Low Latency Attention Module

October 2023

1 Notation

- the matrix is denoted as capital character
- bold lower character represents vector (column-wise)
- lower character is the scalar
- superscript denotes time index
- subscript indicates element of its correspondent entity
- $\mathcal{L}$ represents the loss

2 Streaming Attention

2.1 Forward of streaming attention

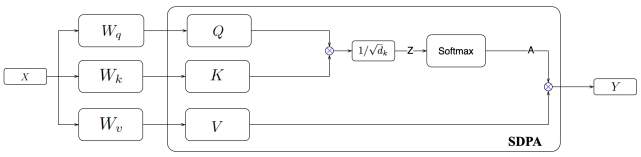

Figure 1: Forward of scale-dot production attention.

Fig. 1 is the forward structure of scaled-dot product attention, which is the core part of transformer [1]. The input, intermediate results and output are represented as capital characters because usually more than one frames are processed, which can be conveniently represented by matrices. First, ignore the scalar $1/\sqrt{d_k}$ for simplicity. Also assume we only have look ahead, which is N. Looking back will be considered when generalize it. The formulas for the forwarding part of this structure are as follows. For calculating $\mathbf{z}_t$, we have,

$$\mathbf{z}_t = [\mathbf{k}_t, \mathbf{k}_{t+1}, ..., \mathbf{k}_{t+N}]^T \mathbf{q}_t \tag{1}$$

where t is the time index, ^{T} is transpose operator.

For calculating $\mathbf{a_t}$, we have,

$$\mathbf{a}_t = \text{SoftMax}(\mathbf{z}_t), \quad where \quad a_i^t = \frac{exp(z_i^t)}{\sum_j exp(z_j^t)} \tag{2}$$

where in a_i^t, subscript i is the i^{th} element of vector $\mathbf{a}_t$.

For calculating $\mathbf{y_t}$, we have,

$$\mathbf{y}_t = [\mathbf{v}_t, \mathbf{v}_{t+1}, ..., \mathbf{v}_{t+N}]^T \mathbf{a}_t. \tag{3}$$

In element-wise, it becomes,

$$y_i^t = \sum_{j=0}^{dv-1} v_i^{t+j} a_j^t, \tag{4}$$

where dv is the dimension of $\mathbf{v}$.

2.2 Backward of streaming attention

Backward propagation needs to calculate the derivative of loss function to each vector. This includes $\frac{\partial L}{\partial \mathbf{v_t}}$, $\frac{\partial L}{\partial \mathbf{k_t}}$ and $\frac{\partial L}{\partial \mathbf{q_t}}$, where L is the overall loss for one update step.

2.2.1 Partial derivative of values

First, calculate the derivative of loss to $\mathbf{v_t}$. Take a look at eq. 3, one frame $\mathbf{v_t}$ will affect N output of $\mathbf{y}$, with different time index. The derivative can be viewed as the change direction and value when there is small change in the variable. Then with a small variation of $\mathbf{v_t}$, N output frames will be change and the changing values are specified by elements of $\mathbf{a}$ with same time index of $\mathbf{y}$.

This is specified as

$$\frac{\partial \mathcal{L}}{\partial \mathbf{v_t}} = \sum_n \frac{\partial \mathcal{L}}{\partial \mathbf{y_n}} \frac{\partial \mathbf{y_n}}{\partial \mathbf{v_t}}. \tag{5}$$

We need to sum through n because $\mathbf{v_t}$ and $\mathbf{y_t}$ have time mixing, which means that $\frac{\partial \mathbf{y_n}}{\partial \mathbf{v_t}}$ is not zero for some $n \neq t$.

By looking at eq. 3, the following equation can be derived,

$$\frac{\partial \mathbf{y_n}}{\partial \mathbf{v_t}} = \begin{cases} 0 & \text{others} \\ a_{t-n}^n \cdot \mathrm{I} & \text{for} \quad t - N \leq n \leq t \end{cases}. \tag{6}$$

where I is the $dv \times dv$ identity matrix.

