0.1 Relative Positional Embeddings

0.1.1 问题引入：为何 transformer 要加入位置信息？

现有的 Transformer 自注意力架构是 **Position-Agnostic（位置无关）** 的。transformer 中的 attention 机制主要依靠 Q, K, V 其公式为：$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d}})V$

场景 1：输入为 [A, B]

矩阵运算如下：

$$\begin{bmatrix} x_A \\ x_B \end{bmatrix} \times W_Q = \begin{bmatrix} x_A \cdot W_Q \\ x_B \cdot W_Q \end{bmatrix} = \begin{bmatrix} \mathbf{q_A} \\ q_B \end{bmatrix}$$

- 结果：矩阵第 1 行是 $\mathbf{q_A}$。

场景 2：输入为 [B, A]

矩阵运算如下：

$$\begin{bmatrix} x_B \\ x_A \end{bmatrix} \times W_Q = \begin{bmatrix} x_B \cdot W_Q \\ x_A \cdot W_Q \end{bmatrix} = \begin{bmatrix} q_B \\ \mathbf{q_A} \end{bmatrix}$$

- 结果：矩阵第 2 行是 $\mathbf{q_A}$。

结论：虽然 $\mathbf{q_A}$ 跑到了第二行，但它的数值内容是由 x_A 和 W_Q 决定的。因为 x_A 没变，W_Q 没变，所以 $\mathbf{q_A}$ 里的每一个浮点数都一模一样。

同理计算 attention 中的 K 时也会是同样的结果，因而输入句子的顺序不会改变 QK^T 的值。也就是说，不加入位置信息的话，transformer 将无法识别类似于 “我爱你” 与 “你爱我” 这样的句子的差异。

0.1.2 相对位置编码及其缺陷

在介绍相对位置编码及其缺陷前需要先介绍一下标准 transformer 中的时间复杂度问题。

普通的 transformer 存在一个缺陷：标准的 Attention 公式为：$\text{Self-Attention} = \text{Softmax}(\frac{\mathbf{QK}^T}{\sqrt{d}})\mathbf{V}$，需要计算所有 Query 和 Key 的两两内积，生成一个 $N \times N$ 的注意力矩阵，这导致其时间复杂度为 $O(N^2)$，在处理长序列时就会非常痛苦。

因而需要思考，**有没有办法能够降低时间复杂度?**

灵感：矩阵乘法的结合律

若忽略 Softmax 和缩放因子，标准注意力的计算过程：$\text{Result} = (\mathbf{QK}^T)\mathbf{V}$

这里三个矩阵的维度为：$\mathbf{Q}$: $(N \times d)$、$\mathbf{K}^T$: $(d \times N)$、$\mathbf{V}$: $(N \times d)$

标准做法（先算括号里）：

1. 先算 $\mathbf{A} = \mathbf{QK}^T$，得到一个 $N \times N$ 的矩阵（复杂度 $O(N^2)$）。
2. 再算 $\mathbf{AV}$，得到结果（复杂度 $O(N^2)$）。

线性注意力的思想（结合律）：先算后面两个矩阵 $\mathbf{K}^T\mathbf{V}$：$\text{Result} = \mathbf{Q}(\mathbf{K}^T\mathbf{V})$

1. 先算 $\mathbf{M} = \mathbf{K}^T\mathbf{V}$。
   - $\mathbf{K}^T$ 是 $d \times N$，$\mathbf{V}$ 是 $N \times d$。

- 结果 $\mathbf{M}$ 是一个 $d \times d$ 的小矩阵。
- 复杂度：$O(N)$。

2. 再算 $\mathbf{QM}$。
   - $\mathbf{Q}$ 是 $N \times d$，$\mathbf{M}$ 是 $d \times d$。
   - 复杂度：$O(N)$。

如果能做到这样，那么**时间复杂度就会从 $O(N^2)$ 变为 $O(N)$。**

现在存在一个问题：原始的 attention 中存在非线性变换 softmax 使得无法进行上述结合律变换：$\text{Self-Attention} = \text{Softmax}(\frac{\mathbf{QK}^T}{\sqrt{d}})\mathbf{V}$。为了解决这一问题我们需要使用核函数方法。

