

An Introduction to FlashAttention

Chin-Hui Chu, Ya-Xu Liu, Chih-Jen Lin

Last updated: March 13, 2026

Outline

- 1 Introduction
- 2 Standard Attention: Forward Calculation
- 3 Standard Attention: Backward Calculation
- 4 The Bottleneck on Memory Usage and Access
- 5 Flash Attention: Forward Calculation
- 6 Flash Attention: Backward Calculation
- 7 Appendix

Outline

- 1 Introduction
- 2 Standard Attention: Forward Calculation
- 3 Standard Attention: Backward Calculation
- 4 The Bottleneck on Memory Usage and Access
- 5 Flash Attention: Forward Calculation
- 6 Flash Attention: Backward Calculation
- 7 Appendix

Attention and Its Bottleneck I

- The Transformer architecture proposed by **Vaswani et al. (2017)** has become dominant in modern deep learning.
- However, scaling Transformers to long contexts remains challenging due to their attention operations, which have high computation and memory complexity in sequence length (discussed later in this presentation).
- Many prior works have attempted to address this challenge, among which **FlashAttention** (**Dao et al., 2022**) is one of the most well-known.

Attention and Its Bottleneck II

- In these slides, we introduce FlashAttention and its subsequent variants.
- To better understand this challenge, we first analyze the standard forward and backward calculations of the attention operation.
- We then identify the key bottleneck in attention.
- Finally, we derive the FlashAttention algorithm and show how it resolves this bottleneck.

Outline

- 1 Introduction
- 2 Standard Attention: Forward Calculation**
- 3 Standard Attention: Backward Calculation
- 4 The Bottleneck on Memory Usage and Access
- 5 Flash Attention: Forward Calculation
- 6 Flash Attention: Backward Calculation
- 7 Appendix

Attention Operations I

- We first recall the forward calculation of the attention defined in Vaswani et al. (2017),

$$\text{Attention}(Q, K, V) := \text{SoftMax}\left(\frac{QK^\top}{\sqrt{d}}\right)V, \quad (1)$$

where $Q, K, V \in \mathbf{R}^{T \times d}$ are the input matrices, T is the sequence length, and d is the embedding dimension.

Attention Operations II

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

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

Attention Operations III

- However, instead of performing (1), Vaswani et al. (2017) found it more beneficial to perform multi-head attention,

$$\text{MultiHead}(Q, K, V) := \text{Concat}(\text{head}_1, \dots, \text{head}_h)W_O, \quad (3)$$

where h , a divisor of d , is the number of heads, and

$$\begin{aligned} \text{head}_i &= \text{Attention}(QW_Q^i, KW_K^i, VW_V^i) \\ &= \text{SoftMax}\left(\frac{QW_Q^i(KW_K^i)^\top}{\sqrt{d}}\right)VW_V^i \in \mathbf{R}^{T \times d/h}. \end{aligned}$$

Attention Operations IV

- Here $\forall i, W_Q^i, W_K^i, W_V^i \in \mathbf{R}^{d \times d/h}$ and $W_O \in \mathbf{R}^{d \times d}$ are trainable weight matrices.
- Moreover, attention is most commonly used in the self-attention setting, where Q, K, V come from the same input matrix $\tilde{Z} \in \mathbf{R}^{T \times d}$, that is,

$$\text{MultiHead}(Q = \tilde{Z}, K = \tilde{Z}, V = \tilde{Z}).$$

Attention Operations V

- Then, we have the multi-head self-attention,

$$\begin{aligned}
 & \text{MultiHeadSelfAttention}(\tilde{Z}) \\
 & := \text{MultiHead}(Q = \tilde{Z}, K = \tilde{Z}, V = \tilde{Z}) \quad (4) \\
 & = \text{Concat}(\text{head}_1, \dots, \text{head}_h)W_O,
 \end{aligned}$$

where

$$\begin{aligned}
 \text{head}_i & = \text{Attention}(\tilde{Z}W_Q^i, \tilde{Z}W_K^i, \tilde{Z}W_V^i) \\
 & = \text{SoftMax}\left(\frac{\tilde{Z}W_Q^i(\tilde{Z}W_K^i)^\top}{\sqrt{d}}\right)\tilde{Z}W_V^i \in \mathbf{R}^{T \times d/h}.
 \end{aligned}$$

Attention Operations VI

- Because the multi-head situation is similar to the single case, for our discussion we assume $h = 1$ (i.e., one head).
- If we follow [Dao et al. \(2022\)](#) to abuse the notation by redefining Q, K, V as

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

then what we calculate is

$$\text{Attention}(Q, K, V)W_O. \quad (5)$$

Attention Operations VII

- For later discussion, we further decompose the computation in (5) into the following steps:

$$Q = \tilde{Z}W_Q, K = \tilde{Z}W_K, V = \tilde{Z}W_V, \quad (6)$$

$$S = QK^\top, \quad (7)$$

$$P = \text{SoftMax}(S), \quad (8)$$

$$O = PV, \quad (9)$$

$$\tilde{Z}^{\text{out}} = OW_O, \quad (10)$$

Attention Operations VIII

where $\tilde{Z}, \tilde{Z}^{\text{out}} \in \mathbf{R}^{T \times d}$ are the input and output matrices, while $Q, K, V, O \in \mathbf{R}^{T \times d}$ and $S, P \in \mathbf{R}^{T \times T}$ denote the intermediate results.

Attention Operations IX

- The following graph shows the connection between variables:

Attention Operations X

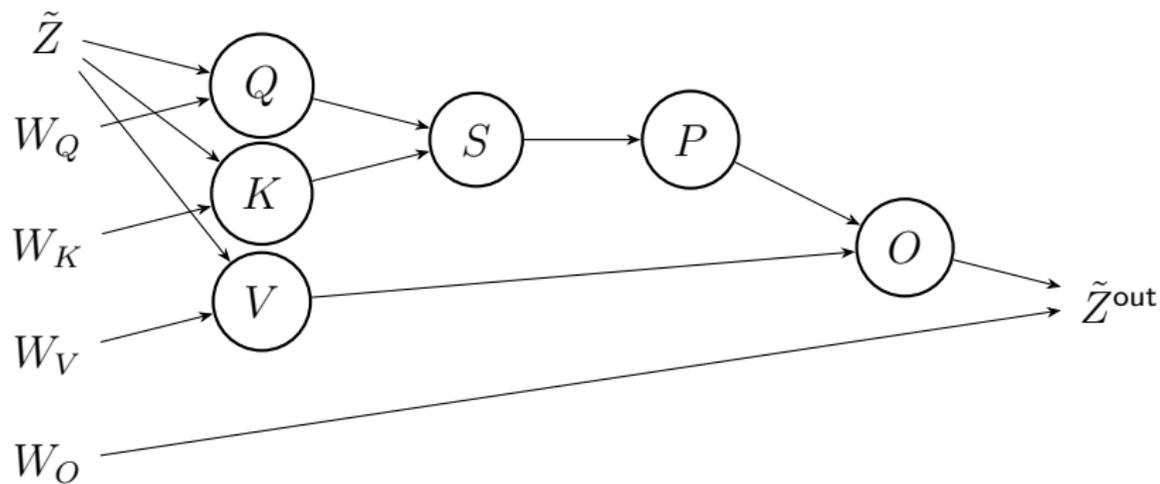


Figure 1: The computation graph of attention.

Attention Operations XI

- From the graph, we can clearly see that

$$O = \text{Attention}(Q, K, V) \quad (11)$$

is a sub-graph.

- We can treat (11) as a general component for the use in (5) and other places.

Attention Operations XII

- Therefore, in this document, we focus on the efficient forward and backward computation of

input: $Q, K, V,$

output: $O = \text{Attention}(Q, K, V),$

which includes (7)-(9).

- In the Appendix, we extend the derivation to cover (10) and (6).

Outline

- 1 Introduction
- 2 Standard Attention: Forward Calculation
- 3 Standard Attention: Backward Calculation**
- 4 The Bottleneck on Memory Usage and Access
- 5 Flash Attention: Forward Calculation
- 6 Flash Attention: Backward Calculation
- 7 Appendix

The Need to Derive Backward Calculation

- Modern machine learning tools apply automatic differentiation to compute gradients through backpropagation. Thus, for any given network architecture, we generally do not need to derive explicit backward operations.
- However, as attention is often the bottleneck in the entire computation, explicitly implementing the backward operation improves the efficiency.

The Need to Derive Backward Calculation

II

- In addition, memory-efficient attention algorithms such as FlashAttention typically have a low-level implementation invoked by higher-level frameworks such as PyTorch or TensorFlow.
- The automatic differentiation functionality of general-purpose deep learning frameworks may not be able to access low-level implementations, so cannot automatically generate the necessary gradient operations.

The Need to Derive Backward Calculation III

- Subsequently, we derive the backward calculation of attention.

Backward Pass: Problem Setup I

- We assume that, in the forward calculation, the output matrix O from (11) is passed to subsequent operations, and a scalar-valued loss L is computed at the end.
- During backward propagation, the attention operation receives from its subsequent operations the gradient

$$\frac{\partial L}{\partial O} \in \mathbf{R}^{T \times d}. \quad (12)$$

Backward Pass: Problem Setup II

- With (12), our objective is to compute the gradients with respect to the input matrices Q , K , and V ,

$$\frac{\partial L}{\partial Q}, \quad \frac{\partial L}{\partial K}, \quad \text{and} \quad \frac{\partial L}{\partial V}.$$

- We derive the gradients by applying the chain rule backward through (9), (8), and (7).

