

Convolutional Neural Networks

End-to-end learning

Self-Driving Cars

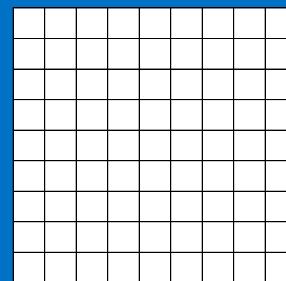
Madhur Behl
Principles of Modeling for Cyber-Physical Systems-Fall

Slides credits: Brandon Rohrer, Fei-Fei Li, Justin Johnson & Serena Yeung

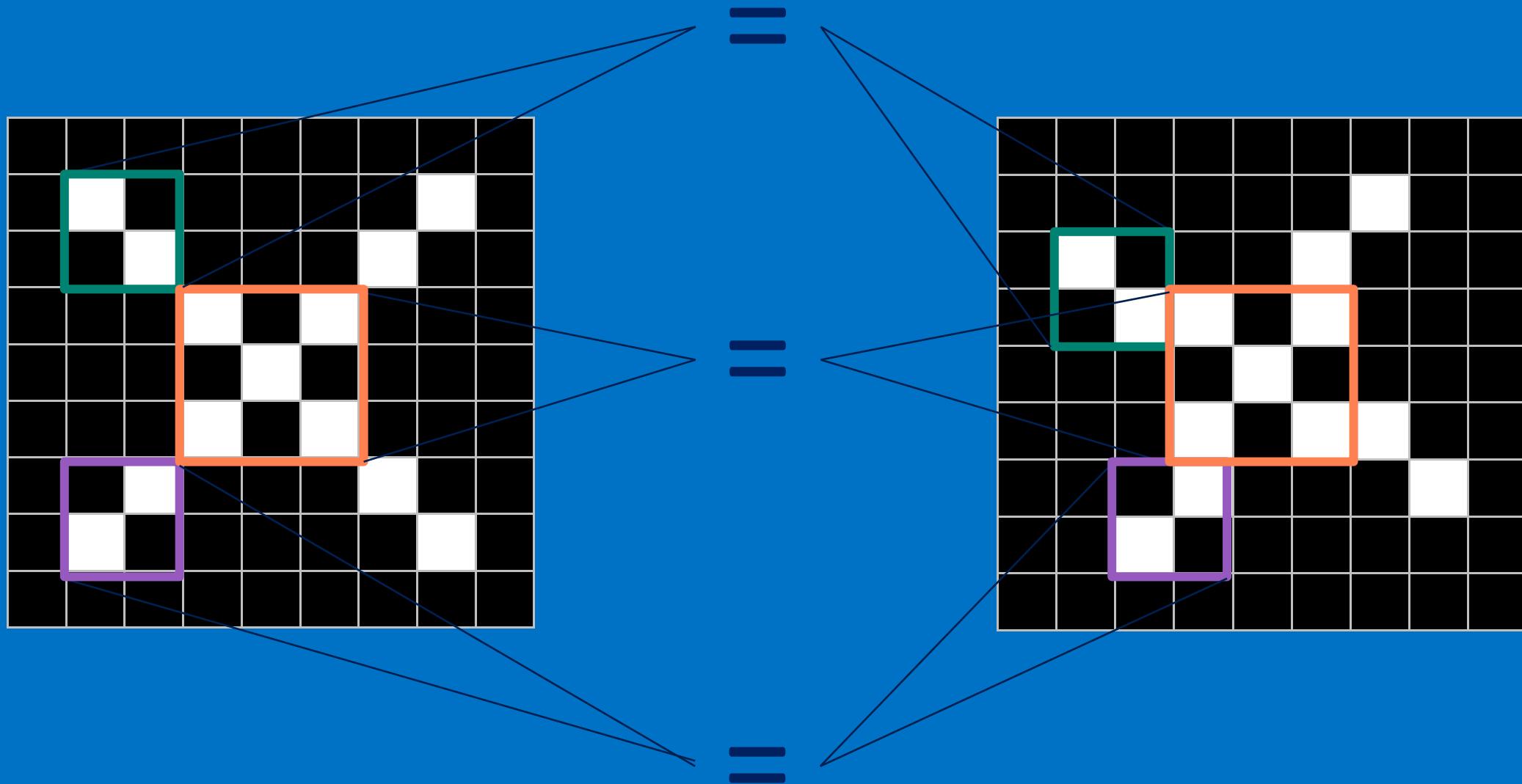
A toy ConvNet: X's and O's

Says whether a picture is of an X or an O

A two-dimensional
array of pixels



ConvNets match pieces of the image



Features match pieces of the image

| | | |
|----|----|----|
| 1 | -1 | -1 |
| -1 | 1 | -1 |
| -1 | -1 | 1 |

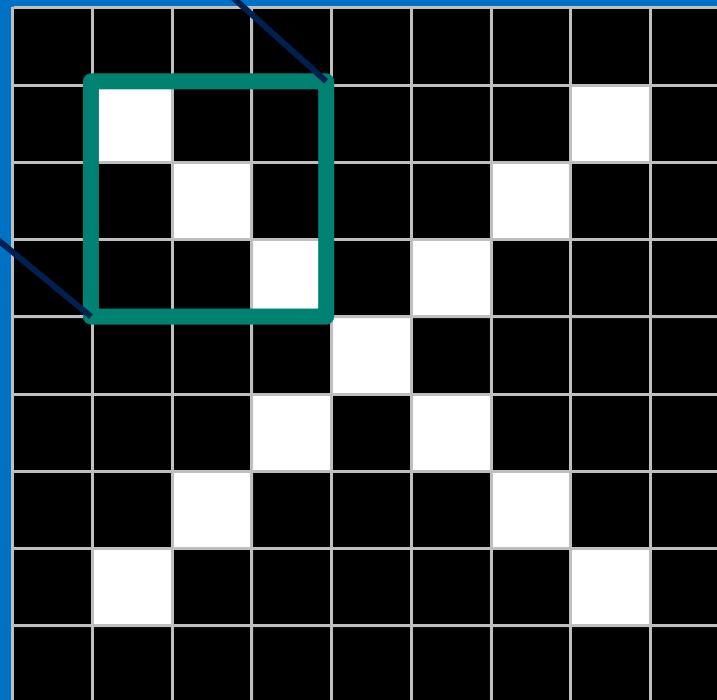
| | | |
|----|----|----|
| 1 | -1 | 1 |
| -1 | 1 | -1 |
| 1 | -1 | 1 |

| | | |
|----|----|----|
| -1 | -1 | 1 |
| -1 | 1 | -1 |
| 1 | -1 | -1 |

| | | |
|----|----|----|
| 1 | -1 | -1 |
| -1 | 1 | -1 |
| -1 | -1 | 1 |

| | | |
|----|----|----|
| 1 | -1 | 1 |
| -1 | 1 | -1 |
| 1 | -1 | 1 |

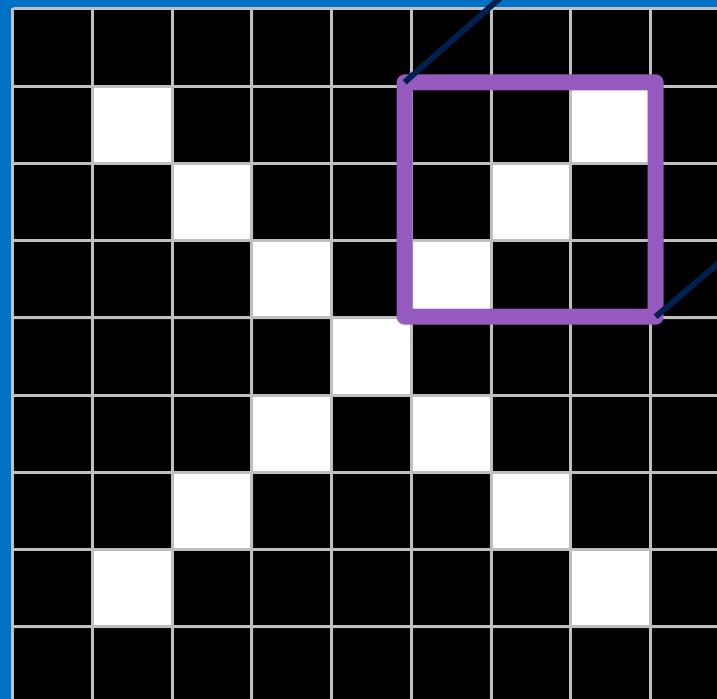
| | | |
|----|----|----|
| -1 | -1 | 1 |
| -1 | 1 | -1 |
| 1 | -1 | -1 |



| | | |
|----|----|----|
| 1 | -1 | -1 |
| -1 | 1 | -1 |
| -1 | -1 | 1 |

| | | |
|----|----|----|
| 1 | -1 | 1 |
| -1 | 1 | -1 |
| 1 | -1 | 1 |

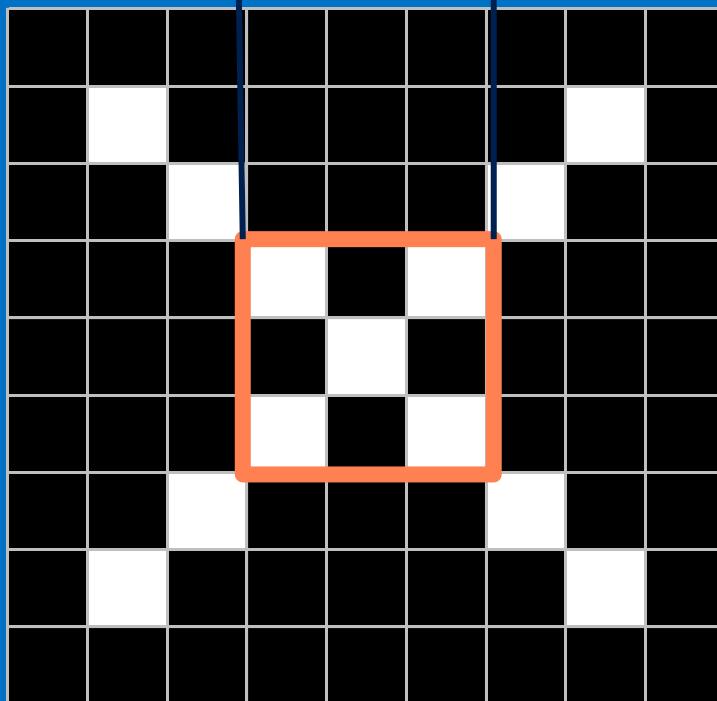
| | | |
|----|----|----|
| -1 | -1 | 1 |
| -1 | 1 | -1 |
| 1 | -1 | -1 |



| | | |
|----|----|----|
| 1 | -1 | -1 |
| -1 | 1 | -1 |
| -1 | -1 | 1 |

| | | |
|----|----|----|
| 1 | -1 | 1 |
| -1 | 1 | -1 |
| 1 | -1 | 1 |

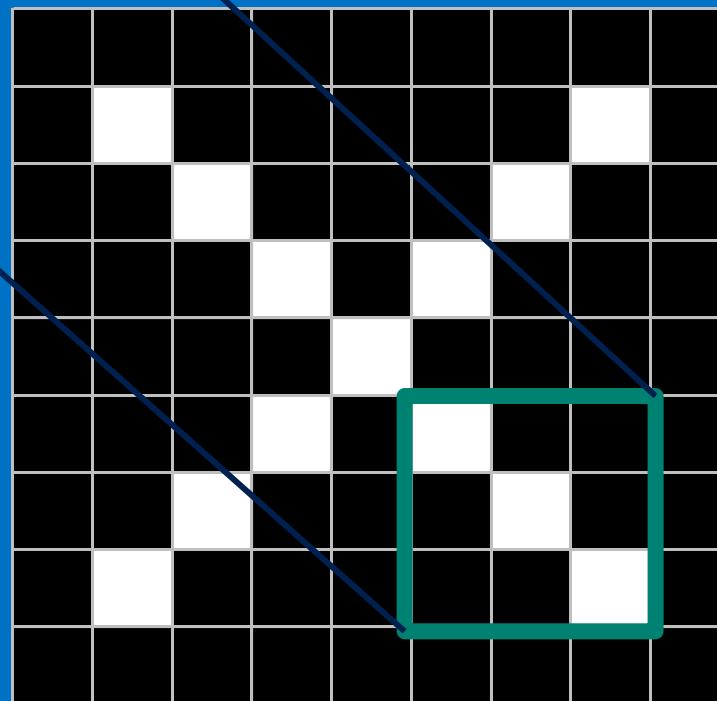
| | | |
|----|----|----|
| -1 | -1 | 1 |
| -1 | 1 | -1 |
| 1 | -1 | -1 |



| | | |
|----|----|----|
| 1 | -1 | -1 |
| -1 | 1 | -1 |
| -1 | -1 | 1 |

| | | |
|----|----|----|
| 1 | -1 | 1 |
| -1 | 1 | -1 |
| 1 | -1 | 1 |

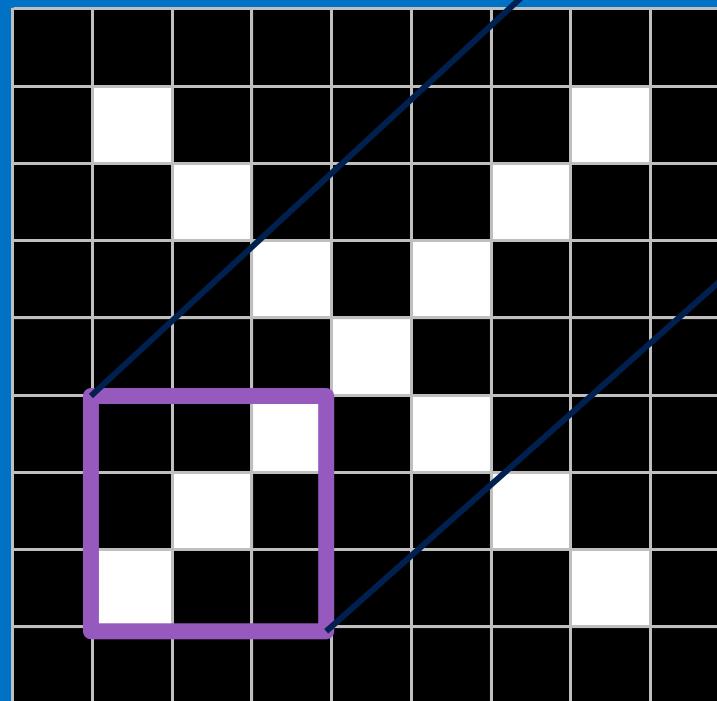
| | | |
|----|----|----|
| -1 | -1 | 1 |
| -1 | 1 | -1 |
| 1 | -1 | -1 |



| | | |
|----|----|----|
| 1 | -1 | -1 |
| -1 | 1 | -1 |
| -1 | -1 | 1 |

| | | |
|----|----|----|
| 1 | -1 | 1 |
| -1 | 1 | -1 |
| 1 | -1 | 1 |

| | | |
|----|----|----|
| -1 | -1 | 1 |
| -1 | 1 | -1 |
| 1 | -1 | -1 |



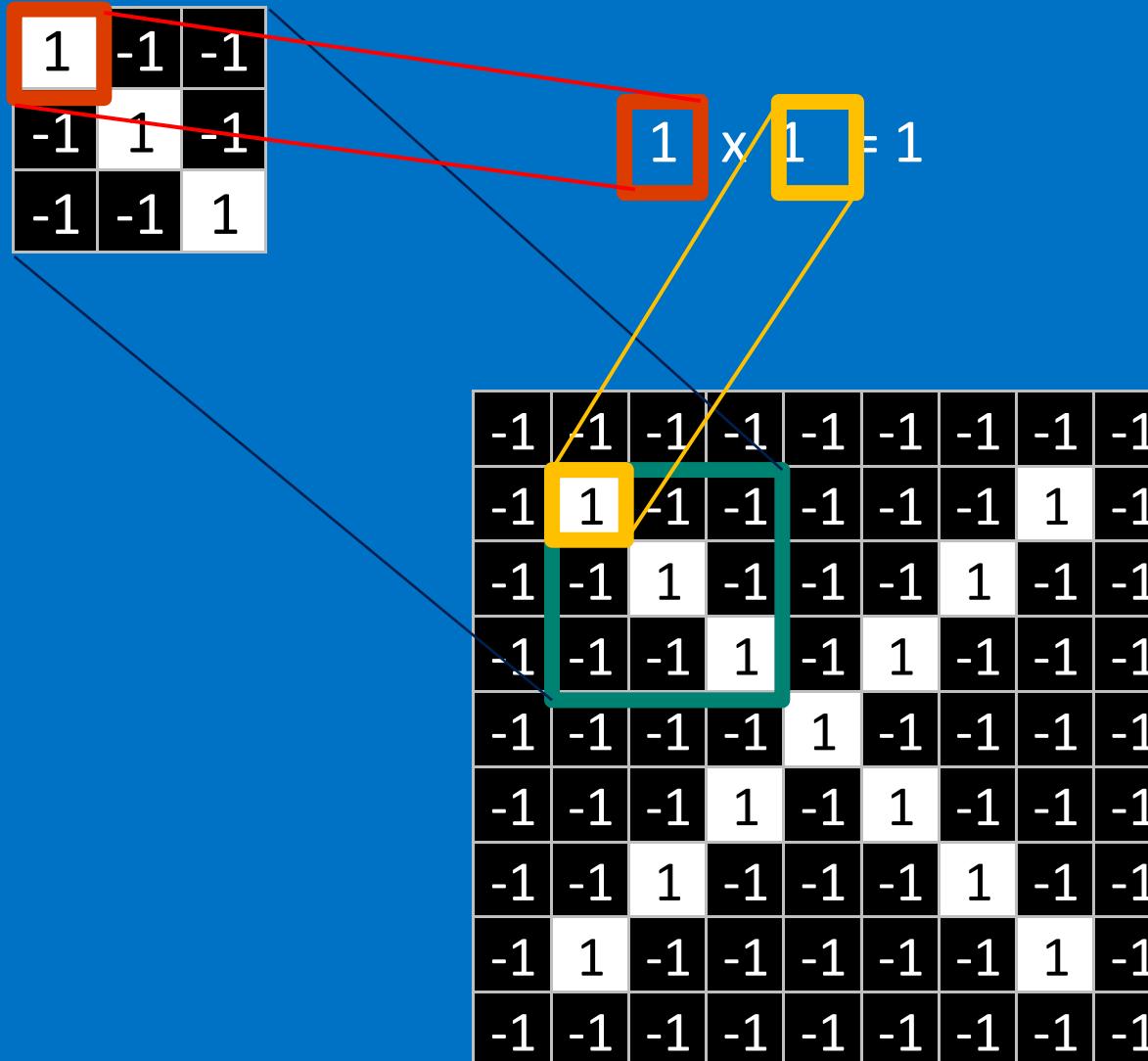
Filtering: The math behind the match

| | | |
|----|----|----|
| 1 | -1 | -1 |
| -1 | 1 | -1 |
| -1 | -1 | 1 |

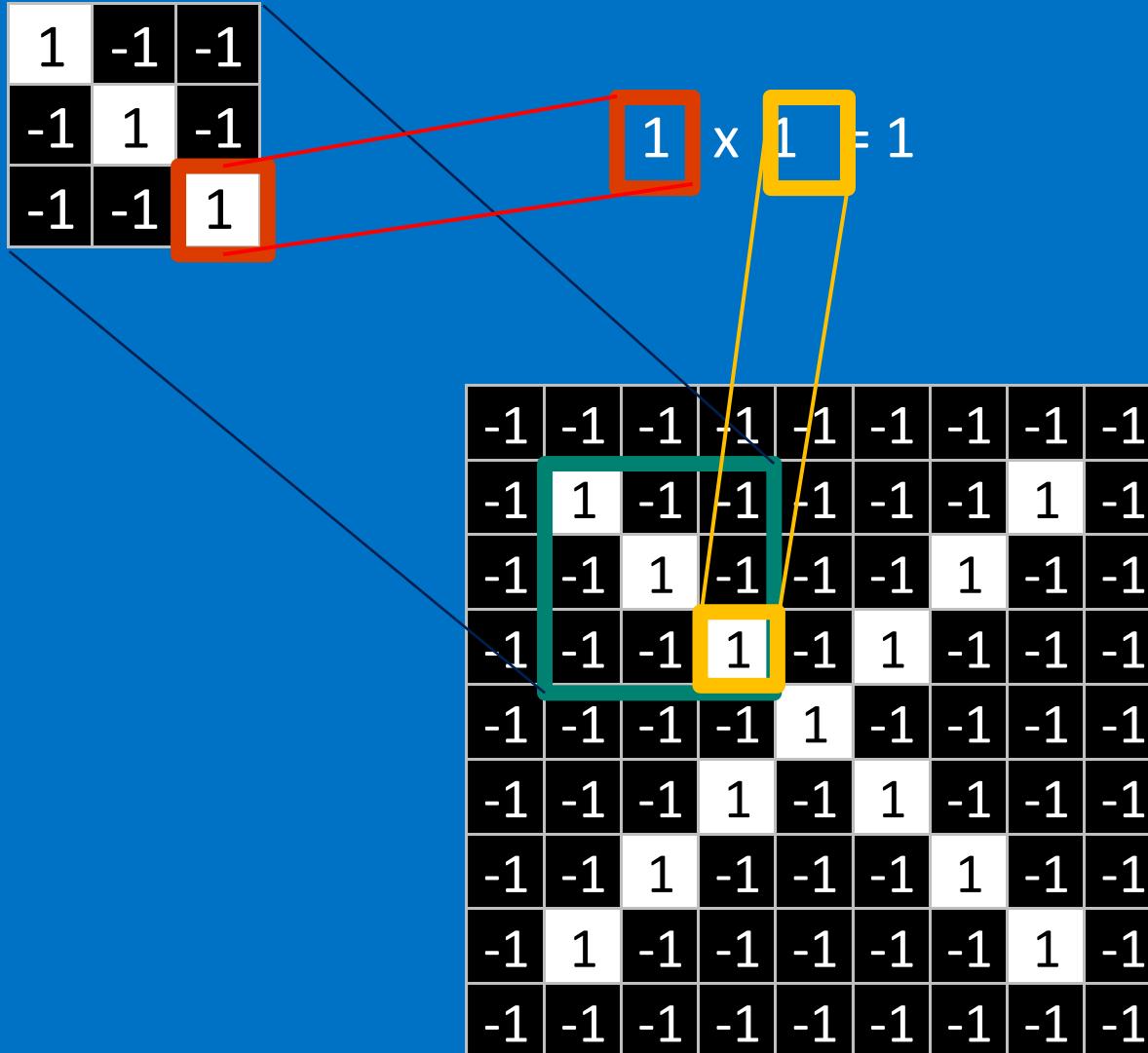
Filtering: The math behind the match

1. Line up the feature and the image patch.
2. Multiply each image pixel by the corresponding feature pixel.
3. Add them up.
4. Divide by the total number of pixels in the feature.

Filtering: The math behind the match



Filtering: The math behind the match



| | | |
|---|---|---|
| 1 | 1 | 1 |
| 1 | 1 | 1 |
| 1 | 1 | 1 |

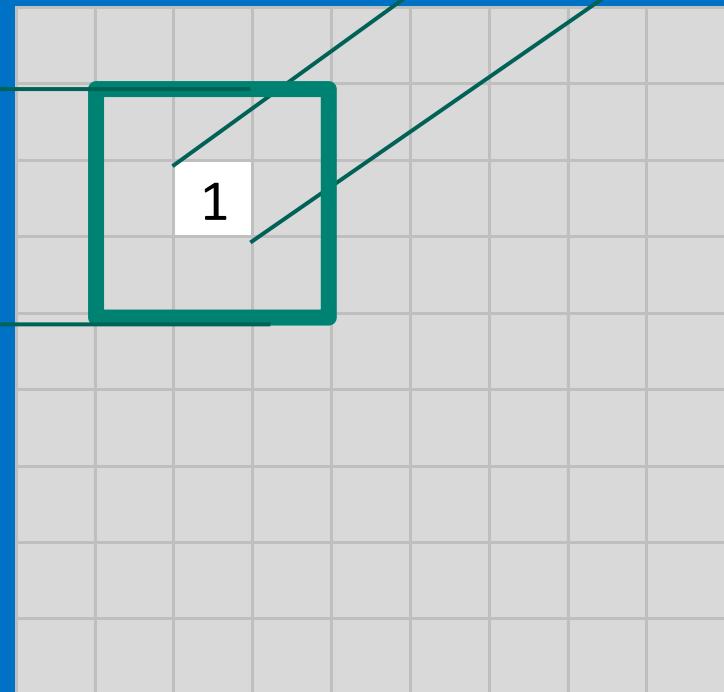
Filtering: The math behind the match

| | | |
|----|----|----|
| 1 | -1 | -1 |
| -1 | 1 | -1 |
| -1 | -1 | 1 |

| | | |
|---|---|---|
| 1 | 1 | 1 |
| 1 | 1 | 1 |
| 1 | 1 | 1 |

$$\frac{1 + 1 + 1 + 1 + 1 + 1 + 1 + 1}{9} = 1$$

| | | | | | | | | |
|----|----|----|----|----|----|----|----|----|
| -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 |
| -1 | 1 | -1 | -1 | -1 | -1 | -1 | 1 | -1 |
| -1 | -1 | 1 | -1 | -1 | -1 | 1 | -1 | -1 |
| -1 | -1 | -1 | 1 | -1 | 1 | -1 | -1 | -1 |
| -1 | -1 | -1 | -1 | 1 | -1 | -1 | -1 | -1 |
| -1 | -1 | -1 | -1 | 1 | -1 | -1 | -1 | -1 |
| -1 | -1 | -1 | 1 | -1 | 1 | -1 | -1 | -1 |
| -1 | -1 | 1 | -1 | -1 | -1 | 1 | -1 | -1 |
| -1 | 1 | -1 | -1 | -1 | -1 | -1 | 1 | -1 |
| -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 |



Convolution: Trying every possible match

| | | |
|----|----|----|
| 1 | -1 | -1 |
| -1 | 1 | -1 |
| -1 | -1 | 1 |

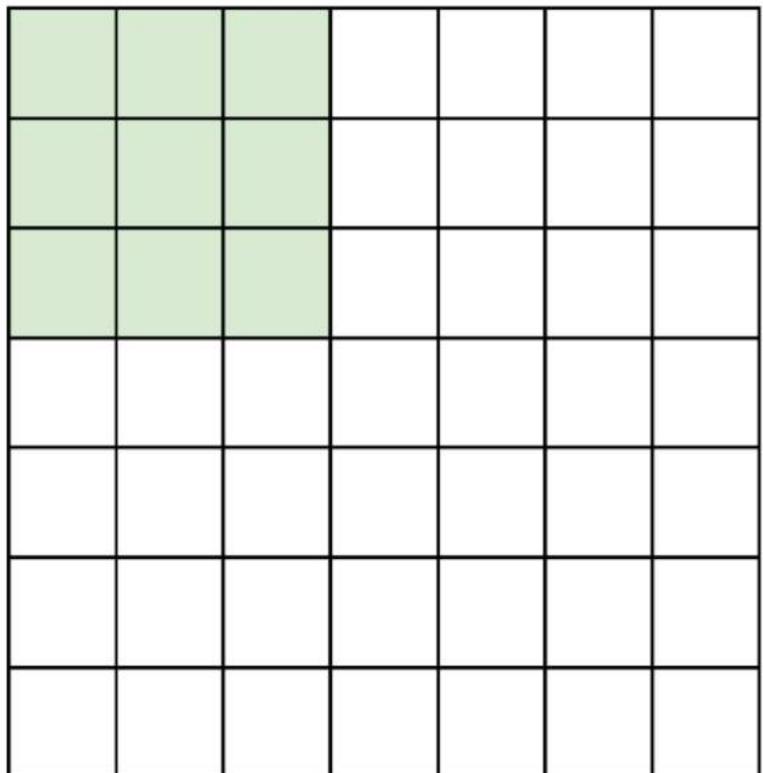
| | | | | | | | | |
|----|----|----|----|----|----|----|----|----|
| -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 |
| -1 | 1 | -1 | -1 | -1 | -1 | -1 | 1 | -1 |
| -1 | -1 | 1 | -1 | -1 | -1 | 1 | -1 | -1 |
| -1 | -1 | -1 | 1 | -1 | 1 | -1 | -1 | -1 |
| -1 | -1 | -1 | -1 | 1 | -1 | -1 | -1 | -1 |
| -1 | -1 | -1 | 1 | -1 | 1 | -1 | -1 | -1 |
| -1 | -1 | -1 | 1 | -1 | 1 | -1 | -1 | -1 |
| -1 | -1 | 1 | -1 | -1 | -1 | 1 | -1 | -1 |
| -1 | 1 | -1 | -1 | -1 | -1 | -1 | 1 | -1 |
| -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 |



| | | | | | | |
|-------|-------|-------|-------|-------|-------|-------|
| 0.77 | -0.11 | 0.11 | 0.33 | 0.55 | -0.11 | 0.33 |
| -0.11 | 1.00 | -0.11 | 0.33 | -0.11 | 0.11 | -0.11 |
| 0.11 | -0.11 | 1.00 | -0.33 | 0.11 | -0.11 | 0.55 |
| 0.33 | 0.33 | -0.33 | 0.55 | -0.33 | 0.33 | 0.33 |
| 0.55 | -0.11 | 0.11 | -0.33 | 1.00 | -0.11 | 0.11 |
| -0.11 | 0.11 | -0.11 | 0.33 | -0.11 | 1.00 | -0.11 |
| 0.33 | -0.11 | 0.55 | 0.33 | 0.11 | -0.11 | 0.77 |

A closer look at spatial dimensions:

7

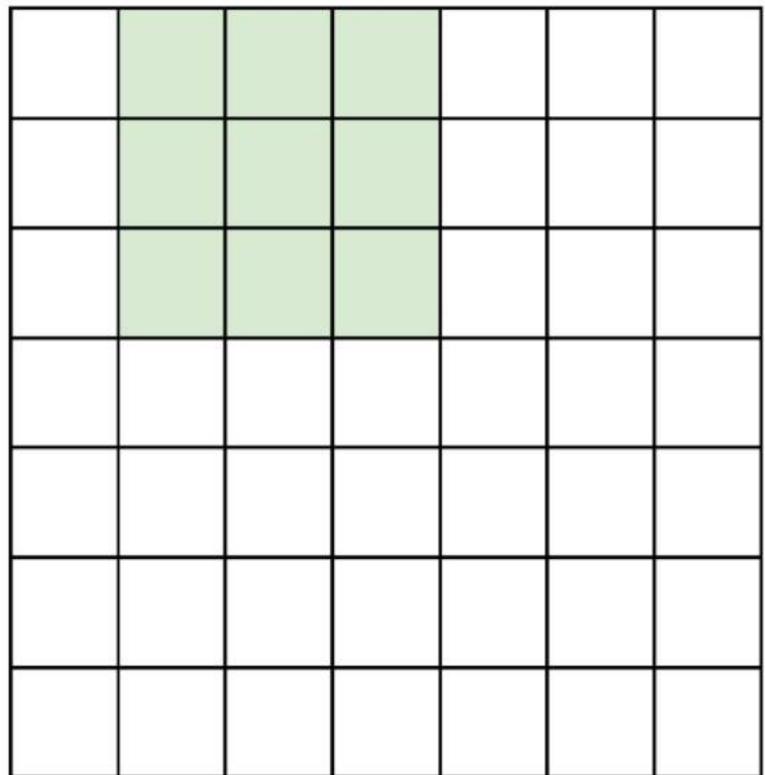


7x7 input (spatially)
assume 3x3 filter

7

A closer look at spatial dimensions:

7

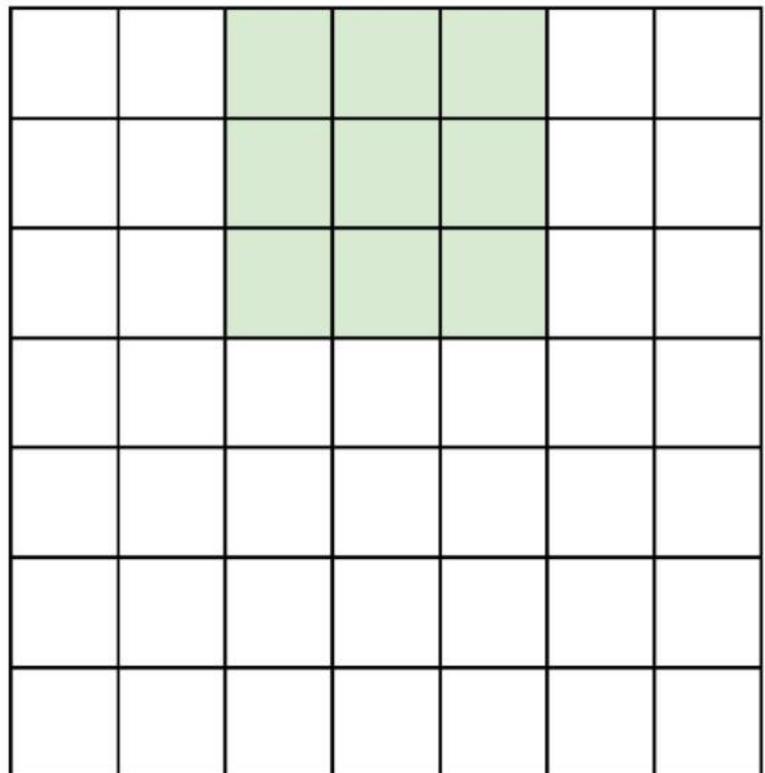


7x7 input (spatially)
assume 3x3 filter

7

A closer look at spatial dimensions:

7

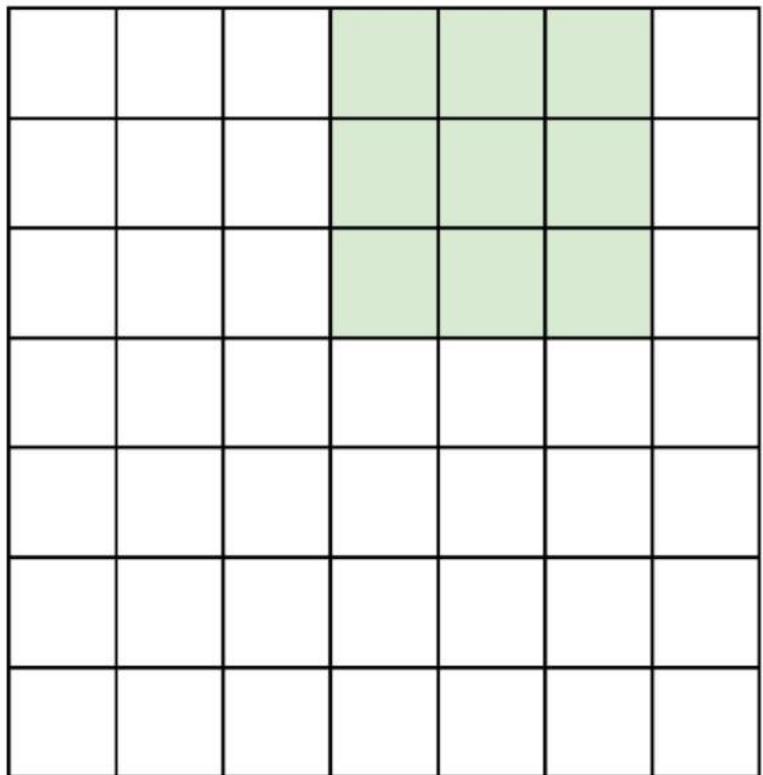


7x7 input (spatially)
assume 3x3 filter

7

A closer look at spatial dimensions:

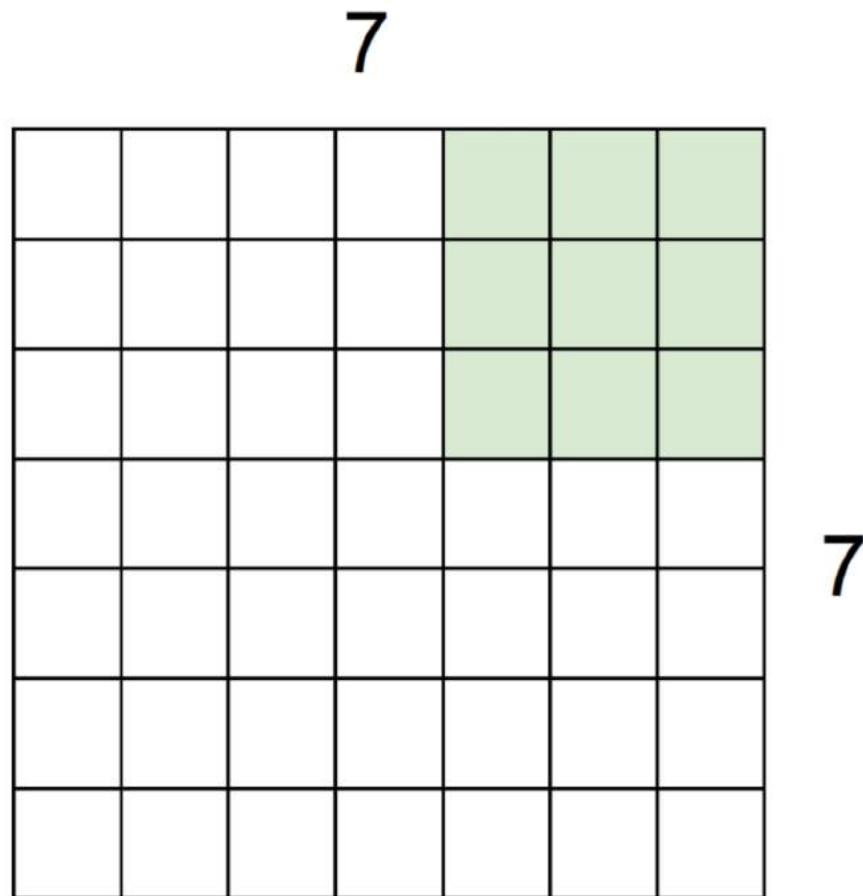
7



7x7 input (spatially)
assume 3x3 filter

7

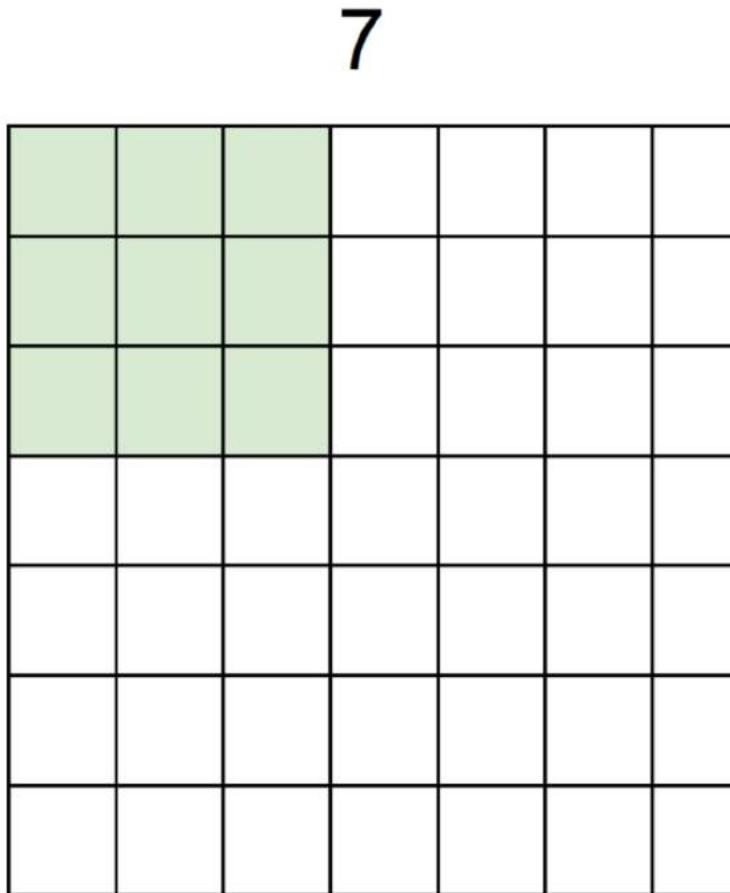
A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter

