Supplementary Materials for "Newton Methods for Convolutional Neural Networks"

Chien-Chih Wang, Kent Loong Tan, Chih-Jen Lin
Department of Computer Science, National Taiwan University, Taipei 10617, Taiwan

I List of Symbols

Notation	Description
$\boldsymbol{y}^i$	The label vector of the ith training instance.
$Z^{1,i}$	The input image of the ith training instance.
l	The number of training instances.
K	The number of classes.
$\boldsymbol{\theta}$	The model vector (weights and biases) of the neural network.
ξ	The loss function.
ξ_i	The training loss of the ith instance.
f	The objective function.
C	The regularization parameter.
L	The number of layers of the neural network.
L^c	The number of convolutional layers of the neural network.
L^f	The number of fully-connected layers of the neural network.
n_m	The number of neurons in the mth layer ($L^c < m \leq L+1$).
n	The total number of weights and biases.
a^m	Height of the input image at the mth layer ($1 \leq m \leq L^c$).
a^m_{pad}	Height of the image after padding at the mth layer ($1 \leq m \leq L^c$).
a^m_{conv}	Height of the image after convolution at the mth layer ($1 \leq m \leq L^c$).
b^m	Width of the input image at the mth layer ($1 \leq m \leq L^c$).
b^m_{pad}	Width of the image after padding at the mth layer ($1 \leq m \leq L^c$).
b^m_{conv}	Width of the image after convolution the mth layer ($1 \leq m \leq L^c$).
d^m	Depth (or the number of channels) of the data at the mth layer ($1 \leq m \leq L^c$).
h^m	Height (width) of the filters at the mth layer ($1 \leq m \leq L^c$).
W^m	The weight matrix in the mth layer.
$\boldsymbol{b}^m$	The bias vector in the mth layer.
$S^{m,i}$	The result of $(W^m)^T\phi(\text{pad}(Z^{m,i})) + \boldsymbol{b}^m \mathbb{1}^T_{a^m b^m}$ in the mth layer for the ith instance ($1 \leq m \leq L^c$).
$Z^{m+1,i}$	The output matrix (element-wise application of the activation function on $S^{m,i}$) in the mth layer for the ith instance ($1 \leq m \leq L^c$).
$\boldsymbol{s}^{m,i}$	The result of $(W^m)^T\boldsymbol{z}^{m,i} + \boldsymbol{b}^m$ in the mth layer for the ith instance ($L^c < m \leq L$).

Notation	Description
$\boldsymbol{z}^{m+1,i}$	The output vector (element-wise application of the activation function on $\boldsymbol{s}^{m,i}$) in the mth layer for the ith instance ($L^c \le m \le L$).
σ	The activation function.
J^i	The Jacobian matrix of $\boldsymbol{z}^{L+1,i}$ with respect to $\boldsymbol{\theta}$.
$\mathcal{I}$	An identity matrix.
α	Step size in taking a direction to update $\boldsymbol{\theta}$.
ρ	The ratio between the actual and the predicted function value reduction.
λ	A parameter in the Levenberg-Marquardt method.

II Implementation Details

We show that with a careful design, a Newton method for CNN can be implemented by a simple and short program. A *MATLAB* implementation is given as an illustration though modifications for other languages such as *Python* should be straightforward.

For the discussion in Section 3, we check each individual data. However, for practical implementations, all instances must be considered together for memory and computational efficiency. In our implementation, we store $Z^{m,i},\ \forall i = 1, \dots, l$ as the following matrix.

$$\begin{bmatrix} Z^{m,1} & Z^{m,2} & \dots & Z^{m,l} \end{bmatrix} \in R^{d^m \times a^m b^m l}. \tag{II.1}$$

Similarly, we store $\partial \xi_i / \partial \text{vec}(S^{m,i})^T,\ \forall i$ as

$$\begin{bmatrix} \frac{\partial \xi_1}{\partial S^{m,1}} & \dots & \frac{\partial \xi_l}{\partial S^{m,l}} \end{bmatrix} \in R^{d^{m+1} \times a^m_{\text{conv}} b^m_{\text{conv}} l}. \tag{II.2}$$

For $\partial \boldsymbol{z}^{L+1,i} / \partial \text{vec}(S^{m,i})^T,\ \forall i$, we consider

$$\begin{bmatrix} \frac{\partial z_1^{L+1,1}}{\partial S^{m,1}} & \dots & \frac{\partial z_{n_{L+1}}^{L+1,1}}{\partial S^{m,1}} & \dots & \frac{\partial z_{n_{L+1}}^{L+1,l}}{\partial S^{m,l}} \end{bmatrix} \in R^{d^{m+1} \times a^m_{\text{conv}} b^m_{\text{conv}} n_{L+1} l} \tag{II.3}$$

and will explain our decision. Note that (II.1)-(II.3) are only the main setting to store these matrices because for some operations they may need to be re-shaped.

For an easy description in some places we follow Section 2.1 to let

$$Z^{\text{in},i} \text{ and } Z^{\text{out},i}$$

be the input and output images of a layer, respectively.

II.1 Details of Padding Operation

To implement zero-padding, we first capture the linear indices of the input image in the padded image. For example, if the size of the input image is 3×3 and the output padded image is 5×5, we have

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix},$$

Listing I: *MATLAB* implementation for the index of zero-padding

```
function [idx_pad] = find_index_padding(model,m)

a = model.ht_input(m);
b = model.wd_input(m);
p = model.wd_pad_added(m);

newa = 2*p + a;
idx_pad = reshape( (p+1:p+a)' + newa*(p:p+b-1), [], 1);
```

where "1" values indicate positions of the input image. Based on the column-major order, we derive

$$\texttt{pad\_idx} = \{7,\ 8,\ 9,\ 11,\ 12,\ 13,\ 16,\ 17,\ 18\}.$$

This index set, obtained in the beginning of the training procedure, is used in the following situations. First, `pad_idx` contains row indices in P^m_{pad} of (14) that correspond to the input image. We can use it to conduct the padding operation in (14). Second, from (56) and (66) in gradient and Jacobian evaluations, we need

$$\boldsymbol{v}^T P^m_{\text{pad}}.$$

This can be considered as the inverse of the padding operation: we would like to remove zeros and get back the original image. We give details of finding `pad_idx` below.