Gradient with Respect to V I

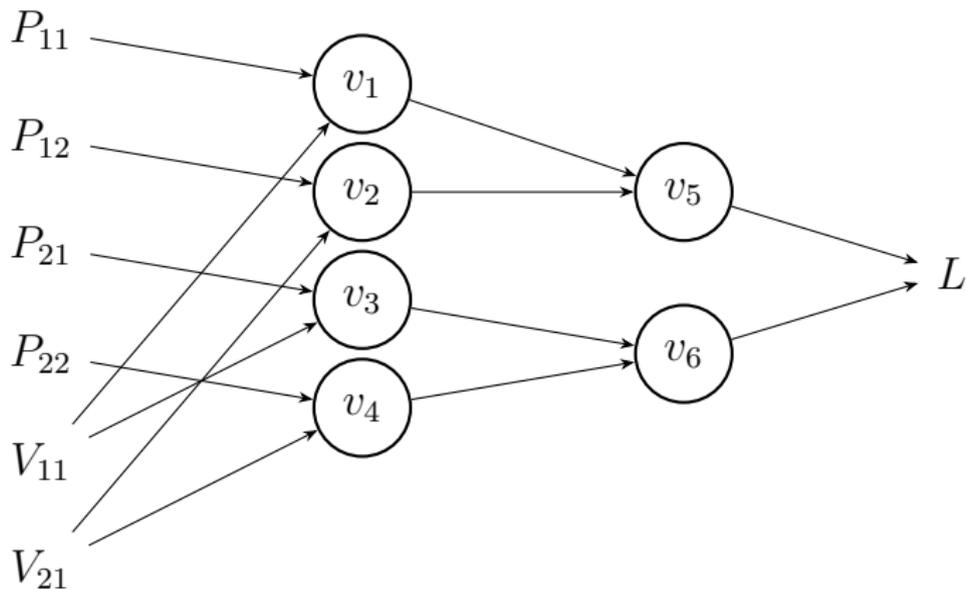
- We begin by deriving the gradient with respect to the value matrix $V \in \mathbf{R}^{T \times d}$.
- From (9),

$$O_{ik} = \sum_{j=1}^T P_{ij} V_{jk}, \quad (13)$$

for $i = 1, \dots, T$ and $k = 1, \dots, d$.

- We observe that each element V_{jk} influences all output elements O_{ik} in the k -th column of O .
- The following computational graph illustrates this relationship for a simplified $T = 2, d = 1$ case.

Gradient with Respect to V II



Gradient with Respect to V III

- The intermediate nodes in the graph represent the following computations:

$$v_1 = P_{11}V_{11},$$

$$v_2 = P_{12}V_{21},$$

$$v_3 = P_{21}V_{11},$$

$$v_4 = P_{22}V_{21},$$

$$v_5 = v_1 + v_2 = O_{11}, \quad v_6 = v_3 + v_4 = O_{21}.$$

- To find the gradient of the loss L with respect to an element, such as V_{11} , we apply the chain rule and

Gradient with Respect to V IV

sum the contributions from all paths through which V_{11} affects L .

$$\begin{aligned}
 \frac{\partial L}{\partial V_{11}} &= \frac{\partial L}{\partial v_5} \frac{\partial v_5}{\partial v_1} \frac{\partial v_1}{\partial V_{11}} + \frac{\partial L}{\partial v_6} \frac{\partial v_6}{\partial v_3} \frac{\partial v_3}{\partial V_{11}} \\
 &= \frac{\partial L}{\partial O_{11}} \cdot 1 \cdot P_{11} + \frac{\partial L}{\partial O_{21}} \cdot 1 \cdot P_{21} \\
 &= \sum_{i=1}^T \frac{\partial L}{\partial O_{i1}} P_{i1}.
 \end{aligned} \tag{14}$$

Gradient with Respect to V

- For an arbitrary element V_{jk} , we can generalize (14) to

$$\frac{\partial L}{\partial V_{jk}} = \sum_{i=1}^T \frac{\partial L}{\partial O_{ik}} \frac{\partial O_{ik}}{\partial V_{jk}} = \sum_{i=1}^T \frac{\partial L}{\partial O_{ik}} P_{ij}. \quad (15)$$

- We can represent (15) in the following matrix form

$$\frac{\partial L}{\partial V} = P^\top \frac{\partial L}{\partial O} \in \mathbf{R}^{T \times d},$$

Gradient with Respect to V VI

because of

$$\left(P^\top \frac{\partial L}{\partial O} \right)_{jk} = \sum_{i=1}^T (P^\top)_{ji} \left(\frac{\partial L}{\partial O} \right)_{ik} = \sum_{i=1}^T \frac{\partial L}{\partial O_{ik}} P_{ij}.$$

Gradient with Respect to P I

- The derivation for $\partial L / \partial P$ is similar to $\partial L / \partial V$, since both appear in the matrix multiplication $O = PV$.
- An element P_{ij} influences outputs through

$$O_{ik} = \sum_{j'=1}^T P_{ij'} V_{j'k},$$

for $i = 1, \dots, T$ and $k = 1, \dots, d$.

Gradient with Respect to P II

- Applying the chain rule gives

$$\frac{\partial L}{\partial P_{ij}} = \sum_{k=1}^d \frac{\partial L}{\partial O_{ik}} \frac{\partial O_{ik}}{\partial P_{ij}} = \sum_{k=1}^d \frac{\partial L}{\partial O_{ik}} V_{jk}. \quad (16)$$

- We can represent (16) in the following matrix form

$$\frac{\partial L}{\partial P} = \frac{\partial L}{\partial O} V^{\top} \in \mathbf{R}^{T \times T}, \quad (17)$$

Gradient with Respect to P III

because of

$$\left(\frac{\partial L}{\partial O} V^\top\right)_{ij} = \sum_{k=1}^d \left(\frac{\partial L}{\partial O}\right)_{ik} (V^\top)_{kj} = \sum_{k=1}^d \frac{\partial L}{\partial O_{ik}} V_{jk}. \quad (18)$$

Gradient with Respect to S I

- From (8), next we should derive the gradient with respect to the matrix $S \in \mathbf{R}^{T \times T}$.
- Given

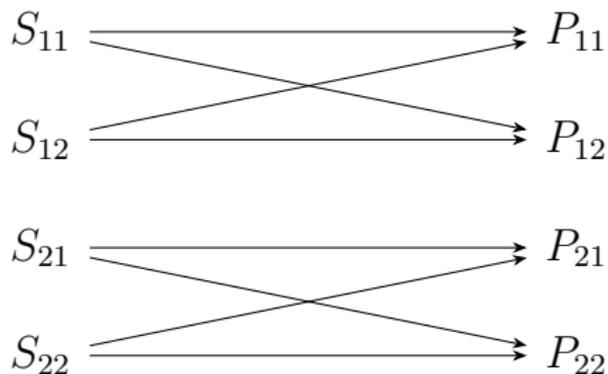
$$\frac{\partial L}{\partial P} \in \mathbf{R}^{T \times T},$$

we propagate this gradient through the softmax operation defined in (8).

- We note that softmax operation independently normalizes each row of S to form the corresponding row of P .

Gradient with Respect to S II

- The following computational graph gives an illustration of $T = 2$.



Gradient with Respect to S III

- In the graph, S_{11} affects the loss L through both P_{11} and P_{12} , so the chain rule yields

$$\frac{\partial L}{\partial S_{11}} = \frac{\partial L}{\partial P_{11}} \frac{\partial P_{11}}{\partial S_{11}} + \frac{\partial L}{\partial P_{12}} \frac{\partial P_{12}}{\partial S_{11}}.$$

- Since $\partial L/\partial P$ is known from (17), it remains to compute

$$\frac{\partial P_{1k}}{\partial S_{11}}, \text{ for } k = 1, 2.$$

Gradient with Respect to S IV

We defer the derivation and instead check the general form of

$$\frac{\partial L}{\partial S_{ij}} \quad \forall i, j = 1, \dots, T.$$

Because S_{ij} affects the loss L through all elements in row i of P , the chain rule yields

$$\frac{\partial L}{\partial S_{ij}} = \sum_{k=1}^T \frac{\partial L}{\partial P_{ik}} \frac{\partial P_{ik}}{\partial S_{ij}}. \quad (19)$$

Gradient with Respect to $S V$

- To compute the partial derivatives $\partial P_{ik} / \partial S_{ij}$, we take the logarithm of both sides of the softmax expression

$$P_{ik} = \frac{e^{S_{ik}}}{\sum_{l=1}^T e^{S_{il}}}$$

to obtain

$$\ln P_{ik} = S_{ik} - \ln \sum_{l=1}^T e^{S_{il}}.$$

Gradient with Respect to S VI

- Then,

$$\frac{1}{P_{ik}} \frac{\partial P_{ik}}{\partial S_{ij}} = \delta_{kj} - \frac{e^{S_{ij}}}{\sum_{l=1}^T e^{S_{il}}} = \delta_{kj} - P_{ij},$$

where

$$\delta_{kj} = \begin{cases} 1 & \text{if } k = j, \\ 0 & \text{otherwise.} \end{cases}$$

- Therefore,

$$\frac{\partial P_{ik}}{\partial S_{ij}} = \begin{cases} P_{ik}(1 - P_{ik}) & \text{if } k = j, \\ -P_{ik}P_{ij} & \text{if } k \neq j. \end{cases} \quad (20)$$

Gradient with Respect to S VII

- Substituting (20) into (19) gives

$$\begin{aligned}
 \frac{\partial L}{\partial S_{ij}} &= \frac{\partial L}{\partial P_{ij}} P_{ij} (1 - P_{ij}) + \sum_{k:k \neq j} \frac{\partial L}{\partial P_{ik}} (-P_{ik} P_{ij}) \\
 &= P_{ij} \left(\frac{\partial L}{\partial P_{ij}} - \sum_{k=1}^T P_{ik} \frac{\partial L}{\partial P_{ik}} \right). \tag{21}
 \end{aligned}$$

Gradient with Respect to S VIII

- By defining

$$\gamma_i = \sum_{k=1}^T P_{ik} \frac{\partial L}{\partial P_{ik}}, \quad (22)$$

$$\gamma = \text{rowsum} \left(\frac{\partial L}{\partial P} \odot P \right) \in \mathbf{R}^T, \quad (23)$$

we can further write (21) in a matrix form

$$\frac{\partial L}{\partial S} = P \odot \left(\frac{\partial L}{\partial P} - \gamma \mathbf{1}^\top \right) \in \mathbf{R}^{T \times T},$$

Gradient with Respect to S IX

where $\mathbf{1} \in \mathbf{R}^T$ is the vector of all ones.

