Early LLM (transformer) layers pay attention to many tokens to gain context, later layers focus on fewer ”important” tokens

早期 LLM (Transformer) 层关注许多令牌以获取上下文，而后期层则专注于少数“重要”令牌

Vuk Rosić

vukrosic

For the impatient experts, I will explain the conclusion of my experiment immediately (beginners, don't worry, I will explain everything later step by step).
对于心急的专家，我会立即解释实验结论（初学者不用担心，稍后我会一步步解释）。

I pre-trained a small 80 million parameter LLM on 8M text tokens and looked at the attention's **key** vectors:
我预训练了一个包含 8000 万参数的小型 LLM，使用了 800 万个文本令牌，并观察了注意力的**键（Key）**向量：

We can see that L2 Norm of attention keys in early layers is becoming lower as LLM trains, which means LLM is paying attention to more tokens and taking context from them.
我们可以看到，随着 LLM 的训练，早期层注意力键的 L2 范数正在变低，这意味着 LLM 正在关注更多的令牌并从中获取上下文。

L2 Norm has an effect of making softmax probabilities more uniform (similar), so LLM is paying attention to more tokens equally.
L2 范数具有使 softmax 概率更均匀（相似）的效果，因此 LLM 会平等地关注更多的令牌。

In later layers L2 Norm is becoming higher, which means LLM is focusing on fewer ”important” tokens.
在后期层中，L2 范数变得更高，这意味着 LLM 正在专注于少数“重要”的令牌。

Now let me explain everything step by step
现在让我一步步解释这一切

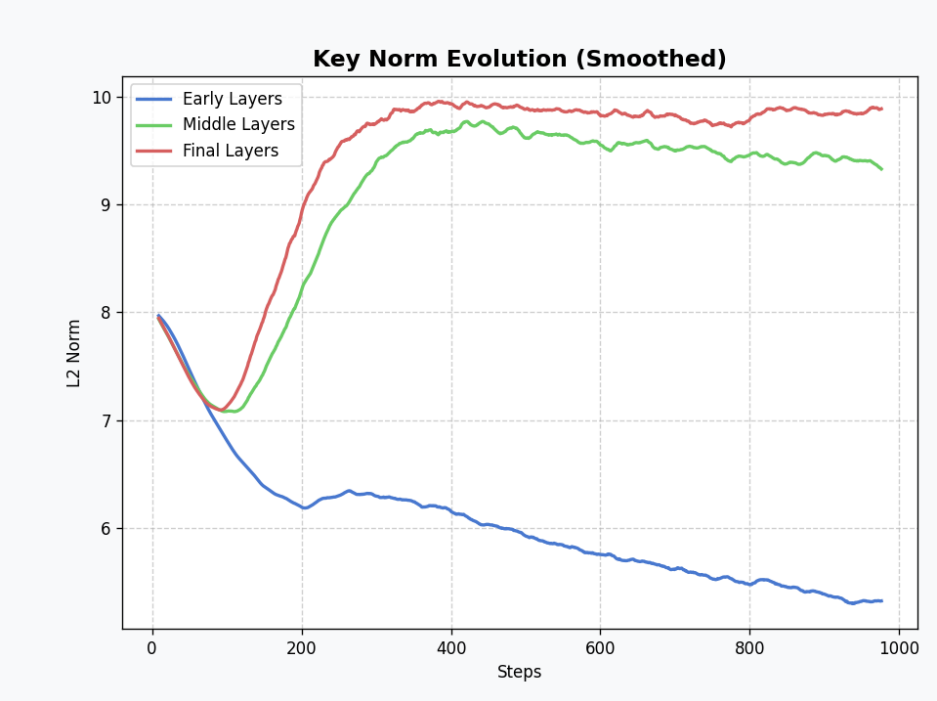


图 1: Evolution of the average Key Norm over ∼1000 training steps. ”Early Layers” = layers 0-1, ”Middle Layers” = layers 10-11, ”Final Layers” = layers 20-21.
∼1000 个训练步长内平均键范数（Key Norm）的演变。“早期层”= 第 0-1 层，“中间层”= 第 10-11 层，“最终层”= 第 20-21 层。