Assume the input image is

$$Z \in R^{a\times b}.$$

We would like to add p zeros on each dimension so that the resulting image is as Figure II.1. We notice that Z corresponds to the following elements in the output image:

$$\begin{matrix} (p+1,p+1) & \dots & (p+1,p+b) \\ & \vdots & \\ (p+a,p+1) & \dots & (p+a,p+b) \end{matrix}$$

The size of the new image is

$$(2p+a)\times(2p+b).$$

The linear indices in the new matrix are

$$p\bar{a} + \begin{bmatrix} p+1 \\ \vdots \\ p+a \end{bmatrix},\ (p+1)\bar{a} + \begin{bmatrix} p+1 \\ \vdots \\ p+a \end{bmatrix},\ \dots,\ (p+b-1)\bar{a} + \begin{bmatrix} p+1 \\ \vdots \\ p+a \end{bmatrix},$$

where

$$\bar{a} = 2p + a. \tag{II.4}$$

Together they can be obtained by applying *MATLAB*'s '+' operator on the following two arrays:

$$\begin{bmatrix} p+1 \\ \vdots \\ p+a \end{bmatrix} \text{ and } \bar{a}\begin{bmatrix} p & \cdots & p+b-1 \end{bmatrix}.$$

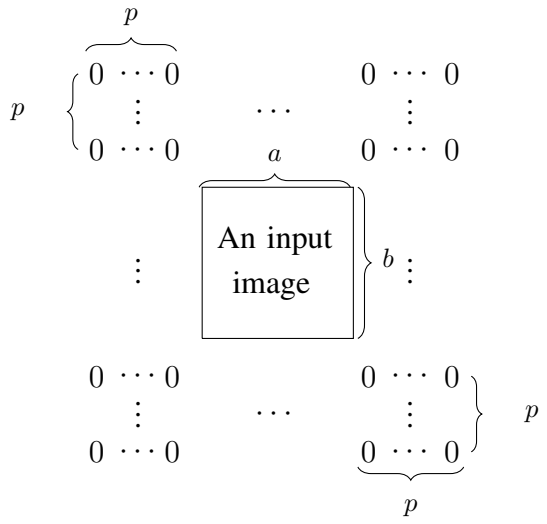

Figure II.1: A padding example.

II.2 Generation of $\phi(\text{pad}(Z^{m,i}))$

MATLAB has a built-in function `im2col` that can generate $\phi(\text{pad}(Z^{m,i}))$ for $s = 1$ and $s = h$. For general s, we notice that $\phi(\text{pad}(Z^{m,i}))$ is a sub-matrix of the output matrix of using *MATLAB*'s `im2col` under $s = 1$. Therefore, we can apply *MATLAB*'s `im2col` with $s = 1$ and extract a sub-matrix as $\phi(\text{pad}(Z^{m,i}))$. See a detailed procedure in Section II.2.1.

The above setting is not ideal because first in other languages a subroutine like *MATLAB*'s `im2col` may not be available, and second, generating a larger matrix under $s = 1$ causes extra time and memory. Therefore, here we show an efficient implementation without relying on a subroutine like *MATLAB*'s `im2col`. For an easy description we follow Section 2.1 to consider

$$\text{pad}(Z^{m,i}) = Z^{\text{in},i} \rightarrow Z^{\text{out},i} = \phi(Z^{\text{in},i}).$$

Consider the following linear indices[1] (i.e., counting elements in a column-oriented way) of $Z^{\text{in},i}$:

$$\begin{bmatrix} 1 & d^{\text{in}}+1 & \dots & (b^{\text{in}}a^{\text{in}}-1)d^{\text{in}}+1 \\ 2 & d^{\text{in}}+2 & \dots & (b^{\text{in}}a^{\text{in}}-1)d^{\text{in}}+2 \\ \vdots & \vdots & \ddots & \vdots \\ d^{\text{in}} & 2d^{\text{in}} & \dots & (b^{\text{in}}a^{\text{in}})d^{\text{in}} \end{bmatrix} \in R^{d^{\text{in}} \times a^{\text{in}} b^{\text{in}}}. \tag{II.5}$$

Because every element in

$$\phi(Z^{\text{in},i}) \in R^{hhd^{\text{in}} \times a^{\text{out}} b^{\text{out}}},$$

is extracted from $Z^{\text{in},i}$, the task is to find the mapping between a linear index of $Z^{\text{in},i}$ and each element in $\phi(Z^{\text{in},i})$. Consider the following example.

$$a^{\text{in}} = 3,\ b^{\text{in}} = 2,\ d^{\text{in}} = 1,\ s = 1,\ h = 2.$$

[1]Linear indices refer to the sequence of how elements in a matrix are stored. Here we consider a column-oriented setting.

Because $d^{\text{in}} = 1$, we omit the channel subscript. In addition, we omit the instance index i, so the image is

$$\begin{bmatrix} z_{11} & z_{12} \\ z_{21} & z_{22} \\ z_{31} & z_{32} \end{bmatrix}.$$

By our representation in (5),

$$Z^{\text{in}} = \begin{bmatrix} z_{11} & z_{21} & z_{31} & z_{12} & z_{22} & z_{32} \end{bmatrix}$$

and the linear indices from (II.5) are

$$\begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 \end{bmatrix}.$$

From (9),

$$\phi(Z^{\text{in}}) = \begin{bmatrix} z_{11} & z_{21} \\ z_{21} & z_{31} \\ z_{12} & z_{22} \\ z_{22} & z_{32} \end{bmatrix}.$$

Thus we store the following vector to indicate the mapping between linear indices of Z^{in} and elements in $\phi(Z^{\text{in}})$.

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

It also corresponds to column indices of non-zero elements in P_ϕ^m.

To have a general setting we begin with checking how linear indices of $Z^{\text{in},i}$ can be mapped to the first column of $\phi(Z^{\text{in},i})$. For simplicity, we consider only channel j. From (9) and (II.5), we have

$$\left[\begin{array}{c|c} j & z^{\text{in}}_{1,1,j} \\ d^{\text{in}} + j & z^{\text{in}}_{2,1,j} \\ \vdots & \vdots \\ (h-1)d^{\text{in}} + j & z^{\text{in}}_{h,1,j} \\ a^{\text{in}}d^{\text{in}} + j & z^{\text{in}}_{1,2,j} \\ \vdots & \vdots \\ ((h-1) + a^{\text{in}})d^{\text{in}} + j & z^{\text{in}}_{h,2,j} \\ \vdots & \vdots \\ ((h-1) + (h-1)a^{\text{in}})d^{\text{in}} + j & z^{\text{in}}_{h,h,j} \end{array}\right], \tag{II.7}$$

where the left column gives the linear indices in $Z^{\text{in},i}$, while the right column shows the

corresponding values. We rewrite linear indices in (II.7) as

$$\begin{bmatrix} 0+0a^{\text{in}} \\ \vdots \\ (h-1)+0a^{\text{in}} \\ 0+1a^{\text{in}} \\ \vdots \\ (h-1)+1a^{\text{in}} \\ \vdots \\ 0+(h-1)a^{\text{in}} \\ \vdots \\ (h-1)+(h-1)a^{\text{in}} \end{bmatrix} d^{\text{in}}+j. \tag{II.8}$$

Clearly, every linear index in (II.8) can be represented as

$$(p+qa^{\text{in}})d^{\text{in}}+j, \text{ where } p,\ q \in \{0,\dots,h-1\} \tag{II.9}$$

correspond to the pixel position in the convolutional filter.[2]

Next we consider other columns in $\phi(Z^{\text{in},i})$ by still fixing the channel to be j. From (9), similar to the right column in (II.7), each column contains the following elements from the jth channel of $Z^{\text{in},i}$.