1. 符号定义 (Setup)

   假设我们有三个输入矩阵：**Query (Q)**: $(L \times d)$、**Key (K)**: $(L \times d)$、**Value (V)**: $(L \times d)$

   其中 L 是序列长度，d 是特征维度。将第 i 行向量分别记为 $\mathbf{q}_i, \mathbf{k}_i, \mathbf{v}_i$（均为 d 维行向量）。

2. 原始 Attention

   标准 Attention 计算第 i 个 Token 的输出 $\mathbf{o}_i$ 的公式为：$\mathbf{o}_i = \sum_{j=1}^{L} \alpha_{i,j}\mathbf{v}_j$

   其中 $\alpha_{i,j}$ 是归一化后的注意力权重（Softmax）：$\alpha_{i,j} = \frac{\text{sim}(\mathbf{q}_i,\mathbf{k}_j)}{\sum_{l=1}^{L}\text{sim}(\mathbf{q}_i,\mathbf{k}_l)}$

   在原始 Transformer 中，相似度函数 sim 是指数函数（即 Softmax 的分子）：

   $$\text{sim}(\mathbf{q}_i, \mathbf{k}_j) = \exp\left(\frac{\mathbf{q}_i \mathbf{k}_j^T}{\sqrt{d}}\right)$$

   要计算分母 $\sum_{l=1}^{L}\exp(\ldots)$，对于每一个 i，都要和所有的 L 个 k 进行计算。总共有 L 个 i，所以总复杂度是 $O(L^2)$。

3. 核方法引入 (Kernel Trick)

   假设存在一个特征映射函数 $\phi : \mathbb{R}^d \to \mathbb{R}^D$，使得我们可以用内积来近似（或替代） exp 函数：$\text{sim}(\mathbf{q}_i, \mathbf{k}_j) \approx \phi(\mathbf{q}_i)\phi(\mathbf{k}_j)^T$

   注：为了保证注意力分数为正，$\phi(\cdot)$ 的值通常要求是非负的（例如在《Transformers are RNNs: Fast Autoregressive Transformers with Linear Attention》中： $\phi(x) = \text{elu}(x) + 1$*）。*

   将这个近似代入 $\mathbf{o}_i$ 的公式：

   $$\mathbf{o}_i = \frac{\sum_{j=1}^{L}\left(\phi(\mathbf{q}_i)\phi(\mathbf{k}_j)^T\right)\mathbf{v}_j}{\sum_{l=1}^{L}\phi(\mathbf{q}_i)\phi(\mathbf{k}_l)^T}$$

4. 结合律变换

   分子项：$\text{Numerator}_i = \sum_{j=1}^{L}\left(\phi(\mathbf{q}_i)\phi(\mathbf{k}_j)^T\right)\mathbf{v}_j$

   由于矩阵乘法满足结合律，且 $\phi(\mathbf{q}_i)$ 与求和下标 j 无关，我们可以把它提到求和号外面：$\text{Numerator}_i = \phi(\mathbf{q}_i)\left(\sum_{j=1}^{L}\phi(\mathbf{k}_j)^T\mathbf{v}_j\right)$

   同理，分母项：$\text{Denominator}_i = \sum_{l=1}^{L}\phi(\mathbf{q}_i)\phi(\mathbf{k}_l)^T = \phi(\mathbf{q}_i)\left(\sum_{l=1}^{L}\phi(\mathbf{k}_l)^T\right)$

5. 线性复杂度算法流程

   通过上面的变换，可将计算过程重构为以下两步：

   第一步：

这一步只涉及 K 和 V，与 Q 无关。我们可以遍历一次序列，把所有的 Key 和 Value 聚合起来。定义矩阵 $\mathbf{S}_{KV}$：

$$\mathbf{S}_{KV} = \sum_{j=1}^{L} \underbrace{\phi(\mathbf{k}_j)^T}_{d\times 1} \underbrace{\mathbf{v}_j}_{1\times d} \in \mathbb{R}^{d\times d}$$

