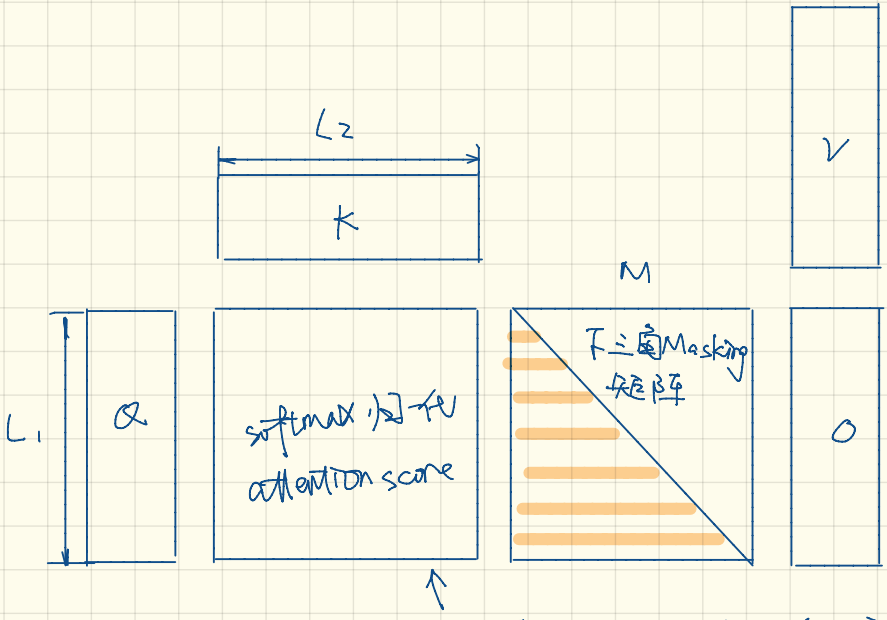
训练时 attention 以全并行方式进行计算，也就是矩阵化公式

$$Q\ K\ V = xW_Q,\ xW_K,\ xW_V$$

$$O = (\mathrm{softmax}(QK^T) \odot M)\ V$$



中间结果的时间和空间复杂度是序列长度的二次方

并行但 I/O 复杂度高

预测时以recurrent-form计算

$$q_t, k_t, v_t = x_t W_Q \quad x_t W_K \quad x_t W_V$$

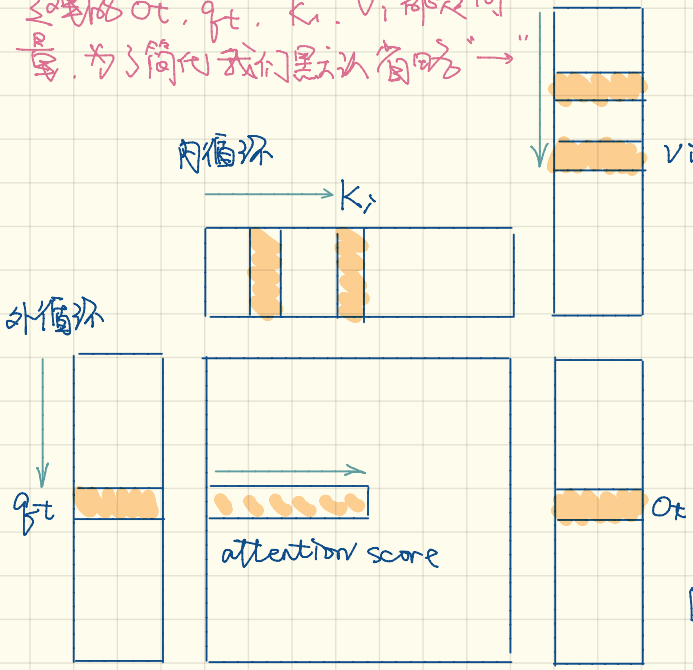
$$O_t = \frac{\sum_{i=1}^{t} \exp(q_t \cdot k_i^T) v_i}{\sum_{i=1}^{t} \exp(q_i \cdot k_i^T)}$$

将这里的softmax核看作是相似性计算

随着外循环长度的增加 $\{\vec{k}_i\}\{\vec{v}_i\}$（都是向量）的增加在增加，

k-v cache的问题

这里的 $\vec{o}_t, \vec{q}_t, \vec{k}_i, \vec{v}_i$ 都是向量，为了简化我们默认省略"→"



