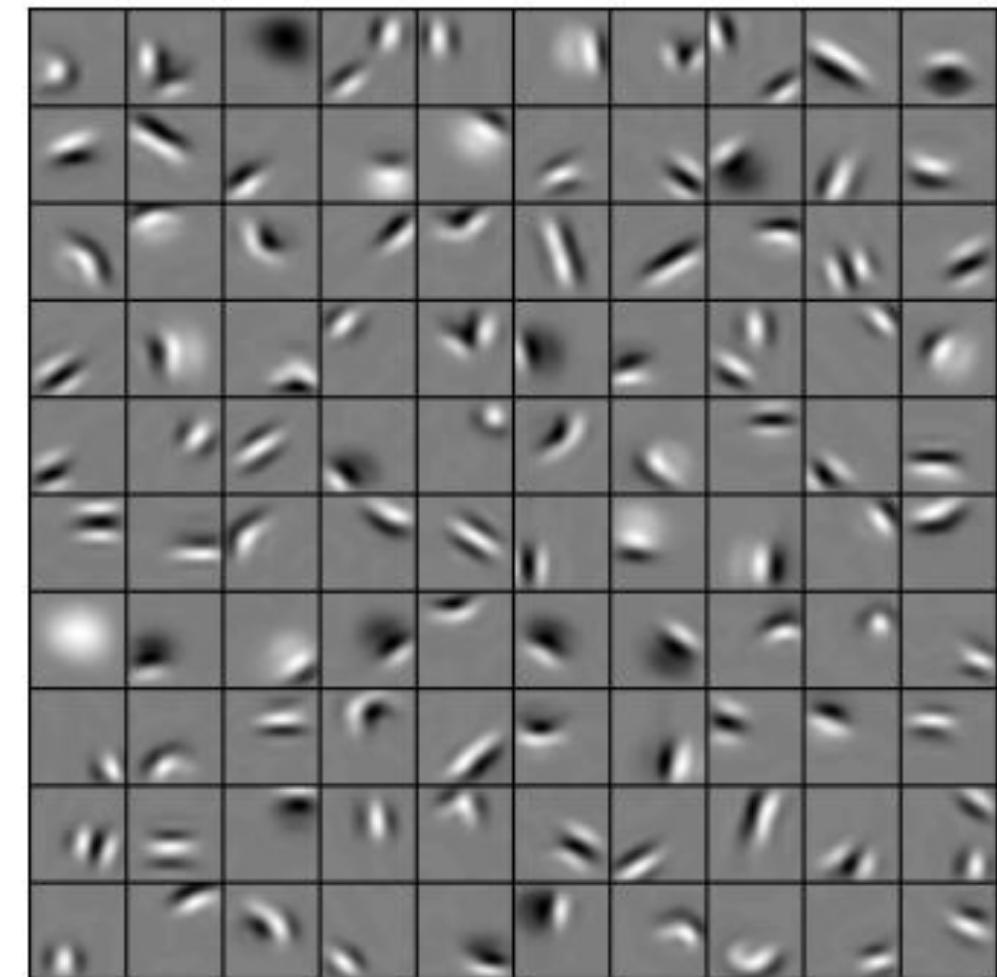
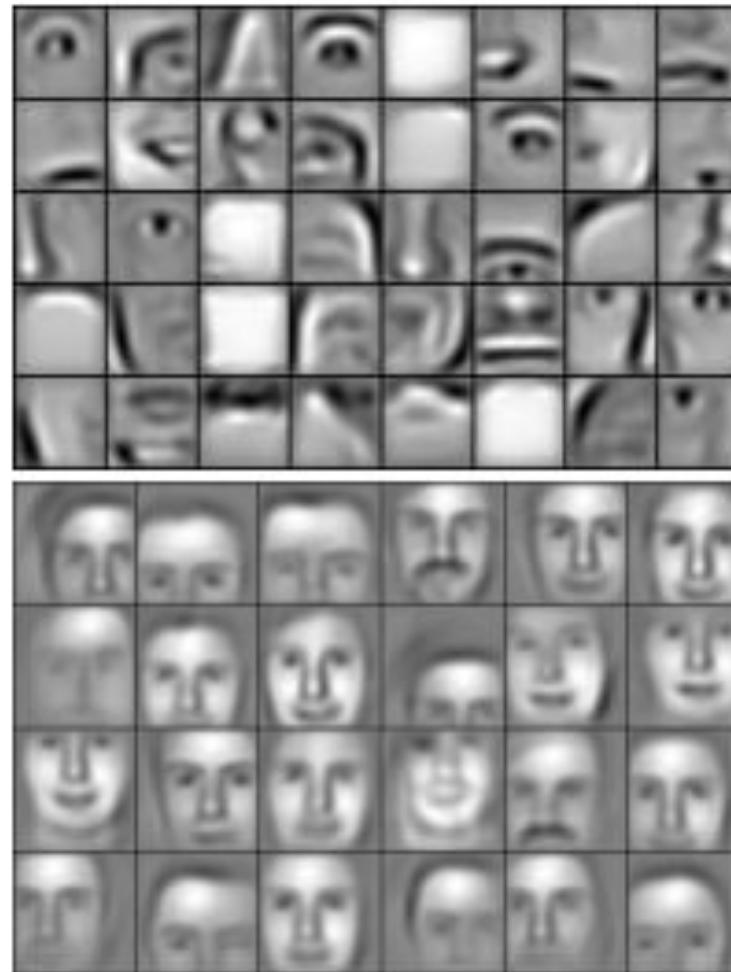