复杂度分析：需要遍历 L 个位置。在每个位置，计算 $\phi(\mathbf{k}_j)^T\mathbf{v}_j$，复杂度是 $O(d^2)$。这一步的总复杂度：$O(Ld^2)$。

第二步：

定义向量 $\mathbf{z}_K$：

$$\mathbf{z}_K = \sum_{l=1}^{L} \phi(\mathbf{k}_l)^T \in \mathbb{R}^{d\times 1}$$

现在，对于任意的 Query $\mathbf{q}_i$，我们不需要再去找 L 个 Key 了，直接查上面算好的全局矩阵 $\mathbf{S}_{KV}$ 和向量 $\mathbf{z}_K$。

$$\mathbf{o}_i = \frac{\phi(\mathbf{q}_i)\mathbf{S}_{KV}}{\phi(\mathbf{q}_i)\mathbf{z}_K}$$

复杂度分析：

- 分子计算：向量 $\phi(\mathbf{q}_i)$ $(1\times d)$ 乘以矩阵 $\mathbf{S}_{KV}$ $(d\times d)$。计算量 $O(d^2)$。
- 分母计算：向量 $\phi(\mathbf{q}_i)$ $(1\times d)$ 点乘向量 $\mathbf{z}_K$ $(d\times 1)$。计算量 $O(d)$。
- 我们需要对 L 个 Query 做这个操作。
- 这一步的总复杂度：$O(Ld^2)$。

6. 最终结果

将两步合并，线性注意力（Linear Attention）的总公式为：

$$\text{LinearAttention}(\mathbf{Q},\mathbf{K},\mathbf{V}) = \frac{\phi(\mathbf{Q})\left(\phi(\mathbf{K})^T\mathbf{V}\right)}{\phi(\mathbf{Q})\left(\sum_j \phi(\mathbf{K})_j^T\right)}$$

总时间复杂度为：$O(Ld^2)+O(Ld^2)=O(Ld^2)$

现在讲解相对位置编码及其缺陷：

当前很多模型（如T5）会采用相对位置编码来引入位置信息。在计算注意力时，会采用如下公式：

$$\text{Score}(Q,K) = \text{Softmax}\left(QK^T + \mathbf{B}\right)$$

这里的 $\mathbf{B}$ 是一个 $L\times L$ 的矩阵，其中 $\mathbf{B}_{i,j}$ 是一个标量，代表 i 和 j 的相对距离偏置。但这样会直接导致无法使用线性注意力，从而没法优化时间复杂度。

在 T5 或其他使用加法相对位置编码的模型中，Attention 分数（Score）的计算公式是：

$$Score_{i,j} = \underbrace{\mathbf{q}_i\cdot\mathbf{k}_j}_{\text{内容相似度}} + \underbrace{\mathbf{B}_{i,j}}_{\text{位置偏置}}$$

易知：

$$\begin{aligned}\text{AttentionWeight}_{i,j} &\propto \exp(Score_{i,j}) \\ &= \exp(\mathbf{q}_i\cdot\mathbf{k}_j + \mathbf{B}_{i,j}) \\ &= \exp(\mathbf{q}_i\cdot\mathbf{k}_j)\cdot\exp(\mathbf{B}_{i,j})\end{aligned}$$

记：$P_{i,j} = \exp(\mathbf{B}_{i,j})$

利用核方法可知

$$\exp(\mathbf{q}_i \cdot \mathbf{k}_j) \approx \phi(\mathbf{q}_i) \cdot \phi(\mathbf{k}_j)^T$$

因而：

$$\text{AttentionWeight}_{i,j} \approx \phi(\mathbf{q}_i) \cdot \phi(\mathbf{k}_j)^T \cdot P_{i,j}$$

