$

根据相似度

计算查询结果

```
Output =
0.02 * v1  +
0.01 * v2 +
0.8 * v3 +
0.03 * v4 +
0.03 * v5 +
0.1 * v6 +
0.01 * v7
```

模糊、可微

注意力（Attention）

❖ Q, K, V = Linear(X)

$$\text{Attention}(Q,K,V) = \underbrace{\text{softmax}(\frac{QK^T}{\sqrt{d_k}})}_{P}V$$

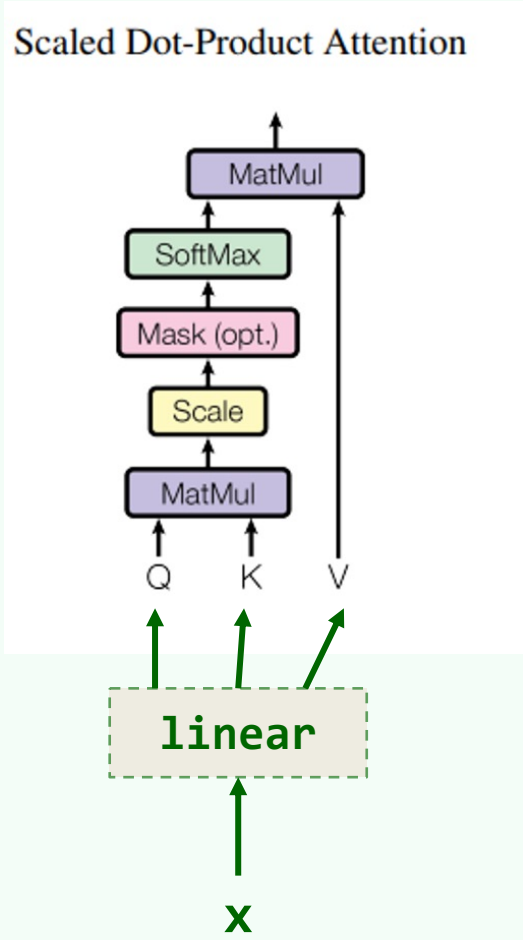
Q		K
	*	

softmax ⇨

归一化

P	k1	k2	k3	k4	k5	k6
q1	.1			.9		
q2		.6			.4	
q3	.1		.8		.1	
q4		.1		.7		.2
q5		.2			.8	
q6			.5			.5

V



直观解释

- Q，K：每人的“桌号”
- V：每人掌握的信息

Q
100
010
100
010
010
001

*

K
100
010
100
010
010
001

softmax ⇨

P	k1	k2	k3	k4	k5	k6
q1	.5		.5			
q2		.33		.33	.33	
q3	.5		.5			
q4		.33		.33	.33	
q5		.33		.33	.33	
q6						1

*

V
100000
010000
001000
000100
000010
000001

=

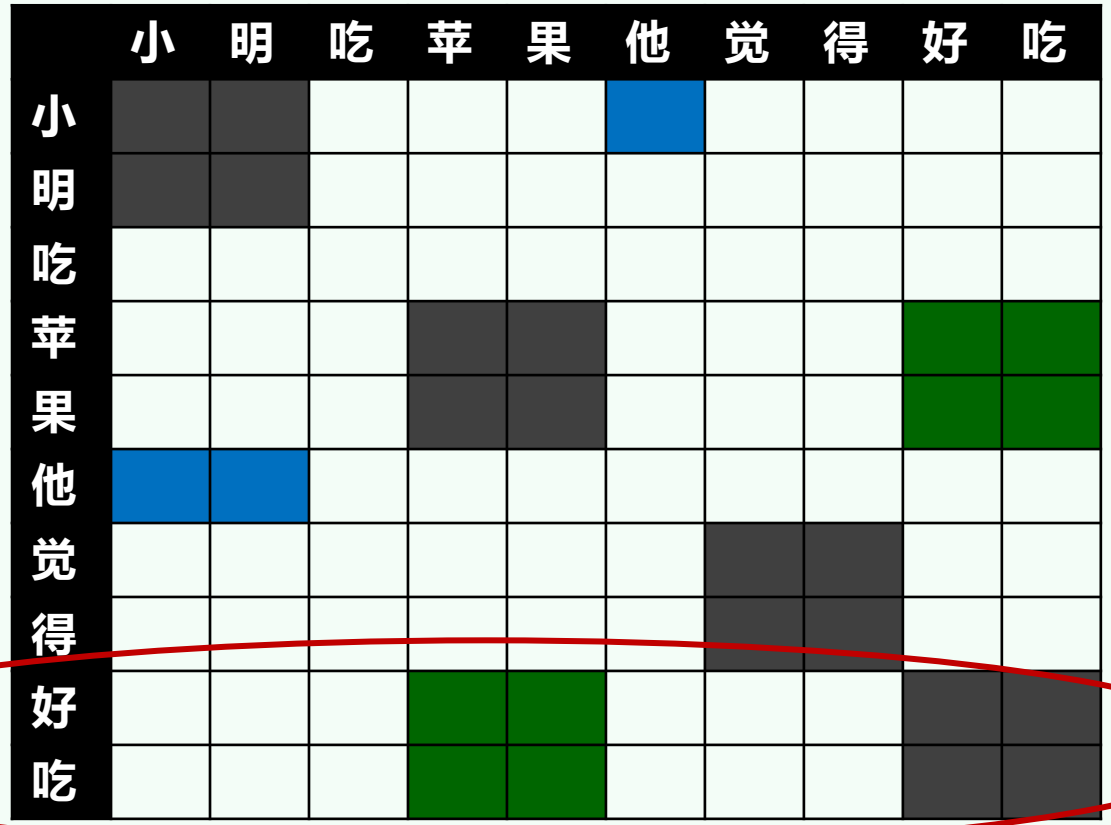
	O
0.5*	101000
0.33*	010110
0.5*	101000
0.33*	010110
0.33*	010110
	000001

注意力能做什么?

K特征提取器 = 本词词性

Q特征提取器 = 能与该词连用的词的词性

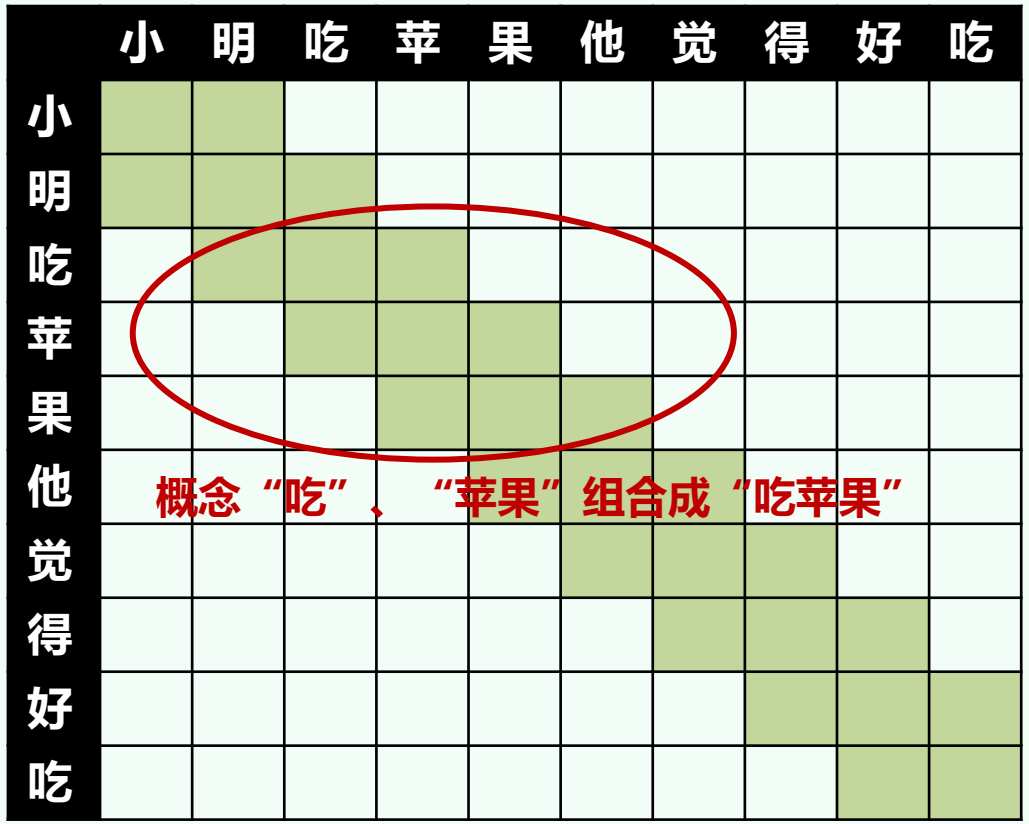
- 看到“苹果”
- 尝试找动词（苹果被怎么了？）
- 尝试找主语（被谁？）



概念“苹果”、“好吃”组合成“苹果好吃”

注意力能做什么?

Q，K特征提取器 = 位置

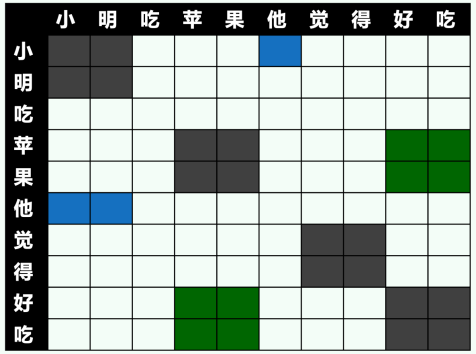


理解注意力：作为代码

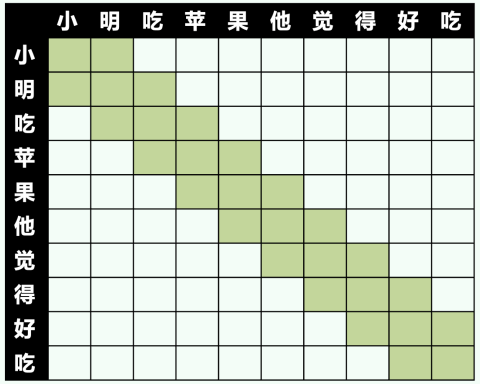
```
For each query token i
  data = 0
  for each token j
    data += sim(query(xi), key(xj)) * value(xj)
```

多头注意力

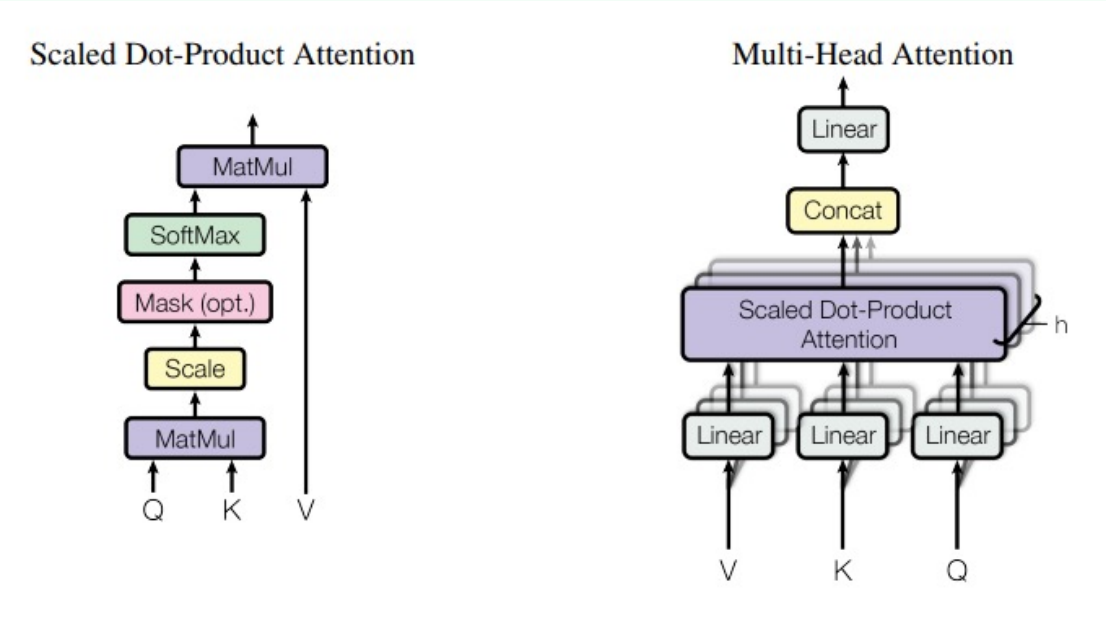
- 问题：一组QK只能捕捉一种连接模式
- 解法：用多个注意力“头”