$$\begin{aligned} z^{\text{in},i}_{1+p+as,1+q+bs,j}\ ,\ & a=0,1,\dots,a^{\text{out}}-1, \\ & b=0,1,\dots,b^{\text{out}}-1, \end{aligned} \tag{II.10}$$

where $(1+as,\ 1+bs)$ denotes the top-left position of a sub-image in the channel j of $Z^{\text{in},i}$. From (II.5), the linear index of each element in (II.10) is

$$\begin{aligned} & ((1+p+as-1)+(1+q+bs-1)a^{\text{in}})d^{\text{in}}+j \\ =\ & (a+ba^{\text{in}})sd^{\text{in}}+\underbrace{(p+qa^{\text{in}})d^{\text{in}}+j}_{\text{see (II.9)}}. \end{aligned} \tag{II.11}$$

Listing II shows our implementation about finding the mapping of the linear indices of each element and Listing IV shows the generation of $\phi(\text{pad}(Z^{m,i}))$. First, we compute elements in (II.8) with $j=1$ by applying *MATLAB*'s '+' operator, which has the implicit expansion behavior, to compute the outer sum of the following two arrays.

$$\begin{bmatrix} 1 \\ d^{\text{in}}+1 \\ \vdots \\ (h-1)d^{\text{in}}+1 \end{bmatrix} \quad \text{and} \quad \begin{bmatrix} 0 & a^{\text{in}}d^{\text{in}} & \dots & (h-1)a^{\text{in}}d^{\text{in}} \end{bmatrix}.$$

The result is the following matrix

$$\begin{bmatrix} 1 & a^{\text{in}}d^{\text{in}}+1 & \dots & (h-1)a^{\text{in}}d^{\text{in}}+1 \\ d^{\text{in}}+1 & (1+a^{\text{in}})d^{\text{in}}+1 & \dots & (1+(h-1)a^{\text{in}})d^{\text{in}}+1 \\ \vdots & \vdots & \dots & \vdots \\ (h-1)d^{\text{in}}+1 & ((h-1)+a^{\text{in}})d^{\text{in}}+1 & \dots & ((h-1)+(h-1)a^{\text{in}})d^{\text{in}}+1 \end{bmatrix}, \tag{II.12}$$

[2]More precisely, $p+1$ and $q+1$ are the pixel position.

Listing II: *MATLAB* implementation for finding the mapping between the linear indices of $Z^{m,i}$ and elements in $\phi(Z^{\text{in},i})$

```
1 function idx = find_index_phiZ(a,b,d,h,s)
2
3 first_channel_idx = ([0:h-1]*d+1)' + [0:h-1]*a*d;
4 first_col_idx = first_channel_idx(:) + [0:d-1];
5 a_out = floor((a - h)/s) + 1;
6 b_out = floor((b - h)/s) + 1;
7 column_offset = ([0:a_out-1]' + [0:b_out-1]*a)*s*d;
8 idx = column_offset(:)' + first_col_idx(:);
9 idx = idx(:);
```

whose columns, if concatenated, lead to values in (II.8) with $j = 1$; see line 3 of the code. To get (II.9) for all channels $j = 1, \ldots, d^{\text{in}}$, we compute the outer sum of the vector form of (II.12) and

$$\begin{bmatrix} 0 & 1 & \ldots & d^{\text{in}} - 1 \end{bmatrix},$$

and then vectorize the resulting matrix; see line 4.

To obtain other columns in $\phi(Z^{\text{in},i})$, we first calculate a^{out} and b^{out} by (4) in lines 5-6. In the linear indices in (II.11), the second term corresponds to indices of the first column, while the first term is the following column offset

$$\begin{aligned}(a + ba^{\text{in}})sd^{\text{in}},\ \forall a = 0, 1, \ldots, a^{\text{out}} - 1,\\ b = 0, 1, \ldots, b^{\text{out}} - 1.\end{aligned}$$

This is the outer sum of the following two arrays.

$$\begin{bmatrix} 0 \\ \vdots \\ a^{\text{out}} - 1 \end{bmatrix} \times sd^{\text{in}} \quad \text{and} \quad \begin{bmatrix} 0 & \ldots & b^{\text{out}} - 1 \end{bmatrix} \times a^{\text{in}} sd^{\text{in}};$$

see line 7 in the code. Finally, we compute the outer sum of the column offset and the linear indices in the first column of $\phi(Z^{\text{in},i})$; see line 8. In the end what we keep is the following vector

$$\begin{bmatrix} \text{Column index of non-zero} \\ \text{in each row of } P_\phi^m \end{bmatrix}_{hhd^{\text{in}}a^{\text{out}}b^{\text{out}}}. \tag{II.13}$$

Note that each row in the $0/1$ matrix P_ϕ^m contains exactly only one non-zero element. We also see that (II.6) is an example of (II.13).

The obtained linear indices are independent of the values of $Z^{\text{in},i}$. Thus the above procedure only needs to be run once in the beginning. For any $Z^{\text{in},i}$, we apply the indices in (II.13) to extract $\phi(Z^{\text{in},i})$; see line 6-7 in Listing IV.

For the pooling operation $\phi(Z^{\text{in},i})$ is needed in (15). The same implementation can be used.

Listing III: An alternative implementation by using *MATLAB*'s `im2col` for finding the mapping between the linear indices from $Z^{m,i}$ and elements in $\phi(Z^{\text{in},i})$

```
function output_idx = find_index_phiZ(a,b,d,h,s)

input_idx = reshape(([1:a*b]-1)*d+1,a,b);
output_idx = im2col(input_idx,[h,h],'sliding');
a_bar = a-h+1;
b_bar = b-h+1;
a_idx = 1:s:a_bar;
b_idx = 1:s:b_bar;
select_idx = a_idx'+a_bar*(b_idx-1);
output_idx = output_idx(:,select_idx)';
output_idx = reshape(output_idx(:)+[0:d-1],[],h*h*d)';
```

Listing IV: *MATLAB* implementation for generating $\phi(Z^{\text{in},i})$

```
function phiZ = padding_and_phiZ(model, net, m)

num_data = net.num_sampled_data;
phiZ = padding(model, net, m);
% Calculate phiZ
phiZ = reshape(phiZ, [], num_data);
phiZ = phiZ(net.idx_phiZ{m}, :);

h = model.wd_filter(m);
d = model.ch_input(m);
phiZ = reshape(phiZ, h*h*d, []);
```

II.2.1 Generation of $\phi(Z^{\text{in},i})$ with *MATLAB*'s `im2col`

For the alternative method here, we use *MATLAB*'s `im2col` with $s = 1$ and extract a sub-matrix as $\phi(Z^{\text{in},i})$. The program is presented in Listing III.