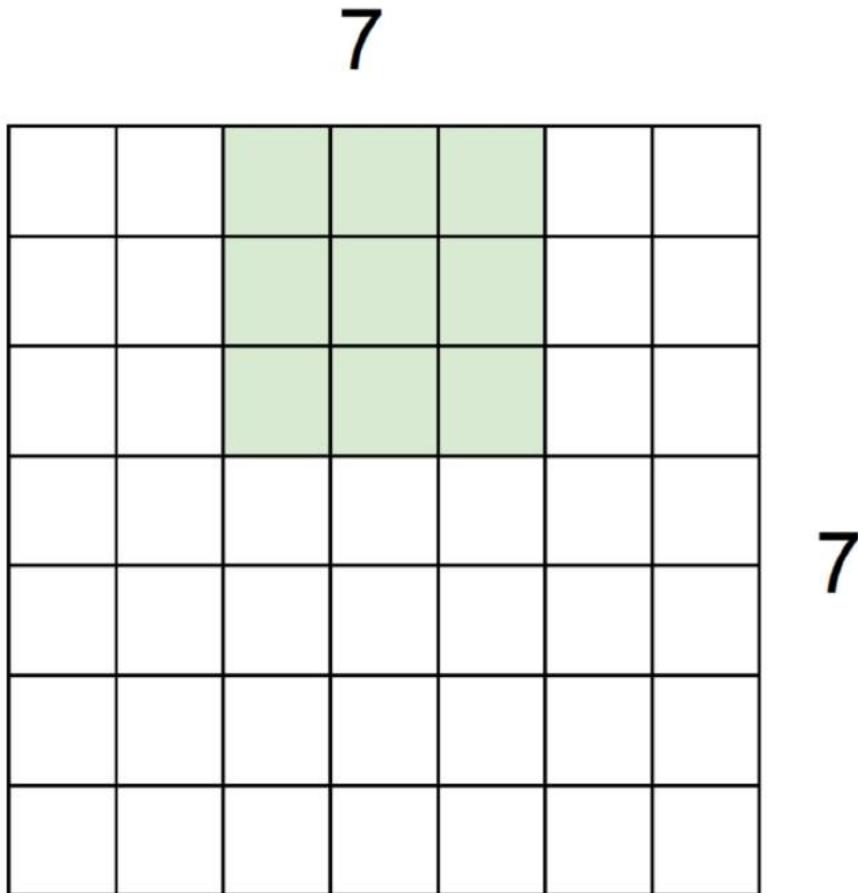
=> 5x5 output

A closer look at spatial dimensions:



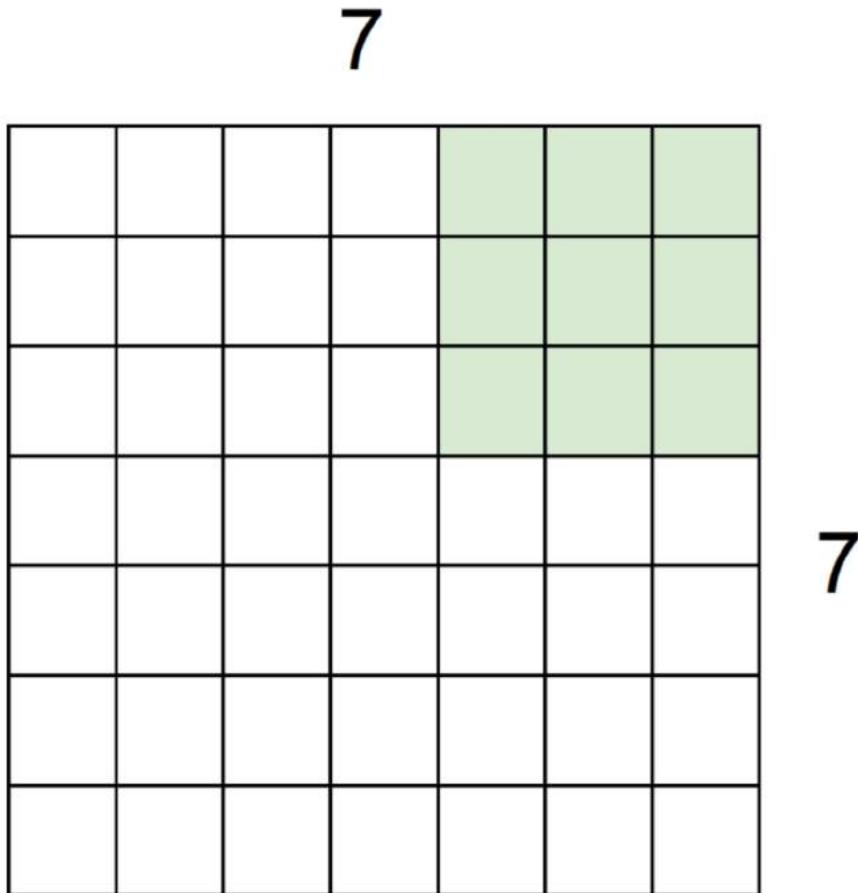
7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**

A closer look at spatial dimensions:



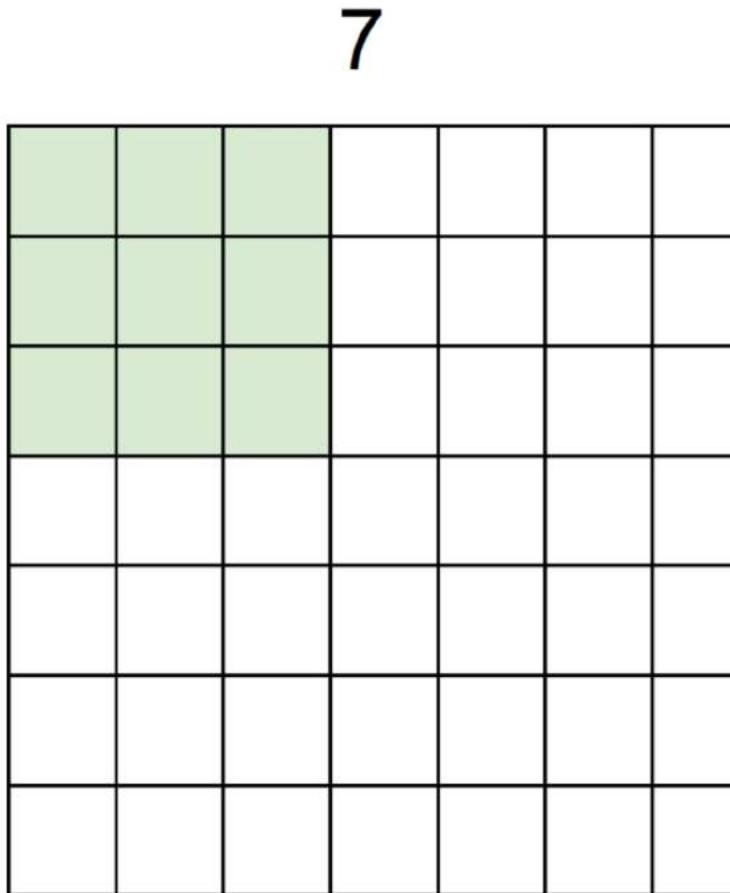
7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**

A closer look at spatial dimensions:



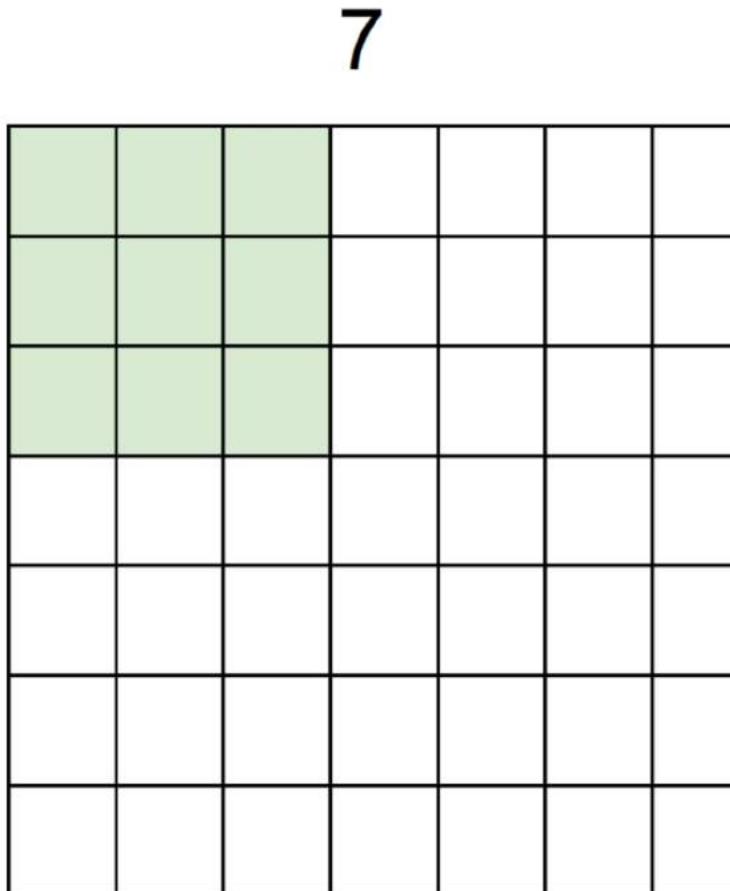
7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**
=> 3x3 output!

A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter
applied **with stride 3?**

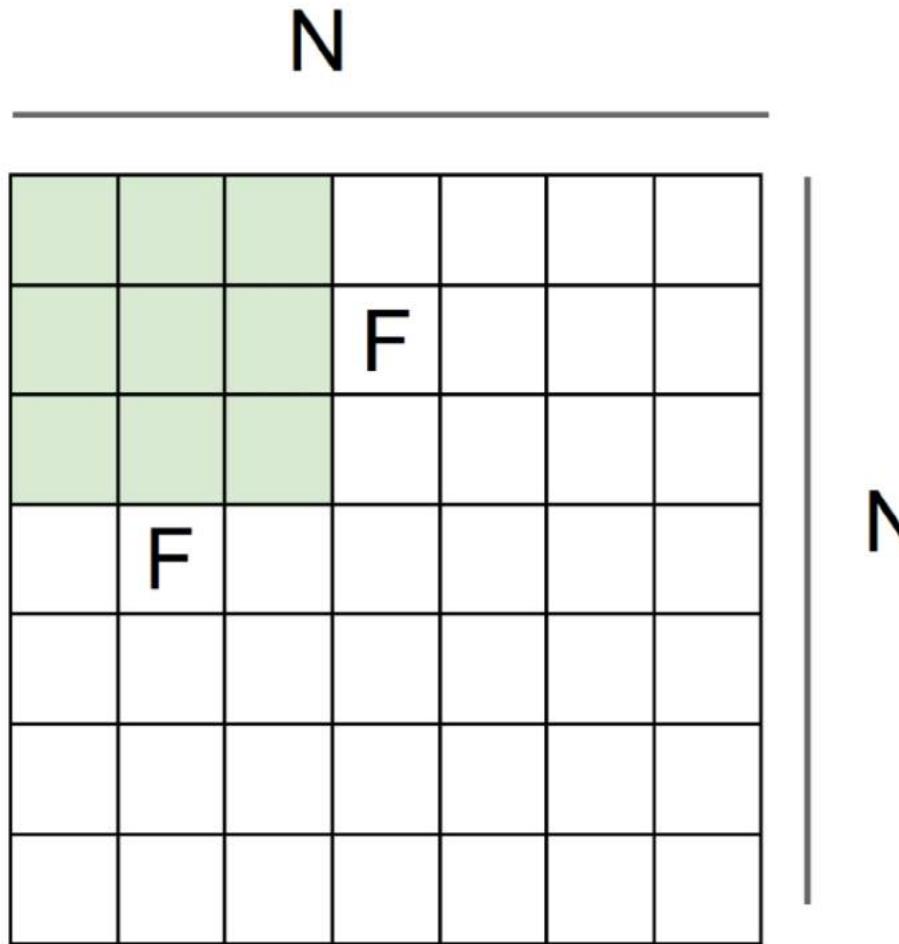
A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter
applied **with stride 3?**

7

doesn't fit!
cannot apply 3x3 filter on
7x7 input with stride 3.



Output size:
 $(N - F) / \text{stride} + 1$

e.g. $N = 7$, $F = 3$:
stride 1 => $(7 - 3)/1 + 1 = 5$
stride 2 => $(7 - 3)/2 + 1 = 3$
stride 3 => $(7 - 3)/3 + 1 = 2.33$:\

In practice: Common to zero pad the border

| | | | | | | | | |
|---|---|---|---|---|---|--|--|--|
| 0 | 0 | 0 | 0 | 0 | 0 | | | |
| 0 | | | | | | | | |
| 0 | | | | | | | | |
| 0 | | | | | | | | |
| 0 | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |

e.g. input 7x7

3x3 filter, applied with stride 1

pad with 1 pixel border => what is the output?

(recall:)

$$(N - F) / \text{stride} + 1$$

In practice: Common to zero pad the border

| | | | | | | | |
|---|---|---|---|---|---|--|--|
| 0 | 0 | 0 | 0 | 0 | 0 | | |
| 0 | | | | | | | |
| 0 | | | | | | | |
| 0 | | | | | | | |
| 0 | | | | | | | |
| | | | | | | | |
| | | | | | | | |
| | | | | | | | |

e.g. input 7x7

3x3 filter, applied with stride 1

pad with 1 pixel border => what is the output?

7x7 output!

In practice: Common to zero pad the border

| | | | | | | | |
|---|---|---|---|---|---|--|--|
| 0 | 0 | 0 | 0 | 0 | 0 | | |
| 0 | | | | | | | |
| 0 | | | | | | | |
| 0 | | | | | | | |
| 0 | | | | | | | |
| | | | | | | | |
| | | | | | | | |
| | | | | | | | |

e.g. input 7x7

3x3 filter, applied with stride 1

pad with 1 pixel border => what is the output?

7x7 output!

in general, common to see CONV layers with stride 1, filters of size FxF, and zero-padding with $(F-1)/2$. (will preserve size spatially)

e.g. $F = 3 \Rightarrow$ zero pad with 1

$F = 5 \Rightarrow$ zero pad with 2

$F = 7 \Rightarrow$ zero pad with 3

Input Volume (+pad 1) (7x7x3)

 $x[:, :, 0]$

| | | | | | | |
|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 2 | 2 | 0 | 1 | 2 | 0 |
| 0 | 2 | 0 | 2 | 0 | 2 | 0 |
| 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 | 2 | 1 | 0 |
| 0 | 1 | 2 | 1 | 1 | 1 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Filter W0 (3x3x3)

 $w0[:, :, 0]$

| | | |
|----|---|----|
| 1 | 1 | 0 |
| -1 | 0 | 0 |
| 0 | 0 | -1 |

 $w0[:, :, 1]$

| | | |
|----|----|----|
| -1 | 1 | 1 |
| -1 | -1 | -1 |
| 0 | 1 | -1 |

 $w0[:, :, 2]$

| | | |
|----|----|---|
| -1 | 1 | 0 |
| 1 | -1 | 0 |
| 1 | 0 | 1 |

Bias b0 (1x1x1)

 $b0[:, :, 0]$

| |
|---|
| 1 |
|---|

Filter W1 (3x3x3)

 $w1[:, :, 0]$

| | | |
|---|----|---|
| 0 | -1 | 1 |
| 0 | 0 | 1 |
| 0 | -1 | 0 |

 $w1[:, :, 1]$

| | | |
|----|----|----|
| 1 | 0 | -1 |
| 1 | 0 | -1 |
| -1 | -1 | -1 |

 $w1[:, :, 2]$

| | | |
|----|---|----|
| -1 | 1 | 0 |
| 1 | 0 | -1 |
| 0 | 1 | 0 |

Bias b1 (1x1x1)

 $b1[:, :, 0]$

| |
|---|
| 0 |
|---|

Output Volume (3x3x2)

 $o[:, :, 0]$

| | | |
|------------|----|----|
| -4 | -1 | 4 |
| 4 | -2 | 1 |
| -2 | 2 | 0 |
| o[:, :, 1] | -5 | -1 |
| -3 | 0 | -4 |
| 3 | 2 | 5 |

toggle movement

Summary. To summarize, the Conv Layer:

- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires four hyperparameters:
 - Number of filters K ,
 - their spatial extent F ,
 - the stride S ,
 - the amount of zero padding P .
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
 - $W_2 = (W_1 - F + 2P)/S + 1$
 - $H_2 = (H_1 - F + 2P)/S + 1$ (i.e. width and height are computed equally by symmetry)
 - $D_2 = K$
- With parameter sharing, it introduces $F \cdot F \cdot D_1$ weights per filter, for a total of $(F \cdot F \cdot D_1) \cdot K$ weights and K biases.
- In the output volume, the d -th depth slice (of size $W_2 \times H_2$) is the result of performing a valid convolution of the d -th filter over the input volume with a stride of S , and then offset by d -th bias.

Common settings:

Summary. To summarize, the Conv Layer:

- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires four hyperparameters:
 - Number of filters K ,
 - their spatial extent F ,
 - the stride S ,
 - the amount of zero padding P .
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
 - $W_2 = (W_1 - F + 2P)/S + 1$
 - $H_2 = (H_1 - F + 2P)/S + 1$ (i.e. width and height are computed equally by symmetry)
 - $D_2 = K$
- With parameter sharing, it introduces $F \cdot F \cdot D_1$ weights per filter, for a total of $(F \cdot F \cdot D_1) \cdot K$ weights and K biases.
- In the output volume, the d -th depth slice (of size $W_2 \times H_2$) is the result of performing a valid convolution of the d -th filter over the input volume with a stride of S , and then offset by d -th bias.

K = (powers of 2, e.g. 32, 64, 128, 512)

- $F = 3, S = 1, P = 1$
- $F = 5, S = 1, P = 2$
- $F = 5, S = 2, P = ?$ (whatever fits)
- $F = 1, S = 1, P = 0$

Convolution: Trying every possible match

| | | | | | | | | | |
|----|----|----|----|----|----|----|----|----|----|
| -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 |
| -1 | 1 | -1 | -1 | -1 | -1 | -1 | 1 | -1 | |
| -1 | -1 | 1 | -1 | -1 | -1 | 1 | -1 | -1 | -1 |
| -1 | -1 | -1 | 1 | -1 | 1 | -1 | -1 | -1 | -1 |
| -1 | -1 | -1 | -1 | 1 | -1 | -1 | -1 | -1 | -1 |
| -1 | -1 | -1 | -1 | 1 | -1 | -1 | -1 | -1 | -1 |
| -1 | -1 | -1 | 1 | -1 | 1 | -1 | -1 | -1 | -1 |
| -1 | -1 | 1 | -1 | -1 | -1 | 1 | -1 | -1 | -1 |
| -1 | 1 | -1 | -1 | -1 | -1 | -1 | 1 | -1 | -1 |
| -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 |



| | | |
|----|----|----|
| 1 | -1 | -1 |
| -1 | 1 | -1 |
| -1 | -1 | 1 |

=

| | | | | | | |
|-------|-------|-------|-------|-------|-------|-------|
| 0.77 | -0.11 | 0.11 | 0.33 | 0.55 | -0.11 | 0.33 |
| -0.11 | 1.00 | -0.11 | 0.33 | -0.11 | 0.11 | -0.11 |
| 0.11 | -0.11 | 1.00 | -0.33 | 0.11 | -0.11 | 0.55 |
| 0.33 | 0.33 | -0.33 | 0.55 | -0.33 | 0.33 | 0.33 |
| 0.55 | -0.11 | 0.11 | -0.33 | 1.00 | -0.11 | 0.11 |
| -0.11 | 0.11 | -0.11 | 0.33 | -0.11 | 1.00 | -0.11 |
| 0.33 | -0.11 | 0.55 | 0.33 | 0.11 | -0.11 | 0.77 |

| | | | | | | | | |
|----|----|----|----|----|----|----|----|----|
| -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 |
| -1 | 1 | -1 | -1 | -1 | -1 | -1 | 1 | -1 |
| -1 | -1 | 1 | -1 | -1 | -1 | 1 | -1 | -1 |
| -1 | -1 | -1 | 1 | -1 | 1 | -1 | -1 | -1 |
| -1 | -1 | -1 | -1 | 1 | -1 | -1 | -1 | -1 |
| -1 | -1 | -1 | 1 | -1 | 1 | -1 | -1 | -1 |
| -1 | -1 | -1 | 1 | -1 | 1 | -1 | -1 | -1 |
| -1 | -1 | 1 | -1 | -1 | -1 | 1 | -1 | -1 |
| -1 | 1 | -1 | -1 | -1 | -1 | 1 | 1 | -1 |
| -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 |



$$\begin{array}{|c|c|c|} \hline 1 & -1 & -1 \\ \hline -1 & 1 & -1 \\ \hline -1 & -1 & 1 \\ \hline \end{array}$$

=

| | | | | | | |
|-------|-------|-------|-------|-------|-------|-------|
| 0.77 | -0.11 | 0.11 | 0.33 | 0.55 | -0.11 | 0.33 |
| -0.11 | 1.00 | -0.11 | 0.33 | -0.11 | 0.11 | -0.11 |
| 0.11 | -0.11 | 1.00 | -0.33 | 0.11 | -0.11 | 0.55 |
| 0.33 | 0.33 | -0.33 | 0.55 | -0.33 | 0.33 | 0.33 |
| 0.55 | -0.11 | 0.11 | -0.33 | 1.00 | -0.11 | 0.11 |
| -0.11 | 0.11 | -0.11 | 0.33 | -0.11 | 1.00 | -0.11 |
| 0.33 | -0.11 | 0.55 | 0.33 | 0.11 | -0.11 | 0.77 |

| | | | | | | | | |
|----|----|----|----|----|----|----|----|----|
| -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 |
| -1 | 1 | -1 | -1 | -1 | -1 | -1 | 1 | -1 |
| -1 | -1 | 1 | -1 | -1 | -1 | 1 | -1 | -1 |
| -1 | -1 | -1 | 1 | -1 | 1 | -1 | -1 | -1 |
| -1 | -1 | -1 | -1 | 1 | -1 | -1 | -1 | -1 |
| -1 | -1 | -1 | 1 | -1 | 1 | -1 | -1 | -1 |
| -1 | -1 | 1 | -1 | -1 | -1 | 1 | -1 | -1 |
| -1 | 1 | -1 | -1 | -1 | -1 | 1 | 1 | -1 |
| -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 |



$$\begin{array}{|c|c|c|} \hline 1 & -1 & 1 \\ \hline -1 & 1 & -1 \\ \hline 1 & -1 & 1 \\ \hline \end{array}$$

=

| | | | | | | |
|-------|-------|-------|-------|-------|-------|-------|
| 0.33 | -0.55 | 0.11 | -0.11 | 0.11 | -0.55 | 0.33 |
| -0.55 | 0.55 | -0.55 | 0.33 | -0.55 | 0.55 | -0.55 |
| 0.11 | -0.55 | 0.55 | -0.77 | 0.55 | -0.55 | 0.11 |
| -0.11 | 0.33 | -0.77 | 1.00 | -0.77 | 0.33 | -0.11 |
| 0.11 | -0.55 | 0.55 | -0.77 | 0.55 | -0.55 | 0.11 |
| -0.55 | 0.55 | -0.55 | 0.33 | -0.55 | 0.55 | -0.55 |
| 0.33 | -0.55 | 0.11 | -0.11 | 0.11 | -0.55 | 0.33 |

| | | | | | | | | |
|----|----|----|----|----|----|----|----|----|
| -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 |
| -1 | 1 | -1 | -1 | -1 | -1 | -1 | 1 | -1 |
| -1 | -1 | 1 | -1 | -1 | -1 | 1 | -1 | -1 |
| -1 | -1 | -1 | 1 | -1 | 1 | -1 | -1 | -1 |
| -1 | -1 | -1 | -1 | 1 | -1 | -1 | -1 | -1 |
| -1 | -1 | -1 | 1 | -1 | 1 | -1 | -1 | -1 |
| -1 | -1 | 1 | -1 | -1 | -1 | 1 | -1 | -1 |
| -1 | 1 | -1 | -1 | -1 | -1 | 1 | 1 | -1 |
| -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 |



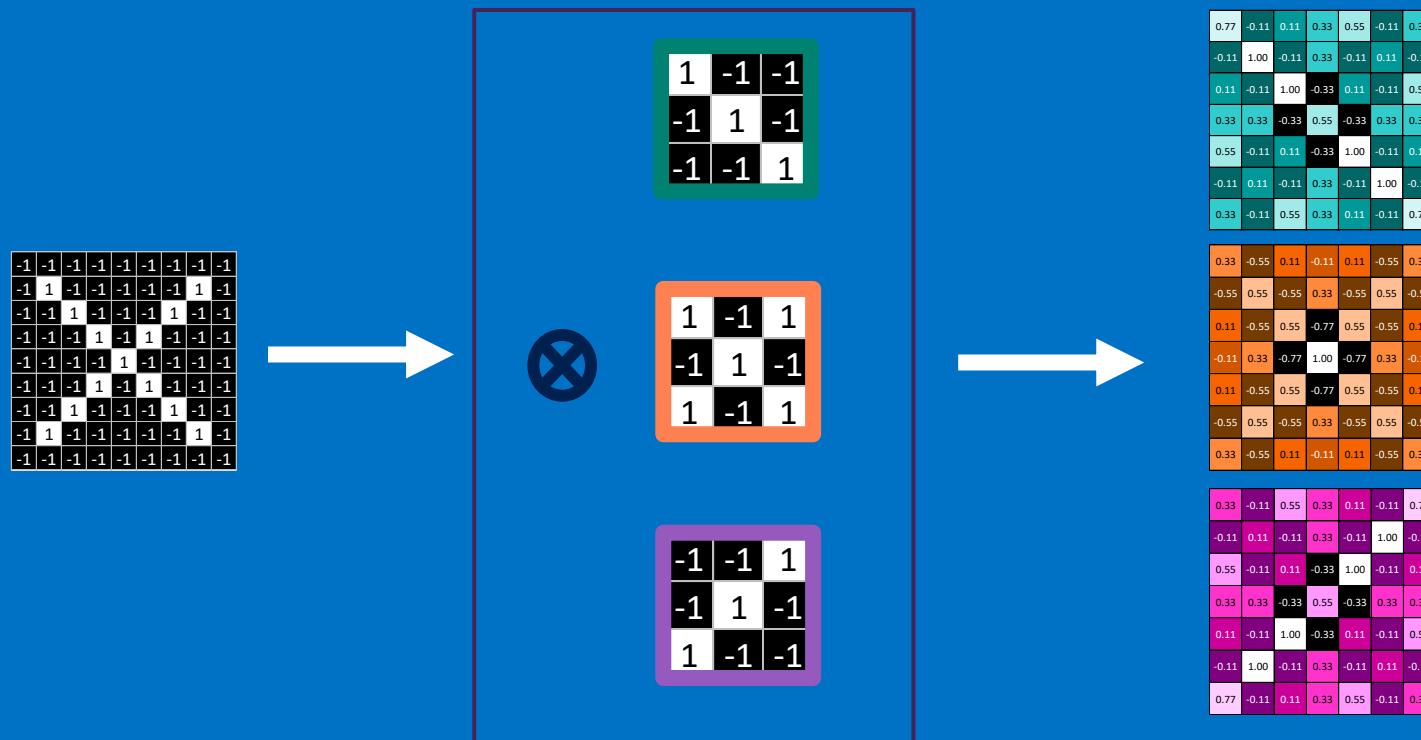
$$\begin{array}{|c|c|c|} \hline -1 & -1 & 1 \\ \hline -1 & 1 & -1 \\ \hline 1 & -1 & -1 \\ \hline \end{array}$$

=

| | | | | | | |
|-------|-------|-------|-------|-------|-------|-------|
| 0.33 | -0.11 | 0.55 | 0.33 | 0.11 | -0.11 | 0.77 |
| -0.11 | 0.11 | -0.11 | 0.33 | -0.11 | 1.00 | -0.11 |
| 0.55 | -0.11 | 0.11 | -0.33 | 1.00 | -0.11 | 0.11 |
| 0.33 | 0.33 | -0.33 | 0.55 | -0.33 | 0.33 | 0.33 |
| 0.11 | -0.11 | 1.00 | -0.33 | 0.11 | -0.11 | 0.55 |
| -0.11 | 1.00 | -0.11 | 0.33 | -0.11 | 0.11 | -0.11 |
| 0.77 | -0.11 | 0.11 | 0.33 | 0.55 | -0.11 | 0.33 |

Convolution layer

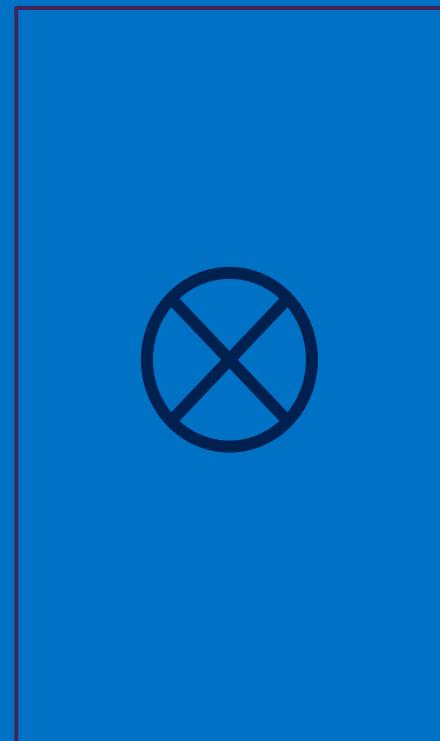
One image becomes a stack of filtered images



Convolution layer

One image becomes a stack of filtered images

| | | | | | | | | | |
|----|----|----|----|----|----|----|----|----|----|
| -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 |
| -1 | 1 | -1 | -1 | -1 | -1 | -1 | 1 | -1 | |
| -1 | -1 | 1 | -1 | -1 | -1 | 1 | -1 | -1 | |
| -1 | -1 | -1 | 1 | -1 | 1 | -1 | -1 | -1 | |
| -1 | -1 | -1 | -1 | 1 | 1 | -1 | -1 | -1 | |
| -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | |
| -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | |
| -1 | 1 | -1 | -1 | -1 | -1 | -1 | 1 | -1 | |
| -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | |



| | | | | | | |
|-------|-------|-------|-------|-------|-------|-------|
| 0.77 | -0.11 | 0.11 | 0.33 | 0.55 | -0.11 | 0.33 |
| -0.11 | 1.00 | -0.11 | 0.33 | -0.11 | 0.11 | -0.11 |
| 0.11 | -0.11 | 1.00 | -0.33 | 0.11 | -0.11 | 0.55 |
| 0.33 | 0.33 | -0.33 | 0.55 | -0.33 | 0.33 | 0.33 |
| 0.55 | -0.11 | 0.11 | -0.33 | 1.00 | -0.11 | 0.11 |
| -0.11 | 0.11 | -0.11 | 0.33 | -0.11 | 1.00 | -0.11 |
| 0.33 | -0.11 | 0.55 | 0.33 | 0.11 | -0.11 | 0.77 |

| | | | | | | |
|-------|-------|-------|-------|-------|-------|-------|
| 0.33 | -0.55 | 0.11 | -0.11 | 0.11 | -0.55 | 0.33 |
| -0.55 | 0.55 | -0.55 | 0.33 | -0.55 | 0.55 | -0.55 |
| 0.11 | -0.55 | 0.55 | -0.77 | 0.55 | -0.55 | 0.11 |
| -0.11 | 0.33 | -0.77 | 1.00 | -0.77 | 0.33 | -0.11 |
| 0.11 | -0.55 | 0.55 | -0.77 | 0.55 | -0.55 | 0.11 |
| -0.55 | 0.55 | -0.55 | 0.33 | -0.55 | 0.55 | -0.55 |
| 0.33 | -0.55 | 0.11 | -0.11 | 0.11 | -0.55 | 0.33 |

| | | | | | | |
|-------|-------|-------|-------|-------|-------|-------|
| 0.33 | -0.11 | 0.55 | 0.33 | 0.11 | -0.11 | 0.77 |
| -0.11 | 0.11 | -0.11 | 0.33 | -0.11 | 1.00 | 0.11 |
| 0.55 | -0.11 | 0.11 | -0.33 | 1.00 | -0.11 | 0.11 |
| 0.33 | 0.33 | -0.33 | 0.55 | -0.33 | 0.33 | 0.33 |
| 0.11 | -0.11 | 1.00 | -0.33 | 0.11 | -0.11 | 0.55 |
| -0.11 | 1.00 | -0.11 | 0.33 | -0.11 | 0.11 | -0.11 |
| 0.77 | -0.11 | 0.11 | 0.33 | 0.55 | -0.11 | 0.33 |

Pooling: Shrinking the image stack

1. Pick a window size (usually 2 or 3).
2. Pick a stride (usually 2).
3. Walk your window across your filtered images.
4. From each window, take the maximum value.