头1

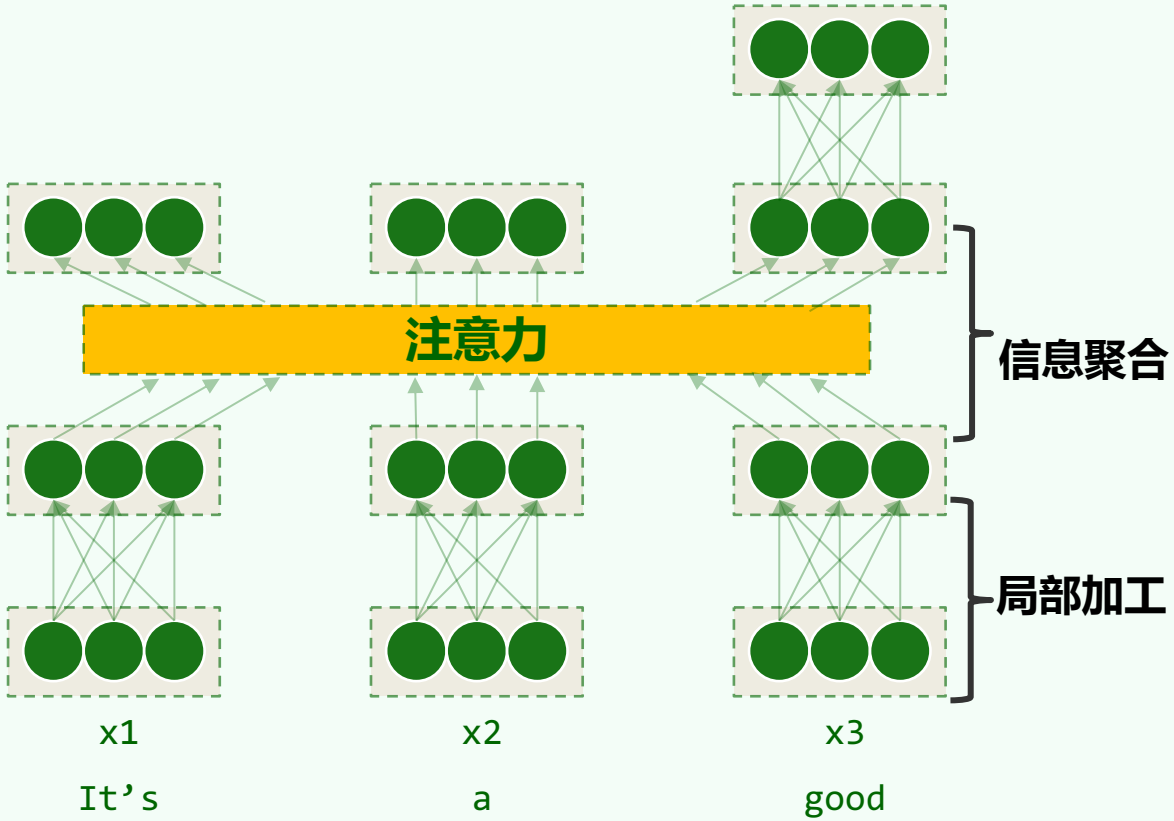


头2



$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, ..., \text{head}_\text{h})W^O$$
$$\text{where head}_\text{i} = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

Transformer网络



❖ **网络结构**

宽度D，每头d_k维，D/d_k个头

❖ **参数量**

通道混合：D^2个

注意力（QKV投影）：$3D^2$个

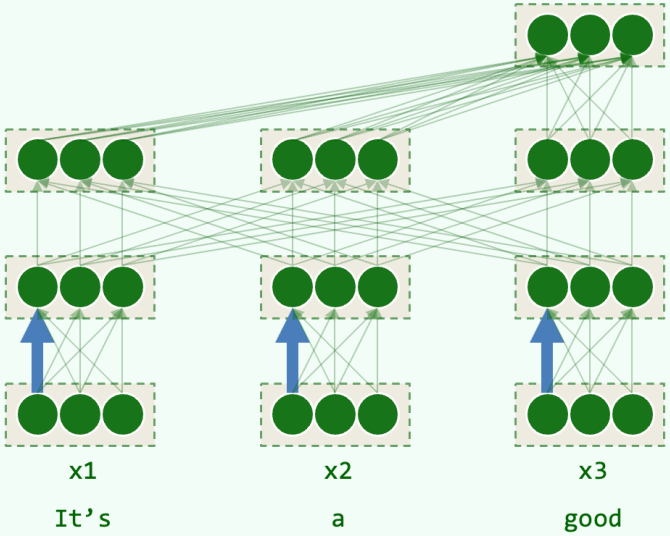
D=2048，100层**仅需1.67B个参数**

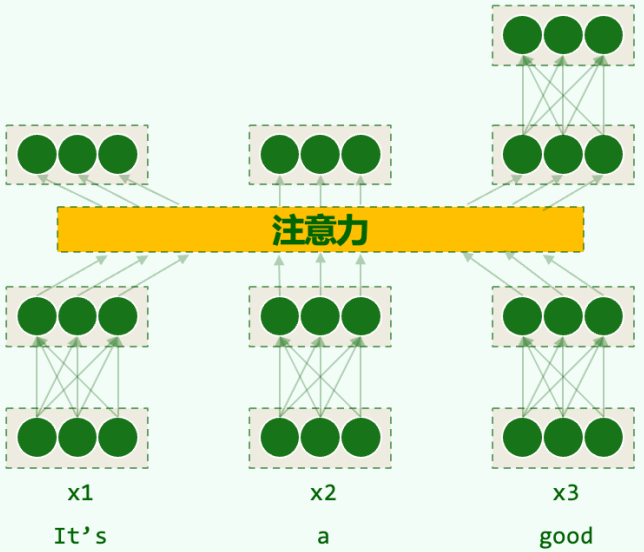
❖ **计算量（每词元）**

通道混合+注意力投影：$4D^2$次 乘加（MAC）

注意力：$\frac{D}{d_k} \times T d_k \times 2 = 2DT$次乘加

对比





MLP-Mixer

- ❖ 直接将词元间的通信P作为可学习参数
- ❖ P为静态，与数据无关
- ❖ 单头

Transformer

- ❖ 学习“词元→桌号”的映射，参数量与序列长度无关
- ❖ P随数据动态变化（桌子固定，人不固定）
- ❖ 多头

核函数

核函数（kernel）： $K: \mathcal{X} \times \mathcal{X} \to \mathbb{R}$，**满足** $K(x, y) = K(y, x)$

半正定核：对任意 $n > 0$ **及** $x_1, \ldots, x_n \in \mathcal{X}$，**矩阵** $\begin{pmatrix} K(x_1, x_1) & \cdots & K(x_1, x_n) \\ \vdots & \ddots & \vdots \\ K(x_n, x_1) & \cdots & K(x_n, x_n) \end{pmatrix}$ **均为半正定**

核代表“相似度”：线性核 $K(x, y) = x^T y$，**高斯核** $K(x, y) = \exp\left(-\frac{\|x-y\|^2}{2\sigma^2}\right)$

默瑟定理（Mercer's Theorem）

对任意半正定核K，存在映射$\varphi: \mathcal{X} \to \mathcal{H}$，使得$\forall x, y \in \mathcal{X}$，有

$$K(x, y) = \langle \varphi(x), \varphi(y) \rangle$$

高维特征空间

注意力： $\varphi(x) = xW_k^T$

注意力可以看作一种可学习的核函数

Transformer

词嵌入

清华大学计算机系 陈键飞

《大模型计算》课程团队

词嵌入

- 考虑网络的第一层：
- 输入[N，T，V]独热（[批大小，序列长度，词表大小]）
- 输出[N，T，D]　　（[批大小，序列长度，维度]）
- 权重[V，D]　词嵌入表

Id	词	嵌入
1	a	
2	abandon	
3000	day	
10000	good	

```
def __init__(self, config: LlamaConfig):
  self.embed_tokens = nn.Embedding(config.vocab_size, config.hidden_size, self.padding_idx)

def forward(…):
  inputs_embeds: torch.Tensor = self.embed_tokens(input_ids)
```

https://github.com/huggingface/transformers/blob/ebbcf00ad1b06fa87effe179d128e73390255844/src/transformers/models/llama/modeling_llama.py#L367

填充词元

🎯 主要作用

padding_idx 的核心作用可归纳为以下两点：

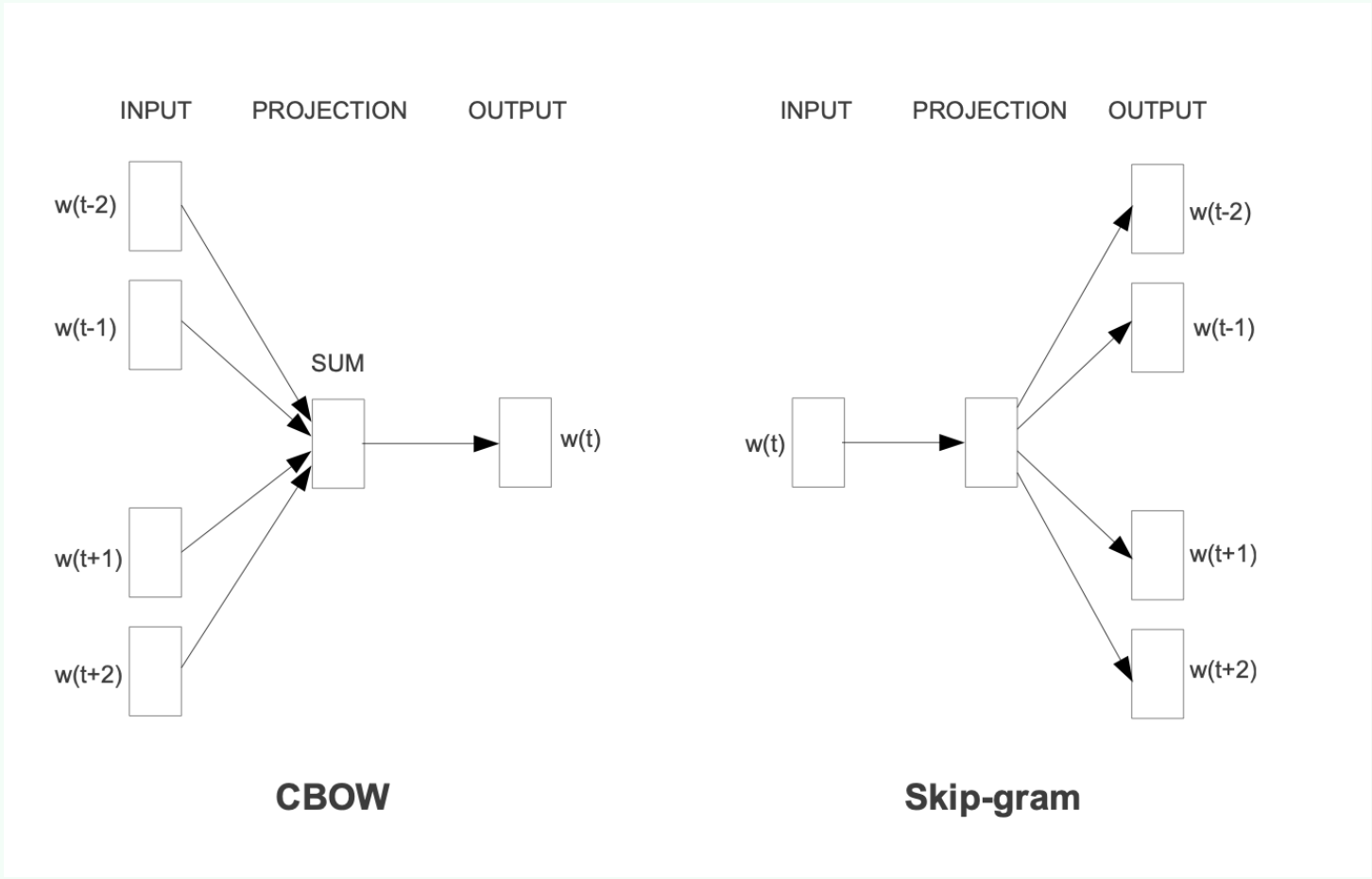
作用描述	说明
屏蔽梯度更新 1 4 6	确保模型在训练过程中**不会更新**padding_idx指定索引处的嵌入向量权重。
初始化并保持为零 1 4 6	在训练开始时，将该索引对应的嵌入向量**初始化为全零**，并且在整个训练过程中其值会**保持固定**（通常为零）。