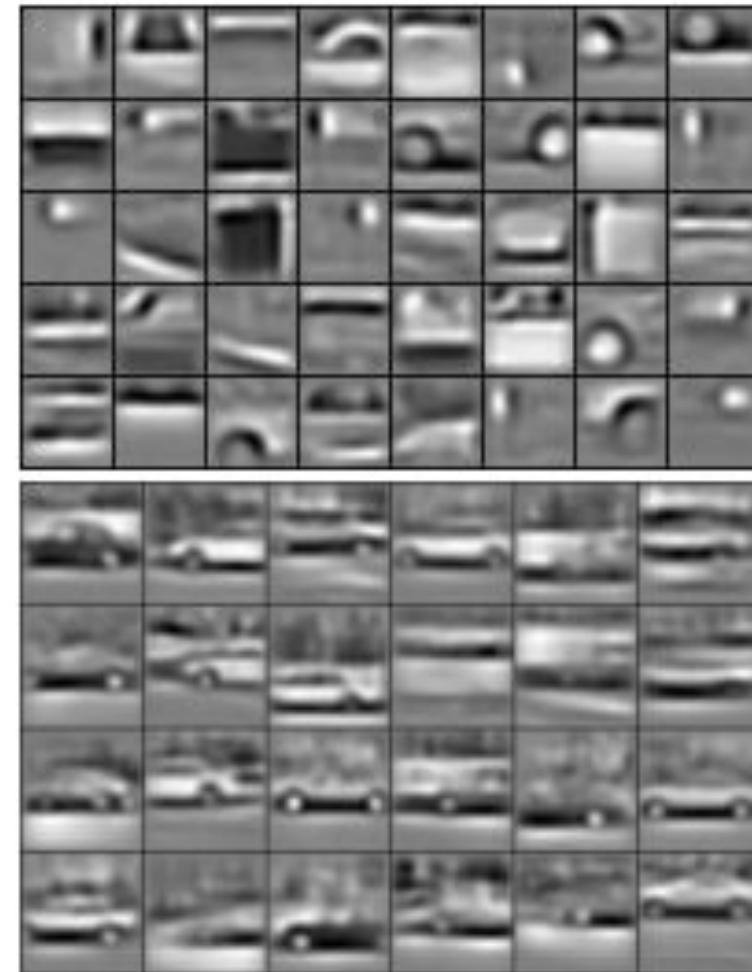
How
Convolutional Neural
Networks
Work



faces



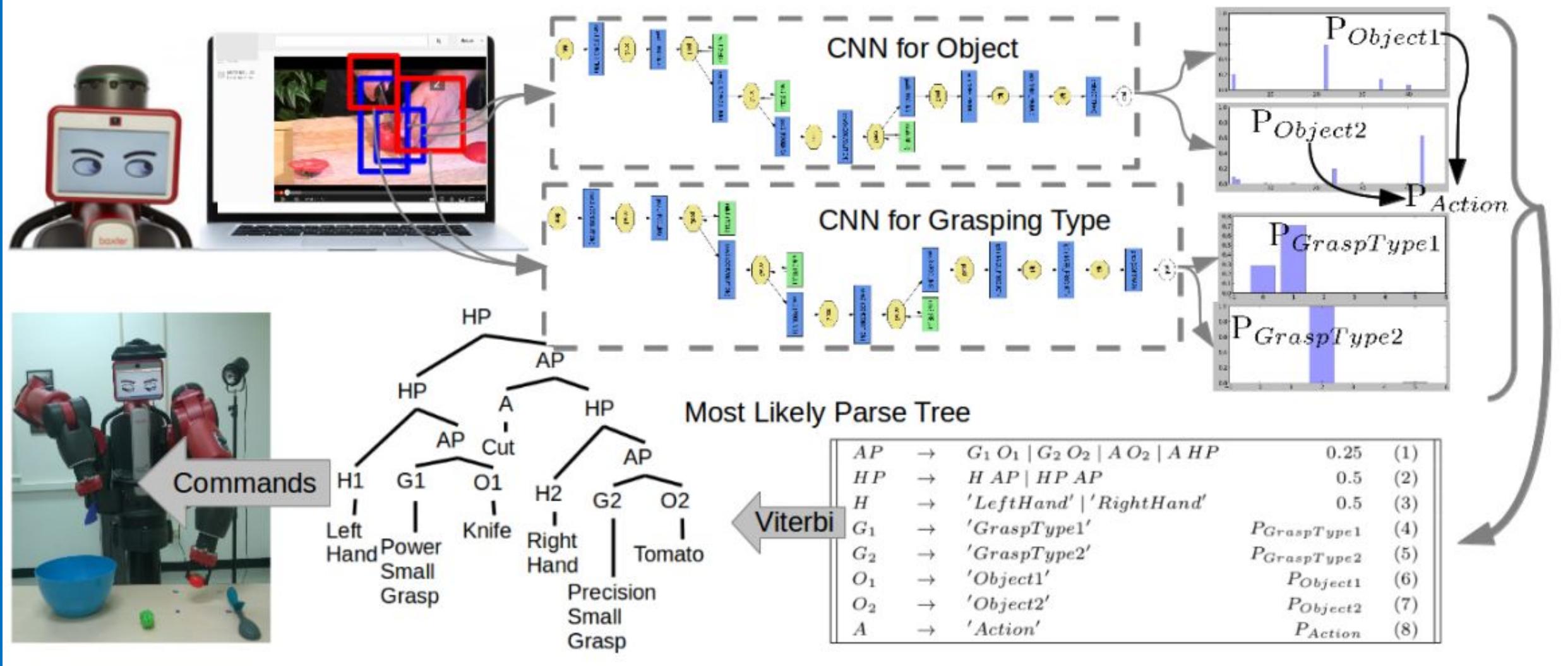
cars



Convolutional Deep Belief Networks for Scalable Unsupervised
Learning of Hierarchical Representations
Honglak Lee, Roger Grosse, Rajesh Ranganath, Andrew Y. Ng



