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

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

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

$$(\boldsymbol{v}^i)^T P_\phi^m, \tag{1}$$

  where

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

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

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

- We have

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

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

- Thus

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

  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 $\text{pad}(Z^{m,i})$.

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

- In MATLAB, given indices

$$\begin{bmatrix} 1 & 2 & 4 & 5 & 2 & 3 & 5 & 6 \end{bmatrix}^T \tag{4}$$

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

- Example:

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

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

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

   0.406445
   0.558407
   0.369743
   0.036638
   0.708987
   0.054324
```

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

$$
\begin{bmatrix} (\boldsymbol{v}^1)^T P_\phi^m \\ \vdots \\ (\boldsymbol{v}^l)^T P_\phi^m \end{bmatrix}^T \Rightarrow \text{a vector} \begin{bmatrix} (P_\phi^m)^T \boldsymbol{v}^1 \\ \vdots \\ (P_\phi^m)^T \boldsymbol{v}^l \end{bmatrix} \tag{5}
$$

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

- We can apply MATLAB's `accumarray` on the vector

$$\begin{bmatrix} \boldsymbol{v}^1 \\ \vdots \\ \boldsymbol{v}^l \end{bmatrix}, \tag{6}$$

by giving the following indices as the input.

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

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

where

$$a^m_{\text{pad}} b^m_{\text{pad}} d^m \text{ is the size of pad}(Z^{m,i})$$

and

$$h^m h^m d^m a^m_{\text{conv}} b^m_{\text{conv}} \text{ is the size of } \phi(\text{pad}(Z^{m,i})) \text{ and } \boldsymbol{v}_i.$$

- That is, by using the offset $(i-1)a^m_{\text{pad}} b^m_{\text{pad}} d^m$, `accumarray` accumulates $\boldsymbol{v}^i$ to the following positions:

$$(i-1)a^m_{\text{pad}} b^m_{\text{pad}} d^m + 1, \ \ldots, \ i a^m_{\text{pad}} b^m_{\text{pad}} d^m. \tag{8}$$

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

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

$$\text{vec}((4) + \begin{bmatrix} 0 & \dots & l-1 \end{bmatrix} a_{\text{pad}}^m b_{\text{pad}}^m d^m)$$

- To obtain

$$\begin{bmatrix} \boldsymbol{v}^1 \\ \vdots \\ \boldsymbol{v}^l \end{bmatrix}$$

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

$$\text{vec}\left((W^m)^T \begin{bmatrix} \frac{\partial \xi_1}{\partial S^{m,1}} & \dots & \frac{\partial \xi_l}{\partial S^{m,l}} \end{bmatrix}\right). \quad (9)$$

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

- 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

$$\begin{bmatrix} \frac{\partial \xi_1}{\partial S^{m,1}} & \ldots & \frac{\partial \xi_l}{\partial S^{m,l}} \end{bmatrix} \in R^{d^{m+1} \times a^m_{\text{conv}} b^m_{\text{conv}} l}.$$

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

$$V = \begin{bmatrix} \boldsymbol{v}_1 & \cdots & \boldsymbol{v}_l \end{bmatrix}$$

  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