🛠️ 工作原理与效果

当你在nn.Embedding中设置了padding_idx（例如padding_idx=0），模型会做出以下调整：

- **稳定训练**：通过确保填充位置的嵌入不贡献梯度，避免了模型从这些无意义的填充中学习，从而**提高了训练的稳定性和效率**。
- **减少干扰**：填充符被映射为零向量，其与任何其他向量的点积为零，这有助于**防止填充token影响注意力机制的计算**（特别是在Transformer模型中）或其他基于相似度的计算。

线性词嵌入模型



$$h = 2C \sum_{c=1}^{C} v_{t-c} + v_{t+c}$$

$$P(w_t = i|\text{context}) = \frac{\exp\left(u_i^T h\right)}{\sum_{k=1}^{V} \exp(u_k^T h)}$$

Mikolov, Tomas, et al. “Efficient estimation of word representations in vector space.” *arXiv preprint arXiv:1301.3781* (2013).
NeurIPS 2023 “时间检验奖”

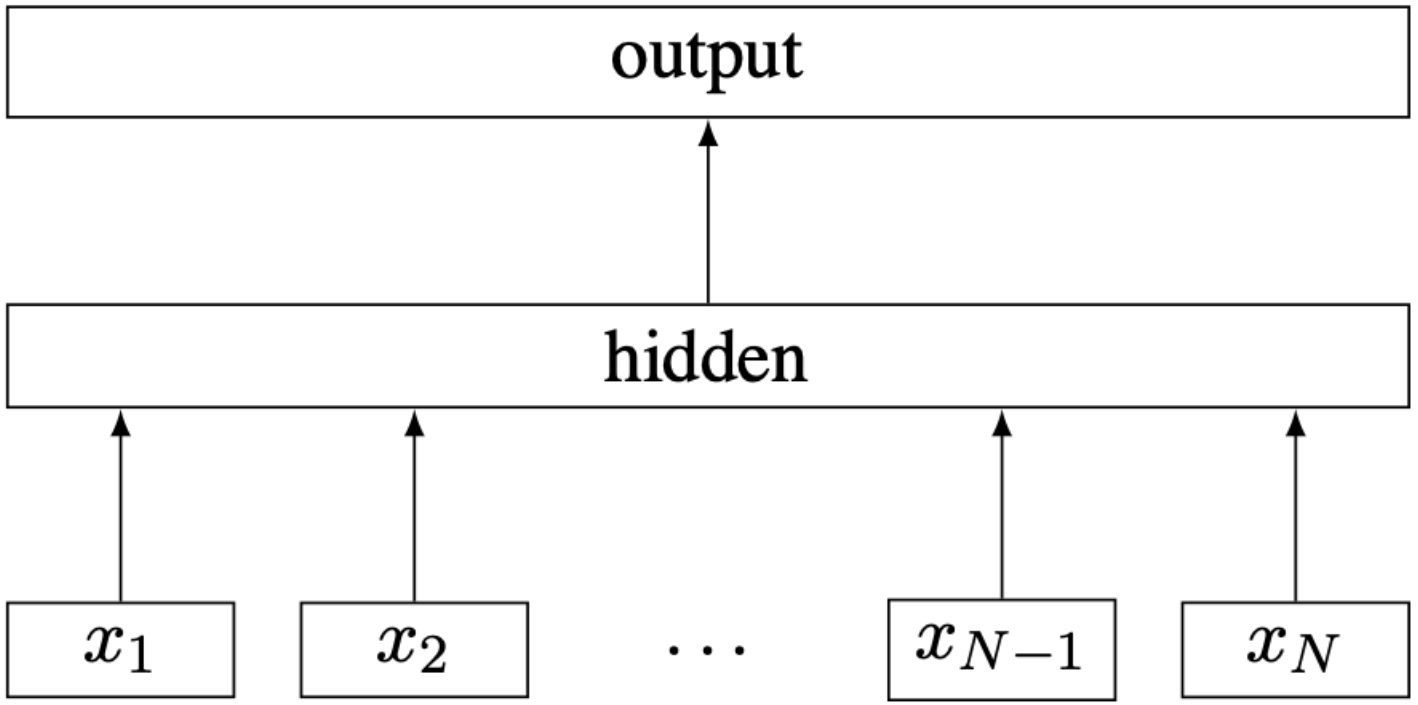
单词线性运算

Paris – France + Italy = Rome

Relationship	Example 1	Example 2	Example 3
France - Paris	Italy: Rome	Japan: Tokyo	Florida: Tallahassee
big - bigger	small: larger	cold: colder	quick: quicker
Miami - Florida	Baltimore: Maryland	Dallas: Texas	Kona: Hawaii
Einstein - scientist	Messi: midfielder	Mozart: violinist	Picasso: painter
Sarkozy - France	Berlusconi: Italy	Merkel: Germany	Koizumi: Japan
copper - Cu	zinc: Zn	gold: Au	uranium: plutonium
Berlusconi - Silvio	Sarkozy: Nicolas	Putin: Medvedev	Obama: Barack
Microsoft - Windows	Google: Android	IBM: Linux	Apple: iPhone
Microsoft - Ballmer	Google: Yahoo	IBM: McNealy	Apple: Jobs
Japan - sushi	Germany: bratwurst	France: tapas	USA: pizza

FastText分类器

❖ 利用CBOW，时间复杂度O(len * D)，空间复杂度O(V * D)



将word embedding平均

启示：词嵌入可以直接做加法得到组合概念

注意力中用“加权平均”做特征组合有意义

A. Joulin, E. Grave, P. Bojanowski, T. Mikolov, *Bag of Tricks for Efficient Text Classification*

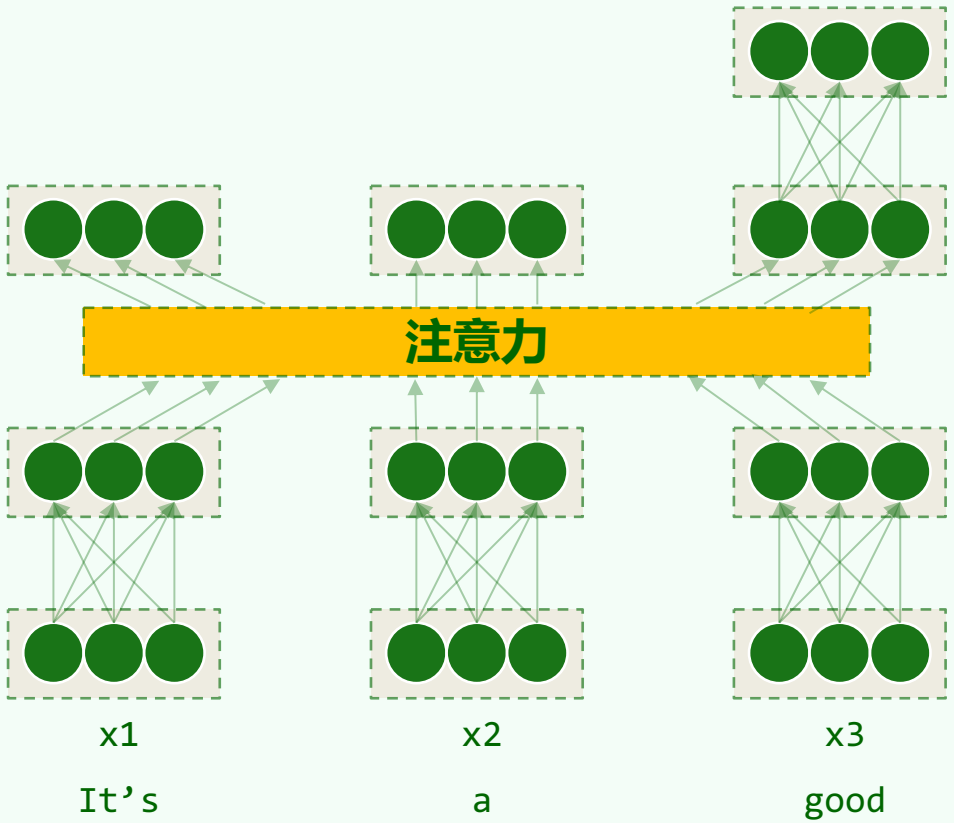
Transformer

位置编码

清华大学计算机系 陈键飞

《大模型计算》课程团队

问题



❖ 单词顺序互换，表征似乎不变?