Playing Atari with Deep Reinforcement Learning.
Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex
Graves, Ioannis Antonoglou, Daan Wierstra, Martin
Riedmiller

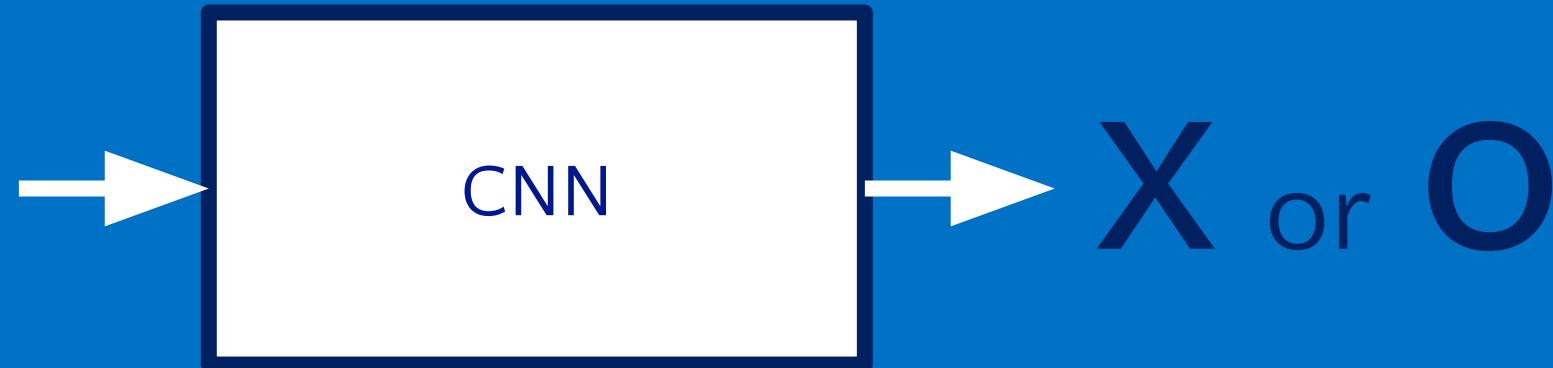
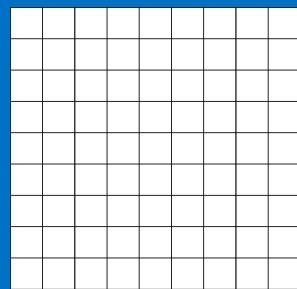


Robot Learning Manipulation Action Plans by “Watching” Unconstrained Videos from the World Wide Web.
 Yezhou Yang, Cornelia Fermüller, Yiannis Aloimonos

