



CENG 4480

Embedded System Development & Applications

Lec 05: Sparse Conv

Bei Yu

CSE Department, CUHK

byu@cse.cuhk.edu.hk

(Latest update: October 6, 2025)

2025 Fall



- ① Kernel Sparse Convolution
- ② Submanifold Sparse Convolution

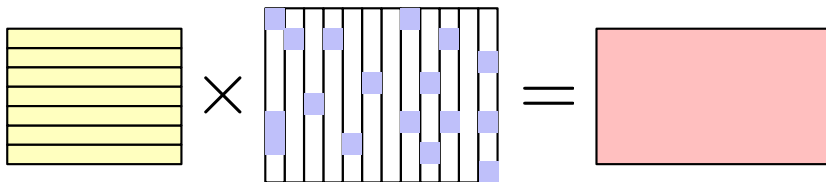


- 1 Kernel Sparse Convolution
- 2 Submanifold Sparse Convolution



Kernel Sparse Convolution

- Our DNN may be **redundant**, and sometimes the filters may be **sparse**
- Sparsity can be helpful to **overcome over-fitting**





X			
0	0	3	0
7	0	0	0
0	0	4	8
6	5	3	0
2	0	0	1
0	0	0	8

*

W
0
0
4
8

Algorithm Sparse Convolution Naive 1

```
1: for all  $w[i]$  do  
2:   if  $w[i] = 0$  then  
3:     Continue;  
4:   end if  
5:   output feature map  $Y \leftarrow X \times w[i]$ ;  
6: end for
```



$$\begin{array}{c}
 X \\
 \begin{array}{|c|c|c|c|}
 \hline
 0 & 0 & 3 & 0 \\
 \hline
 7 & 0 & 0 & 0 \\
 \hline
 0 & 0 & 4 & 8 \\
 \hline
 6 & 5 & 3 & 0 \\
 \hline
 2 & 0 & 0 & 1 \\
 \hline
 0 & 0 & 0 & 8 \\
 \hline
 \end{array}
 \end{array}
 *
 \begin{array}{c}
 W \\
 \begin{array}{|c|}
 \hline
 0 \\
 \hline
 0 \\
 \hline
 4 \\
 \hline
 8 \\
 \hline
 \end{array}
 \end{array}$$

Algorithm Sparse Convolution Naive 1

```

1: for all  $w[i]$  do
2:   if  $w[i] = 0$  then
3:     Continue;
4:   end if
5:   output feature map  $Y \leftarrow X \times w[i]$ ;
6: end for
    
```

BAD implementation for Pipeline!

Instr. No.	Pipeline Stage						
1	IF	ID	EX	MEM	WB		
2		IF	ID	EX	MEM	WB	
3			IF	ID	EX	MEM	WB
4				IF	ID	EX	MEM
5					IF	ID	EX
Clock Cycle	1	2	3	4	5	6	7

A

0	0	3	0
7	0	0	0
0	0	4	8
6	5	3	0
2	0	0	1
0	0	0	8

**A matrix
example**

rowptr

•	→ row0 (3,2)
•	→ row1 (7,0)
•	→ row2 (4,2), (8,3)
•	→ row3 (6,0), (5,1), (3,2)
•	→ row4 (2,0), (1,3)
•	→ row5 (8,3)

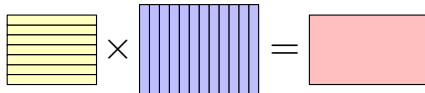
**Compressed
Sparse Row
(CSR)**

colptr

•	→ col0 (7,1), (6,3), (2,4)
•	→ col1 (5,3)
•	→ col2 (3,0), (4,2), (3,3)
•	→ col3 (8,2), (1,4), (8,5)

**Compressed
Sparse Column
(CSC)**

- **CSR**: Good for operation on **feature maps**
- **CSC**: Good for operation on **filters**
- We have **better control on filters**, thus usually CSC.