Prerequisites
预备知识

1. Dot Products Scale with Dimension
1. 点积随维度缩放

The dot product of two vectors measures how much they ”agree” - how well they point in the same direction. You compute it by multiplying corresponding numbers and adding them up.
两个向量的点积衡量它们有多“一致”——它们朝同一方向指向的程度。你通过将相应的数字相乘并将它们相加来计算它。

Example with 2D vectors:
二维向量示例：

Take $a = [2, 3]$ and $b = [4, 1]$:
取 $a = [2, 3]$ 和 $b = [4, 1]$：

$$a \cdot b = (2 \times 4) + (3 \times 1) = 8 + 3 = 11$$

The result is a single number. A **large positive** dot product means the vectors point in a similar direction; a value **near zero** means they are unrelated; a **large negative** value means they point in opposite directions.
结果是一个单一的数字。**大的正**点积意味着向量指向相似的方向；**接近零**的值意味着它们无关；**大的负**值意味着它们指向相反的方向。

Key insight: If you add more dimensions (more numbers to each vector), the dot product tends to grow in magnitude. This is because you are summing up more terms.
关键见解：如果你增加更多的维度（给每个向量增加更多的数字），点积往往会在量级上增长。这是因为你正在累加更多的项。

Example - same vectors, but with extra dimensions added:
示例 - 相同的向量，但增加了额外的维度：

$a = [2, 3, 1, 2]$, $b = [4, 1, 3, 2]$:
$a = [2, 3, 1, 2]$, $b = [4, 1, 3, 2]$:

$$a \cdot b = (2 \times 4) + (3 \times 1) + (1 \times 3) + (2 \times 2) = 8 + 3 + 3 + 4 = 18$$

We went from 2 dimensions (dot product = 11) to 4 dimensions (dot product = 18). More dimensions → more terms → larger magnitude. For random vectors, the expected magnitude of the dot product grows proportionally to $\sqrt{d}$, where d is the number of dimensions.
我们从 2 维（点积 = 11）增加到了 4 维（点积 = 18）。更多的维度 → 更多的项 → 更大的量级。对于随机向量，点积的预期量级与 $\sqrt{d}$ 成正比增长，其中 d 是维数。

This is the foundation of the entire analysis - and it is exactly why Transformers divide by $\sqrt{d_k}$ in attention: to cancel out this dimension-dependent growth.
这是整个分析的基础——这也正是为什么 Transformer 在注意力机制中除以 $\sqrt{d_k}$ 的原因：为了抵消这种依赖于维度的增长。

2. What is Softmax?
2. 什么是 Softmax?

Softmax converts a list of raw numbers (called **"logits"**) into a **probability distribution** - a list of values between 0 and 1 that all add up to 1. You can think of the outputs as **percentages**: they tell you what fraction of "attention" goes to each item.
Softmax 将一组原始数字（称为**"logits"**）转换为**概率分布**——一组介于 0 和 1 之间的值，且总和为 1。你可以将输出视为**百分比**：它们告诉你分配给每个项目的"注意力"比例。

Example 1 - Simple logits:
示例 1 - 简单 logits：

Logits: $[1.0, 2.0, 3.0]$
$\rightarrow$ Softmax output (probabilities): $[0.09, 0.24, 0.67]$

Read this as: 9% goes to the first item, 24% to the second, and 67% to the third. They add up to 1.0 (100%).
解读为：9% 分配给第一个项目，24% 分配给第二个，67% 分配给第三个。它们的总和为 1.0 (100%)。

Example 2 - Similar logits:
示例 2 - 相似 logits：

Logits: $[5.0, 5.1, 4.9]$
$\rightarrow$ Softmax output: $[0.33, 0.34, 0.33]$

When the logits are almost the same, the probabilities are nearly equal - the model "can't decide" and spreads attention evenly.
当 logits 几乎相同时，概率也几乎相等——模型"无法决定"并均匀地分散注意力。

Example 3 - Very different logits:
示例 3 - 差异巨大的 logits：

Logits: $[10.0, 1.0, 1.0]$
$\rightarrow$ Softmax output: $[0.9998, 0.0001, 0.0001]$

When one logit is much bigger, softmax gives almost 100% of the probability to that item. The model is "laser-focused."
当一个 logit 大得多时，softmax 几乎将 100% 的概率全部分配给该项。模型处于"激光聚焦"状态。