- We note that (23) requires $O(T^2)$ operations.
- To reduce the number of operations, we rewrite (23) to use

$$O \in \mathbf{R}^{T \times d} \quad \text{and} \quad \frac{\partial L}{\partial O} \in \mathbf{R}^{T \times d}.$$

Gradient with Respect to $S X$

- Substituting (17) into (22) and applying (13), we have

$$\begin{aligned}
 \sum_{k=1}^T P_{ik} \frac{\partial L}{\partial P_{ik}} &= \sum_{k=1}^T P_{ik} \sum_{l=1}^d \frac{\partial L}{\partial O_{il}} V_{kl} \\
 &= \sum_{l=1}^d \frac{\partial L}{\partial O_{il}} \sum_{k=1}^T P_{ik} V_{kl} \\
 &= \sum_{l=1}^d \frac{\partial L}{\partial O_{il}} O_{il}.
 \end{aligned}$$

Gradient with Respect to S XI

- Thus, we can rewrite (23) as

$$\gamma = \text{rowsum} \left(\frac{\partial L}{\partial O} \odot O \right) \in \mathbf{R}^T. \quad (24)$$

- The number of operations is reduced to $O(Td)$.

Gradient with Respect to Q and K I

- From (7), next we derive

$$\frac{\partial L}{\partial Q} \in \mathbf{R}^{T \times d} \quad \text{and} \quad \frac{\partial L}{\partial K} \in \mathbf{R}^{T \times d}.$$

We note that (7) is in a similar form to (9), where they are respectively

$$S = QK^{\top} \quad \text{and} \quad O = PV.$$

Gradient with Respect to Q and K II

- The results derived earlier for

$$\frac{\partial L}{\partial V} \quad \text{and} \quad \frac{\partial L}{\partial P}$$

directly lead us to have

$$\frac{\partial L}{\partial Q} = \frac{\partial L}{\partial S} (K^T)^T = \frac{\partial L}{\partial S} K,$$

$$\frac{\partial L}{\partial K} = \left(\frac{\partial L}{\partial (K^T)} \right)^T = \left(Q^T \frac{\partial L}{\partial S} \right)^T = \left(\frac{\partial L}{\partial S} \right)^T Q.$$

Summary of Backward Calculation I

- From the given $\partial L / \partial O \in \mathbf{R}^{T \times d}$, a summary of operations is as follows:

$$\frac{\partial L}{\partial V} = P^\top \frac{\partial L}{\partial O} \in \mathbf{R}^{T \times d}$$
$$\frac{\partial L}{\partial P} = \frac{\partial L}{\partial O} V^\top \in \mathbf{R}^{T \times T}$$

Summary of Backward Calculation II

$$\frac{\partial L}{\partial S} = P \odot \left(\frac{\partial L}{\partial P} - \gamma \mathbf{1}^\top \right) \in \mathbf{R}^{T \times T},$$

where $\gamma = \text{rowsum} \left(\frac{\partial L}{\partial O} \odot O \right) \in \mathbf{R}^T$

$$\frac{\partial L}{\partial Q} = \frac{\partial L}{\partial S} K \in \mathbf{R}^{T \times d}$$

$$\frac{\partial L}{\partial K} = \left(\frac{\partial L}{\partial S} \right)^\top Q \in \mathbf{R}^{T \times d}$$

Outline

- 1 Introduction
- 2 Standard Attention: Forward Calculation
- 3 Standard Attention: Backward Calculation
- 4 The Bottleneck on Memory Usage and Access**
- 5 Flash Attention: Forward Calculation
- 6 Flash Attention: Backward Calculation
- 7 Appendix

Memory Accesses in Attention I

- We 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 (11) and check intermediate values during computation.

Memory Accesses in Attention II

- We need

$$QK^{\top} \in \mathbf{R}^{T \times T} \quad (25)$$

$$\text{SoftMax}(QK^{\top}) \in \mathbf{R}^{T \times T} \quad (26)$$

$$\text{SoftMax}(QK^{\top})V \in \mathbf{R}^{T \times d} \quad (27)$$

- As in general

$$T \gg d,$$

even though the output

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

Memory Accesses in Attention III

is smaller, storing $T \times T$ intermediate matrices is the main difficulty.

Insufficient Memory to Store $T \times T$ Intermediate Matrices I

- Our first analysis is to assume that

$$T \times T$$

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

- If we consider (25)-(27) as independent operations, immediately we see the following major memory accesses:

Insufficient Memory to Store $T \times T$ Intermediate Matrices II

- write

$$QK^{\top} \in \mathbf{R}^{T \times T} \quad (28)$$

to secondary memory,

- load the matrix (28) from secondary memory to calculate

$$\text{SoftMax}(QK^{\top}) \quad (29)$$

and write results back to secondary memory,
and

Insufficient Memory to Store $T \times T$ Intermediate Matrices III

- load the matrix in (29) for calculating

$$\text{SoftMax}(QK^T)V \in \mathbf{R}^{T \times d}.$$

- We assume that even though storing a $T \times T$ matrix in main memory is not possible, the computer has a way to sequentially work on part of the input data in main memory and gradually generate/save part of the whole output to secondary memory.

Insufficient Memory to Store $T \times T$ Intermediate Matrices IV

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

$$4 \times T^2 \quad (30)$$

accesses between main and secondary memory.

Memory Versus Computation I

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

$$QK^{\top} : 2T^2d,$$

$$\text{SoftMax}(QK^{\top}) : 3T^2,$$

$$\text{SoftMax}(QK^{\top})V : 2T^2d.$$

Memory Versus Computation II

- Thus, the total computational cost is around $4T^2d$. With (30), we see that if

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

then attention is memory bounded.

- We give the following example.
 - Consider $T = 1,024$ and $d = 64$, and assume that we run on an A100 80GB SXM GPU.

Memory Versus Computation III

- This GPU sustains up to 312 tera floating point operations per second (TFLOPS) for half-precision floating-point format (FP16) operations¹.
- The GPU is equipped with 80 GB of high-bandwidth memory (HBM), corresponding to the secondary memory in our setting and offering a memory bandwidth of 2.04 tera bytes per second (TB/s).

Memory Versus Computation IV

- Then, the total computational cost is

$$\frac{4 \times 1024^2 \times 64 \text{ FLOPs}}{312 \text{ TFLOPS}} \approx 0.86 \mu\text{s}.$$

- The total memory-access cost is

$$\frac{4 \times 1024^2 \times 2 \text{ bytes}}{2.04 \text{ TB/s}} \approx 3.74 \mu\text{s}.$$

Note that we assume each value is stored in FP16 (two bytes).

Memory Versus Computation V

- Based on the above discussion, we should reduce memory accesses in order to accelerate the attention operation.

¹[https:](https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/a100/pdf/nvidia-a100-datasheet-nvidia-us-2188504-web.pdf)

[//www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/a100/pdf/nvidia-a100-datasheet-nvidia-us-2188504-web.pdf](https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/a100/pdf/nvidia-a100-datasheet-nvidia-us-2188504-web.pdf)

Backward Pass: Insufficient Memory to Store $T \times T$ Intermediate Matrices I

- The backward pass also requires computing and storing $T \times T$ intermediate matrices as in the forward pass.
- Specifically, we derived the following $T \times T$ intermediate gradients in (16) and (21):

$$\frac{\partial L}{\partial P} \quad \text{and} \quad \frac{\partial L}{\partial S} \in \mathbf{R}^{T \times T}.$$

Backward Pass: Insufficient Memory to Store $T \times T$ Intermediate Matrices II

- As in the forward pass, our first analysis assumes that

$$T \times T$$

matrices cannot be stored in main memory.

- Under this assumption, the major memory accesses are as follows.

Backward Pass: Insufficient Memory to Store $T \times T$ Intermediate Matrices III

- load $P \in \mathbf{R}^{T \times T}$ from secondary memory (stored during forward pass) to compute

$$\frac{\partial L}{\partial V} = P^\top \frac{\partial L}{\partial O} \in \mathbf{R}^{T \times d},$$

- compute and write

$$\frac{\partial L}{\partial P} = \frac{\partial L}{\partial O} V^\top \in \mathbf{R}^{T \times T} \quad (31)$$

to secondary memory,

Backward Pass: Insufficient Memory to Store $T \times T$ Intermediate Matrices IV

- load P and (31) from secondary memory to compute

$$\frac{\partial L}{\partial S} = P \odot \left(\frac{\partial L}{\partial P} - \gamma \mathbf{1}^\top \right) \in \mathbf{R}^{T \times T}, \quad (32)$$

and write the result to secondary memory,
and

Backward Pass: Insufficient Memory to Store $T \times T$ Intermediate Matrices V

- load (32) from secondary memory twice to compute

$$\frac{\partial L}{\partial Q} = \frac{\partial L}{\partial S} K \in \mathbf{R}^{T \times d},$$

$$\frac{\partial L}{\partial K} = \left(\frac{\partial L}{\partial S} \right)^\top Q \in \mathbf{R}^{T \times d}.$$

Backward Pass: Insufficient Memory to Store $T \times T$ Intermediate Matrices VI

- We assume that although storing a complete $T \times T$ matrix in main memory is not possible, the computer can sequentially process parts of the input data to gradually generate the output.
- A simple count of load and write operations in the above procedure indicates approximately

$$7 \times T^2$$

accesses between main and secondary memory for the $T \times T$ intermediate matrices.