Together with eq. 5 and eq. 6, the full derivative of $\mathbf{v_t}$ is presented as,

$$\boxed{\frac{\partial \mathcal{L}}{\partial \mathbf{v_t}} = \sum_{n=t-N}^{t} a_{t-n}^n \frac{\partial L}{\partial \mathbf{y_n}}.} \tag{7}$$

Note that a_{t-n}^n is a scalar.

2.2.2 Backward of query

Backward propagation of query is simpler since there is no time mixing between $\mathbf{y}$ and $\mathbf{q}$. First, $\partial L/\partial \mathbf{a}$ and $\partial L/\partial \mathbf{z}$ need to be calculated. The first step is $\partial \mathbf{y_t}/\partial \mathbf{a_t}$. From eq. 3, it is straightforward that,

$$\frac{\partial \mathbf{y_t}}{\partial \mathbf{a_t}} = \underbrace{[\mathbf{v_t}, \mathbf{v_{t+1}}, ..., \mathbf{v_{t+N}}]}_{\text{N+1 columns}}. \tag{8}$$

The Jacobian matrix of softmax operator can be found online and it is as the form,

$$\frac{\partial \mathbf{a_t}}{\partial \mathbf{z_t}} = \mathrm{J}, \quad where \quad J_{ij} = a_i^t(\delta_{ij} - a_j^t), \tag{9}$$

where $\delta_{ij} = 1$ if $i = j$, otherwise is zero. Then combine eq. 8 and eq. 9, we can get,

$$\boxed{\frac{\partial \mathbf{y_t}}{\partial \mathbf{z_t}} = [\mathbf{v_t}, \mathbf{v_{t+1}}, ..., \mathbf{v_{t+N}}]\mathrm{J}}. \tag{10}$$

From eq. 1, it is straightforward that

$$\frac{\partial \mathbf{z_t}}{\partial \mathbf{q_t}} = [\mathbf{k_t}, \mathbf{k_{t+1}}, ..., \mathbf{k_{t+N}}]^T. \tag{11}$$

Then combine eq. 11 with eq. 10, the derivative for query is given as below,

$$\boxed{\frac{\partial \mathcal{L}}{\partial \mathbf{q_t}} = \frac{\partial \mathcal{L}}{\partial \mathbf{y_t}} \frac{\partial \mathbf{y_t}}{\partial \mathbf{z_t}} \frac{\partial \mathbf{z_t}}{\partial \mathbf{q_t}} = \frac{\partial \mathcal{L}}{\partial \mathbf{y_t}} [\mathbf{v_t}, \mathbf{v_{t+1}}, ..., \mathbf{v_{t+N}}]\mathrm{J}[\mathbf{k_t}, \mathbf{k_{t+1}}, ..., \mathbf{k_{t+N}}]^T.} \tag{12}$$

2.2.3 Backward of key

Backward propagation of key is more complicated than query, because there is time mixing between $\mathbf{y}$ and $\mathbf{k}$.

Recall eq. 1, we can get,

$$\frac{\partial \mathbf{z_n}}{\partial \mathbf{k_t}} = \begin{cases} \mathrm{M_n} & \text{for} \quad t - N \leq n \leq t \\ 0 & \text{others} \end{cases}, \tag{13}$$

and $\mathrm{M_n}$ is given by,

$$\mathrm{M_n} = \begin{bmatrix} 0 & \dots & 0 \\ \vdots & \vdots & \vdots \\ q_n^0 & \dots & q_n^{dk} \\ \vdots & \vdots & \vdots \\ 0 & \dots & 0 \end{bmatrix} \Bigg\}_{\text{(t-n)}^{th}} \tag{14}$$

where q_n^{dk} is the dk^{th} element of vector $\mathbf{q}_n$, $\mathbf{q}_n^T$ is in the $(t-n)^{th}$ row of the result matrix and the dimensionality of the matrix is $(nlook\_ahead \times dk)$.