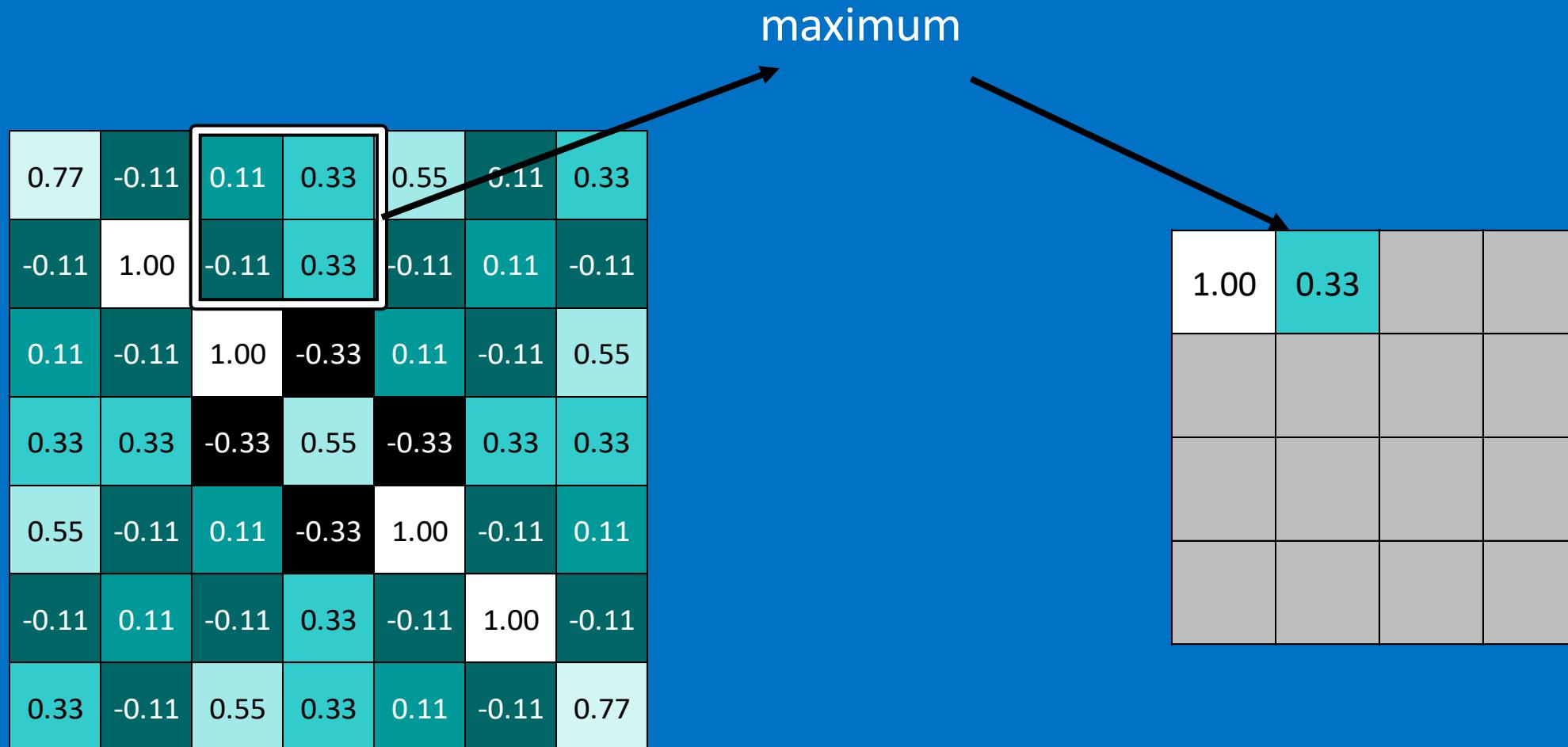
Pooling

| | | | | | | |
|-------|-------|-------|-------|-------|-------|-------|
| 0.77 | -0.11 | 0.11 | 0.33 | 0.55 | -0.11 | 0.33 |
| -0.11 | 1.00 | -0.11 | 0.33 | -0.11 | 0.11 | -0.11 |
| 0.11 | -0.11 | 1.00 | -0.33 | 0.11 | -0.11 | 0.55 |
| 0.33 | 0.33 | -0.33 | 0.55 | -0.33 | 0.33 | 0.33 |
| 0.55 | -0.11 | 0.11 | -0.33 | 1.00 | -0.11 | 0.11 |
| -0.11 | 0.11 | -0.11 | 0.33 | -0.11 | 1.00 | -0.11 |
| 0.33 | -0.11 | 0.55 | 0.33 | 0.11 | -0.11 | 0.77 |

maximum

| | | | |
|------|--|--|--|
| 1.00 | | | |
| | | | |
| | | | |
| | | | |

Pooling



Pooling

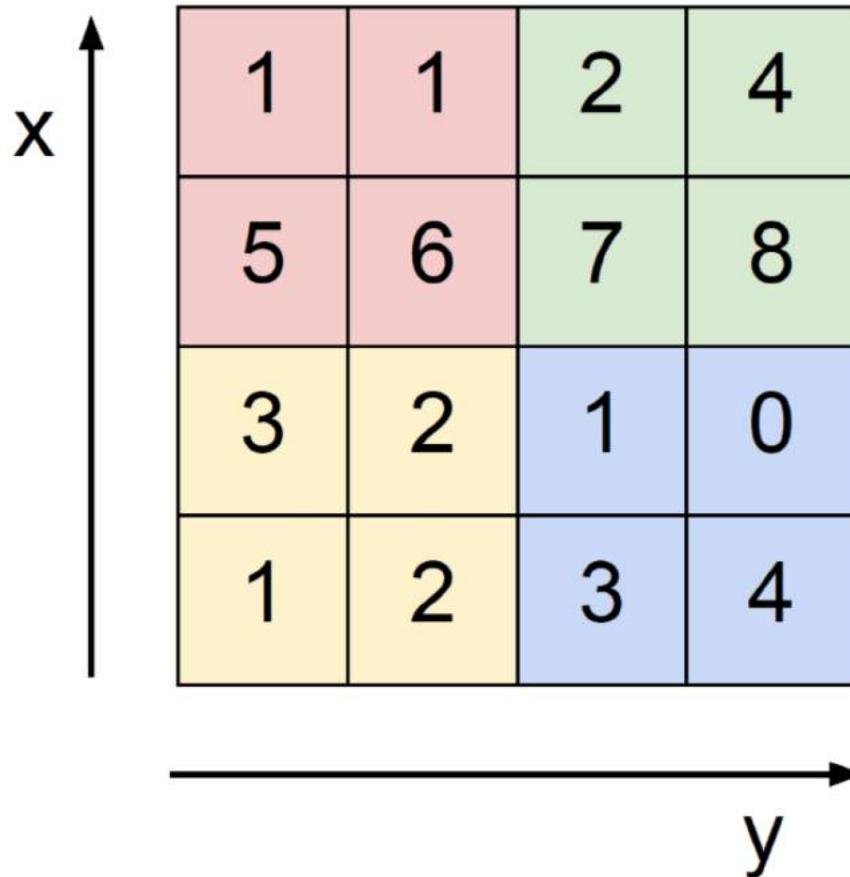
| | | | | | | |
|-------|-------|-------|-------|-------|-------|-------|
| 0.77 | -0.11 | 0.11 | 0.33 | 0.55 | -0.11 | 0.33 |
| -0.11 | 1.00 | -0.11 | 0.33 | -0.11 | 0.11 | -0.11 |
| 0.11 | -0.11 | 1.00 | -0.33 | 0.11 | -0.11 | 0.55 |
| 0.33 | 0.33 | -0.33 | 0.55 | -0.33 | 0.33 | 0.33 |
| 0.55 | -0.11 | 0.11 | -0.33 | 1.00 | -0.11 | 0.11 |
| -0.11 | 0.11 | -0.11 | 0.33 | -0.11 | 1.00 | -0.11 |
| 0.33 | -0.11 | 0.55 | 0.33 | 0.11 | -0.11 | 0.77 |

max pooling

| | | | |
|------|------|------|------|
| 1.00 | 0.33 | 0.55 | 0.33 |
| 0.33 | 1.00 | 0.33 | 0.55 |
| 0.55 | 0.33 | 1.00 | 0.11 |
| 0.33 | 0.55 | 0.11 | 0.77 |

MAX POOLING

Single depth slice



max pool with 2x2 filters
and stride 2

| | |
|---|---|
| 6 | 8 |
| 3 | 4 |

- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires three hyperparameters:
 - their spatial extent F ,
 - the stride S ,
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
 - $W_2 = (W_1 - F)/S + 1$
 - $H_2 = (H_1 - F)/S + 1$
 - $D_2 = D_1$
- Introduces zero parameters since it computes a fixed function of the input
- Note that it is not common to use zero-padding for Pooling layers

Common settings:

- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires three hyperparameters:
 - their spatial extent F ,
 - the stride S ,
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
 - $W_2 = (W_1 - F)/S + 1$
 - $H_2 = (H_1 - F)/S + 1$
 - $D_2 = D_1$
- Introduces zero parameters since it computes a fixed function of the input
- Note that it is not common to use zero-padding for Pooling layers

$F = 2, S = 2$

$F = 3, S = 2$

| | | | | | | |
|-------|-------|-------|-------|-------|-------|-------|
| 0.77 | -0.11 | 0.11 | 0.33 | 0.55 | -0.11 | 0.33 |
| -0.11 | 1.00 | -0.11 | 0.33 | -0.11 | 0.11 | -0.11 |
| 0.11 | -0.11 | 1.00 | -0.33 | 0.11 | -0.11 | 0.55 |
| 0.33 | 0.33 | -0.33 | 0.55 | -0.33 | 0.33 | 0.33 |
| 0.55 | -0.11 | 0.11 | -0.33 | 1.00 | -0.11 | 0.11 |
| -0.11 | 0.11 | -0.11 | 0.33 | -0.11 | 1.00 | -0.11 |
| 0.33 | -0.11 | 0.55 | 0.33 | 0.11 | -0.11 | 0.77 |



| | | | |
|------|------|------|------|
| 1.00 | 0.33 | 0.55 | 0.33 |
| 0.33 | 1.00 | 0.33 | 0.55 |
| 0.55 | 0.33 | 1.00 | 0.11 |
| 0.33 | 0.55 | 0.11 | 0.77 |

| | | | | | | |
|-------|-------|-------|-------|-------|-------|-------|
| 0.33 | -0.55 | 0.11 | -0.11 | 0.11 | -0.55 | 0.33 |
| -0.55 | 0.55 | -0.55 | 0.33 | -0.55 | 0.55 | -0.55 |
| 0.11 | -0.55 | 0.55 | -0.77 | 0.55 | -0.55 | 0.11 |
| -0.11 | 0.33 | -0.77 | 1.00 | -0.77 | 0.33 | -0.11 |
| 0.11 | -0.55 | 0.55 | -0.77 | 0.55 | -0.55 | 0.11 |
| -0.55 | 0.55 | -0.55 | 0.33 | -0.55 | 0.55 | -0.55 |
| 0.33 | -0.55 | 0.11 | -0.11 | 0.11 | -0.55 | 0.33 |



| | | | |
|------|------|------|------|
| 0.55 | 0.33 | 0.55 | 0.33 |
| 0.33 | 1.00 | 0.55 | 0.11 |
| 0.55 | 0.55 | 0.55 | 0.11 |
| 0.33 | 0.11 | 0.11 | 0.33 |

| | | | | | | |
|-------|-------|-------|-------|-------|-------|-------|
| 0.33 | -0.11 | 0.55 | 0.33 | 0.11 | -0.11 | 0.77 |
| -0.11 | 0.11 | -0.11 | 0.33 | -0.11 | 1.00 | -0.11 |
| 0.55 | -0.11 | 0.11 | -0.33 | 1.00 | -0.11 | 0.11 |
| 0.33 | 0.33 | -0.33 | 0.55 | -0.33 | 0.33 | 0.33 |
| 0.11 | -0.11 | 1.00 | -0.33 | 0.11 | -0.11 | 0.55 |
| -0.11 | 1.00 | -0.11 | 0.33 | -0.11 | 0.11 | -0.11 |
| 0.77 | -0.11 | 0.11 | 0.33 | 0.55 | -0.11 | 0.33 |



| | | | |
|------|------|------|------|
| 0.33 | 0.55 | 1.00 | 0.77 |
| 0.55 | 0.55 | 1.00 | 0.33 |
| 1.00 | 1.00 | 0.11 | 0.55 |
| 0.77 | 0.33 | 0.55 | 0.33 |

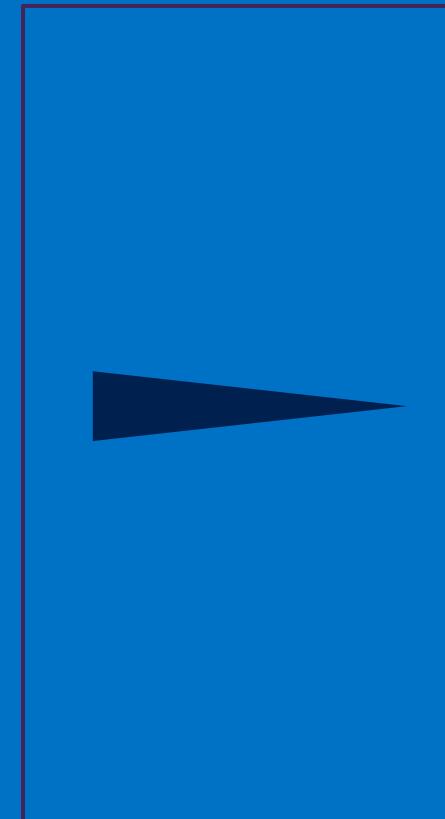
Pooling layer

A stack of images becomes a stack of smaller images.

| | | | | | | |
|-------|-------|-------|-------|-------|-------|-------|
| 0.77 | -0.11 | 0.11 | 0.33 | 0.55 | -0.11 | 0.33 |
| -0.11 | 1.00 | -0.11 | 0.33 | -0.11 | 0.11 | -0.11 |
| 0.11 | -0.11 | 1.00 | -0.33 | 0.11 | -0.11 | 0.55 |
| 0.33 | 0.33 | -0.33 | 0.55 | -0.33 | 0.33 | 0.33 |
| 0.55 | -0.11 | 0.11 | -0.33 | 1.00 | -0.11 | 0.11 |
| -0.11 | 0.11 | -0.11 | 0.33 | -0.11 | 1.00 | -0.11 |
| 0.33 | -0.11 | 0.55 | 0.33 | 0.11 | -0.11 | 0.77 |

| | | | | | | |
|-------|-------|-------|-------|-------|-------|-------|
| 0.33 | -0.55 | 0.11 | -0.11 | 0.11 | -0.55 | 0.33 |
| -0.55 | 0.55 | -0.55 | 0.33 | -0.55 | 0.55 | -0.55 |
| 0.11 | -0.55 | 0.55 | -0.77 | 0.55 | -0.55 | 0.11 |
| -0.11 | 0.33 | -0.77 | 1.00 | -0.77 | 0.33 | -0.11 |
| 0.11 | -0.55 | 0.55 | -0.77 | 0.55 | -0.55 | 0.11 |
| -0.55 | 0.55 | -0.55 | 0.33 | -0.55 | 0.55 | -0.55 |
| 0.33 | -0.55 | 0.11 | -0.11 | 0.11 | -0.55 | 0.33 |

| | | | | | | |
|-------|-------|-------|-------|-------|-------|-------|
| 0.33 | -0.11 | 0.55 | 0.33 | 0.11 | -0.11 | 0.77 |
| -0.11 | 0.11 | -0.11 | 0.33 | -0.11 | 1.00 | -0.11 |
| 0.55 | -0.11 | 0.11 | -0.33 | 1.00 | -0.11 | 0.11 |
| 0.33 | 0.33 | -0.33 | 0.55 | -0.33 | 0.33 | 0.33 |
| 0.11 | -0.11 | 1.00 | -0.33 | 0.11 | -0.11 | 0.55 |
| -0.11 | 1.00 | -0.11 | 0.33 | -0.11 | 0.11 | -0.11 |
| 0.77 | -0.11 | 0.11 | 0.33 | 0.55 | -0.11 | 0.33 |

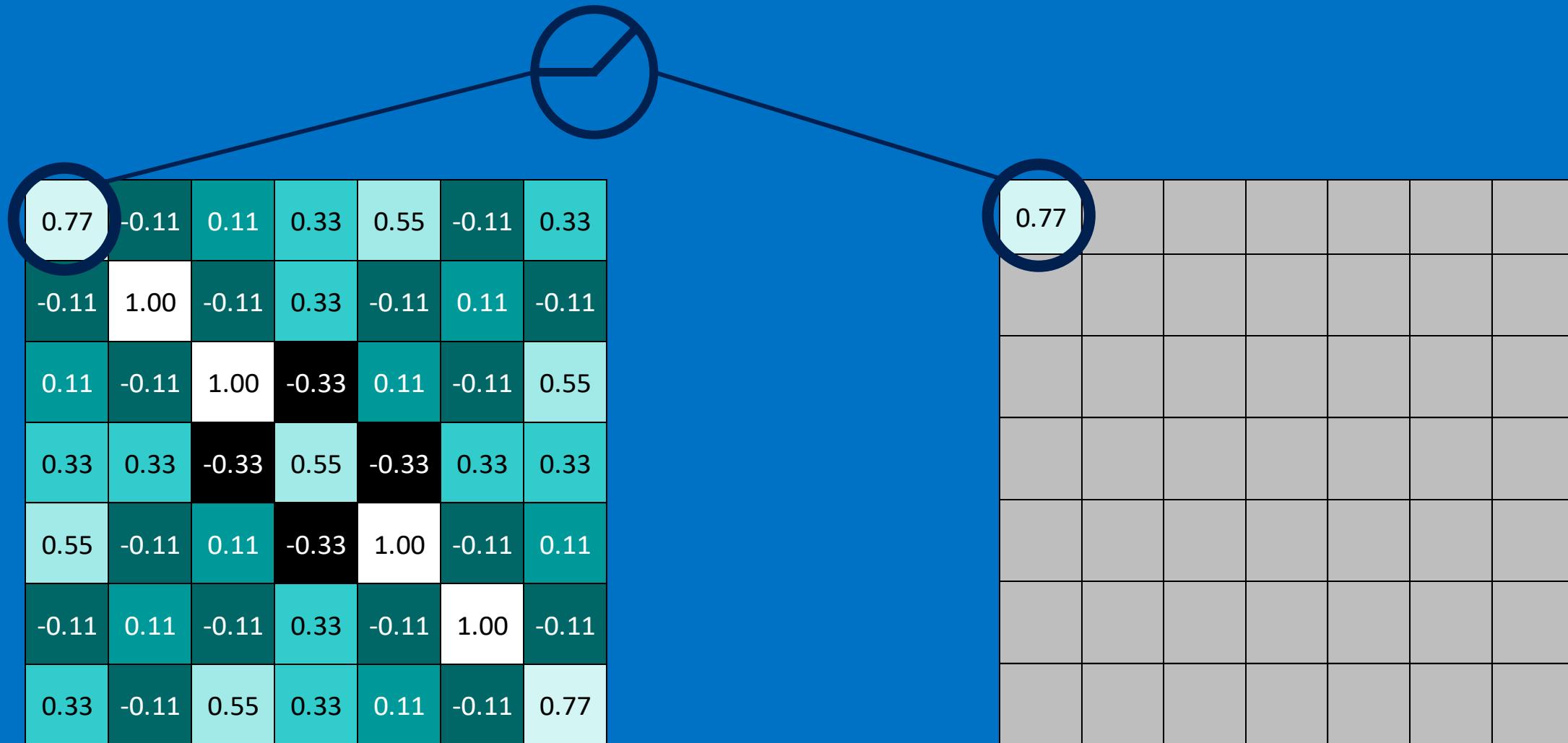


| | | | |
|------|------|------|------|
| 1.00 | 0.33 | 0.55 | 0.33 |
| 0.33 | 1.00 | 0.33 | 0.55 |
| 0.55 | 0.33 | 1.00 | 0.11 |
| 0.33 | 0.55 | 0.11 | 0.77 |

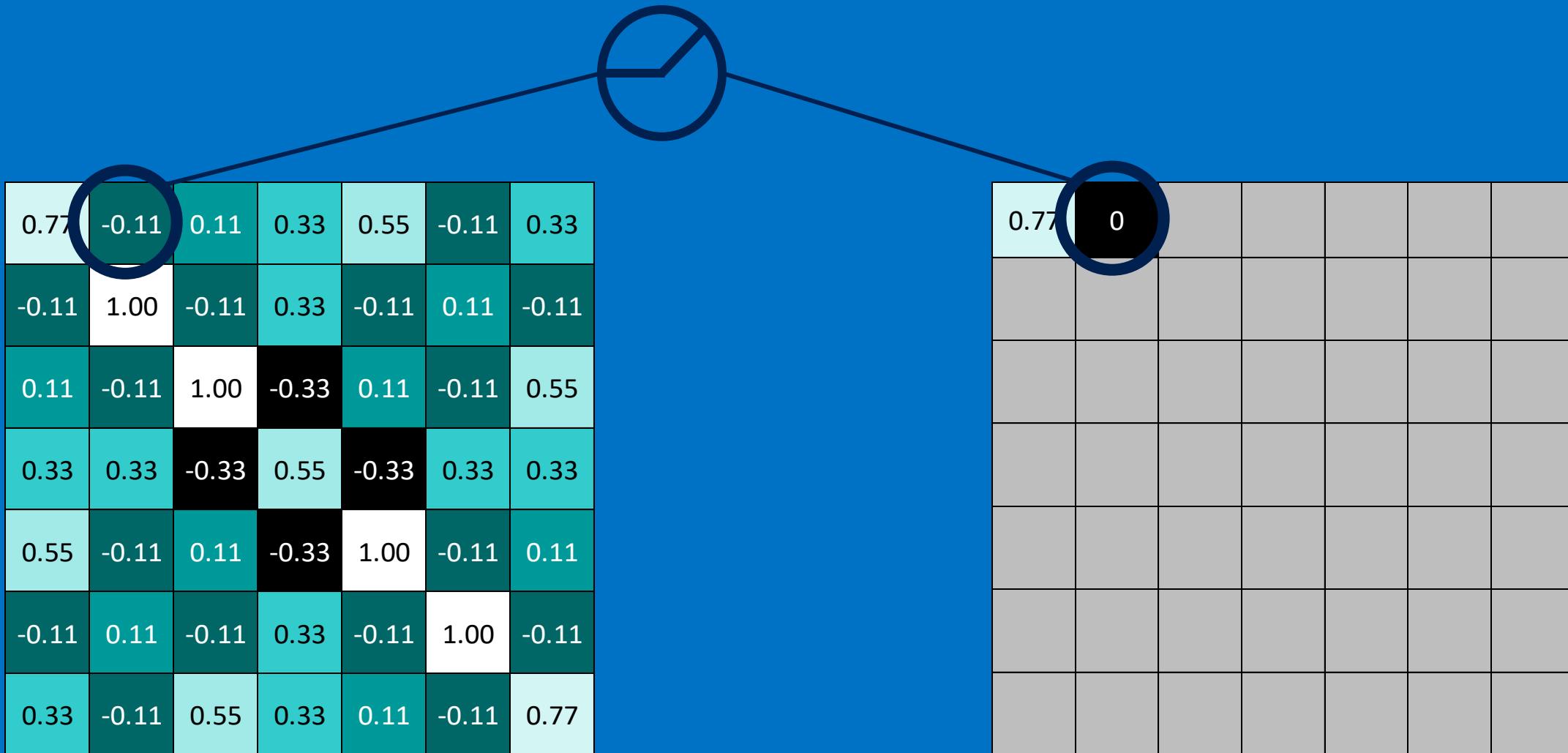
| | | | |
|------|------|------|------|
| 0.55 | 0.33 | 0.55 | 0.33 |
| 0.33 | 1.00 | 0.55 | 0.11 |
| 0.55 | 0.55 | 0.55 | 0.11 |
| 0.33 | 0.11 | 0.11 | 0.33 |

| | | | |
|------|------|------|------|
| 0.33 | 0.55 | 1.00 | 0.77 |
| 0.55 | 0.55 | 1.00 | 0.33 |
| 1.00 | 1.00 | 0.11 | 0.55 |
| 0.77 | 0.33 | 0.55 | 0.33 |

Rectified Linear Units (ReLUs)



Rectified Linear Units (ReLUs)



Rectified Linear Units (ReLUs)

| | | | | | | |
|-------|-------|-------|-------|-------|-------|-------|
| 0.77 | -0.11 | 0.11 | 0.33 | 0.55 | -0.11 | 0.33 |
| -0.11 | 1.00 | -0.11 | 0.33 | -0.11 | 0.11 | -0.11 |
| 0.11 | -0.11 | 1.00 | -0.33 | 0.11 | -0.11 | 0.55 |
| 0.33 | 0.33 | -0.33 | 0.55 | -0.33 | 0.33 | 0.33 |
| 0.55 | -0.11 | 0.11 | -0.33 | 1.00 | -0.11 | 0.11 |
| -0.11 | 0.11 | -0.11 | 0.33 | -0.11 | 1.00 | -0.11 |
| 0.33 | -0.11 | 0.55 | 0.33 | 0.11 | -0.11 | 0.77 |



| | | | | | | |
|------|---|------|------|------|---|------|
| 0.77 | 0 | 0.11 | 0.33 | 0.55 | 0 | 0.33 |
| | | | | | | |
| | | | | | | |
| | | | | | | |
| | | | | | | |

Rectified Linear Units (ReLUs)

| | | | | | | |
|-------|-------|-------|-------|-------|-------|-------|
| 0.77 | -0.11 | 0.11 | 0.33 | 0.55 | -0.11 | 0.33 |
| -0.11 | 1.00 | -0.11 | 0.33 | -0.11 | 0.11 | -0.11 |
| 0.11 | -0.11 | 1.00 | -0.33 | 0.11 | -0.11 | 0.55 |
| 0.33 | 0.33 | -0.33 | 0.55 | -0.33 | 0.33 | 0.33 |
| 0.55 | -0.11 | 0.11 | -0.33 | 1.00 | -0.11 | 0.11 |
| -0.11 | 0.11 | -0.11 | 0.33 | -0.11 | 1.00 | -0.11 |
| 0.33 | -0.11 | 0.55 | 0.33 | 0.11 | -0.11 | 0.77 |



| | | | | | | |
|------|------|------|------|------|------|------|
| 0.77 | 0 | 0.11 | 0.33 | 0.55 | 0 | 0.33 |
| 0 | 1.00 | 0 | 0.33 | 0 | 0.11 | 0 |
| 0.11 | 0 | 1.00 | 0 | 0.11 | 0 | 0.55 |
| 0.33 | 0.33 | 0 | 0.55 | 0 | 0.33 | 0.33 |
| 0.55 | 0 | 0.11 | 0 | 1.00 | 0 | 0.11 |
| 0 | 0.11 | 0 | 0.33 | 0 | 1.00 | 0 |
| 0.33 | 0 | 0.55 | 0.33 | 0.11 | 0 | 0.77 |

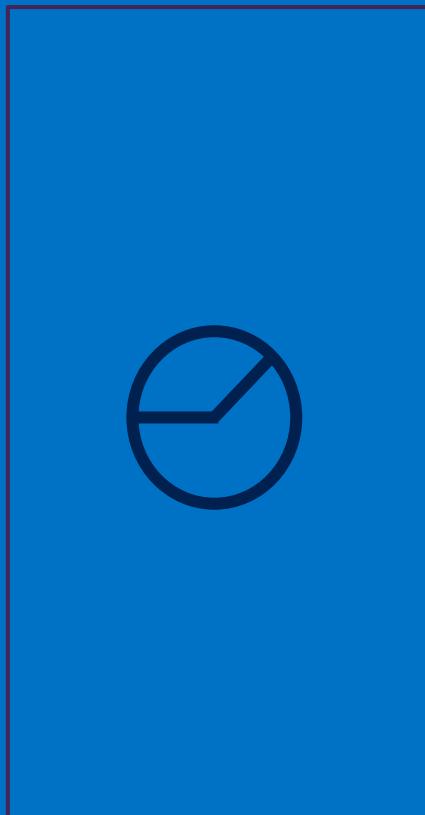
ReLU layer

A stack of images becomes a stack of images with no negative values.

| | | | | | | |
|-------|-------|-------|-------|-------|-------|-------|
| 0.77 | -0.11 | 0.11 | 0.33 | 0.55 | -0.11 | 0.33 |
| -0.11 | 1.00 | -0.11 | 0.33 | -0.11 | 0.11 | -0.11 |
| 0.11 | -0.11 | 1.00 | -0.33 | 0.11 | -0.11 | 0.55 |
| 0.33 | 0.33 | -0.33 | 0.55 | -0.33 | 0.33 | 0.33 |
| 0.55 | -0.11 | 0.11 | -0.33 | 1.00 | -0.11 | 0.11 |
| -0.11 | 0.11 | -0.11 | 0.33 | -0.11 | 1.00 | -0.11 |
| 0.33 | -0.11 | 0.55 | 0.33 | 0.11 | -0.11 | 0.77 |

| | | | | | | |
|-------|-------|-------|-------|-------|-------|-------|
| 0.33 | -0.55 | 0.11 | -0.11 | 0.11 | -0.55 | 0.33 |
| -0.55 | 0.55 | -0.55 | 0.33 | -0.55 | 0.55 | -0.55 |
| 0.11 | -0.55 | 0.55 | -0.77 | 0.55 | -0.55 | 0.11 |
| -0.11 | 0.33 | -0.77 | 1.00 | -0.77 | 0.33 | -0.11 |
| 0.11 | -0.55 | 0.55 | -0.77 | 0.55 | -0.55 | 0.11 |
| -0.55 | 0.55 | -0.55 | 0.33 | -0.55 | 0.55 | 0.55 |
| 0.33 | -0.55 | 0.11 | -0.11 | 0.11 | -0.55 | 0.33 |

| | | | | | | |
|-------|-------|-------|-------|-------|-------|-------|
| 0.33 | -0.11 | 0.55 | 0.33 | 0.11 | -0.11 | 0.77 |
| -0.11 | 0.11 | -0.11 | 0.33 | -0.11 | 1.00 | -0.11 |
| 0.55 | -0.11 | 0.11 | -0.33 | 1.00 | -0.11 | 0.11 |
| 0.33 | 0.33 | -0.33 | 0.55 | -0.33 | 0.33 | 0.33 |
| 0.11 | -0.11 | 1.00 | -0.33 | 0.11 | -0.11 | 0.55 |
| -0.11 | 1.00 | -0.11 | 0.33 | -0.11 | 0.11 | -0.11 |
| 0.77 | -0.11 | 0.11 | 0.33 | 0.55 | -0.11 | 0.33 |



| | | | | | | |
|------|------|------|------|------|------|------|
| 0.77 | 0 | 0.11 | 0.33 | 0.55 | 0 | 0.33 |
| 0 | 1.00 | 0 | 0.33 | 0 | 0.11 | 0 |
| 0.11 | 0 | 1.00 | 0 | 0.11 | 0 | 0.55 |
| 0.33 | 0.33 | 0 | 0.55 | 0 | 0.33 | 0.33 |
| 0.55 | 0 | 0.11 | 0 | 1.00 | 0 | 0.11 |
| 0 | 0.11 | 0 | 0.33 | 0 | 1.00 | 0 |
| 0.33 | 0 | 0.55 | 0.33 | 0.11 | 0 | 0.77 |

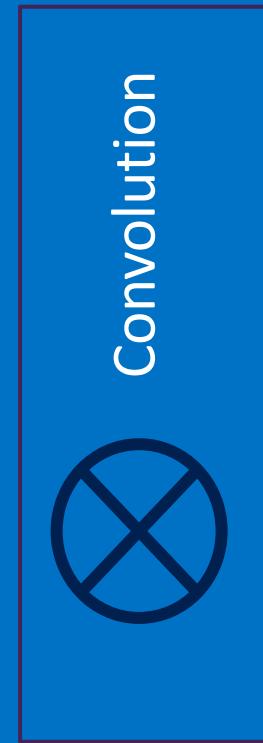
| | | | | | | |
|------|------|------|------|------|------|------|
| 0.33 | 0 | 0.11 | 0 | 0.11 | 0 | 0.33 |
| 0 | 0.55 | 0 | 0.33 | 0 | 0.55 | 0 |
| 0.11 | 0 | 0.55 | 0 | 0.55 | 0 | 0.11 |
| 0 | 0.33 | 0 | 1.00 | 0 | 0.33 | 0 |
| 0.11 | 0 | 0.55 | 0 | 0.55 | 0 | 0.11 |
| 0 | 0.55 | 0 | 0.33 | 0 | 0.55 | 0 |
| 0.33 | 0 | 0.11 | 0 | 0.11 | 0 | 0.33 |

| | | | | | | |
|------|------|------|------|------|------|------|
| 0.33 | 0 | 0.55 | 0.33 | 0.11 | 0 | 0.77 |
| 0 | 0.11 | 0 | 0.33 | 0 | 1.00 | 0 |
| 0.55 | 0 | 0.11 | 0 | 1.00 | 0 | 0.11 |
| 0.33 | 0.33 | 0 | 0.55 | 0 | 0.33 | 0.33 |
| 0.11 | 0 | 1.00 | 0 | 0.11 | 0 | 0.55 |
| 0 | 1.00 | 0 | 0.33 | 0 | 0.11 | 0 |
| 0.77 | 0 | 0.11 | 0.33 | 0.55 | 0 | 0.33 |

Layers get stacked

The output of one becomes the input of the next.

| | | | | | | | | | |
|----|----|----|----|----|----|----|----|----|----|
| -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 |
| -1 | 1 | -1 | -1 | -1 | -1 | -1 | 1 | -1 | -1 |
| -1 | -1 | 1 | -1 | -1 | -1 | 1 | -1 | -1 | -1 |
| -1 | -1 | -1 | 1 | -1 | 1 | -1 | -1 | -1 | -1 |
| -1 | -1 | -1 | -1 | 1 | -1 | -1 | -1 | -1 | -1 |
| -1 | -1 | -1 | -1 | 1 | -1 | 1 | -1 | -1 | -1 |
| -1 | -1 | 1 | -1 | -1 | -1 | 1 | -1 | -1 | -1 |
| -1 | 1 | -1 | -1 | -1 | -1 | -1 | 1 | -1 | -1 |
| -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 |



| | | | |
|------|------|------|------|
| 1.00 | 0.33 | 0.55 | 0.33 |
| 0.33 | 1.00 | 0.33 | 0.55 |
| 0.55 | 0.33 | 1.00 | 0.11 |
| 0.33 | 0.55 | 0.11 | 0.77 |

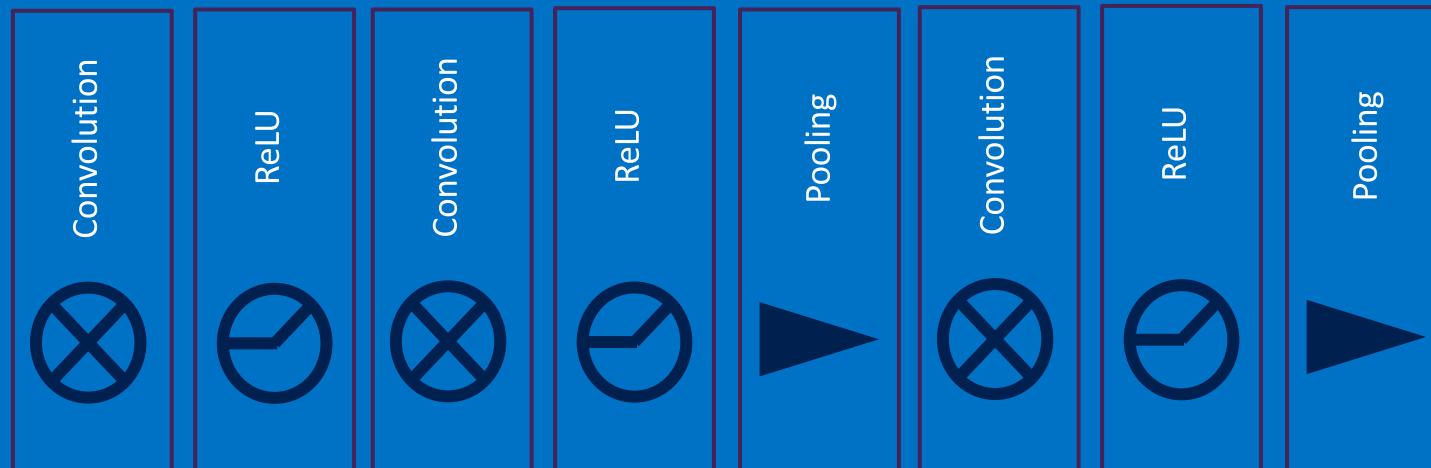
| | | | |
|------|------|------|------|
| 0.55 | 0.33 | 0.55 | 0.33 |
| 0.33 | 1.00 | 0.55 | 0.11 |
| 0.55 | 0.55 | 0.55 | 0.11 |
| 0.33 | 0.11 | 0.11 | 0.33 |

| | | | |
|------|------|------|------|
| 0.33 | 0.55 | 1.00 | 0.77 |
| 0.55 | 0.55 | 1.00 | 0.33 |
| 1.00 | 1.00 | 0.11 | 0.55 |
| 0.77 | 0.33 | 0.55 | 0.33 |

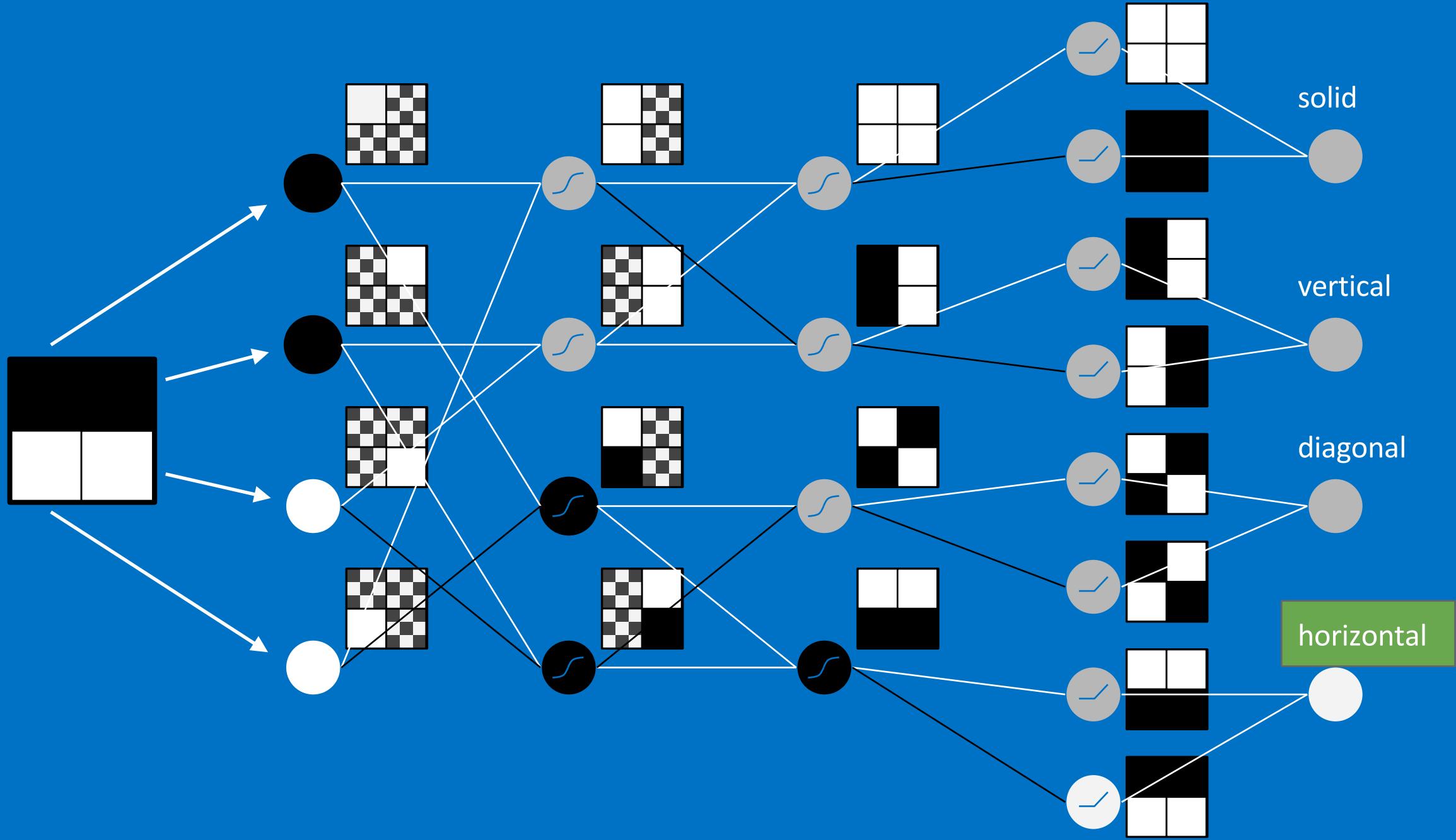
Deep stacking

Layers can be repeated several (or many) times.

| | | | | | | | | | |
|----|----|----|----|----|----|----|----|----|----|
| -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 |
| -1 | 1 | -1 | -1 | -1 | -1 | -1 | 1 | -1 | -1 |
| -1 | -1 | 1 | -1 | -1 | -1 | 1 | -1 | -1 | -1 |
| -1 | -1 | -1 | 1 | -1 | 1 | -1 | -1 | -1 | -1 |
| -1 | -1 | -1 | -1 | 1 | -1 | -1 | -1 | -1 | -1 |
| -1 | -1 | -1 | -1 | 1 | -1 | -1 | -1 | -1 | -1 |
| -1 | -1 | -1 | 1 | -1 | 1 | -1 | -1 | -1 | -1 |
| -1 | -1 | 1 | -1 | -1 | -1 | 1 | -1 | -1 | -1 |
| -1 | 1 | -1 | -1 | -1 | -1 | -1 | 1 | -1 | -1 |
| -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 |