We now explain each line of the program. To find P_ϕ^m, from (9) what we need is to extract elements in $Z^{\text{in},i}$. In line 3, we start with obtaining the linear indices of the first row of $Z^{\text{in},i}$, which corresponds to the first channel of the image. In line 4, we use `im2col` to build $\phi(Z^{\text{in},i})$ under $s = d^{\text{in}} = 1$, though contents of the input matrix are linear indices of $Z^{\text{in},i}$ rather than values. For $\phi(Z^{\text{in},i})$ under $s = d^{\text{in}} = 1$, the matrix size is

$$hh \times \bar{a}\bar{b},$$

where from (4),

$$\bar{a} = a^{\text{in}} - h + 1,\ \bar{b} = b^{\text{in}} - h + 1.$$

From (9), when a general s is considered, we must select some columns, whose column

indices are the following subset of $\{1, \dots, \bar{a}\bar{b}\}$:

$$\mathbb{1}_{b^{\text{out}}} \otimes \left(\begin{bmatrix} 0 \\ \vdots \\ a^{\text{out}} - 1 \end{bmatrix} s + \mathbb{1}_{a^{\text{out}}} \right) + \left(\begin{bmatrix} 0 \\ \vdots \\ b^{\text{out}} - 1 \end{bmatrix} s \right) \otimes \begin{bmatrix} \bar{a} \\ \vdots \\ \bar{a} \end{bmatrix}_{a^{\text{out}} \times 1}, \tag{II.14}$$

where a^{out} and b^{out} are defined in (4). More precisely, (II.14) comes from the following mapping between the first row of $\phi(Z^{\text{in},i})$ in (9) and $\{1, \dots, \bar{a}\bar{b}\}$:

$$\begin{bmatrix} (1,1) \\ (1+s,1) \\ \vdots \\ (1+(a^{\text{out}}-1)s,1) \\ (1,1+s) \\ (1+s,1+s) \\ \vdots \\ (1+(a^{\text{out}}-1)s,1+s) \\ \vdots \end{bmatrix} \longrightarrow \begin{bmatrix} \begin{bmatrix} 0 \\ \vdots \\ a^{\text{out}}-1 \end{bmatrix} s + \mathbb{1}_{a^{\text{out}}} + \mathbf{0}_{a^{\text{out}} s\bar{a}} \\ \begin{bmatrix} 0 \\ \vdots \\ a^{\text{out}}-1 \end{bmatrix} s + \mathbb{1}_{a^{\text{out}}} + \mathbb{1}_{a^{\text{out}} s\bar{a}} \\ \vdots \end{bmatrix}$$

Next we discuss how to extend the linear indices of the first channel to others. From (5), each column of $Z^{\text{in},i}$ contains values of the same pixel in different channels. Therefore, because we consider a column-major order, indices in $Z^{\text{in},i}$ for a given pixel are a continuous segment. Then in (9) for $\phi(Z^{\text{in},i})$, essentially we have d^{in} segments ordered vertically and elements in two consecutive segments are from two consecutive rows in $Z^{\text{in},i}$. Therefore, the following index matrix can be used to extract all needed elements in $Z^{\text{in},i}$ for $\phi(Z^{\text{in},i})$.

$$\mathbb{1}_{d^{\text{in}}} \otimes \begin{bmatrix} \text{linear indices of } Z^{\text{in},i} \text{ for} \\ \text{1st channel of } \phi(Z^{\text{in},i}) \end{bmatrix}_{hh \times a^{\text{out}} b^{\text{out}}} + \left(\begin{bmatrix} 0 \\ \vdots \\ d^{\text{in}} - 1 \end{bmatrix} \otimes \mathbb{1}_{hh} \right) \otimes \mathbb{1}^T_{a^{\text{out}} b^{\text{out}}}. \tag{II.15}$$

The implementation is in line 11. Since *OCTAVE* does not have the function `repelem`,[3] we use different approach to calculate (II.15). First we calculate

$$\text{vec}\left(\begin{bmatrix} \text{linear indices of } Z^{\text{in},i} \text{ for} \\ \text{1st channel of } \phi(Z^{\text{in},i}) \end{bmatrix}^T \right) + \begin{bmatrix} 0 & \dots & d^{\text{in}} - 1 \end{bmatrix} \in R^{a^{\text{out}} b^{\text{out}} hh \times d^{\text{in}}}. \tag{II.16}$$

Then, to obtain the desired matrix, we reshape (II.16) into $R^{a^{\text{out}} b^{\text{out}} \times hh d^{\text{in}}}$ and transpose it.

Similar to the previous section, we apply the indices in line 11 to extract $\phi(Z^{\text{in},i})$;

II.3 Construction of $P^{m,i}_{\text{pool}}$

Following (18), we use $Z^{\text{in},i}$ and $Z^{\text{out},i}$ to represent the input

$$\sigma(S^{m,i}) \in R^{d^{m+1} \times a^m_{\text{conv}} b^m_{\text{conv}}}$$

[3]The `repelem` in *MATLAB* is syntax incompatible with the `repelems` in *OCTAVE*.

and the output

$$Z^{m+1,i} \in R^{d^{m+1} \times a^{m+1} b^{m+1}}$$

of the pooling operation, respectively. We need to store $P_{\text{pool}}^{m,i}$ because, besides function evaluations, it is used in gradient and Jacobian evaluations; see (54) and (65).[4] From (19), we need both $P_{\phi}^{m,i}$ and $W^{m,i}$. Because $P_{\phi}^{m,i}$ is for partitioning each image to non-overlapping sub-regions in (15) and (16), it is iteration independent. We obtain it in the beginning of the training procedure by the method in Section II.2.

For $W^{m,i}$, it is iteration dependent because the maximal value of each sub-image is not a constant. Therefore, we construct

$$P_{\text{pool}}^{m,i} = W^{m,i} P_{\phi}^{m,i} \in R^{d^{m+1} a^{m+1} b^{m+1} \times d^{m+1} a_{\text{conv}}^{m} b_{\text{conv}}^{m}}$$

at the beginning of each Newton iteration.

In Section II.3, we have discussed why $P_{\text{pool}}^{m,i}$ is needed to be constructed. Here we give Listing V with explanation in details. To begin, we get

$$Z^{\text{in},i},\ i = 1, \ldots, l, \tag{II.17}$$

which are stored as a matrix in (II.1). Because (16) may not hold with a^{out} and b^{out} being integers, we consider a setting the same as (4). In line 11, we extract the linear indices of $Z^{\text{in},i}$ to appear in $\text{vec}(\phi(Z^{\text{in},i}))$, which as we mentioned has been generated in the beginning of the training procedure. The resulting vector `P` contains

$$hhd^{m+1} a^{m+1} b^{m+1}$$

elements and each element is in the range of

$$1,\ \ldots,\ d^{m+1} a_{\text{conv}}^{m} b_{\text{conv}}^{m}.$$

In line 12-13, we use `P` to generate

$$\left[\text{vec}(\phi(Z^{\text{in},1}))\ \cdots\ \text{vec}(\phi(Z^{\text{in},l}))\right]\ \in R^{hhd^{m+1} a^{m+1} b^{m+1} \times l}. \tag{II.18}$$

Next we rewrite the above matrix so that each column contains a sub-region:

$$\begin{bmatrix} z_{1,1,1}^{m,1} & z_{1,1,2}^{m,1} & \cdots & z_{1+(a^{m+1}-1)\times s, 1+(b^{m+1}-1)\times s, d^{m+1}}^{m,l} \\ \vdots & \vdots & \ddots & \vdots \\ z_{h,h,1}^{m,1} & z_{h,h,2}^{m,1} & \cdots & z_{h+(a^{m+1}-1)\times s, h+(b^{m+1}-1)\times s, d^{m+1}}^{m,l} \end{bmatrix} \in R^{hh \times d^{m+1} a^{m+1} b^{m+1} l}. \tag{II.19}$$

We apply a max function to get the largest value of each column and its index in the range of $1, \ldots, hh$. The resulting row vector has $d^{m+1} a^{m+1} b^{m+1} l$ elements; see line 14. In line 15, we reformulate it to be

$$d^{m+1} \times a^{m+1} b^{m+1} l$$

[4]Note that we do not really generate a sparse matrix $P_{\text{pool}}^{m,i}$ in (18). We only store column indices of non-zeros in $P_{\text{pool}}^{m,i}$.

as the output $Z^{\text{out},i},\ \forall i$.

