

# 现代大模型综述：从 Transformer 到线性注意力

## 1. 引言

近年来，大语言模型（LLM）彻底改变了人工智能领域。从早期的 RNN 到如今的 Transformer，再到新兴的线性注意力架构，模型架构的演进始终围绕着两个核心矛盾：**训练效率（并行性）与推理效率（成本与长序列能力）**。本文将梳理这一演化历史，重点阐述 Transformer 为何成功，以及线性注意力（Linear Attention）如何通过三代演进，试图在保持 Transformer 性能的同时找回 RNN 的推理优势。

## 2. RNN：串行计算

在 Transformer 出现之前，RNN（循环神经网络）及其变体 LSTM/GRU 是处理序列数据的主流。然而，它们在处理大规模数据和长序列时遇到了难以克服的物理障碍。

### 2.1 无法并行训练 (Sequential Computation)

RNN 的核心公式定义了当前时刻的状态  $h_t$  严格依赖于上一时刻的状态  $h_{t-1}$ ：

$$h_t = \sigma(W_h h_{t-1} + W_x x_t + b)$$

其中  $\sigma$  是非线性激活函数（如 tanh 或 ReLU）。

- 时间依赖链**：为了计算第 100 个时间步的  $h_{100}$ ，必须先计算  $h_{99}$ ，而  $h_{99}$  又依赖  $h_{98}$  ... 直到  $h_0$ 。
- GPU 利用率低**：现代 GPU 擅长进行大规模矩阵并行运算（如一次性计算所有 Token 的  $Q \cdot K^T$ ），但 RNN 这种“接力跑”式的计算模式迫使 GPU 必须等待上一步完成才能进行下一步。这导致在长序列训练时，GPU 大部分计算单元处于闲置状态，训练速度极慢。

### 2.2 梯度消失与爆炸 (Gradient Vanishing & Exploding)

RNN 难以捕捉长距离依赖（Long-term Dependencies）的根本原因在于反向传播算法（BPTT）。

假设我们要计算时刻  $T$  的损失  $L_T$  对时刻  $0$  的输入  $x_0$  的梯度，根据链式法则：

$$\frac{\partial L_T}{\partial x_0} = \frac{\partial L_T}{\partial h_T} \cdot \frac{\partial h_T}{\partial h_{T-1}} \cdot \frac{\partial h_{T-1}}{\partial h_{T-2}} \cdot \dots \cdot \frac{\partial h_1}{\partial h_0} \cdot \frac{\partial h_0}{\partial x_0}$$

其中关键项是连乘部分：

$$\prod_{k=1}^T \frac{\partial h_k}{\partial h_{k-1}} = \prod_{k=1}^T W_h^T \cdot \text{diag}(\sigma'(z_k))$$

- 指数级衰减/增长**：如果权重矩阵  $W_h$  的特征值（Spectral Radius）小于 1，经过  $T$  次连乘后，梯度会趋近于 0（**梯度消失**）；如果大于 1，则会趋近于无穷大（**梯度爆炸**）。
- 例子**：考虑一个简单的句子：“**Alice** went to the kitchen ... [100 words] ... and **she** cooked dinner.”
  - 为了预测 “she”，模型需要利用 100 个词之前的 “Alice” 的信息。
  - 在 RNN 中，“Alice” 的信息需要经过 100 次矩阵乘法和非线性变换。如果每次变换保留 90% 的信息 (0.9)，那么 100 步后仅剩  $0.9^{100} \approx 0.000026$ 。

- 这意味着模型在更新参数时，几乎“感觉”不到 "Alice" 对 "she" 的影响，从而导致模型“遗忘”了主语。

虽然 LSTM 和 GRU 通过引入门控机制（Gating）缓解了这个问题，但并未从根本上解决串行计算的瓶颈。

### 3. Transformer：并行与注意力

2017 年，Google 提出的《Attention Is All You Need》改变了一切。它不仅是一个新架构，更是一种计算范式的转移。Transformer 的强大不仅仅在于并行训练，更在于其组件设计的精妙之处。