$(QK^T)V$

$[L, d][d \times L][L, d]$

$L \gg d$

$[L \times L][L, d]$

中间的结果

$[L, d]$

如果可以先计算

$Q(K^T V)$

$[L, d][d, L][L, d]$

$[L, d]$ $[d, d]$

中间的结果

$[L, d]$

影响先计算 $K^T V$ 的是softmax于是引入了结合律，linear attention寻求对softmax的改造，使得可以

全并行和串行RNN式计算（这里的串行可以以较少的并行比如parallel scan进行并行化增强）

$$o_t = \frac{\sum_{i=1}^{t} \exp(q_t \cdot k_i^T) v_i}{\sum_{i=1}^{t} \exp(q_t k_i^T)}$$

内积形式

用 kernel 替代 $\exp(q \cdot k)$

$\exp(\vec{q} \cdot \vec{k}) = \langle \phi(\vec{x}), \phi(\vec{y}) \rangle$，让 o_t 的计算被简化。

$$o_t = \frac{\sum_{i=1}^{t} \phi(q_t)\phi(k_i)^T v_i}{\sum_{i=1}^{t} \phi(q_t)\phi(k_i)^T}$$

$$= \frac{\phi(q_t) \boxed{\sum_{i=1}^{t} \phi(k_i) v_i}}{\boxed{\sum_{i=1}^{t} \phi(k_i)}} \quad \triangleq s_t \quad \triangleq z_t$$

$$\begin{cases} o_t = \dfrac{\phi(q_t) s_t}{\boxed{\phi(q_t) z_t}} \\ s_t = s_{t-1} + \phi(k_t)^T v_t \\ \boxed{z_t = z_{t-1} + \phi(k_t)} \end{cases}$$

这里在 attention score 的计算中最一个归一化项，一些研究验证了可以使用线性 kernel（例如将 ϕ 设置为 identity 函数）使用未归一化的 score，于是这个带了 × 版本的 attention 可以被简化

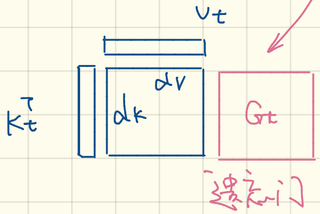
这是 linear attention 的 recurrent-form 的基本框架。SSM/RWKV 这样的新型 RNN 从另一个理论视角得到这样形式的形式

$$\begin{cases} o_t = \phi(q_t) s_t \\ s_t = s_{t-1} + \phi(k_t)^T v_t \end{cases}$$

$S_t = S_{t-1} + K_t^T V_t$, $O_t = q_t S_t$

linear attention在学习效果建模上有一个缺点，类似于LSTM中遗忘门的衰减机制对学习效果帮助很大，这里没有decay

← 这里是一个外积操作，linear attention是一个隐状态是2D的并行RNN



$K_t^T V_t = (X_t W_k)^T (W_v X_t)$

可学习参数

$O = ((QK^T) \odot M) V$

但由于M的存在破坏了矩阵乘法的结合性，O的计算还是需要从左到右，因此依然是L^2空间复杂度

Gated Linear Unit

RetNet引入静态衰减

Mamba / Mamba-2 data-dependent的decay 并行难度高

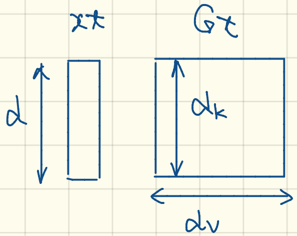
GLA的基本形式：$S_t = G_t \odot S_{t-1} + K_t^T V_t$ 这个公式可以统一许多模型的计算过程