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

$$\Pi\big(\text{Attention}(\text{Q}, \text{K}, \text{V})\big) = \text{Attention}(\Pi(\text{Q}), \Pi(\text{K}), \Pi(\text{V}))$$

❖ 人 咬 狗

❖ 狗 咬 人

❖ 问题：注意力机制对单词顺序无感

绝对位置编码

❖ 思想：将位置信息与单词信息一起输入

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

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

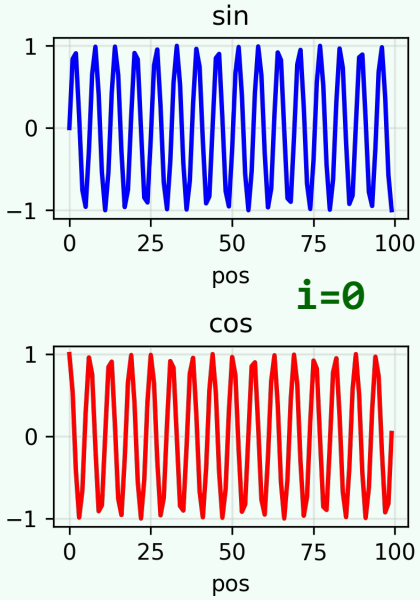
[1, 10000)

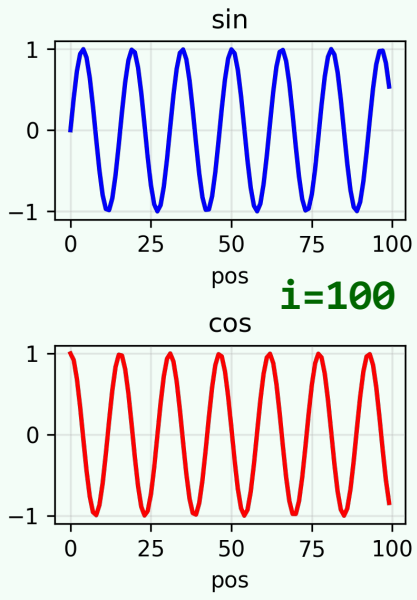
例：d_{model}=2048, T=100

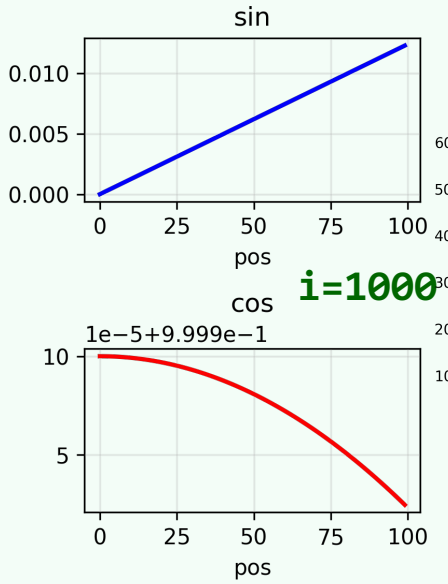
周期指数增长

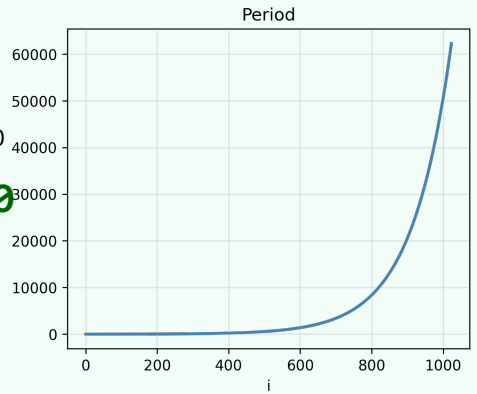
最小周期=6

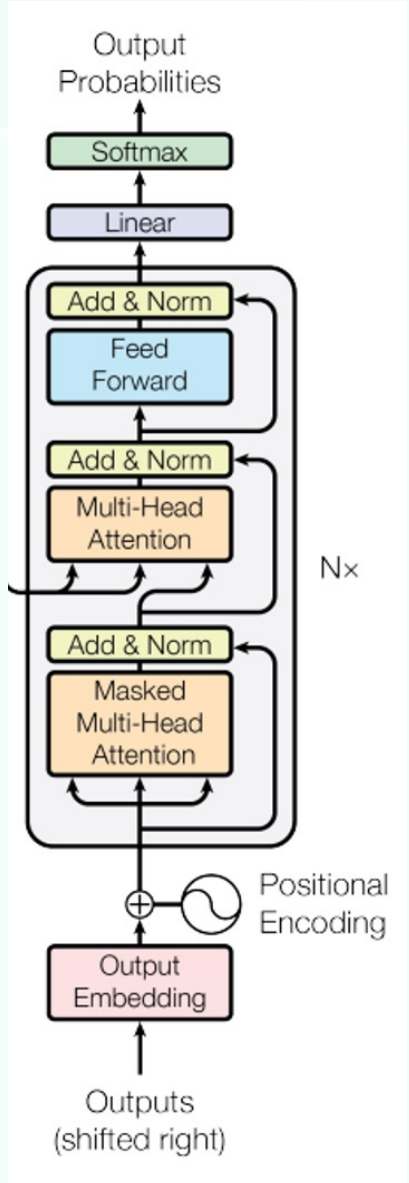
最大周期=60000











https://github.com/facebookresearch/fairseq/blob/main/fairseq/modules/sinusoidal_positional_embedding.py

绝对位置编码

❖ 绝对位置编码 + 单词嵌入

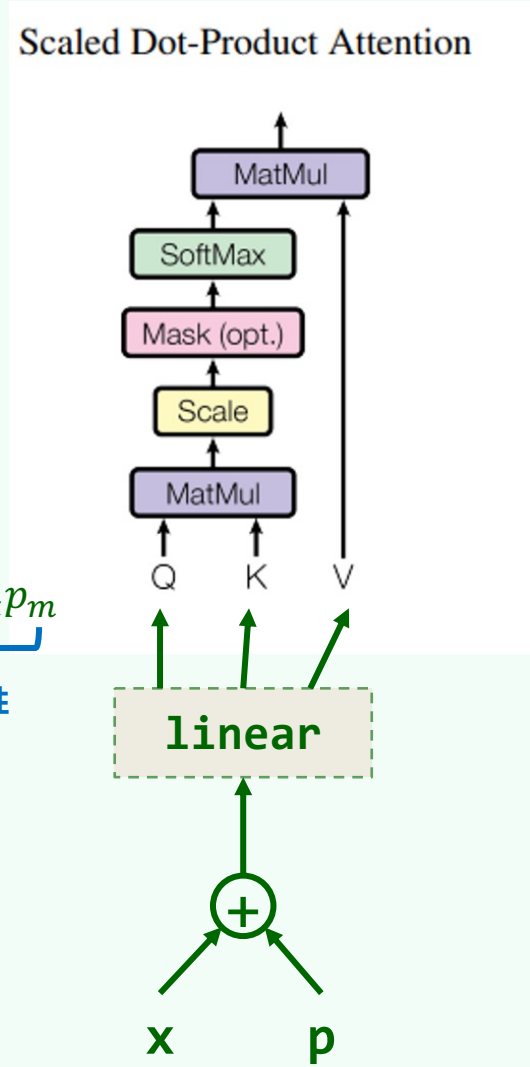
$$z = x + p$$

```
# forward the GPT model itself
tok_emb = self.transformer.wte(idx) # token embeddings of shape (b, t, n_embd)
pos_emb = self.transformer.wpe(pos) # position embeddings of shape (t, n_embd)
x = tok_emb + pos_emb
```

$$q = W_q(x+p), k = W_k(x+p)$$

$$\langle q_n, k_m \rangle = (x_n + p_n)^T W_q^T W_k (x_m + p_m) = \underbrace{x_n^T W_q^T W_k x_m}_{\text{内容相关性}} + \underbrace{x_n^T W_q^T W_k p_m + p_n^T W_q^T W_k x_m}_{\text{交叉相关性}} + \underbrace{p_n^T W_q^T W_k p_m}_{\text{位置相关性}}$$

“在文章的第一个自然段中……”



1. 为什么是加法（而不是拼接）？
2. 为什么要用固定的sin、cos，不直接学?

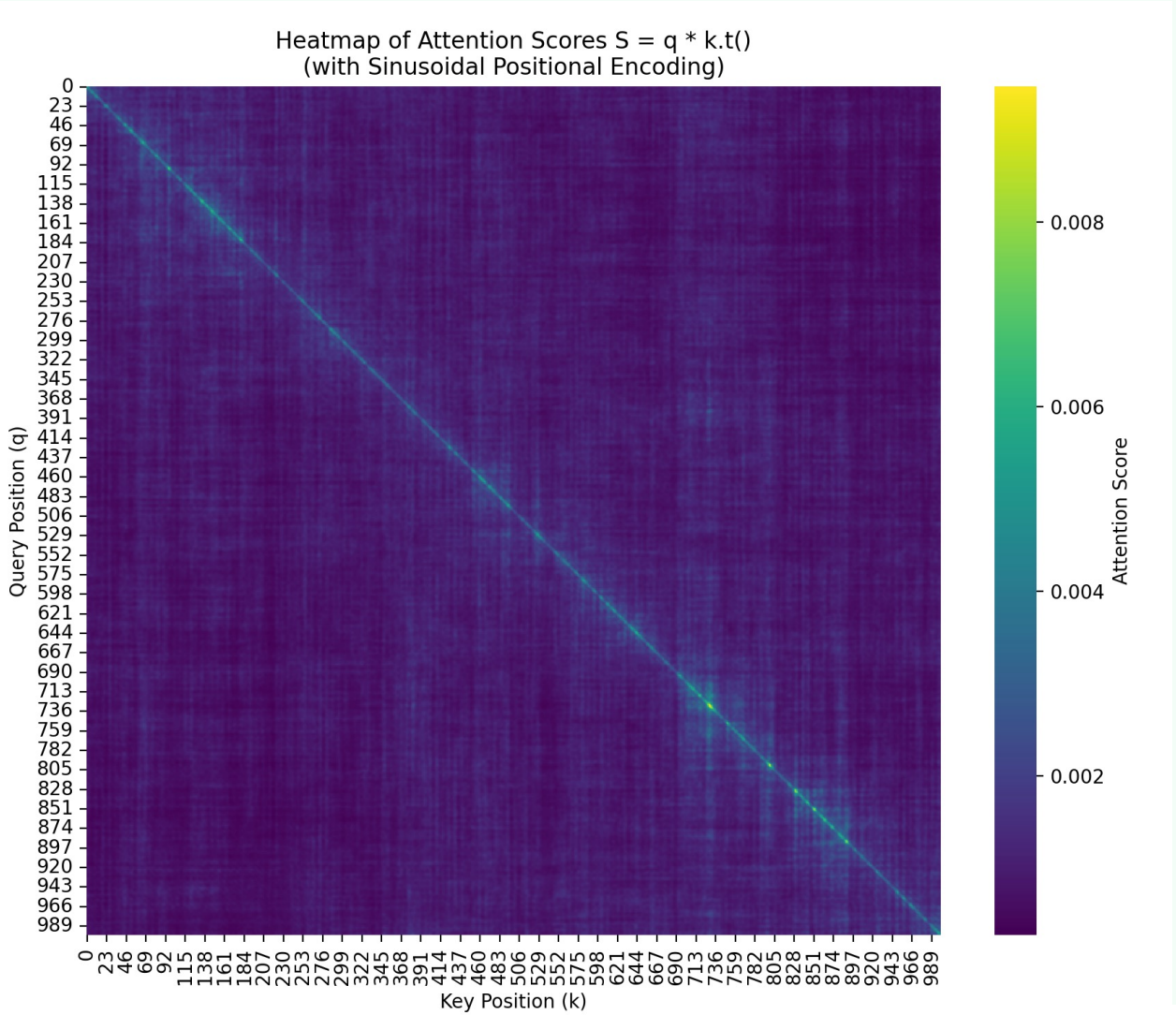
https://github.com/karpathy/nanoGPT/blob/93a43d9a5c22450bbf06e78da2cb6eeef084b717/model.py#L177