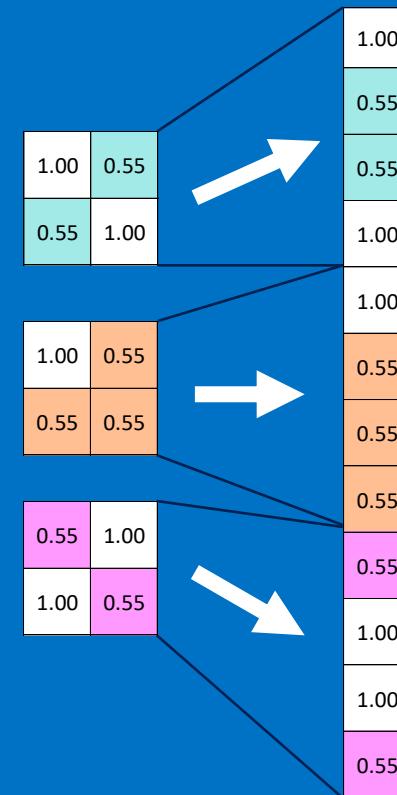


| | |
|------|------|
| 1.00 | 0.55 |
| 0.55 | 1.00 |
| 1.00 | 0.55 |
| 0.55 | 0.55 |
| 0.55 | 1.00 |
| 1.00 | 0.55 |



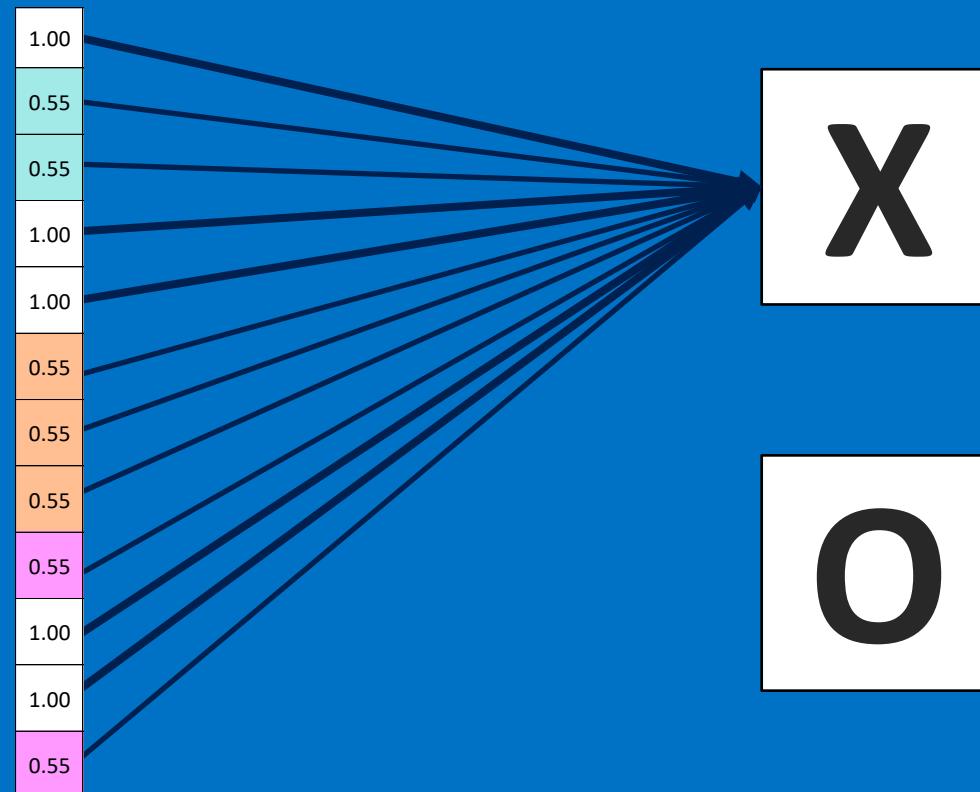
Fully connected layer

Every value gets a vote



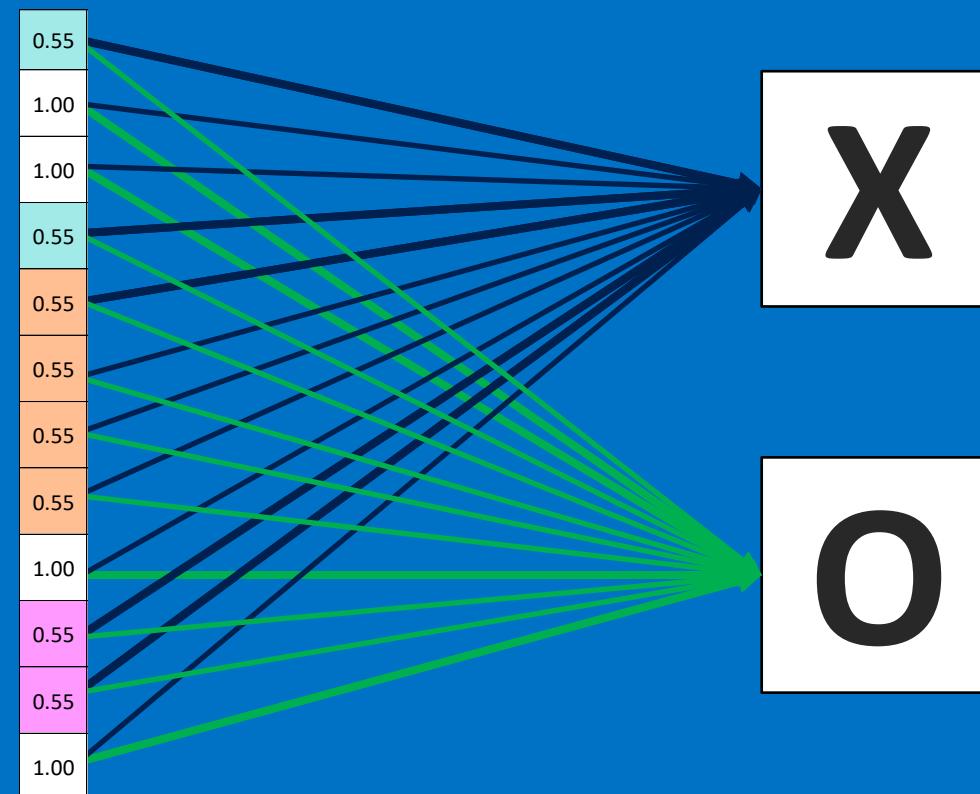
Fully connected layer

Vote depends on how strongly a value predicts X or O



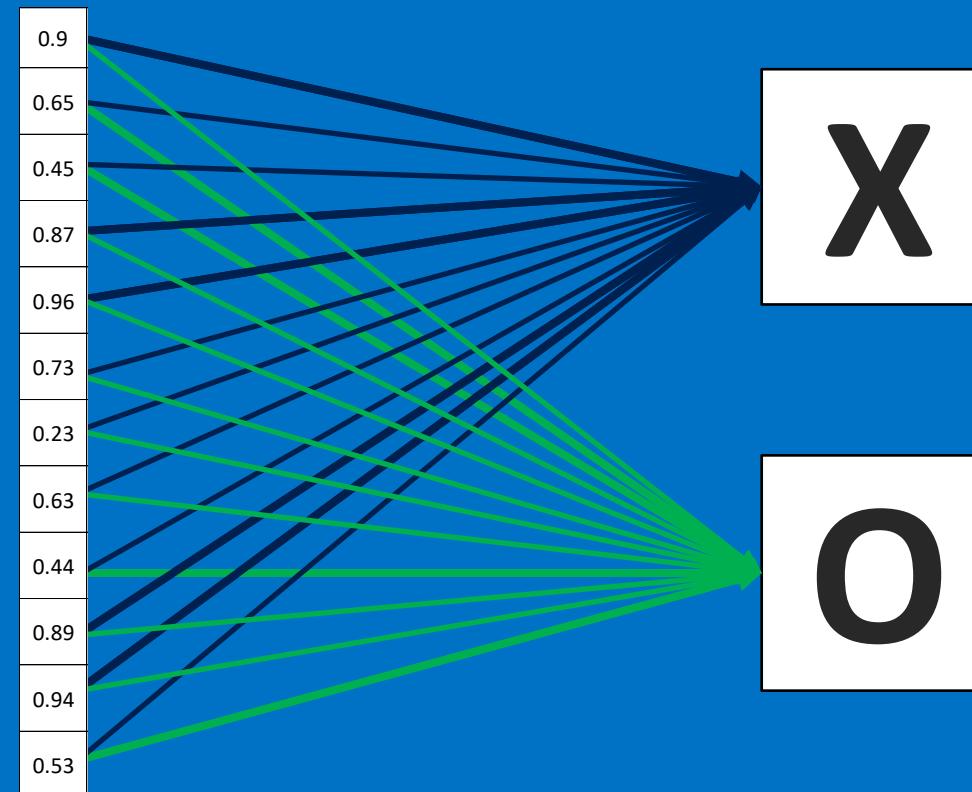
Fully connected layer

Vote depends on how strongly a value predicts X or O



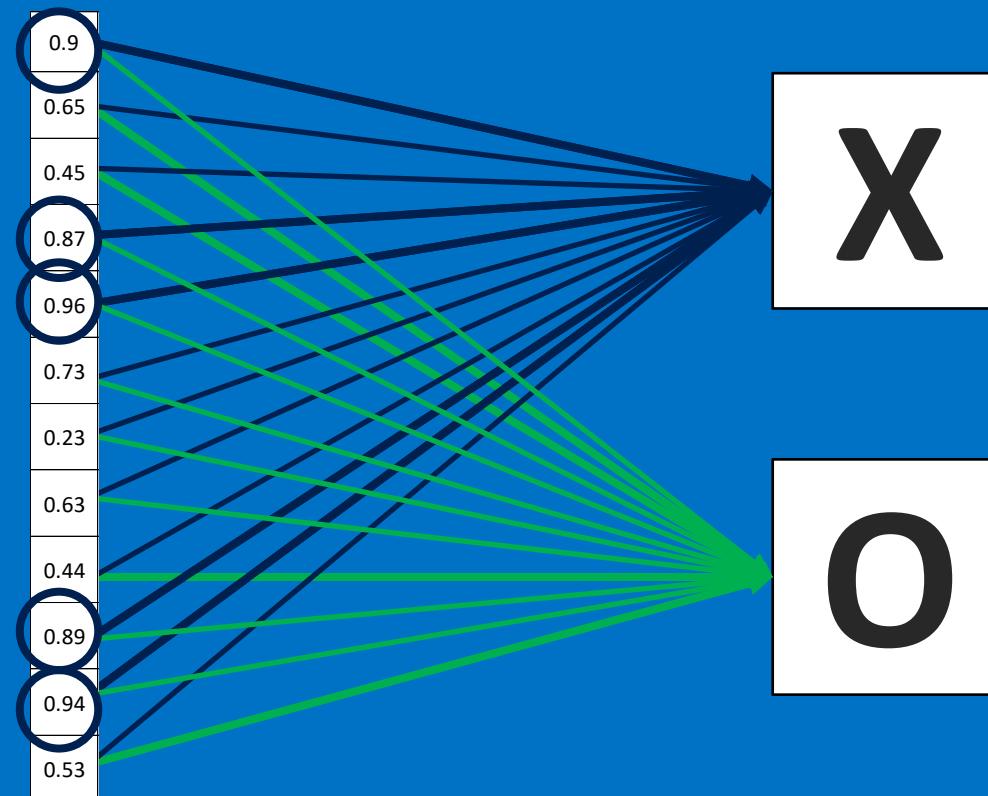
Fully connected layer

Future values vote on X or O



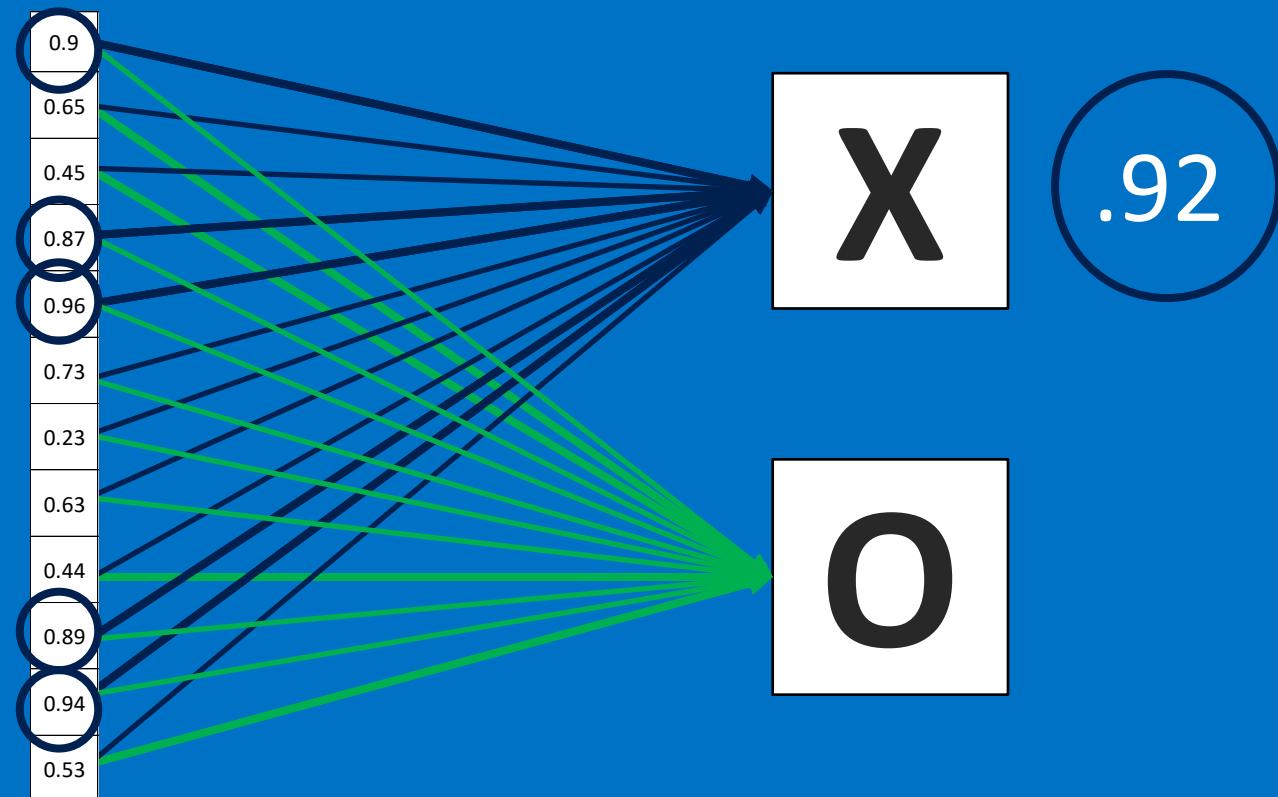
Fully connected layer

Future values vote on X or O



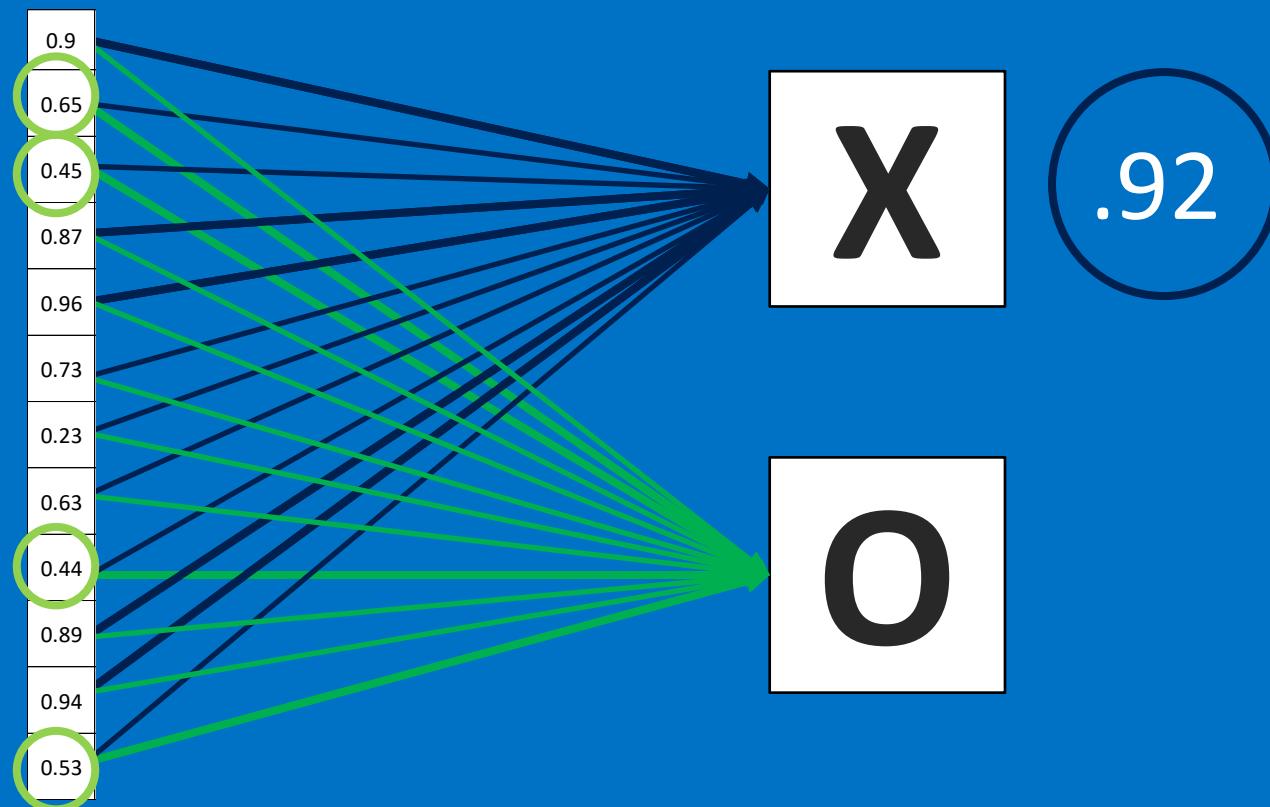
Fully connected layer

Future values vote on X or O



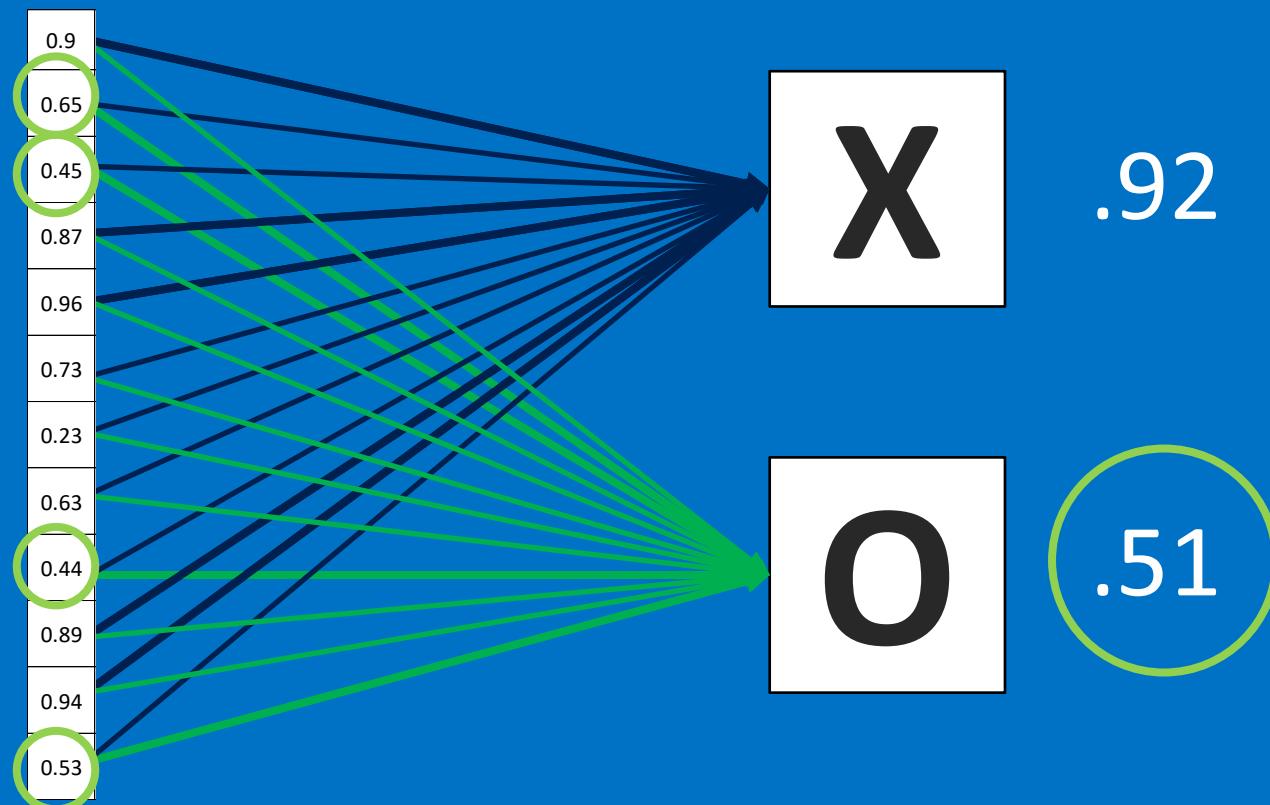
Fully connected layer

Future values vote on X or O



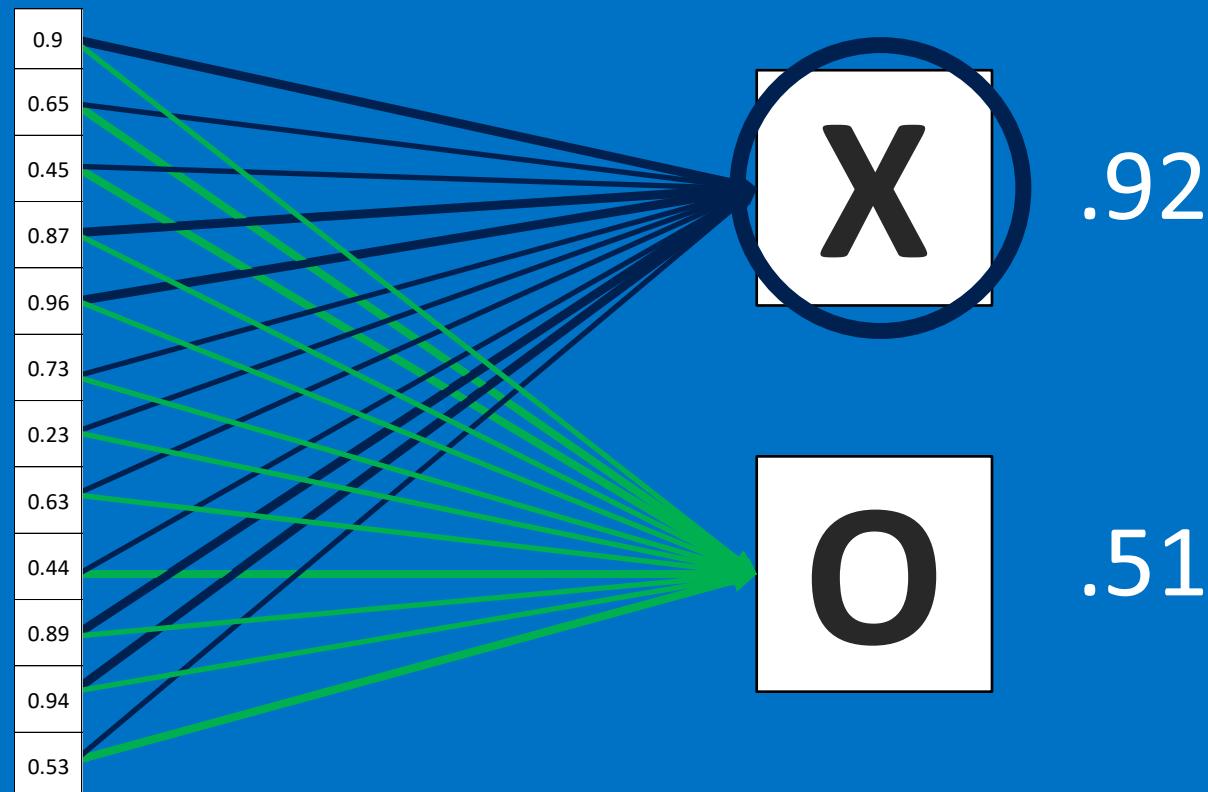
Fully connected layer

Future values vote on X or O



Fully connected layer

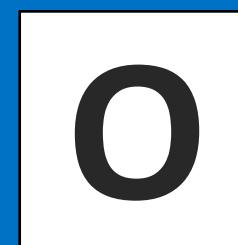
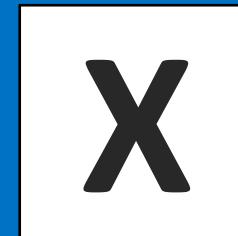
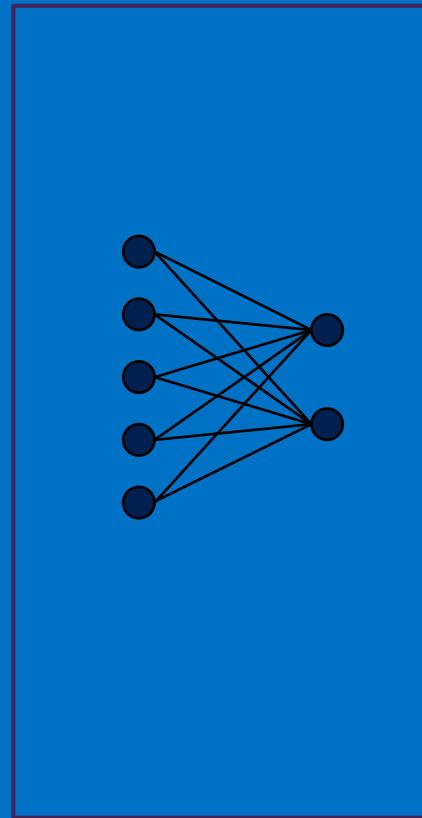
Future values vote on X or O



Fully connected layer

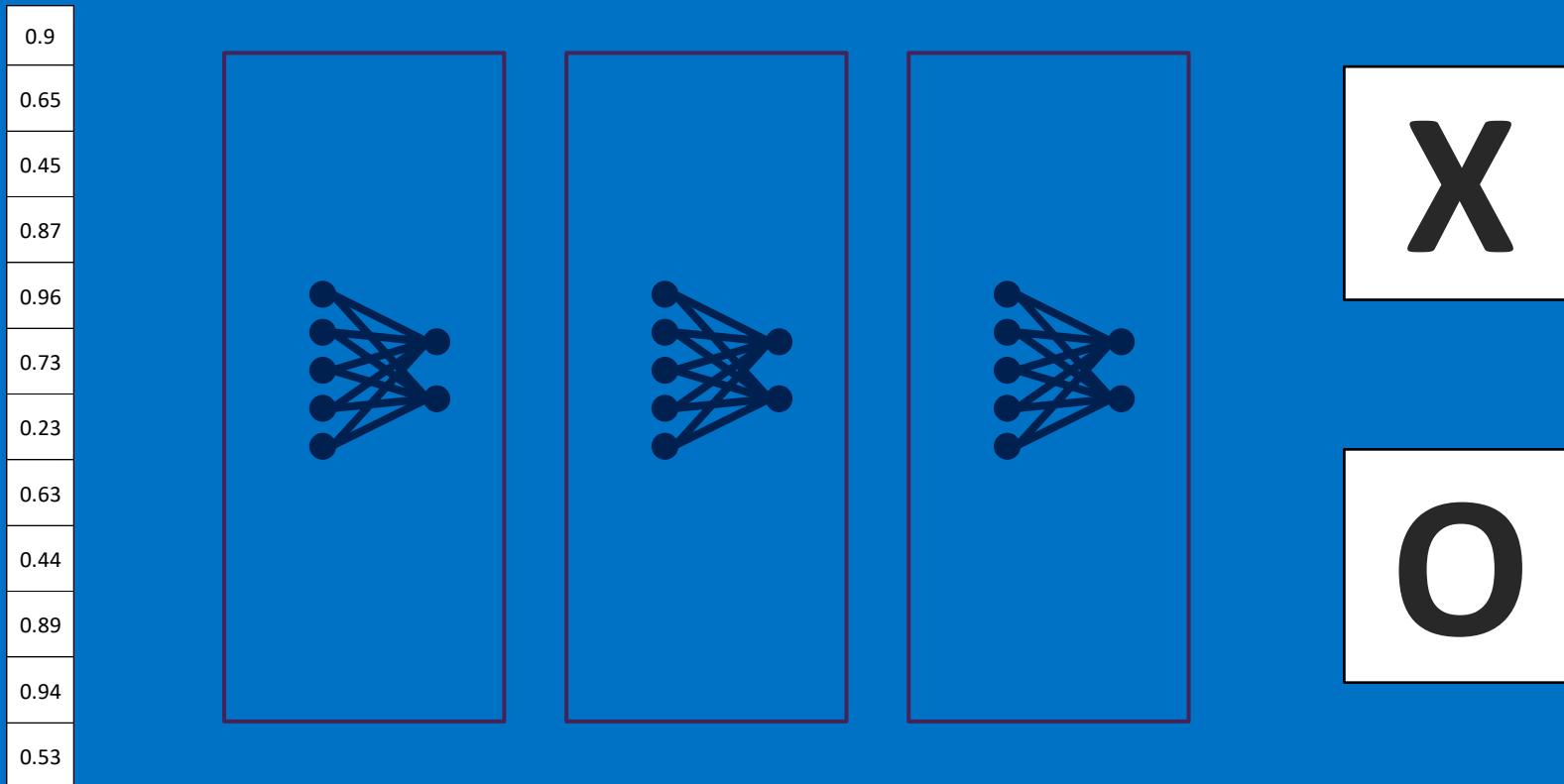
A list of feature values becomes a list of votes.

| |
|------|
| 0.9 |
| 0.65 |
| 0.45 |
| 0.87 |
| 0.96 |
| 0.73 |
| 0.23 |
| 0.63 |
| 0.44 |
| 0.89 |
| 0.94 |
| 0.53 |

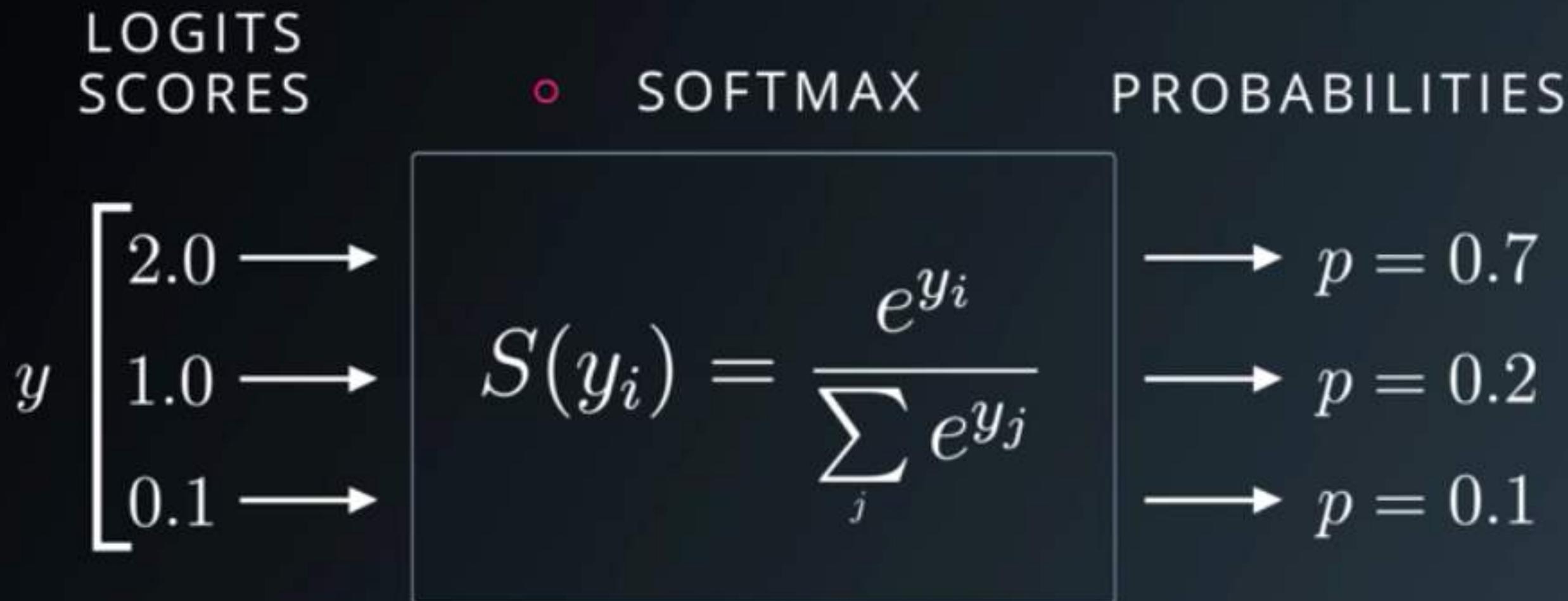


Fully connected layer

These can also be stacked.



logits layer is popularly used for the last neuron layer of neural network for classification task which produces raw prediction values as real numbers ranging from [-infinity, +infinity]



Softmax function turns logits [2.0, 1.0, 0.1] into probabilities* which sum to one

LOGITS
SCORES

◦ SOFTMAX

PROBABILITIES

y $\begin{bmatrix} 2.0 \rightarrow \\ 1.0 \rightarrow \\ 0.1 \rightarrow \end{bmatrix}$

$$S(y_i) = \frac{e^{y_i}}{\sum_j e^{y_j}}$$

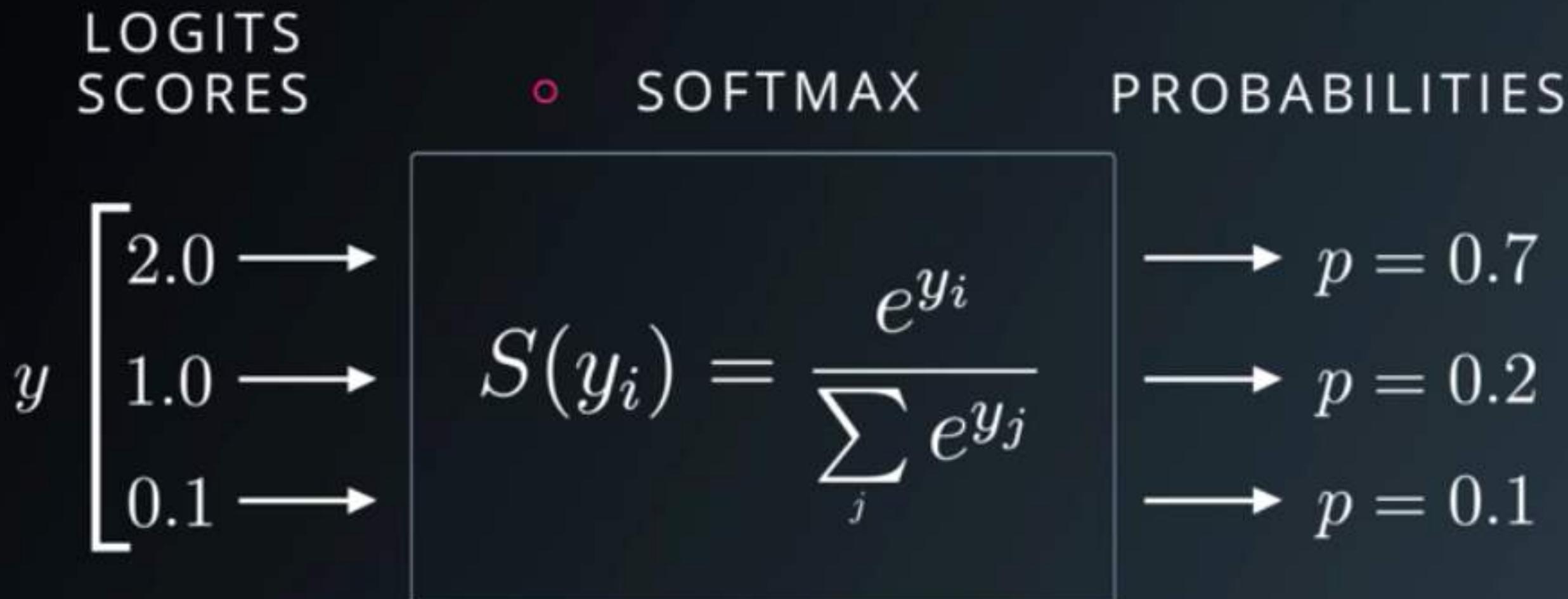
$$\rightarrow p = 0.7$$

$$\rightarrow p = 0.2$$

$$\rightarrow p = 0.1$$

Why not just divide each logits by the sum of logits?

Why do we need exponents?

