An Introduction to FlashAttention

Chih-Jen Lin
National Taiwan University

Last updated: October 5, 2025

Outline

Inefficiency of Attention Operation I

- Similar to the memory-access issue discussed before for matrix-matrix products, a possible bottleneck of attention is on moving data (i.e., matrices) between lower-level and upper-level memory.
- To analyze this issue, we must check the number of memory accesses.

Attention Operations I

- For the discussion, first we recall details of attention. For simplicity, we consider the single-head attention.
- If the input matrix is

$$\tilde{Z} \in \mathbf{R}^{T \times d},$$

the attention operation is

$$\text{SoftMax}\left(\frac{\tilde{Z} W_Q W_K^\top (\tilde{Z})^\top}{\sqrt{d}}\right) \tilde{Z} W_V. \qquad (1)$$

Attention Operations II

- We consider three trainable weight matrices

$$W_Q \in \mathbf{R}^{d\times d}, W_K \in \mathbf{R}^{d\times d}, W_V \in \mathbf{R}^{d\times d}$$

  to convert the input matrix $\tilde{Z}$ to

$$\tilde{Z}W_Q \in \mathbf{R}^{T\times d}, \quad \tilde{Z}W_K \in \mathbf{R}^{T\times d}, \quad \tilde{Z}W_V \in \mathbf{R}^{T\times d}.$$

Attention Operations III

- In (1), the SoftMax function is applied on each row z of an input matrix in the following way.

$$\text{SoftMax}(\mathbf{z}) = \begin{bmatrix} \frac{\exp(z_1)}{\sum_j \exp(z_j)} \\ \vdots \\ \frac{\exp(z_T)}{\sum_j \exp(z_j)} \end{bmatrix}. \tag{2}$$

Attention Operations IV

- FlashAttention (Dao et al., 2022) defines that

$$\mathbf{Q} := \tilde{Z}W_Q, \quad \mathbf{K} := \tilde{Z}W_K, \quad \mathbf{V} := \tilde{Z}W_V,$$

  and assumes that $\mathbf{Q}, \mathbf{K}, \mathbf{V}$ had been already precomputed.
- Omitting $1/\sqrt{d}$ for simplification, FlashAttention turns (1) into

$$\mathbf{O} := \text{SoftMax}(\mathbf{Q}\mathbf{K}^\top)\mathbf{V}, \tag{3}$$

  where $\mathbf{O} \in \mathbf{R}^{T \times d}$ is the output matrix of the attention.

Memory Accesses in Attention I

- We still assume that our machine has only two layers of memory:
    - main memory, and
    - secondary memory.
- If an operand is not available in main memory, we must transport it from secondary memory.
- Now consider (3) and check intermediate values during computation.

Memory Accesses in Attention II

- We need

$$\mathbf{QK}^\top \in R^{N\times N}, \tag{4}$$
$$\text{SoftMax}(\mathbf{QK}^\top) \in R^{N\times N}, \tag{5}$$
$$\text{SoftMax}(\mathbf{QK}^\top)\mathbf{V} \in R^{N\times d}. \tag{6}$$

- As

$$N \gg d$$

in general, even though the output

$$\text{SoftMax}(\mathbf{QK}^\top)\mathbf{V} \in R^{N\times d},$$

Memory Accesses in Attention III

we can see that storing $N \times N$ matrices is the main difficulty.

- Our first analysis is to assume that

$$N \times N$$

matrices cannot be stored in the main memory, and check the need to move these matrices

- If we consider (4)-(6) as independent operations, immediately we see the following major memory accesses:

Memory Accesses in Attention IV

- write
$$\mathbf{Q}\mathbf{K}^\top \in R^{N\times N} \tag{7}$$
to secondary memory,
- load the matrix (7) from secondary memory to calculate
$$\text{SoftMax}(\mathbf{Q}\mathbf{K}^\top) \tag{8}$$
and write results back to secondary memory, and
- load the matrix in (8) for calculating
$$\text{SoftMax}(\mathbf{Q}\mathbf{K}^\top)\mathbf{V} \in R^{N\times d}.$$

Memory Accesses in Attention V

- We assume that even though storing an $N \times N$ matrix in main memory is not possible, the computer has a way to sequentially work on part of the data and gradually generate the whole results.