#### 3.1 输入层：词向量空间 (Token Embeddings)

在进入复杂的注意力机制之前，我们首先要解决的问题是：**如何让计算机理解单词？**

- **离散符号到连续向量**：计算机无法直接处理 "Apple" 或 "Banana" 这样的字符串。我们需要建立一个查找表（Embedding Table），将每个词映射为一个固定长度的实数向量（例如 4096 维）。
- **语义空间**：这个向量空间具有良好的几何性质。语义相近的词，在空间中的距离更近。
  - **经典例子**： $\text{Vector}(\text{King}) - \text{Vector}(\text{Man}) + \text{Vector}(\text{Woman}) \approx \text{Vector}(\text{Queen})$ 。
  - 这意味着模型在输入层就已经具备了初步的推理能力。

#### 3.2 训练范式：自回归生成 (Autoregression)

现代 LLM（如 GPT 系列）通常采用\*\*自回归（Autoregressive）\*\*的方式进行训练。

- **目标**：预测下一个 Token。即计算概率  $P(w_t | w_1, w_2, \dots, w_{t-1})$ 。
- **过程**：模型读入当前的上下文，预测下一个最可能出现的词。然后将这个词加入上下文，继续预测下一个词。这就像人类写文章一样，一个字一个字地往后写。

#### 3.3 Self-Attention：信息的“路由”与“聚合”

Transformer 的核心在于自注意力机制（Self-Attention）。对于初学者来说，直接看矩阵公式  $\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d}})V$  可能比较抽象。让我们把它拆解开来，看看对于序列中的某一个词，到底发生了什么。

##### 3.3.1 核心直觉：回顾历史 (Looking Back)

在自回归推理中，模型只能看到**当前和过去**的信息。想象模型正在生成句子：“**Alice likes ...**”。当前输入是 "likes"，模型需要预测下一个词（比如 "Bob"）。为了做出准确预测，模型必须回顾历史，搞清楚“谁”在喜欢。

Self-Attention 的本质就是：**用当前词（Query）去查询历史记忆中所有的词（Key），根据匹配程度（Attention Score）将它们的内容（Value）加权融合过来。**

**Q 与 K 的相似度理论**：为什么我们要计算  $Q \cdot K^T$ ？在几何上，两个向量的点积（Dot Product）衡量了它们的**相似度或对齐程度**。

- 如果  $Q$  和  $K$  指向相同的方向，点积最大（关注度最高）。
- 如果  $Q$  和  $K$  垂直（无关），点积为 0（不关注）。
- 这就像数据库查询： $Q$  是你的搜索关键词， $K$  是数据库中每条记录的索引标签。点积越高，说明这条记录越符合你的搜索意图。

### 3.3.2 序列形式公式 (The Sequence Form)

对于序列中的第  $t$  个 Token，其输出向量  $y_t$  的计算公式如下（注意求和上限是  $t$ ，不能看未来）：

$$y_t = \sum_{i=1}^t \underbrace{\text{softmax}\left(\frac{q_t \cdot k_i^T}{\sqrt{d}}\right)}_{\text{注意力权重}} \cdot v_i$$

- $q_t$ ：当前词  $t$  的查询向量 (Query)。
- $k_i$ ：历史中第  $i$  个词的键向量 (Key)， $i \leq t$ 。
- $v_i$ ：历史中第  $i$  个词的值向量 (Value)。
- $\alpha_{t,i}$ ：第  $t$  个词对第  $i$  个词的关注度。

### 3.3.3 图解与例子：一次真实的推理步骤

假设模型已经处理了 "Alice"，现在轮到 "likes"。当前时刻： $t=2$  输入：likes 历史上下文：[Alice, likes] (注意：Bob 还没出现，是我们要预测的目标)

我们来看看如何计算 "likes" 的输出向量，以便预测下一个词。

**第一步：打分 (Matching)** "likes" 发出查询  $q_{\text{likes}}$ ，回顾历史（包括自己）：

$$\begin{aligned} q_{\text{likes}} \cdot k_{\text{Alice}} &= 0.9 && \text{（很高，因为需要找到主语是谁）} \\ q_{\text{likes}} \cdot k_{\text{likes}} &= 0.5 && \text{（关注自己，提取动词本身的含义）} \end{aligned}$$

(注：此时模型完全不知道 Bob 的存在)