Next we find linear indices that correspond to the largest elements obtained from (II.19). Because of operations discussed in Section II.4.3 and II.4.4, we decide to record linear indices in each $Z^{\text{in},i}$ corresponding to the selected elements, rather than linear indices in the whole matrix (II.1) of all $Z^{m,i},\ \forall i$. We begin with obtaining the following vector of linear indices of $Z^{\text{in},i}$:

$$\begin{bmatrix} 1 \\ \vdots \\ d^{m+1}a^m_{\text{conv}}b^m_{\text{conv}} \end{bmatrix}. \tag{II.20}$$

Then we generate

$$\phi((\text{II.20})), \tag{II.21}$$

which has $hhd^{m+1}a^{m+1}b^{m+1}$ elements; see line 22. Next, we mentioned that in line 14, not only the maximal value in each sub-region is obtained, but also the corresponding index in $\{1,\dots,hh\}$ is derived. Therefore, for the selected max values of all instances, their positions in the range of

$$1,\ \dots, hhd^{m+1}a^{m+1}b^{m+1}$$

are

$$\text{mat}\left(\begin{bmatrix}\text{row indices of}\\ \text{max values in (II.19)}\end{bmatrix}\right)_{d^{m+1}a^{m+1}b^{m+1}\times l} + hh\left(\begin{bmatrix} 0 \\ \vdots \\ d^{m+1}a^{m+1}b^{m+1}-1 \end{bmatrix}\otimes \mathbb{1}_l^T\right); \tag{II.22}$$

see line 19. Next in line 23 we use (II.22) to extract values in (II.21) and obtain linear indices of the selected max values in each $Z^{\text{in},i}$. To be more precise, the resulting matrix is

$$\begin{bmatrix}\text{Column index of non-zero}\\ \text{in each row of } P^{m,1}_{\text{pool}}\end{bmatrix} \cdots \begin{bmatrix}\text{Column index of non-zero}\\ \text{in each row of } P^{m,l}_{\text{pool}}\end{bmatrix} \in R^{d^{m+1}a^{m+1}b^{m+1}\times l}. \tag{II.23}$$

The reason is that because $P^{m,i}_{\text{pool}}$ is a $0/1$ matrix and each row contains exactly only one value "1" to indicate the selected entry by max pooling, we collects the column index of the non-zero at each row to be a vector for future use.

II.4 Evaluation of $(\boldsymbol{v}^i)^T P^m_\phi$ and $(\boldsymbol{v}^i)^T P^{m,i}_{\text{pool}}$ in Gradient and Jacobian Evaluations

We show that several operations in gradient and Jacobian evaluations are either

$$(\boldsymbol{v}^i)^T P^m_\phi \quad \text{or} \quad (\boldsymbol{v}^i)^T P^{m,i}_{\text{pool}},$$

where $\boldsymbol{v}^i$ is a vector. Here we give Listing VI with details explained.

Listing V: *MATLAB* implementation for $P_{\text{pool}}^{m,i}$

```
function [Zout, idx_pool] = maxpooling(model, net, m)

a = model.ht_conv(m);
b = model.wd_conv(m);
d = model.ch_input(m+1);
h = model.wd_subimage_pool(m);

% Z input: sigma(S_m)
Z = net.Z{m+1};

P = net.idx_phiZ_pool{m};
Z = reshape(Z, d*a*b, []);
Z = Z(P, :);
[Z, max_id] = max(reshape(Z, h*h, []));
Zout = reshape(Z, d, []);

outa = model.ht_input(m+1);
outb = model.wd_input(m+1);
max_id = reshape(max_id, d*outa*outb, []) + h*h*[0:d*outa*outb
    -1]';

idx_pool = [1:d*a*b];
idx_pool = idx_pool(P);
idx_pool = idx_pool(max_id);
```

II.4.1 Evaluation of $(\boldsymbol{v}^i)^T P_\phi^m$ in Gradient Evaluations

For (56) and (66), the following operation is applied.

$$(\boldsymbol{v}^i)^T P_\phi^m, \tag{II.24}$$

where

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

for (56) and

$$\boldsymbol{v}_u^i = \text{vec}\left((W^m)^T \frac{\partial z_u^{L+1,i}}{\partial S^{m,i}}\right),\ u = 1, \ldots, n_{L+1} \tag{II.25}$$

for (66).

Consider the same example in Section II.2. We note that

$$(P_\phi^m)^T \boldsymbol{v}^i = [\, v_1 \ \ v_2 + v_5 \ \ v_6 \ \ v_3 \ \ v_4 + v_7 \ \ v_8 \,]^T, \tag{II.26}$$

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 $\text{pad}(Z^{m,i})$. In *MATLAB*, given indices in (II.6), a function `accumarray` can directly generate the vector (II.26).

To calculate (56) over a batch of instances, we aim to have

$$\begin{bmatrix} (P_\phi^m)^T \boldsymbol{v}^1 \\ \vdots \\ (P_\phi^m)^T \boldsymbol{v}^l \end{bmatrix}^T. \tag{II.27}$$

We can manage to apply *MATLAB*'s `accumarray` on the vector

$$\begin{bmatrix} \boldsymbol{v}^1 \\ \vdots \\ \boldsymbol{v}^l \end{bmatrix}, \tag{II.28}$$

by giving the following indices as the input.

$$\begin{bmatrix} \text{(II.13)} \\ \text{(II.13)} + a_{\text{pad}}^m b_{\text{pad}}^m d^m \mathbb{1}_{h^m h^m d^m a_{\text{conv}}^m b_{\text{conv}}^m} \\ \text{(II.13)} + 2a_{\text{pad}}^m b_{\text{pad}}^m d^m \mathbb{1}_{h^m h^m d^m a_{\text{conv}}^m b_{\text{conv}}^m} \\ \vdots \\ \text{(II.13)} + (l-1) a_{\text{pad}}^m b_{\text{pad}}^m d^m \mathbb{1}_{h^m h^m d^m a_{\text{conv}}^m b_{\text{conv}}^m} \end{bmatrix}, \tag{II.29}$$

where from Section 2.1.3,

$a_{\text{pad}}^m b_{\text{pad}}^m d^m$ 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 $\boldsymbol{v}_i$.