  It is just like that we do matrix-matrix products all the time, but never worry that our highest-level memory (i.e., registers) is insufficient to store operands.

Memory Accesses in Attention VI

- Now we conclude that a naive implementation of attention leads to

$$4 \times N^2$$

accesses between main and secondary memory.

Memory Versus Computation I

- We see attention involves the following operations and list their respective cost.

$$\begin{aligned} \mathbf{Q}\mathbf{K}^\top &: 2N^2 d, \\ \text{SoftMax}(\mathbf{Q}\mathbf{K}^\top) &: 3N^2, \\ \text{SoftMax}(\mathbf{Q}\mathbf{K}^\top)\mathbf{V} &: 2N^2 d. \end{aligned}$$

Memory Versus Computation II

- Clearly, if

$$4N^2d \times \text{ cost per operation} < 4N^2 \times \text{ cost per memory access},$$

  then attention is memory bounded.
- Example:
- Therefore, we must reduce the number of memory accesses.

Reducing Memory Accesses I

- One possible strategy to reduce the number of memory accesses is to avoid loading and storing intermediate results.
- That is, we should generate the attention results "part by part." All we need is to sequentially store the finished part back to the secondary memory.
- If the main memory is sufficiently to store

$$N \times d \text{ matrices such as } Q, K, \text{ and } V,$$

we can conduct the following procedure:

Reducing Memory Accesses II

- Load $Q, K,$ and V to main memory.
- For $i = 1, \ldots, N$, calculate

$$\begin{aligned} \mathbf{Q}_{i,:}\mathbf{K}^\top &\in R^{1\times N}, \\ \text{SoftMax}(\mathbf{Q}_{i,:}\mathbf{K}^\top) &\in R^{1\times N}, \\ \text{SoftMax}(\mathbf{Q}_{i,:}\mathbf{K}^\top)\mathbf{V} &\in R^{1\times d} \end{aligned}$$

  and store the ith row of the output matrix to the secondary memory.

Reducing Memory Accesses III

- By this way, the number of memory accesses is reduced to
$$O(Nd).$$
because we never load/store any $N \times N$ matrices.
- Unfortunately, our assumption of that $N \times d$ matrices can be stored in main memory is often untrue
- In this situation, we must load K and V multiple times even for calculating one output row.

Reducing Memory Accesses IV

- A possible strategy is to calculate $|I|$ rows together:

$$\text{SoftMax}(\mathbf{Q}_{I,:}\mathbf{K}^\top)\mathbf{V},$$

  where I is the block of rows that we intend to calculate.
- In this calculation, we must load

$$|I| \text{ rows of } Q, \text{ and the whole } K \text{ and } V.$$

Reducing Memory Accesses V

- We also need to store intermediate block of

$$\mathbf{Q}_{I,:}\mathbf{K}^\top, \tag{9}$$

  which requires

$$|I| \times n$$

  space.
- The largest possible $|I|$ is

$$\frac{M}{n},$$

  where M is the size of the main memory.

Reducing Memory Accesses VI

- Thus the total number of memory accesses is

$$O(\frac{N}{M/n}) \times O(Nd) = O(\frac{N^3 d}{M}). \qquad (10)$$

- In the above discussion, we see that the main bottleneck is to store the intermediate matrix in (9).
- Because the number of rows in (9) is a large number N, $|I|$ must be small. Thus we get a large first term in the calculation of (10).

FlashAttention I

- To reduce the number of memory accesses, let us see if we may avoid storing the intermediate matrix in (9).
- Assume that we split Q to the following row-block form:

$$\begin{bmatrix} Q_{I_1,:} \\ \vdots \\ Q_{I_{\bar{N}},:} \end{bmatrix}$$

with

$$|I_1| = \cdots = |I_{\bar{N}}|.$$

We do the same split for K, V.