位置相关性

- 对大部分维度，相邻位置的p会接近
- 意味着相邻token的“相关性”会更高

$$W_q = W_k = randn$$

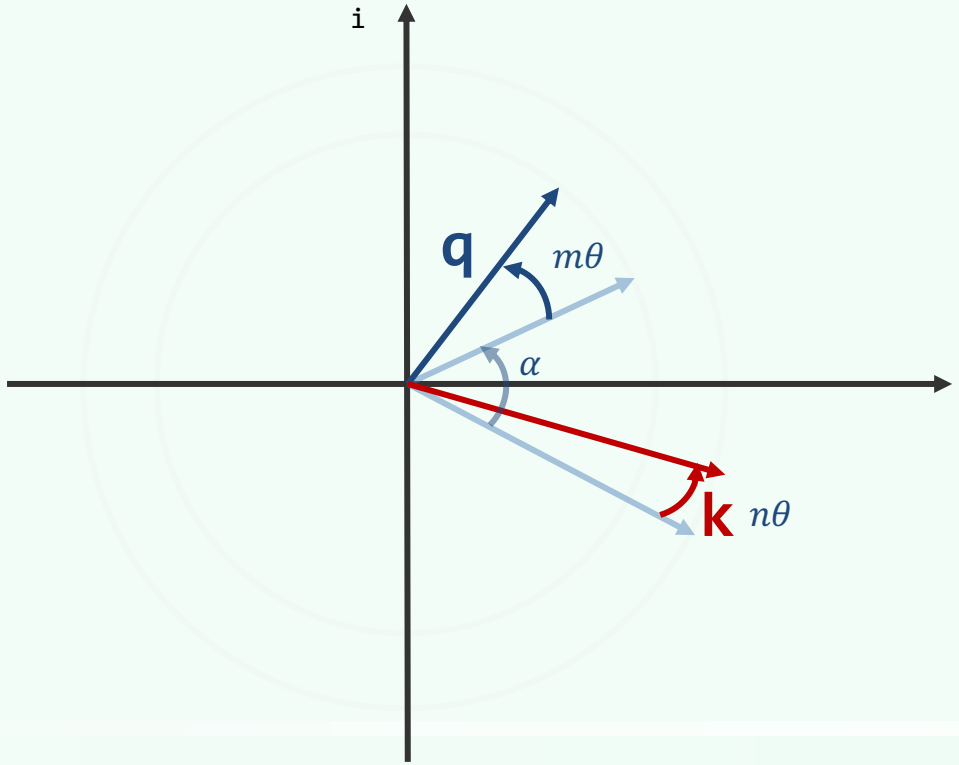
- 按照位置与附近的token交换信息



相对位置编码（Rotary Position Embedding, RoPE）

首先考虑q、k仅有2维的特殊情况

$$f_q(\boldsymbol{x}_m, m) = (\boldsymbol{W}_q \boldsymbol{x}_m) e^{im\theta}$$
$$f_k(\boldsymbol{x}_n, n) = (\boldsymbol{W}_k \boldsymbol{x}_n) e^{in\theta}$$
$$g(\boldsymbol{x}_m, \boldsymbol{x}_n, m-n) = \mathrm{Re}[(\boldsymbol{W}_q \boldsymbol{x}_m)(\boldsymbol{W}_k \boldsymbol{x}_n)^* e^{i(m-n)\theta}]$$



向量(x, y) ⇔ 复数x + yi

向量内积$\langle (x_1, y_1), (x_2, y_2) \rangle$

⇔ **复数内积实部** $\mathrm{Re}[(x_1 + y_1 i)(x_2 - y_2 i)]$

向量逆时针旋转 = 复数乘法

内积$\mathrm{Re}[qk^*] = \|q\|\|k\| \cos(\alpha + (m-n)\theta)$

内容差异度$\in [-\pi, \pi]$

位置差异度$\in [0, +\infty]$

结果只与相对位置m-n有关

与绝对位置m-n无关

高维情况

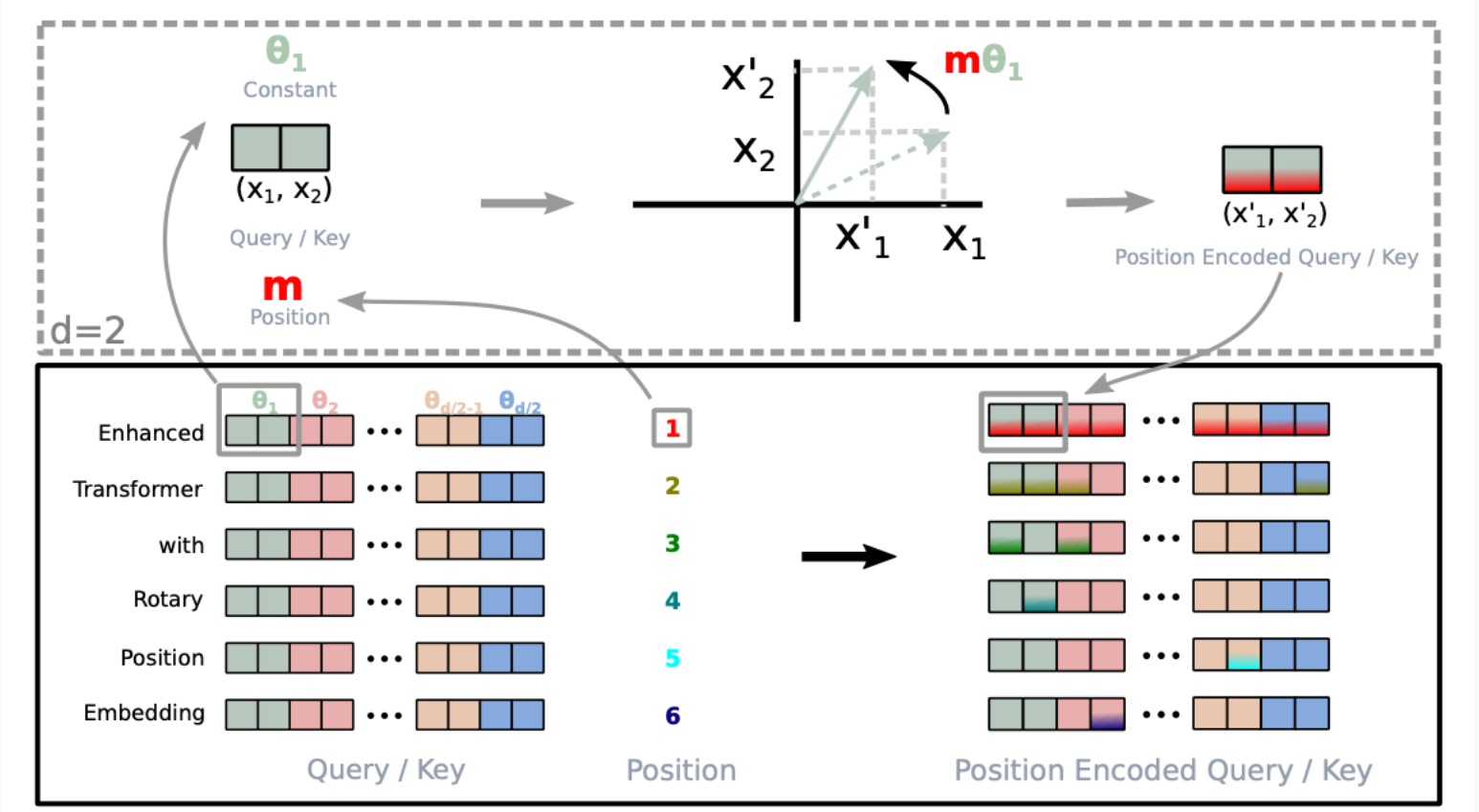
会不会本来内容和位置都差异很大，加起来差异抵消了？

内积 $\mathrm{Re}[qk^*] = \|q\|\|k\|\cos(\alpha + (m-n)\theta)$

两维两维旋转

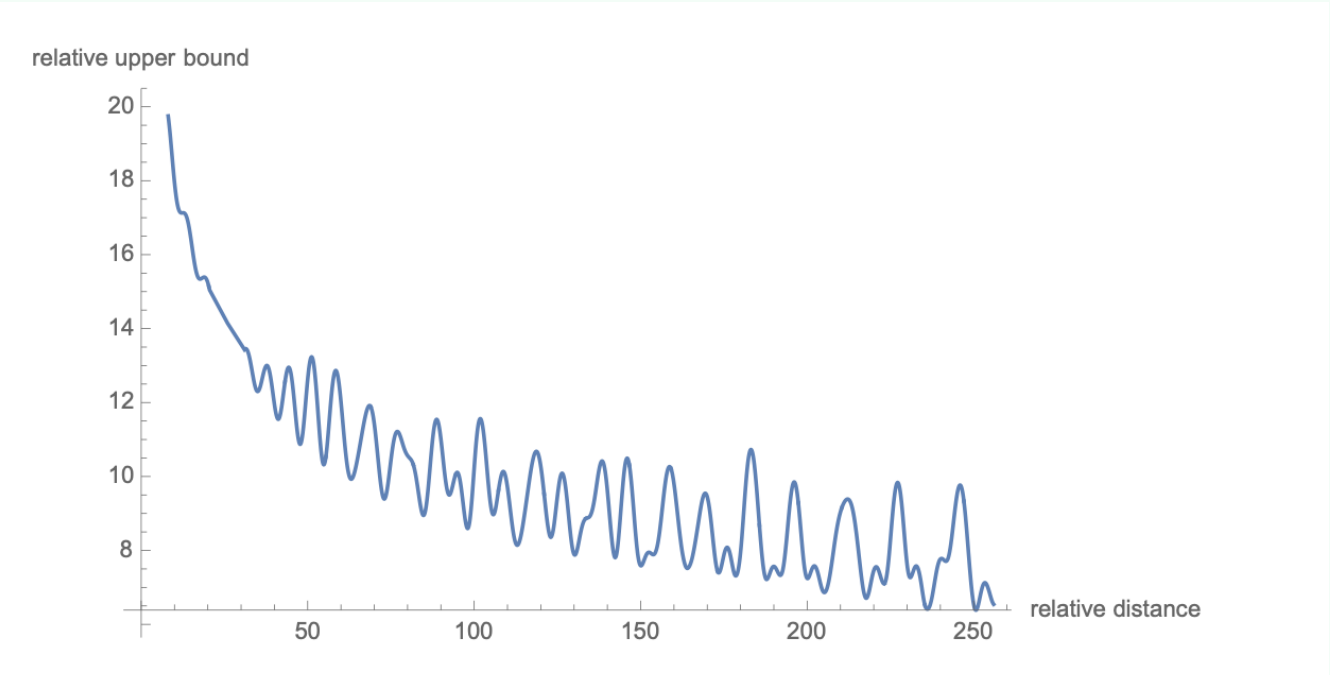
$$\theta_i = 10000^{-\frac{2(i-1)}{d_{\text{model}}}}$$

对大部分i来说，θ_i都很小……



相关性随距离衰减

- 会不会本来内容和位置都差异很大，加起来差异抵消了?
- 不同频率θ_i很多，只有$m-n$小才能让所有组的$(m-n)\theta_i \mod 2\pi$ 都小
- 最乐观的情况：内容差异度$\alpha = 0$
- 不失一般性，设每组$\|q\| = \|k\| = 1$
- $\langle \boldsymbol{q}, \boldsymbol{k} \rangle = \sum_{i=1}^{d/2} \langle q_i, k_i \rangle = \sum_{i=1}^{d/2} \cos(\theta_i \Delta)$
- $\Delta = \mathrm{m} - \mathrm{n}$，相对距离
- 思考：如果$\alpha \neq 0$，曲线怎么变?



RoPE好在哪?