为隐状态引入遗忘门，随时间而变化

$G_t = [d_t \times d_v] \in 0 \cdot 1$

在算法设计上要着重考虑如何得到Gt

$X_t \longrightarrow G_t$ 求解一个参数化的映射



如果通过某种映射矩阵得到 G_t，会需要 $d * d_k * d_v$ 的参数量，这是低效的

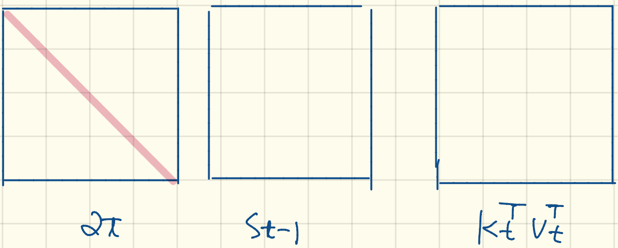
在 Mamba 中，使用向量 α 与矩阵 A 之间的运算得到 G_t，缺点是没法使用 tensor core

$$G_t = \alpha_t^T 1$$

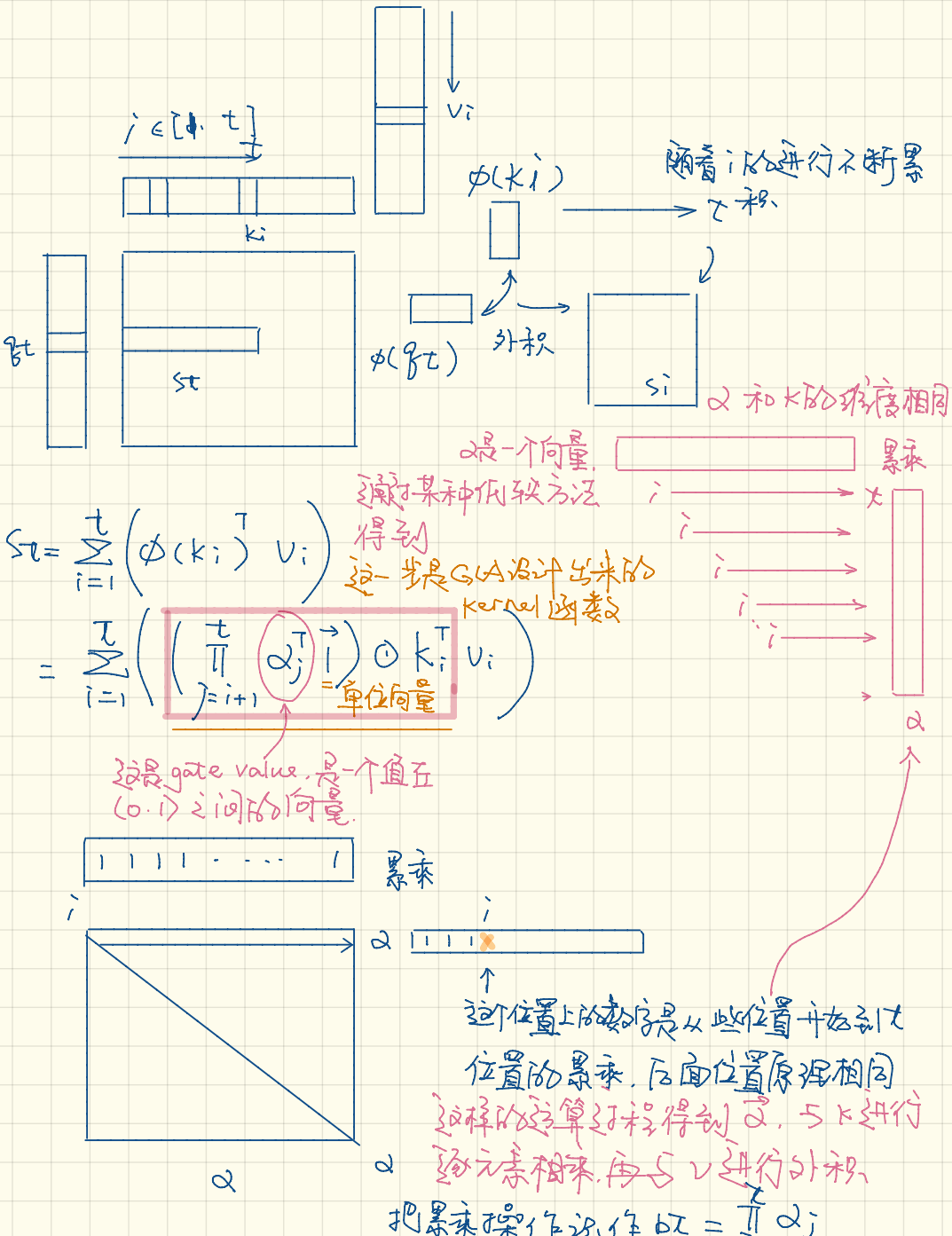
$$S_t = (\alpha_t^T 1) \odot S_{t-1} + k_t^T v_t$$