Backward Pass: Computational Complexity I

- The computational cost of operations involved in the backward pass is as follows.

$$\frac{\partial L}{\partial V} = P^\top \frac{\partial L}{\partial O} : 2T^2d,$$

$$\frac{\partial L}{\partial P} = \frac{\partial L}{\partial O} V^\top : 2T^2d,$$

$$\frac{\partial L}{\partial S} = P \odot \left(\frac{\partial L}{\partial P} - \gamma \mathbf{1}^\top \right) : 2T^2,$$

Backward Pass: Computational Complexity II

$$\gamma = \text{rowsum} \left(\frac{\partial L}{\partial O} \odot O \right) : 2Td,$$

$$\frac{\partial L}{\partial Q} = \frac{\partial L}{\partial S} K : 2T^2d,$$

$$\frac{\partial L}{\partial K} = \left(\frac{\partial L}{\partial S} \right)^\top Q : 2T^2d.$$

- The total computational cost for the backward pass is

$$8T^2d + 2T^2 + 2Td.$$

Outline

- 1 Introduction
- 2 Standard Attention: Forward Calculation
- 3 Standard Attention: Backward Calculation
- 4 The Bottleneck on Memory Usage and Access
- 5 Flash Attention: Forward Calculation**
- 6 Flash Attention: Backward Calculation
- 7 Appendix

Situation When Storing $T \times d$ Matrices Is Possible I

- Now we assume that the main memory is sufficient to store

$T \times d$ matrices such as Q , K , and V .

- Let us develop strategies to reduce the number of memory accesses in (30).
- In particular, we would like to avoid loading and storing intermediate results.

Situation When Storing $T \times d$ Matrices Is Possible II

- 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.
- We can conduct the following procedure:

Algorithm 1: Row-wise forward pass

Load Q , K , and V to main memory.

Situation When Storing $T \times d$ Matrices Is Possible III

For $i = 1, \dots, T$, calculate

$$Q_{i,:}K^{\top} \in \mathbf{R}^{1 \times T} \quad (33)$$

$$\text{SoftMax}(Q_{i,:}K^{\top}) \in \mathbf{R}^{1 \times T} \quad (34)$$

$$\text{SoftMax}(Q_{i,:}K^{\top})V \in \mathbf{R}^{1 \times d} \quad (35)$$

and store the i th row of the output matrix to the secondary memory.

Situation When Storing $T \times d$ Matrices Is Possible IV

- By this way, the number of memory accesses is reduced to

$$O(Td).$$

because we never load/store any $T \times T$ matrices.

Situation When Storing $T \times d$ Matrices Is Not Possible I

- Unfortunately, our assumption that $T \times d$ matrices can be stored in main memory is often untrue.
- We discuss the situation of

$$d \leq M \leq Td,$$

where M is the size of the main memory.

- Then in (33)-(35), we must load the whole K and V once even for calculating just one output row.

Situation When Storing $T \times d$ Matrices Is Not Possible II

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

$$\text{SoftMax}(Q_{I,:}K^T)V, \quad (36)$$

where I is the block of rows that we intend to calculate.

- In calculating (36), we must access $|I|$ rows of Q , and the whole K and V (by row blocks).

Situation When Storing $T \times d$ Matrices Is Not Possible III

- We also need to store the intermediate blocks of

$$Q_{I,:}K^{\top}, \quad (37)$$

or

$$\text{SoftMax}(Q_{I,:}K^{\top}), \quad (38)$$

which requires

$$|I| \times T$$

Situation When Storing $T \times d$ Matrices Is Not Possible IV

space. If we assume (38) overwrites the space of (37), then the largest possible $|I|$ is

$$\frac{M}{T},$$

where M is the size of the main memory.

- Thus the total number of memory accesses is

$$O\left(\frac{T}{M/T}\right) \times O(Td) = O\left(\frac{T^3 d}{M}\right). \quad (39)$$

Situation When Storing $T \times d$ Matrices Is Not Possible V

- In the above discussion, we see that the main bottleneck is to store the intermediate matrix in (37).
- Because the number of rows in (37) is a large number T , $|I|$ must be small. Thus we get a large first term in (39).

FlashAttention I

- To reduce the number of memory accesses, let us see if we may avoid storing the intermediate matrix in (37).
- Assume that we split Q to the following row-block form:

$$\begin{bmatrix} Q_{I_1, :} \\ \vdots \\ Q_{I_{T_r}, :} \end{bmatrix} \quad (40)$$

with

$$|I_1| = \dots = |I_{T_r}|.$$

FlashAttention II

- For K, V , we respectively split them to

$$\begin{bmatrix} K_{J_1, :} \\ \vdots \\ K_{J_{T_c}, :} \end{bmatrix}, \begin{bmatrix} V_{J_1, :} \\ \vdots \\ V_{J_{T_c}, :} \end{bmatrix}. \quad (41)$$

- Similarly, we assume that $|J_1| = \dots = |J_{T_c}|$.
- We explain later why the split of Q should be different from that of K, V , even though they have the same size.
- In our discussion, we let I, J be any one of $|I_1|, \dots, |I_{T_r}|$ and $|J_1|, \dots, |J_{T_c}|$, respectively.

FlashAttention III

- We have

$$\begin{aligned} & \text{SoftMax}(Q_{I,:} [K_{J_1,:}^\top \cdots K_{J_{T_c},:}^\top]) \begin{bmatrix} V_{J_1,:} \\ \vdots \\ V_{J_{T_c},:} \end{bmatrix} \\ &= \text{SoftMax}([Q_{I,:}K_{J_1,:}^\top \cdots Q_{I,:}K_{J_{T_c},:}^\top]) \begin{bmatrix} V_{J_1,:} \\ \vdots \\ V_{J_{T_c},:} \end{bmatrix} \end{aligned}$$

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

$$(Q_{I,:}K_{J_1,:}^\top)V_{J_1,:} + \cdots + (Q_{I,:}K_{J_{T_c},:}^\top)V_{J_{T_c},:}$$

FlashAttention IV

- We have

$$\begin{aligned}
 (Q_{I,:} K_{J_1,:}^\top) V_{J_1,:} &\in \mathbf{R}^{|I| \times d}, \\
 &\vdots \\
 (Q_{I,:} K_{J_{T_c},:}^\top) V_{J_{T_c},:} &\in \mathbf{R}^{|I| \times d}.
 \end{aligned}$$

- If we sequentially generate each term, there is no need to store the intermediate sub-matrix in (37).

FlashAttention V

- In this situation, the algorithm can be as follows.

Algorithm 2: Block-wise forward pass (without SoftMax)

For $i = 1, \dots, T_r$
 For $j = 1, \dots, T_c$

$$O_{I_i,:} = O_{I_i,:} + (Q_{I_i,:} K_{J_j,:}^\top) V_{J_j,:} \quad (42)$$

Save $O_{I_i,:}$ to secondary memory.

FlashAttention VI

- At any time point we must store the following **four** blocks in main memory

- $Q_{I,:} \in \mathbf{R}^{|I| \times d}$

- $O_{I,:} \in \mathbf{R}^{|I| \times d}$

- $K_{J,:} \in \mathbf{R}^{|J| \times d}$ **or** $V_{J,:} \in \mathbf{R}^{|J| \times d}$

Note that we only need space for storing one of them. It can be overwritten once used.

- $Q_{I,:} K_{J,:}^{\top} \in \mathbf{R}^{|I| \times |J|}$

FlashAttention VII

- We assume that (42) does not require extra space to store the intermediate result $(Q_{I_i,:}K_{J_j,:}^\top)V_{J_j,:}$. This is possible as any resulting element or sub-matrix can be immediately used to update O .
- Therefore, we select $|I|$ and $|J|$ to satisfy

$$|I|d < \frac{M}{4}, \quad |J|d < \frac{M}{4}, \quad |I||J| < \frac{M}{4}. \quad (43)$$

- We also need that

$|I|$ as large as possible,

FlashAttention VIII

so K, V are less frequently loaded. The reason is that for any I , we must access the whole K, V .

- If we choose

$$|I| = \left\lfloor \frac{M}{4d} \right\rfloor \quad \text{and} \quad |J| = \min\left(\left\lfloor \frac{M}{4d} \right\rfloor, d\right) \quad (44)$$

then (43) holds.

FlashAttention IX

- The condition (44) indicates that we choose $|I|$ and $|J|$ differently. We consider the largest possible $|I|$ while having $|J|$ to satisfy (43). This situation explains why in (40) and (41), Q is split differently from K and V .
- In the end, the number of memory accesses is

$$\frac{T}{|I|} \times O(Td) = O\left(\frac{T}{M/d}\right) \times O(Td) = O\left(\frac{T^2 d^2}{M}\right). \quad (45)$$

- This value is smaller than the one we had earlier in (39).

FlashAttention X

- Unfortunately, we need the whole intermediate matrix in (37) because the SoftMax function involves all elements in each row.
- A crucial observation is that we hope to have

FlashAttention XI

$$\begin{aligned}
 & \text{SoftMax}([Q_{I,:}K_{J_1,:}^\top \cdots Q_{I,:}K_{J_j,:}^\top]) \begin{bmatrix} V_{J_1,:} \\ \vdots \\ V_{J_j,:} \end{bmatrix} = \\
 & \mathbf{v} \odot \left(\text{SoftMax}([Q_{I,:}K_{J_1,:}^\top \cdots Q_{I,:}K_{J_{j-1},:}^\top]) \begin{bmatrix} V_{J_1,:} \\ \vdots \\ V_{J_{j-1},:} \end{bmatrix} \right) \\
 & + \mathbf{w} \odot \left(\text{SoftMax}(Q_{I,:}K_{J_j,:}^\top) V_{J_j,:} \right), \tag{46}
 \end{aligned}$$

where \odot is the component-wise product and $\mathbf{v}, \mathbf{w} \in \mathbf{R}^{|I|}$ are two vectors. If \mathbf{v}, \mathbf{w} are easily

FlashAttention XII

available, then we can do an operation similar to (42).