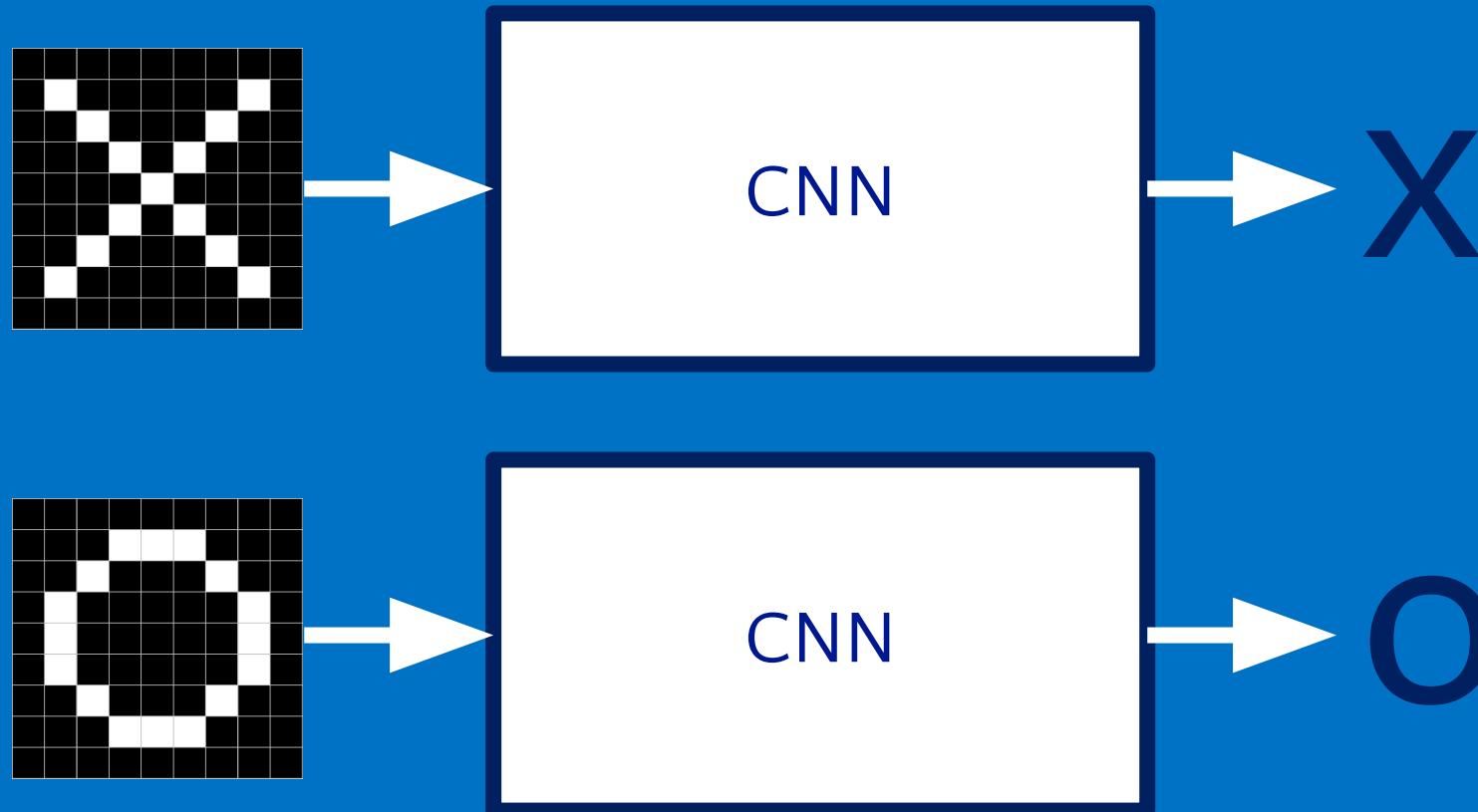
A toy ConvNet: X's and O's

Says whether a picture is of an X or an O

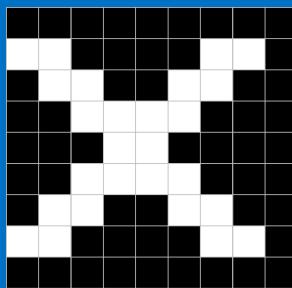
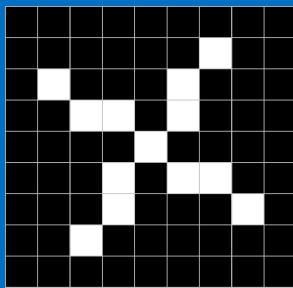
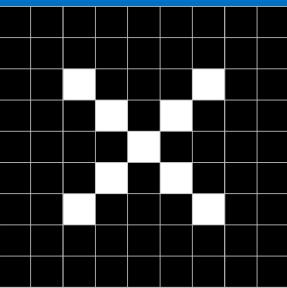
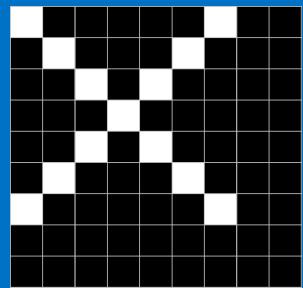
A two-dimensional
array of pixels



For example



Trickier cases

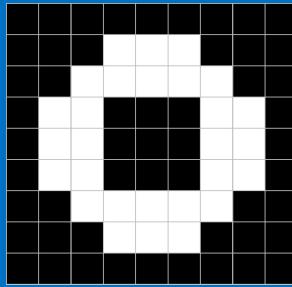
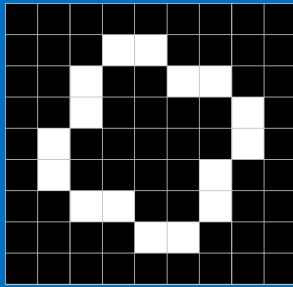
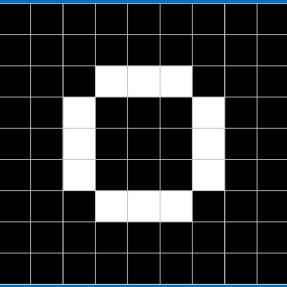
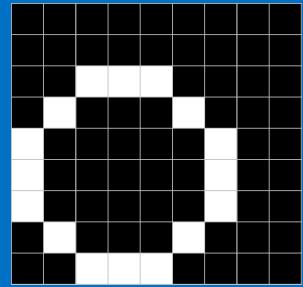