Since there are time mixing between $\mathbf{y}$ and $\mathbf{k}$, we can wrote the partial derivative of $\mathcal{L}$ to $\mathbf{k}$ as:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{k_t}} = \sum_n \frac{\partial \mathcal{L}}{\partial \mathbf{y_n}} \frac{\partial \mathbf{y_n}}{\partial \mathbf{z_n}} \frac{\partial \mathbf{z_n}}{\partial \mathbf{k_t}}. \tag{15}$$

By bring eq. 13 into eq. 15, we can finally get the derivative by chain rule as,

$$\boxed{\frac{\partial \mathcal{L}}{\partial \mathbf{k_t}} = \sum_n \frac{\partial \mathcal{L}}{\partial \mathbf{y_n}} \frac{\partial \mathbf{y_n}}{\partial \mathbf{z_n}} \frac{\partial \mathbf{z_n}}{\partial \mathbf{k_t}} = \sum_{n=t-N}^{t} \frac{\partial \mathcal{L}}{\partial \mathbf{y_n}} \frac{\partial \mathbf{y_n}}{\partial \mathbf{z_n}} \mathrm{M_n}.} \tag{16}$$

where $\frac{\partial \mathbf{y_n}}{\partial \mathbf{z_n}}$ is defined in eq. 10 and M_n is given by eq. 14.

2.3 Implementation of the Backward Propagation for Streaming Attention

The backward propagation of streaming attention needs to calculate the partial derivative of loss to each frame of value, query, and key in the scaled-dot product attention. This means we need to implement eq. 7, eq. 12 and eq. 16. Note that $\frac{\partial \mathcal{L}}{\partial \mathbf{y_t}}$ comes from outside of the scaled-dot product attention.

To efficiently calculate the derivatives, intermediate variable values need to be stored. This includes $\mathbf{a}$ and $\mathbf{z}$.

Calculation of eq. 16 can also be simplified, because M_n only has one row is not zero vector. This simplify the matrix multiplication as using $(t-n)^{th}$ column of $\frac{\partial \mathbf{y_n}}{\partial \mathbf{z_n}}$ to dot-multiply $(t-n)^{th}$ row of M_n.

It also can generalize to the case with look back information (e.g. $nlook\_back = B$). The change to the derivative is then just change the index. In eq. 7, the up limit for t is $t+B$. For eq. 12, the matrix $[\mathbf{v}_t, \mathbf{v}_{t+1}, ..., \mathbf{v}_{t+N}]$ becomes $[\mathbf{v}_{t-B}, \mathbf{v}_{t-B+1}, ..., \mathbf{v}_t, \mathbf{v}_{t+1}, ..., \mathbf{v}_{t+N}]$. The same change needs to be applied to $[\mathbf{k}_t, \mathbf{k}_{t+1}, ..., \mathbf{k}_{t+N}]$ as well. Derivative to the key needs to be changed accordingly. This includes eq. 13 and eq. 15. In eq. 13, upper bound of n is replaced as $t+B$ and the same is applied to 15 with substituting the $\frac{\partial \mathbf{y}_n}{\partial \mathbf{z}_n}$ with the one computed with information before the current frame.

3 Low Latency Streaming Attention

The method specified in Section 2 implemented the time-restricted attention [1]. It does not solve problem that the latency accumulating through different layers.

In order to prevent the latency accumulating as the layer increase, here we describe the low latency streaming attention kernel. We start with the inference.

If the model is trained with look ahead as 2 and look back as 3, the inference which exactly matches the offline training can be depicted as in figure 2.

