



10417/617

ATTENTION MASK, FLASH ATTENTION, MULTI-QUERY ATTENTION

► The most fundamental layer in the transformer:
Multi-head attention.

► Given vectors v_1, v_2, \dots, v_n , each in R^d , a multi-head attention layer is defined as:

►
$$v'_i = C \times \text{concatenate} \left(V_r^T \sum_j \alpha_{i,j}^r v_j \right)_{r \in [d/m]} + b$$

► Where $\left(\alpha_{i,j}^r \right)_{j \in [n]} = \text{softmax} \left(v_i^T Q_r K_r^T v_j + p_{i,j}^r \right)_{j \in [n]}$

► Here, C is a $d \times d$ trainable matrix.

► Each v_i looks for the “most similar v_j ”, according to $[d/m]$ many projection matrices Q_r and K_r .

Transformer Architecture

- ▶ A (post-layernorm) transformer block is defined as:
- ▶ Given input $W = \mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_n$, each \mathbf{v}_i in \mathbb{R}^d .
 - ▶ (1). Apply Multi-Head Attention (input dimension d , output dimension d) on W to get $V^{(1)} = \mathbf{v}_1^{(1)}, \mathbf{v}_2^{(1)}, \dots, \mathbf{v}_n^{(1)}$.
 - ▶ (2). Apply layer-norm on each of the $\mathbf{v}_i^{(1)}$ to get $\mathbf{v}_i^{(2)}$.
 - ▶ (3). Apply residual link: $\mathbf{v}_i^{(3)} = \mathbf{v}_i^{(2)} + \mathbf{v}_i$.
 - ▶ (4). Apply a one hidden layer MLP h (input dimension d , output dimension d) on each $\mathbf{v}_i^{(3)}$ to get $\mathbf{v}_i^{(4)} = h(\mathbf{v}_i^{(3)})$ (all the $\mathbf{v}_i^{(k)}$ in the uses the same h per layer, different h for different layers).
 - ▶ (5). Apply layer-norm on each of the $\mathbf{v}_i^{(4)}$ to get $\mathbf{v}_i^{(5)}$.
 - ▶ (6). Apply residual link: $\mathbf{v}_i^{(6)} = \mathbf{v}_i^{(5)} + \mathbf{v}_i^{(3)}$.
- ▶ The output $V^{(6)} = \mathbf{v}_1^{(6)}, \mathbf{v}_2^{(6)}, \dots, \mathbf{v}_n^{(6)}$, each $\mathbf{v}_i^{(6)}$ in \mathbb{R}^d .

Transformer Architecture

- ▶ A (pre-layernorm) transformer block is defined as:
- ▶ Given input $W = \mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_n$, each \mathbf{v}_i in \mathbb{R}^d .
 - ▶ (1). Apply layer-norm on each of the \mathbf{v}_i to get $\mathbf{v}_i^{(1)}$.
 - ▶ (2). Apply Multi-Head Attention on $V^{(1)}$ to get $V^{(2)} = \mathbf{v}_1^{(2)}, \mathbf{v}_2^{(2)}, \dots, \mathbf{v}_n^{(2)}$.
 - ▶ (3). Apply residual link: $\mathbf{v}_i^{(3)} = \mathbf{v}_i^{(2)} + \mathbf{v}_i$.
 - ▶ (4). Apply layer-norm on each of the $\mathbf{v}_i^{(3)}$ to get $\mathbf{v}_i^{(4)}$.
 - ▶ (5). Apply a one hidden layer MLP h on each $\mathbf{v}_i^{(4)}$ to get $\mathbf{v}_i^{(5)} = h(\mathbf{v}_i^{(4)})$ (all the $\mathbf{v}_i^{(k)}$ in the uses the same h per layer, different h for different layers).
 - ▶ (6). Apply residual link: $\mathbf{v}_i^{(6)} = \mathbf{v}_i^{(5)} + \mathbf{v}_i^{(3)}$.

Computation Time of Transformer Block

- ▶ A transformer block = MHA (m heads) + MLP.
- ▶ Assuming the context length is n and the embedding dimension is d.
- ▶ Forward/Backward time:
 - ▶ $nd^2(mlp) + (nd^2 + n^2d)$ (MHA)
- ▶ (Forward) Backward Memory:
 - ▶ $nd(mlp) + (nd + n^2m)$ (MHA)

Reducing Memory Usage of Attention

- ▶ Main Memory Usage:
- ▶ For each attention head, we need to store the $n \times n$ attention matrix:
- ▶
$$\left[\text{softmax}\left(v_i^T Q_r K_r^T v_j + p_{i,j}^r\right)_{j \in [n]} \right]_{i \in [n]}$$
- ▶ Let's just consider one row:
 - ▶ $\text{softmax}\left(v_i^T Q_r K_r^T v_j + p_{i,j}^r\right)_{j \in [n]}$
- ▶ Key idea of Flash-Attention:
 - ▶ We store $K_r^T v_j, Q_r^T v_j$ for every r and j , this takes memory $d \times n$.
 - ▶ We do not store the full softmax matrix, we will “compute them on the fly” to save memory.

