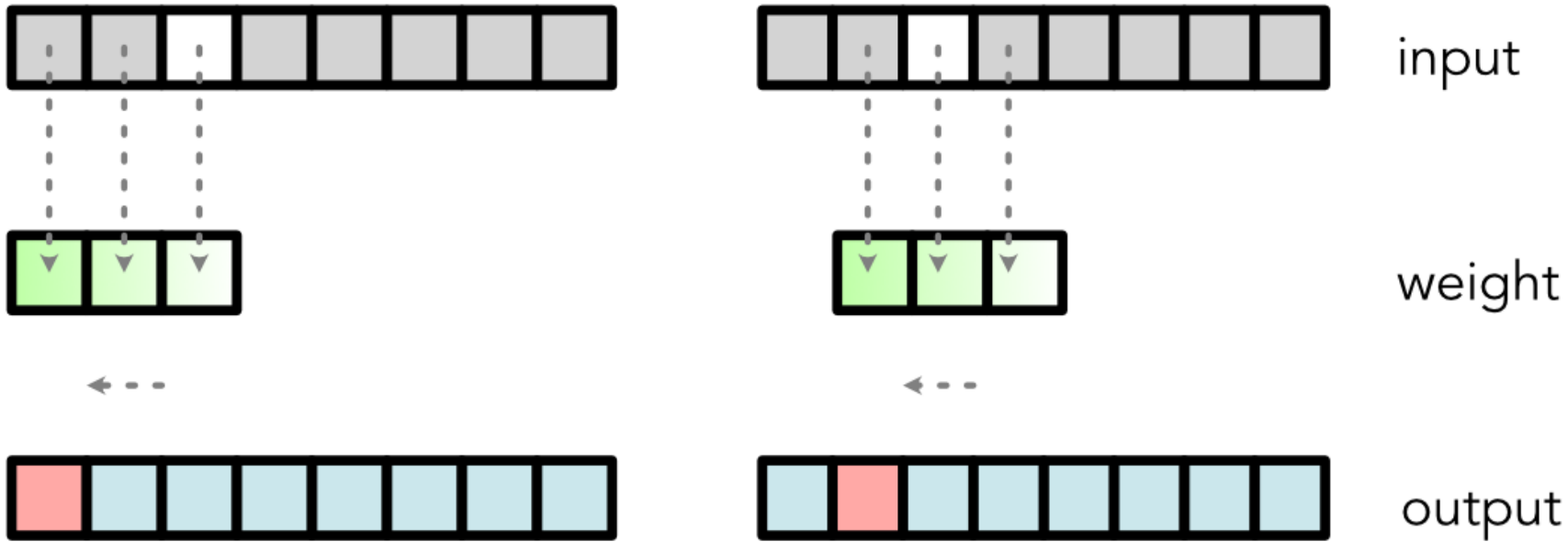
Module 4.2 - Shapes


input
weight
output

Computation

Output Values

```
output[0] = weight[0] * input[0] + weight[1] * input[1]  + weight[2] * input[2]
output[1] = weight[0] * input[1] + weight[1] * input[2]  + weight[2] * input[3]
output[2] = weight[0] * input[2] + weight[1] * input[3]  + weight[2] * input[4]
```

Alternative View

Unroll

```
def unroll(input, T, K):
    out = [[input[i + k] if i + k < T else 0
            for k in range(K)]
           for i in range(T)]
    return tensor(out)
```

Alternative View

Unroll

```
input = tensor([1, 2, 3, 4, 5, 6])
K = 3
T = input.shape[0]
unrolled_input = unroll(input, T, K)
print(unrolled_input)
```

```
[
        [1.00 2.00 3.00]
        [2.00 3.00 4.00]
        [3.00 4.00 5.00]
        [4.00 5.00 6.00]
        [5.00 6.00 0.00]
        [6.00 0.00 0.00]]
```