- We discuss why w can be easily calculated. While w is a vector, we give an illustration to get one of its components.
- We have $\forall t \in J_1 \cup \dots \cup J_{j-1}$,

$$\frac{\exp(z_t)}{\sum_{s \in J_1 \cup \dots \cup J_j} \exp(z_s)}$$

$$= \underbrace{\left(\frac{\sum_{s \in J_1 \cup \dots \cup J_{j-1}} \exp(z_s)}{\sum_{s \in J_1 \cup \dots \cup J_j} \exp(z_s)} \right)}_v \frac{\exp(z_t)}{\sum_{s \in J_1 \cup \dots \cup J_{j-1}} \exp(z_s)},$$

FlashAttention XIII

and $\forall t \in J_j$,

$$\frac{\exp(z_t)}{\sum_{s \in J_1 \cup \dots \cup J_j} \exp(z_s)} = \underbrace{\left(\frac{\sum_{s \in J_j} \exp(z_s)}{\sum_{s \in J_1 \cup \dots \cup J_j} \exp(z_s)} \right)}_w \frac{\exp(z_t)}{\sum_{s \in J_j} \exp(z_s)}.$$

- Clearly, all we need is to maintain

$$\sum_{s \in J_1 \cup \dots \cup J_j} \exp(z_s). \quad (47)$$

FlashAttention XIV

- When handling j , we get

$$\sum_{s \in J_j} \exp(z_s),$$

so we can update (47) by

$$\begin{aligned} & \sum_{s \in J_1 \cup \dots \cup J_j} \exp(z_s) \\ = & \sum_{s \in J_1 \cup \dots \cup J_{j-1}} \exp(z_s) + \sum_{s \in J_j} \exp(z_s). \end{aligned}$$

FlashAttention XV

- This $O(|I|)$ cost for storing (47) in main memory is affordable.
- To have an algorithm, let's define

$$\Delta_I^{\text{old}} = \exp([Q_{I,:}K_{J_1,:}^\top \quad \cdots \quad Q_{I,:}K_{J_{j-1},:}^\top])'s \text{ row sum}$$

$$\Delta_I = \exp([Q_{I,:}K_{J_1,:}^\top \quad \cdots \quad Q_{I,:}K_{J_j,:}^\top])'s \text{ row sum}$$

$$\hat{\Delta}_I = \exp(Q_{I,:}K_{J_j,:}^\top)'s \text{ row sum}$$

FlashAttention XVI

- Then (46) becomes,

$$\begin{aligned}
 & \frac{\Delta_I^{\text{old}}}{\Delta_I} \odot \left(\text{SoftMax}([Q_{I,:}K_{J_1,:}^\top \cdots Q_{I,:}K_{J_{j-1},:}^\top]) \begin{bmatrix} V_{J_1,:} \\ \vdots \\ V_{J_{j-1},:} \end{bmatrix} \right) \\
 & + \frac{\hat{\Delta}_I}{\Delta_I} \odot \left(\text{SoftMax}(Q_{I,:}K_{J_j,:}^\top) V_{J_j,:} \right) \\
 & = \frac{\Delta_I^{\text{old}}}{\Delta_I} \odot \left(\text{SoftMax}([Q_{I,:}K_{J_1,:}^\top \cdots Q_{I,:}K_{J_{j-1},:}^\top]) \begin{bmatrix} V_{J_1,:} \\ \vdots \\ V_{J_{j-1},:} \end{bmatrix} \right) \\
 & + \frac{1}{\Delta_I} \odot \left(\exp(Q_{I,:}K_{J_j,:}^\top) V_{J_j,:} \right),
 \end{aligned}$$

FlashAttention XVII

where the fraction denotes element-wise division and $\mathbf{1} \in \mathbf{R}^{|I|}$ is the all-ones vector.

- Finally, we have the following algorithm.

Algorithm 3: A flash attention implementation (FlashAttention-2 in [Dao, 2024](#))

For $i = 1, \dots, T_r$

 Load Q_{I_i} :

FlashAttention XVIII

For $j = 1, \dots, T_c$

if $j > 1$

$$\Delta_{I_i}^{\text{old}} = \Delta_{I_i}$$

else

$$\Delta_{I_i}^{\text{old}} = \mathbf{0}$$

Load $K_{J_j,:}$

$$P_{I_i,J_j} = \exp(Q_{I_i,:} K_{J_j,:}^\top) \quad (48)$$

$$\Delta_{I_i} = \Delta_{I_i}^{\text{old}} + \text{rowsum}(P_{I_i,J_j}) \quad (49)$$

Load $V_{J_j,:}$ to overwrite $K_{J_j,:}$

FlashAttention XIX

$$O_{I_i,:} \leftarrow \frac{\Delta_{I_i}^{\text{old}}}{\Delta_{I_i}} \odot O_{I_i,:} + \frac{1}{\Delta_{I_i}} \odot (P_{I_i,J_j} V_{J_j,:})$$

Save $O_{I_i,:}$ and Δ_{I_i} to secondary memory.

- At first glance, Δ_I is a working array, so we do not need to keep it. However, we show later that the backward calculation also needs Δ . Thus we must save it to the secondary memory.
- For (48), we can calculate $Q_{I_i,:} K_{J_j,:}^\top$ first and calculate the exp value of each component. Thus, we only need space for one matrix.

FlashAttention XX

- By our trick in (46), the number of memory accesses is the same as the algorithm analyzed in (42) and (45),

$$O\left(\frac{T^2 d^2}{M}\right). \quad (50)$$

FlashAttention and FlashAttention-2 I

- FlashAttention was first introduced in [Dao et al. \(2022\)](#).
- Later [Dao \(2024\)](#) proposed an improved version FlashAttention-2.
- Interestingly, what we described earlier is FlashAttention-2.

FlashAttention and FlashAttention-2 II

- In the original FlashAttention, the algorithm is as follows.

Algorithm 4: A flash attention implementation (FlashAttention in Dao et al., 2022)

For $j = 1, \dots, T_c$
Load $V_{J_j,:}$ and $K_{J_j,:}$

FlashAttention and FlashAttention-2 III

For $i = 1, \dots, T_r$

Load $\Delta_{I_i}, Q_{I_i,:}$

$$P_{I_i, J_j} = \exp(Q_{I_i,:} K_{J_j,:}^\top)$$

if $j > 1$

$$\Delta_{I_i}^{\text{old}} = \Delta_{I_i}$$

else

$$\Delta_{I_i}^{\text{old}} = \mathbf{0}$$

$$\Delta_{I_i} = \Delta_{I_i}^{\text{old}} + \text{rowsum}(P_{I_i, J_j})$$

Load $O_{I_i,:}$ to overwrite $Q_{I_i,:}$

FlashAttention and FlashAttention-2 IV

$$O_{I_i,:} \leftarrow \frac{\Delta_{I_i}^{\text{old}}}{\Delta_{I_i}} \odot O_{I_i,:} + \frac{\mathbf{1}}{\Delta_{I_i}} \odot (P_{I_i,J_j} V_{J_j,:})$$

Save $O_{I_i,:}$, Δ_{I_i} to secondary memory

FlashAttention and FlashAttention-2 V

- A comparison with FlashAttention-2 shows that here i and j are swapped.
- Like FlashAttention-2, FlashAttention also maintains four matrices in the main memory
 - 1 $K_{J,:} \in \mathbf{R}^{|J| \times d}$
 - 2 $V_{J,:} \in \mathbf{R}^{|J| \times d}$
 - 3 $Q_{I,:} \in \mathbf{R}^{|I| \times d}$ or $O_{I,:} \in \mathbf{R}^{|I| \times d}$
 Note that we only need space for storing one of them. It can be overwritten once used.
 - 4 $Q_{I,:} K_{J,:}^\top \in \mathbf{R}^{|I| \times |J|}$
- In the inner loop, we now need to

FlashAttention and FlashAttention-2 VI

- load two matrices, and
- save one matrix and one vector, from/to secondary memory, respectively.
- However, the earlier version (i.e., FlashAttention-2) needs to load only two matrices.
- The main issue is that FlashAttention sequentially uses

$$O_{I_i,:}, \forall i$$

in the inner loop, but we cannot afford to store them. Thus, we must save $O_{I_i,:}$ back to secondary memory.

FlashAttention and FlashAttention-2 VII

- Another issue is that we must ensure that the save operation of $O_{I_i,:}$ is finished before the load operation in iteration $i + 1$. The reason is to free space for loading $Q_{I_{i+1},:}$ or P .
- The above discussion explains why FlashAttention-2 is more memory efficient than FlashAttention.

Outline

- 1 Introduction
- 2 Standard Attention: Forward Calculation
- 3 Standard Attention: Backward Calculation
- 4 The Bottleneck on Memory Usage and Access
- 5 Flash Attention: Forward Calculation
- 6 Flash Attention: Backward Calculation**
- 7 Appendix

Backward Pass: Situation When Storing $T \times d$ Matrices Is Possible I

- Now we assume that the main memory is sufficient to store

$T \times d$ matrices such as Q, K, V , and $\frac{\partial L}{\partial O}$.

- To avoid loading and storing $T \times T$ matrices, as in the forward pass, we devise a strategy to generate results in a row-wise setting.

