$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

$$\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

more I/O bound on HBM.

GPU

HBM (Global)

shared

alt : compute attention in shared memory (which is smaller)

$\Downarrow$

Blocked computation.

- numerical instability of softmax due to 'e^x'

Safe softmax

$$\frac{e^{x_i}}{\sum_{j=1}^{N} e^{x_j}} = \frac{e^{x_i - k}}{\sum_{j=1}^{N} e^{x_j - k}}$$

"Sneak in" some constant.

$$k = \max_i (x_i)$$

Pseudocode

$m_0 = -\infty$

for $i = 1$ to N
$\quad m_i = \max(m_{i-1}, x_i)$ → ①

$l_0 = 0$
for $j = 1$ to N
$\quad l_j = l_{j-1} + e^{x_j - m_N}$ → ②

for $k = 1$ to N
$\quad x_k \leftarrow \dfrac{e^{x_k - m_N}}{l_N}$ → ③

① accessing vector three times.

(Three sequential operations)

for given query fill its row in this way.

→ To fuse the computation ①, and ②, use local maxima (m_i) instead of global maxima (m_N) and fix it on fly using correction factor.

new pseudocode

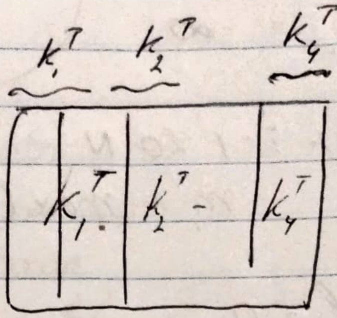
Online softmax

$m_0 = -\infty, \; l_0 = 0$
for $i = 1$ to N
$\quad m_i = \max(m_{i-1}, x_i)$
$\quad l_i = l_{i-1} \cdot e^{m_{i-1} - m_i} + e^{x_i - m_i}$
for $k = 1$ to N
$\quad x_k \leftarrow \dfrac{e^{x_k - m_N}}{l_N}$

Proof by Induction:
① Prove it holds for $N = 1$
② Given it holds for N, does it hold for $N+1$?
~~sized vector~~

Block Matrix Multiplication

×



Q $\qquad\qquad$ K^T

Original (8, 128) $\qquad$ Original (128, 8)

$\downarrow$ to $\qquad\qquad$ $\downarrow$ to

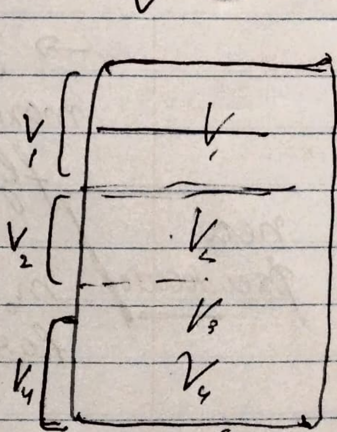
4 blocks of size (2, 128) $\qquad$ 4 blocks of size (128, 2)

$\Downarrow$ $\qquad\qquad$ $\Downarrow$

(4, 128) $\qquad\qquad$ (128, 4)

✓

$$S = \begin{bmatrix} Q_1K_1^T & Q_1K_2^T & \cdots & Q_1K_4^T \\ Q_2K_1^T & \vdots & & \vdots \\ \vdots & \vdots & & \vdots \\ \cdots & \cdots & \cdots & \boxed{Q_4K_4^T} \end{bmatrix}$$



org: (8, 128)
block: (4, 128)

not scalars but 2x2 matrix.

Shape (8, 8)

each block is made up of two rows.