translation

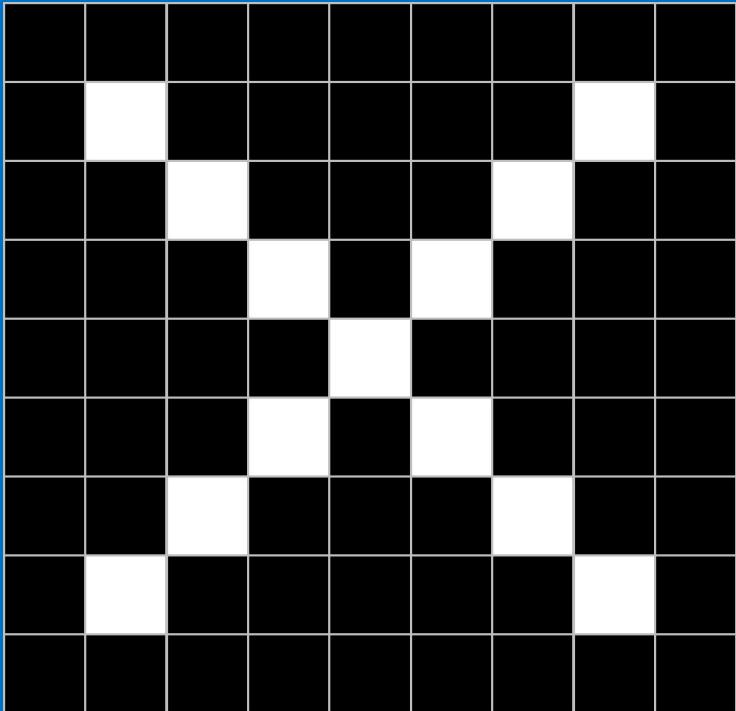
scaling

rotation

weight

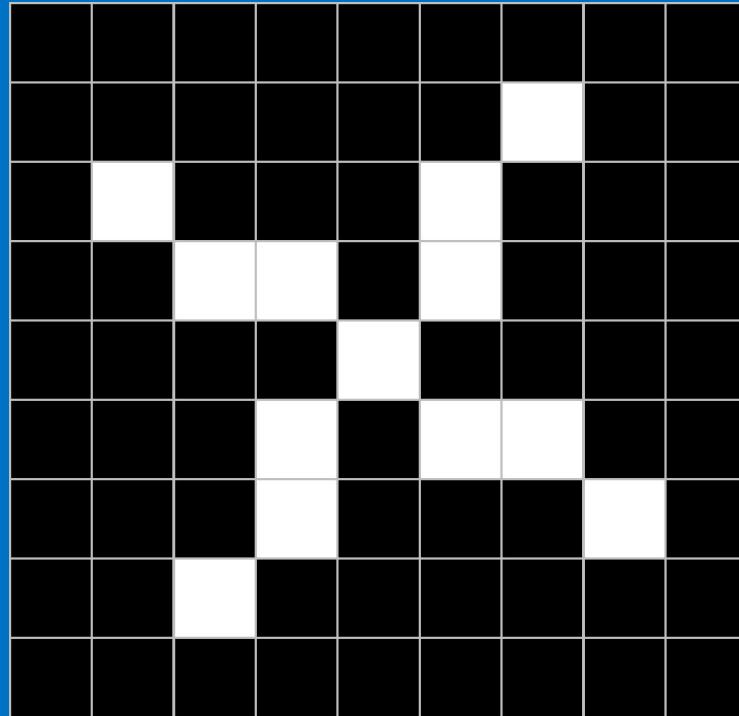


Deciding is hard



?

—
—
—

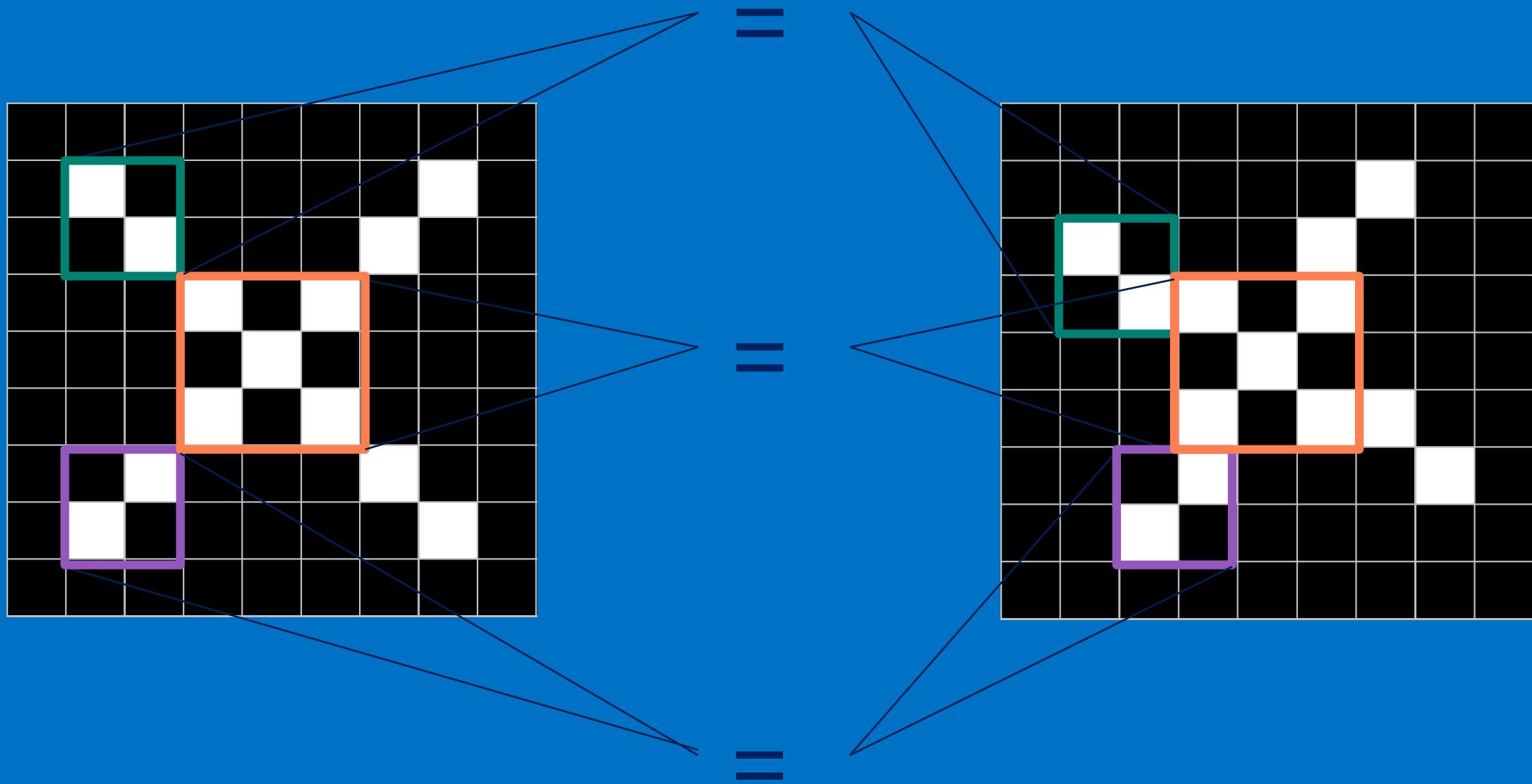


What computers see

What is the best way to approach the question of whether or not to have a child?