matrix * sparse vector

$$\begin{array}{|c|c|c|c|} \hline 0 & 0 & \boxed{3} & 0 \\ \hline 7 & 0 & 0 & 0 \\ \hline 0 & 0 & 4 & 8 \\ \hline 6 & 5 & 3 & 0 \\ \hline 2 & 0 & 0 & 1 \\ \hline 0 & 0 & 0 & 8 \\ \hline \end{array} \begin{array}{c} \text{X} \\ \text{w} \\ \text{Y} \end{array} \begin{array}{c} 0 \\ 0 \\ \boxed{4} \\ 8 \end{array} = \begin{array}{|c|} \hline 12 \\ \hline 0 \\ \hline 16 \\ \hline 12 \\ \hline 0 \\ \hline 0 \\ \hline \end{array}$$

$$\begin{array}{|c|c|c|c|} \hline 0 & 0 & 3 & \boxed{0} \\ \hline 7 & 0 & 0 & 0 \\ \hline 0 & 0 & 4 & 8 \\ \hline 6 & 5 & 3 & 0 \\ \hline 2 & 0 & 0 & 1 \\ \hline 0 & 0 & 0 & \boxed{8} \\ \hline \end{array} \begin{array}{c} 0 \\ 0 \\ \boxed{4} \\ 8 \end{array} = \begin{array}{|c|} \hline 12 \\ \hline 0 \\ \hline 80 \\ \hline 12 \\ \hline 8 \\ \hline 64 \\ \hline \end{array}$$

- **BAD** implementation for Spatial Locality!
- **Poor** memory access patterns

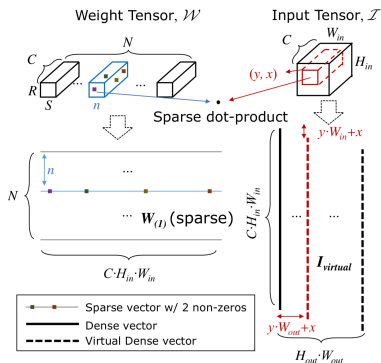


Figure 1: Conceptual view of the direct sparse convolution algorithm. Computation of output value at (y,x) th position of n th output channel is highlighted.

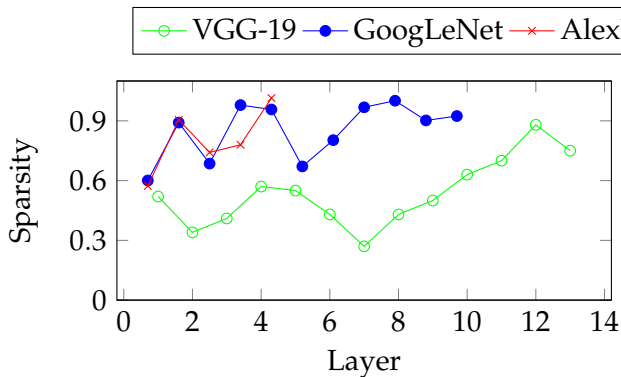
```

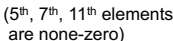
for each output channel n {
  for j in [W.rowptr[n], W.rowptr[n+1]] {
    off = W.colidx[j]; coeff = W.value[j]
    for (int y = 0; y < H_OUT; ++y) {
      for (int x = 0; x < W_OUT; ++x) {
        out[n][y][x] += coeff*in[off+f(0,y,x)]
      }
    }
  }
}
    
```

Figure 2: Sparse convolution pseudo code. Matrix $\mathbf{W}$ has *compressed sparse row* (CSR) format, where $\text{rowptr}[n]$ points to the first non-zero weight of n th output channel. For the j th non-zero weight at (n,c,r,s) , $\mathbf{W}.\text{colidx}[j]$ contains the offset to (c,r,s) th element of tensor in , which is pre-computed by layout function as $f(c,r,s)$. If in has CHW format, $f(c,r,s) = (cH_{in} + r)W_{in} + s$. The “virtual” dense matrix is formed on-the-fly by shifting in by $(0,y,x)$.

1

- Sparsity is a desired property for computation acceleration. (cuSPARSE library, direct sparse convolution, etc.)
- Sometimes not only the **filters** but also the **input feature maps** are sparse.