$$= \mathrm{Diag}(\alpha_t) S_{t-1} + k_t^T v_t$$

这个公式的通用性很好并且存在硬件友好的并行实现



通过 x_t 之上的低秩线性层 + sigmoid 得到 α_t

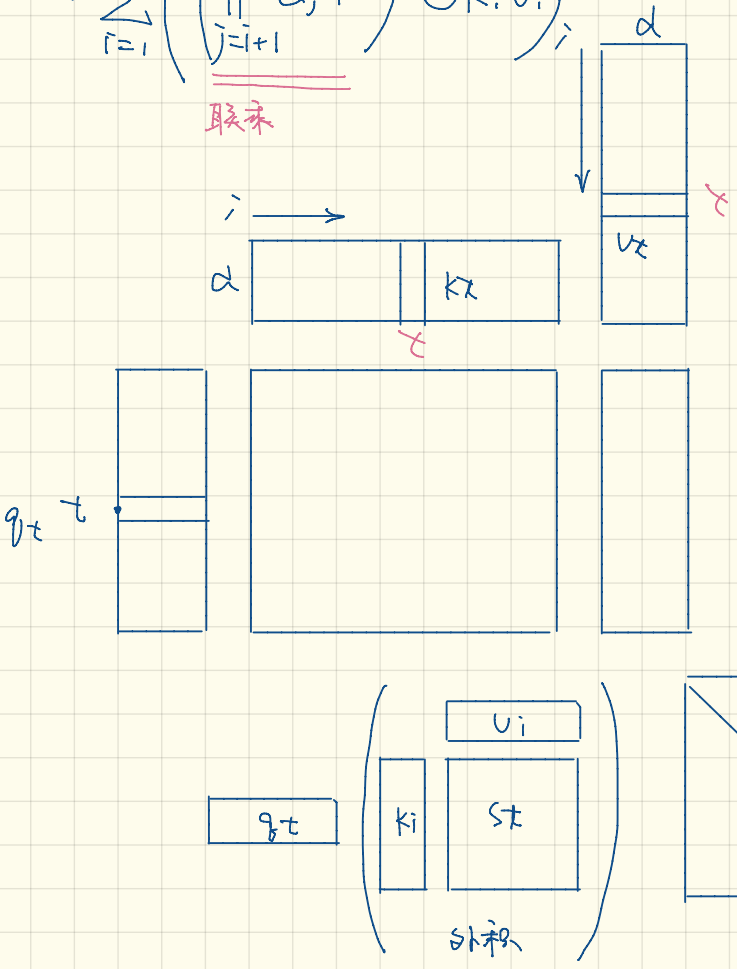

v_i
$i \in [1, t]$
k_i
$\phi(k_i)$
随着 i 的进行不断累乘
$\phi(q_t)$
外积
q_t
s_t
s_i
q 和 k 的形状相同
是一个向量
累乘
通过某种优秀方法得到
这一步是GLA设计出来的kernel函数
$S_t = \sum_{i=1}^{t} \left(\phi(k_i)^T v_i \right)$
$= \sum_{i=1}^{t} \left(\left(\prod_{j=i+1}^{t} \alpha_j^T \vec{1} \right) \odot k_i^T v_i \right)$
单位向量
这是 gate value，是一个值在 (0,1) 之间的向量
累乘
这个位置上的数字是从此位置开始到 t 位置的累乘，后面位置原理相同
这样的运算过程得到 α，与 k 进行逐元素相乘，再与 v 进行外积
把累乘操作运作成 $= \prod_{j=1}^{t} \alpha_j$