FlashAttention II

- Consider I to be any one of $|I_1|, \ldots, |I_{\bar{N}}|$. We have

$$\begin{aligned}
&\text{SoftMax}(\mathbf{Q}_{I,:}\begin{bmatrix}\mathbf{K}_{I_1,:}^\top & \cdots & \mathbf{K}_{I_{\bar{N}},:}^\top\end{bmatrix})\begin{bmatrix}\mathbf{V}_{I_1,:} \\ \vdots \\ \mathbf{V}_{I_{\bar{N}},:}\end{bmatrix} \\
=&\text{SoftMax}(\begin{bmatrix}\mathbf{Q}_{I,:}\mathbf{K}_{I_1,:}^\top & \cdots & \mathbf{Q}_{I,:}\mathbf{K}_{I_{\bar{N}},:}^\top\end{bmatrix})\begin{bmatrix}\mathbf{V}_{I_1,:} \\ \vdots \\ \mathbf{V}_{I_{\bar{N}},:}\end{bmatrix}
\end{aligned}$$

- If there is no SoftMax, we can see the result is

$$(\mathbf{Q}_{I,:}\mathbf{K}_{I_1,:}^\top)\mathbf{V}_{I_1,:} + \cdots + (\mathbf{Q}_{I,:}\mathbf{K}_{I_{\bar{N}},:}^\top)\mathbf{V}_{I_{\bar{N}},:}.$$

FlashAttention III

- We have

$$(\mathbf{Q}_{I,:}\mathbf{K}_{I_1,:}^\top)\mathbf{V}_{I_1,:} \in |I|\times d, \ldots, (\mathbf{Q}_{I,:}\mathbf{K}_{I_{\bar{N}},:}^\top)\mathbf{V}_{I_{\bar{N}},:} \in |I|\times d.$$

- If we sequentially generate each term, there is no need to store the intermediate sub-matrix in (9).
- In this situation, because all we need is a few $|I| \times d$ blocks, we have

$$|I| = O(\frac{M}{d}).$$

FlashAttention IV

Therefore, the number of memory accesses is

$$O(\frac{N}{m/d}) \times O(Nd) = O(\frac{N^2 d^2}{M}).$$

- Unfortunately, we need the whole intermediate matrix in (9) because the SoftMax function involves all elements in each row.

FlashAttention V

- A crucial observation is that if we can have

$$
\begin{aligned}
&\text{SoftMax}(\begin{bmatrix} \mathbf{Q}_{I,:}\mathbf{K}_{I_1,:}^\top & \cdots & \mathbf{Q}_{I,:}\mathbf{K}_{I_{s+1},:}^\top \end{bmatrix}) \begin{bmatrix} \mathbf{V}_{I_1,:} \\ \vdots \\ \mathbf{V}_{I_{s+1},:} \end{bmatrix} \\
=&\Delta_s \odot \left(\text{SoftMax}(\begin{bmatrix} \mathbf{Q}_{I,:}\mathbf{K}_{I_1,:}^\top & \cdots & \mathbf{Q}_{I,:}\mathbf{K}_{I_s,:}^\top \end{bmatrix}) \begin{bmatrix} \mathbf{V}_{I_1,:} \\ \vdots \\ \mathbf{V}_{I_s,:} \end{bmatrix} \right) \\
&+ \Delta_{s+1} \odot \left(\text{SoftMax}(\mathbf{Q}_{I,:}\mathbf{K}_{I_{s+1},:}^\top)\mathbf{V}_{I_{s+1},:} \right),
\end{aligned}
$$

FlashAttention VI

where $\odot$ is the component-wise product, and $\Delta_s, \Delta_{s+1} \in R^d$ are available, then we can manage to get the result

- We have $\forall i \in I_1 \cup \cdots \cup I_s$,

$$\frac{\exp(z_i)}{\sum_{j \in I_1 \cup \cdots \cup I_{s+1}} \exp(z_j)} = \underbrace{\left(\frac{\sum_{j \in I_1 \cup \cdots \cup I_s} \exp(z_j)}{\sum_{j \in I_1 \cup \cdots \cup I_{s+1}} \exp(z_j)} \right)}_{\Delta_1} \frac{\exp(z_i)}{\sum_{j \in I_1 \cup \cdots \cup I_s} \exp(z_j)},$$

FlashAttention VII

and $\forall i \in I_{s+1}$,

$$\frac{\exp(z_i)}{\sum_{j \in I_1 \cup \cdots \cup I_{s+1}} \exp(z_j)} = \underbrace{\left(\frac{\sum_{j \in I_{s+1}} \exp(z_j)}{\sum_{j \in I_1 \cup \cdots \cup I_{s+1}} \exp(z_j)} \right)}_{\Delta_2} \frac{\exp(z_i)}{\sum_{j \in I_{s+1}} \exp(z_j)}.$$