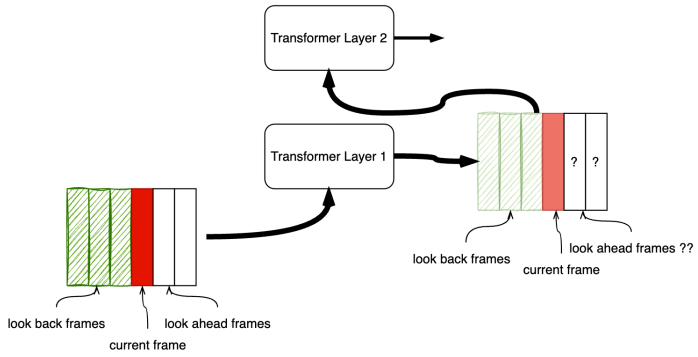

Figure 2: Latency accumulation by number of layers.

In the first layer, when process the current frame, 2 look ahead frames will be delayed. After the first transformer layer, though the look back frames can be stored in buffer, the current frame has no look ahead. To process the second transformer layer, 2 extra frames will again be delayed. This process is repeated as process layer 3 further, which means the latency is 2 frames multiplied by the number of layers.

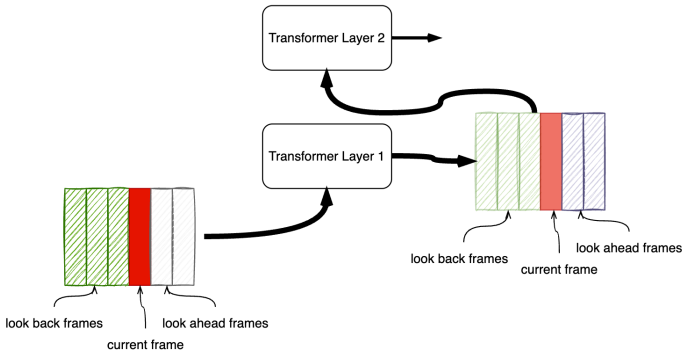

Figure 3: Latency accumulation by number of layers.

3.1 Forward of low latency streaming attention

To prevent the latency accumulation, the inference can be modified. This is achieved by processing the look ahead frames of the current frame by using less look ahead information. For example, in figure 3, to process the current frame,

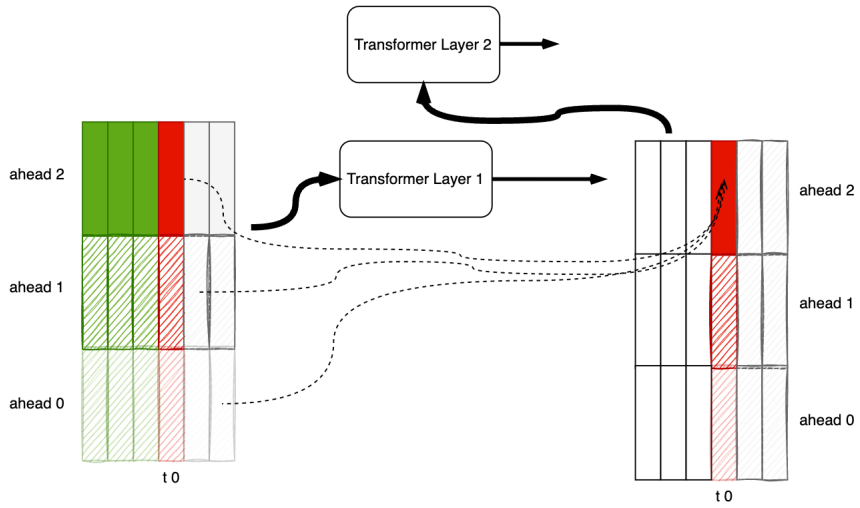

Figure 4: Low Latency Training illustration.