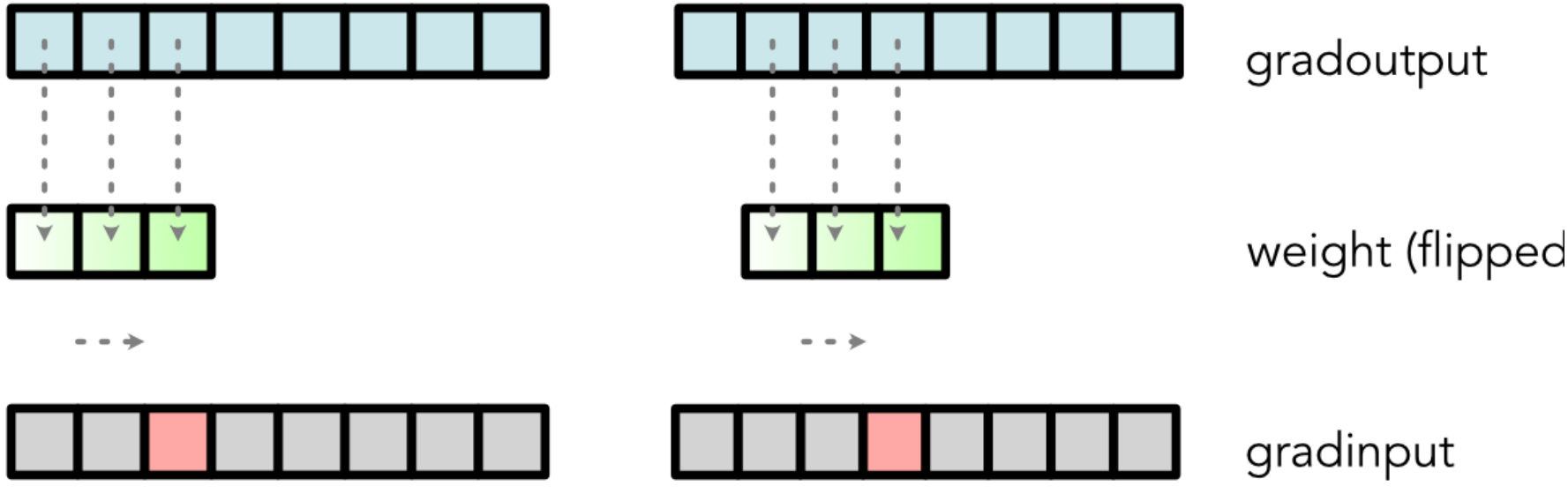
Gradient

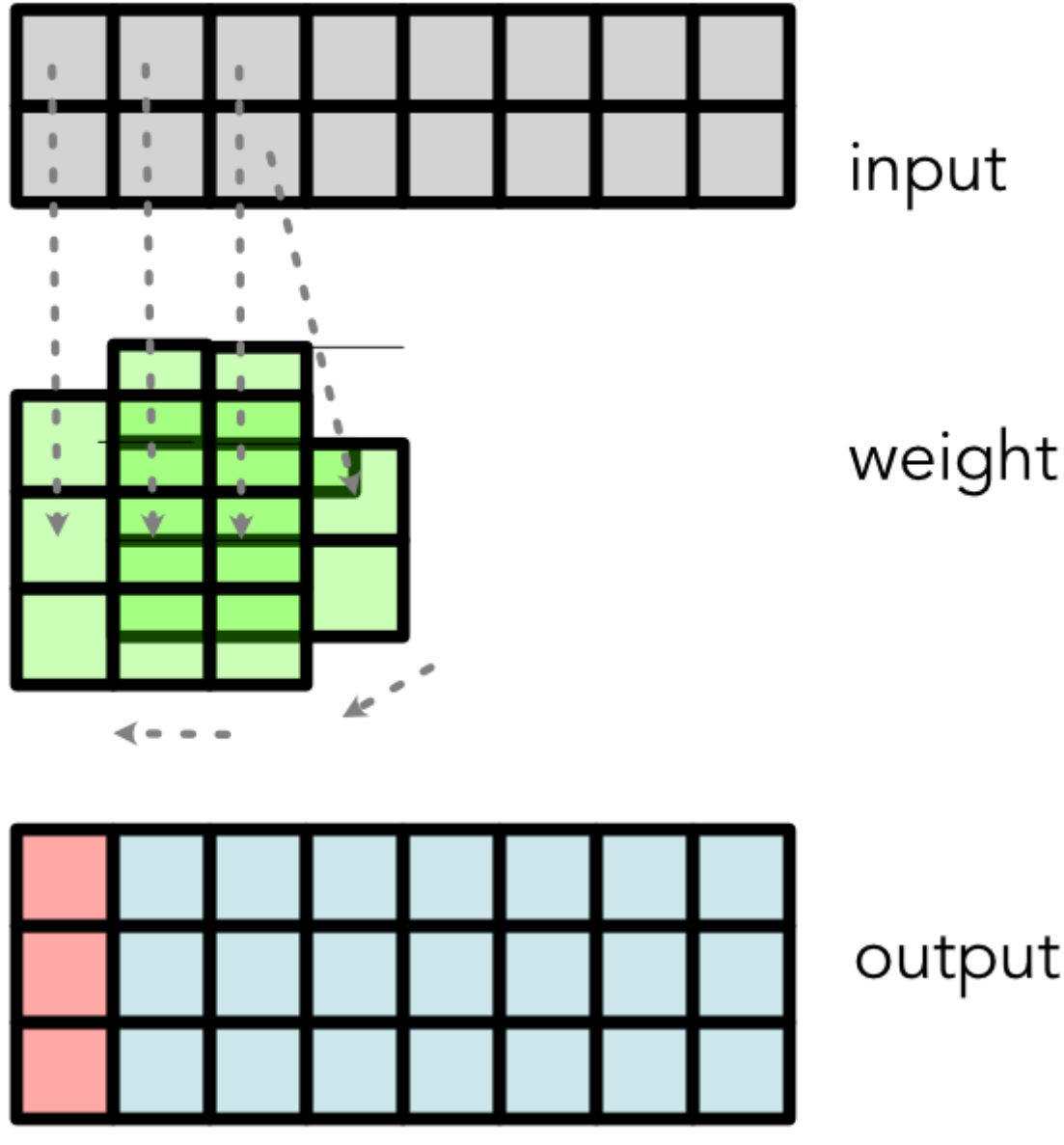
```
class Conv:

    @staticmethod
    def backward(ctx, d):
        ...
        grad_input[2] = weight[0] * d[2] + weight[1] * d[1]  + weight[2] * d[0]
        ...
```