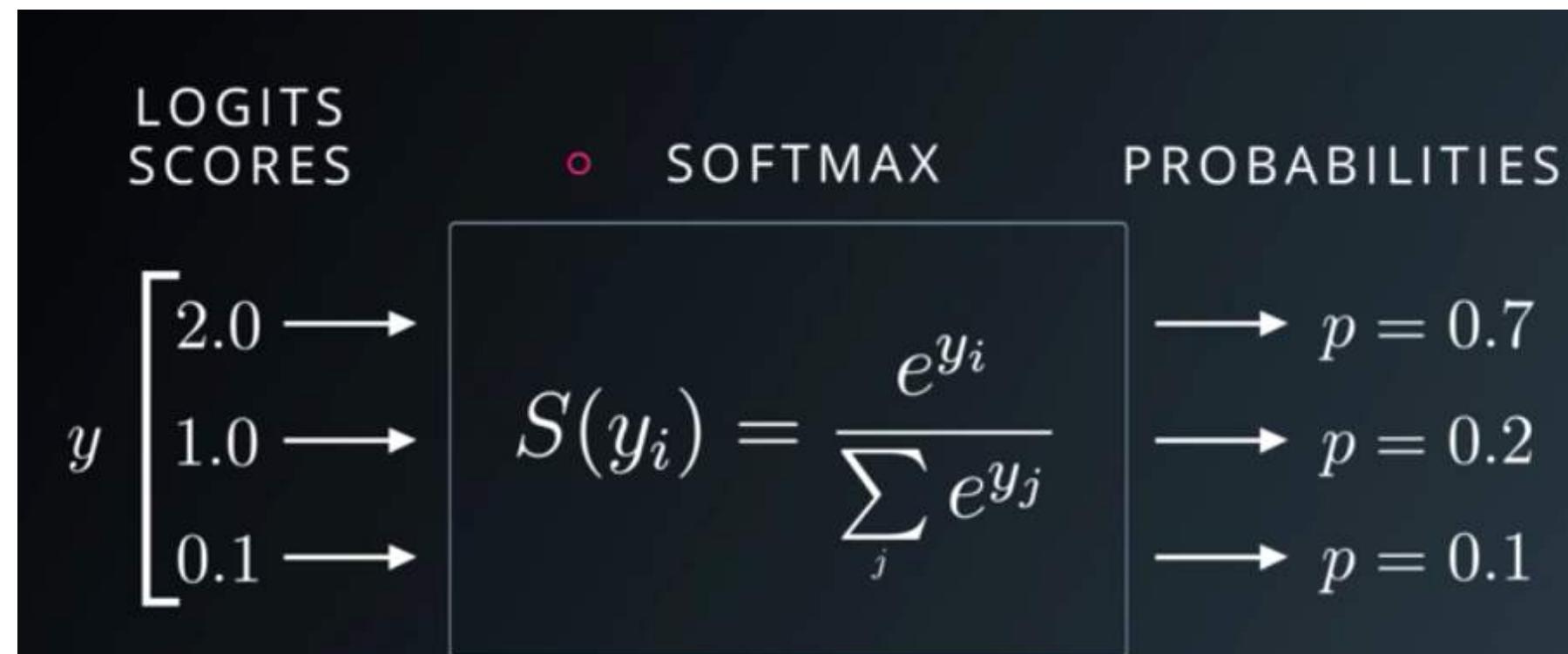


Why not just divide each logits by the sum of logits?

When logits are negative, adding it together does not give us the correct normalization

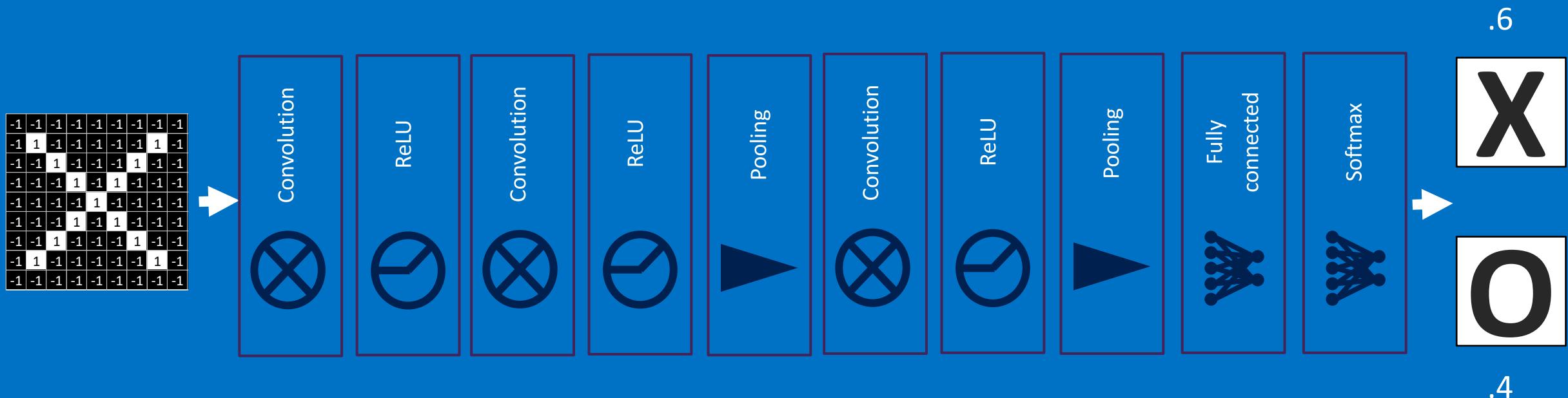
Why do we need exponents?

*Easy summation and derivative calculation. $\log(a*b) = \log(a)+\log(b)$*



Putting it all together

A set of pixels becomes a set of votes.



Learning

Q: Where do all the magic numbers come from?

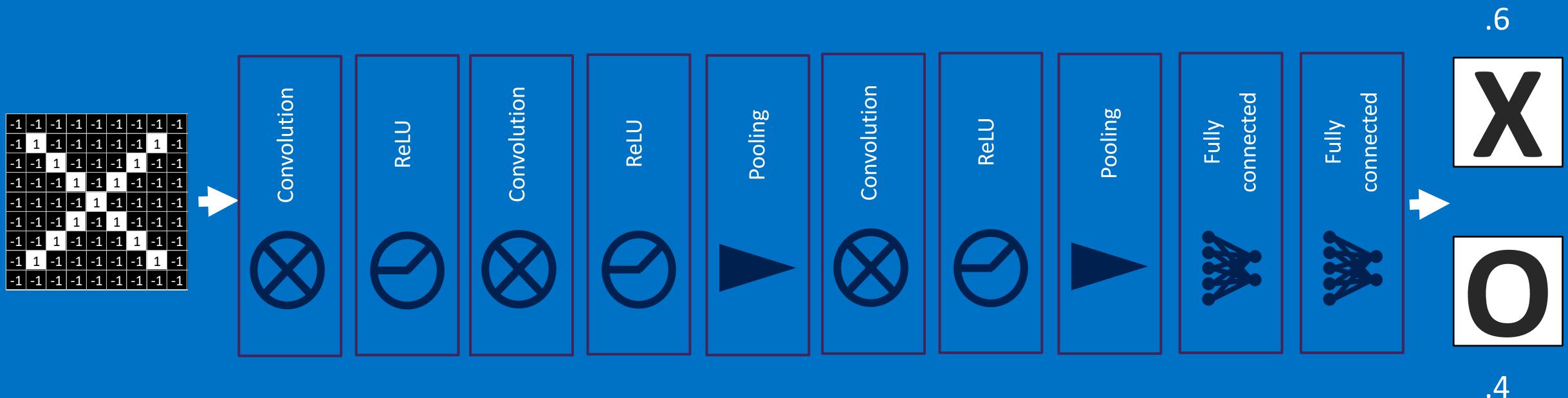
Features in convolutional layers

Voting weights in fully connected layers

A: Backpropagation

Backprop

Error = right answer – actual answer



Case Study: AlexNet

[Krizhevsky et al. 2012]

Full (simplified) AlexNet architecture:

[227x227x3] INPUT

[55x55x96] CONV1: 96 11x11 filters at stride 4, pad 0

[27x27x96] MAX POOL1: 3x3 filters at stride 2

[27x27x96] NORM1: Normalization layer

[27x27x256] CONV2: 256 5x5 filters at stride 1, pad 2

[13x13x256] MAX POOL2: 3x3 filters at stride 2

[13x13x256] NORM2: Normalization layer

[13x13x384] CONV3: 384 3x3 filters at stride 1, pad 1

[13x13x384] CONV4: 384 3x3 filters at stride 1, pad 1

[13x13x256] CONV5: 256 3x3 filters at stride 1, pad 1

[6x6x256] MAX POOL3: 3x3 filters at stride 2

[4096] FC6: 4096 neurons

[4096] FC7: 4096 neurons

[1000] FC8: 1000 neurons (class scores)

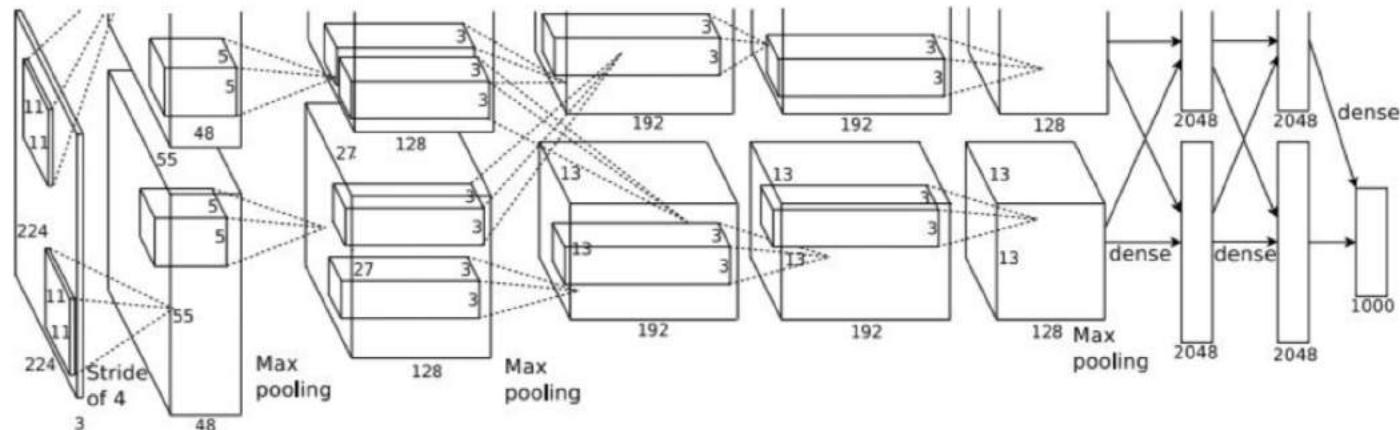


Figure copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.

Hyperparameters (knobs)

Convolution

Number of features

Size of features

Pooling

Window size

Window stride

Fully Connected

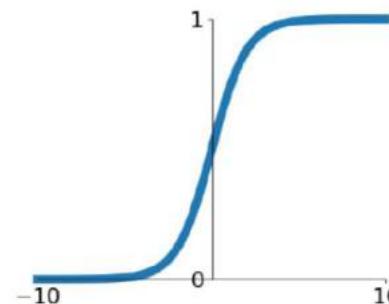
Number of neurons

Activation Functions

Activation Functions

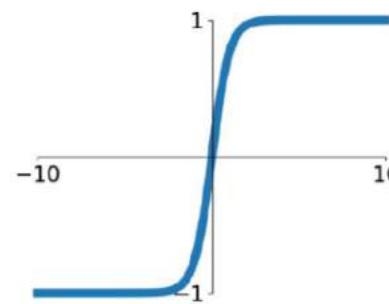
Sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



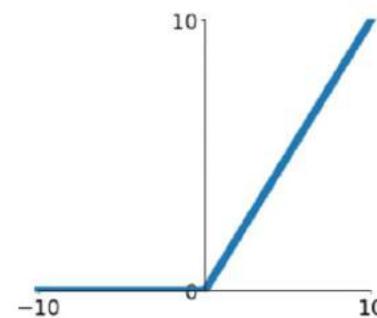
tanh

$$\tanh(x)$$



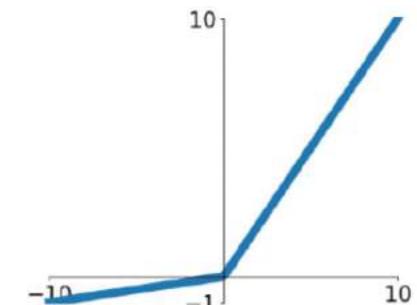
ReLU

$$\max(0, x)$$



Leaky ReLU

$$\max(0.1x, x)$$

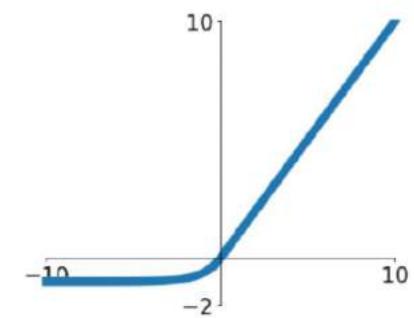


Maxout

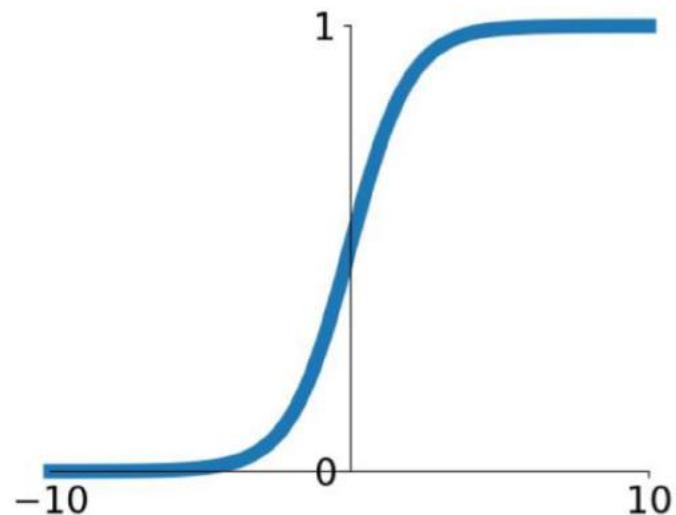
$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



Activation Functions



Sigmoid

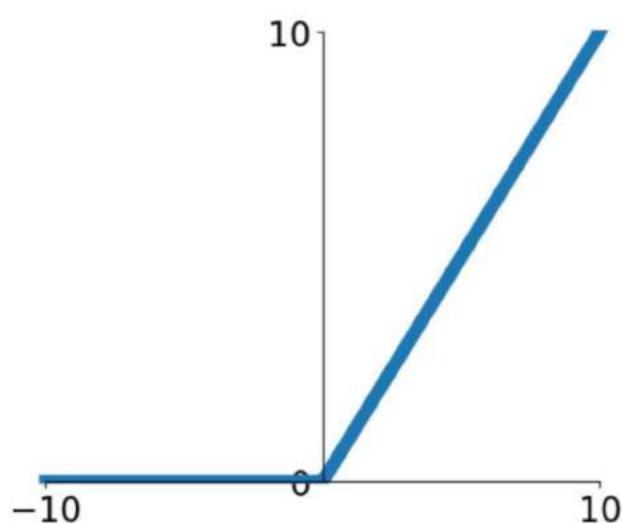
$$\sigma(x) = 1/(1 + e^{-x})$$

- Squashes numbers to range [0,1]
- Historically popular since they have nice interpretation as a saturating “firing rate” of a neuron

3 problems:

1. Saturated neurons “kill” the gradients
2. Sigmoid outputs are not zero-centered
3. $\exp()$ is a bit compute expensive

Activation Functions



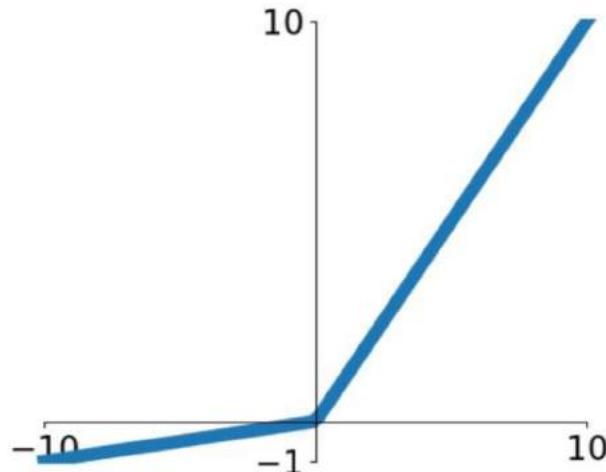
- Computes $f(x) = \max(0, x)$
- Does not saturate (in +region)
- Very computationally efficient
- Converges much faster than sigmoid/tanh in practice (e.g. 6x)
- Actually more biologically plausible than sigmoid

ReLU
(Rectified Linear Unit)

[Krizhevsky et al., 2012]

Activation Functions

[Mass et al., 2013]
[He et al., 2015]



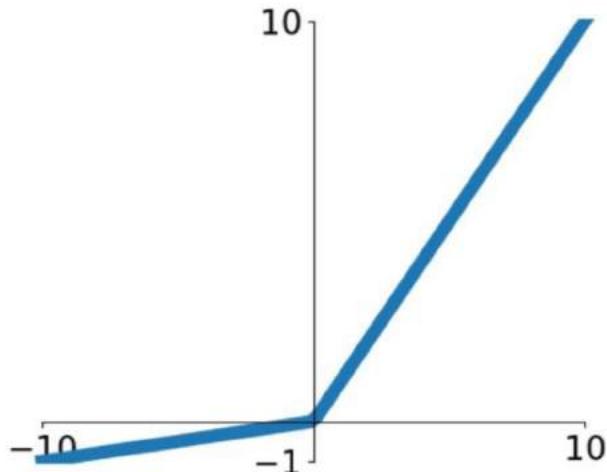
- Does not saturate
- Computationally efficient
- Converges much faster than sigmoid/tanh in practice! (e.g. 6x)
- **will not “die”.**

Leaky ReLU

$$f(x) = \max(0.01x, x)$$

Activation Functions

[Mass et al., 2013]
[He et al., 2015]



Leaky ReLU

$$f(x) = \max(0.01x, x)$$

- Does not saturate
- Computationally efficient
- Converges much faster than sigmoid/tanh in practice! (e.g. 6x)
- **will not “die”.**

Parametric Rectifier (PReLU)

$$f(x) = \max(\alpha x, x)$$

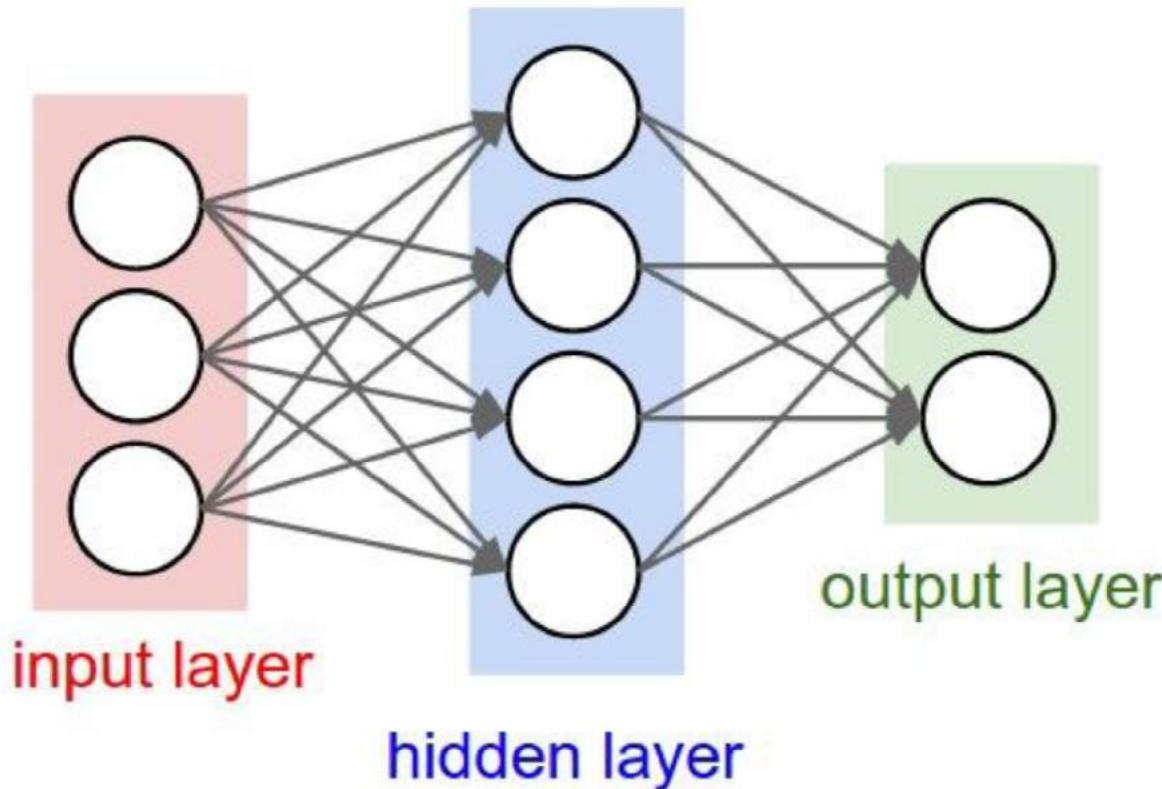
backprop into $\backslash\alpha$
(parameter)

TLDR: In practice:

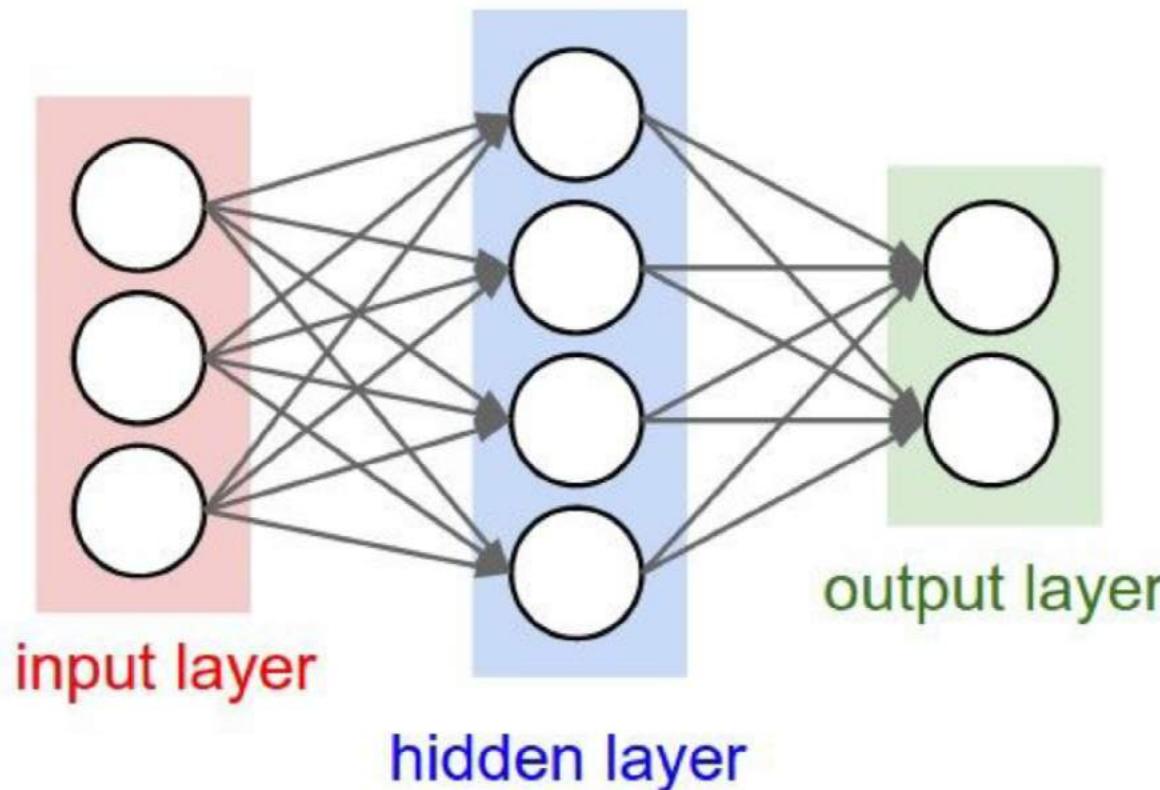
- Use ReLU. Be careful with your learning rates
- Try out Leaky ReLU / Maxout / ELU
- Try out tanh but don't expect much
- Don't use sigmoid

Weight Initialization

- Q: what happens when $W=0$ init is used?



- Q: what happens when $W=0$ init is used?



Having zero (or equal) weights to start with will prevent the network from learning. The errors backpropagated through the network is proportional to the value of the weights. If all the weights are the same, then the backpropagated errors will be the same, and consequently, all of the weights will be updated by the same amount.

- First idea: **Small random numbers**
(gaussian with zero mean and 1e-2 standard deviation)

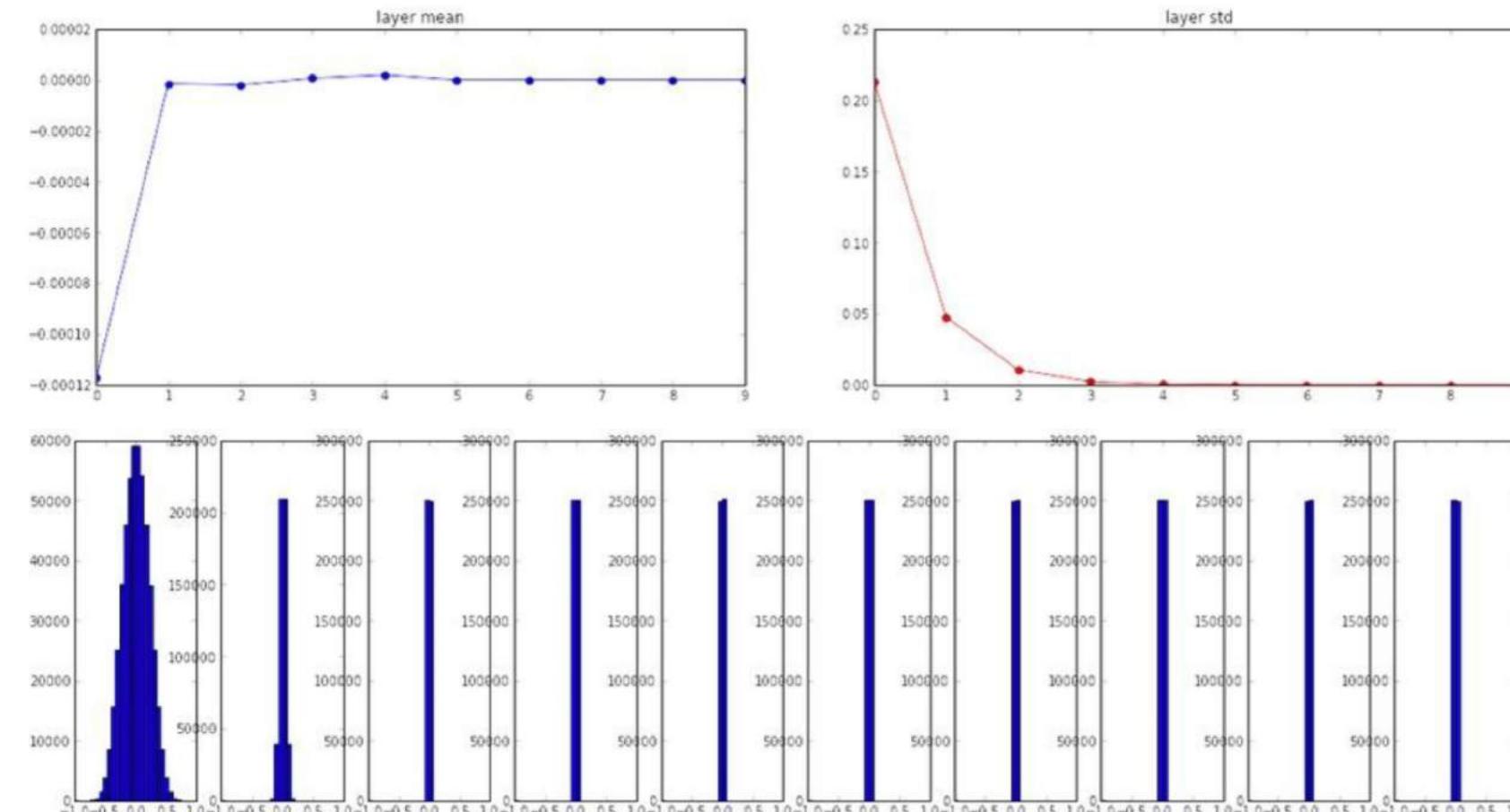
```
W = 0.01* np.random.randn(D,H)
```

- First idea: **Small random numbers**
(gaussian with zero mean and 1e-2 standard deviation)

```
W = 0.01* np.random.randn(D,H)
```

Works ~okay for small networks, but problems with deeper networks.

```
input layer had mean 0.000927 and std 0.998388  
hidden layer 1 had mean -0.000117 and std 0.213081  
hidden layer 2 had mean -0.000001 and std 0.047551  
hidden layer 3 had mean -0.000002 and std 0.010630  
hidden layer 4 had mean 0.000001 and std 0.002378  
hidden layer 5 had mean 0.000002 and std 0.000532  
hidden layer 6 had mean -0.000000 and std 0.000119  
hidden layer 7 had mean 0.000000 and std 0.000026  
hidden layer 8 had mean -0.000000 and std 0.000006  
hidden layer 9 had mean 0.000000 and std 0.000001  
hidden layer 10 had mean -0.000000 and std 0.000000
```



All activations
become zero!

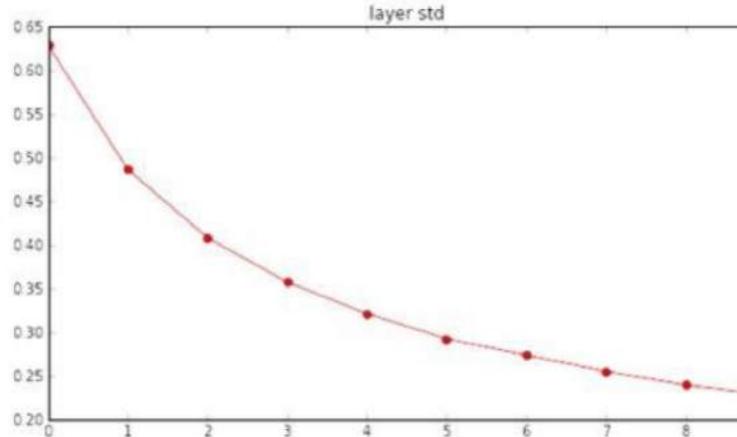
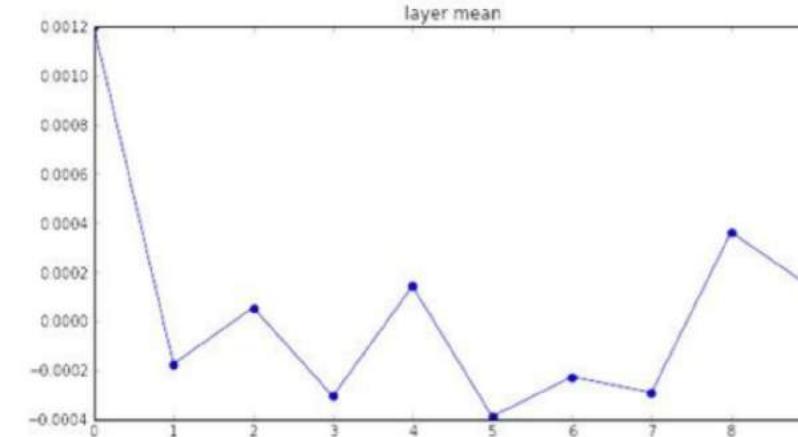
Q: think about the
backward pass.
What do the
gradients look like?

Hint: think about backward
pass for a W^*X gate.

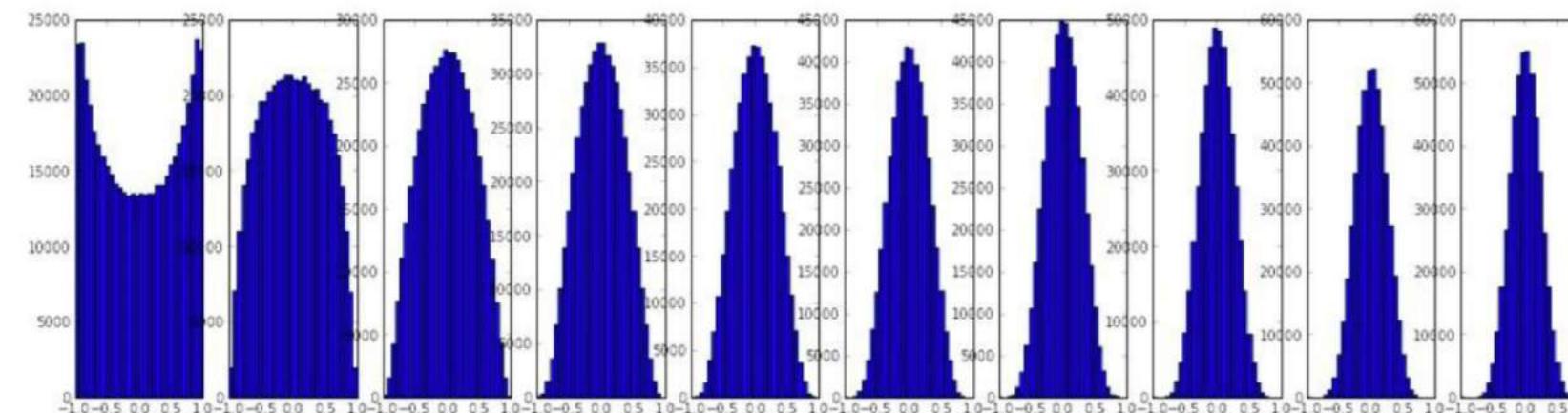
```
input layer had mean 0.001800 and std 1.001311  
hidden layer 1 had mean 0.001198 and std 0.627953  
hidden layer 2 had mean -0.000175 and std 0.486051  
hidden layer 3 had mean 0.000055 and std 0.407723  
hidden layer 4 had mean -0.000306 and std 0.357108  
hidden layer 5 had mean 0.000142 and std 0.320917  
hidden layer 6 had mean -0.000389 and std 0.292116  
hidden layer 7 had mean -0.000228 and std 0.273387  
hidden layer 8 had mean -0.000291 and std 0.254935  
hidden layer 9 had mean 0.000361 and std 0.239266  
hidden layer 10 had mean 0.000139 and std 0.228008
```

```
W = np.random.randn(fan_in, fan_out) / np.sqrt(fan_in) # layer initialization
```

“Xavier initialization”
[Glorot et al., 2010]



Reasonable initialization.
(Mathematical derivation
assumes linear activations)



Proper initialization is an active area of research...