计算输出 $\mathbf{o}_i$：

$$\mathbf{o}_i = \sum_{j=1}^{L} \left(\phi(\mathbf{q}_i) \cdot \phi(\mathbf{k}_j)^T \cdot P_{i,j}\right)\mathbf{v}_j = \phi(\mathbf{q}_i) \cdot \sum_{j=1}^{L} \left(\phi(\mathbf{k}_j)^T \cdot \mathbf{v}_j \cdot \mathbf{P_{i,j}}\right)$$

可看出求和的结果取决于 i 是几。由于对不同的 i ，$P_{i,j}$ 有不同值。因而必须针对每一个 i，都重新去遍历一遍所有的 j，这导致复杂度又回到了 $O(L^2)$。

0.1.3 Rotary position embedding（RoPE）

为了处理上述问题（能体现位置信息，同时可使用线性注意力以降低复杂度），RoPE 诞生辣！在了解 RoPE 的具体细节前先知道它的本质是有必要的。

回顾传统 transformer 中的 Self-Attention 计算过程，其中核心的一环是计算注意力分数，假设现在要计算一句话中第 m 和第 n 个位置词语的注意力分数，没有位置信息时

$$\text{AttentionScore} = \mathbf{q}_m^T \cdot \mathbf{k}_n$$

若采用绝对位置编码加入位置信息，则有：

$$\text{AttentionScore} = (\mathbf{q}_m + \mathbf{p}_m)^T \cdot (\mathbf{k}_n + \mathbf{p}_n)$$

而 RoPE 则是通过旋转位置编码来加入位置信息，它能够体现句子中词语的“相对位置”：

$$\text{AttentionScore} = (\mathcal{R}_m\mathbf{q}_m) \cdot (\mathcal{R}_n\mathbf{k}_n)$$

其中 $\mathcal{R}_m, \mathcal{R}_n$ 均为旋转矩阵，它表示旋转 $m\theta$ 角度。熟悉线代的朋友们应该还快就会发现，对于旋转矩阵我们有：$\mathcal{R}_m^T = \mathcal{R}_{-m}$ 、 $\mathcal{R}_a \cdot \mathcal{R}_b = \mathcal{R}_{a+b}$ 。那么对于 RoPE：

$$\begin{aligned}\text{AttenionScore} &= (\mathcal{R}_m\mathbf{q}_m)^T(\mathcal{R}_n\mathbf{k}_n) \\ &= \mathbf{q}_m^T\mathcal{R}_m^T\mathcal{R}_n\mathbf{k}_n \\ &= \mathbf{q}_m^T\mathcal{R}_{n-m}\mathbf{k}_n\end{aligned}$$

此时就能发现无论 m 和 n 怎么变，只要它们的相对位置 $m-n$ 不变，那么注意力分数就不会改变。这便是 RoPE 的强大之处。接下来我们来看在 $Roformer$ 这篇论文中，作者是如何介绍旋转位置编码这一突破性技术的。

通过上述我们知道，如果我们希望模型能捕捉“相对位置”，那么 q_m 和 k_n 的内积结果，应该只与它们之间的相对距离 $(m-n)$ 有关，而与绝对位置 m, n 无关。也就是说，下述公式应该被满足：

$$\langle f_q(x_m, m), f_k(x_n, n)\rangle = g(x_m, x_n, m-n)$$

- 左边：是我们在 Attention 里实际算的内积。f_q 和 f_k 是我们需要寻找的编码函数（把词向量 x 和位置 m 融合起来的函数）。
- 右边：是我们期望的结果。函数 g 的参数里，位置信息只以 $m-n$ 的形式出现。