Conv Back - Input

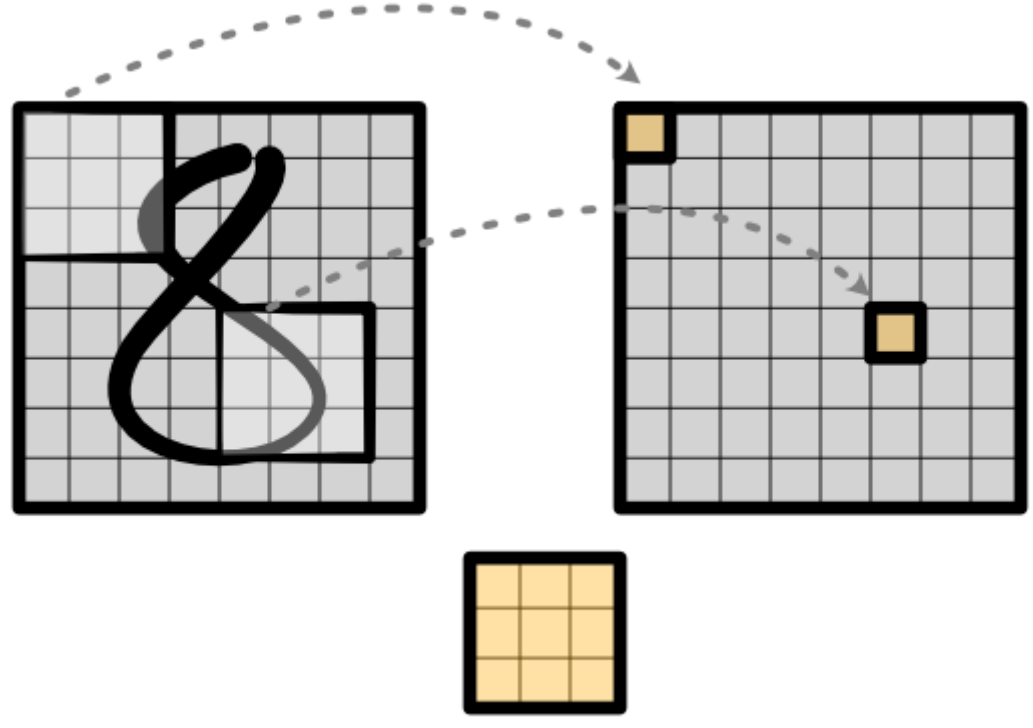
Reverse the convolutional anchor




input
weight
output

Two Dimensional Convolution

- Instead of line, now use box
- Box is anchored at the top-left
- Zip-reduce is over full box!