The formula looks scary, but don't worry about it - just know that softmax converts a list of numbers into another list that adds up to 1 and can be used as probabilities:
公式看起来很吓人，但不要担心——只要知道 softmax 将一组数字转换为另一组总和为 1 且可用作概率的数字即可：

$$\text{Softmax}(z_i) = \frac{e^{z_i}}{\sum_j e^{z_j}}$$

Critical property: Softmax is sensitive to the *spread* (range) of its input values.
关键属性：Softmax 对其输入值的分布（范围）非常敏感。

- If the inputs are close together (e.g., $[2.1, 2.2, 1.9]$), the softmax output is nearly uniform: $[\sim 0.34, \sim 0.38, \sim 0.28]$.
  如果输入很接近（例如 $[2.1, 2.2, 1.9]$），则 softmax 输出几乎是均匀的：$[\sim 0.34, \sim 0.38, \sim$

0.28]。

- If the inputs are far apart (e.g., $[30, -10, 5]$), the softmax output concentrates on the largest: $[\sim 1.0, \sim 0.0, \sim 0.0]$.
  如果输入相差很大（例如 $[30, -10, 5]$），则 softmax 输出会集中在最大的那一个上：$[\sim 1.0, \sim 0.0, \sim 0.0]$。

This will be important later - **the attention's "key norm" in our experiment controls how spread-out those logits are**, which in turn controls how "sharp" or "diffuse" the attention becomes (if layer is distributing attention across more tokens, or focusing it on a few important tokens).
这在稍后会变得非常重要——**我们实验中的注意力"键范数"控制了这些 logits 的分散程度**，进而控制了注意力变得多么"锐利（集中）"或"弥散（分散）"（即该层是将注意力分配给更多的令牌，还是聚焦于少数重要的令牌）。

3. What is the L2 Norm?
3. 什么是 L2 范数?

The L2 Norm measures the total "length" or "magnitude" of a vector:
L2 范数衡量向量的总"长度"或"量级"：

$$\|x\|_2 = \sqrt{x_1^2 + x_2^2 + \ldots + x_d^2}$$

Example: A vector $[3, 4]$ has L2 norm $\sqrt{9 + 16} = 5$.
示例：向量 $[3, 4]$ 的 L2 范数为 $\sqrt{9 + 16} = 5$。

Scaling a vector changes its norm proportionally. If you multiply every component by the same factor, the norm multiplies by that same factor:
对向量进行缩放会按比例改变其范数。如果你将每个分量乘以相同的因子，范数也会乘以该相同的因子：

- Original vector: $[3, 4] \rightarrow$ Norm $= \sqrt{9 + 16} = 5$
  原始向量：$[3, 4] \rightarrow$ 范数 $= \sqrt{9 + 16} = 5$
- Doubled: $[6, 8] \rightarrow$ Norm $= \sqrt{36 + 64} = 10$ (exactly 2×5)
  翻倍：$[6, 8] \rightarrow$ 范数 $= \sqrt{36 + 64} = 10$（正好是 2×5）
- Halved: $[1.5, 2] \rightarrow$ Norm $= \sqrt{2.25 + 4} = 2.5$ (exactly 0.5×5)
  减半：$[1.5, 2] \rightarrow$ 范数 $= \sqrt{2.25 + 4} = 2.5$（正好是 0.5×5）

During our LLM pretraining experiment we will see that L2 Norm is growing in some layers and shrinking in others.
在我们的 LLM 预训练实验中，我们将看到 L2 范数在某些层中增长，在其他层中缩小。

L2 Norm Shows How Focused Attention Is
L2 范数显示了注意力的集中程度

This model applies `RMSNorm` to the Key vectors before attention. RMSNorm is a normalization technique that works in two steps:
该模型在注意力机制之前对键向量应用了 `RMSNorm`。RMSNorm 是一种分两步进行的归一化技术：

1. **Normalize:** Divide the vector by its root-mean-square, forcing the average squared component to be 1:
1. **归一化**：将向量除以其均方根，强制平均平方分量为 1：

$$\hat{x} = \frac{x}{\text{RMS}(x)}, \quad \text{where} \quad \text{RMS}(x) = \sqrt{\frac{1}{d}\sum x_i^2}$$