- 显式只考虑相对位置
- 长序列泛化可能更好：不管序列多长，只要最长的距离不超过训练时的长度……
  - 比如100篇8K长度的文章拼在一起，彼此独立……
- 每层都注入位置信息，位置信息较强

- 相应的：有时用NoPE（无位置编码）更好

https://arxiv.org/pdf/2305.19466

https://arxiv.org/pdf/2501.18795

https://www.kexue.fm/archives/10907

实现

3.2.2 General form

In order to generalize our results in 2D to any $\boldsymbol{x}_i \in \mathbb{R}^d$ where d is even, we divide the d-dimension space into $d/2$ sub-spaces and combine them in the merit of the linearity of the inner product, turning $f_{\{q,k\}}$ into:

$$f_{\{q,k\}}(\boldsymbol{x}_m, m) = \boldsymbol{R}^d_{\Theta,m} \boldsymbol{W}_{\{q,k\}} \boldsymbol{x}_m \tag{14}$$

where

$$\boldsymbol{R}^d_{\Theta,m} = \begin{pmatrix} \cos m\theta_1 & -\sin m\theta_1 & 0 & 0 & \cdots & 0 & 0 \\ \sin m\theta_1 & \cos m\theta_1 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & \cos m\theta_2 & -\sin m\theta_2 & \cdots & 0 & 0 \\ 0 & 0 & \sin m\theta_2 & \cos m\theta_2 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & \cos m\theta_{d/2} & -\sin m\theta_{d/2} \\ 0 & 0 & 0 & 0 & \cdots & \sin m\theta_{d/2} & \cos m\theta_{d/2} \end{pmatrix} \tag{15}$$

注意：此处d是每个head的dim

实现

```
def apply_rotary_pos_emb(q, k, cos, sin, position_ids=None, unsqueeze_dim=1):
    cos = cos.unsqueeze(unsqueeze_dim)
    sin = sin.unsqueeze(unsqueeze_dim)
    q_embed = (q * cos) + (rotate_half(q) * sin)
    k_embed = (k * cos) + (rotate_half(k) * sin)
    return q_embed, k_embed


def rotate_half(x):
    x1 = x[..., : x.shape[-1] // 2]
    x2 = x[..., x.shape[-1] // 2 :]
    return torch.cat((-x2, x1), dim=-1)
```

https://github.com/huggingface/transformers/blob/main/src/transformers/models/llama/modeling_llama.py

实现

```
def forward(self, x, position_ids):
    inv_freq_expanded = self.inv_freq[None, :, None].float().expand(position_ids.shape[0], -1, 1)
    position_ids_expanded = position_ids[:, None, :].float()
    freqs = (inv_freq_expanded.float() @ position_ids_expanded.float()).transpose(1, 2)
    emb = torch.cat((freqs, freqs), dim=-1)
    cos = emb.cos() * self.attention_scaling
    sin = emb.sin() * self.attention_scaling

    return cos, sin
```

https://github.com/huggingface/transformers/blob/main/src/transformers/models/llama/modeling_llama.py

Transformer

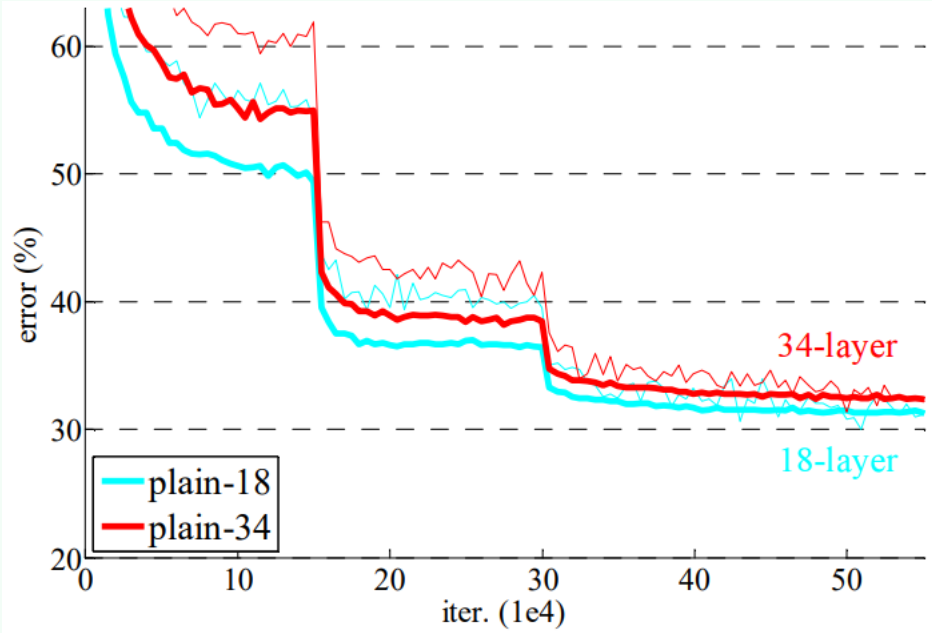
其他关键组件

清华大学计算机系 陈键飞

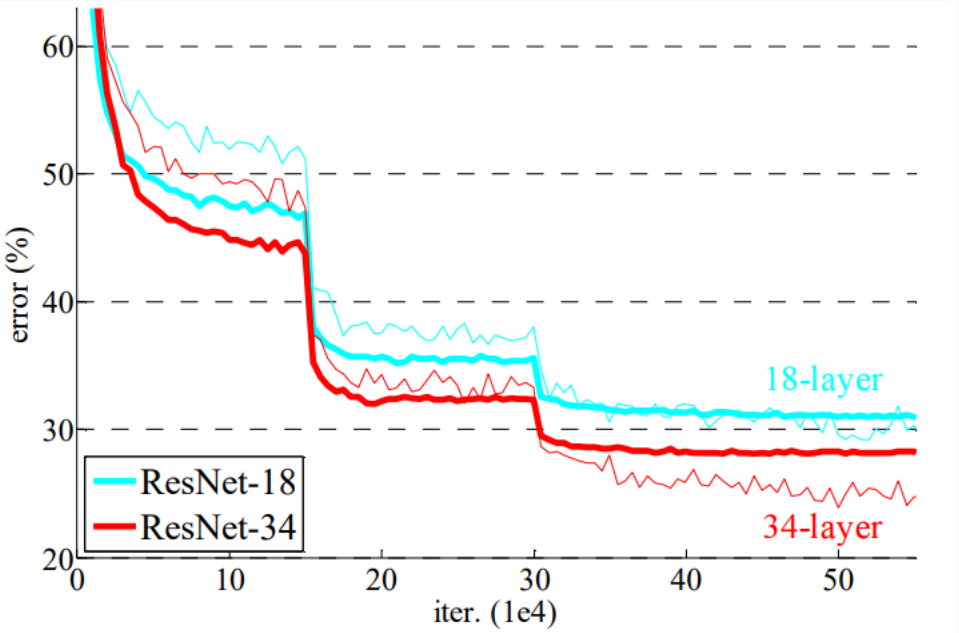
《大模型计算》课程团队

跳跃连接

$y = f(x)$



$y = x + f(x)$



He, Kaiming, et al. "Deep residual learning for image recognition." *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016.

归一化

❖ 信用卡逾期预测

姓别x1	年龄x2	余额x3	逾期y
0	28	3000	1
1	60	1000000000000	0

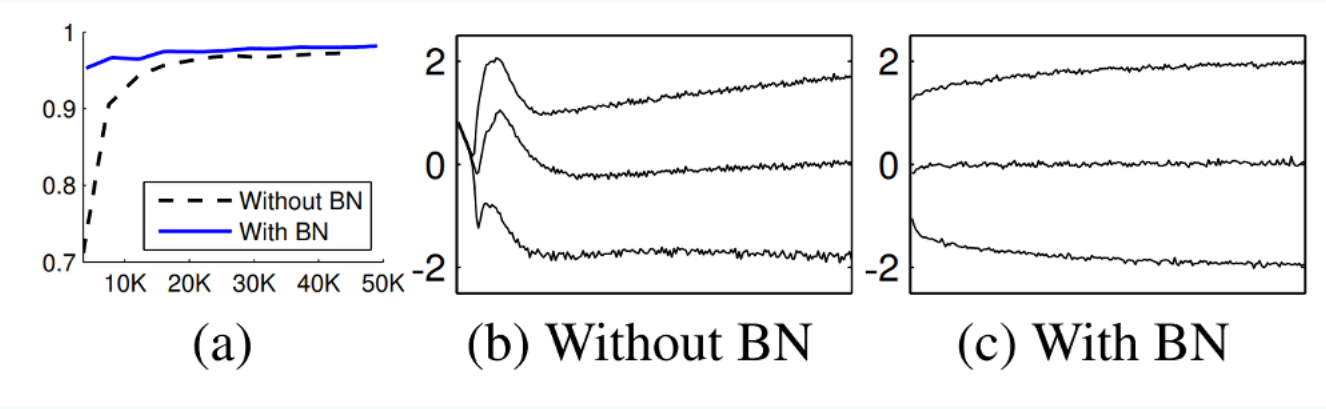
$$y = 0.1 * x1 - 0.1 * x2 + 0.01 * x3$$

只有特征x3会起作用

❖ 归一化：将所有数据变成0均值1标准差

层归一化

- 解决：“协方差漂移”（covariance shift）
- 提升：训练稳定性及速度
- 举例：考前一天修改核心概念



$$\mathbf{h}^t = f\left[\frac{\mathbf{g}}{\sigma^t} \odot \left(\mathbf{a}^t - \mu^t\right) + \mathbf{b}\right] \qquad \mu^t = \frac{1}{H}\sum_{i=1}^{H} a_i^t \qquad \sigma^t = \sqrt{\frac{1}{H}\sum_{i=1}^{H}\left(a_i^t - \mu^t\right)^2}$$

Ba, Jimmy Lei, Jamie Ryan Kiros, and Geoffrey E. Hinton. "Layer normalization." *arXiv preprint arXiv:1607.06450* (2016).

自然语言处理中的层归一化

- 输入：[N，T，D]
- mu、sigma：[N，T，1]
- a，b：[D]

```
def layer_norm(x, normalized_shape, gamma=None, beta=None, eps=1e-5):
    # 沿着最后一个维度（D）计算均值和方差，并保持维度以便广播
    mean = x.mean(dim=-1, keepdim=True) # 形状变为 [N, T, 1]
    var = x.var(dim=-1, keepdim=True, unbiased=False) # 形状变为 [N, T, 1]
    # 归一化: (x - mean) / sqrt(var + eps)
    x_normalized = (x - mean) / torch.sqrt(var + eps)
    # 如果提供了 gamma 和 beta，则进行仿射变换
    return gamma * x_normalized + beta
```

$$\mathbf{h}^t = f\left[\frac{\mathbf{g}}{\sigma^t} \odot \left(\mathbf{a}^t - \mu^t\right) + \mathbf{b}\right] \qquad \mu^t = \frac{1}{H}\sum_{i=1}^{H} a_i^t \qquad \sigma^t = \sqrt{\frac{1}{H}\sum_{i=1}^{H}\left(a_i^t - \mu^t\right)^2}$$

均方差归一化

- 与LN区别：去掉mean
- 计算更快
- 实际效果往往与LN相当或略优
  - 可能是因为均值易受离群值扰动?