**第二步：归一化 (Softmax)** 将分数转化为概率：

$$\text{Softmax}([0.9, 0.5]) \approx [0.6, 0.4]$$

↑

Alice

↑

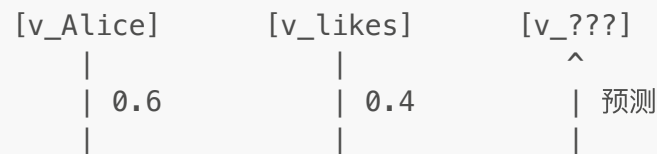
likes

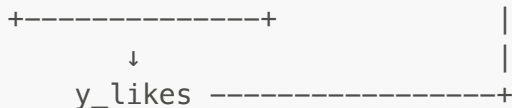
**第三步：聚合 (Aggregation)** 根据权重，融合历史信息：

$$y_{\text{likes}} = 0.6 * v_{\text{Alice}} + 0.4 * v_{\text{likes}}$$

**结果：**得到的  $y_{\text{likes}}$  向量融合了 "Alice" (主语) 和 "likes" (谓语) 的信息。这个向量经过后续层处理，最终会预测出高概率的词："Bob" 或 "Apples"。

可视化流程：





### 3.3.4 多头注意力 (Multi-Head Attention)

- 如果只有一个头, "likes" 可能只能关注到语法成分。但它还需要关注情感色彩等。
- 多头机制允许模型在不同的子空间里并行关注不同的特征。Head 1 关注指代, Head 2 关注句法依存, Head 3 关注上下文情感。

## 3.4 MLP (Feed-Forward Networks): 知识的“存储”与“加工”

在 Attention 层之后, 总是紧跟着一个 MLP (多层感知机) 层。为什么?

- **结构:** 通常是 **Linear**  $\rightarrow$  **Activation (GeLU/SwiGLU)**  $\rightarrow$  **Linear**。它独立地作用于每个 Token, 不涉及 Token 间的交互。
- **分工:**
  - **Attention** 负责 **Token 之间** 的信息流动 (Mixing information between tokens)。它把上下文信息搬运到当前 Token。
  - **MLP** 负责 **Token 内部** 的信息加工 (Mixing information within a token)。
- **深度理解: 键值记忆网络 (Key-Value Memories):**
  - 有研究 (Geva et al.) 认为, MLP 层充当了模型的**知识库**。
  - 第一层 Linear 类似于检测某种模式 (Key), 激活函数筛选模式, 第二层 Linear 输出该模式对应的属性或结果 (Value)。
  - 例如, Attention 把 "法国" 和 "首都" 搬运到了一起, MLP 层检测到这个组合, 然后输出 "巴黎"。

## 3.5 位置编码: 赋予序列秩序

Self-Attention 本质上是集合运算 (置换不变性), 无法区分 "A hit B" 和 "B hit A"。必须显式注入位置信息。

- **绝对位置编码 (Sinusoidal / Learnable):**
  - 直接将位置向量  $P_i$  加到输入  $X_i$  上。
  - **缺点:** 外推性差。训练时只见过长度 1024, 推理时遇到 2048 就不知道  $P_{\{2048\}}$  是什么, 或者正弦波从未见过的相位会导致混乱。
- **相对位置编码 (ALiBi):**
  - 不加在输入上, 而是直接加在 Attention Score 上。距离越远, 扣分越多 ( $QK^T - m \cdot |i-j|$ )。
  - **优点:** 外推性极强, 训练短序列, 推理长序列效果好。
- **旋转位置编码 (RoPE - Rotary Positional Embedding):**
  - **当前主流** (LLaMA, Qwen 等使用)。
  - 通过旋转向量的角度来编码相对位置。 $q_i$  旋转  $i \cdot \theta$ ,  $k_j$  旋转  $j \cdot \theta$ , 它们的点积只与相对距离  $(i-j) \cdot \theta$  有关。
  - **优点:** 完美结合了绝对位置的实现便利性和相对位置的数学性质, 外推性较好。

## 3.6 残差连接与层归一化: 深度的基石 (Residuals & LayerNorm)

为什么 Transformer 可以堆叠到上百层而不会梯度消失? 这归功于 **残差连接 (Residual Connection)**。

### 3.6.1 残差连接：缓解梯度消失

在深层网络中，如果每一层都是  $x_{l+1} = F(x_l)$ ，那么反向传播时的梯度是连乘的： $\frac{\partial L}{\partial x_0} = \frac{\partial L}{\partial x_L} \cdot \prod \frac{\partial F}{\partial x}$ 。一旦某层的导数小于 1，连乘几十次后梯度就会趋近于 0，导致**梯度消失**。