Softmax Recomputation

- ▶ Consider $O = \sum_{i \in [n]} y_i \times \text{softmax}(x)_i$
- ▶ Where for each x_i, y_i , we need computation time d/m to retrieve it.
- ▶ Stupid-Attention computation:
 - ▶ For i in $\text{range}(n)$:
 - ▶ Compute $\text{norm_factor} = \text{norm_factor} + \exp(x_i)$.
 - ▶ Compute $O = O + y_i \exp(x_i)$
 - ▶ Return $O/\text{norm_factor}$
- ▶ This only requires memory $O(M)$, where $M = d/m$ is the dimension of y_i

From Stupid Attention to Flash Attention

- ▶ Why is Stupid Attention Stupid?
- ▶ Floating Point accuracy. We can not compute $\sum \exp(x_i)$ accurately!
No such accuracy.
- ▶ Stupid Attention V2:
 - ▶ Go through i , compute the max of x_i as $m(x)$
 - ▶ For i in range(n):
 - ▶ Compute $norm_factor = norm_factor + \exp(x_i - m(x))$.
 - ▶ Compute $O = O + y_i \exp(x_i - m(x))$
 - ▶ Return $O/norm_factor$
- ▶ But then we need to compute x_i twice, unless we store it in the memory...

From Stupid Attention V2 to Flash Attention

- ▶ Stupid Attention V3 is an upgrade of stupid attention v2, where we only compute x_i once and maintain the correct floating-point accuracy.
- ▶ For i in range(n):
 - ▶ Compute $m_{new}(x) = \max(m(x), x_i)$
 - ▶ Compute $norm = \exp(m(x) - m_{new}(x)) norm + \exp(x_i - m_{new}(x))$.
 - ▶ Compute $O = \exp(m(x) - m_{new}(x))O + y_i \exp(x_i - m_{new}(x))$
 - ▶ Update $m(x) = m_{new}(x)$
- ▶ Output $O/norm$.

From Stupid Attention V3 to Flash Attention

Now the memory usage is good.

Main problem: For i in range(n).

- Cuda operates on the so-called "Thread Block", so the computation is very fast for operations of "certain sizes".

In stupid attention v3, the computation inside for loop is:

- Vector of size $M = d/m$ per i . This is typically smaller than the "certain sizes" when m is large.

So we need to do some chunking...

Flash Attention

- ▶ Flash attention is a little bit more involved than the previous slides.
- ▶ It divides the computation in chunks of R
- ▶ For i in range($n//R$):
 - ▶ Compute the softmax for $x[iR:iR + R]$ using the fastest way, which uses memory R. Then compute
$$O_i = \sum_{j \in [iR, iR+R)} y_j \times \text{softmax}(x[iR:iR + R])_j$$
 (only store this O_i in SRAM).
 - ▶ Store the max of $x[j]$ for j in $[iR, iR + R)$ in memory as $m[i]$.
 - ▶ Store the normalization factor of the softmax (after subtracting the max) of $x[iR:iR + R]$ in memory as $norm[i]$.
 - ▶ Update $m_{new}(x) = \max(m(x), m[i])$
 - ▶ Update $O = O \exp(m(x) - m_{new}(x)) + \exp(m[i] - m_{new}(x)) O_i \times norm[i]$
 - ▶ Update $norm = \exp(m(x) - m_{new}(x)) norm + norm[i] \times \exp(m[i] - m_{new}(x))$.
 - ▶ Update $m(x) = m_{new}(x)$

Autoregressive Training

Recall in the autoregressive training objective

- Given $X[0:i]$, we want to predict $X[i]$, for every i in $[context_length]$

Naïve implementation: Treat $X[0:i]$ as a separate input with label $X[i]$.

- Total computation time: $context_length * \text{computation time on input } X[0:context_length]$

Can we do it more efficiently in computation time of a single $X[0:context_length]$?

Attention Mask

- ▶ The core of MHA is the soft-max attention score:
- ▶ $(\alpha_{i,j}^r)_{j \in [n]} = \text{softmax}(v_i^T Q_r K_r^T v_j + p_{i,j}^r)_{j \in [n]}$
- ▶ Key observation: We can set $p_{i,j}^r = -\infty$ if and only if $i < j$ (attention mask).
- ▶ In this way, the new value
 - ▶ $v'_i = C \times$
 $\text{concatenate}(V_r^T \sum_j \alpha_{i,j}^r v_j)_{r \in [d/m]} + b$
- ▶ v'_i only depends on v_j for $j \leq i$.

Inference

After autoregressive training, we can use the autoregressive language model to generate texts.



Given a prompt s (text), we can

Tokenize the prompt s into a list of integers S .

* Feed S into the autoregressive language model, and obtain its prediction S_{pred} .

Update $S = \text{concatenate}(S, S_{pred})$.

Repeat Step *.

Multi-Query Attention

- ▶ Optimized for inference speed.
- ▶ Time-consuming step for inference:
 - ▶ Feed S into the autoregressive language model, and obtain its prediction S_{pred} .
 - ▶ We do not want to recompute $\text{model}(S)$ every time we update S .
- ▶ Key observation: Caching.
 - ▶ We can cache the past $K_r^T v_j$ and $V_r^T v_j$ values for all $j < \text{len}(S)$, and no need to recompute them.
 - ▶ However, this requires us to cache
 - ▶ $d \times \text{len}(S)$ many values.



Multi-Query Attention

- ▶ Multi-query attention:
- ▶ Instead of using $(\alpha_{i,j}^r)_{j \in [n]} = \text{softmax}(v_i^T Q_r K_r^T v_j + p_{i,j}^r)_{j \in [n]}$
- ▶ $v'_i = C \times \text{concatenate}(V_r^T \sum_j \alpha_{i,j}^r v_j)_{r \in [d/m]} + b$
- ▶ We now use $(\alpha_{i,j}^r)_{j \in [n]} = \text{softmax}(v_i^T Q_r \mathbf{K}^T v_j + p_{i,j}^r)_{j \in [n]}$
- ▶ $v'_i = C \times \text{concatenate}(\mathbf{V}^T \sum_j \alpha_{i,j}^r v_j)_{r \in [d/m]} + b$
- ▶ So every head shares the same K, V
 - ▶ (of dimension embed_dim x head_dim).

