

We discuss the evaluation of $(\mathbf{v}^i)^T P_{\phi}^m$

$$(\mathbf{v}^i)^T P_{\phi}^m \mid$$

- In the backward process, the following operation is applied.

$$(\mathbf{v}^i)^T P_{\phi}^m, \quad (1)$$

where

$$\mathbf{v}^i = \text{vec} \left((W^m)^T \frac{\partial \xi_i}{\partial S^{m,i}} \right) \quad (2)$$

- Consider the same example used for explaining $\phi(Z^{\text{in},i})$

$(v^i)^T P_{\phi}^m \parallel$

- We have

$$P_{\phi}^m = \begin{bmatrix} 1 & & & & \\ & 1 & & & \\ & & 1 & & \\ & & & 1 & \\ & 1 & & & \\ & & 1 & & \\ & & & 1 & \\ & & & & 1 \end{bmatrix}$$

$(\mathbf{v}^i)^T P_{\phi}^m$ III

- Thus

$$(\mathbf{v}^i)^T P_{\phi}^m = [v_1 \quad v_2 + v_5 \quad v_6 \quad v_3 \quad v_4 + v_7 \quad v_8], \quad (3)$$

which is a kind of “inverse” operation of $\phi(\text{pad}(Z^{m,i}))$

- We accumulate elements in $\phi(\text{pad}(Z^{m,i}))$ back to their original positions in $\text{pad}(Z^{m,i})$.

$(\mathbf{v}^i)^T P_{\phi}^m$ IV

- In MATLAB, given indices

$$[1 \ 2 \ 4 \ 5 \ 2 \ 3 \ 5 \ 6]^T \quad (4)$$

and the vector $\mathbf{v}$, a function `accumarray` can directly generate the vector (3).

- Example:

$$(v^i)^T P_{\phi}^m V$$

```
octave:18> [v a]
```

```
ans =
```

1	0.406445
2	0.067872
4	0.036638
5	0.279801
2	0.490535
3	0.369743
5	0.429186
6	0.054324

$$(v^i)^T P_{\phi}^m VI$$

```
octave:19> accumarray(v,a)
ans =
```

0.406445

0.558407

0.369743

0.036638

0.708987

0.054324

$(\mathbf{v}^i)^T \mathbf{P}_{\phi}^m$ VII

- We can see that the second position is

$$\begin{aligned} & a(2) + a(5) \\ = & 0.067872 + 0.490535 \\ = & 0.558407 \end{aligned}$$

- To do the calculation over a batch of instances, we aim to have

$$\begin{bmatrix} (\mathbf{v}^1)^T \mathbf{P}_{\phi}^m \\ \vdots \\ (\mathbf{v}^I)^T \mathbf{P}_{\phi}^m \end{bmatrix}^T \Rightarrow \text{a vector} \begin{bmatrix} (\mathbf{P}_{\phi}^m)^T \mathbf{v}^1 \\ \vdots \\ (\mathbf{P}_{\phi}^m)^T \mathbf{v}^I \end{bmatrix} \quad (5)$$

$(\mathbf{v}^i)^T \mathbf{P}_{\phi}^m$ VIII

- We can apply MATLAB's accumarray on the vector

$$\begin{bmatrix} \mathbf{v}^1 \\ \vdots \\ \mathbf{v}^I \end{bmatrix}, \quad (6)$$

by giving the following indices as the input.

$$\begin{bmatrix} (4) \\ (4) + a_{\text{pad}}^m b_{\text{pad}}^m d^m \mathbf{1}_{h^m h^m d^m a_{\text{conv}}^m b_{\text{conv}}^m} \\ (4) + 2a_{\text{pad}}^m b_{\text{pad}}^m d^m \mathbf{1}_{h^m h^m d^m a_{\text{conv}}^m b_{\text{conv}}^m} \\ \vdots \\ (4) + (I - 1)a_{\text{pad}}^m b_{\text{pad}}^m d^m \mathbf{1}_{h^m h^m d^m a_{\text{conv}}^m b_{\text{conv}}^m} \end{bmatrix}, \quad (7)$$

$$(\mathbf{v}^i)^T P_{\phi}^m \mathbf{X}$$

where

$a_{\text{pad}}^m b_{\text{pad}}^m d^m$ is the size of $\text{pad}(Z^{m,i})$

and

$h^m h^m d^m a_{\text{conv}}^m b_{\text{conv}}^m$ is the size of $\phi(\text{pad}(Z^{m,i}))$ and $\mathbf{v}_i$.

- That is, by using the offset $(i-1)a_{\text{pad}}^m b_{\text{pad}}^m d^m$, accumarray accumulates $\mathbf{v}^i$ to the following positions:

$$(i-1)a_{\text{pad}}^m b_{\text{pad}}^m d^m + 1, \dots, ia_{\text{pad}}^m b_{\text{pad}}^m d^m. \quad (8)$$

$$(\mathbf{v}^i)^T P_{\phi}^m X$$

- (7) can be easily obtained by the following outer sum

$$\text{vec}((4) + [0 \ \dots \ I - 1] a_{\text{pad}}^m b_{\text{pad}}^m d^m)$$

- To obtain

$$\begin{bmatrix} \mathbf{v}^1 \\ \vdots \\ \mathbf{v}^I \end{bmatrix}$$

we note from (2) that it is the same as

$$\text{vec} \left((W^m)^T \left[\frac{\partial \xi_1}{\partial S^{m,1}} \ \dots \ \frac{\partial \xi_I}{\partial S^{m,I}} \right] \right) \cdot \quad (9)$$

$$(v^i)^T P_{\phi}^m X$$

- Thus we do a matrix-matrix multiplication
- From (9), we have a reason that in our implementation

$$\frac{\partial \xi_i}{\partial \text{vec}(S^{m,i})^T}$$

over a batch of instances are stored in the form of

$$\left[\frac{\partial \xi_1}{\partial S^{m,1}} \quad \cdots \quad \frac{\partial \xi_l}{\partial S^{m,l}} \right] \in R^{d^{m+1} \times a_{\text{conv}}^m b_{\text{conv}}^m l}.$$

A Simple Code I

```
a_prev = model.ht_pad(m);  
b_prev = model.wd_pad(m);  
d_prev = model.ch_input(m);  
  
idx = net.idx_phiZm(:) +  
      [0:num_v-1]*d_prev*a_prev*b_prev;  
vTP = accumarray(idx(:), V(:),  
                  [d_prev*a_prev*b_prev*num_v 1])';
```

A Simple Code II

- Here we assume

$$V = [\mathbf{v}_1 \ \cdots \ \mathbf{v}_I]$$

and `num_v` is the number of columns

- Note that the third parameter of `accumarray` is to specify the size of the resulting vector as some entries may not accumulate any value (so we have zero there)

Discussion I

- If a package provides efficient implementations of the following operations
 - matrix-matrix products
 - matrix expansion for $\phi(\text{pad}(Z^{m,i}))$
 - outer sum
 - `accumarray`

then we can easily have a good CNN implementation

- Unfortunately, the difficulty to optimize these operations may vary

Discussion II

- To work on instances together, it's difficult to decide the best storage settings
- Further, storage settings affect the implementations