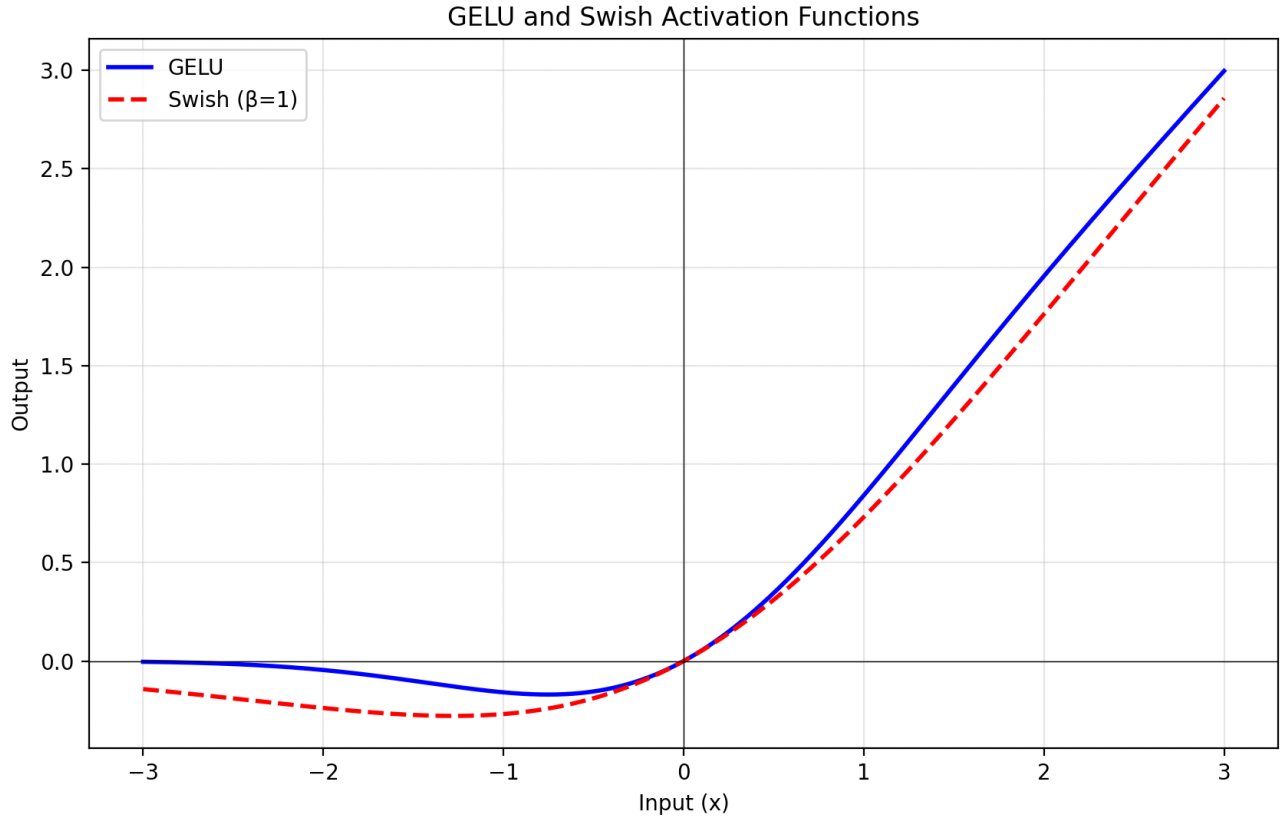
$$\bar{a}_i = \frac{a_i}{\mathrm{RMS}(\mathbf{a})} g_i, \quad \text{where } \mathrm{RMS}(\mathbf{a}) = \sqrt{\frac{1}{n}\sum_{i=1}^{n} a_i^2}.$$

Zhang, Biao, and Rico Sennrich. "Root mean square layer normalization." *Advances in neural information processing systems* 32 (2019).

激活函数：GeLU和Swish

$$\mathrm{GELU}(x) = x \cdot \Phi(x) = x \cdot \frac{1}{2}\left[1 + \mathrm{erf}\left(\frac{x}{\sqrt{2}}\right)\right]$$

$$\mathrm{Swish}(x) = x \cdot \sigma(\beta x) = \frac{x}{1 + e^{-\beta x}}$$



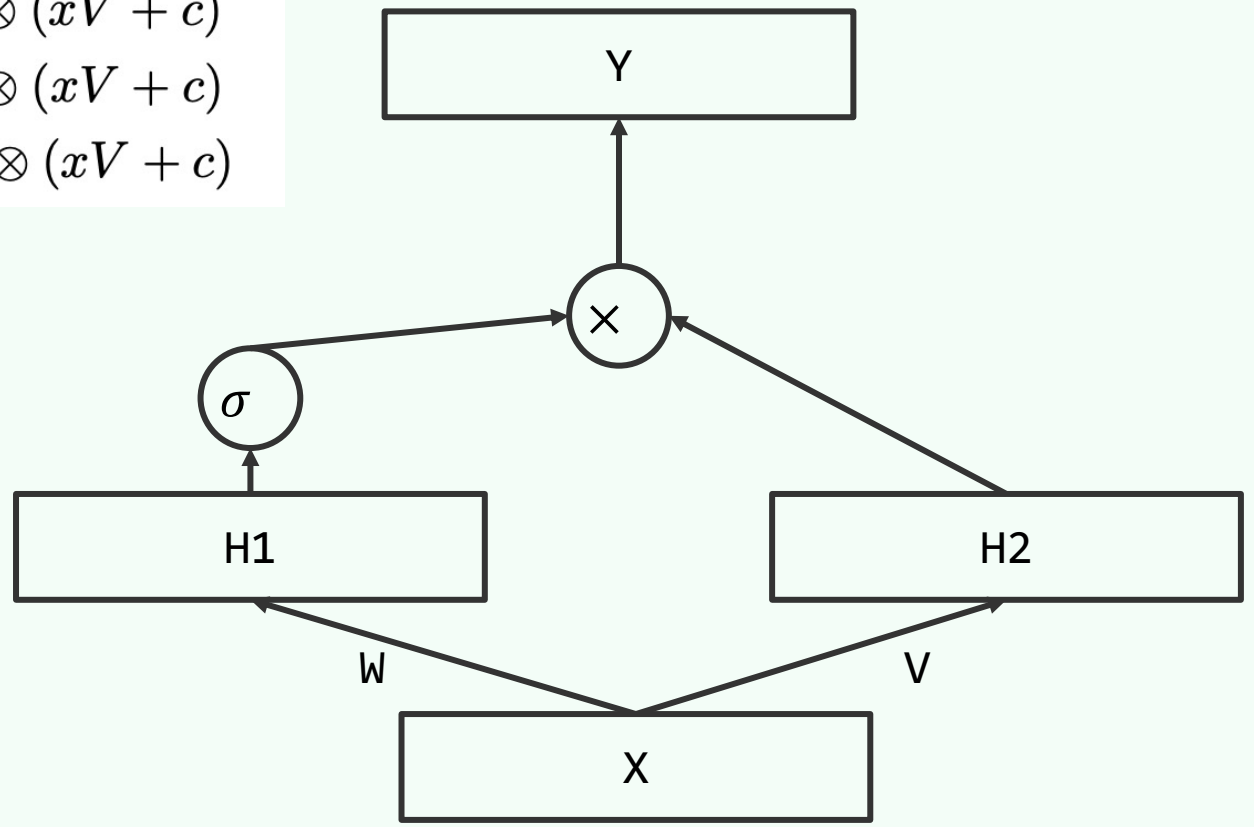
不单调

均非零

门控线性单元Gated Linear Unit (GLU)

$$\mathrm{ReGLU}(x, W, V, b, c) = \max(0, xW + b) \otimes (xV + c)$$
$$\mathrm{GEGLU}(x, W, V, b, c) = \mathrm{GELU}(xW + b) \otimes (xV + c)$$
$$\mathrm{SwiGLU}(x, W, V, b, c, \beta) = \mathrm{Swish}_\beta(xW + b) \otimes (xV + c)$$

- 为什么能好?
- 最大能给到平方的激活
- 创造离群值（outlier）



Shazeer, Noam. "Glu variants improve transformer." *arXiv preprint arXiv:2002.05202* (2020).
Dauphin, Yann N., et al. "Language modeling with gated convolutional networks." *International conference on machine learning*. PMLR, 2017.

Transformer

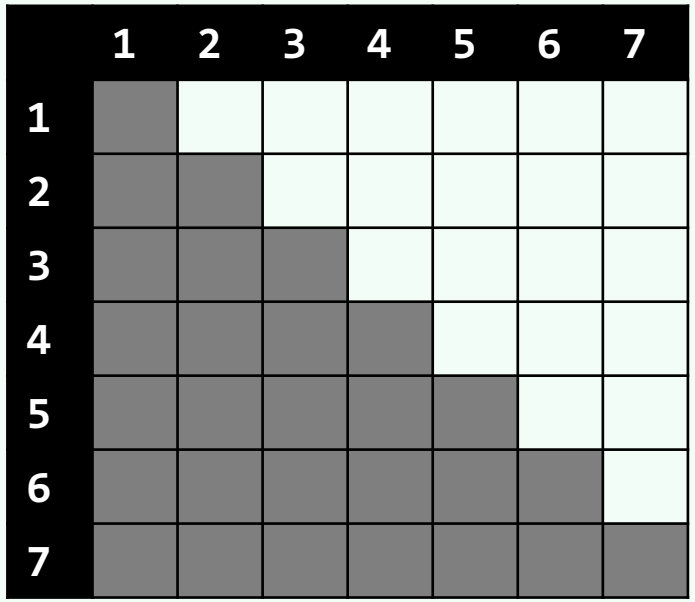
因果注意力

清华大学计算机系 陈键飞

《大模型计算》课程团队

问题

- 单个next word prediction任务的时间复杂度为$O(D^2T + DT^2)$
- 每增加一个词，要重算之前所有词的特征
- 思路：让前面词的特征不依赖于后面词的特征=>为注意力增加掩码（mask）



$$\text{Attention}(Q,K,V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}} + \boldsymbol{M}\right)V$$

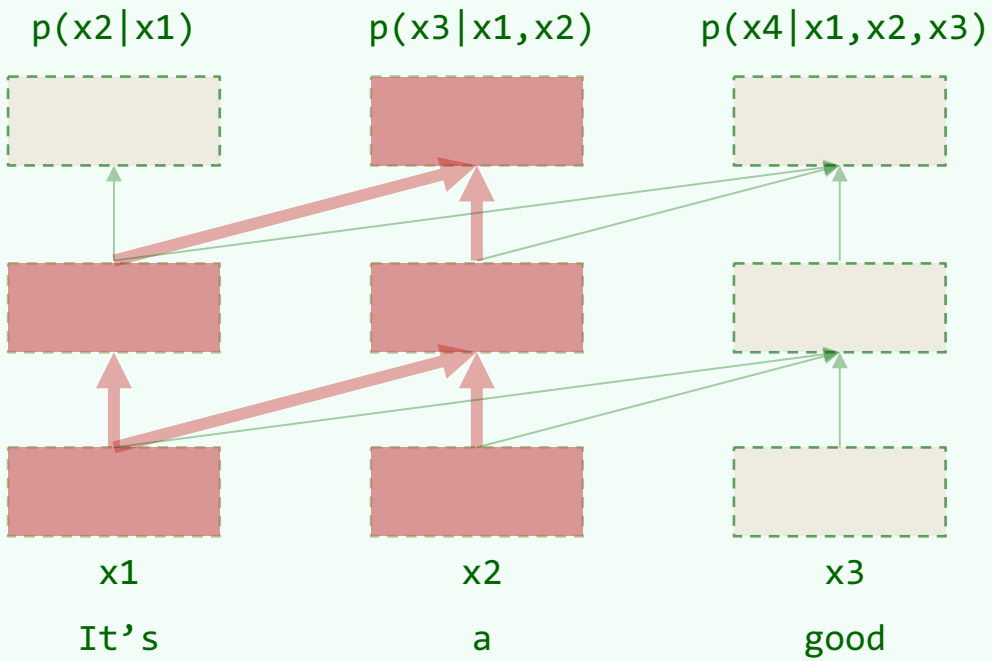
$$M_{ij} = \begin{cases} 0 & \text{if } i \geq j \\ -\infty & \text{otherwise} \end{cases}$$