2 look ahead frames are required. If we follow exactly the training procedure use the stream attention specified above, 2 extra frames needed to process the second layer. But instead of waiting for 2 extra frames in the first layer, the look ahead frames can be processed through the first transformer layer with less look ahead frames. If the current frame is index as $t = 0$, the $t = 1$ frame is processed using 1 look ahead and $t = 2$ frame used no look ahead information. Then the output of first layer still has two look ahead, though they are different compared with those of figure 2. This inference is then repeated for the further layers without latency accumulated and so the latency is 2 frames.

Figure 4 illustrates the low latency streaming attention kernel. The idea of this training is that instead processing each frame with 2 look ahead during training, 2 extra processed frames are constructed for each layer. For example in figure 4, frame at $t = 0$ and $ahead = 2$ row after first transformer is processed using frames $t = 1, ahead = 1$ and $t = 2, ahead = 0$ as look ahead. Frame at $t = 0$ and $ahead = 1$ row after first transformer is processed using frames $t = 1, ahead = 0$ as look ahead, which means it only used one look ahead frame during training. Frame at $t = 0$ and $ahead = 0$ row after first transformer is processed using no look ahead frames. It can be seen that each frame has three (number of look ahead plus 1) different forms, each of which uses different amount of look ahead frames and will be used for next other frames when process the next layer.

3.2 Backward propagation of low latency streaming attention

Backward propagation of Section 3.1 is more complicated then the one described in Section 2.2, but they share many the same contents.

Again, backward propagation needs to calculate the derivative of loss function to each vector, including $\frac{\partial \mathcal{L}}{\partial \mathbf{v}_{t,ah}}$, $\frac{\partial \mathcal{L}}{\partial \mathbf{k}_{t,ah}}$ and $\frac{\partial \mathcal{L}}{\partial \mathbf{q}_{t,ah}}$, where the subscript $_{ah}$ denotes the look ahead row.

3.2.1 Partial derivative of values

First, calculate the derivative of loss to $\mathbf{v}_{t,ah}$. Unlike eq. 5, the look ahead row needs to be summed, which can be written as

$$\frac{\partial \mathcal{L}}{\partial \mathbf{v}_{t,ah2}} = \sum_{ah1} \sum_{n} \frac{\partial \mathcal{L}}{\partial \mathbf{y}_{n,ah1}} \frac{\partial \mathbf{y}_{n,ah1}}{\partial \mathbf{v}_{t,ah2}}. \tag{17}$$

However, not all the combination of n, ah are non-zeros. Those $n + ah = nlook\_ahead$ are non-zeros. This makes the derivative as,

$$\frac{\partial \mathbf{y}_{n,ah2}}{\partial \mathbf{v}_{t,ah2}} = \begin{cases} 0 & \text{others} \\ a^n_{t-n,ah1} \cdot \mathrm{I} & \text{for} \quad t - N + ah2 \leq n \leq t + ah2 \text{ and } ah1 \leq nlook\_ahead \end{cases}. \tag{18}$$

where I is the $dv \times dv$ identity matrix.
Together with eq. 17 and eq. 18, the full derivative of $\mathbf{v}_{t,ah}$ is specified as,

$$\boxed{\frac{\partial \mathcal{L}}{\partial \mathbf{v}_{t,ah2}} = \sum_{ah1} \sum_{n=t-N}^{t} a^n_{t-n,ah1} \frac{\partial \mathcal{L}}{\partial \mathbf{y}_{n,ah1}}, \text{where } ah_1 \leq nlook\_ahead.} \tag{19}$$

Note that $a^n_{t-n,ah1}$ is a scalar.

3.2.2 Backward of query

Same as in Section 2.2.2, $\partial \mathcal{L}/\partial \mathbf{a}$ and $\partial L/\partial \mathbf{z}$ are first calculated. The first step is $\partial \mathbf{y}_t / \partial \mathbf{a}_t$.