二维场合（$d_{model} = 2$）

把二维向量 (x_1, x_2) 看作复平面上的一个点 $x_1 + ix_2$。f_q 和 f_k 定义如下：

$$f_q(x_m, m) = (W_q x_m)e^{im\theta}$$

$$f_k(x_n, n) = (W_k x_n)e^{in\theta}$$

在复数中，两个向量 q 和 k 的内积，等于 q 乘以 k 的共轭（k^*）的实部，故：

$$\begin{aligned}\text{Inner Product} = \mathrm{Re}[f_q \cdot f_k^*] &= \mathrm{Re}[\underbrace{(W_q x_q)e^{im\theta}}_{\text{Query}} \cdot \underbrace{((W_k x_k)e^{in\theta})^*}_{\text{Key 的共轭}}] \\ &= \mathrm{Re}[(W_q x_q)e^{im\theta} \cdot (W_k x_k)^* e^{-in\theta}] \\ &= \mathrm{Re}[\underbrace{(W_q x_q)(W_k x_k)^*}_{\text{原本的Score}} \cdot \underbrace{e^{i(m-n)\theta}}_{\text{相对位置}}]\end{aligned}$$

其中 $Re[\cdot]$ 是复数的实部。可以看到，我么确实找到了上述讨论的函数 f 和 g 。

由于

$$\mathbf{h} = W_q x_m = \begin{pmatrix} W_q^{(11)} & W_q^{(12)} \\ W_q^{(21)} & W_q^{(22)} \end{pmatrix} \begin{pmatrix} x_m^{(1)} \\ x_m^{(2)} \end{pmatrix} = \begin{pmatrix} h^{(1)} \\ h^{(2)} \end{pmatrix}$$

我们将上面的向量 $\mathbf{h}$ 视为复平面上的复数 z，同时利用欧拉公式展开 $e^{im\theta}$：

1. 向量转复数： $z = h^{(1)} + i \cdot h^{(2)}$
2. 旋转项展开： $e^{im\theta} = \cos m\theta + i \sin m\theta$

经过复数乘法后得到：

$$\begin{aligned}z' &= z \cdot e^{im\theta} \\ &= (h^{(1)} + ih^{(2)})(\cos m\theta + i \sin m\theta)\end{aligned}$$

将上述公式展开后可得：

$$z' = \underbrace{(h^{(1)} \cos m\theta - h^{(2)} \sin m\theta)}_{\text{新的实部}} + i \cdot \underbrace{(h^{(1)} \sin m\theta + h^{(2)} \cos m\theta)}_{\text{新的虚部}}$$

我们将这个结果还原回二维向量形式，新的向量 $\mathbf{h}'$ 为：

$$\mathbf{h}' = \begin{pmatrix} h^{(1)} \cos m\theta - h^{(2)} \sin m\theta \\ h^{(1)} \sin m\theta + h^{(2)} \cos m\theta \end{pmatrix}$$

即：

$$\mathbf{h}' = \begin{pmatrix} \cos m\theta & -\sin m\theta \\ \sin m\theta & \cos m\theta \end{pmatrix} \begin{pmatrix} h^{(1)} \\ h^{(2)} \end{pmatrix}$$

最后便可得到：

$$f_q(x_m, m) = \begin{pmatrix} \cos m\theta & -\sin m\theta \\ \sin m\theta & \cos m\theta \end{pmatrix} \begin{pmatrix} W_q^{(11)} & W_q^{(12)} \\ W_q^{(21)} & W_q^{(22)} \end{pmatrix} \begin{pmatrix} x_m^{(1)} \\ x_m^{(2)} \end{pmatrix}$$

f_k 同理可得。

General form（$d_{model} = 768$ 或 $d_{model} = 4096$ 等）

现假设 $d_{model} = d$ 那么对于某个位置上的一个 token： $\mathbf{x_m} = [x_1, x_2, x_3, x_4, \ldots, x_d]$，我们依然希望有：

$$f_{\{q,k\}}(x_m, m) = R^d_{\Theta,m} W_{\{q,k\}} x_m$$

那么可以设计正交阵：

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

它是一个块对角矩阵，将 d 维空间切分为 $d/2$ 个子空间，每个子空间独立旋转。

接着我们可以验证 Query (q_m) 和 Key (k_n) 的内积：

$$q_m^\top k_n = (R^d_{\Theta,m} W_q x_m)^\top (R^d_{\Theta,n} W_k x_n) = x_m^\top W_q^\top R^d_{\Theta,n-m} W_k x_n$$