qi只能看到kj（j<=i）

注意力只能按顺序计算

因果（causal）注意力

带因果注意力的transformer



- ❖ 每个词只能看到之前层的隐含表征
- ❖ 后面词不会影响之前词的计算结果

- ❖ 训练：并行计算所有条件概率$O(D^2T + DT^2)$
- ❖ 推理：从左到右依次计算每个塔$O(D^2 + DT)$/词

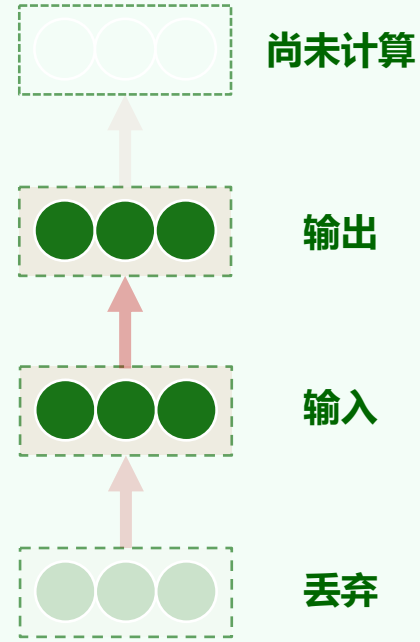
For each token xi

计算塔i和条件概率$p(x_{i+1}|x_{\leq i})$

采样$x_{i+1} \sim p(x_{i+1}|x_{\leq i})$

显存管理

❖ 显存中仅需存储对未来计算仍有用的张量



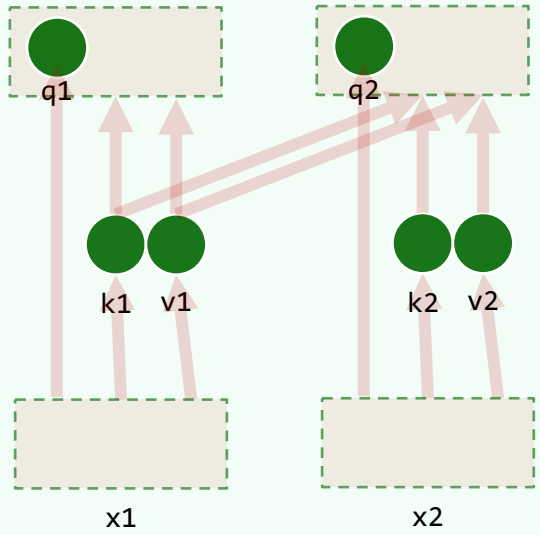

```
h2 = f(h1)
h3 = f(h2)
h4 = f(h3)
```

❖ 对Transformer来讲：

唯二对后续词元有用的张量是（K，V）

❖ 称作KV缓存（KV cache）

❖ 决定推理时的空间消耗



实现：注意力

```
Query:  [N, H, T_Q, D]
Key:    [N, H, T_K, D]
Value:  [N, H, T_K, D]
```

```python
def eager_attention_forward(module, query, key, value, attention_mask, scaling, **kwargs):
    attn_weights = torch.matmul(query, key.transpose(2, 3)) * scaling        #[N, H, T_Q, T_K]
    if attention_mask is not None:
        causal_mask = attention_mask[:, :, :, : key.shape[-2]]
        attn_weights = attn_weights + causal_mask

    attn_weights = nn.functional.softmax(attn_weights, dim=-1, dtype=torch.float32).to(query.dtype)
    attn_output = torch.matmul(attn_weights, value_states)                   #[N, H, T_Q, D]

    attn_output = attn_output.transpose(1, 2).contiguous()                   #[N, T_Q, H, D]

    return attn_output, attn_weights
```

https://github.com/huggingface/transformers/blob/f64354e89a55a456e024b301468da2dbb10e11b3/src/transformers/models/llama/modeling_llama.py#L171

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

实现：KV缓存

```
def forward(self, hidden_states, position_embeddings, attention_mask, past_key_values, cache_position):
    计算qkv，计算rope，得到query_states, key_states, value_states
    if past_key_values is not None:      # 将当前KV存入缓存，并从缓存中读取过去KV
        cache_kwargs = {"sin": sin, "cos": cos, "cache_position": cache_position}
        key_states, value_states = past_key_values.update(key_states, value_states, self.layer_idx, cache_kwargs)

    attention_interface: Callable = eager_attention_forward
    attn_output, attn_weights = attention_interface(self, query_states, key_states, value_states, …)

if use_cache and past_key_values is None:  # 初始化KV缓存
    past_key_values = DynamicCache(config=self.config)

if cache_position is None:
    past_seen_tokens = past_key_values.get_seq_length() if past_key_values is not None else 0
    cache_position: torch.Tensor = torch.arange(
        past_seen_tokens, past_seen_tokens + inputs_embeds.shape[1], device=inputs_embeds.device
)                                          # 本次forward的词元
```

https://github.com/huggingface/transformers/blob/7258ea44bc0c0a425a468f66f8559d1de8c4126d/src/transformers/models/llama/modeling_llama.py#L370C1-L377C14

实现：KV缓存

```
def update(self, key_states, value_states, cache_kwargs):    # [N, H, T, D]
"""
Update the key and value caches in-place, and return the necessary keys and value states.
"""
    self.keys = torch.cat([self.keys, key_states], dim=-2)   # 拼接上新的KV
    self.values = torch.cat([self.values, value_states], dim=-2)
    return self.keys, self.values                            # 返回过去所有KV
```

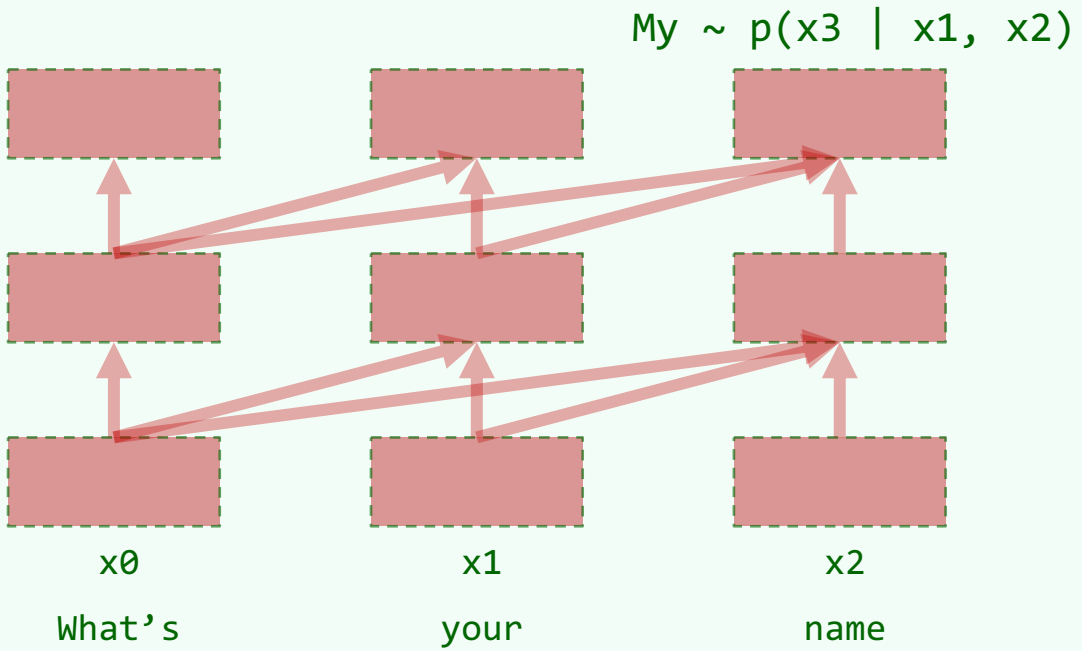
```
>>> from transformers import AutoTokenizer, AutoModelForCausalLM, DynamicCache
>>> model = AutoModelForCausalLM.from_pretrained("Qwen/Qwen2-0.5B-Instruct")
>>> tokenizer = AutoTokenizer.from_pretrained("Qwen/Qwen2-0.5B-Instruct")
>>> inputs = tokenizer(text="My name is Qwen2", return_tensors="pt")
>>> # Prepare a cache class and pass it to model's forward
>>> past_key_values = DynamicCache(config=model.config)
>>> outputs = model(**inputs, past_key_values=past_key_values, use_cache=True)
>>> outputs.past_key_values # access cache filled with key/values from generation
```

自回归解码：流程

预填充Prefill

```
inputs = tokenizer(text=”What’s your name")
model(inputs, past_key_values)

cache_position = [0, 3)
```

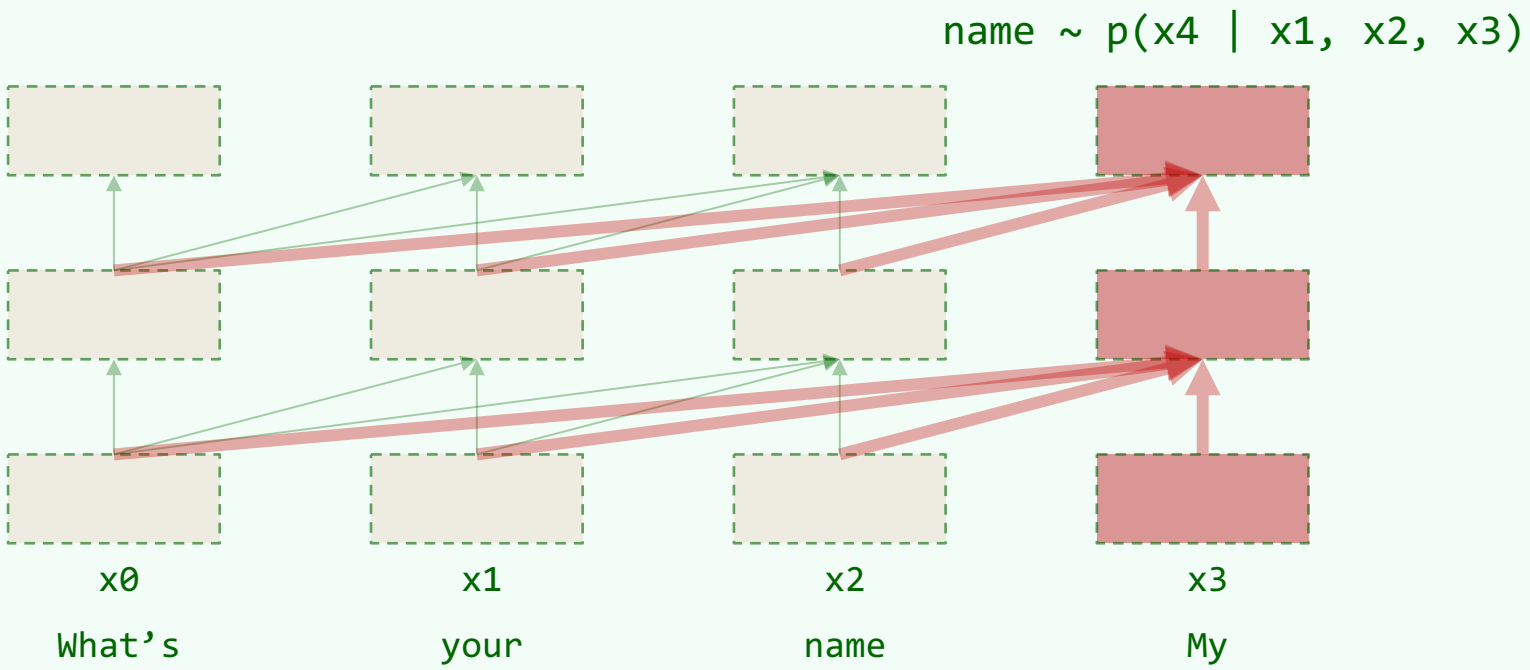


自回归解码：流程

解码Decode

```
model(inputs, past_key_values)

cache_position = [3, 4)
```



自回归解码：流程

解码Decode

```
model(inputs, past_key_values)

cache_position = [4, 5)
```