What computers see

ConvNets match pieces of the image



Features match pieces of the image

1	-1	-1
-1	1	-1
-1	-1	1

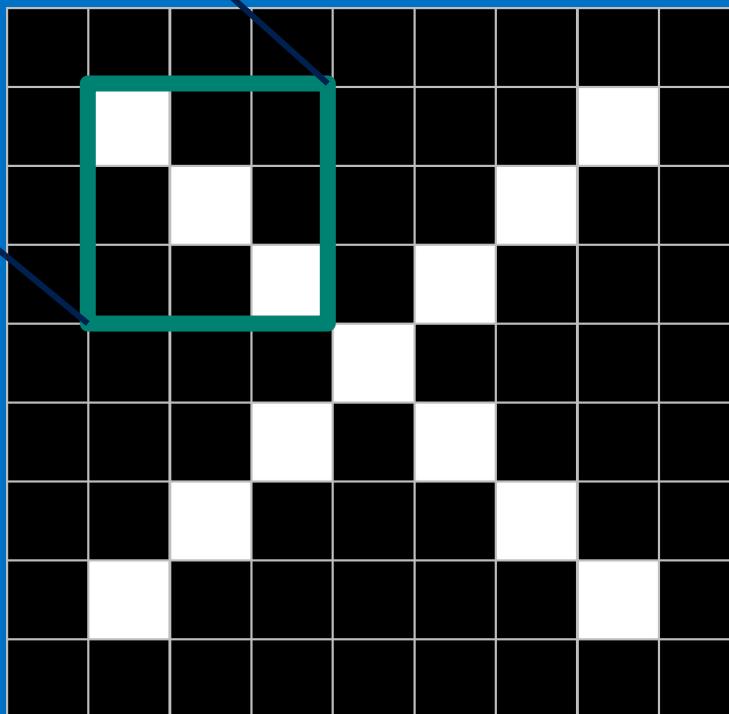
1	-1	1
-1	1	-1
1	-1	1

-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

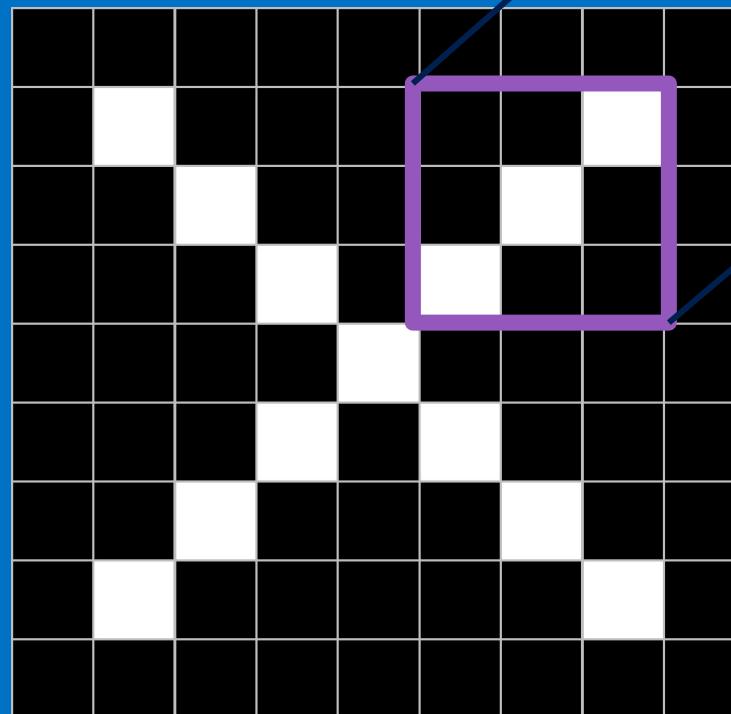
-1	-1	1
-1	1	-1
1	-1	-1



1	-1	-1
-1	1	-1
-1	-1	1

1	-1	1
-1	1	-1
1	-1	1

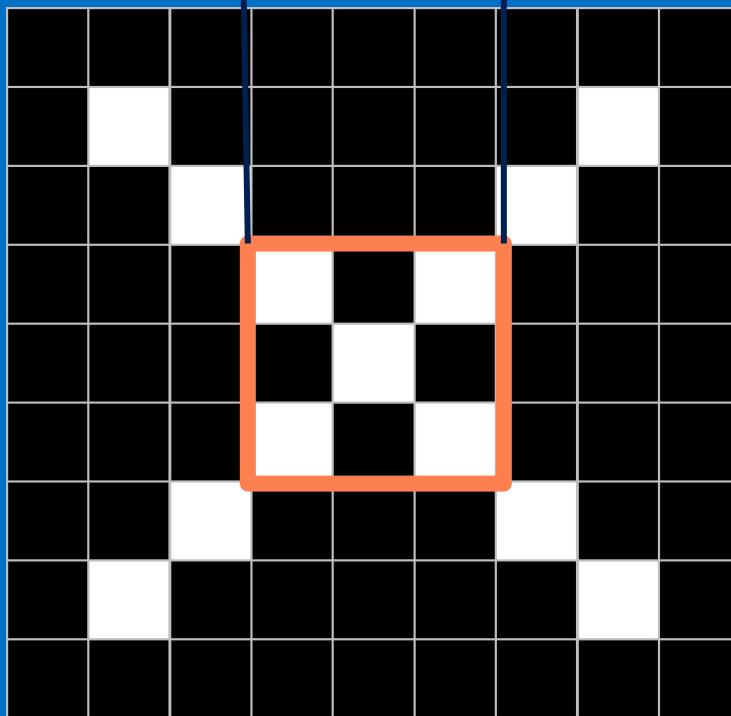
-1	-1	1
-1	1	-1
1	-1	-1



1	-1	-1
-1	1	-1
-1	-1	1

1	-1	1
-1	1	-1
1	-1	1

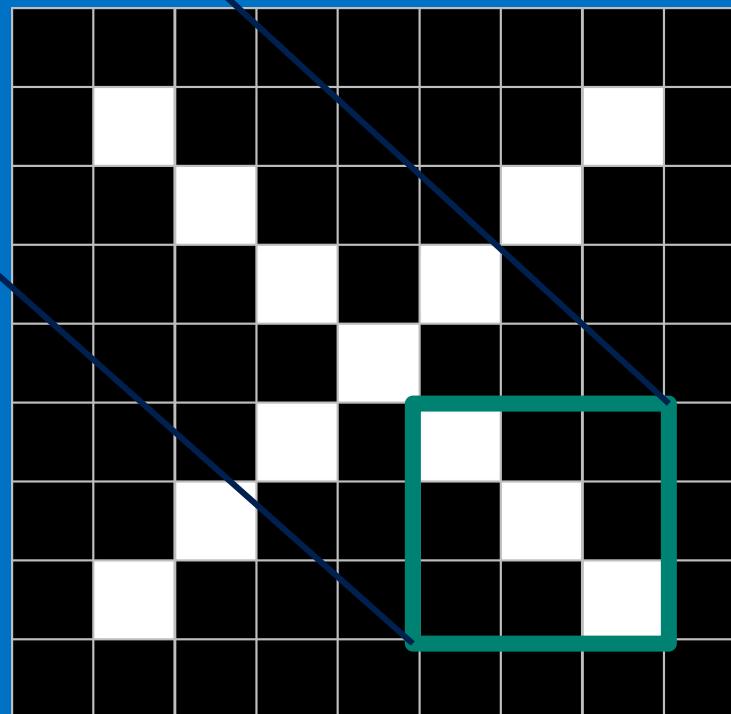
-1	-1	1
-1	1	-1
1	-1	-1