- 10/25

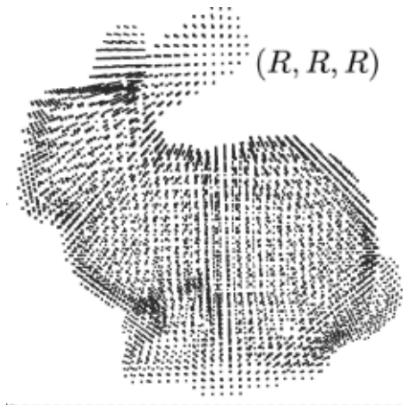


- 1 Kernel Sparse Convolution
- 2 Submanifold Sparse Convolution

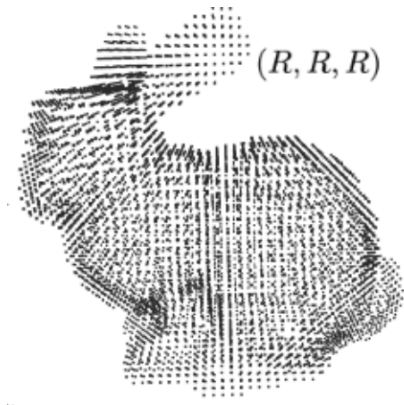


Submanifold Sparse Convolution

In real world, we have to handle voxel data sometimes. For example, in point cloud analysis, 3D voxel data is widely used. A simple example is shown here and it can be viewed as $V \in (1, R, R, R)$.



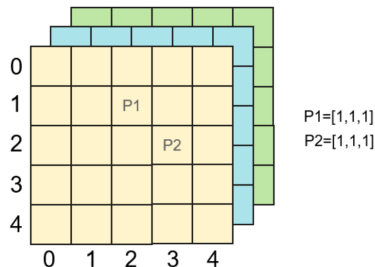
Here is a rabbit with shape $V \in (1, 64, 64, 64)$. If using traditional convolution to extract its feature, the GPU will out of memory very soon because the input $V \in (1, 64, 64, 64)$ can be viewed as an image $I \in (1, 4096, 64)$.



To overcome this issue, we use 3D sparse convolution for voxel data analysis. Sparse convolution only calculate the data points where voxel data exists.



In this Lab, we are going to build a sparse convolution from scratch. Here we use the example input:



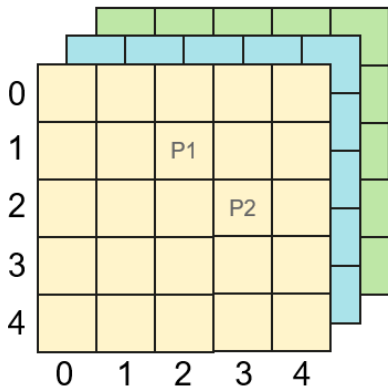
where P1 and P2 has pixel value of 1 in 3 channels.



Firstly, we build a hash table to store the input data. Considering the following case:

```
conv2D(kernel_size=3, out_channels=2, stride=1, padding=0)
```

We can build an input table H_{in} like this:



$P1=[1,1,1]$

$P2=[1,1,1]$

H_{in}

0	(2,1)
1	(3,2)

Then we build an output hash table. Firstly, we generate a P_{out} table as follow:

		P1		

P1		

(0,0)

Then we build an output hash table. Firstly, we generate a P_{out} table as follow:

		P1		

P1	P1	

(0,0)
(1,0)

Then we build an output hash table. Firstly, we generate a P_{out} table as follow:

		P1		

P1	P1	P1

(0,0)
(1,0)
(2,0)



Then we build an output hash table. Firstly, we generate a P_{out} table as follow:

		P1		

P1	P1	P1
P1		

(0,0)
(1,0)
(2,0)
(0,1)

Then we build an output hash table. Firstly, we generate a P_{out} table as follow:

		P1		

P1	P1	P1
P1	P1	

(0,0)
(1,0)
(2,0)
(0,1)
(1,1)

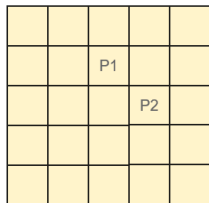
Then we build an output hash table. Firstly, we generate a P_{out} table as follow:

		P1		

P1	P1	P1
P1	P1	P1

(0,0)
(1,0)
(2,0)
(0,1)
(1,1)
(2,1)

After applying the same process to P_0 , we get an output hash table H_{out} via P_{out} merge



P1	P1	P1
P1	P1	P1

	P2	P2
	P2	P2
	P2	P2

P_{out}

(0,0)
(1,0)
(2,0)
(0,1)
(1,1)
(2,1)