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

$$(i-1)a_{\text{pad}}^m b_{\text{pad}}^m d^m + 1, \ \ldots, \ i a_{\text{pad}}^m b_{\text{pad}}^m d^m. \tag{II.30}$$

To obtain (II.28), we can do a matrix-matrix multiplication as follows.

$$\text{(II.28)} = \text{vec}\left((W^m)^T \begin{bmatrix} \frac{\partial \xi_1}{\partial S^{m,1}} & \ldots & \frac{\partial \xi_l}{\partial S^{m,l}} \end{bmatrix} \right). \tag{II.31}$$

From (II.31), we can see why $\partial \xi_i / \partial \text{vec}(S^{m,i})^T$ over a batch of instances are stored in the form of (II.2). In line 26, the indices shown in (II.29) are generated and the variable `V(:)` in line 31 is the vector (II.28) calculated by (II.31).

II.4.2 Evaluation of $(\boldsymbol{v}^i)^T P_\phi^m$ in Jacobian Evaluations

To calculate (66) over a batch of instances, similar to (II.27) we conduct

$$\begin{bmatrix} (P_\phi^m)^T \boldsymbol{v}_1^1 \\ \vdots \\ (P_\phi^m)^T \boldsymbol{v}_{n_{L+1}}^1 \\ \vdots \\ (P_\phi^m)^T \boldsymbol{v}_{n_{L+1}}^l \end{bmatrix}^T, \text{ where } \boldsymbol{v}_u^i = \text{vec}\left((W^m)^T \frac{\partial z_u^{L+1,i}}{\partial S^{m,i}} \right) \in R^{h^m h^m d^m a_{\text{conv}}^m b_{\text{conv}}^m \times 1}, \ u = 1, \ldots, n_{L+1}. \tag{II.32}$$

Similar to (II.31), we can calculate the vector

$$\begin{bmatrix} \boldsymbol{v}_1^1 \\ \vdots \\ \boldsymbol{v}_{n_{L+1}}^1 \\ \vdots \\ \boldsymbol{v}_{n_{L+1}}^l \end{bmatrix} \tag{II.33}$$

by

$$\text{vec}\left((W^m)^T \begin{bmatrix} \frac{\partial z_1^{L+1,1}}{\partial S^{m,1}} & \cdots & \frac{\partial z_{n_{L+1}}^{L+1,1}}{\partial S^{m,1}} & \cdots & \frac{\partial z_{n_{L+1}}^{L+1,l}}{\partial S^{m,l}} \end{bmatrix}\right). \tag{II.34}$$

The formulation in (II.34) leads us to store

$$\frac{\partial \boldsymbol{z}^{L+1,i}}{\partial \text{vec}(S^{m,i})},\ \forall i$$

in the form of (II.3).

From (II.32), because each vector $\boldsymbol{v}_u^i$ is accumulated to the following positions:

$$((i-1)n_{L+1}+(u-1))a_{\text{pad}}^m b_{\text{pad}}^m d^m+1,\ \ldots,\ ((i-1)n_{L+1}+u)a_{\text{pad}}^m b_{\text{pad}}^m d^m,$$

we can apply `accumarray` on the vector (II.33) with the following input indices.

$$\begin{bmatrix} \text{(II.13)} \\ \vdots \\ \text{(II.13)} + (n_{L+1}-1)d^m a_{\text{pad}}^m b_{\text{pad}}^m \mathbb{1}_{h^m h^m d^m a_{\text{conv}}^m b_{\text{conv}}^m} \\ \vdots \\ \text{(II.13)} + (n_{L+1}l-1)d^m a_{\text{pad}}^m b_{\text{pad}}^m \mathbb{1}_{h^m h^m d^m a_{\text{conv}}^m b_{\text{conv}}^m} \end{bmatrix}. \tag{II.35}$$

The implementation is the same as that for evaluating (56), except that (II.32) involves $n_{L+1}l$ vectors rather than l.

II.4.3 Evaluation of $(\boldsymbol{v}^i)^T P_{\text{pool}}^{m,i}$ in Gradient Evaluations

Similar to (II.27), we calculate

$$\begin{bmatrix} (P_{\text{pool}}^{m,1})^T \boldsymbol{v}^1 \\ \vdots \\ (P_{\text{pool}}^{m,l})^T \boldsymbol{v}^l \end{bmatrix}^T \in R^{d^{m+1}a_{\text{conv}}^m b_{\text{conv}}^m l \times 1} \tag{II.36}$$

to have (82). In Section II.3 we have obtained the linear indices of (II.1) that correspond to the max values without considering the instance offset

$$(d^{m+1}a_{\text{conv}}^m b_{\text{conv}}^m)i.$$

By adding the instance offset, we have the correct mapping to the linear indices of (II.1) for the selected max values. In other words, similar to (II.29), we calculate

$$\begin{aligned}&\text{vec}\left(\text{(II.23)} + \mathbb{1}_{d^{m+1}a^{m+1}b^{m+1}} \times \left[0, d^{m+1}a^m_{\text{conv}}b^m_{\text{conv}}, \ldots, (l-1)d^{m+1}a^m_{\text{conv}}b^m_{\text{conv}}\right]\right)\\ =&\text{vec}\left(\text{(II.23)}\right) + \begin{bmatrix} 0\mathbb{1}_{d^{m+1}a^{m+1}b^{m+1}} \\ d^{m+1}a^m_{\text{conv}}b^m_{\text{conv}}\mathbb{1}_{d^{m+1}a^{m+1}b^{m+1}} \\ \vdots \\ (l-1)d^{m+1}a^m_{\text{conv}}b^m_{\text{conv}}\mathbb{1}_{d^{m+1}a^{m+1}b^{m+1}} \end{bmatrix}\end{aligned} \tag{II.37}$$

before applying `accumarray`. Note that in Listing VI we use matrix operations in line 14 to perform the summation in (II.37) and then produce the whole vector in line 31.

II.4.4 Evaluation of $\boldsymbol{v}^T P^{m,i}_{\text{pool}}$ in Jacobian Evaluations

We would like to have (65) by calculating

$$\begin{bmatrix} (P^{m,1}_{\text{pool}})^T\boldsymbol{v}^1_1 \\ \vdots \\ (P^{m,1}_{\text{pool}})^T\boldsymbol{v}^1_{n_{L+1}} \\ \vdots \\ (P^{m,l}_{\text{pool}})^T\boldsymbol{v}^l_{n_{L+1}} \end{bmatrix}^T \in R^{d^{m+1}a^m_{\text{conv}}b^m_{\text{conv}}n_{L+1}l\times 1}, \tag{II.38}$$

where

$$(\boldsymbol{v}^i_u)^T = \left(\frac{\partial \boldsymbol{z}^{L+1,i}}{\partial \text{vec}(Z^{m+1,i})^T} \odot \left(\mathbb{1}_{n_{L+1}}\text{vec}(I[Z^{m+1,i}])^T\right)\right)_{u,:} \in R^{1\times d^{m+1}a^{m+1}b^{m+1}},$$
$$i = 1, \ldots, l,\ u = 1, \ldots, n_{L+1},$$

and the subscript "$u,:$" indicates the uth row of the matrix. The calculation of (II.38) is the same as the calculation of (II.32). Thus, similar to (II.35), we need the following indices as input of `accumarray`:

$$\mathbb{1}_{n_{L+1}} \otimes \text{vec}(\text{(II.23)}) + \begin{bmatrix} \mathbf{0}_{d^{m+1}a^{m+1}b^{m+1}} \\ d^{m+1}a^m_{\text{conv}}b^m_{\text{conv}}\mathbb{1}_{d^{m+1}a^{m+1}b^{m+1}} \\ \vdots \\ (n_{L+1}-1)d^{m+1}a^m_{\text{conv}}b^m_{\text{conv}}\mathbb{1}_{d^{m+1}a^{m+1}b^{m+1}} \\ \vdots \\ (n_{L+1}l-1)d^{m+1}a^m_{\text{conv}}b^m_{\text{conv}}\mathbb{1}_{d^{m+1}a^{m+1}b^{m+1}} \end{bmatrix}.$$