Start from output to attention score,

$$\frac{\partial \mathbf{y}_{t,ah}}{\partial \mathbf{a}_{t,ah}} = \underbrace{[\mathbf{v}_{t-ah,N}, \mathbf{v}_{t-ah+1,N-1}, ..., \mathbf{v}_{t-ah+N,0}]}_{\text{N+1 columns}}. \tag{20}$$

The Jacobian matrix is the same as eq. 9, except $_{ah}$ should be included. Then combine eq. 20 and Jacobian matrix of SoftMax operator, we can get,

$$\boxed{\frac{\partial \mathbf{y}_{t,ah}}{\partial \mathbf{z}_{t,ah}} = [\mathbf{v}_{t-ah,N}, \mathbf{v}_{t-ah+1,N-1}, ..., \mathbf{v}_{t-ah+N,0}]\mathrm{J}}. \tag{21}$$

From the forward operator, the derivative of $\mathbf{z}_{t,ah}$ to $\mathbf{q}_{t,ah}$ can be calculated as,

$$\frac{\partial \mathbf{z}_{t,ah}}{\partial \mathbf{q}_{t,ah}} = [\mathbf{k}_{t-ah,N}, \mathbf{k}_{t-ah+1,N-1}, ..., \mathbf{k}_{t-ah+N,0}]^T. \tag{22}$$

Then combine eq. 22 with eq. 21, the derivative for query is given as below,

$$\boxed{\frac{\partial \mathcal{L}}{\partial \mathbf{q}_{t,ah}} = \frac{\partial \mathcal{L}}{\partial \mathbf{y}_{t,ah}} \frac{\partial \mathbf{y}_{t,ah}}{\partial \mathbf{z}_{t,ah}} \frac{\partial \mathbf{z}_{t,ah}}{\partial \mathbf{q}_{t,ah}} = \frac{\partial \mathcal{L}}{\partial \mathbf{y}_{t,ah}} [\mathbf{v}_{t-ah,N}, \mathbf{v}_{t-ah+1,N-1}, ..., \mathbf{v}_{t-ah+N,0}] \mathrm{J} [\mathbf{k}_{t-ah,N}, \mathbf{k}_{t-ah+1,N-1}, ..., \mathbf{k}_{t-ah+N,0}]^T.} \tag{23}$$

3.2.3 Backward of key

The same as section 2.2.3, backward propagation of key is more complicated than query as the time mixing between $\mathbf{y}$ and $\mathbf{k}$.

We can use the same format in section 2.2.3. First let us think about $\mathbf{z}_{n,ah}$ over $\mathbf{k}_{t,ah}$, we can get,

$$\frac{\partial \mathbf{z}_{n,ah1}}{\partial \mathbf{k}_{t,ah2}} = \begin{cases} \mathrm{M}_{n,ah2} & \text{for} \quad t - N + ah2 \leq n \leq t + ah2 \\ 0 & \text{others} \end{cases}, \tag{24}$$

and $M_{n,ah2}$ is given by,

$$\mathrm{M}_{n,ah2} = \left[\begin{array}{ccc} 0 & \dots & 0 \\ \vdots & \vdots & \vdots \\ q^0_{n,ah1} & \dots & q^{dk}_{n,ah1} \\ \vdots & \vdots & \vdots \\ 0 & \dots & 0 \end{array} \right] \Bigg\}_{\text{(t-n+ah2)}^{th}} \tag{25}$$

where $q^{n,ah1}_{dk}$ denotes the dk^{th} element of vector $\mathbf{q}_{n,ah1}$, $\mathbf{q}^T_{n,ah1}$ is in the $(t - n + ah2)^{th}$ row of the result matrix and the dimensionality of the matrix is $(nlook\_ahead \times dk)$.