Backward Pass: Situation When Storing $T \times d$ Matrices Is Possible II

- We can now conduct the following procedure:

Algorithm 5: Row-wise backward pass

Load

$$Q, K, V, O, \frac{\partial L}{\partial O} \in \mathbf{R}^{T \times d}$$

Backward Pass: Situation When Storing $T \times d$ Matrices Is Possible III

For $i = 1, \dots, T$

$$P_{i,:} = \text{SoftMax}(Q_{i,:}K^\top) \in \mathbf{R}^{1 \times T} \quad (51)$$

$$\frac{\partial L}{\partial V} \leftarrow \frac{\partial L}{\partial V} + P_{i,:}^\top \frac{\partial L}{\partial O_{i,:}} \in \mathbf{R}^{T \times d} \quad (52)$$

$$\frac{\partial L}{\partial P_{i,:}} = \frac{\partial L}{\partial O_{i,:}} V^\top \in \mathbf{R}^{1 \times T} \quad (53)$$

$$\gamma_i = \text{rowsum} \left(\frac{\partial L}{\partial O_{i,:}} \odot O_{i,:} \right) \in \mathbf{R} \quad (54)$$

Backward Pass: Situation When Storing $T \times d$ Matrices Is Possible IV

$$\frac{\partial L}{\partial S_{i,:}} = P_{i,:} \odot \left(\frac{\partial L}{\partial P_{i,:}} - \gamma_i \mathbf{1}^\top \right) \in \mathbf{R}^{1 \times T} \quad (55)$$

$$\frac{\partial L}{\partial Q_{i,:}} = \frac{\partial L}{\partial S_{i,:}} K \in \mathbf{R}^{1 \times d} \quad (56)$$

$$\frac{\partial L}{\partial K} \leftarrow \frac{\partial L}{\partial K} + \frac{\partial L}{\partial S_{i,:}}^\top Q_{i,:} \in \mathbf{R}^{T \times d} \quad (57)$$

Store the resulting gradient $\partial L / \partial Q$, $\partial L / \partial K$ and $\partial L / \partial V$ to the secondary memory.

Backward Pass: Situation When Storing $T \times d$ Matrices Is Possible V

- The total memory access cost is

$$O(Td),$$

since we never load/store any $T \times T$ matrices.

Backward Pass: Situation When Storing $T \times d$ Matrices Is Not Possible I

- We discuss the implementation of the backward pass under the situation of

$$d \leq M \leq Td,$$

where M is the size of the main memory.

- Similar to the forward calculation, we can no longer store $T \times d$ matrices such as K .

Backward Pass: Situation When Storing $T \times d$ Matrices Is Not Possible II

- Therefore in (51)–(56), for every row we must expensively load the whole K , V , $\partial L/\partial V$, and $\partial L/\partial K$.
- A possible strategy to reduce the memory access is to calculate $|I|$ rows together.

Backward Pass: Situation When Storing $T \times d$ Matrices Is Not Possible III

- The calculation for $|I|$ rows is as follows:

$$P_{I,:} = \text{SoftMax}(Q_{I,:}K^\top) \in \mathbf{R}^{|I| \times T}, \quad (58)$$

$$\frac{\partial L}{\partial V} \leftarrow \frac{\partial L}{\partial V} + P_{I,:}^\top \frac{\partial L}{\partial O_{I,:}} \in \mathbf{R}^{T \times d}, \quad (59)$$

$$\frac{\partial L}{\partial P_{I,:}} = \frac{\partial L}{\partial O_{I,:}} V^\top \in \mathbf{R}^{|I| \times T}, \quad (60)$$

$$\gamma_I = \text{rowsum} \left(\frac{\partial L}{\partial O_{I,:}} \odot O_{I,:} \right) \in \mathbf{R}^{|I|}, \quad (61)$$

Backward Pass: Situation When Storing $T \times d$ Matrices Is Not Possible IV

$$\frac{\partial L}{\partial S_{I,:}} = P_{I,:} \odot \left(\frac{\partial L}{\partial P_{I,:}} - \gamma_I \mathbf{1}^\top \right) \in \mathbf{R}^{|I| \times T}, \quad (62)$$

$$\frac{\partial L}{\partial Q_{I,:}} = \frac{\partial L}{\partial S_{I,:}} K \in \mathbf{R}^{|I| \times d}, \quad (63)$$

$$\frac{\partial L}{\partial K} \leftarrow \frac{\partial L}{\partial K} + \frac{\partial L}{\partial S_{I,:}}^\top Q_{I,:} \in \mathbf{R}^{T \times d}. \quad (64)$$

Backward Pass: Situation When Storing $T \times d$ Matrices Is Not Possible V

- In calculating (58)–(64), we must access

$$|I| \text{ rows of } Q, \frac{\partial L}{\partial O}, \text{ and } O, \quad (65)$$

and

$$\text{the whole } K, V, \frac{\partial L}{\partial V}, \text{ and } \frac{\partial L}{\partial K} \text{ (by row blocks)}. \quad (66)$$

Backward Pass: Situation When Storing $T \times d$ Matrices Is Not Possible VI

- We also need to store the intermediate blocks of

$$P_{I,:} = \text{SoftMax}(Q_{I,:}K^\top) \in \mathbf{R}^{|I| \times T}, \quad (67)$$

$$\frac{\partial L}{\partial P_{I,:}} = \frac{\partial L}{\partial O_{I,:}} V^\top \in \mathbf{R}^{|I| \times T}, \quad (68)$$

and

$$\frac{\partial L}{\partial S_{I,:}} = P_{I,:} \odot \left(\frac{\partial L}{\partial P_{I,:}} - \gamma_I \mathbf{1}^\top \right) \in \mathbf{R}^{|I| \times T}, \quad (69)$$

Backward Pass: Situation When Storing $T \times d$ Matrices Is Not Possible VII

which requires $2 \times |I| \times T$ space because (69) can overwrite either (67) or (68).

- As the memory access in (65) cost $O(|I|d)$ and we can control the block size in (66), the $2 \times |I| \times T$ for (67)–(68) is the bottleneck.

Backward Pass: Situation When Storing $T \times d$ Matrices Is Not Possible VIII

- Thus, the largest possible $|I|$ is

$$\frac{M}{2T},$$

and the total number of memory accesses is

$$O\left(\frac{T}{M/(2T)}\right) \times O(Td) = O\left(\frac{T^3d}{M}\right), \quad (70)$$

which is the same as (39) for the forward pass.

FlashAttention Backward Pass I

- To reduce the number of memory accesses in (70), we identify that the bottleneck is the storage of the intermediate matrices in (67) and (68).
- This bottleneck is similar to what we encountered in the forward pass, where we addressed it by splitting the $|I| \times T$ matrices into smaller sub-matrices and sequentially using them.
- We adopt the same strategy in the backward pass, and examine whether the calculations from (58) to (64) can be performed under this strategy.

FlashAttention Backward Pass II

- For Q , O and $\partial L/\partial O$, we respectively split them into

$$\begin{bmatrix} Q_{I_1,:} \\ \vdots \\ Q_{I_{T_r},:} \end{bmatrix}, \begin{bmatrix} O_{I_1,:} \\ \vdots \\ O_{I_{T_r},:} \end{bmatrix}, \begin{bmatrix} \frac{\partial L}{\partial O_{I_1,:}} \\ \vdots \\ \frac{\partial L}{\partial O_{I_{T_r},:}} \end{bmatrix},$$

with

$$|I_1| = \dots = |I_{T_r}|.$$

FlashAttention Backward Pass III

- For $K, V, \partial L/\partial K$ and $\partial L/\partial V$, we respectively split them into

$$\begin{bmatrix} K_{J_1, :} \\ \vdots \\ K_{J_{T_c}, :} \end{bmatrix}, \begin{bmatrix} V_{J_1, :} \\ \vdots \\ V_{J_{T_c}, :} \end{bmatrix}, \begin{bmatrix} \frac{\partial L}{\partial K_{J_1, :}} \\ \vdots \\ \frac{\partial L}{\partial K_{J_{T_c}, :}} \end{bmatrix}, \begin{bmatrix} \frac{\partial L}{\partial V_{J_1, :}} \\ \vdots \\ \frac{\partial L}{\partial V_{J_{T_c}, :}} \end{bmatrix},$$

with

$$|J_1| = \dots = |J_{T_c}|.$$

FlashAttention Backward Pass IV

- For $P_{I,:}$ in (58), since we already calculated and stored

$$\Delta = \text{rowsum}(\exp(QK^\top))$$

in (49), from (67) we can express $P_{I,:}$ as follows:

$$P_{I,:} = \text{Diag}(\Delta_I)^{-1} \exp(Q_{I,:} [K_{J_1,:}^\top \cdots K_{J_{T_c},:}^\top]), \quad (71)$$

where $\text{Diag}(\Delta_I)$ denotes the diagonal matrix with the elements of Δ_I on its diagonal.

FlashAttention Backward Pass V

- Thus, while $P_{I,:} \in \mathbf{R}^{|I| \times T}$ is too large to be stored, we can calculate each block $P_{I,J}$ sequentially

$$P_{I,J} = \text{Diag}(\Delta_I)^{-1} \exp(Q_{I,:} K_{J,:}^\top) \in \mathbf{R}^{|I| \times |J|}. \quad (72)$$

- Although this strategy works for (58), we must check that $P_{I,J_1}, \dots, P_{I,J_{T_c}}$ blocks can be sequentially used in the remaining calculation (59)–(64).

FlashAttention Backward Pass VI

- For $\partial L/\partial V$, from (59) and (71), we have

$$P_{I,:}^\top \frac{\partial L}{\partial O_{I,:}} = \begin{bmatrix} P_{I,J_1}^\top \\ \vdots \\ P_{I,J_{T_c}}^\top \end{bmatrix} \frac{\partial L}{\partial O_{I,:}} = \begin{bmatrix} P_{I,J_1}^\top \frac{\partial L}{\partial O_{I,:}} \\ \vdots \\ P_{I,J_{T_c}}^\top \frac{\partial L}{\partial O_{I,:}} \end{bmatrix}.$$

- If we split $\partial L/\partial V$ to row blocks corresponding to J_1, \dots, J_{T_c} , then (59) becomes to sequentially calculate

$$\left(\frac{\partial L}{\partial V} \right)_{J,:} \leftarrow \left(\frac{\partial L}{\partial V} \right)_{J,:} + P_{I,J}^\top \frac{\partial L}{\partial O_{I,:}}. \quad (73)$$

FlashAttention Backward Pass VII

- By this way, we need only $P_{I,J}$ each time instead of the whole $P_{I,:}$.
- Next, for $\partial L / \partial P_{I,:}$ in (60), we have

$$\begin{aligned} \frac{\partial L}{\partial P_{I,:}} &= \frac{\partial L}{\partial O_{I,:}} V^\top \\ &= \frac{\partial L}{\partial O_{I,:}} [V_{J_1,:}^\top \quad \cdots \quad V_{J_{T_c},:}^\top] \in \mathbf{R}^{|I| \times T}. \end{aligned}$$

FlashAttention Backward Pass VIII

- We cannot store an $|I| \times T$ matrix, but instead we calculate one block at a time:

$$\frac{\partial L}{\partial P_{I,J}} = \frac{\partial L}{\partial O_{I,:}} V_{J,:}^\top \in \mathbf{R}^{|I| \times |J|}. \quad (74)$$

- We then see where $\partial L / \partial P_{I,:}$ is used. It is in (62) for calculating $\partial L / \partial S_{I,:}$.