1	-1	-1
-1	1	-1
-1	-1	1

1	-1	1
-1	1	-1
1	-1	1

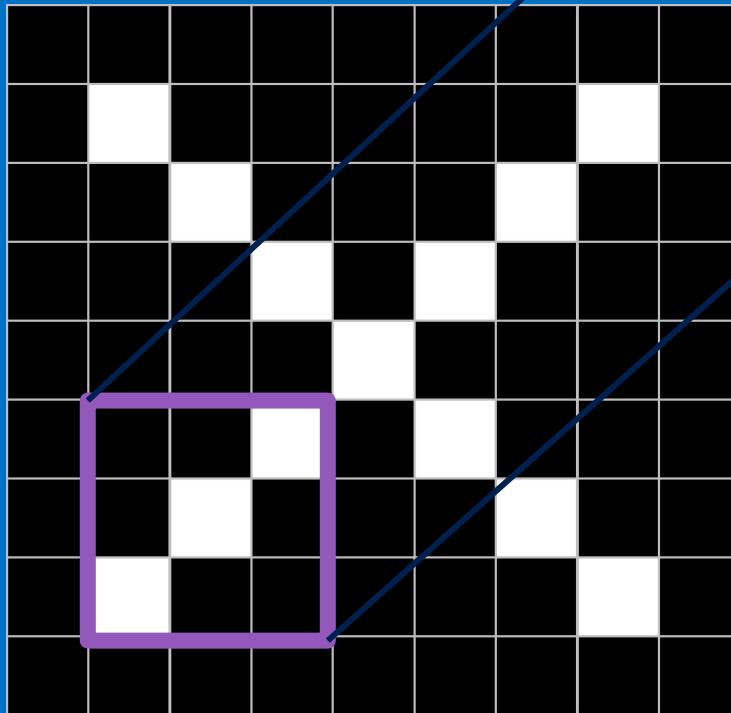
-1	-1	1
-1	1	-1
1	-1	-1



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



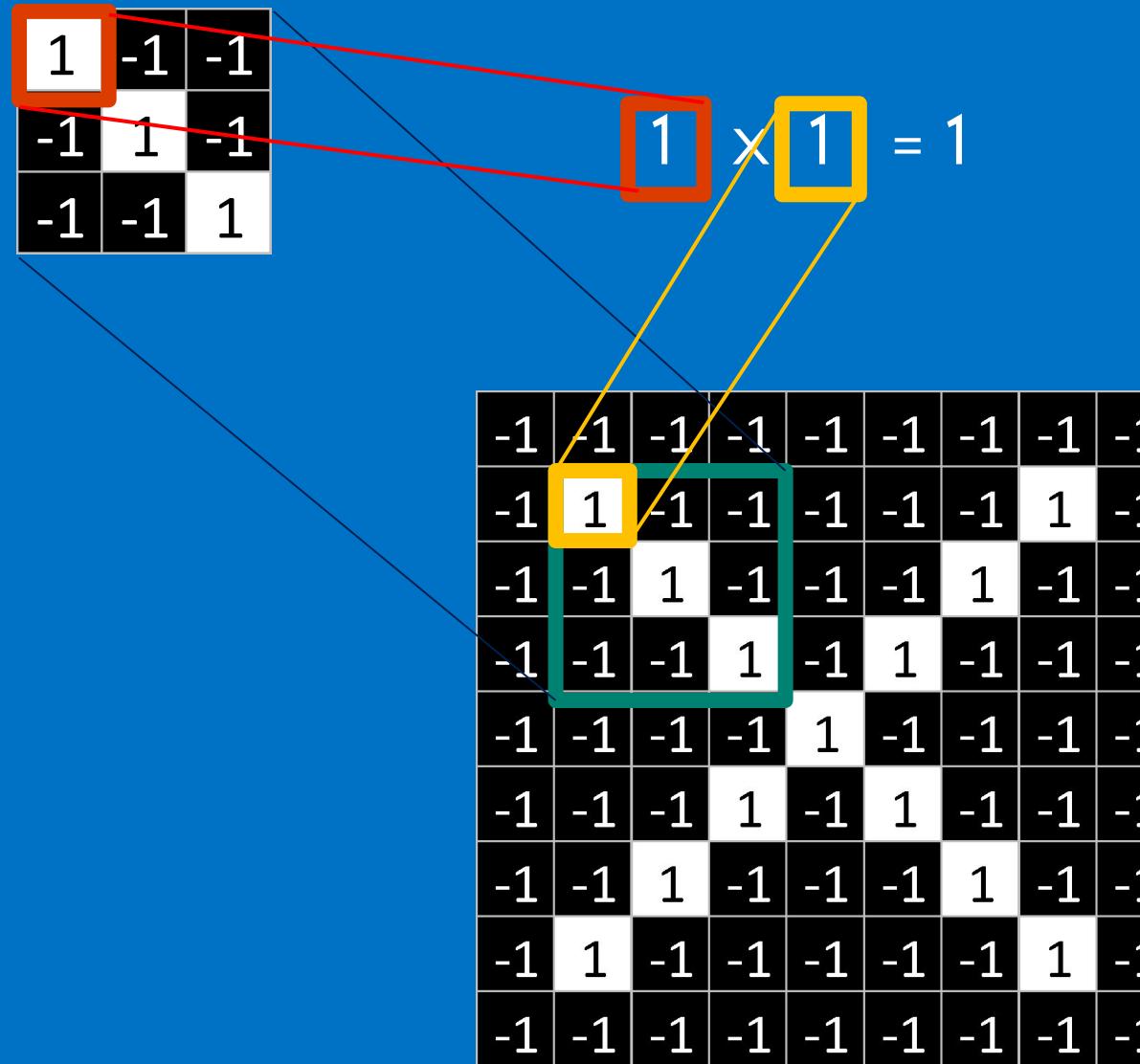
Filtering: The math behind the match

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

$$1 \times 1 = 1$$

-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



Filtering: The math behind the match

1	-1	-1
-1	1	-1
-1	-1	1

$$-1 \times -1 = 1$$

-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	