Understanding the difficulty of training deep feedforward neural networks

by Glorot and Bengio, 2010

Exact solutions to the nonlinear dynamics of learning in deep linear neural networks by

Saxe et al, 2013

Random walk initialization for training very deep feedforward networks by Sussillo and

Abbott, 2014

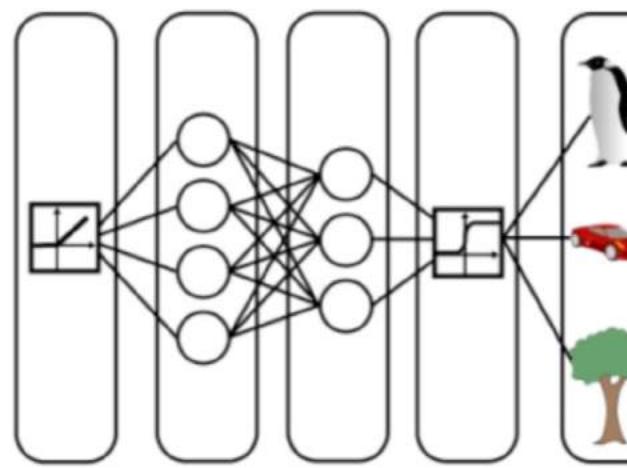
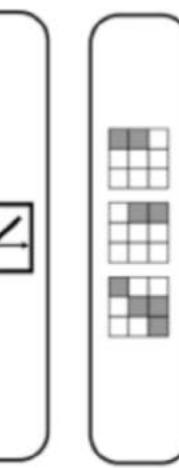
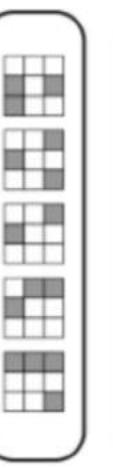
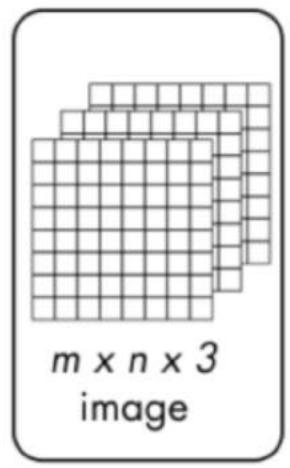
Delving deep into rectifiers: Surpassing human-level performance on ImageNet

classification by He et al., 2015

Data-dependent Initializations of Convolutional Neural Networks by Krähenbühl et al., 2015

All you need is a good init, Mishkin and Matas, 2015

Adjusting the weights



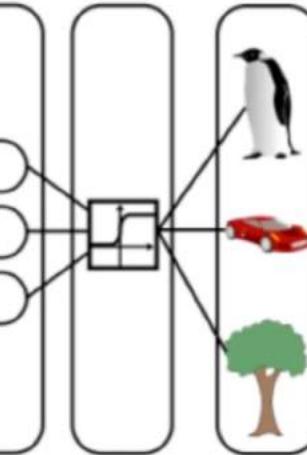
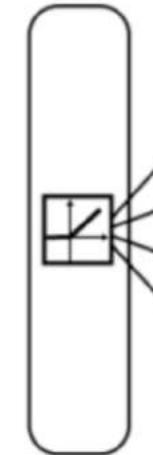
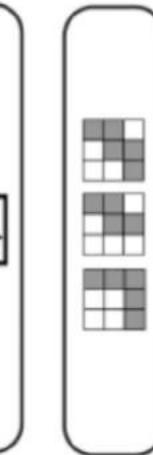
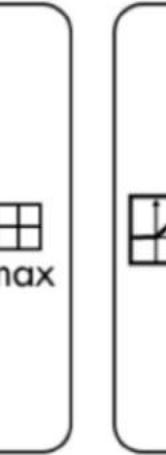
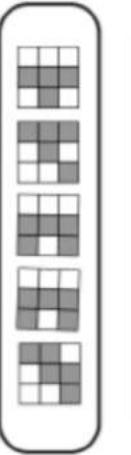
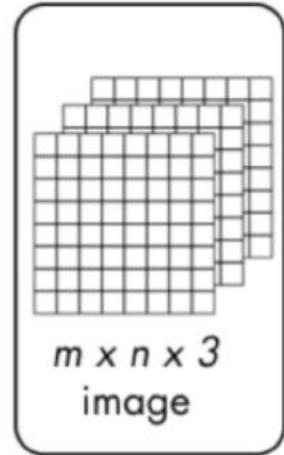
| Prediction | Reality |
|------------|-----------|
| penguin | ✓ penguin |
| car | ✓ car |
| penguin | ✓ penguin |
| tree | ✓ tree |
| tree | ✓ tree |
| penguin | ✗ car |
| car | ✗ tree |
| car | ✗ penguin |



Training

Validation

Testing



Prediction

car



Reality

car

penguin



penguin

tree



tree



Validation
loss



car

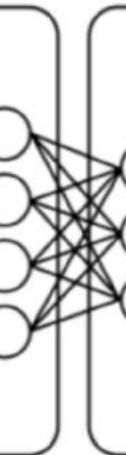
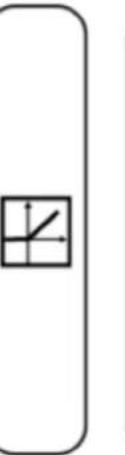
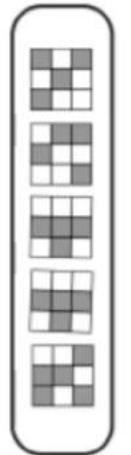
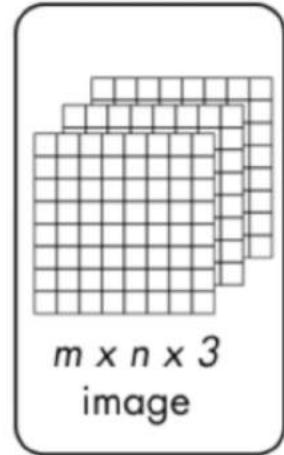
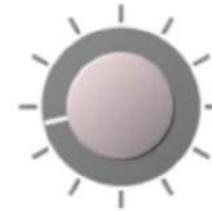
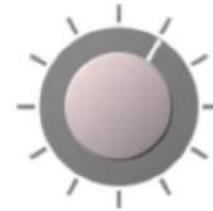


car

baseball



car



Prediction

car



Reality

car

penguin



penguin

tree



tree



Validation
loss



car

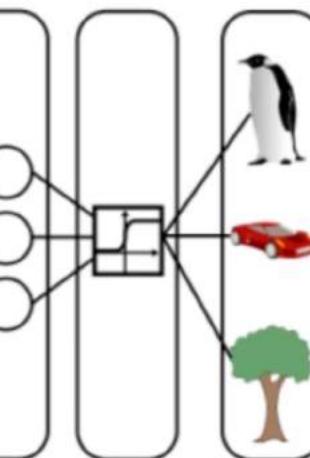
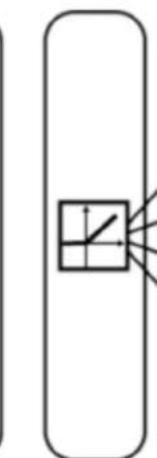
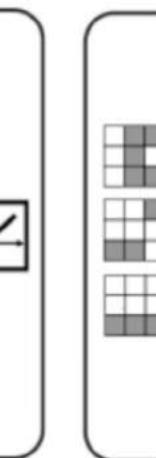
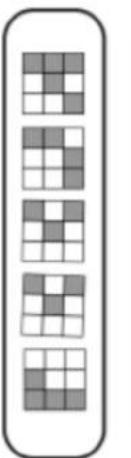
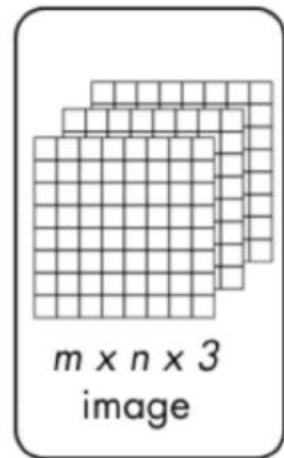
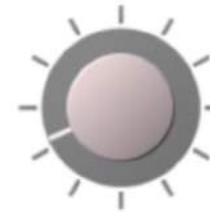
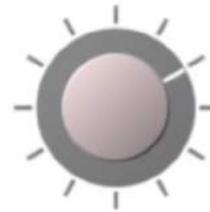


car

baseball



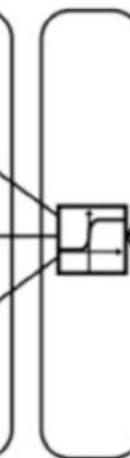
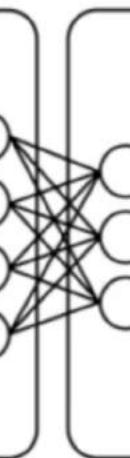
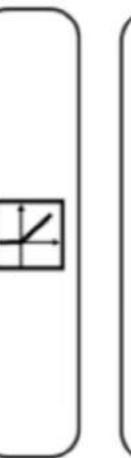
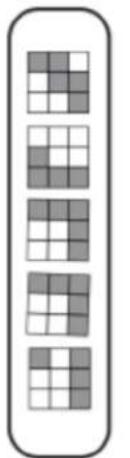
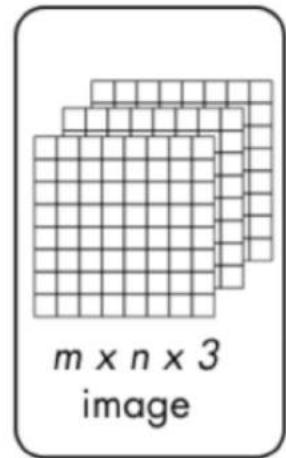
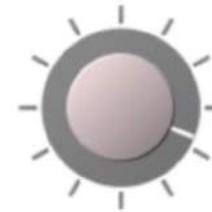
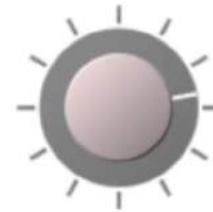
car



Validation
loss



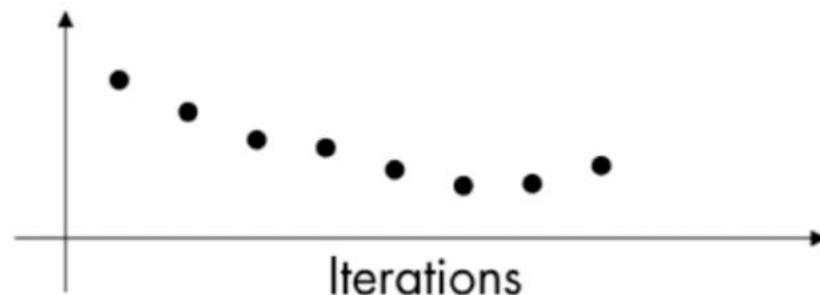
| | | |
|---------|---|-----|
| car | ✓ | car |
| penguin | ✗ | car |



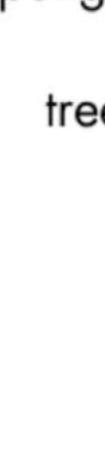
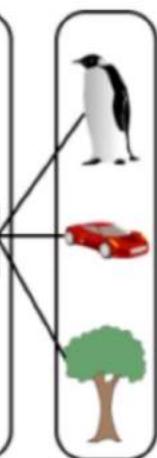
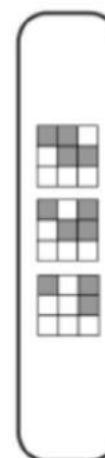
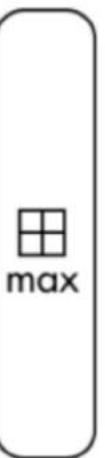
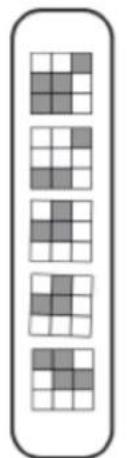
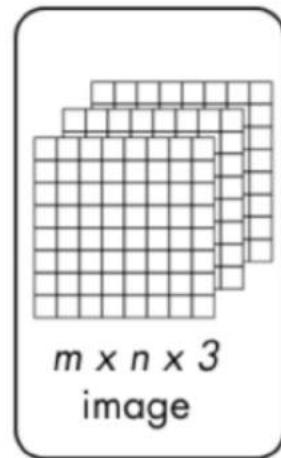
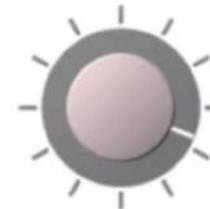
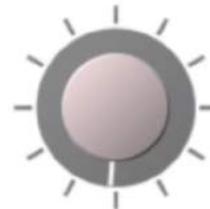
| | Prediction | Reality |
|---------|------------|---------|
| car | ✓ | car |
| penguin | ✓ | penguin |
| tree | ✓ | tree |



Validation
loss



| | | |
|-----|---|-----|
| car | ✓ | car |
| car | ✓ | car |



Prediction

car



car

penguin



penguin

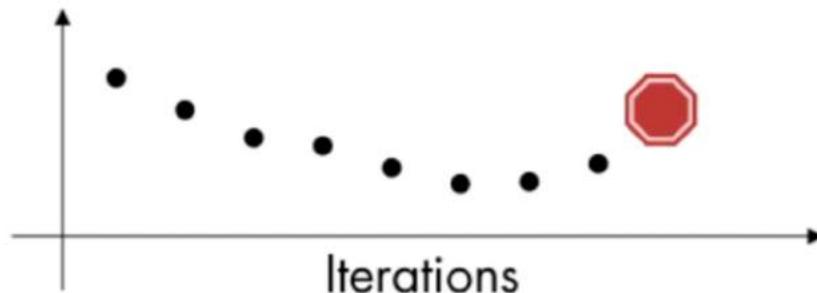
tree



tree



Validation
loss



car



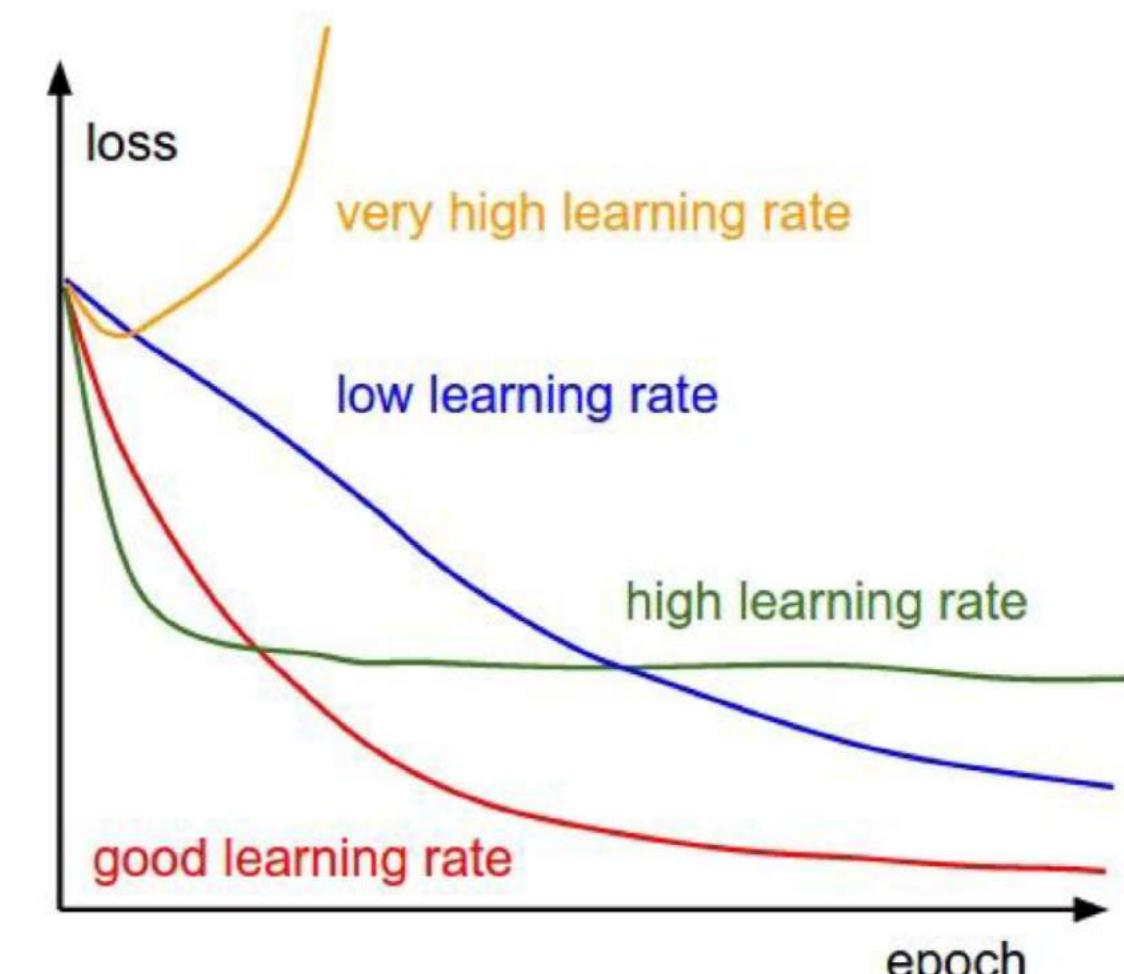
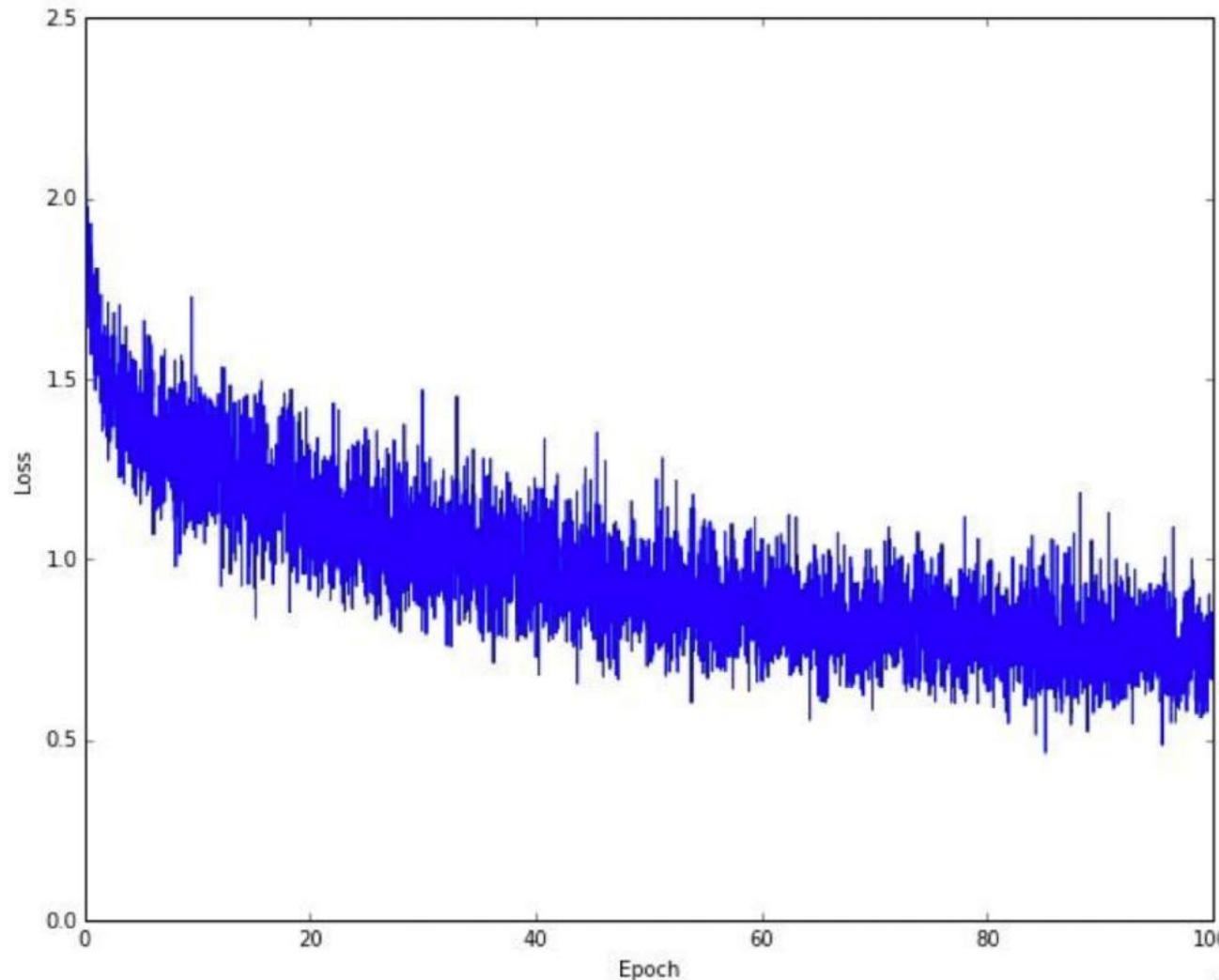
car