II.5 Evaluation of Gauss-Newton Matrix-Vector Products

From (72), we conduct the Gauss-Newton matrix-vector products in two subroutines. The first subroutine is to evaluate (75). The second subroutine is to evaluate

$$\sum_{i=1}^{l}(J^{m,i})^T\boldsymbol{q}^i. \tag{II.39}$$

II.5.1 Details of Evaluating (75)

To take advantage of the fast computation in matrix form, we arrange (75) of all instances into

$$\begin{bmatrix} \sum_{m=1}^{L} J^{m,1}\boldsymbol{v}^m \\ \vdots \\ \sum_{m=1}^{L} J^{m,l}\boldsymbol{v}^m \end{bmatrix} \in R^{n_{L+1}l\times 1}. \tag{II.40}$$

From (74), for a particular m, we have

$$\begin{aligned} \begin{bmatrix} J^{m,1}\boldsymbol{v}^m \\ \vdots \\ J^{m,l}\boldsymbol{v}^m \end{bmatrix} &= \begin{bmatrix} \frac{\partial \boldsymbol{z}^{L+1,1}}{\partial \text{vec}(S^{m,1})^T}\text{vec}\left(\text{mat}(\boldsymbol{v}^m)\begin{bmatrix}\phi(\text{pad}(Z^{m,1}))\\ \mathbb{1}^T_{a^m_{\text{conv}}b^m_{\text{conv}}}\end{bmatrix}\right) \\ \vdots \\ \frac{\partial \boldsymbol{z}^{L+1,l}}{\partial \text{vec}(S^{m,l})^T}\text{vec}\left(\text{mat}(\boldsymbol{v}^m)\begin{bmatrix}\phi(\text{pad}(Z^{m,l}))\\ \mathbb{1}^T_{a^m_{\text{conv}}b^m_{\text{conv}}}\end{bmatrix}\right) \end{bmatrix} \\ &= \begin{bmatrix} \frac{\partial \boldsymbol{z}^{L+1,1}}{\partial \text{vec}(S^{m,1})^T}\boldsymbol{p}^{m,1} \\ \vdots \\ \frac{\partial \boldsymbol{z}^{L+1,l}}{\partial \text{vec}(S^{m,l})^T}\boldsymbol{p}^{m,l} \end{bmatrix}, \end{aligned} \tag{II.41}$$

where

$$\text{mat}(\boldsymbol{v}^m) \in R^{d^{m+1}\times(h^m h^m d^m+1)}$$

and

$$\boldsymbol{p}^{m,i} = \text{vec}\left(\text{mat}(\boldsymbol{v}^m)\begin{bmatrix}\phi(\text{pad}(Z^{m,i}))\\ \mathbb{1}^T_{a^m_{\text{conv}}b^m_{\text{conv}}}\end{bmatrix}\right). \tag{II.42}$$

We present our *MATLAB* implementation in Listing VII and explain the details here. Given $\boldsymbol{v}^m$, we calculate

$$\text{mat}(\boldsymbol{v}^m)\begin{bmatrix}\phi(\text{pad}(Z^{m,1})) & \cdots & \phi(\text{pad}(Z^{m,l}))\\ \mathbb{1}^T_{a^m_{\text{conv}}b^m_{\text{conv}}} & \cdots & \mathbb{1}^T_{a^m_{\text{conv}}b^m_{\text{conv}}}\end{bmatrix} \in R^{d^{m+1}\times a^m_{\text{conv}}b^m_{\text{conv}}l}; \tag{II.43}$$

see line 24. Next, we calculate

$$J^{m,i}\boldsymbol{v}^m = \frac{\partial \boldsymbol{z}^{L+1,i}}{\partial \text{vec}(S^{m,i})^T}\boldsymbol{p}^{m,i},\ i=1,\ldots,l. \tag{II.44}$$

Because (II.44) involves l independent matrix-vector products, we consider the following trick to avoid a `for` loop in a *MATLAB* script. We note that (II.44) can be calculated by summing up all rows of the following matrix

$$\begin{bmatrix}\frac{\partial z^{L+1,i}_1}{\partial \text{vec}(S^{m,i})}\cdots\frac{\partial z^{L+1,i}_{n_{L+1}}}{\partial \text{vec}(S^{m,i})}\end{bmatrix}_{d^{m+1}a^m_{\text{conv}}b^m_{\text{conv}}\times n_{L+1}} \odot \left[\boldsymbol{p}^{m,i}\cdots\boldsymbol{p}^{m,i}\right]_{d^{m+1}a^m_{\text{conv}}b^m_{\text{conv}}\times n_{L+1}}. \tag{II.45}$$

The result will be a row vector of $1\times n_{L+1}$, which is the transpose of $J^{m,i}\boldsymbol{v}^m$. To do the above operation on all instances together, we reformulate (II.3) and (II.43) respectively to the following three-dimensional matrices:

$$d^{m+1}a^m_{\text{conv}}b^m_{\text{conv}}\times n_{L+1}\times l \quad \text{and} \quad d^{m+1}a^m_{\text{conv}}b^m_{\text{conv}}\times 1\times l.$$

We then apply the `.*` operator in *MATLAB* and sum results along the first dimension; see line 25. The resulting matrix has the size

$$1 \times n_{L+1} \times l$$

and can be aggregated to the vector in (II.41); see line 26.

II.5.2 Details of Evaluating (II.39)

After deriving (II.40), from (76), we must calculate

$$\boldsymbol{q} = \begin{bmatrix} \boldsymbol{q}^1 \\ \vdots \\ \boldsymbol{q}^l \end{bmatrix} = \begin{bmatrix} B^1 \sum_{m=1}^{L} J^{m,1} \boldsymbol{v}^m \\ \vdots \\ B^l \sum_{m=1}^{L} J^{m,l} \boldsymbol{v}^m \end{bmatrix}. \tag{II.46}$$

From (77), (II.46) can be derived by multiplying every element of (II.40) by two.