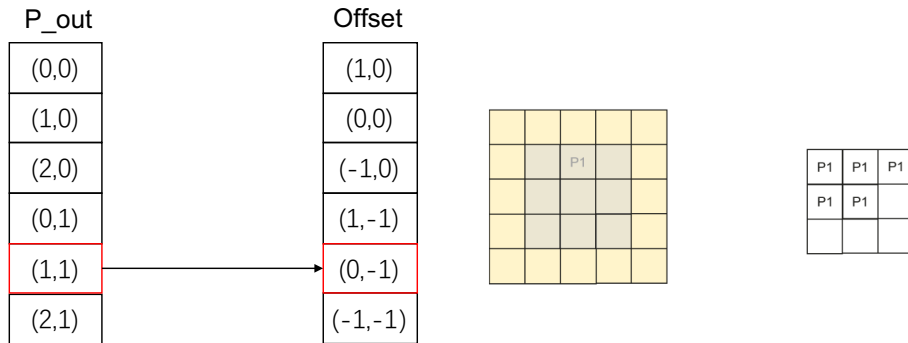
(1,0)
(2,0)
(1,1)
(2,1)
(1,2)
(2,2)

Merge P_{out}

H_{out}

0	(0,0)
1	(1,0)
2	(2,0)
3	(0,1)
4	(1,1)
5	(2,1)
6	(1,2)
7	(2,2)

- Next we build up a Rulebook to realize H_{in} to H_{out} .
- To build the rule book, we have to build an offset map like this:

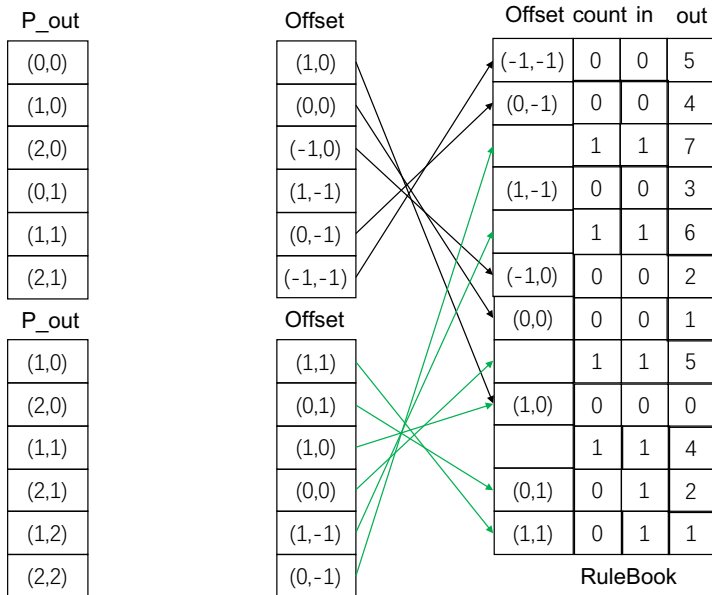




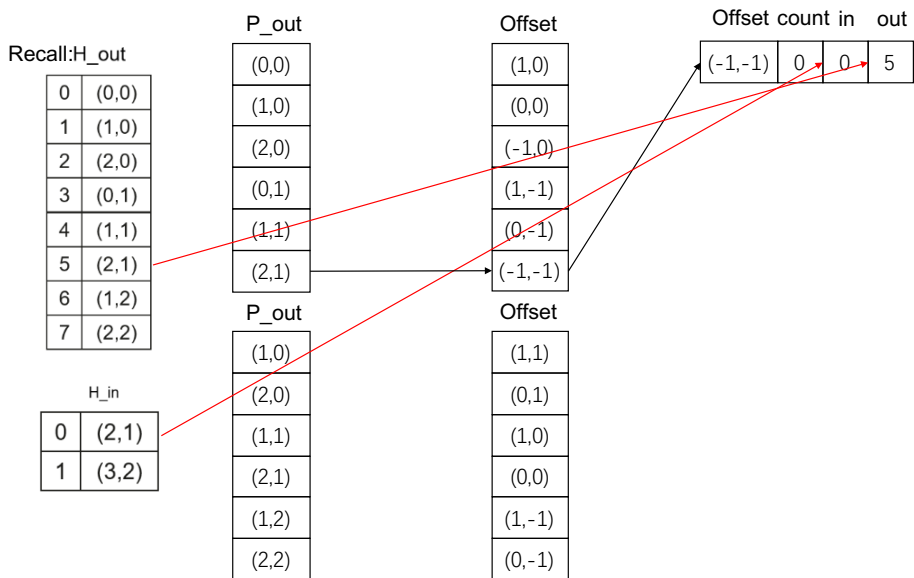
Quick Question:

Please write the offset map of P_2 by yourself.

