FlashAttentionNote

yeeboxie

April 17, 2025

Translate flashattention from column first to row first.

1 Memory-efficient forward pass

Recall that given input sequences $Q, K, V \in \mathbb{R}^{N \times d}$, we want to compute the attention output $O \in \mathbb{R}^{N \times d}$:

$$\begin{aligned} S &= QK^T \in \mathbb{R}^{N \times N}, \\ P &= \mathrm{softmax}(S) \in \mathbb{R}^{N \times N}, \\ O &= PV \in \mathbb{R}^{N \times d}. \end{aligned}$$

We have that $S_{ij} = q_i k_j^T$ where q_i and k_j are the i-th and j-th rows of Q and K respectively. Define the normalization constants of softmax:

$$L_i = \sum_j e^{q_i k_j^T}. \tag{1}$$

Let v_j be the j-th row of V, then the i-th row of the output is:

$$o_i = P_{i:} V = \sum_j P_{ij} v_j = \sum_j \frac{e^{q_i k_j^T}}{L_i} v_j \tag{2}$$

We see that once L_i is computed, we can compute o_i without extra memory by repeatedly summing $\frac{e^{q_i k_j^T}}{L_i}$. Therefore the forward pass can be computed with $O(n)$ extra memory.

1. Compute L_i for all i according to Eq. (1), which takes $O(n)$ extra memory.

2. Compute o_i for all i according to Eq. (2), which takes $O(d)$ extra memory.

2 Memory-efficient backward pass

We derive the backward pass of attention and show that it can also be computed with linear memory. We instead derive the backward pass explicitly and show how it can be computed in a memory-efficient manner.

Suppose that there is a scalar loss function ϕ, and let the output gradient be $\mathbf{dO} \in \mathbb{R}^{n \times d}$ (where $\mathbf{dO}$ denotes $\frac{\partial \phi}{\partial \mathbf{O}}$). We want to compute the input gradients $\mathbf{dQ}$, $\mathbf{dK}$, $\mathbf{dV} \in \mathbb{R}^{n \times d}$ (where $\mathbf{dQ}$, $\mathbf{dK}$, $\mathbf{dV}$ denote $\frac{\partial \phi}{\partial \mathbf{Q}}$, $\frac{\partial \phi}{\partial \mathbf{K}}$, $\frac{\partial \phi}{\partial \mathbf{V}}$ respectively).

The gradient $\mathbf{dV}$ is easy to see. Applying reverse-mode autodiff by hand (aka the chain rule), we obtain (in matrix notation) $\mathbf{dV} = \mathbf{P}^T \mathbf{dO}$. As:

$$dv_j = \sum_i P_{ij} do_i = \sum_i \frac{e^{q_i k_j^T}}{L_i} do_i \tag{3}$$

Since we already computed L_i, dv_j can be computed without extra memory by repeated summing.

The gradients $\mathbf{dQ}$ and $\mathbf{dK}$ are a little more complicated. We go through the gradients $\mathbf{dP}$ and $\mathbf{dS}$ first. From Eq. (2), we have that $\mathbf{dP} = \mathbf{dO}\mathbf{V}^T$, and so:

$$dP_{ij} = do_i v_j^T \tag{4}$$

Recall that $P_{i:} = \text{softmax}(S_i)$. Using the fact that the Jacobian of $y = \text{softmax}(x)$ is $\text{diag}(y) - yy^T$, we have that (This is column first, you need to tanspose this.)

$$dS_{i:} = (\text{diag}(P_{i:}) - P_{i:} P_{i:}^T) dP_{i:} = P_{i:} \circ dP_{i:} - (P_{i:}^T dP_{i:}) P_{i:} \tag{5}$$

where $\circ$ denotes pointwise multiplication.

Define

$$D_i = P_{i:} dP_{i:}^T = \sum_j \frac{e^{q_i k_j^T}}{L_i} do_i v_j^T = do_i \sum_j \frac{e^{q_i k_j^T}}{L_i} v_j^T = do_i o_i^T \tag{6}$$

then

$$dS_{ij} = P_{ij} dP_{ij} - D_i P_{ij} \tag{7}$$

Now we can get the gradients dQ and dK. Recall that $S_{ij} = q_i k_j^T$, so

$$dq_i = \sum_j dS_{ij} k_j = \sum_j P_{ij} (do_j v_j^T - D_i) k_j = \sum_j \frac{e^{q_i k_j^T}}{L_i} (do_j v_j^T - D_i) k_j \tag{8}$$

Similarly,

$$dk_j = \sum_i dS_{ij} q_i = \sum_i P_{ij} (do_j v_j^T - D_i) q_i = \sum_j \frac{e^{q_i k_j^T}}{L_i} (do_j v_j^T - D_i) q_i \tag{9}$$

Therefore the backward pass can also be computed with $O(n)$ extra memory:

1. Compute dv_j for all j according to Eq. (3), which takes $O(d)$ extra memory;
2. Compute D_i for all i according to Eq. (6), which takes $O(n)$ extra memory;
3. Compute dq_i for all i according to Eq. (8),which takes $O(d)$ extra memory;
4. Compute dk_j for al j according to Eq. (9), which takes $O(d)$ extra memory;