Next, for each layer m, from (72) and (78) we calculate (II.39) by

$$\begin{aligned} &\sum_{i=1}^{l} \text{vec} \left(\text{mat} \left(\left(\frac{\partial \boldsymbol{z}^{L+1,i}}{\partial \text{vec}(S^{m,i})^T} \right)^T \boldsymbol{q}^i \right)_{d^{m+1} \times a^m_{\text{conv}} b^m_{\text{conv}}} \left[\phi(\text{pad}(Z^{m,i}))^T \; \mathbb{1}_{a^m_{\text{conv}} b^m_{\text{conv}}} \right] \right) \\ &= \text{vec} \left(\begin{bmatrix} \text{mat}(\boldsymbol{u}^{m,1})_{d^{m+1} \times a^m_{\text{conv}} b^m_{\text{conv}}} & \dots & \text{mat}(\boldsymbol{u}^{m,l})_{d^{m+1} \times a^m_{\text{conv}} b^m_{\text{conv}}} \end{bmatrix} \begin{bmatrix} \phi(\text{pad}(Z^{m,1}))^T & \mathbb{1}_{a^m_{\text{conv}} b^m_{\text{conv}}} \\ \vdots & \vdots \\ \phi(\text{pad}(Z^{m,l}))^T & \mathbb{1}_{a^m_{\text{conv}} b^m_{\text{conv}}} \end{bmatrix} \right), \end{aligned} \tag{II.47}$$

where

$$\boldsymbol{u}^{m,i} = \left(\frac{\partial \boldsymbol{z}^{L+1,i}}{\partial \text{vec}(S^{m,i})^T} \right)^T \boldsymbol{q}^i.$$

A *MATLAB* implementation is shown in Listing VIII. To begin, we have the matrix (II.3) and the vector $\boldsymbol{q}$ in (II.46). We reshape (II.3) to

$$\begin{bmatrix} \frac{\partial z_1^{L+1,1}}{\partial \text{vec}(S^{m,1})} & \dots & \frac{\partial z_{n_{L+1}}^{L+1,1}}{\partial \text{vec}(S^{m,1})} & \dots & \frac{\partial z_{n_{L+1}}^{L+1,l}}{\partial \text{vec}(S^{m,l})} \end{bmatrix} \in R^{d^{m+1} a^m_{\text{conv}} b^m_{\text{conv}} \times n_{L+1} l}. \tag{II.48}$$

Then we calculate

$$\begin{bmatrix} \boldsymbol{u}^{m,1} & \dots & \boldsymbol{u}^{m,l} \end{bmatrix} \tag{II.49}$$

together by reshaping

$$\big[\text{(II.48)}\big] \odot \left(\mathbb{1}_{d^{m+1} a^m_{\text{conv}} b^m_{\text{conv}}} \boldsymbol{q}^T \right) \in R^{d^{m+1} a^m_{\text{conv}} b^m_{\text{conv}} \times n_{L+1} l} \tag{II.50}$$

to

$$R^{d^{m+1} a^m_{\text{conv}} b^m_{\text{conv}} \times n_{L+1} \times l}$$

and summing along the second dimension; see line 26-27. After having (II.49), we reshape it to

$$R^{d^{m+1} \times a^m_{\text{conv}} b^m_{\text{conv}} l}$$

and calculate (II.47) by a matrix multiplication in line 28.

Listing VI: *MATLAB* implementation to evaluate $(\boldsymbol{v}^i)^T P_\phi^m$ and $(\boldsymbol{v}^i)^T P_{\text{pool}}^{m,i}$

```
function vTP = vTP(param, model, net, m, V, op)
% output vTP: a row vector, where mat(vTP) is with dimension $
   d_prev a_prev b_prev \times num_v$.

nL = param.nL;
num_data = net.num_sampled_data;

switch op
case {'pool_gradient', 'pool_Jacobian'}
        a_prev = model.ht_conv(m);
        b_prev = model.wd_conv(m);
        d_prev = model.ch_input(m+1);
        if strcmp(op, 'pool_gradient')
                num_v = num_data;
                idx = net.idx_pool{m} + [0:num_data-1]*d_prev*
                   a_prev*b_prev;
        else
                num_v = nL*num_data;
                idx = reshape(net.idx_pool{m}, [], 1, num_data)
                    + reshape([0:nL*num_data-1]*d_prev*a_prev*
                   b_prev, 1, nL, num_data);
        end
case {'phi_gradient', 'phi_Jacobian'}
        a_prev = model.ht_pad(m);
        b_prev = model.wd_pad(m);
        d_prev = model.ch_input(m);

        if strcmp(op, 'phi_gradient'); num_v = num_data; else;
           num_v = nL*num_data; end

        idx = net.idx_phiZ{m}(:) + [0:num_v-1]*d_prev*a_prev*
           b_prev;
otherwise
        error('Unknown operation in function vTP.');
end

vTP = accumarray(idx(:), V(:), [d_prev*a_prev*b_prev*num_v 1])
   ';
```

Listing VII: *MATLAB* implementation for Jv

```
function Jv = Jv(param, model, net, v)

nL = param.nL;
L = param.L;
LC = param.LC;
num_data = net.num_sampled_data;
var_ptr = model.var_ptr;
Jv = zeros(nL*num_data, 1);

for m = L : -1 : LC+1
        var_range = var_ptr(m) : var_ptr(m+1) - 1;
        n_m = model.full_neurons(m-LC);

        p = reshape(v(var_range), n_m, []) * [net.Z{m}; ones(1,
            num_data)];
        p = sum(reshape(net.dzdS{m}, n_m, nL, []) .* reshape(p,
            n_m, 1, []),1);
        Jv = Jv + p(:);
end

for m = LC : -1 : 1
        var_range = var_ptr(m) : var_ptr(m+1) - 1;
        ab = model.ht_conv(m)*model.wd_conv(m);
        d = model.ch_input(m+1);

        p = reshape(v(var_range), d, []) * [net.phiZ{m}; ones
            (1, ab*num_data)];
        p = sum(reshape(net.dzdS{m}, d*ab, nL, []) .* reshape(p
            , d*ab, 1, []),1);
        Jv = Jv + p(:);
end
```

Listing VIII: *MATLAB* implementation for $J^T \boldsymbol{q}$

```
function u = JTq(param, model, net, q)

nL = param.nL;
L = param.L;
LC = param.LC;
num_data = net.num_sampled_data;
var_ptr = model.var_ptr;
n = var_ptr(end) - 1;
u = zeros(n, 1);

for m = L : -1 : LC+1
        var_range = var_ptr(m) : var_ptr(m+1) - 1;

        u_m = net.dzdS{m} .* q';
        u_m = sum(reshape(u_m, [], nL, num_data), 2);
        u_m = reshape(u_m, [], num_data) * [net.Z{m}' ones(
           num_data, 1)];
        u(var_range) = u_m(:);
end

for m = LC : -1 : 1
        a = model.ht_conv(m);
        b = model.wd_conv(m);
        d = model.ch_input(m+1);
        var_range = var_ptr(m) : var_ptr(m+1) - 1;

        u_m = reshape(net.dzdS{m}, [], nL*num_data) .* q';
        u_m = sum(reshape(u_m, [], nL, num_data), 2);
        u_m = reshape(u_m, d, []) * [net.phiZ{m}' ones(a*b*
           num_data, 1)];
        u(var_range) = u_m(:);
end
```