- Clearly, all we need is to maintain

$$\sum_{j \in I_1 \cup \cdots \cup I_s} \exp(z_j). \tag{11}$$

FlashAttention VIII

- When handling $s+1$, we get

$$\sum_{j \in I_{s+1}} \exp(z_j),$$

so we can update (11) by

$$\begin{aligned} & \sum_{j \in I_1 \cup \cdots \cup I_{s+1}} \exp(z_j) \\ = & \sum_{j \in I_1 \cup \cdots \cup I_s} \exp(z_j) + \sum_{j \in I_{s+1}} \exp(z_j). \end{aligned}$$

FlashAttention IX

- The $O(d)$ cost for storing (11) is affordable.
- **We may be more general instead of doing row blocks**

Practical Implementation I

- **talk about the way to calculate and maintain** (11)

Outline

Memory Hierarchy in the GPU I

- In these slides, we will introduce the work "FlashAttention" (Dao et al., 2022), a method to accelerate on-GPU computation in attention layers.
- To understand FlashAttention, it is necessary to first review the GPU memory hierarchy.
- The GPU memory hierarchy is similar to the CPU's introduced in the video borrowed from the course "Numerical Methods".
- Based on this similarity, we also assume that the GPU has only two layers of memory

Memory Hierarchy in the GPU II

- **Cache:** Small and fast, typically around 100 kilobytes (KB) and close to the processor
- **Main Memory:**[1] Large but slower, typically around 10 gigabytes (GB)

- With this hierarchy, GPUs also experience page faults: an operand is not available in the cache and must be transported from the main memory.
- The transportations of operands also take time and are typically measured by the number of memory accesses.

Memory Hierarchy in the GPU III

- When an operation on GPUs has a large number of memory accesses and takes more runtimes on data transportation than computation, it is **memory bounded**; otherwise, it is **computation bounded**.
- However, comparing runtimes usually requires concrete hardware specifications, such as how many floating-point operations per second (FLOPS) a GPU can perform at a given precision.
- For simplicity, we can instead compare complexities to determine what bounds an operation.

Memory Hierarchy in the GPU IV

- Since computation on GPUs is much faster than memory access, when the computation complexity is no greater than the memory-access complexity, the operation is considered memory-bounded.
- The work "FlashAttention" argues that attention is **memory bound** and thus accelerates it by reducing memory usage and access.

[1]Some documents also refer to the main memory as High Bandwidth Memory (HBM) or simply DRAM, since HBM is a high-bandwidth type of DRAM used as the main memory.

Outline

Memory-Insufficient Issue in Attention I

- During the computation (3), there are intermediate values, like
$$QK^\top \in R^{T\times T}.$$
- When T is large, such intermediate values are impossible to be stored in the cache.
- Take GPT-2 for example, $T = 1024$, leading to a memory usage of
$$1024^2 \times 32 \text{ bits} = 4 \text{ megabytes (MB)},$$
when each float number is stored in the single-precision floating-point format.

Memory-Insufficient Issue in Attention II

- This memory usage is much larger than the sizes of most caches, like the 192 kilobytes (KB) of the A100 GPU.
- This memory-insufficient issue cause page faults in the GPU, leading to writes and reads of intermediate results (e.g., $\mathbf{QK}^\top$) to and from the main memory.

Standard Attention Implementation I

- Given the memory-insufficient issue, the standard attention implementation is divided into four steps.

Standard Attention Implementation II

Algorithm 0 Standard Attention Implementation

Require: Matrices $\mathbf{Q}, \mathbf{K}, \mathbf{V} \in \mathbb{R}^{T \times d}$ in the main memory.

1: Load $\mathbf{Q}, \mathbf{K}$ by blocks from the main memory, compute $\mathbf{S} = \mathbf{Q}\mathbf{K}^\top \in \mathbf{R}^{T \times T}$, write $\mathbf{S}$ to the main memory.

2: Read $\mathbf{S}$ from the main memory, compute $\mathbf{P} = \text{SoftMax}(\mathbf{S}) \in \mathbf{R}^{T \times T}$, write $\mathbf{P}$ to the main memory.