(S.V)

$$\text{Output} = \begin{bmatrix} \underbrace{(Q_1K_1^T)}_{S_{11}}V_1 + \underbrace{(Q_1K_2^T)}_{S_{12}}V_2 + \underbrace{(Q_1K_3^T)}_{S_{13}}V_3 + \underbrace{(Q_1K_4^T)}_{S_{14}}V_4 \\ \vdots \end{bmatrix}$$

Pseudocode

For each block Q_i

O_i = zeroes(2, 128)

For each block k_j

$O_i \leftarrow O_i + (Q_i k_j^T) V_j$

End For

End For

But with (normalization) softmax, (For block)

Initialization

$$m_0 = \begin{bmatrix} -\infty \\ -\infty \end{bmatrix}, \quad l_0 = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$O_0 = \begin{bmatrix} 0 & 0 & 0 & - & - & 0 \\ 0 & 0 & 0 & - & - & 0 \end{bmatrix}$$ 2x128 matrix

Step 1

$$m_1 = \max(\mathrm{rowmax}(\overset{S_{11}}{Q_1 k_1^T}), m_0)$$

$$S_1 = Q_1 k_1^T$$

$$l_1 = \mathrm{rowsum}\left[\exp(S_1 - m_1)\right] + l_0 \cdot \exp(m_0 - m_1)$$

$$P_{11} = \exp(S_1 - m_1)$$

$$O_1 = \mathrm{diag}(\exp(m_0 - m_1)) O_0 + P_{11} V_1$$

Step 2

$$m_2 = \max(\text{rowmax}(\overbrace{Q_1 K_2^T}^{S_{12}}),\ m_1)$$

$$S_2 = Q_1 K_2^T$$

$$l_2 = \text{rowsum}\left[\exp(S_2 - m_2)\right] + l_1 \cdot \exp(m_1 - m_2)$$

$$P_{12} = \exp(S_2 - m_2)$$

$$O_1 = \text{diag}(\exp(m_1 - m_2)) \cdot O_1 + P_{12} V_2$$

$$\begin{bmatrix} \exp(m_1 - m_2)_1 & 0 \\ 0 & \exp(m_1 - m_2)_2 \end{bmatrix} \times \begin{bmatrix} O_{11} & O_{12} & O_{13} & \text{---} \\ O_{21} & O_{22} & O_{23} & \text{---} \end{bmatrix}$$

$$= \begin{bmatrix} O_{11}\exp(m_1 - m_2)_1 & \text{---} \\ O_{21}\exp(m_1 - m_2)_2 & \text{---} \end{bmatrix}$$

(correcting each row from previous block row.)

And so on until the last step. Then at the end we apply "l" normalization factor.

$$O_f = [\text{diag}(l_4)]^{-1} O_4$$

Because $\left(\begin{bmatrix} l_4^{(1)} & 0 \\ 0 & l_4^{(2)} \end{bmatrix}\right)^{-1} = \begin{bmatrix} \frac{1}{l_4^{(1)}} & 0 \\ 0 & \frac{1}{l_4^{(2)}} \end{bmatrix}$

GPU : Hardware
CUDA : Software for GPU

Cuda Programming Example : "Vector Addition"

Allocate memory in cuda and copy A and B to those from the host m/c.
Execute the kernel => launches N threads (N Parallel operations).

Each thread get its ID (0 — N-1) and we're supposed to define what we want to run in each.

(launches near two-multiple
Ex : 34 -> 64)

control units are expensive part

Group of threads run (by) same instruction, but data (may) differ. (same control unit)
>> SINGLE INSTRUCTION MULTIPLE DATA
>> " " " " " " THREADS

Blocks of threads. we'll have $\lceil \frac{N}{32} \rceil$ blocks.

no. of threads in each block (or) Blocksize

element_id = B_id × Block_size + T_id

We are specifying this, not cuda.

- Access to DRAM is slow.

  but access to shared memory is very fast.

CPU → GPU (global mem which is slower) → use shared memory, copy to it whenever needed

(shared by all threads in the same block)

⇒ Each kernel works with one "Query" block and iterates through "key" blocks

⇒ Each "head" computes attention independently

⇒ Each Q blocks independently.

max parallel programs we can run. → BATCH_SIZE × NUM_HEADS × (~~BLOCK_SIZE_Q~~ SEQ_LEN / BLOCK_SIZE_Q).

num of block we can divide our Query Sequence into