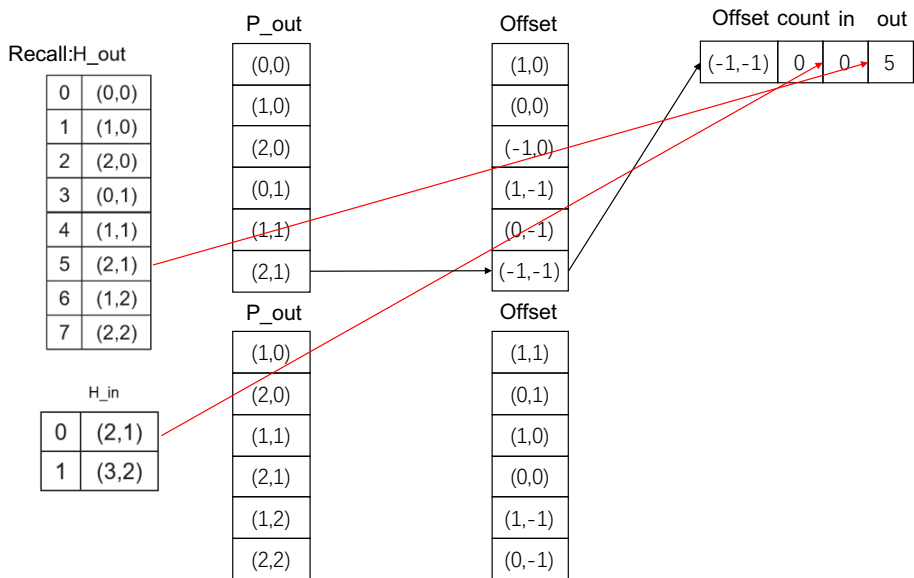
After obtaining the offset map, we can finally build up the rule book as follow:



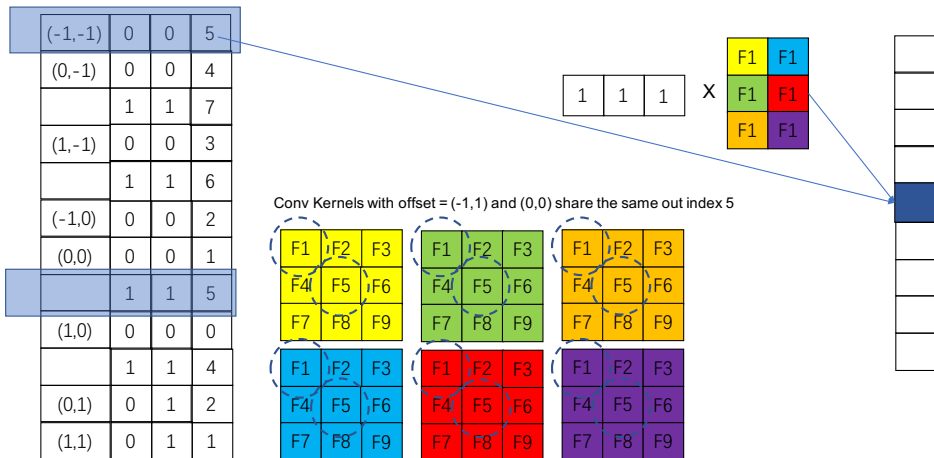
Recalling the H_{in} and H_{out} , the rulebook is generated as follow:



If the offset already exists, we simply add 1 in count:



After getting rulebook, we can apply sparse convolution:



For P_1 , the results is shown above, which is the blue points in 5-th row. Please practice P_2 by yourself

Temporary page!

$\text{\LaTeX}$ was unable to guess the total number of pages correctly. As there was some unprocessed data that should have been added to the final page this extra page has been added to receive it.

If you rerun the document (without altering it) this surplus page will go away, because $\text{\LaTeX}$ now knows how many pages to expect for this document.