EL-Attention: Memory Efficient Lossless Attention for Generation Supplementary Material

A. Pseudocode

To better understand differences between EL-attention and multi-head attention in generation, we list their pseudocode for the comparison. For simplicity, bias term and some non-essential operations are omitted.

Algorithm 1 shows the generation process for a typical encoder-decoder model. First, input is encoded by model, then decoder starts repetitively executing, one token is generated per step. The high level logic is the same for both attention methods, differences are in the cache and computation.

The generation process with multi-head attention is listed in Algorithm 2. First, the encoder output is repeated(h) beam size times to match query's first dimension. Second, the key and value cache are built for every layer. So its cache size is linear to number of the decoder layer L, beam size x and batch size b.

EL-attention (See Algorithm 3) does not need to repeat the encoder output and get rid of the per layer cache for storing key and value. To achieve these benefits, EL-attention shifts some computation on key and value to the query side. Due to the length of query is always one which is much shorter than the length of key and value in most cases, it reduces the computations significantly.

EL-attention can achieve faster speed due to time saved from reorder_cache(), two torch.bmm operations, and the ability to use much larger batch size.

B. Proof for EL-Attention

In this section, we will present the proof that EL-attention can have the same result as multi-head attention via the choice of FFN functions. And again, there is no explicit conversion on key and value.

Recall that, by expanding Equation 1 and 2 from the paper, multi-head attention can be formulated as:

$$\begin{aligned}\text{MultiHead}(Q,K,V) &= \sum_{i=1}^{h} \overbrace{\text{softmax}(\frac{Q_i K_i^T}{\sqrt{d_k}})}^{\text{Prob}_i} V_i W_i^O \\ \text{where } Q_i &= QW_i^Q + b_i^Q,\ K_i = KW_i^K + b_i^K, \\ V_i &= VW_i^V + b_i^V, \text{ and } H = K = V\end{aligned} \tag{6}$$

Here, Q $\in \mathbb{R}^{1\times d_m}$, H $\in \mathbb{R}^{n\times d_m}$, W_i^Q, W_i^K, $W_i^V \in \mathbb{R}^{d_m\times d_k}$ and $W_i^O \in \mathbb{R}^{d_k\times d_m}$. We include bias term in this proof, it is omitted in previous equations for simplification.

The multiplication of single head query and single head key can be replaced by the multiplication of expanded query and original key, derived as:

$$\begin{aligned}Q_i K_i^T &= (QW_i^Q + b_i^Q)(KW_i^K + b_i^K)^T \\ &= (QW_i^Q + b_i^Q)((KW_i)^T \\ &\quad + (QW_i^Q + b_i^Q)(b_i^K)^T \\ &= \text{FFN}_i^Q(Q)K^T + Q_i(b_i^K)^T \\ \text{where FFN}_i^Q(Q) &= (QW_i^Q + b_i^Q)(W_i^K)^T \\ \text{and } Q_i &= QW_i^Q + b_i^Q\end{aligned} \tag{7}$$

Below is the EL-attention conversion from single head attention result to final output in multi-head attention:

$$\begin{aligned}\text{Prob}_i \cdot V_i \cdot W_i^O &= \text{Prob}_i(VW_i^V + b_i^V)W_i^O \\ &= \text{Prob}_i(VW_i^V)W_i^O \\ &\quad + \text{Prob}_i \cdot \text{Repeat}(b_i^V) \cdot W_i^O \\ &= \text{FFN}_i^O(X) + b_i^V W_i^O \\ \text{where FFN}_i^O(X) &= XW_i^V W_i^O \\ &\text{and } X = \text{Prob}_i \cdot V \\ &\text{and Repeat}(b_i^V) \text{ is broadcasting dim}\end{aligned} \tag{8}$$

To ensure the equivalence to multi-head attention, we adjust EL-attention as:

$$\begin{aligned}\text{EL}(Q,K,V) &= \sum_{i=1}^{h} \text{FFN}_i^O(\text{Prob}_i \cdot V) + \sum_{i=1}^{h} b_i^V W_i^O \\ \text{where Prob}_i &= \text{softmax}(\frac{\text{FFN}_i^Q(Q)K^T + Q_i(b_i^K)^T}{\sqrt{d_k}}) \\ \text{and } H &= K = V\end{aligned} \tag{9}$$

By leveraging the associative property of matrix multiplication, Equation 6 and Equation 9 are interchangeable.