FlashAttention Backward Pass IX

- Recall that $\partial L / \partial S_{I,:} \in \mathbf{R}^{|I| \times T}$ is too large to be stored. From (72) and (74), each time we calculate one block as follows.

$$\begin{aligned} \frac{\partial L}{\partial S_{I,J}} &= P_{I,J} \odot \left(\frac{\partial L}{\partial P_{I,J}} - \gamma_I \mathbf{1}^\top \right) \\ &= P_{I,J} \odot \left(\frac{\partial L}{\partial O_{I,:}} V_{J,:}^\top - \gamma_I \mathbf{1}^\top \right) \in \mathbf{R}^{|I| \times |J|}. \end{aligned} \tag{75}$$

FlashAttention Backward Pass X

- From (75), there is no need to store $\partial L / \partial P_{I,J}$ as we can directly apply $\partial L / \partial O_{I,:}$ and $V_{J,:}^\top$ for the calculation.
- We also need to check how to compute γ_I . From (24), we know that γ_I can be computed with

$$\gamma_I = \text{rowsum} \left(\frac{\partial L}{\partial O_{I,:}} \odot O_{I,:} \right) \in \mathbf{R}^{|I|},$$

which can be calculated before we go through all $\partial L / \partial S_{I,J}$ blocks.

- Next, we check how to calculate $\partial L / \partial Q_{I,:}$ and $\partial L / \partial K_{J,:}$.

FlashAttention Backward Pass XI

- From (63), using (75) and $K_{J_j, \cdot}$, we have

$$\begin{aligned}
 \frac{\partial L}{\partial Q_{I, \cdot}} &= \frac{\partial L}{\partial S_{I, \cdot}} K \\
 &= \left[\frac{\partial L}{\partial S_{I, J_1}} \quad \cdots \quad \frac{\partial L}{\partial S_{I, J_{T_c}}} \right] \begin{bmatrix} K_{J_1, \cdot} \\ \vdots \\ K_{J_{T_c}, \cdot} \end{bmatrix} \\
 &= \sum_{j=1}^{T_c} \frac{\partial L}{\partial S_{I, J_j}} K_{J_j, \cdot}
 \end{aligned} \tag{76}$$

FlashAttention Backward Pass XII

- By this setting, we no longer need the whole $\partial L / \partial S_{I,:}$. Instead, we sequentially calculate $\partial L / \partial S_{I,J_j}$ via (75) and add

$$\frac{\partial L}{\partial S_{I,J_j}} K_{J_j,:} \quad \text{to} \quad \frac{\partial L}{\partial Q_{I,:}}.$$

- For $\partial L / \partial K$, the situation is the same as that for $\partial L / \partial V$ due to the similar form of (59) and (64).

FlashAttention Backward Pass XIII

- Thus, what we do is an update like (73):

$$\frac{\partial L}{\partial K_{J,:}} \leftarrow \frac{\partial L}{\partial K_{J,:}} + \frac{\partial L}{\partial S_{I,J}}{}^\top Q_{I,:}. \quad (77)$$

- In (72)–(77), $O_{I,:}$ is only required for calculating γ_I . Because γ_I is independent of operations on blocks J_1, \dots, J_{T_c} , and can be pre-computed, once we have used $O_{I,:}$, the same space can be available for $Q_{I,:}$.

FlashAttention Backward Pass XIV

- Finally, we have the following algorithm.

Algorithm 6: A flash attention implementation for the backward pass (i, j version)

For $i = 1, \dots, T_r$

Load $O_{I_i,:}$, $\partial L / \partial O_{I_i,:}$, and Δ_{I_i}

Compute

$$\gamma_{I_i} = \text{rowsum} \left(\frac{\partial L}{\partial O_{I_i,:}} \odot O_{I_i,:} \right)$$

Load $Q_{I_i,:}$ to overwrite $O_{I_i,:}$

FlashAttention Backward Pass XV

For $j = 1, \dots, T_c$

Load $K_{J_j,:}$ and $V_{J_j,:}$

$$P_{I_i,J_j} = \text{Diag}(\Delta_{I_i})^{-1} \exp(Q_{I_i,:} K_{J_j,:}^\top)$$

$$\frac{\partial L}{\partial V_{J_j,:}} \leftarrow \frac{\partial L}{\partial V_{J_j,:}} + P_{I_i,J_j}^\top \frac{\partial L}{\partial O_{I_i,:}}$$

$$\frac{\partial L}{\partial S_{I_i,J_j}} = P_{I_i,J_j} \odot \left(\frac{\partial L}{\partial O_{I_i,:}} V_{J_j,:}^\top - \gamma_{I_i} \mathbf{1}^\top \right)$$

$$\frac{\partial L}{\partial K_{J_j,:}} \leftarrow \frac{\partial L}{\partial K_{J_j,:}} + \frac{\partial L}{\partial S_{I_i,J_j}}^\top Q_{I_i,:}$$

FlashAttention Backward Pass XVI

$$\frac{\partial L}{\partial Q_{I_i,:}} \leftarrow \frac{\partial L}{\partial Q_{I_i,:}} + \frac{\partial L}{\partial S_{I_i,J_j}} K_{J_j,:}$$

Save $\partial L / \partial V_{J_j,:}$ and $\partial L / \partial K_{J_j,:}$ to secondary memory.

Save $\partial L / \partial Q_{I_i,:}$ to secondary memory.

- At any time point of the inner loop, we must store the following blocks in main memory:
 - $Q_{I_i,:} \in \mathbf{R}^{|I| \times d}$
 - $K_{J_j,:} \in \mathbf{R}^{|J| \times d}$
 - $V_{J_j,:} \in \mathbf{R}^{|J| \times d}$

FlashAttention Backward Pass XVII

- 4 $P_{I,J} \in \mathbf{R}^{|I| \times |J|}$ or $\partial L / \partial S_{I,J} \in \mathbf{R}^{|I| \times |J|}$
- 5 $\partial L / \partial O_{I,:} \in \mathbf{R}^{|I| \times d}$
- 6 $\partial L / \partial Q_{I,:} \in \mathbf{R}^{|I| \times d}$
- 7 $\partial L / \partial K_{J,:} \in \mathbf{R}^{|J| \times d}$
- 8 $\partial L / \partial V_{J,:} \in \mathbf{R}^{|J| \times d}$

- Similar to the forward pass, we assume that operations in (72)–(77) do not require extra space to store intermediate results, such as

$$P_{I,J} \frac{\partial L}{\partial O_{I,:}}, \frac{\partial L}{\partial O_{I,:}} V_J \text{ and } \left(\frac{\partial L}{\partial S_{I,J}} \right)^\top Q_{I,:},$$

FlashAttention Backward Pass XVIII

as any results can be immediately used to update the matrix on the left-hand side.

- Therefore, we select $|I|$ and $|J|$ to satisfy

$$|I|d < \frac{M}{8}, \quad |J|d < \frac{M}{8}, \quad |I||J| < \frac{M}{8}. \quad (78)$$

- If we choose

$$|I| = \left\lfloor \frac{M}{8d} \right\rfloor \quad \text{and} \quad |J| = \min \left(\left\lfloor \frac{M}{8d} \right\rfloor, d \right), \quad (79)$$

then (78) holds.

FlashAttention Backward Pass XIX

- Similar to the forward calculation, under each block I , we must access several $T \times d$ matrices. Thus the number of memory accesses is still

$$\frac{T}{|I|} \times O(Td) = O\left(\frac{T^2 d^2}{M}\right)$$

as in (50).

FlashAttention Backward Pass: (i, j) Versus (j, i) I

- What we have derived so far is not yet the FlashAttention backward pass. Recall that in forward FlashAttention, we have a different version by swapping the loop order of i and j . We referred to the i, j version as FlashAttention-2, while the j, i version is referred to as FlashAttention.
- We showed that the i, j version is slightly more memory efficient.
- Thus, we would like to check the j, i version of the backward implementation.

FlashAttention Backward Pass: (i, j) Versus (j, i) II

- The procedure is as follows.

Algorithm 7: A draft of flash attention backward pass

For $j = 1, \dots, T_c$
 Load $K_{J_j,:}$ and $V_{J_j,:}$

FlashAttention Backward Pass: (i, j) Versus (j, i) III

For $i = 1, \dots, T_r$

$$\begin{aligned} &\text{Load } O_{I_i,:}, \frac{\partial L}{\partial O_{I_i,:}}, \text{ and } \Delta_{I_i} \\ &\gamma_{I_i} = \text{rowsum} \left(\frac{\partial L}{\partial O_{I_i,:}} \odot O_{I_i,:} \right) \quad (80) \\ &\dots (\text{rest omitted for now}) \end{aligned}$$

FlashAttention Backward Pass: (i, j) Versus (j, i) IV

- In (80), the same γ_{I_i} is computed T_c times. As it is independent of J blocks, we can pre-compute it before the main loop. By this way, we can also prevent the frequent loading of O_{I_i} .
- The new procedure is as follows.