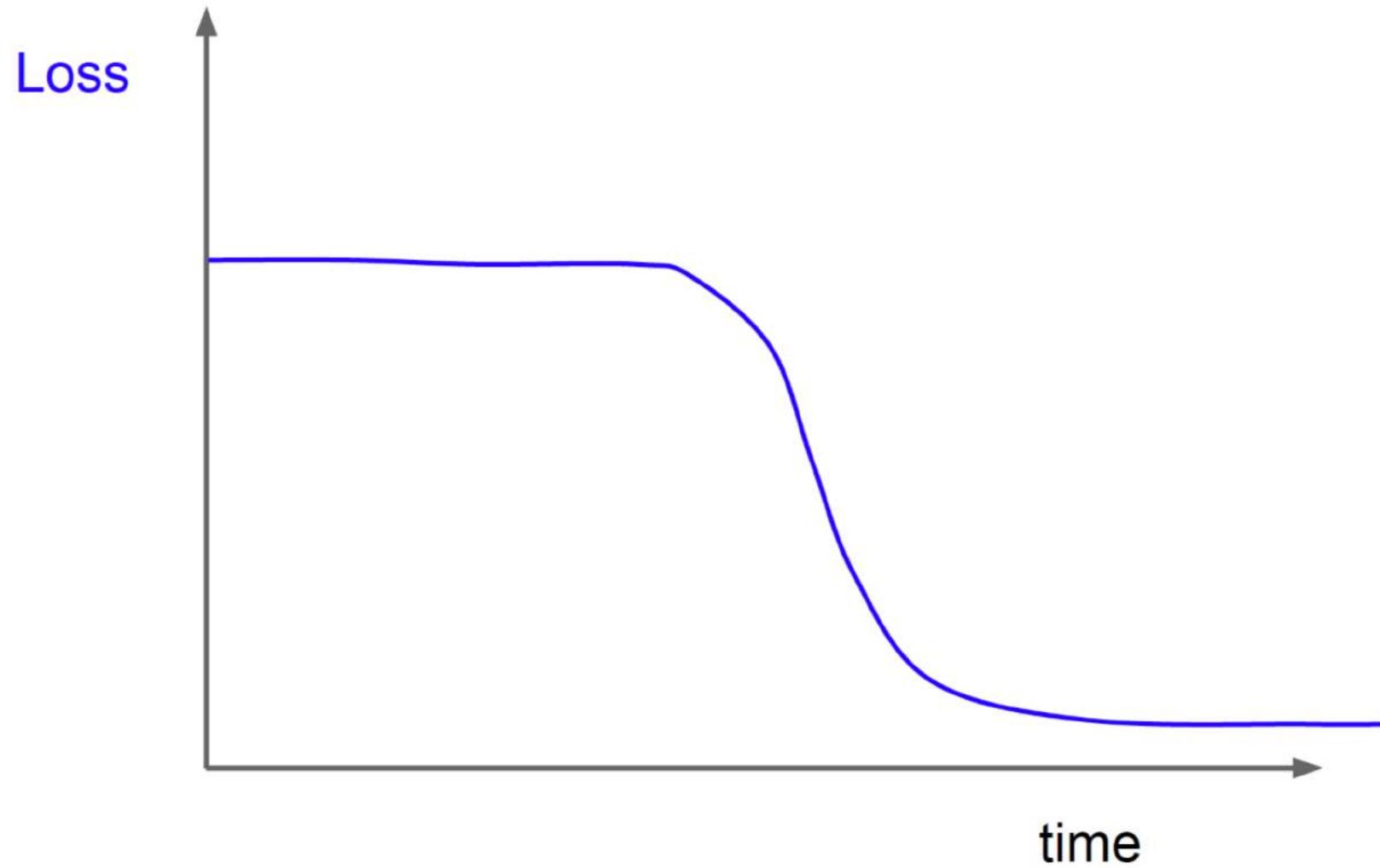
baseball

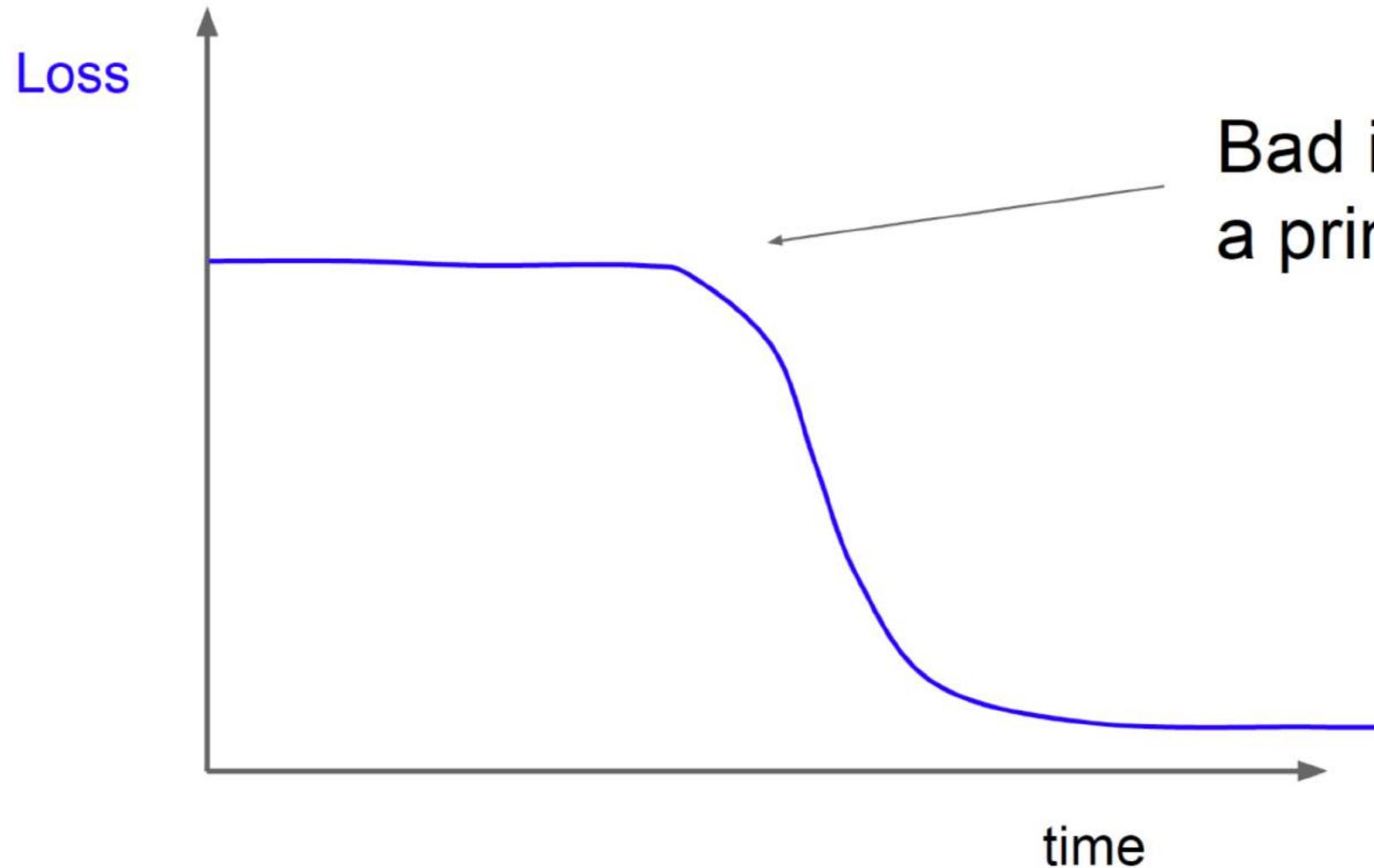


car

Monitor and visualize the loss curve

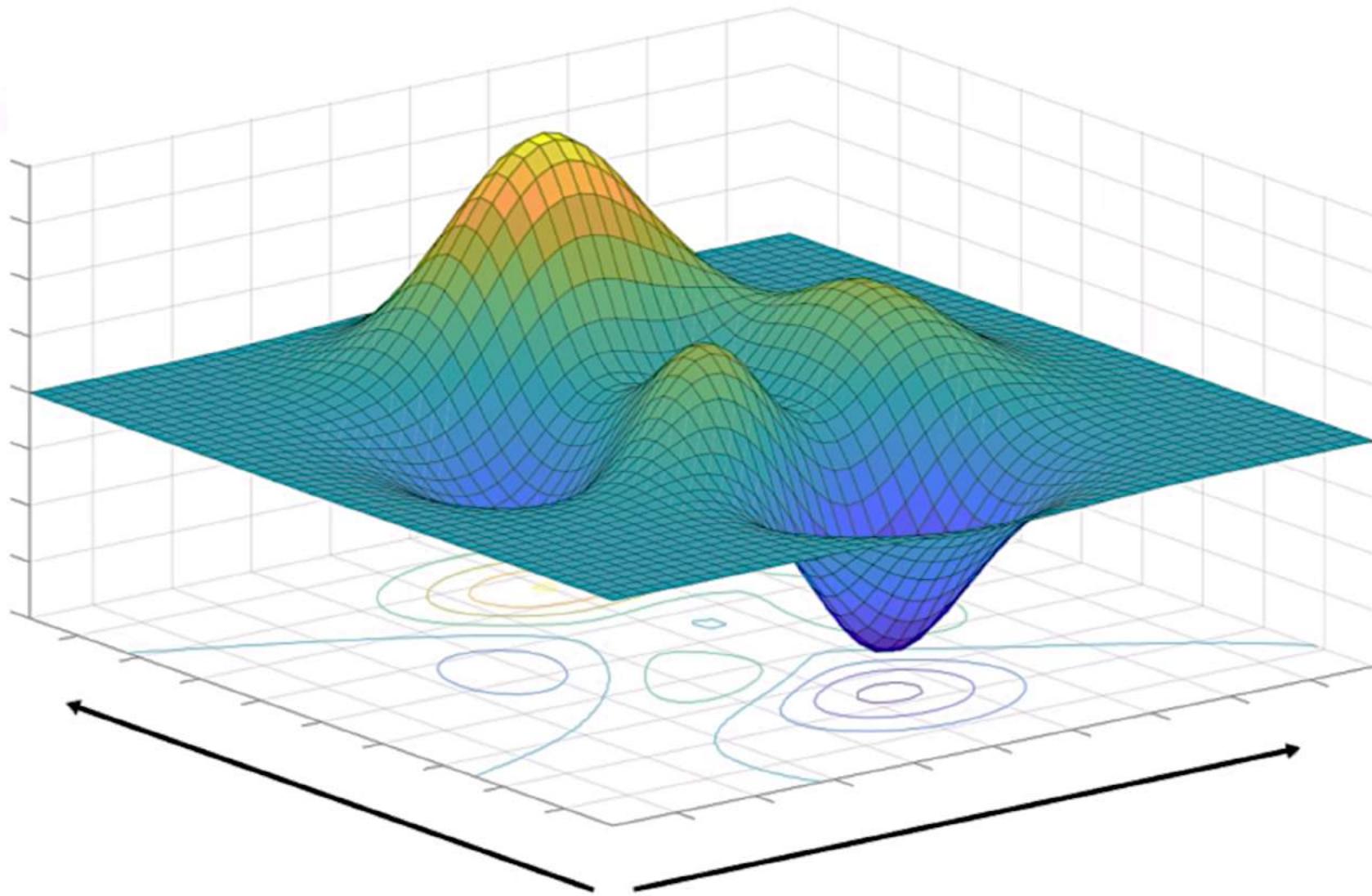


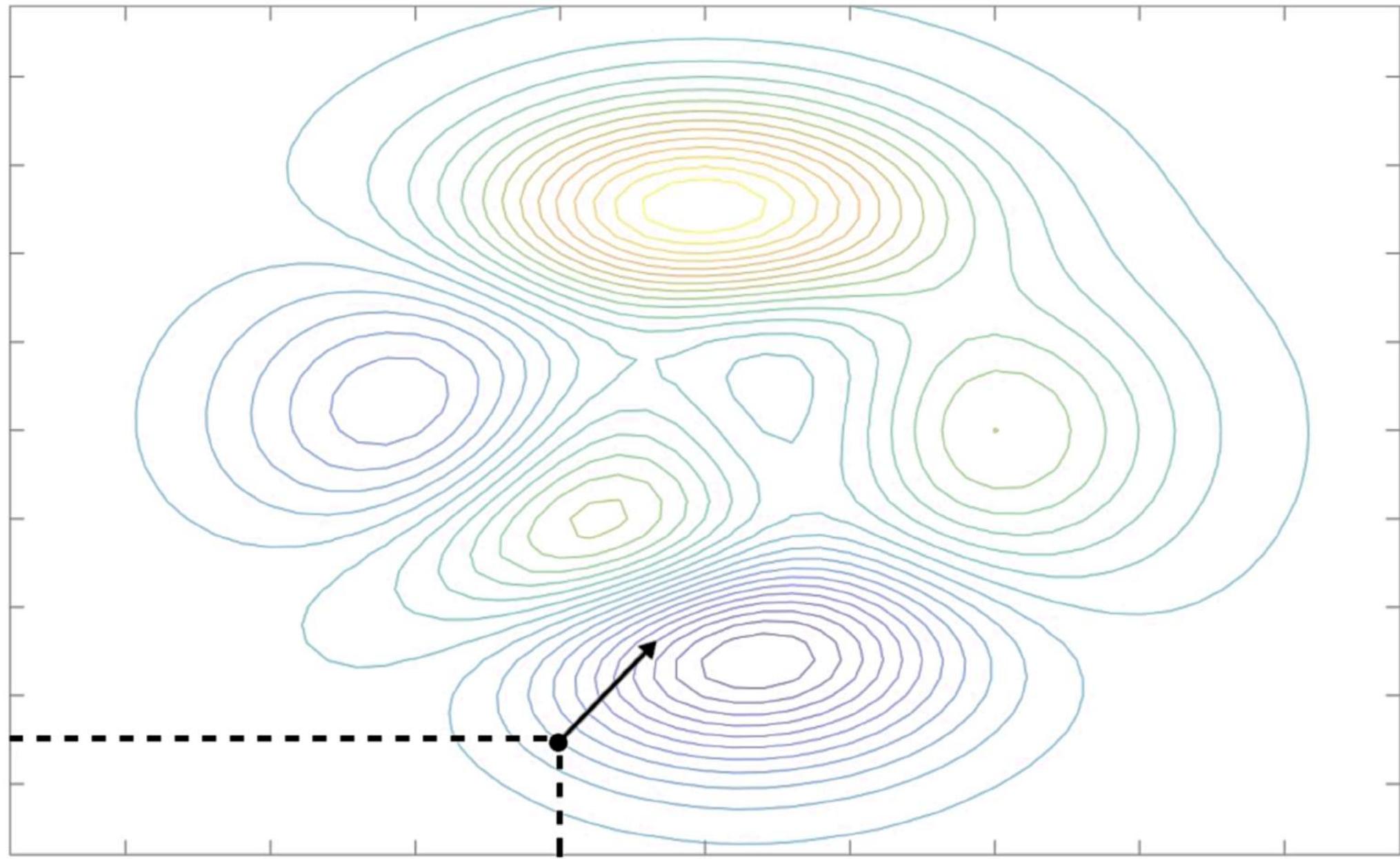


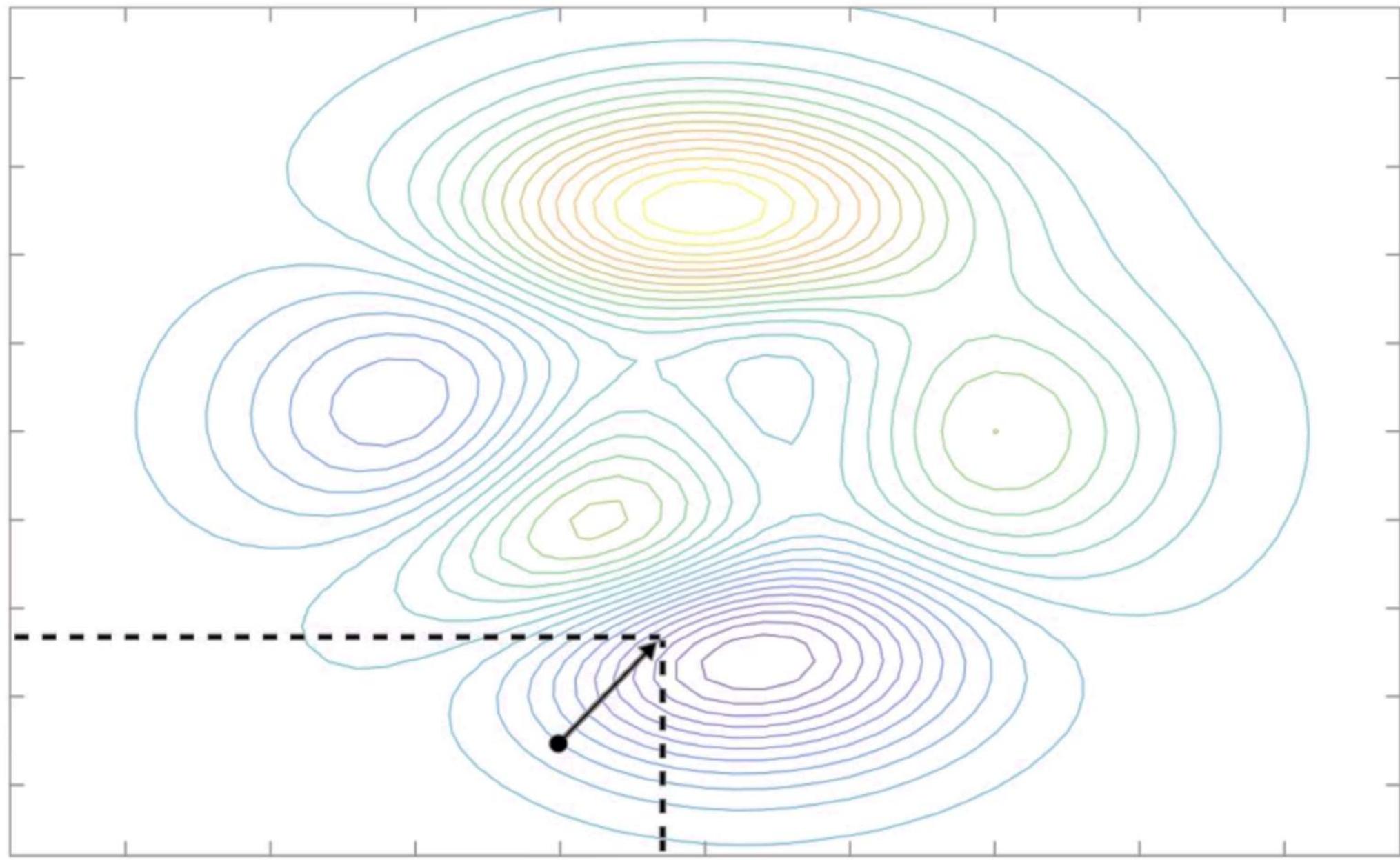


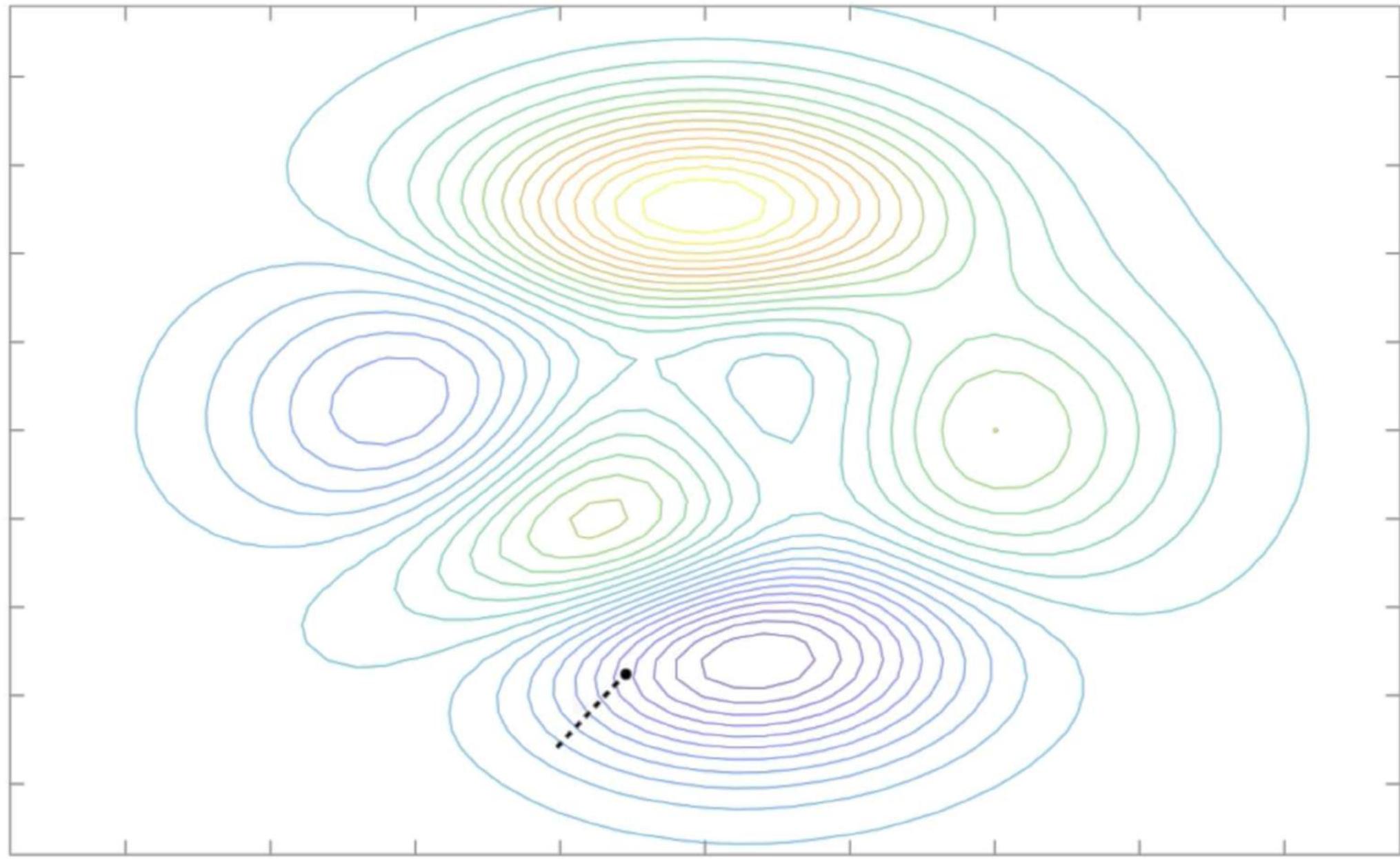
Bad initialization
a prime suspect

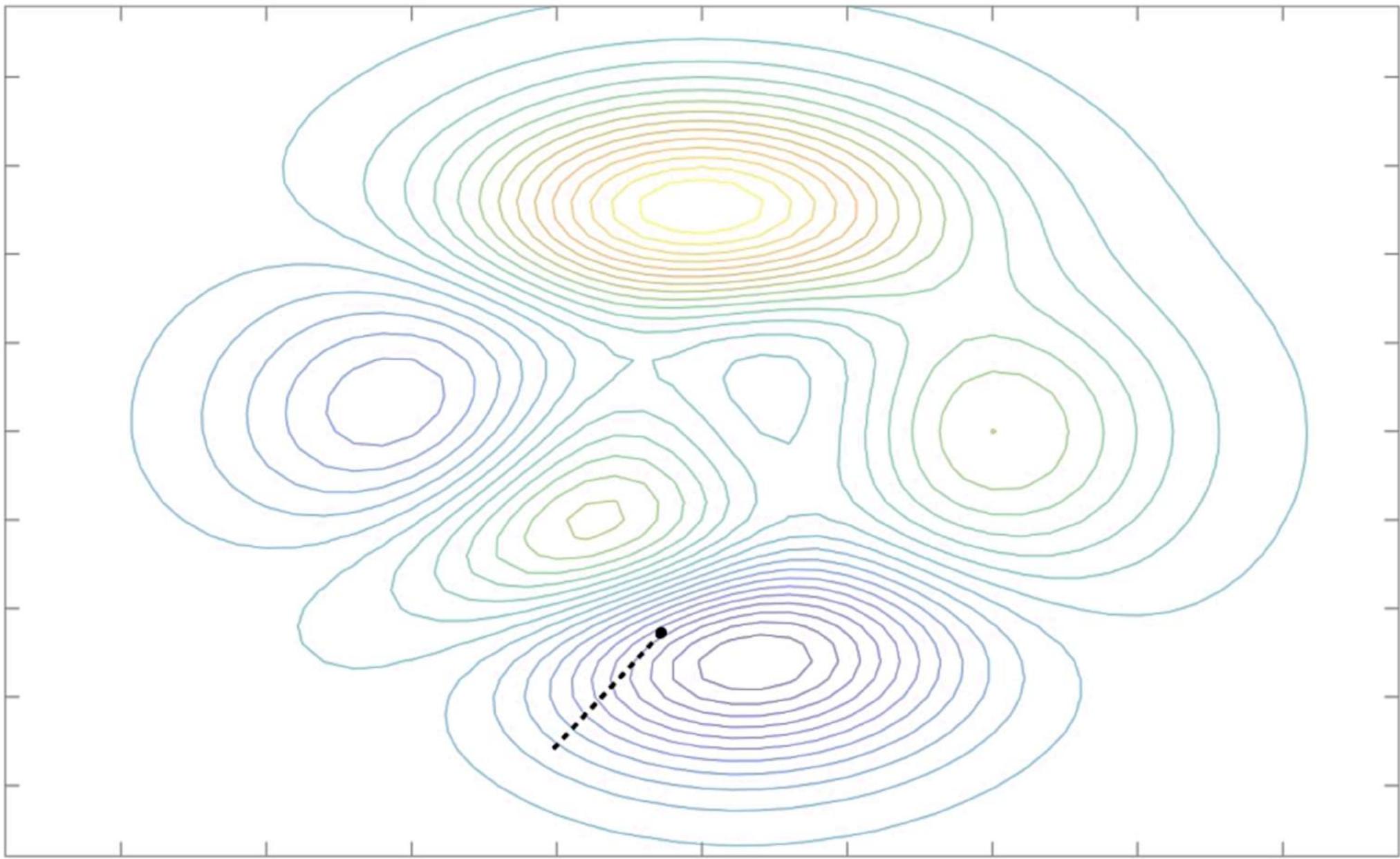
Loss

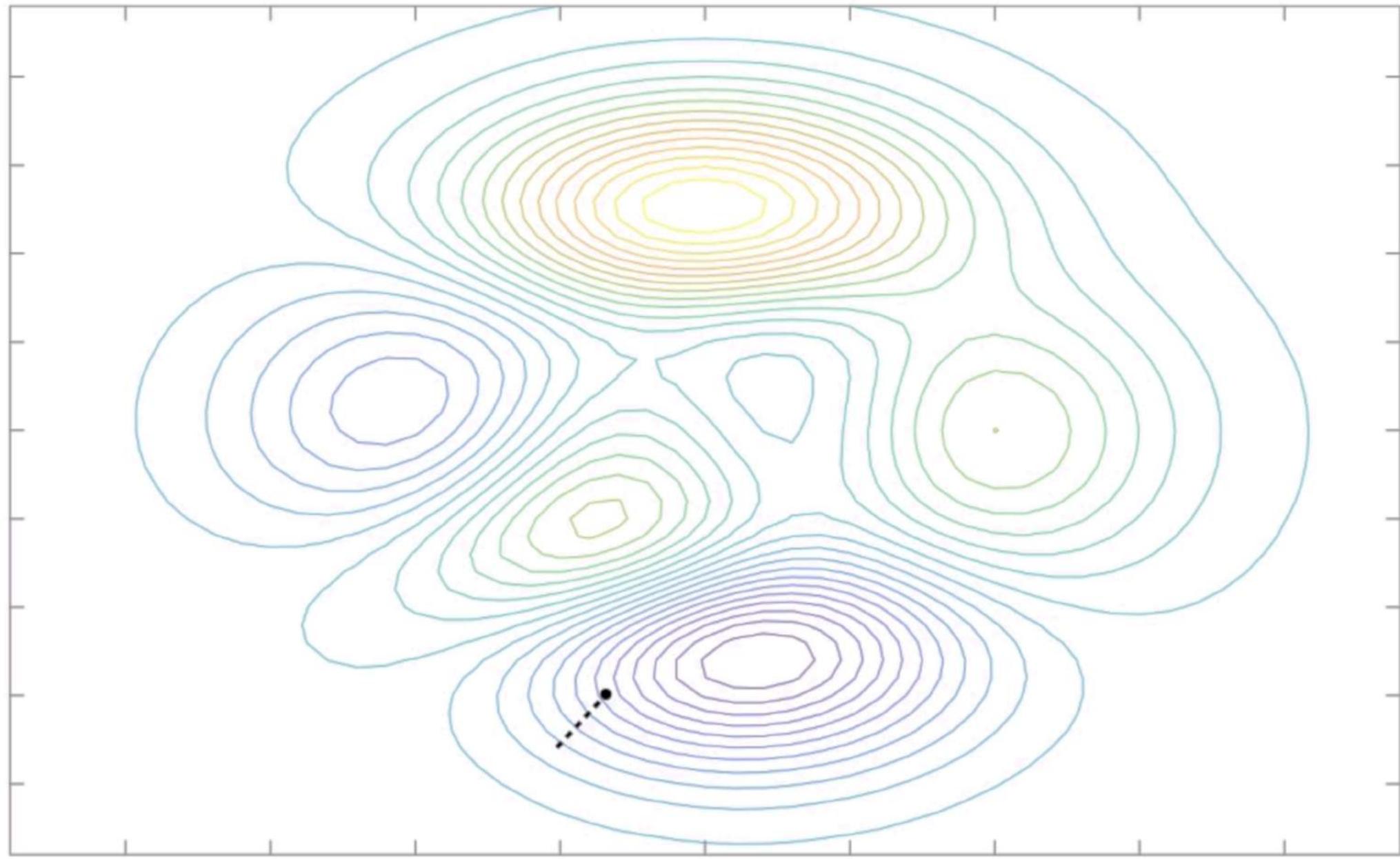


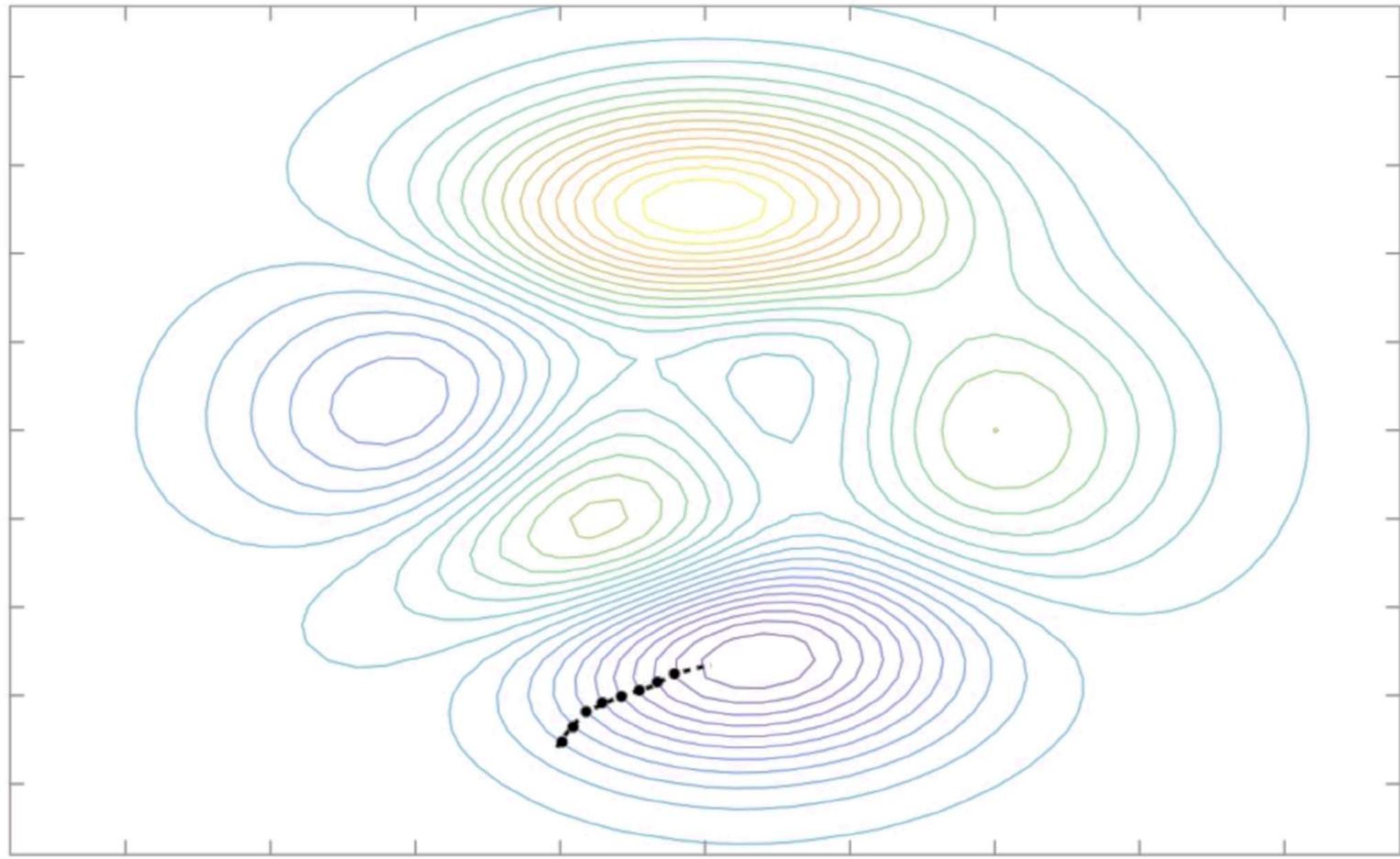


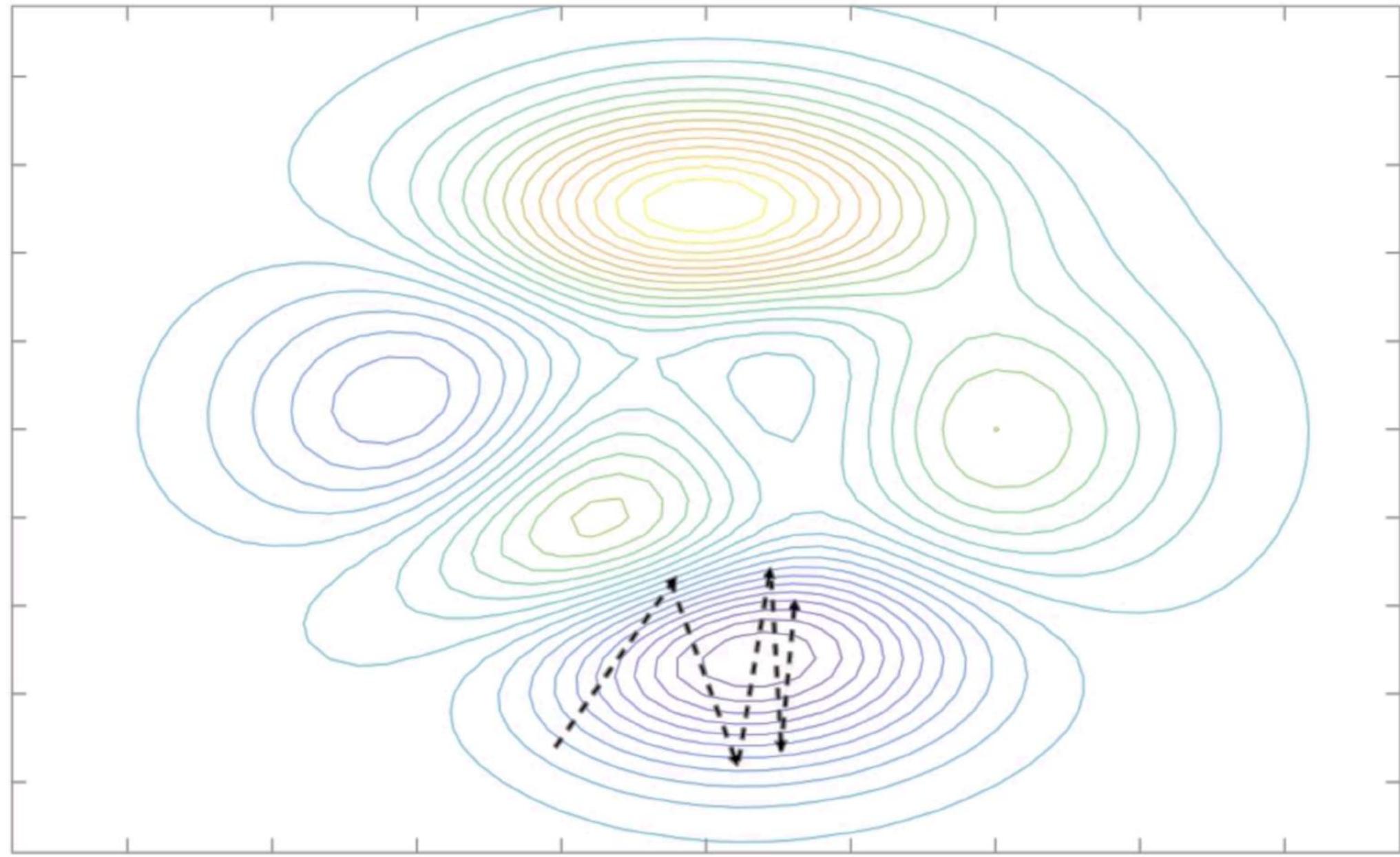


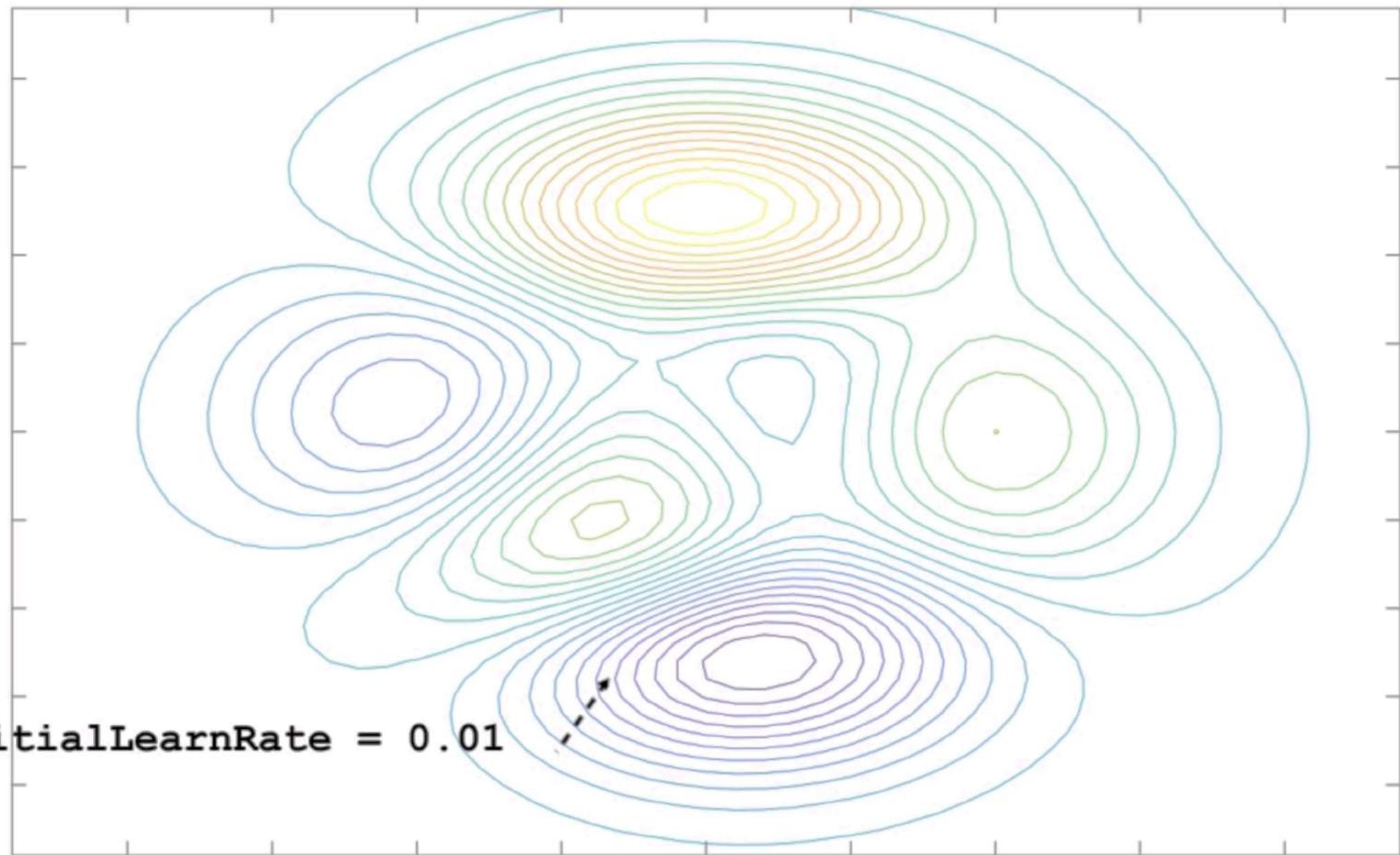




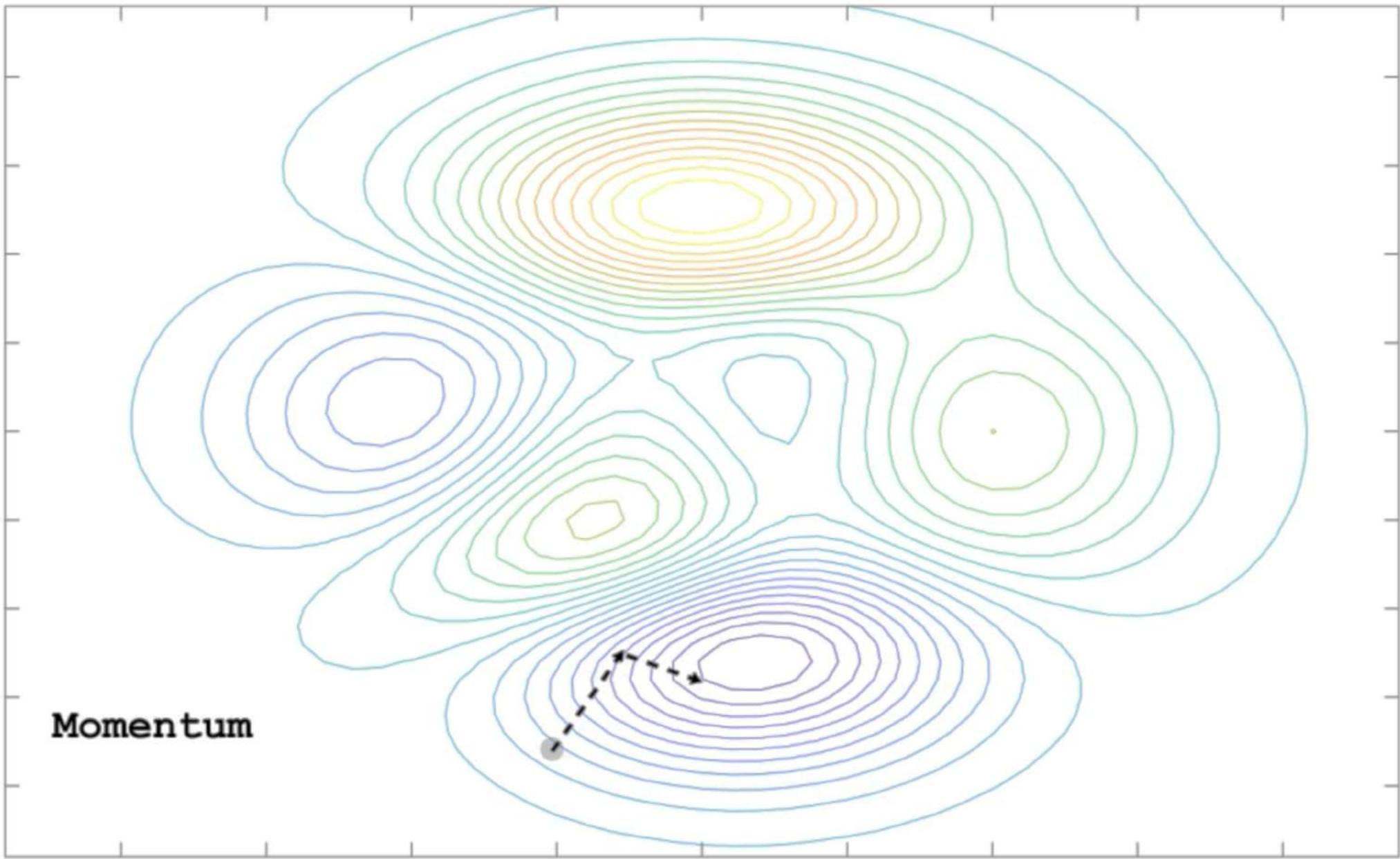


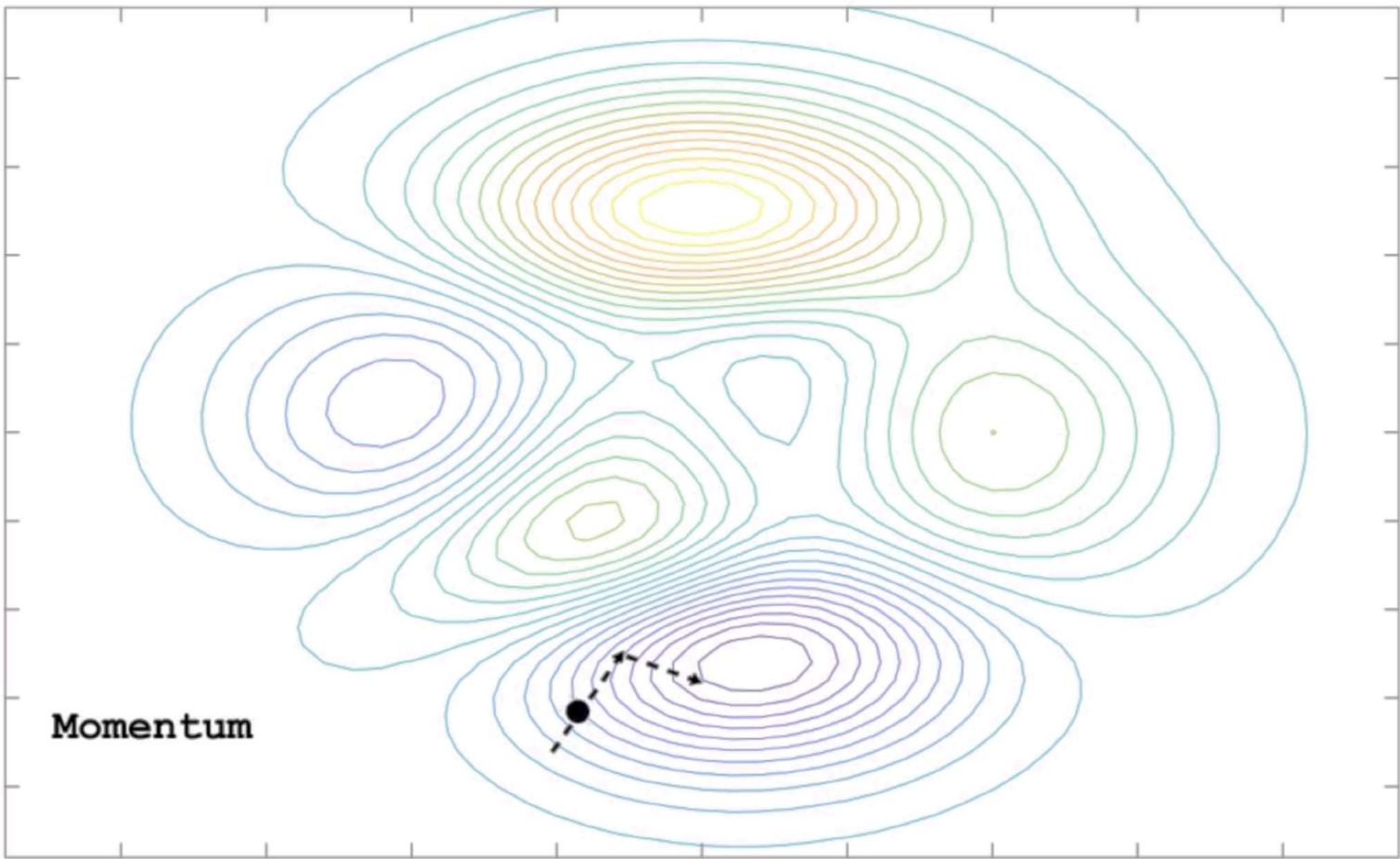


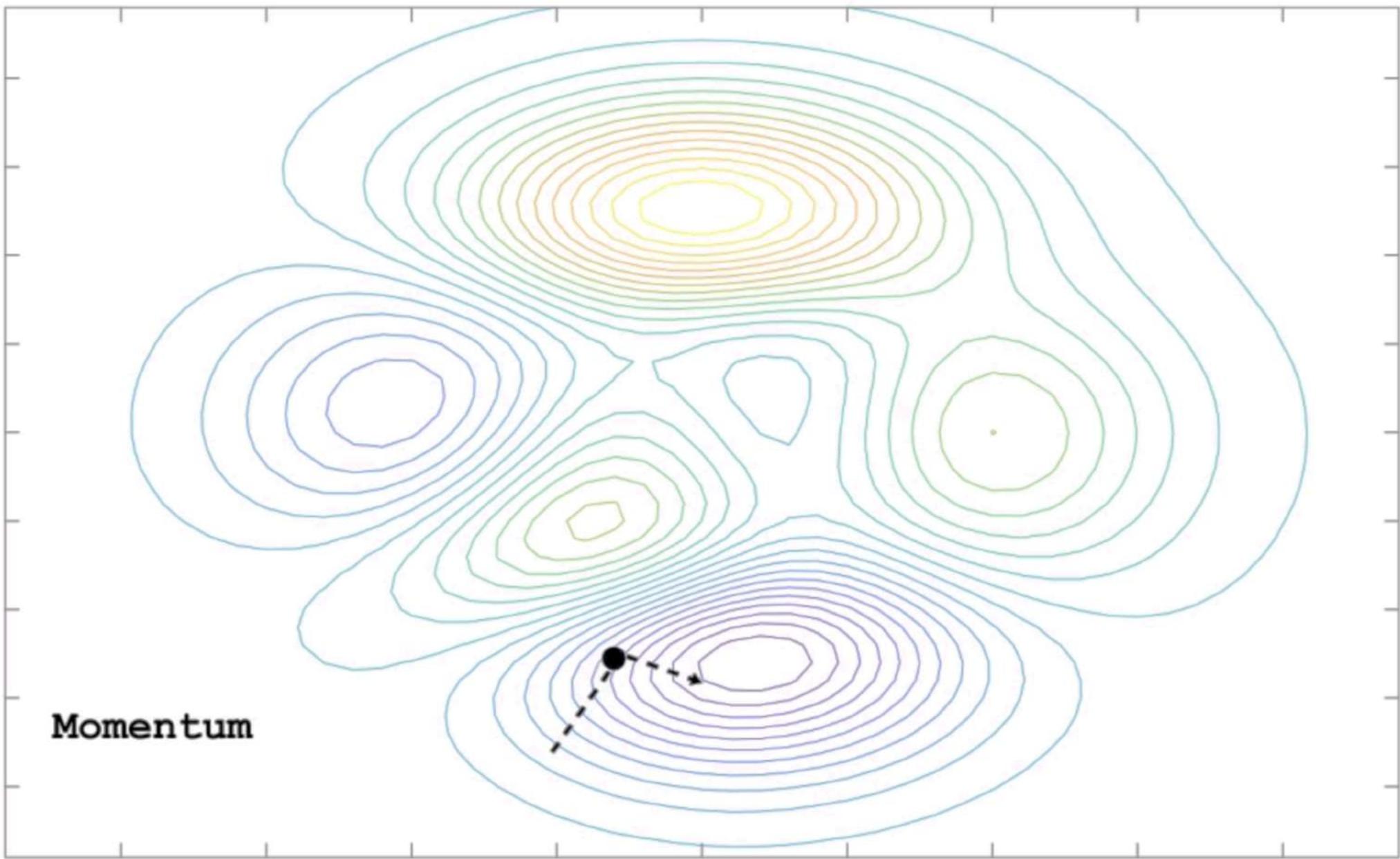


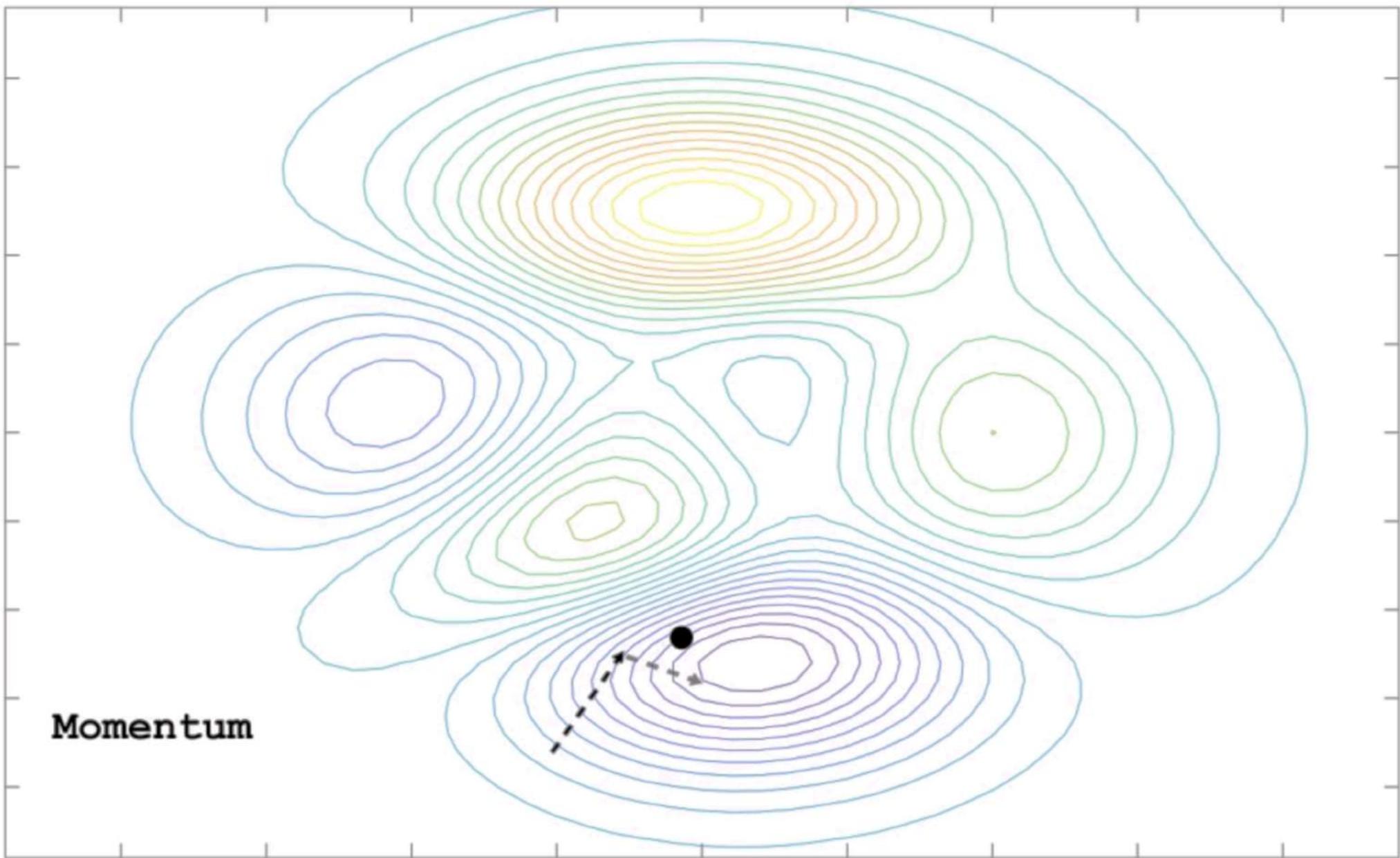


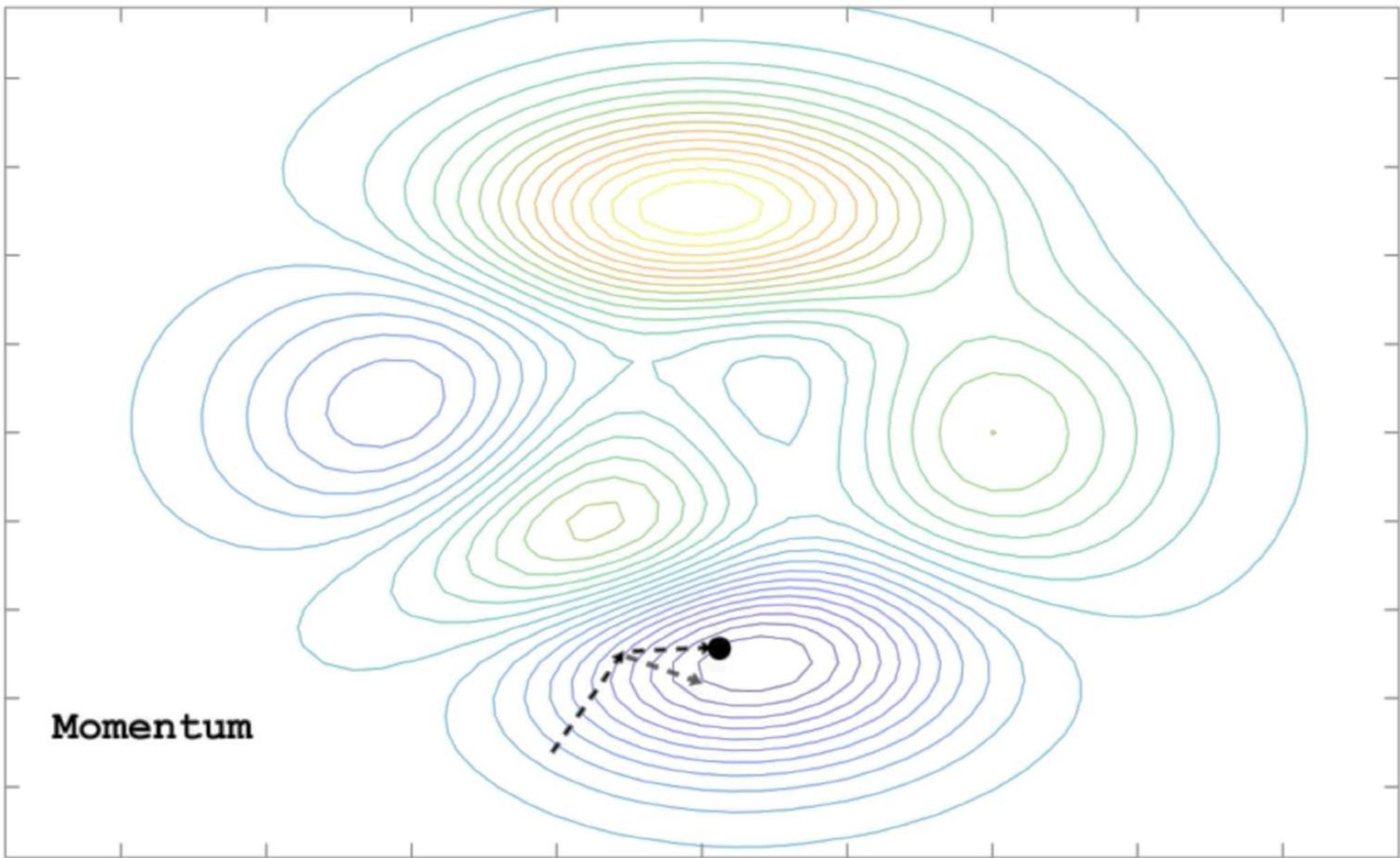
InitialLearnRate = 0.01



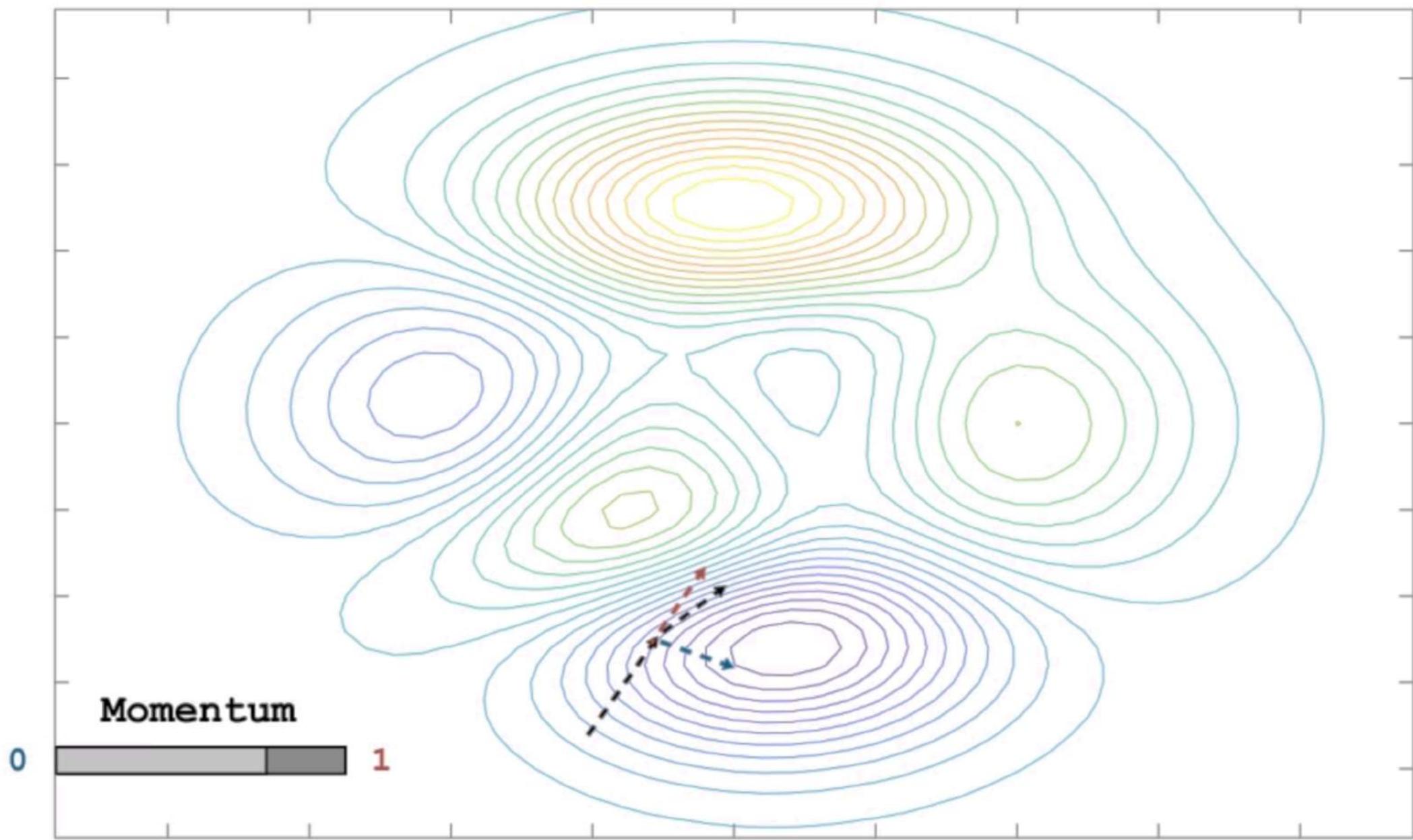


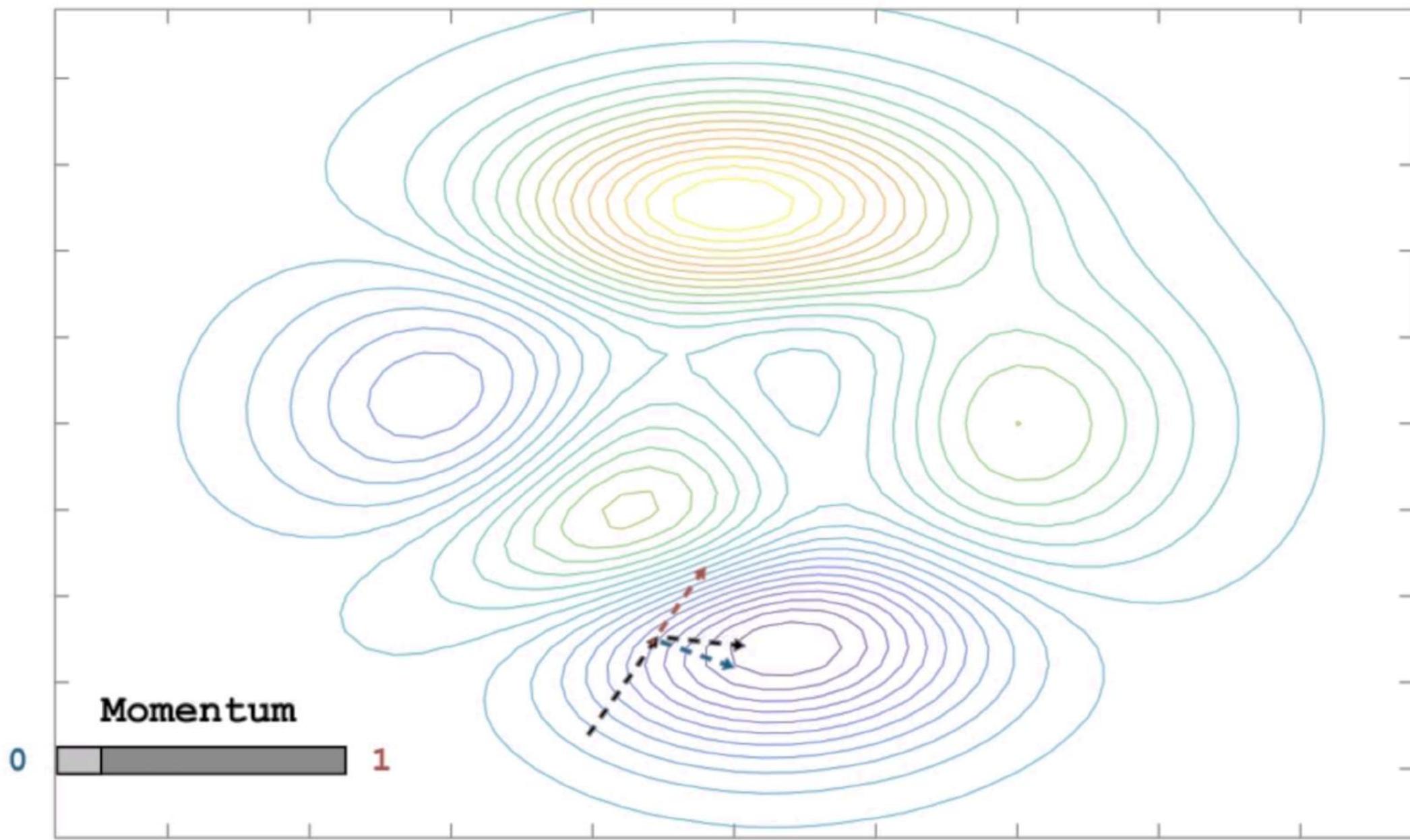




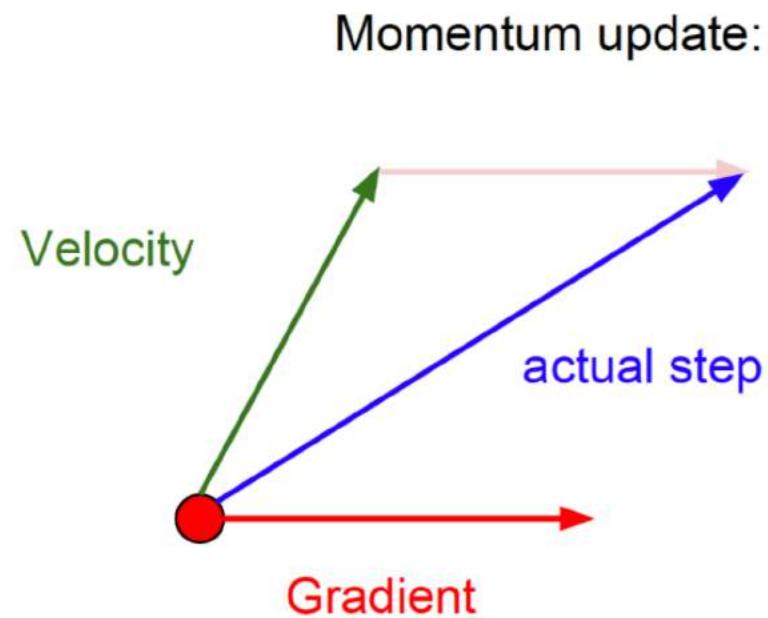


Momentum





SGD + Momentum



SGD + Momentum

SGD

$$x_{t+1} = x_t - \alpha \nabla f(x_t)$$

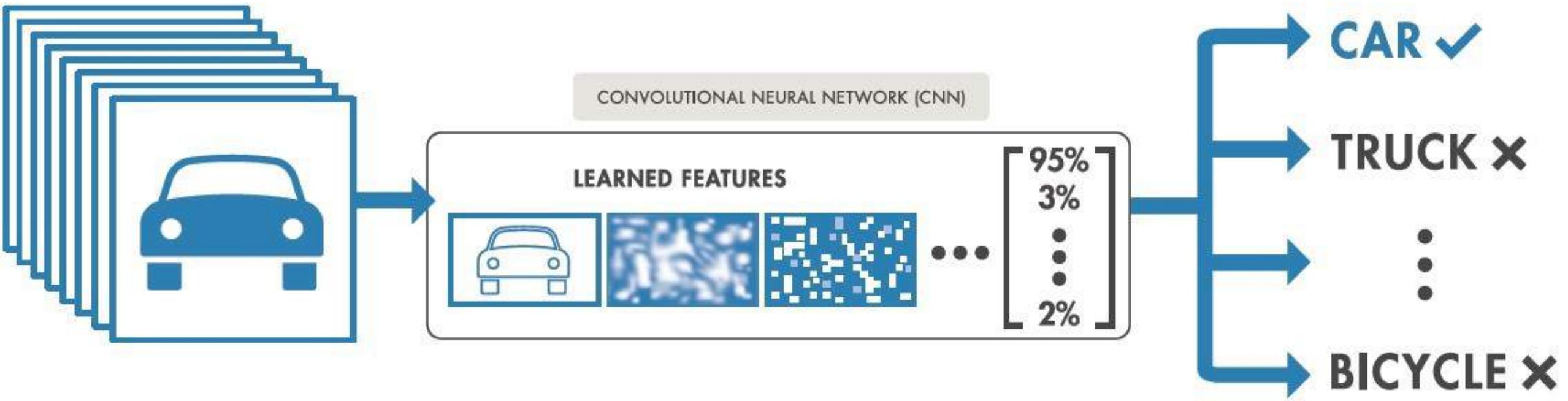
```
while True:  
    dx = compute_gradient(x)  
    x += learning_rate * dx
```

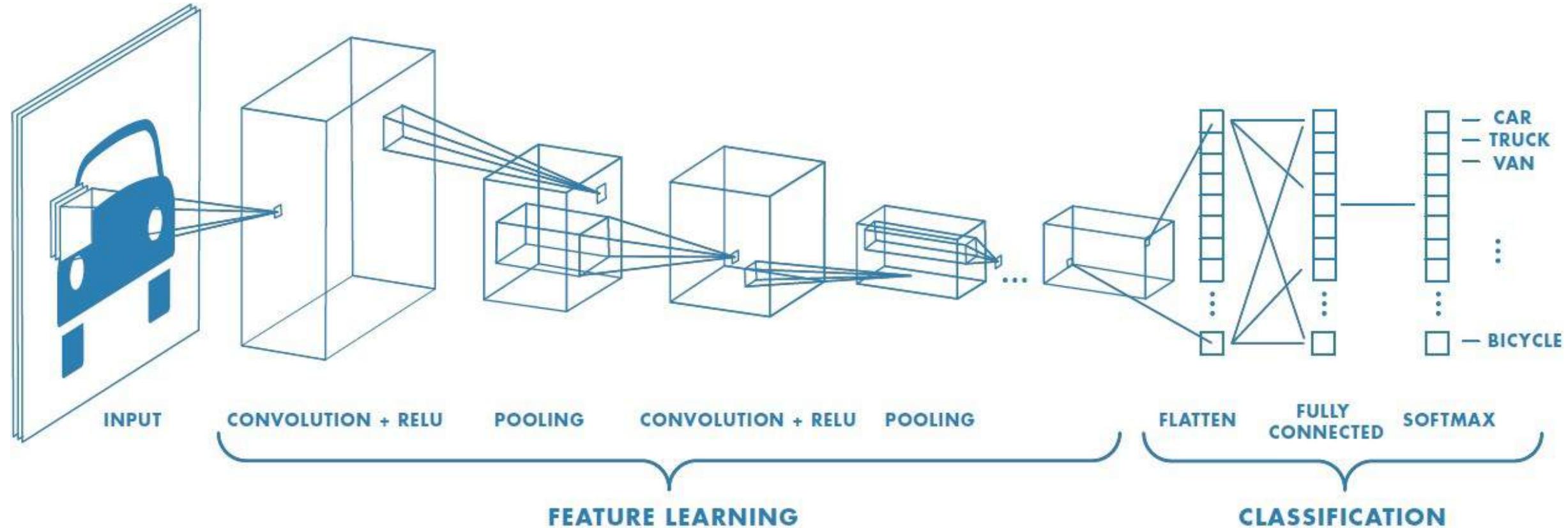
SGD+Momentum

$$\begin{aligned}v_{t+1} &= \rho v_t + \nabla f(x_t) \\x_{t+1} &= x_t - \alpha v_{t+1}\end{aligned}$$

```
vx = 0  
while True:  
    dx = compute_gradient(x)  
    vx = rho * vx + dx  
    x += learning_rate * vx
```

- Build up “velocity” as a running mean of gradients
- Rho gives “friction”; typically rho=0.9 or 0.99







Lets say you have 50,000 images

Dataset might be too large to fit in memory at the same time

Divide the data into Batches

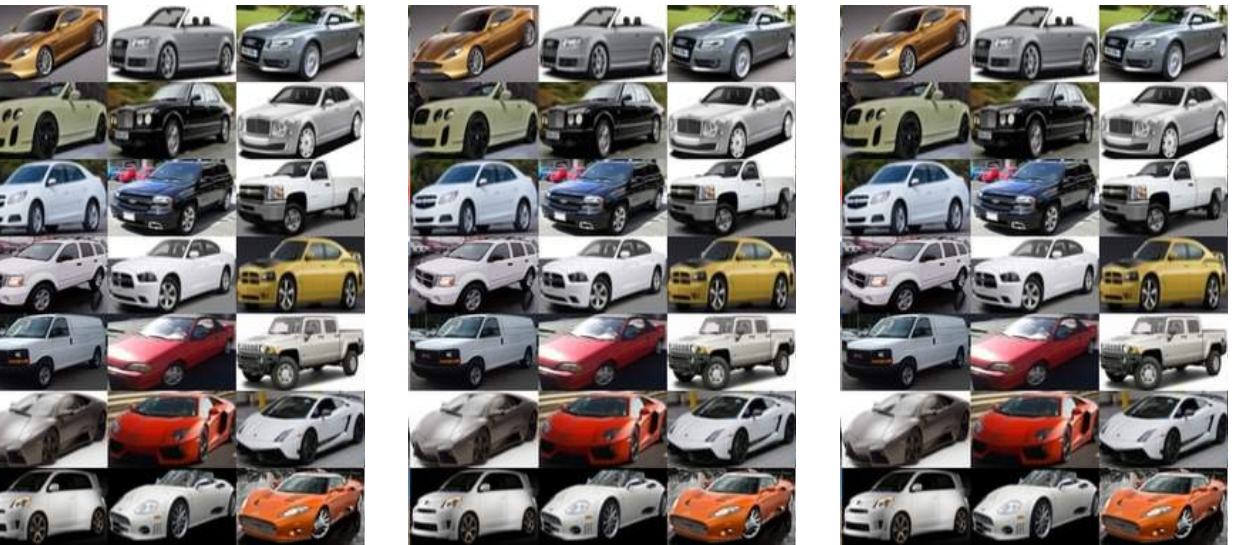


50,000 samples [images]

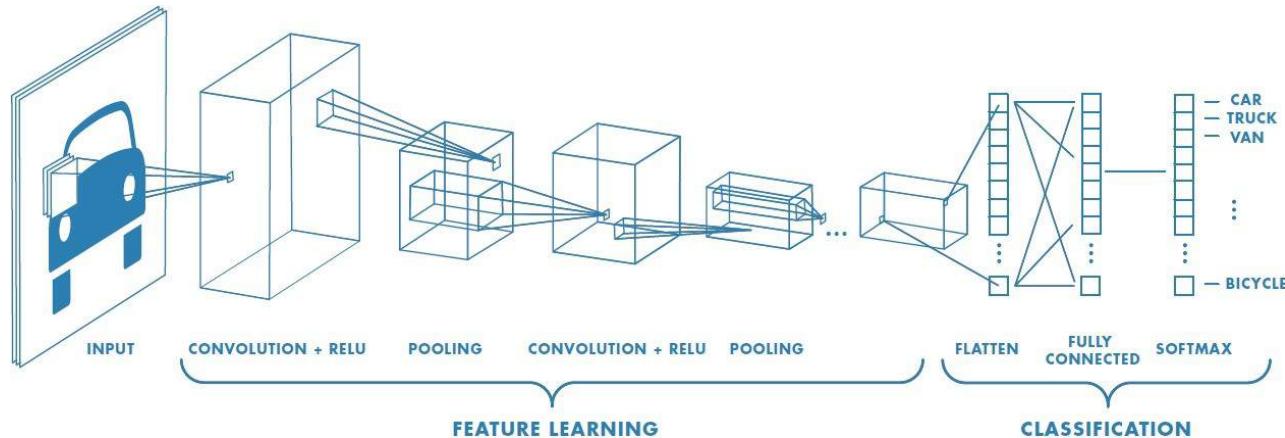
Batch Size of 2500 samples or 20 Batches



...



For each Batch



Loss

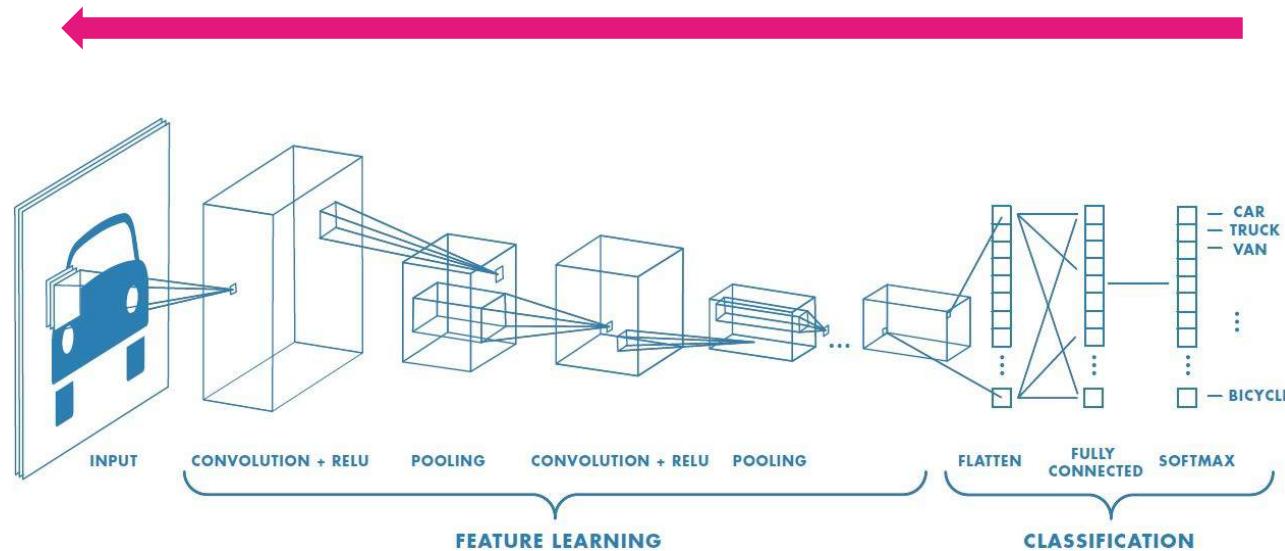
Batch 1
[2500 images]

→

Forward Pass

For each Batch

Backpropogation to adjust the weights

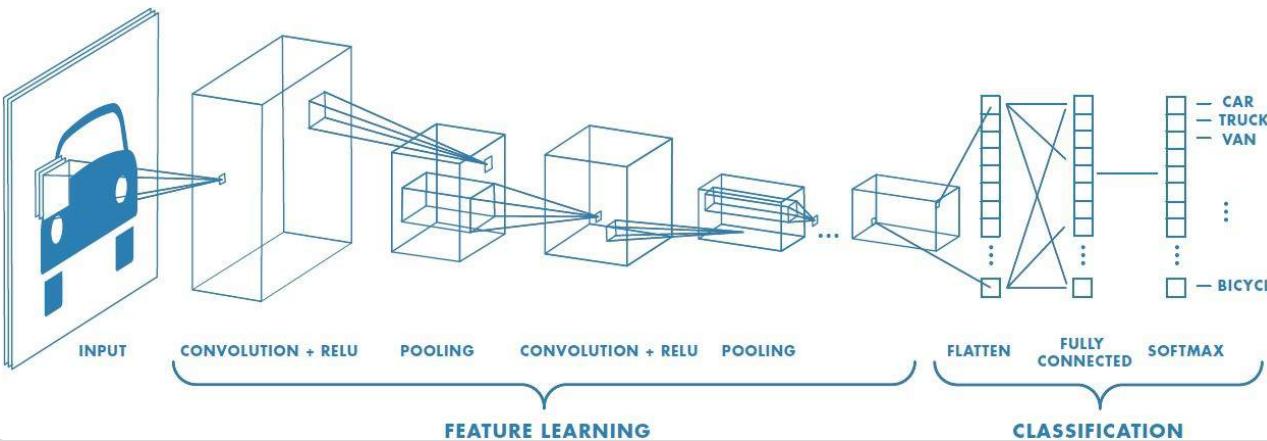