这依然满足 $\langle f_q(x_m, m), f_k(x_n, n) \rangle = g(x_m, x_n, m-n)$。

Properties of RoPE

两大性质分别是：远程衰减（Long-term Decay）和与线性 Attention 的兼容性（RoPE with Linear Attention）。接下来逐一分析。

1. **Long-term Decay**

与之前一样，我们从二维场合 $d_{model} = 2$ 的情况入手，并逐步扩展到一般场合。同时，我们不会采用原文那种比较数学的方式去证明 upper bound 总体上会逐渐减小；我会从一种更好理解的方式说明该性质。

$d_{model} = 2$ 的情形

与先前一样，我们有 $Query$：q_m 和 Key：k_n，由于 $d_{model} = 2$，我们可将它们写为极坐标形式：

$$\mathbf{q} = \begin{pmatrix} r_q \cos\alpha \\ r_q \sin\alpha \end{pmatrix}, \quad \mathbf{k} = \begin{pmatrix} r_k \cos\beta \\ r_k \sin\beta \end{pmatrix}$$

现在进行 RoPE 操作，位置 m 转 $m\theta$，位置 n 转 $n\theta$，可得：

$$\mathbf{q}' = \begin{pmatrix} r_q \cos(\alpha + m\theta) \\ r_q \sin(\alpha + m\theta) \end{pmatrix}, \quad \mathbf{k}' = \begin{pmatrix} r_k \cos(\beta + n\theta) \\ r_k \sin(\beta + n\theta) \end{pmatrix}$$

做内积后可得：

$$\text{AttentionScore} = r_q r_k \cos\left((\alpha - \beta) + (m-n)\theta\right)$$

因此：

$$\text{RoPE}(m, n) \propto \cos((m-n)\theta)$$

这时我们自然会觉得奇怪，这玩意是个周期函数，也没远程衰减性啊。确实如此，但真正的模型中也不可能有 $d_{model} = 2$ 这种情况。我们继续看一般形式下的 RoPE 。

General form

现假定 $d_{model} = d$ 且 d 为偶数，由前述我们知道：

$$\text{AttenionScore} = (\mathcal{R}_m \mathbf{q}_m)^T (\mathcal{R}_n \mathbf{k}_n)$$

而旋转矩阵 $\mathcal{R}_{m,n}$ 是一个巨大的分块对角阵，其中的每一块都是一个 (2×2) 的旋转矩阵。很自然的一个想法是，我们将维度为 $d_{model} = d$ 的 $Query$：q_m 和 Key：k_n 也分成 $d/2$ 份，其中每一份都是二维的。例如：$d_{model} = 256$ 那么原本 q_m 是一个 (1×256) 维的向量，现在我给它分成：$q_m = (q_{m,1} \cdots q_{m,128})$，其中每一个 $q_{m,j}$ 都是一个二维向量。

现在与一维场合一样，我们将每一个小分量都写成极坐标形式

$$\mathbf{q}_{m,j} = \begin{pmatrix} r_{q,j}\cos(\alpha_j) \\ r_{q,j}\sin(\alpha_j) \end{pmatrix}, \quad \mathbf{k}_{n,j} = \begin{pmatrix} r_{k,j}\cos(\beta_j) \\ r_{k,j}\sin(\beta_j) \end{pmatrix}$$

位置 m 的 Query 逆时针转 $m\theta_j$；位置 n 的 Key 逆时针转 $n\theta_j$：

$$\mathbf{q}'_{m,j} = \begin{pmatrix} r_{q,j}\cos(\alpha_j + m\theta_j) \\ r_{q,j}\sin(\alpha_j + m\theta_j) \end{pmatrix}, \quad \mathbf{k}'_{n,j} = \begin{pmatrix} r_{k,j}\cos(\beta_j + n\theta_j) \\ r_{k,j}\sin(\beta_j + n\theta_j) \end{pmatrix}$$