3: Load $\mathbf{P}$ and $\mathbf{V}$ by blocks from the main memory, compute $\mathbf{O} = \mathbf{P}\mathbf{V}$, write $\mathbf{O}$ to the main memory.

4: Return $\mathbf{O}$.

Standard Attention Implementation III

- Since the $T \times T$ intermediate matrix, $\mathbf{S}$, can not fit in the cache, Step 1 have to calculate $\mathbf{S}$ by blocks in the cache and then reconstruct it in the main memory.
- Then, Step 2 is forced to read $\mathbf{S}$ from the main memory to apply SoftMax.
- The above process results in a large number of memory accesses, and the same thing also repeatedly occurs with $\mathbf{P}$.
- Specifically, the complexities of memory access to the main memory at each step are

Standard Attention Implementation IV

Step	Read	Write
1	$\Theta(Td)$ for $\mathbf{Q}, \mathbf{K}$	$\Theta(T^2)$ for $\mathbf{S}$
2	$\Theta(T^2)$ for $\mathbf{S}$	$\Theta(T^2)$ for $\mathbf{P}$
3	$\Theta(T^2 + Td)$ for $\mathbf{P}, \mathbf{V}$	$\Theta(Td)$ for $\mathbf{O}$
4	–	–
Total	$\Theta(T^2 + Td)$	$\Theta(T^2 + Td)$

- Both the total memory access and usage complexities are $\Theta(T^2 + Td)$.
- As $T \gg d$ in general, the quadratic term T^2 dominates the complexity.

Standard Attention Implementation V

- When T is large, the $\Theta(T^2 + Td)$ memory accesses can account for a large portion of the runtime of attention, while the memory usage can be prohibitive as well.

Attention is Memory-Bounded I

- We next examine whether attention is memory-bound, as argued in FlashAttention.
- Here is a comparison between the complexities of memory access and computation at each step.

Step	Computation	reads + writes
1	$\Theta(T^2 d)$ for $\mathbf{Q}\mathbf{K}^\top$	$\Theta(T^2 + Td)$
2	$\Theta(T^2)$ for SoftMax$(\mathbf{S})$	$\Theta(T^2)$
3	$\Theta(T^2 d)$ for $\mathbf{P}\mathbf{V}$	$\Theta(T^2 + Td)$
4	–	–

Attention is Memory-Bounded II

- In Step 2, the computation and memory access complexities are on the same scale.
- It indicates that this step is memory bounded on GPUs where computation is much faster than memory access.
- This observation aligns with the argument in the work of "FlashAttention", and show that the $\Theta(T^2)$ complexity arises from the two $T \times T$ intermediate matrices, $\mathbf{S}$ and $\mathbf{P}$.

Attention is Memory-Bounded III

- Therefore, if there is a way to avoid explicitly outputting $\mathbf{S}$ and $\mathbf{P}$, a large number of accesses to the main memory can be saved, thereby alleviating the memory bound.
- This avoidance requires restricting all computations in attention to portions of $\mathbf{S}$ and $\mathbf{P}$ that fit in the cache, rather than the entire matrices.

Outline

Decompose SoftMax I

- FlashAttention succeeds to keep all computations on the cache by decomposing SoftMax wisely.
- To illustrate the details, let us review the common way to compute SoftMax first.
- For numerical stability, the SoftMax of a row vector $\boldsymbol{z} \in \mathbf{R}^{1\times T}$ is computed as

$$m(\boldsymbol{z}) := \max_{j\in\{1,\cdots,T\}} z_j, f(\boldsymbol{z}) := [e^{z_1 - m(\boldsymbol{z})} \ \cdots \ e^{z_T - m(\boldsymbol{z})}],$$

$$\ell(\boldsymbol{z}) := \sum\nolimits_{j=1}^{T} f(\boldsymbol{z})_j, \text{SoftMax}(\boldsymbol{z}) := \frac{f(\boldsymbol{z})}{\ell(\boldsymbol{z})}.$$