Transformer 采用了残差结构： $x_{l+1} = x_l + F(x_l)$  其中  $F(x_l)$  是 Attention 或 MLP 层的计算结果。

**数学原理：** 根据链式法则，反向传播时的梯度计算如下：

$$\frac{\partial x_{l+1}}{\partial x_l} = 1 + \frac{\partial F(x_l)}{\partial x_l}$$

最终的梯度流向是： $\frac{\partial L}{\partial x_l} = \frac{\partial L}{\partial x_{l+1}} \cdot \left( 1 + \frac{\partial F}{\partial x_l} \right)$

- **恒等映射 (Identity Mapping)：** 公式中的常数项 **\$1\$** 保证了梯度可以直接传回前一层。
- **梯度保持：** 即使某层的非线性变换部分  $\frac{\partial F}{\partial x_l}$  梯度很小，梯度信号依然可以通过 **\$1\$** 这一项继续向前传播。这有效缓解了梯度消失问题，使得训练深层网络成为可能。

### 3.6.2 层归一化 (LayerNorm / RMSNorm)

$$x_{l+1} = \text{LayerNorm}(x_l + F(x_l))$$

- **作用：** 将每一层的输出归一化到均值为 0，方差为 1。
- **意义：** 这防止了数值在深层网络中剧烈波动（梯度爆炸），进一步稳定了训练过程。

## 3.7 表达能力与深度：电路的升级

为什么 Transformer 需要堆叠几十层？

- **残差流 (Residual Stream)：**
  - Transformer 的主干是一个贯穿始终的向量流  $x + \text{Sublayer}(x)$ 。
  - 每一层（Attention 或 MLP）都是从这个流中“读”出信息，处理后，再“写”回一个增量（Residual Update）。
- **归纳头 (Induction Heads) 与多步推理：**
  - **第 1 层：** 可能只是简单的词法关注。
  - **第 2 层：** 可以利用第 1 层搬运来的信息。
  - **Induction Head** 是 In-context Learning 的核心电路。它由两层 Attention 组成：
    - Layer 1 关注当前 Token 的上一个 Token（复制历史）。
    - Layer 2 搜索历史中出现过类似“上一个 Token”的地方，并把那之后的 Token 搬运过来。
    - **功能：** 实现了“如果 A 后面通常跟 B，那么这次看到 A，我也预测 B”的模式复制能力。
  - 随着层数加深，模型能组合出极其复杂的逻辑电路，实现推理能力。

## 3.8 自回归推理：从向量回归文本

当模型经过几十层的计算，输出了最后一个 Token 的向量  $h_{\text{last}}$  后，我们如何得到文本？

### 1. Unembedding (反嵌入)：

- 将  $h_{\text{last}}$  乘以 Embedding 矩阵的转置（或单独的输出头），得到一个长度为词表大小（如 50,000）的向量 **Logits**。

- Logits 中的每个数值代表对应词的“得分”。

## 2. Softmax:

- 将 Logits 转化为概率分布  $P$ 。
- $P(\text{"apple"}) = 0.1, P(\text{"banana"}) = 0.05, \dots$

## 3. 采样 (Sampling):

- **Greedy**: 直接选概率最大的词。
- **Top-K / Top-P (Nucleus)**: 从概率最高的 K 个词或累积概率为 P 的集合中随机抽取。这增加了生成的多样性。

## 4. 循环 (Loop):

- 选出的词被追加到输入序列末尾，再次输入模型，预测下一个词。这就是自回归过程。

## 3.8 计算复杂度瓶颈

尽管 Transformer 极其强大，但它付出了昂贵的代价：**二次方复杂度**。

- **计算量与显存**: 计算 Attention 分数矩阵  $A = QK^T$  需要生成一个  $N \times N$  的矩阵。
- **直观感受**:
  - 序列长度  $N=1,000$  时,  $N^2 = 1,000,000$ 。
  - 序列长度  $N=100,000$  时,  $N^2 = 10,000,000,000$ 。
- 这导致标准 Transformer 难以处理长文档、长对话或基因序列等超长上下文任务。显存占用随长度爆炸式增长，推理时的 KV Cache 也消耗巨大。

## 4. 线性注意力的演进：回归 $O(N)$ 的尝试

为了解决  $O(N^2)$  问题，研究者试图将 Attention 线性化，使其推理复杂度降回  $O(N)$ （类似 RNN），同时保留并行训练能力。这一过程经历了三代演进。