做内积后可得第 j 组分量的注意力分数为：

$$\text{AttenionScore}_j = r_{q,j}r_{k,j}\cos\left((\alpha_j - \beta_j) + (m-n)\theta_j\right) \propto \cos((m-n)\theta_j)$$

那么可知，总的注意力分数为：

$$\text{Total Score} = \sum_{j=0}^{d/2-1} \text{Score}_j \propto \sum_{j=0}^{d/2-1} \cos((m-n)\theta_j)$$

现在我们再来看一下远程衰减是怎么发生的，原文中设置 $\theta_i = 10000^{-2i/d}$，它是一个递减的序列，为了方便讨论，我们假设只取三个值：$\theta_0 = 1.0$（高频，转得快）、$\theta_1 = 0.3$（中频，转得中等）、$\theta_2 = 0.05$（低频，转得极慢）。记：$\Delta = m - n$。以一种非常不严谨的方式，我们可以将注意力分数写为：

$$S(\Delta) = \cos(\Delta \cdot 1.0) + \cos(\Delta \cdot 0.3) + \cos(\Delta \cdot 0.05)$$

接下来，当 $\Delta = 0$ 时 $S(0) = \mathbf{3.0}$；当 $\Delta = 2$ 时 $S(2) = \mathbf{1.4}$；当 $\Delta = 50$ 时 $S(50) = \mathbf{-0.6}$。可以发现，相对举例越远，注意力分数越低。

这时会有一些注意力惊人的盆友说：当 $\Delta = 20$ 时 $S(20) = \mathbf{1.4} = S(2)$ 呀。确实如此，我们可以看论文中远程衰减的原图就知道，远程衰减本来就是振荡下降的。

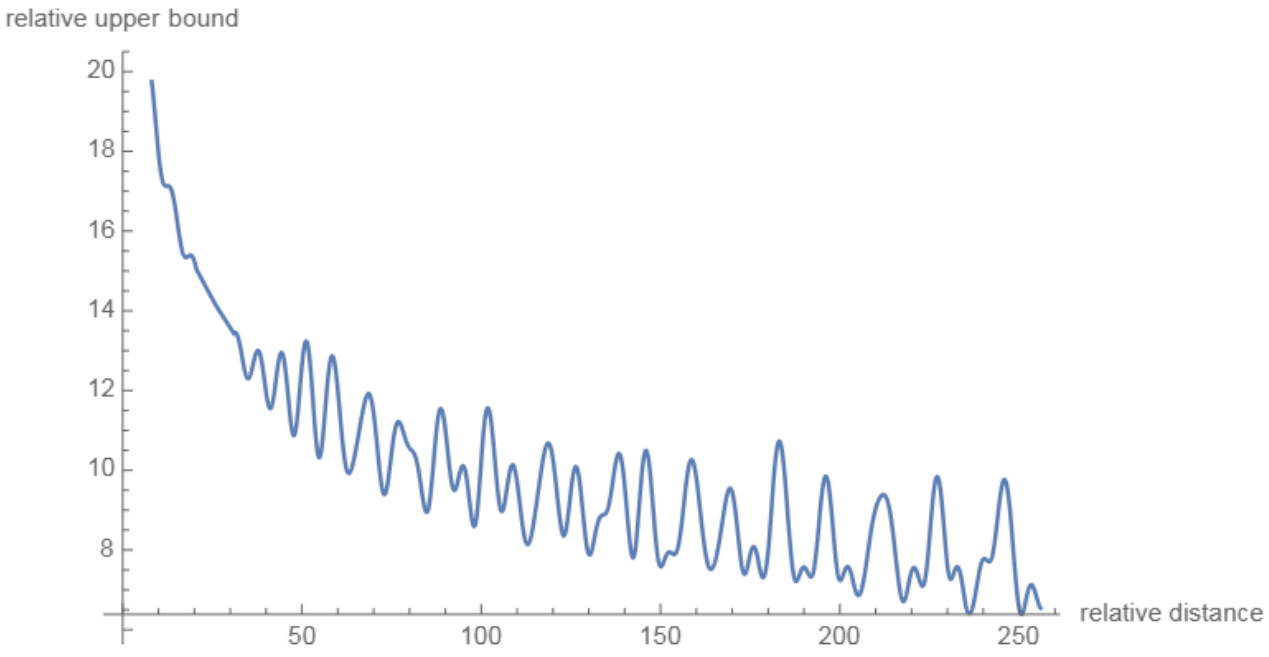

Figure 2: Long-term decay of RoPE.