Decompose SoftMax II

- Consider T is even, and divide $\boldsymbol{z}$ into two blocks $\boldsymbol{z}^{(1)}, \boldsymbol{z}^{(2)} \in R^{1\times B}$, where $2B = T$.
- Then, we have

$$m(\boldsymbol{z}) = m([\boldsymbol{z}^{(1)} \quad \boldsymbol{z}^{(2)}]) = \max\left(m(\boldsymbol{z}^{(1)}), m(\boldsymbol{z}^{(2)})\right),$$

$$f(\boldsymbol{z}) = \left[e^{m(\boldsymbol{z}^{(1)})-m(\boldsymbol{z})} f(\boldsymbol{z}^{(1)}) \quad e^{m(\boldsymbol{z}^{(2)})-m(\boldsymbol{z})} f(\boldsymbol{z}^{(2)})\right],$$

$$\begin{aligned} \ell(\boldsymbol{z}) &= \ell([\boldsymbol{z}^{(1)} \quad \boldsymbol{z}^{(2)}]) \\ &= e^{m(\boldsymbol{z}^{(1)})-m(\boldsymbol{z})} \ell(\boldsymbol{z}^{(1)}) + e^{m(\boldsymbol{z}^{(2)})-m(\boldsymbol{z})} \ell(\boldsymbol{z}^{(2)}). \end{aligned}$$

Decompose SoftMax III

- Therefore,

$$\text{SoftMax}(\boldsymbol{z}) = \frac{f(\boldsymbol{z})}{\ell(\boldsymbol{z})},$$
$$= \frac{\left[e^{m(\boldsymbol{z}^{(1)})-m(\boldsymbol{z})} f(\boldsymbol{z}^{(1)}) \quad e^{m(\boldsymbol{z}^{(2)})-m(\boldsymbol{z})} f(\boldsymbol{z}^{(2)})\right]}{e^{m(\boldsymbol{z}^{(1)})-m(\boldsymbol{z})} \ell(\boldsymbol{z}^{(1)}) + e^{m(\boldsymbol{z}^{(2)})-m(\boldsymbol{z})} \ell(\boldsymbol{z}^{(2)})}. \tag{12}$$

- As shown above, the computations of SoftMax can naturally split into two parts (marked in blue and red), each of which corresponding to one block, $\boldsymbol{z}^{(1)}$ or $\boldsymbol{z}^{(2)}$.

Decompose Attention I

- With (12), we can decompose all the computations in attention into blocks now.
- For illustration, consider that $\mathbf{Q}, \mathbf{K}$, and $\mathbf{V}$ can be decomposed into two blocks, like

$$\mathbf{Q} = \begin{bmatrix} \mathbf{Q}^{(1)} \\ \mathbf{Q}^{(2)} \end{bmatrix}, \mathbf{K} = \begin{bmatrix} \mathbf{K}^{(1)} \\ \mathbf{K}^{(2)} \end{bmatrix}, \mathbf{V} = \begin{bmatrix} \mathbf{V}^{(1)} \\ \mathbf{V}^{(2)} \end{bmatrix},$$

where each block is of shape (B, d), like $\mathbf{Q}^{(1)} \in \mathbf{R}^{B \times d}$.

Decompose Attention II

- Then,

$$\begin{aligned} \mathbf{S} = \mathbf{Q}\mathbf{K}^\top &= \begin{bmatrix} \mathbf{Q}^{(1)}(\mathbf{K}^{(1)})^\top & \mathbf{Q}^{(1)}(\mathbf{K}^{(2)})^\top \\ \mathbf{Q}^{(2)}(\mathbf{K}^{(1)})^\top & \mathbf{Q}^{(2)}(\mathbf{K}^{(2)})^\top \end{bmatrix}, \\ &= \begin{bmatrix} \mathbf{S}^{(11)} & \mathbf{S}^{(12)} \\ \mathbf{S}^{(21)} & \mathbf{S}^{(22)} \end{bmatrix}. \end{aligned} \tag{13}$$

Decompose Attention III

- Since SoftMax is applied on each row of $\mathbf{S}$ independently,

$$\begin{aligned}\mathbf{P} &= \text{SoftMax}(\mathbf{S}),\\ &= \text{SoftMax}\left(\begin{bmatrix}\mathbf{S}^{(11)} & \mathbf{S}^{(12)}\\ \mathbf{S}^{(21)} & \mathbf{S}^{(22)}\end{bmatrix}\right),\\ &= \begin{bmatrix}\text{SoftMax}\left(\begin{bmatrix}\mathbf{S}^{(11)} & \mathbf{S}^{(12)}\end{bmatrix}\right)\\ \text{SoftMax}\left(\begin{bmatrix}\mathbf{S}^{(21)} & \mathbf{S}^{(22)}\end{bmatrix}\right)\end{bmatrix}.\end{aligned}$$