### 4.1 第一代：核方法 (Kernel-based) 与“词袋”困境

为了突破计算瓶颈，研究者们回顾了矩阵乘法的结合律。

#### 4.1.1 线性化原理：结合律的应用

标准 Attention 的计算顺序是先算  $QK^T$ （得到  $N \times N$  矩阵），再乘  $V$ ： $(QK^T)V$

如果我们能去掉 Softmax，或者用某种核函数  $\phi(\cdot)$  近似 Softmax，使得  $\text{Sim}(Q, K) = \phi(Q)\phi(K)^T$ ，那么公式就可以重写为： $\phi(Q)(\phi(K)^T V)$

- **计算优势**:
  - 先计算  $\phi(K)^T V$ ，得到一个  $D \times D$  的矩阵（ $D$  是特征维度，通常很小，如 64 或 128）。这一步复杂度是  $O(N \cdot D^2)$ 。
  - 再用  $\phi(Q)$  去乘这个  $D \times D$  矩阵。
  - 总复杂度从  $O(N^2)$  降到了  $O(N)$ 。这意味着推理速度和显存占用不再随序列长度爆炸。

### 4.1.2 递归形式 (Recurrent View)

这种线性 Attention 可以被写成类似 RNN 的递归形式。定义记忆状态  $S_t = \sum_{i=1}^t \phi(k_i) v_i^T$ ：  
 $S_t = S_{t-1} + \phi(k_t) v_t^T$   
 $y_t = \phi(q_t) S_t$

**键值记忆：** 我们可以将矩阵  $S_t$  理解为一个**联想记忆库**。

- **写入 (Write)：**  $S_t = S_{t-1} + k_t v_t^T$ 。这本质上是 Hebbian Learning（赫布学习规则）的一种形式。我们将键值对  $(k_t, v_t)$  存储到矩阵中。
- **读取 (Read)：**  $y_t = S_t q_t$ 。当我们用查询向量  $q_t$  去乘矩阵时，实际上是在进行检索：  
 $(\sum k_i v_i^T) q_t = \sum v_i (k_i^T q_t)$   
 结果是所有存储的  $v_i$  的加权和，权重正是查询  $q_t$  与键  $k_i$  的相似度。

### 4.1.3 致命缺陷：“词袋”效应 (Bag of Words)

虽然解决了速度问题，但第一代线性 Attention（如 Linear Transformer, Performer）效果通常不如标准 Transformer。

- **记忆是简单的累加：** 观察公式  $S_t = \sum \phi(k_i) v_i^T$ ，这本质上是一个求和操作。
- **缺乏遗忘机制：** 模型被迫“记住”所有历史信息，无法根据当前上下文丢弃无关信息。随着序列变长， $S_t$  中积累的噪声越来越多，有效信号被淹没。
- **位置信息模糊：** 虽然可以加位置编码，但由于记忆是加性的，模型很难区分“Alice hit Bob”和“Bob hit Alice”在深层语义上的区别（如果位置编码被淹没）。这被称为**词袋问题**——模型知道出现了哪些词，但搞不清它们的精确顺序和结构。

## 4.2 第二代：门控与选择机制 (Gated Decay / Selection)

为了解决“词袋”问题，研究者们意识到：**记忆不能只是简单的累加，必须有选择地遗忘。**

### 4.2.1 核心创新：数据依赖的衰减 (Data-Dependent Decay)

在第一代模型中，如果引入衰减，通常是固定的（如  $S_t = \lambda S_{t-1} + \dots$ ）。而在第二代模型（Mamba, RWKV v5/v6, GLA）中，衰减率  $\alpha_t$  是由当前输入  $x_t$  动态计算的。

$$S_t = \alpha_t \odot S_{t-1} + \phi(k_t) v_t^T$$

其中  $\alpha_t \in [0, 1]$  是一个门控向量。

### 4.2.2 选择机制 (Selection Mechanism)

Gu & Dao 在 Mamba 论文中将这种机制称为**选择机制**。它赋予了模型以下能力：

1. **过滤噪声：** 遇到无意义的词（如停用词、无关插曲）时，将  $\alpha_t$  设为接近 1（保持记忆），同时抑制新信息的写入。
2. **重置上下文：** 当文章从“体育新闻”转折到“财经报道”时，模型可以将  $\alpha_t$  设为接近 0，快速遗忘旧的体育上下文，为新信息腾出空间。

### 4.2.3 例子：抗噪任务

考虑任务：[关键信息 A] ... [大量噪声文本] ... [提问 A?]