Loss

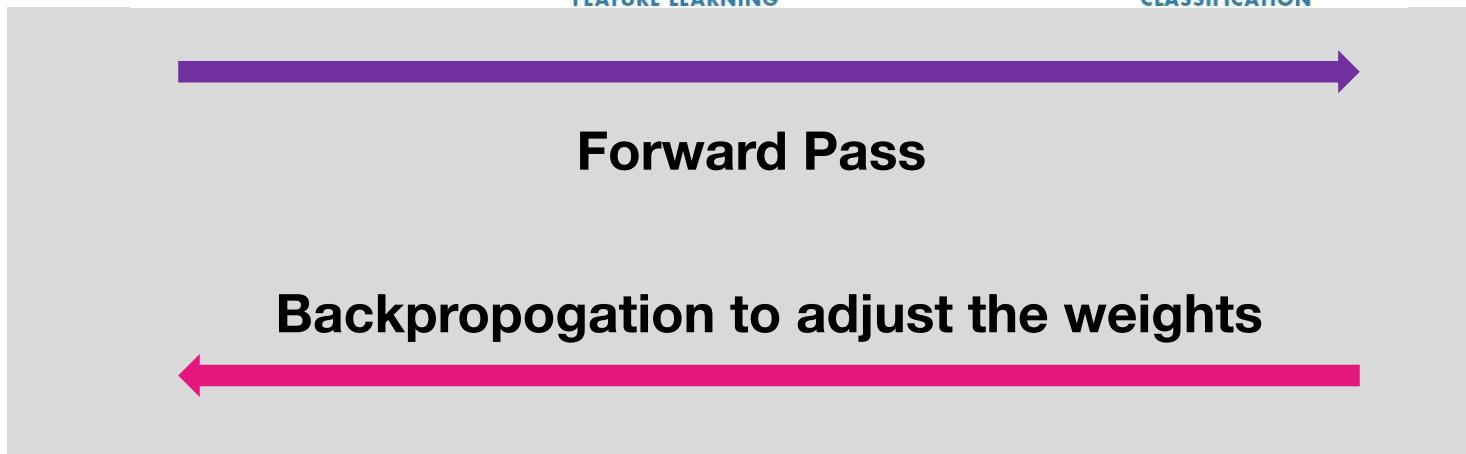
Batch 1
[2500 images]

Iteration – Fwd + Bck pass for one batch

Batch: number of samples to work through before updating the internal model parameters.



Batch 1
[2500 images]



Loss

one
iteration

Forward Pass

Backpropagation to adjust the weights

Epoch



Batch 1

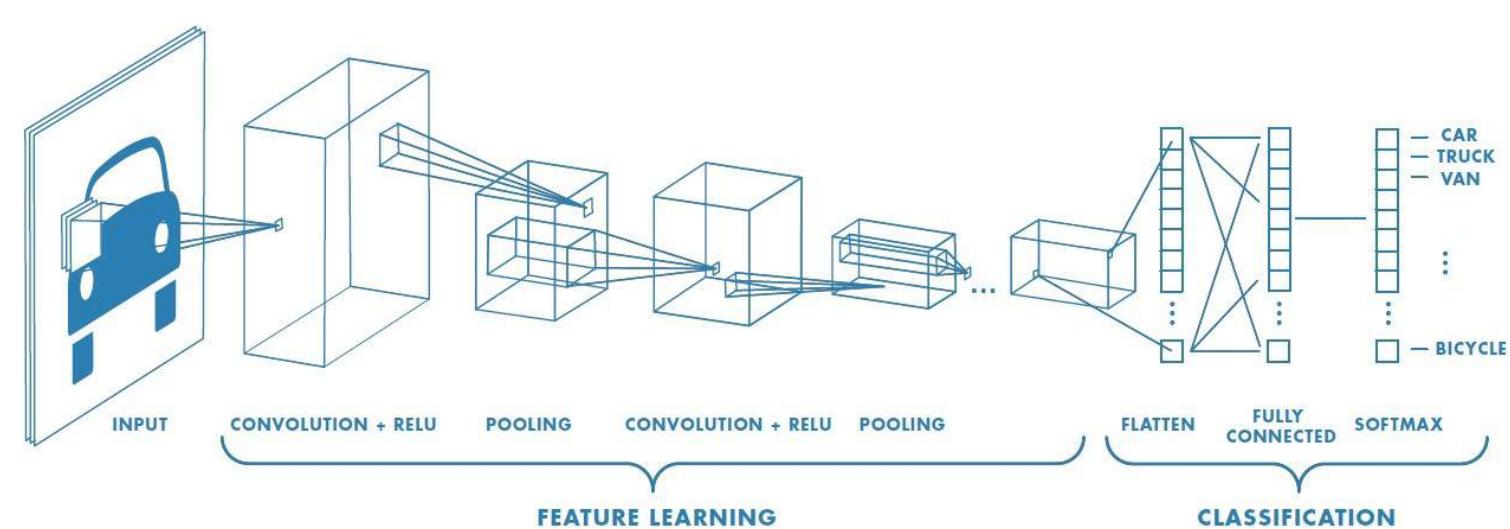


Batch 2



The network has seen all the data (all the batches) once

Batch 20





50,000 samples [images]

Batch Size of 2500 samples or 20 Batches



....



One Epoch = 20 model parameter updates



50,000 samples [images]

Batch Size of 2500 samples or 20 Batches



....



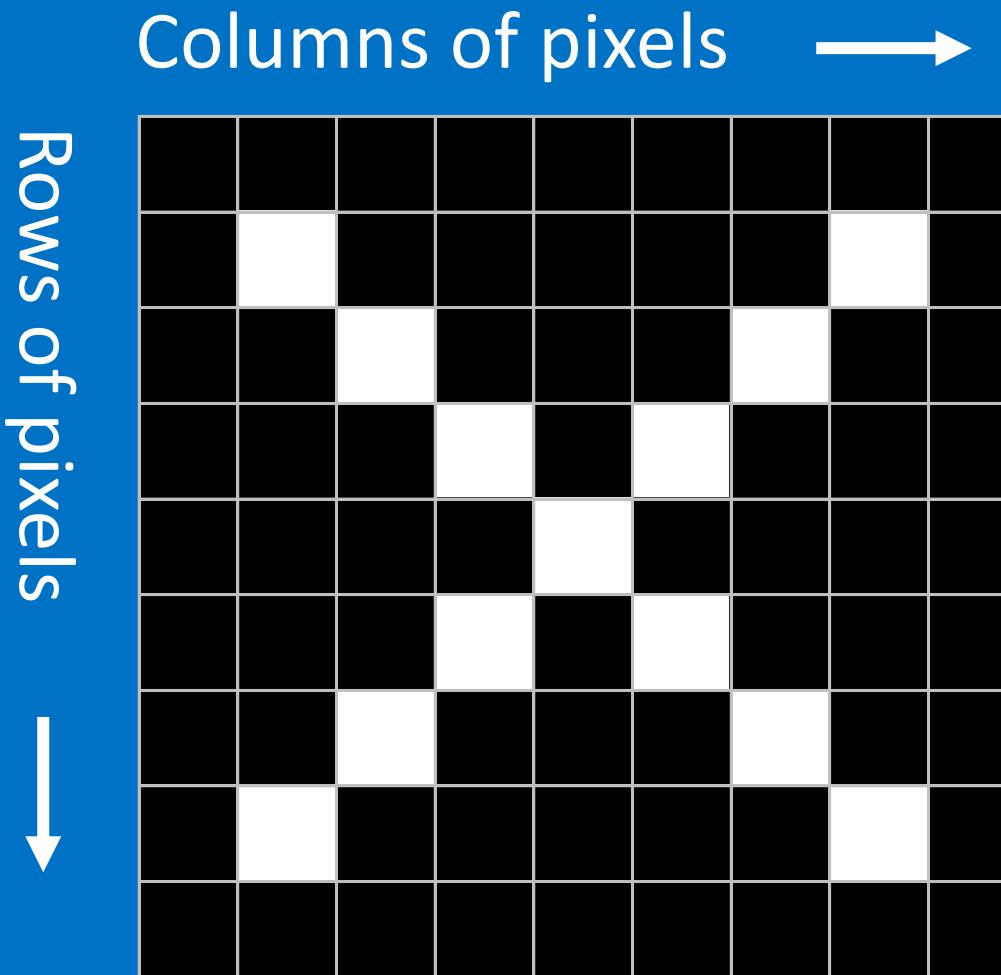
Train for multiple Epochs – e.g. 10 Epochs == 200 Batches

Not just images

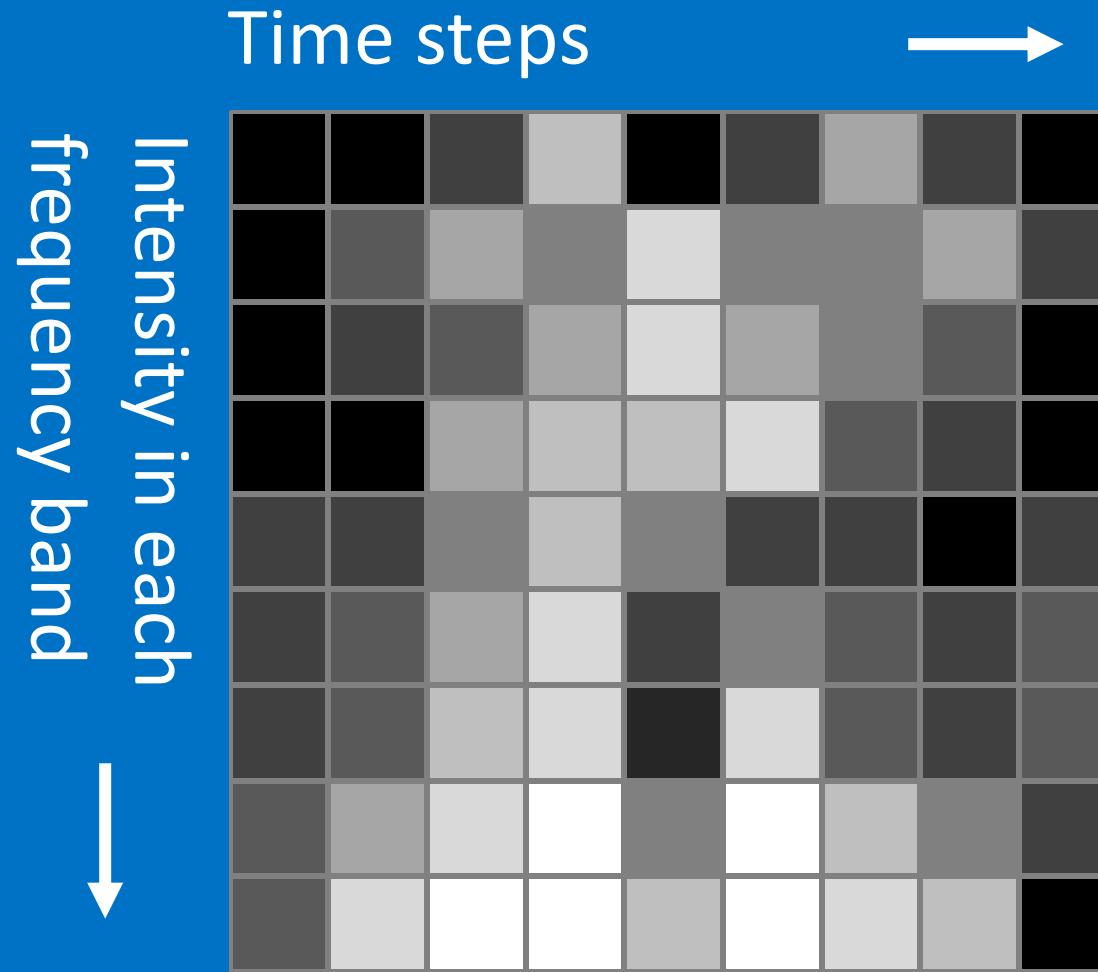
Any 2D (or 3D) data.

Things closer together are more closely related than things far away.

Images



Sound

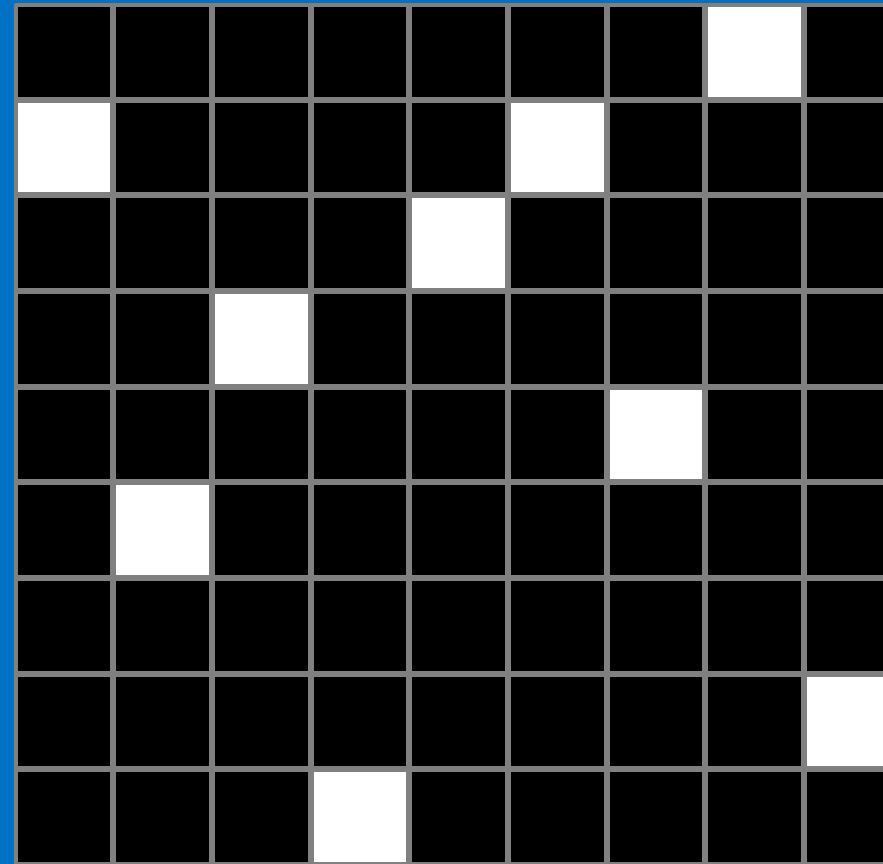


Text

Position in sentence



Words in
dictionary



Limitations

ConvNets only capture local “spatial” patterns in data.
If the data can’t be made to look like an image,
ConvNets are less useful.

Customer data

Customers



Name, age,
address, email,
purchases,
browsing activity,...



| | | | | | | | | |
|---|-----|----|------------------------------|---|----|------|-----|-----|
| A | 22 | 1A | a@a | 1 | aa | a1.a | 123 | aa1 |
| B | 33 | 2B | b@b | 2 | bb | b2.b | 234 | bb2 |
| C | 44 | 3C | c@c | 3 | cc | c3.c | 345 | cc3 |
| D | 55 | 4D | d@d | 4 | dd | d4.d | 456 | dd4 |
| E | 66 | 5E | e@e | 5 | ee | e5.e | 567 | ee5 |
| F | 77 | 6F | f@f | 6 | ff | f6.f | 678 | ff6 |
| G | 88 | 7G | g@g | 7 | gg | g7.g | 789 | gg7 |
| H | 99 | 8H | h@h | 8 | hh | h8.h | 890 | hh8 |
| I | 111 | 9I | i@i | 9 | ii | i9.i | 901 | ii9 |

Rule of thumb

If your data is just as useful after swapping any of your columns with each other, then you can't use Convolutional Neural Networks.

In a nutshell

ConvNets are great at finding patterns and using them to classify images.