2. **Scale:** Multiply by a **learnable gain** vector γ: output $= \gamma \odot \hat{x}$.
2. **缩放**：乘以一个**可学习的增益**向量 γ：输出 $= \gamma \odot \hat{x}$。

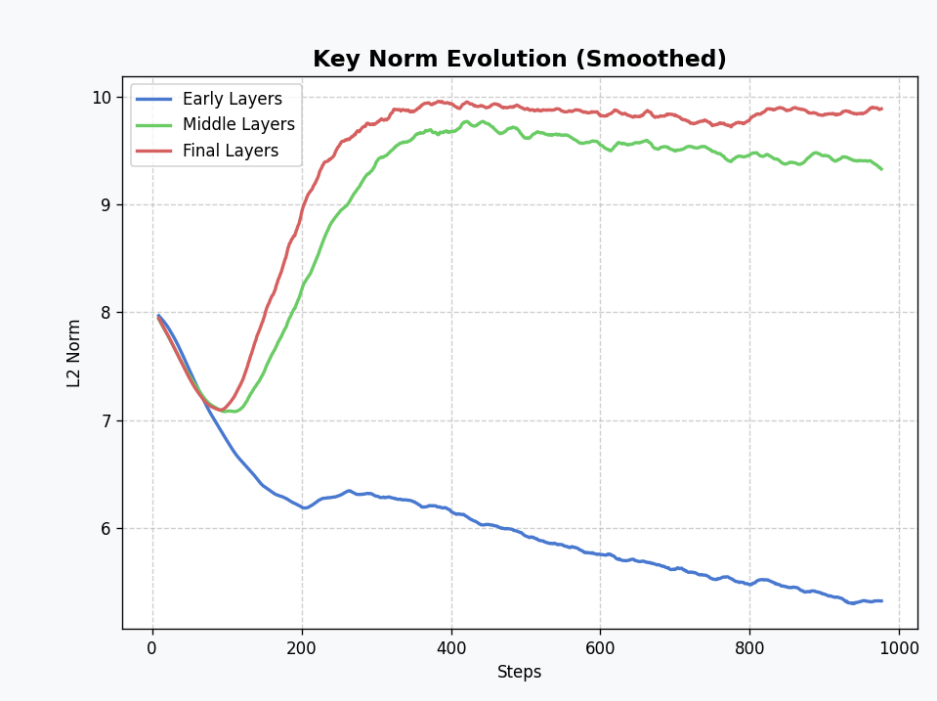
Think of it this way: step 1 "resets" the vector to a standard size. Step 2 lets the model learn *how big* it actually wants each component to be.
可以这样想：第一步将向量"重置"为标准大小。第二步让模型学习它实际上希望每个分量有多大。

At initialization, $\gamma = [1, 1, \ldots, 1]$ (all ones), so step 2 does nothing. After step 1 alone, the mean squared component is forced to be 1, so the expected L2 Norm of the output is:
在初始化时，$\gamma = [1, 1, \ldots, 1]$（全为 1），因此第二步什么都不做。仅经过第一步后，平均平方分量被强制为 1，因此输出的预期 L2 范数为：

$$\| \hat{x} \|_2 = \sqrt{\sum_{i=1}^{d} \hat{x}_i^2} = \sqrt{d \cdot \frac{1}{d}\sum \hat{x}_i^2} = \sqrt{d \cdot 1} = \sqrt{d}$$

With $d = d_{\text{head}} = 64$ (i.e., `d_model=512` divided by `n_heads=8`), the initial L2 Norm is $\sqrt{64} = 8.0$.
由于 $d = d_{\text{head}} = 64$（即 `d_model=512` 除以 `n_heads=8`），初始 L2 范数为 $\sqrt{64} = 8.0$。

During training, gradient descent updates the gain γ. If the model learns gains greater than 1, the output norm rises above 8.0. If it learns gains less than 1, the norm drops below 8.0. **This is the mechanism by which the model controls the Key norm - it adjusts**



RMSNorm's learnable gain parameters.
在训练期间，梯度下降会更新增益 γ。如果模型学习到大于 1 的增益，输出范数将上升到 8.0 以上。如果学习到小于 1 的增益，范数将降至 8.0 以下。**这就是模型控制键范数的机制——它调整 RMSNorm 的可学习增益参数。**