- **第一代模型**：在处理噪声时，噪声不断累加到  $S_t$  中，最终掩盖了 **关键信息 A**。
- **第二代模型**：模型学会了在处理噪声时让“写入门”关闭，或者让“遗忘门”保持开启，从而让  $S_t$  几乎不受噪声干扰，成功将 **关键信息 A** 传递到最后。

这一改进使得线性 Attention 模型首次在困惑度 (Perplexity) 和下游任务上匹敌甚至超越了同等规模的 Transformer。

### 4.3 第三代：Delta 规则与精确状态追踪 (Delta Rule / State Tracking)

尽管第二代模型表现优异，但它们在处理**联想回忆 (Associative Recall)** 和 **精确状态更新** 任务时仍有短板。这催生了以 DeltaNet 和 RWKV v7 为代表的第三代线性注意力。

#### 4.3.1 最后的短板：变量赋值问题

想象一个代码执行任务：

1.  $x = 5$
2. ... (执行其他操作)
3.  $x = 10$
4. `print(x)`

- **第二代模型**：当看到  $x = 10$  时，它会试图记住  $x: 10$ 。但由于它只能“衰减”旧信息， $x: 5$  的残余可能仍然存在。最终记忆可能是  $x: 7.5$  (混合态)。
- **理想行为**：在写入  $x = 10$  之前，应该先**精确擦除掉**  $x$  之前的值，然后再写入新值。

#### 4.3.2 Delta Rule：从优化视角看推导

第三代模型引入了**减性更新 (Subtractive Update)**，其数学本质是在**线梯度下降 (Online Gradient Descent)**。

**推导来源**：我们可以把记忆矩阵  $SS$  看作是一个试图拟合数据的线性模型。在时刻  $t$ ，我们的目标是让记忆  $SS$  能够根据键  $K_t$  正确检索到值  $V_t$  (即让  $SS$  记住这一对 KV 映射)。即希望： $S \cdot K_t \approx V_t$  (注：这是**写入/优化目标**，而模型实际输出时的**读取操作**依然是  $O_t = S \cdot Q_t$ )

为了达到这个目标，我们观察上一时刻的记忆  $S_{t-1}$  在当前数据上的表现，并计算其误差 (Loss)： $\mathcal{L} = \frac{1}{2} | S_{t-1} K_t - V_t |^2$

然后，我们根据这个误差对  $S_{t-1}$  进行修正 (梯度下降)： $\nabla_S \mathcal{L} = (S_{t-1} K_t - V_t) K_t^T$  (其中  $\nabla_S \mathcal{L}$  代表**梯度**，即“误差  $\mathcal{L}$  对记忆矩阵  $SS$  的导数”，它指明了  $SS$  应该如何变化才能最快地增大误差。为了**减小误差**，我们要反其道而行之，减去梯度。)

$$S_t = S_{t-1} - \eta \cdot \nabla_S \mathcal{L}$$

代入梯度公式，并设学习率为  $\beta_t$ ，我们得到 Delta Rule 的核心公式： $S_t = S_{t-1} + \underbrace{\beta_t (V_t - S_{t-1} K_t)}_{\text{Delta (误差)}} \otimes K_t$

**物理意义：从“叠加”到“重写”**

我们可以用**黑板写字**来类比这三代演进：

1. **第一代 (叠加)**：你在黑板上写字，写满了也不擦，直接在旧字上面叠着写新字。最后黑板上一团漆黑，什么也认不出来 (词袋效应)。



2. **第二代（衰减）**：黑板上的字会随时间自动变淡。你在写新字之前，旧字已经淡了一些。但这只是被动的遗忘，你无法主动擦除某个特定的错误。
3. **第三代（Delta / 重写）**：你手里拿了一个黑板擦。
  - **Predict**: 你先看一眼黑板上  $K_t$  这个位置现在写着什么 ( $S_{t-1}K_t$ )。
  - **Error**: 你对比一下你想写的内容  $V_t$  和黑板上现有的内容。
  - **Correct**: 你算出差值，**精准地擦掉**旧的痕迹，然后填上新内容。
  - 这实现了真正的**变量重写**（Variable Overwriting），就像在编程语言中执行  $x = 10$  一样，旧的  $x=5$  被彻底清除了。

