Flash Attention

Complexity of dot-production attention

- Original:
  - Memory $O(N^2)$
  - Computation complexity: $O(N^2)$ → Quadratic

Difficulty in increasing the context length

- Approximate attention methods: $O(N)$ → Linear [Please REFER TO Sparse ATTN]

But often they are not able to achieve wall-clock speedup:
∵ Reducing FLOP operations, at the cost of memory access overhead

key idea: Avoid reading and writing the full attention matrix to and from GPU's DRAM

- Using a tiling mechanism
- I/O-aware. Significantly fewer memory access
- Exact attention (not approximate)
- Wall-clock speedup at training.

Challenges:

Computing the softmax step without accessing the whole QK^T

Solution - Tiling mechanism. gradually computation

Challenges:

Backward-pass (gradient) w/o storing the attention matrix

Solution - Only store the normalization factor of softmax
- Recomputing attention on-chip (Faster than reading the attention matrix from memory)

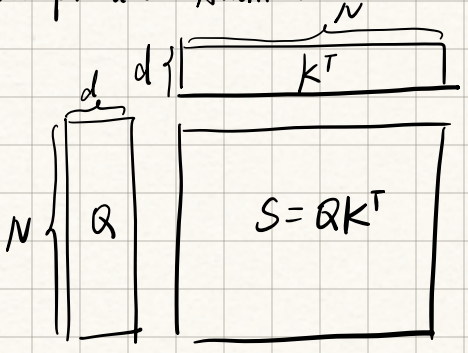
Performance of computing Function $y = F(x)$

- Time for memory access: T_{mem}
- Time for mathematical computations: T_{math}

→ Total time: $T_f = \max(T_{mem}, T_{math})$

if $T_{math} > T_{mem}$ → compute-bound
if $T_{math} < T_{mem}$ → memory-bound

Dot-product Attention:



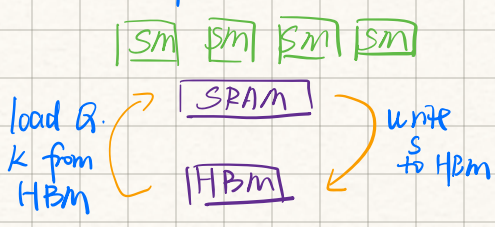
torch.matmul(Q, K^T)

GPU Execution for Attention

```
S = torch.matmul(Q, k.transpose(-2, -1))
P = torch.softmax(S, dim=-1)
output = torch.matmul(P, V)
```

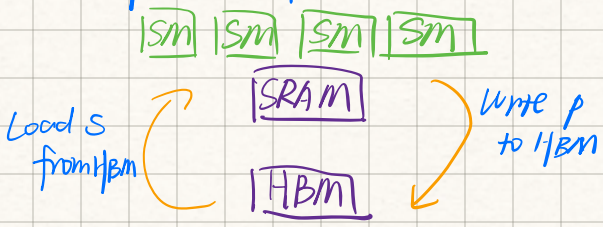
Step1:

Compute $S=QK^T$



Step2:

Compute $P=softmax(S)$



$3 \times [n \times n]$!!

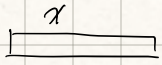
Flash Attention Algorithm

- Tiling (for the forward pass)
  Loading blocks of Q, K, V and compute the output partially
- Re-computation (for the backward pass)
  Using the stored normalization factors for each row

Reformulating Softmax Function

$$softmax(x)_i = \frac{exp(x_i)}{\sum_k exp(x_k)} \times \frac{exp(-m)}{exp(-m)} \Rightarrow \frac{exp(x_i-m)}{\sum_k exp(x_k-m)}$$

To numerical stability, choose $m=max(x)$, ensure all $x_i - m \le 0$, $\therefore e^{x_i-m} \le 1$, prevent overflow



Re-formulating softmax:

$\vec{f}(x) = [e^{x_1-m}, e^{x_2-m}, \dots, e^{x_N-m}]$

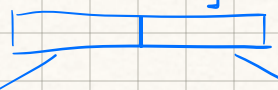
$l(x) = \sum_j f(x)_j$

$$softmax(x) = \frac{\vec{f}(x)}{l(x)}$$

So, how to compute softmax of entire vector x?

Solve:

$\vec{x} = [\vec{x}^{1:3}, \vec{x}^{1:2:3}]$



Block 1

$m_1 = max(\vec{x}^{1:3})$

$\vec{f}_1 = e^{\vec{x}^{1:3} - m_1}$

$l_1 = \sum_j e^{x_j^{1:3} - m_1}$

Block 2

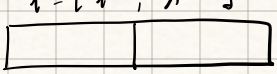
$m_2 = max(\vec{x}^{1:2:3})$

$\vec{f}_2 = e^{\vec{x}^{1:2:3} - m_2}$

$l_2 = \sum_j e^{x_j^{1:2:3} - m_2}$

Softmax of TWO concatenated vectors

$\vec{x} = [\vec{x}^{1:3}, \vec{x}^{1:2:3}]$



Block 1 $\vec{x}^{1:3}$:

$m_1 = max(\vec{x}^{1:3})$

$l_1 = \sum_j e^{x_j - m_1}$

$\vec{f}_1 = e^{\vec{x}^{1:3} - m_1}$

Block 2 $\vec{x}^{1:2:3}$:

$m_2 = max(\vec{x}^{1:2:3})$

$l_2 = \sum_j e^{x_j - m_2}$

$\vec{f}_2 = e^{\vec{x}^{1:2:3} - m_2}$

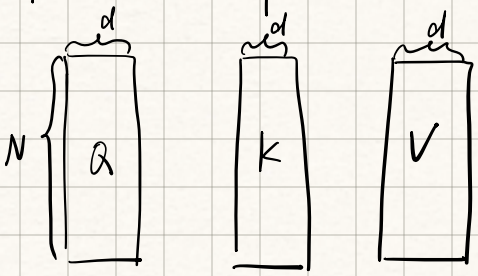
Overall max: $m(x) = max(m_1, m_2)$

Adjusted $\vec{f}_1$ and $\vec{f}_2$: $\vec{f}(x) = [e^{m_1 - m(x)} f_1, e^{m_2 - m(x)} f_2]$

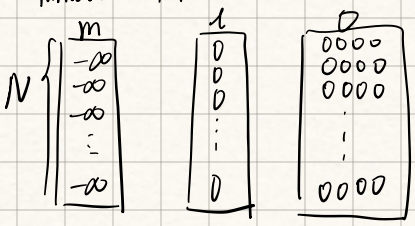
Adjusted normalization factor: $l(x) = e^{m_1 - m(x)} l_1 + e^{m_2 - m(x)} l_2$

$$\Rightarrow softmax(\vec{x}) = \frac{\vec{f}(x)}{l(x)}$$

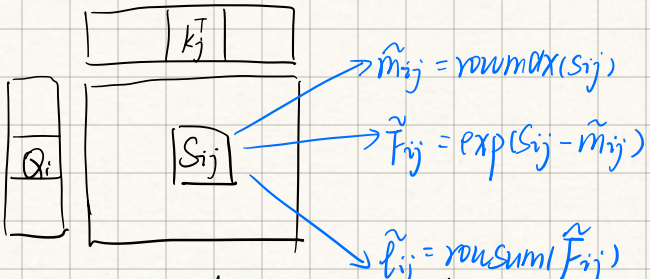
FlashAttention - Input and Initialization

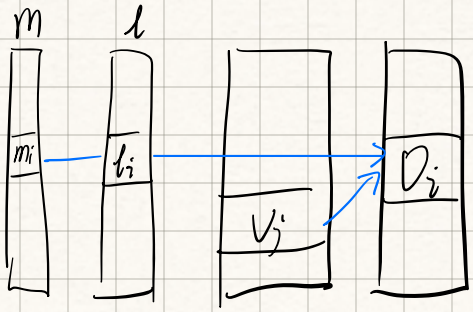


initialize intermedia vectors and the outputs:



FlashAttention – Partial Updates



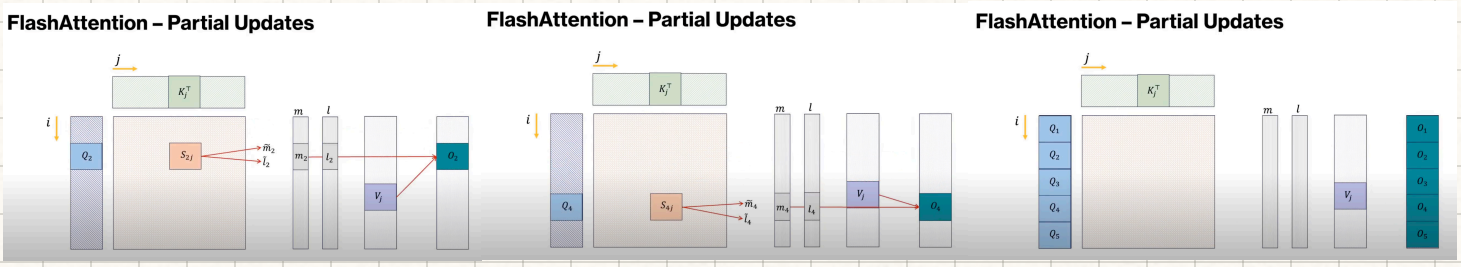


Partial update to block matrix O_i:

$$O_i^{new} = \text{diag}(l_i^{new})^{-1}\left(\text{diag}(l_i)\, e^{m_i - m_i^{new}} O_i + e^{\tilde{m}_{ij} - m_i^{new}} \tilde{F}_{ij} V_j\right)$$

($O_i = \text{softmax}(Q_i K_j^T) V$, $S_{ij} = Q_i K_j^T$, $\tilde{m}_{ij} = \text{rowmax}(S_{ij})$

$\tilde{F}_{ij} = e^{S_{ij} - \tilde{m}_{ij}}$)



HBM Access Comparison

Naive Algorithm $O(Nd + N^2)$ >> Flass Attn $O(N^2 d^2 M^{-1})$

M: Size of SRAM

=> Flash Attn reduces number of HBM access

High Bandwidth Memory (HBM): External GPU memory — large capacity but slow access (large but slow)

Static RAM (SRAM) / on-chip Memory: Fast on-chip cache (shared memory) located inside the GPU core, but small (small but fast)

N: sequence length eg 1024 tokens

d: Hidden dimension

Reference: Machine Learning Studio - Flash Attn: Accelerate LLM training