Mechanism: Sharpness Control
机制：锐利度控制

The attention mechanism computes **scaled dot-product attention**:
注意力机制计算**缩放点积注意力**：

$$\text{Attention}(Q, K, V) = \text{Softmax}\left(\frac{Q \cdot K^T}{\sqrt{d_k}}\right) V$$

The $\frac{1}{\sqrt{d_k}}$ factor normalizes the dot products so they don't grow with dimension (recall from the Prerequisites - dot products grow with $\sqrt{d}$, so dividing by $\sqrt{d_k}$ cancels that out).
$\frac{1}{\sqrt{d_k}}$ 因子对点积进行归一化，使它们不随维度增长（回想预备知识——点积随 $\sqrt{d}$ 增长，因此除以 $\sqrt{d_k}$ 抵消了这一点）。

But on top of this standard scaling, the *magnitude* of K still acts as an additional multiplier. If the Key norm is α times larger, all dot products in $Q \cdot K^T$ are also α times larger, which changes the softmax behavior (recall - softmax is sensitive to the spread of its

inputs).
但在这种标准缩放之上，K 的量级仍然充当额外的乘数。如果键范数大 α 倍，则 $Q \cdot K^T$ 中的所有点积也会大 α 倍，这改变了 softmax 的行为（回想一下——softmax 对输入的分布很敏感）。

- **Larger Key norm** → larger logits → sharper attention → lower effective temperature.
  更大的键范数 → 更大的 logits → 更锐利的注意力 → 更低的有效温度。
- **Smaller Key norm** → smaller logits → more diffuse attention → higher effective temperature.
  更小的键范数 → 更小的 logits → 更弥散的注意力 → 更高的有效温度。

Analyzing the image
分析图像

- **Case 1: Low Norm**
  情况 1：低范数
  - **Norm:** ∼5.3 (below the 8.0 baseline)
    范数： ∼5.3（低于 8.0 基准线）
  - **What happens:** All dot products $Q \cdot K^T$ are scaled down (by a factor of ∼5.3/8.0 ≈ 0.66 compared to baseline). The logits fed into softmax are closer together.
    发生了什么：所有的点积 $Q \cdot K^T$ 都缩小了（与基准线相比，缩小了约 ∼5.3/8.0 ≈ 0.66 倍）。输入到 softmax 的 logits 更加接近。
  - **Result:** A more diffuse attention distribution. The model attends to many tokens roughly equally. This is useful for **aggregating broad context** - for example, building up a representation of the overall sentence structure.
    结果：产生更向弥散的注意力分布。模型大致平等地关注许多令牌。这对于**聚合广泛的上下文**非常有用——例如，构建整体句子结构的表示。
- **Case 2: High Norm**
  情况 2：高范数
  - **Norm:** ∼9.8 (above the 8.0 baseline)
    范数： ∼9.8（高于 8.0 基准线）
  - **What happens:** All dot products are scaled up (by a factor of ∼9.8/8.0 ≈ 1.23 compared to baseline). The logits fed into softmax are pushed further apart.
    发生了什么：所有的点积都放大了（与基准线相比，放大了约 ∼9.8/8.0 ≈ 1.23 倍）。输入到 softmax 的 logits 被推得更开。

– **Result:** A more peaked attention distribution. The model concentrates attention on a small number of tokens (or even a single token). This is useful for **precise information retrieval** - for example, copying a specific fact or entity from earlier in the context.
结果：产生更尖锐的注意力分布。模型集中关注少数令牌（甚至单个令牌）。这对于**精确的信息检索**非常有用——例如，从之前的上下文中复制特定的事实或实体。

Note on the examples: The exact logit values depend on both the Query and Key vectors - specifically, how well they align. The Key norm only controls the *scale* of those logits, not their specific values. Two situations with the same Key norm but different Q-K alignments will produce different logit patterns.
关于示例的说明：确切的 logit 值取决于查询（Query）向量和键向量——具体来说，取决于它们的对齐程度。键范数只控制这些 logits 的*缩放*，而不是它们的具体值。具有相同键范数但 Q-K 对齐方式不同的两种情况将产生不同的 logit 模式。