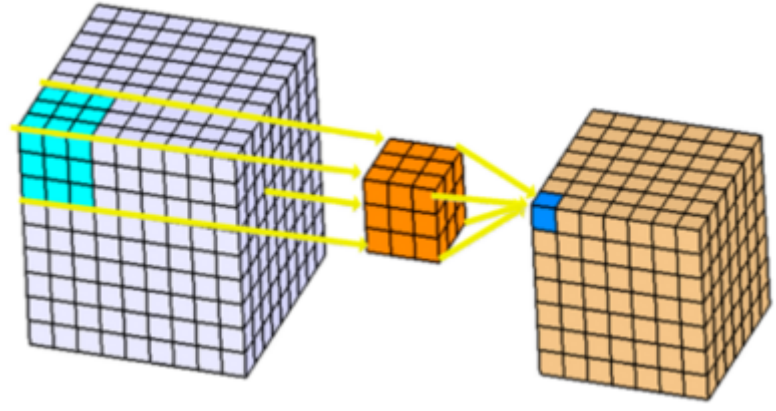
Quiz

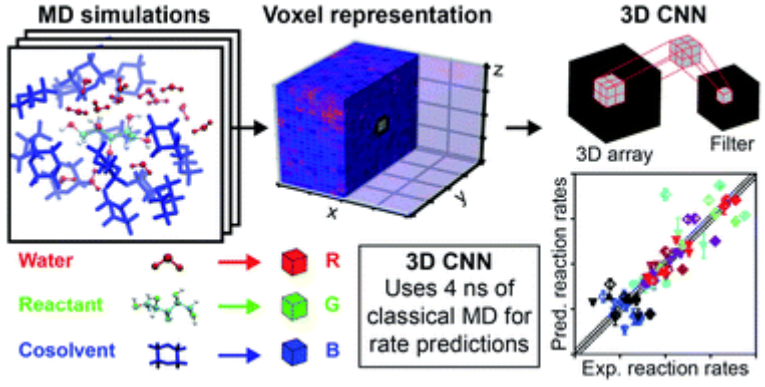
Quiz

3D Convolution

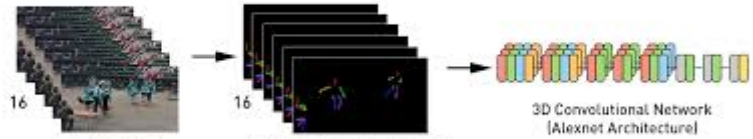
3D Convolution?

- Yeah!
- Several neat versions




MD simulations
Voxel representation
3D CNN
z
x
y
3D array
Filter
Water
Reactant
Cosolvent
R
G
B
3D CNN
Uses 4 ns of classical MD for rate predictions
Pred. reaction rates
Exp. reaction rates

Temporal CNN (Skeletal)



Input Frames

Multi Person Pose Estimation

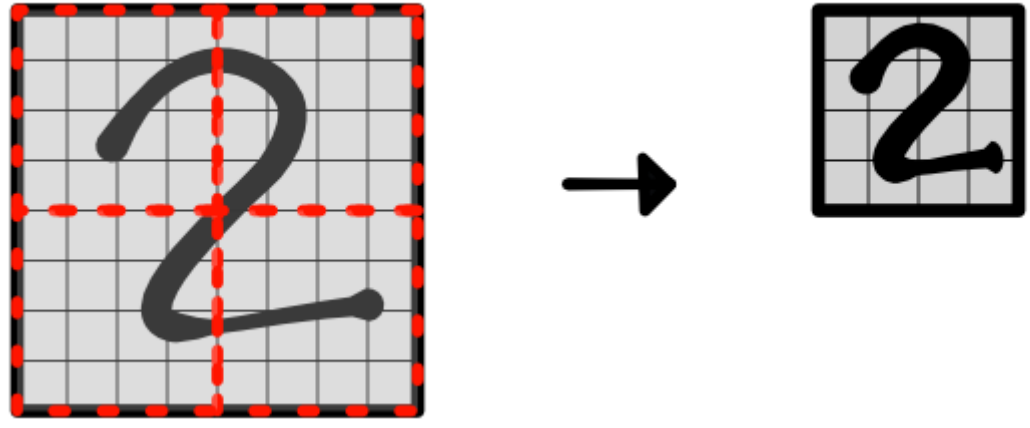
Pooling

Challenge

- How do we look at bigger areas with convolutions?