Decompose Attention IV

- Therefore,

$$\mathbf{PV} = \begin{bmatrix} \text{SoftMax}\left(\begin{bmatrix}\mathbf{S}^{(11)} & \mathbf{S}^{(12)}\end{bmatrix}\right) \\ \text{SoftMax}\left(\begin{bmatrix}\mathbf{S}^{(21)} & \mathbf{S}^{(22)}\end{bmatrix}\right) \end{bmatrix} \begin{bmatrix} \mathbf{V}^{(1)} \\ \mathbf{V}^{(2)} \end{bmatrix},$$

$$= \begin{bmatrix} \text{SoftMax}\left(\begin{bmatrix}\mathbf{S}^{(11)} & \mathbf{S}^{(12)}\end{bmatrix}\right) \begin{bmatrix} \mathbf{V}^{(1)} \\ \mathbf{V}^{(2)} \end{bmatrix} \\ \text{SoftMax}\left(\begin{bmatrix}\mathbf{S}^{(21)} & \mathbf{S}^{(22)}\end{bmatrix}\right) \begin{bmatrix} \mathbf{V}^{(1)} \\ \mathbf{V}^{(2)} \end{bmatrix} \end{bmatrix}. \quad (14)$$

Decompose Attention V

- With 12, the first row of (14) becomes

$$
\begin{aligned}
&\frac{\begin{bmatrix} e^{m(\boldsymbol{z}^{(1)})-m(\boldsymbol{z})} f(\boldsymbol{z}^{(1)}) & e^{m(\boldsymbol{z}^{(2)})-m(\boldsymbol{z})} f(\boldsymbol{z}^{(2)}) \end{bmatrix} \begin{bmatrix} \mathbf{V}^{(1)} \\ \mathbf{V}^{(2)} \end{bmatrix}}{e^{m(\boldsymbol{z}^{(1)})-m(\boldsymbol{z})} \ell(\boldsymbol{z}^{(1)}) + e^{m(\boldsymbol{z}^{(2)})-m(\boldsymbol{z})} \ell(\boldsymbol{z}^{(2)})}, \\
=&\frac{e^{m(\boldsymbol{z}^{(1)})-m(\boldsymbol{z})} f(\boldsymbol{z}^{(1)}) \mathbf{V}^{(1)} + e^{m(\boldsymbol{z}^{(2)})-m(\boldsymbol{z})} f(\boldsymbol{z}^{(2)}) \mathbf{V}^{(2)}}{e^{m(\boldsymbol{z}^{(1)})-m(\boldsymbol{z})} \ell(\boldsymbol{z}^{(1)}) + e^{m(\boldsymbol{z}^{(2)})-m(\boldsymbol{z})} \ell(\boldsymbol{z}^{(2)})},
\end{aligned}
\tag{15}
$$

where we take $\boldsymbol{z}^{(1)} := \mathbf{S}^{(11)}$, and $\boldsymbol{z}^{(2)} := \mathbf{S}^{(12)}$.

Decompose Attention VI

- Note that the blue and red parts of (15) are still not entirely confined within their corresponding block due to $m(\boldsymbol{z})$, which is the global maximum of $\boldsymbol{z}$.
- To deal with $m(\boldsymbol{z})$, FlashAttention iterate over blocks to calculate (15) while caching two variables at very low cost.[2]
- Specifically,

[2] It may seem that we can simply multiply both the numerator and denominator by $e^{m(\boldsymbol{z})}$ to remove $m(\boldsymbol{z})$ and make the blue and red parts independent. However, $m(\boldsymbol{z})$ is essential for ensuring numerical stability and therefore cannot be removed by this way.

References I

T. Dao, D. Fu, S. Ermon, A. Rudra, and C. Ré. FlashAttention: Fast and memory-efficient exact attention with IO-awareness. In *Advances in Neural Information Processing Systems*, volume 35, pages 16344–16359, 2022.