Experimental Analysis
实验分析

- **Model:** 88M parameter Transformer (22 layers, `d_model=512`, 8 heads, `d_head=64`).
模型： 8800 万参数的 Transformer（22 层，`d_model=512`，8 个头，`d_head=64`）。
- **What's measured:** The L2 norm of Key vectors **after RMSNorm** (i.e., after the learnable gain is applied, but before Rotary Position Embeddings are added). This isolates the effect of the learned scaling.
测量内容: **RMSNorm 之后**（即应用了可学习增益后，但在添加旋转位置嵌入之前）键向量的 L2 范数。这隔离了学习缩放带来的影响。
- **Training:** 8M tokens (∼1000 gradient steps, `batch_size=4`, `seq_len=2048`).
训练： 800 万令牌（约 1000 个梯度步长，`batch_size=4`，`seq_len=2048`）。

Very few layers remain near the initialization value of 8.0 by the end of training. The model pushes most layers toward one of two operating regimes (dampened or amplified), suggesting that the "neutral" baseline is suboptimal and the model benefits from **distinct functional specialization** across depths.
训练结束时，很少有层保持在初始值 8.0 附近。模型将大多数层推向两种运行状态之一（变低或变高），这表明"中性"基准并非最优，模型能够从跨深度的**独特功能专业化**中获益。

The final layers have slightly higher norms than the middle layers (∼9.8 vs ∼9.3), suggesting the last layers need the sharpest attention for the final prediction - consistent with their role as the "decision-making" layers closest to the output.

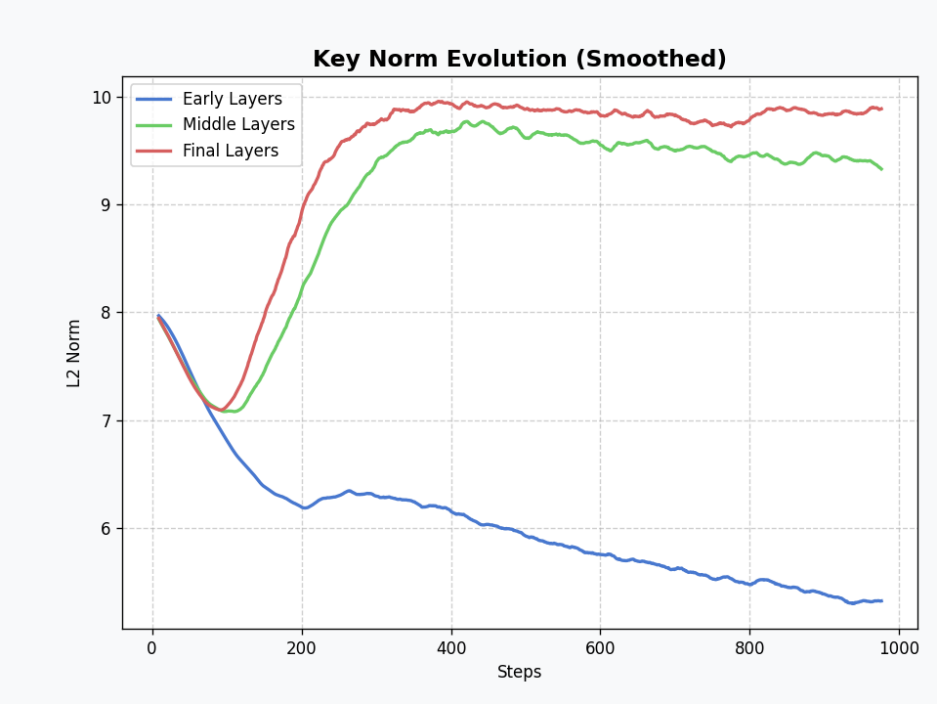


图 2: Figure 1: Evolution of the average Key Norm over ∼1000 training steps. ”Early Layers” = layers 0-1, ”Middle Layers” = layers 10-11, ”Final Layers” = layers 20-21.
图 1：∼1000 个训练步长内平均键范数的演变。“早期层”= 第 0-1 层，“中间层”= 第 10-11 层，“最终层”= 第 20-21 层。

最终层的范数略高于中间层（∼9.8 vs ∼9.3），这表明最后一层需要最锐锐利的注意力来进行最终预测——这与其作为最靠近输出的“决策层”的角色相一致。