Please note that some bias terms can be omitted when training a new model. Like the bias term b_i^K that adding the same value for all attention positions, and b_i^V that contributing constant information to the output, it is independent of query/key/value and the sum of all elements in Prob_i's last dimension is always one.

Algorithm 1 Generation Process

Input: data src_tokens, beam size x
Output: $tokens$
$encoder\_outs = \text{forward\_encoder}(src\_tokens)$
Initialize $previous\_output[:] = \text{BOS}$
Initialize $tokens = array$
for $t = 0$ **to** T **do**
  $logits = \text{forward\_decoder}(previous\_output, encoder\_outs)$
  $previous\_output, order\_index = \text{sample}(logit, x)$
  $tokens = \text{reorder}(tokens, order\_index)$
  $tokens[t, :] = previous\_output$
end for

Algorithm 2 forward_decoder with multi-head attention

function forward_decoder(previous_output, h)
  if $cache$ is None **then**
    $h = \text{repeat}(h)$ {repeat beam size times}
  else
    reorder_cache() {Cache size: O(2BLSD), where B is beam size, L is decoder layer, S is sequence length, D is model dimension.}
  end if
  $x = \text{embedding}(previous\_output)$
  for $i = 0$ **to** layers L **do**
    $x = \text{self\_attention}(x)$
    $x = \text{encoder\_decoder\_attention}(x, h, h)$
    $x = \text{mlp}(x)$
  end for
  return $\text{predict\_on\_vocab}(x, unembedding\_weight)$
end function

function encoder_decoder_attention(query, key, value)
  $\{query \in [bx, 1, d_m]\}$
  if $cache[i, k]$ is None **then**
    $cache[i, k] = \text{reshape}(\text{torch.mm}(key, W_k^i))$
    $cache[i, v] = \text{reshape}(\text{torch.mm}(value, W_v^i))$
  end if
  $k = cache[i, k]$ $\{k \in [bxh, d_k, n]\}$
  $v = cache[i, v]$ $\{v \in [bxh, n, d_k]\}$
  $\{q \in [bxh, 1, d_k]\}$
  $q = \text{reshape}(\text{torch.mm}(query, W_q^i))$
  $weights = \text{torch.bmm}(q, k)$
  $prob = \text{softmax}(weights)$
  $attn = \text{torch.bmm}(prob, v)$
  $attn = \text{torch.mm}(attn, W_o^i)$
end function

Algorithm 3 forward_decoder with EL-attention

function forward_decoder(previous_output, h)
  if $cache$ is not None **then**
    reorder_cache() {Cache size: O(SD), where S is sequence length, D is model dimension. Which is 2BL times less. }
  end if
  $k = \text{reshape}(h)$ $\{k \in [b, d_m, n]\}$
  $v = \text{reshape}(h)$ $\{v \in [b, n, d_m]\}$
  $x = \text{embedding}(previous\_output)$
  for $i = 0$ **to** layers L **do**
    $x = \text{self\_attention}(x)$
    $x = \text{encoder\_decoder\_attention}(x, k, v)$
    $x = \text{mlp}(x)$
  end for
  return $\text{predict\_on\_vocab}(x, unembedding\_weight)$
end function

function encoder_decoder_attention(query, k, v)
  $\{query \in [bx, 1, d_m]\}$
  {No heavy op for building multi-head key/value.}
  {Encoder output is directly used as key and value, and shared among all layers.}
  $q = \text{reshape}(\text{torch.mm}(query, W_q^i))$ $\{q \in [bx, h, d_k]\}$
  $\{W_k^i \in [h, d_k, d_m]\}$
  $q = \text{reshape}(\text{torch.bmm}(q, W_k^i))$ $\{q \in [b, hx, d_m]\}$
  $weights = \text{torch.bmm}(q, k)$
  $prob = \text{softmax}(weights)$
  $attn = \text{torch.bmm}(prob, v)$ $\{attn \in [b, hx, d_m]\}$
  $\{W_v^i \in [h, d_m, d_k]\}$
  $attn = \text{reshape}(\text{torch.bmm}(attn, W_v^i))$
  $attn = \text{torch.mm}(attn, W_o^i)$
end function