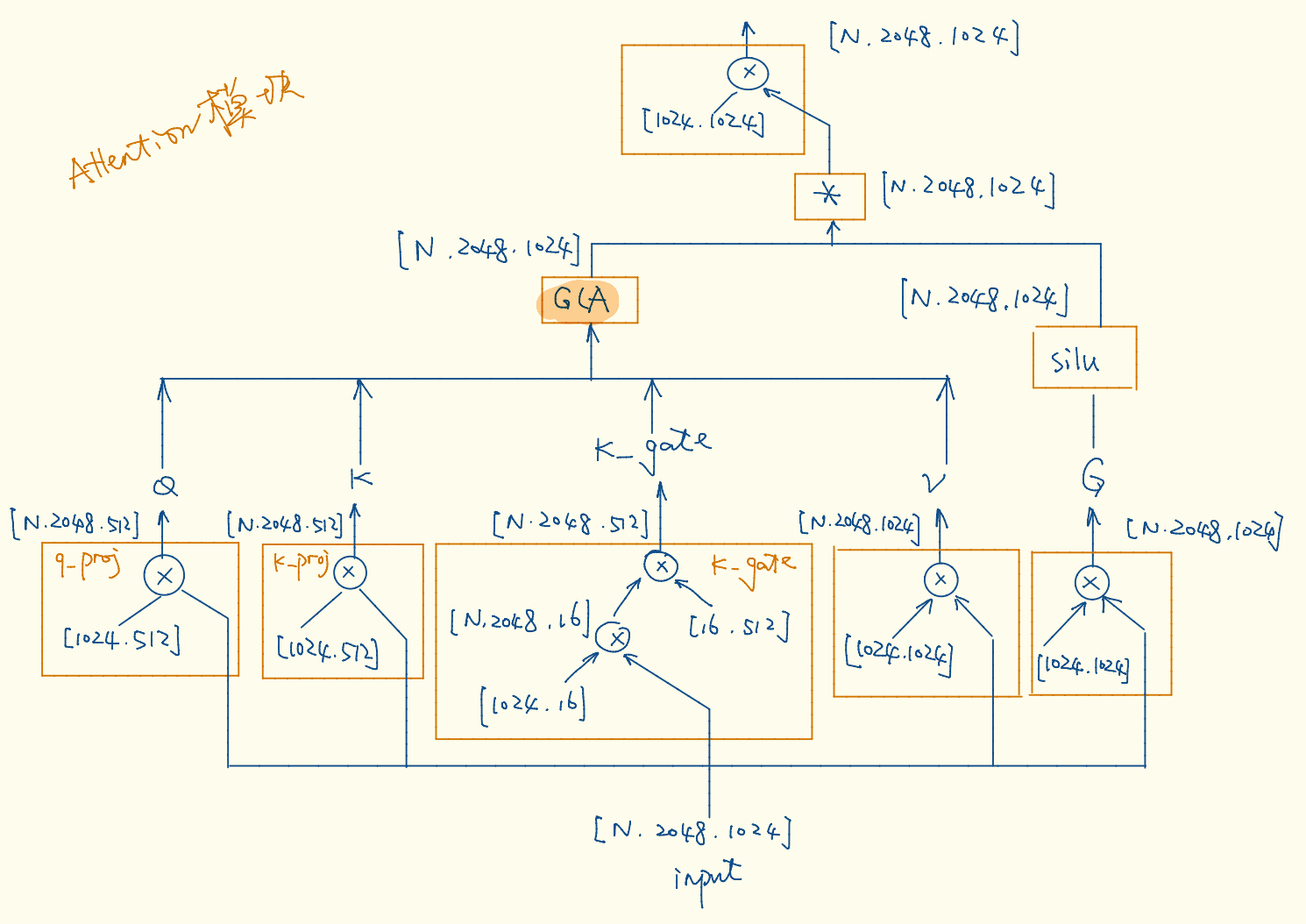
S_t 最初的引入是为了能先计算 $K^T V$，这一步得到的中间结果大小是 $d \times d$，这就小于 $L \times L$

$$S_t = \sum_{i=1}^{t} \phi(k_i^T) v_i =$$

$$= \sum_{i=1}^{t} \left(\left(\prod_{j=i+1}^{t} \alpha_j^T \right) \odot k_i^T v_i \right)$$

联乘




Attention 修改
[N. 2048. 1024]
[1024. 1024]
[N. 2048. 1024]
[N. 2048. 1024]
GLA
[N. 2048. 1024]
silu
Q
K
K_gate
V
G
[N. 2048. 512]
[N. 2048. 512]
[N. 2048. 512]
[N. 2048. 1024]
[N. 2048. 1024]
q-proj
[1024. 512]
k-proj
[1024. 512]
K_gate
[N, 2048, 16]
[16. 512]
[1024. 16]
[1024. 1024]
[1024. 1024]
[N. 2048. 1024]
input