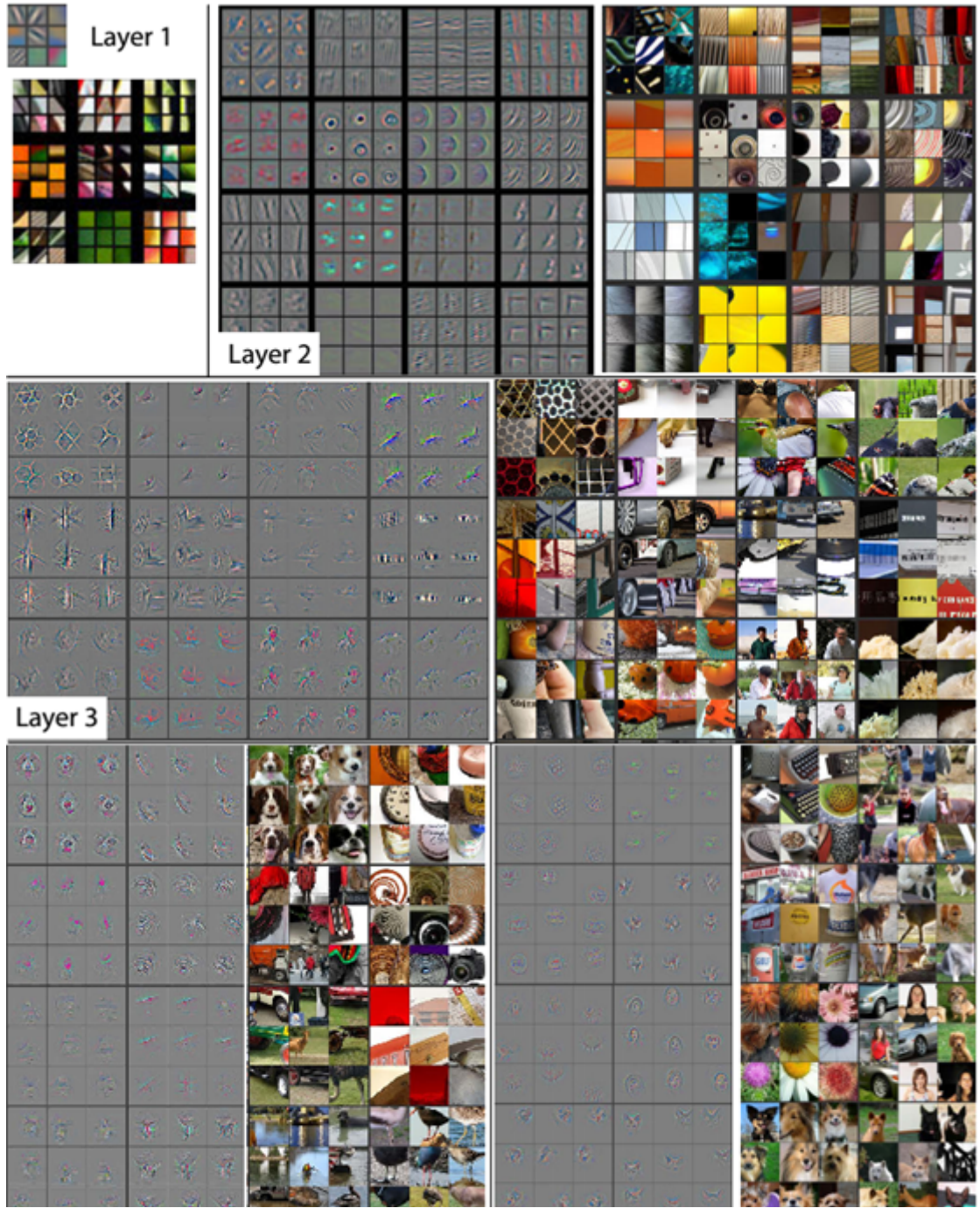
Pooling

- Adjusts the scale at each layer
- Conv stays the same size, image "zooms" out



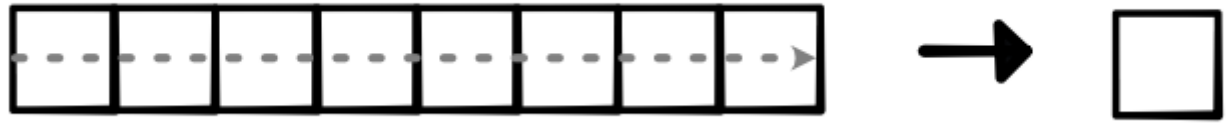
Goal

- Early layers: Capture basic shapes
- Middle layers: How these connect
- Later layers: Full objects