Similar as eq. 15, we can wrote the partial derivative of L to $\mathbf{k}$ as:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{k}_{t,ah2}} = \sum_{ah1} \sum_{n} \frac{\partial \mathcal{L}}{\partial \mathbf{y}_{n,ah1}} \frac{\partial \mathbf{y}_{n,ah1}}{\partial \mathbf{z}_{n,ah1}} \frac{\partial \mathbf{z}_{n,ah1}}{\partial \mathbf{k}_{t,ah2}}. \tag{26}$$

By bring eq. 24 into eq. 26, we can finally get the derivative by chain rule as,

$$\boxed{\frac{\partial \mathcal{L}}{\partial \mathbf{k}_{t,ah2}} = \sum_{ah1} \sum_{n} \frac{\partial \mathcal{L}}{\partial \mathbf{y}_{n,ah1}} \frac{\partial \mathbf{y}_{n,ah1}}{\partial \mathbf{z}_{n,ah1}} \frac{\partial \mathbf{z}_{n,ah1}}{\partial \mathbf{k}_{t,ah2}} = \sum_{ah1} \sum_{n=t-N}^{t} \frac{\partial \mathcal{L}}{\partial \mathbf{y}_{n,ah1}} \frac{\partial \mathbf{y}_{n,ah1}}{\partial \mathbf{z}_{n,ah1}} \mathrm{M}_{n,ah2}.} \tag{27}$$

where $\frac{\partial \mathbf{y_n}}{\partial \mathbf{z}_{n,ah1}}$ is defined in eq. 21 and $M_{n,ah2}$ is given by eq. 25.

3.3 Add look back into low latency streaming attention

Add look back into the forward computation of low latency streaming attention kernel is straightforward. The only difference is that the look back amount would be different when the *ahead* row is different. For example, $ahead = 0$, the number of look back frames is B, suppose B is the default number of look back frames. But for $ahead = 1$ row, the number of look back frames is $B + 1$.

Adding the look back to derivatives are not as straightforward as those in Section 2.3. As the look back information for the first B frames are from the row $ahead = 0$ and extra frames are from $ahead > 0$ row, this add extra complication. But still it is manageable, since the row is chosen by time slice. One can add the look back information to the equation without many modifications.

4 Some Notes

It can be seen that in the neural network, the forward path is composed of different operators. The backward propagation is to calculate those operators Jacobian matrices. The dimension of each Jacobian matrix is $(d_{output} \times d_{input})$, as the input and output is usually denoted as column vector.

Then the chain rule of derivative is used to combine all the different Jacobian matrices to form the entire error propagation path.

The matrix multiplication and vector format is just to make the calculation more compact. The matrix multiplication operator corresponds the summation across the dimension of one vector. For example $c = \mathbf{a}^T\mathbf{b}$ and $\mathbf{b} = \mathrm{M}\mathbf{d}$, the derivative means when one element changes for a small value, what is the proportion that the result will change. We can see that for each element of $\mathbf{a}$ change, c will change according to $\mathbf{b}$, which means $\partial c/\partial b_i = a_i$. We then want to write it in a vector format, which ends with $\partial c/\partial \mathbf{b} = \mathbf{a}$. We will see the convenience the vector format brought soon.

If we want to compute the derivative further, say $\partial c/\partial d_i$, and how we can achieve it. We can see that, for each element b_i, every element of $\mathbf{d}$ was involved. That means when applying the chain rule, each element of $\mathbf{b}$ should be marginalized, $\partial c/\partial d_i = \sum_n \frac{c}{\partial b_n}\frac{\partial b_n}{\partial d_i}$. Note that the summation here is the summation from $\mathbf{b}$ to c. It can be seen that $\frac{\partial b_n}{\partial d_i} = M_{ni}$, and $\frac{c}{\partial b_i} = a_i$. Then it can be written in a vector format, which ends with $\partial c/\partial d_i = \mathbf{a}^T\mathbf{M}_i$, where $\mathbf{M}_i$ is the i^{th} column of M. Thus, write the whole derivative in a vector and matrix format, $\partial c/\partial \mathbf{d} = (\mathbf{a}^T\mathrm{M})^T = \mathrm{M}^T\mathbf{a}$.

References

[1] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.