2. RoPE with Linear Attention

接下来说 RoPE 的另一个性质，也就是它可以兼容线性注意力，从而降低复杂度。我们遵从原文的符号来说明这个性质。

首先，与先前讨论的的一样，我们可以通过核函数方法将 softmax 替换为特征映射函数 $\phi(\cdot)$ 和 $\varphi(\cdot)$。

$$Attention(Q,K,V)_m = \frac{\sum_{n=1}^{N} \phi(q_m)^T \varphi(k_n) v_n}{\sum_{n=1}^{N} \phi(q_m)^T \varphi(k_n)}$$

由于 RoPE 是通过旋转矩阵 R 来实现的，作者选择将旋转操作直接作用于特征映射后的向量 $\phi(q)$ 和 $\varphi(k)$ 上。

$$Attention(Q,K,V)_m = \frac{\sum_{n=1}^{N} (R^d_{\Theta,m}\phi(q_m))^T (R^d_{\Theta,n}\varphi(k_n)) v_n}{\sum_{n=1}^{N} \phi(q_m)^T \varphi(k_n)}$$

注意到我们只在 attention 的分子上进行了旋转操作来加入位置信息，作者解释到，不在分母上加旋转是因为如果加上旋转，分母的求和结果可能会变成 0 甚至负数。这会导致数值极其不稳定，甚至无法计算。

RoPE 的工程实现问题

理论部分已经结束，现在讨论实现 RoPE 会碰到的问题。

由前面理论部分的讨论可知，旋转操作是一个 $d \times d$ 的矩阵 R^d_Θ 乘以向量 $\mathbf{x}$。然而，由于 R^d_Θ 是一个巨大且稀疏的矩阵（绝大部分元素都是 0），直接应用矩阵乘法不仅浪费显存，而且计算效率低下。因此我们需要一种更高效的实现方式来降低计算复杂度。

```python
class RotaryEmbedding(nn.Module):
    def __init__(self, context_length: int, dim: int, theta: float = 10000.0):
        super().__init__()
        self.register_buffer(
            "_freq_cis_cache",
            RotaryEmbedding._init_cache(context_length, dim, theta), persistent=False
        )

    @staticmethod
    def _init_cache(context_length: int, dim: int, theta: float) -> Float[Tensor, " 2 context_length half_dim"]:
        assert dim % 2 == 0

        d = torch.arange(0, dim, 2) / dim
        freqs = theta ** -d
        t = torch.arange(context_length)

        freqs = einsum(t, freqs, "t, f -> t f")

        cos, sin = torch.cos(freqs), torch.sin(freqs)
        return torch.stack((cos, sin))

    def forward(self, x: Float[Tensor, " ... seq d"], pos_ids: Int[Tensor, " ... seq"]) -> Float[Tensor, " ... seq d"]:
        x1, x2 = rearrange(x, '... (half_d y) -> y ... half_d', y=2)

        # Standard
        # cos, sin = self._freq_cis_cache[:, pos_ids, :]

        # einx
        cos, sin = einx.get_at('cos_sin [pos] half_dim, ... -> cos_sin ... half_dim', self._freq_cis_cache, pos_ids)

        # 2D rotation matrix applied to pairs in x
        x1_rot = cos * x1 - sin * x2
        x2_rot = sin * x1 + cos * x2
```

```
        result = einx.rearrange('... x_half, ... x_half -> ... (x_half (1 + 1))', x1_rot, x2_rot).contiguous()
        return result

    def extra_repr(self):
        return f"context_length={self._freq_cis_cache.shape[0]}, dim/2={self._freq_cis_cache.shape[1]}"
```