Layer 1
Layer 2
Layer 3

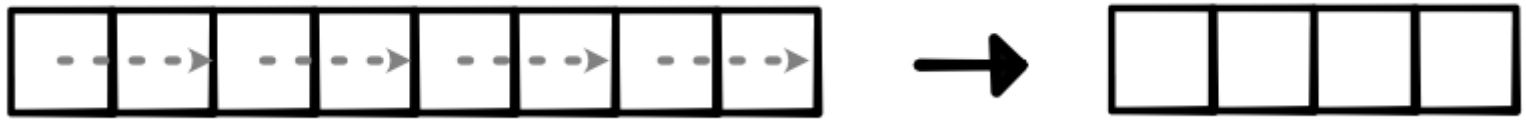
Issues

- Number of parameters scale with weight size
- "Bigger" patterns require more ways to split data.



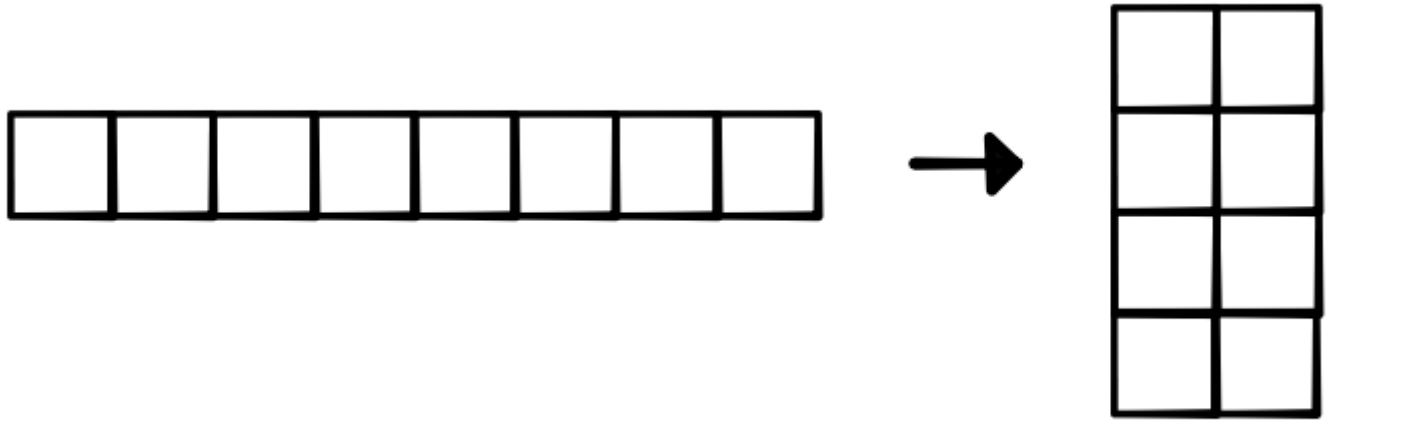
”Pooling”

Reduction applied to each region:



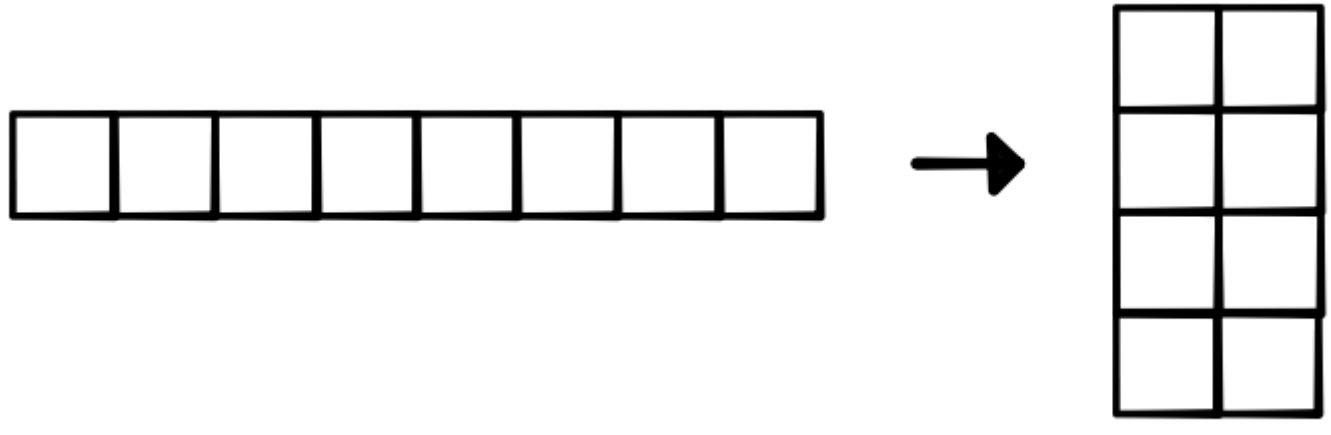
Simple Implementation

- Ensure that it is contiguous
- Use View to "fold" the tensor



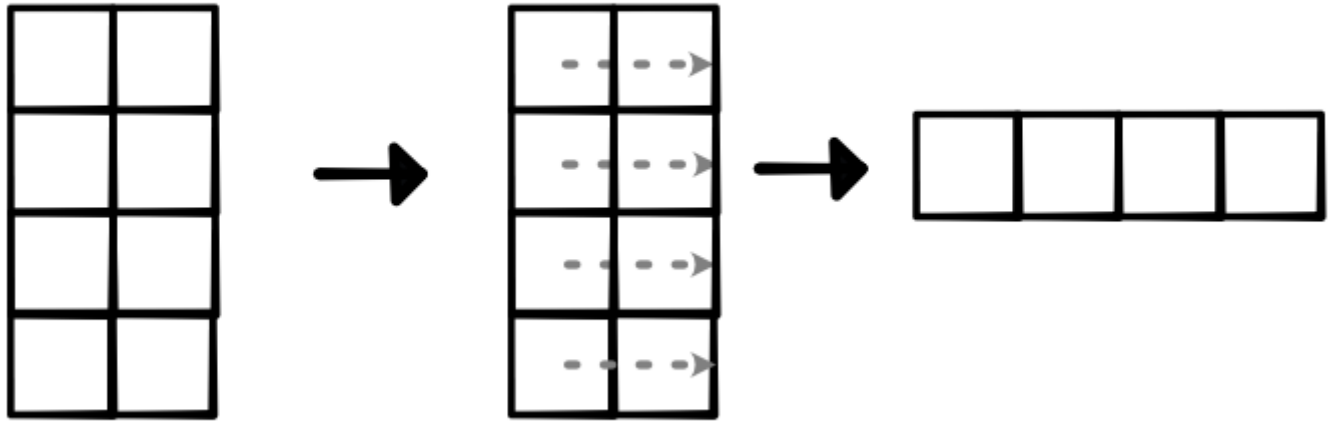
Why does folding work?

- View requires "contiguous" tensor
- View(4, 2) makes strides (2, 1)



Simple Implementation

- Reduce along created fold



2D Pooling

- Need to isolate squares into a single dimension.
- Tensor origami :)

Exercise

- If I have a (10, 10) cube. How do I sum up neighboring rows?
- Goal (5, 10) cube.

Fast Implementations?

- If your reduce is on CUDA, can exploit small groups
- I.e. Prefix sum for each group on one block.

Gradient Flow

- Layers that are used get more updates
- Gradient signals which aspect was important
- Can have extra layers