FlashAttention Backward Pass: (i, j) Versus (j, i) V

Algorithm 8: A flash attention implementation for the backward pass (j, i version)

For $i = 1, \dots, T_r$
 Load $\partial L / \partial O_{I_i, :}$, $O_{I_i, :}$
 Calculate and save

$$\gamma_{I_i} = \text{rowsum} \left(\frac{\partial L}{\partial O_{I_i, :}} \odot O_{I_i, :} \right)$$

FlashAttention Backward Pass: (i, j) Versus (j, i) VI

For $j = 1, \dots, T_c$

Load $K_{J_j,:}$ and $V_{J_j,:}$

For $i = 1, \dots, T_r$

Load $Q_{I_i,:}$, $\frac{\partial L}{\partial O_{I_i,:}}$, Δ_{I_i} and γ_{I_i}

$$P_{I_i, J_j} = \text{Diag}(\Delta_{I_i})^{-1} \exp(Q_{I_i,:} K_{J_j,:}^\top)$$

$$\frac{\partial L}{\partial V_{J_j,:}} \leftarrow \frac{\partial L}{\partial V_{J_j,:}} + P_{I_i, J_j}^\top \frac{\partial L}{\partial O_{I_i,:}}$$

FlashAttention Backward Pass: (i, j) Versus (j, i) VII

$$\frac{\partial L}{\partial S_{I_i, J_j}} = P_{I_i, J_j} \odot \left(\frac{\partial L}{\partial O_{I_i, :}} V_{J_j, :}^\top - \gamma_{I_i} \mathbf{1}^\top \right)$$

$$\frac{\partial L}{\partial K_{J_j, :}} \leftarrow \frac{\partial L}{\partial K_{J_j, :}} + \frac{\partial L}{\partial S_{I_i, J_j}}^\top Q_{I_i, :}$$

$$\frac{\partial L}{\partial Q_{I_i, :}} \leftarrow \frac{\partial L}{\partial Q_{I_i, :}} + \frac{\partial L}{\partial S_{I_i, J_j}} K_{J_j, :}$$

Save $\partial L / \partial Q_{I_i, :}$ to secondary memory

Save $\partial L / \partial V_{J_j, :}$ and $\partial L / \partial K_{J_j, :}$ to secondary memory.

FlashAttention Backward Pass: (i, j) Versus (j, i) VIII

- The above procedure maintains the following eight blocks in main memory:

- $Q_{I,:} \in \mathbf{R}^{|I| \times d}$
- $K_{J,:} \in \mathbf{R}^{|J| \times d}$
- $V_{J,:} \in \mathbf{R}^{|J| \times d}$
- $P_{I,J} \in \mathbf{R}^{|I| \times |J|}$ or $\partial L / \partial S_{I,J} \in \mathbf{R}^{|I| \times |J|}$
- $\partial L / \partial O_{I,:} \in \mathbf{R}^{|I| \times d}$
- $\partial L / \partial Q_{I,:} \in \mathbf{R}^{|I| \times d}$
- $\partial L / \partial K_{J,:} \in \mathbf{R}^{|J| \times d}$
- $\partial L / \partial V_{J,:} \in \mathbf{R}^{|J| \times d}$

FlashAttention Backward Pass: (i, j) Versus (j, i) IX

- The number of blocks is the same as that of the i, j version discussed earlier.
- Thus they have the same memory consumption.
- However, the two versions differ in the number of memory accesses. Let us check their respective inner loops.
- The i, j version requires to
 - load $K_{J_j,:} \in \mathbf{R}^{|J| \times d}$, $V_{J_j,:} \in \mathbf{R}^{|J| \times d}$ and

FlashAttention Backward Pass: (i, j) Versus (j, i) X

- save $\partial L / \partial K_{J_j,:} \in \mathbf{R}^{|J| \times d}$ and $\partial L / \partial V_{J_j,:} \in \mathbf{R}^{|J| \times d}$.
- The j, i version requires to
 - load $Q_{I_i,:} \in \mathbf{R}^{|I| \times d}$, $\partial L / \partial O_{I_i,:} \in \mathbf{R}^{|I| \times d}$ and
 - save $\partial L / \partial Q_{I_i,:} \in \mathbf{R}^{|I| \times d}$.
- Thus, their number of memory accesses is

$$4 \times T_r \times |J| \times d$$

and

$$3 \times T_c \times |I| \times d$$

FlashAttention Backward Pass: (i, j) Versus (j, i) XI

- If we consider

$$T_r \approx T_c \quad \text{and} \quad |I| \approx |J|,$$

then the j, i version is more memory efficient.

- Therefore, it is interesting that for the forward calculation, the i, j version is better but the opposite may occur for the backward calculation.

FlashAttention Backward Pass: (i, j) Versus (j, i) XII

- Another thing worth noticing is that since the j, i version of backward implementation also maintains eight blocks, the block sizes $|I|$ and $|J|$ should also satisfy (79).
- However, the setting in Dao et al. (2022) is

$$|I| = \left\lfloor \frac{M}{4d} \right\rfloor \quad \text{and} \quad |J| = \min \left(\left\lfloor \frac{M}{4d} \right\rfloor, d \right),$$

which we think is incorrect.

Outline

- 1 Introduction
- 2 Standard Attention: Forward Calculation
- 3 Standard Attention: Backward Calculation
- 4 The Bottleneck on Memory Usage and Access
- 5 Flash Attention: Forward Calculation
- 6 Flash Attention: Backward Calculation
- 7 Appendix**

Gradient with Respect to W_O and O

- In (6)–(10), we gave details of multi-head attention, in which

input: $Q, K, V \rightarrow$ output: O

is a sub-process.

- Here we give full details of gradient calculation by considering

input: $\tilde{Z} \rightarrow$ output: \tilde{Z}^{out} .

Gradient with Respect to W_O and O II

- From (10), we derive

$$\frac{\partial L}{\partial W_O} \in \mathbf{R}^{d \times d} \quad \text{and} \quad \frac{\partial L}{\partial O} \in \mathbf{R}^{T \times d}.$$

- We note that (10) is in a similar form to (9), where they are respectively

$$\tilde{Z}^{\text{out}} = OW_O \quad \text{and} \quad O = PV.$$

Gradient with Respect to W_O and O III

- The results derived earlier for

$$\frac{\partial L}{\partial V} \quad \text{and} \quad \frac{\partial L}{\partial P}$$

directly lead us to have

$$\begin{aligned} \frac{\partial L}{\partial W_O} &= O^\top \frac{\partial L}{\partial \tilde{Z}^{\text{out}}} \in \mathbf{R}^{d \times d}, \\ \frac{\partial L}{\partial O} &= \frac{\partial L}{\partial \tilde{Z}^{\text{out}}} W_O^\top \in \mathbf{R}^{T \times d}. \end{aligned}$$

Gradients with Respect to W_Q , W_K , and W_V I

- From (6), next we derive

$$\frac{\partial L}{\partial W_Q}, \frac{\partial L}{\partial W_K}, \frac{\partial L}{\partial W_V} \in \mathbf{R}^{d \times d}.$$

We note that $Q = \tilde{Z}W_Q$, $K = \tilde{Z}W_K$, and $V = \tilde{Z}W_V$ are all in a similar form to (9).

Gradients with Respect to W_Q , W_K , and W_V II

- Thus we directly have

$$\begin{aligned}\frac{\partial L}{\partial W_Q} &= \tilde{Z}^\top \frac{\partial L}{\partial Q} \in \mathbf{R}^{d \times d}, \\ \frac{\partial L}{\partial W_K} &= \tilde{Z}^\top \frac{\partial L}{\partial K} \in \mathbf{R}^{d \times d}, \\ \frac{\partial L}{\partial W_V} &= \tilde{Z}^\top \frac{\partial L}{\partial V} \in \mathbf{R}^{d \times d}.\end{aligned}$$

Gradient with Respect to \tilde{Z} I

- From (6),

$$Q_{ik} = \sum_{\ell=1}^d \tilde{Z}_{i\ell} (W_Q)_{\ell k}, \quad (81)$$

$$K_{ik} = \sum_{\ell=1}^d \tilde{Z}_{i\ell} (W_K)_{\ell k}, \quad (82)$$

$$V_{ik} = \sum_{\ell=1}^d \tilde{Z}_{i\ell} (W_V)_{\ell k}. \quad (83)$$

Gradient with Respect to \tilde{Z} II

- From Figure 1, \tilde{Z} affects the loss L through elements in Q , K , and V .
- More specifically, from (81)–(83) and the similar form of (13) in calculating $\partial L / \partial P$, we see \tilde{Z}_{il} affects L through the i th row of Q , K , and V .

Gradient with Respect to \tilde{Z} III

- Thus, applying the chain rule gives

$$\begin{aligned}\frac{\partial L}{\partial \tilde{Z}_{il}} &= \sum_{k=1}^d \frac{\partial L}{\partial Q_{ik}} \frac{\partial Q_{ik}}{\partial \tilde{Z}_{il}} + \sum_{k=1}^d \frac{\partial L}{\partial K_{ik}} \frac{\partial K_{ik}}{\partial \tilde{Z}_{il}} \\ &\quad + \sum_{k=1}^d \frac{\partial L}{\partial V_{ik}} \frac{\partial V_{ik}}{\partial \tilde{Z}_{il}}\end{aligned}$$

Gradient with Respect to \tilde{Z} IV

$$\begin{aligned}
 &= \sum_{k=1}^d \frac{\partial L}{\partial Q_{ik}} (W_Q)_{lk} + \sum_{k=1}^d \frac{\partial L}{\partial K_{ik}} (W_K)_{lk} \\
 &\quad + \sum_{k=1}^d \frac{\partial L}{\partial V_{ik}} (W_V)_{lk}. \tag{84}
 \end{aligned}$$

- We can represent (84) in the following matrix form

$$\frac{\partial L}{\partial \tilde{Z}} = \frac{\partial L}{\partial Q} W_Q^\top + \frac{\partial L}{\partial K} W_K^\top + \frac{\partial L}{\partial V} W_V^\top \in \mathbf{R}^{T \times d}.$$

References I

- T. Dao. Flashattention-2: Faster attention with better parallelism and work partitioning. In *Proceedings of The Twelfth International Conference on Learning Representations (ICLR)*, 2024.
- 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.
- A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems 30*, pages 5998–6008, 2017.