这与卡尔曼滤波（Kalman Filter）中的“预测-校正”步骤有着异曲同工之妙。

如果展开公式，我们会发现它包含了一个显式的**减法项**：
$$S_t = S_{t-1} - \beta_t (S_{t-1} K_t) \otimes K_t + \beta_t V_t \otimes K_t$$

这意味着模型学会了在写入新信息之前，先将记忆矩阵在  $K_t$  方向上的投影减去。这实现了**正交化**，保证了新旧信息不会混淆。

#### 4.3.3 RWKV v7 的实现

在 RWKV v7 的架构中，这一思想得到了工程化的落地。在 `src/model/rwkv_v7.rs` 或 `time_mix.rs` 中，我们可以观察到状态更新的核心逻辑包含类似以下的结构：

```
// 伪代码示意
let recall = state * k;      // 回忆旧值
let error = v - recall;      // 计算差异
state = state + error * k;    // 更新状态
```

（注：实际实现中包含复杂的 Head 维度处理和 LoRA 门控，但核心数学原理一致）。

这种机制使得 RWKV v7 不仅拥有 RNN 的推理速度，还具备了类似 Transformer 的精确“In-context Learning”能力，能够处理复杂的逻辑推理和状态追踪任务。

#### 4.4 线性与非线性的辩证：为什么现代 RNN 是“线性”的？

在结束这一章之前，我们需要澄清一个常见的误区。当我们说 RWKV 或 Mamba 是“线性注意力”或“线性 RNN”时，指的是**状态更新方程关于状态  $S$  是线性的**。

##### 4.4.1 为什么要线性递归？(Why Linear Recurrence?)

传统 RNN（如 LSTM）的状态更新包含非线性激活函数：
$$h_t = \tanh(W h_{t-1} + \dots)$$
 这种非线性使得  $h_t$  无法直接展开为  $h_0$  的闭式解，必须一步步串行计算。

而现代线性 Attention 的更新方程（如 RWKV）：
$$S_t = A_t S_{t-1} + V_t K_t^T$$
 这里没有  $\tanh$  或  $\text{sigmoid}$  包裹  $S_{t-1}$ 。正是这种**线性递推关系**，使得我们可以利用**结合律**进行并行训练（Parallel Scan / Chunkwise Computation）。
$$S_2 = A_2 (A_1 S_0 + V_1 K_1^T) + V_2 K_2^T$$
 我们可以先算出所有  $A$  的累乘，再并行计算结果。

##### 4.4.2 为什么不需要非线性递归？(Why is Linear Sufficient?)

既然去掉了递归中的非线性，模型会不会变“傻”？毕竟深度学习的核心就是非线性变换。

答案是：不会，因为我们有层数（Depth）。

- 分工明确：
  - **时间轴 (Time Mixing)**：线性递归负责在时间维度上搬运信息。它像一条传送带，高效地把过去的记忆  $S_{t-1}$  传给现在。
  - **特征轴 (Channel Mixing)**：每个 Block 中的 **MLP 层**（以及 Attention 中的 Output Projection）包含大量的非线性激活函数（GeLU, Swish）。
- 堆叠效应：
  - 虽然单层的递归是线性的，但当我们把“线性递归”和“非线性 MLP”交替堆叠几十层后，整个模型对输入的变换就是高度非线性的。
  - 这就像 Transformer：Self-Attention 内部也是线性的（加权求和），非线性主要来自 MLP 和多层堆叠。

**结论**：现代 RNN 放弃了“时间步上的非线性”以换取并行训练能力，并通过“深度的非线性”来保证模型的表达能力。这是一个极其划算的交易。

## 5. 总结

从 RNN 到 Transformer，我们获得了并行性；从 Transformer 到现代线性注意力（RWKV v7, Mamba），我们在保持并行性的同时，找回了  $O(1)$  的推理效率，并正在通过 Delta Rule 攻克线性模型最后的短板——精确状态追踪。

## 6. 待办事项 (TODO)

- **并行训练与矩阵形式**：解释 Transformer 如何在训练阶段利用矩阵运算一次性处理整个序列（Teacher Forcing）。
- **因果掩码 (Causal Mask)**：详细解释如何在并行训练中防止模型“偷看”未来。