Filtering: The math behind the match

1	-1	-1
-1	1	-1
-1	-1	1

$$-1 \times -1 = 1$$

-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

Filtering: The math behind the match

1	-1	-1
-1	1	-1
-1	-1	1

$$-1 \times -1 = 1$$

-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		

Filtering: The math behind the match

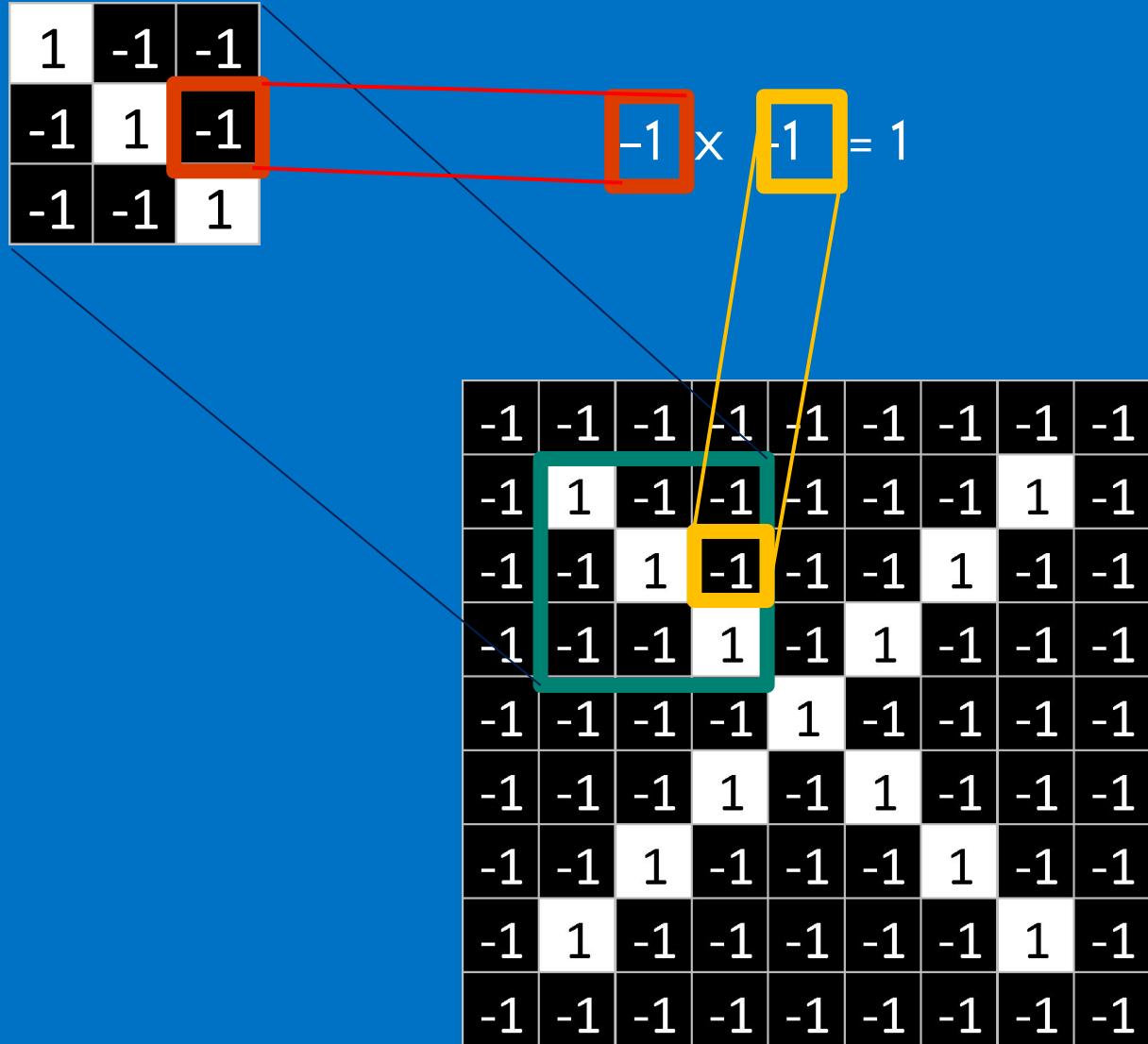
1	-1	-1
-1	1	-1
-1	-1	1

$$1 \times 1 = 1$$

-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	

Filtering: The math behind the match



Filtering: The math behind the match

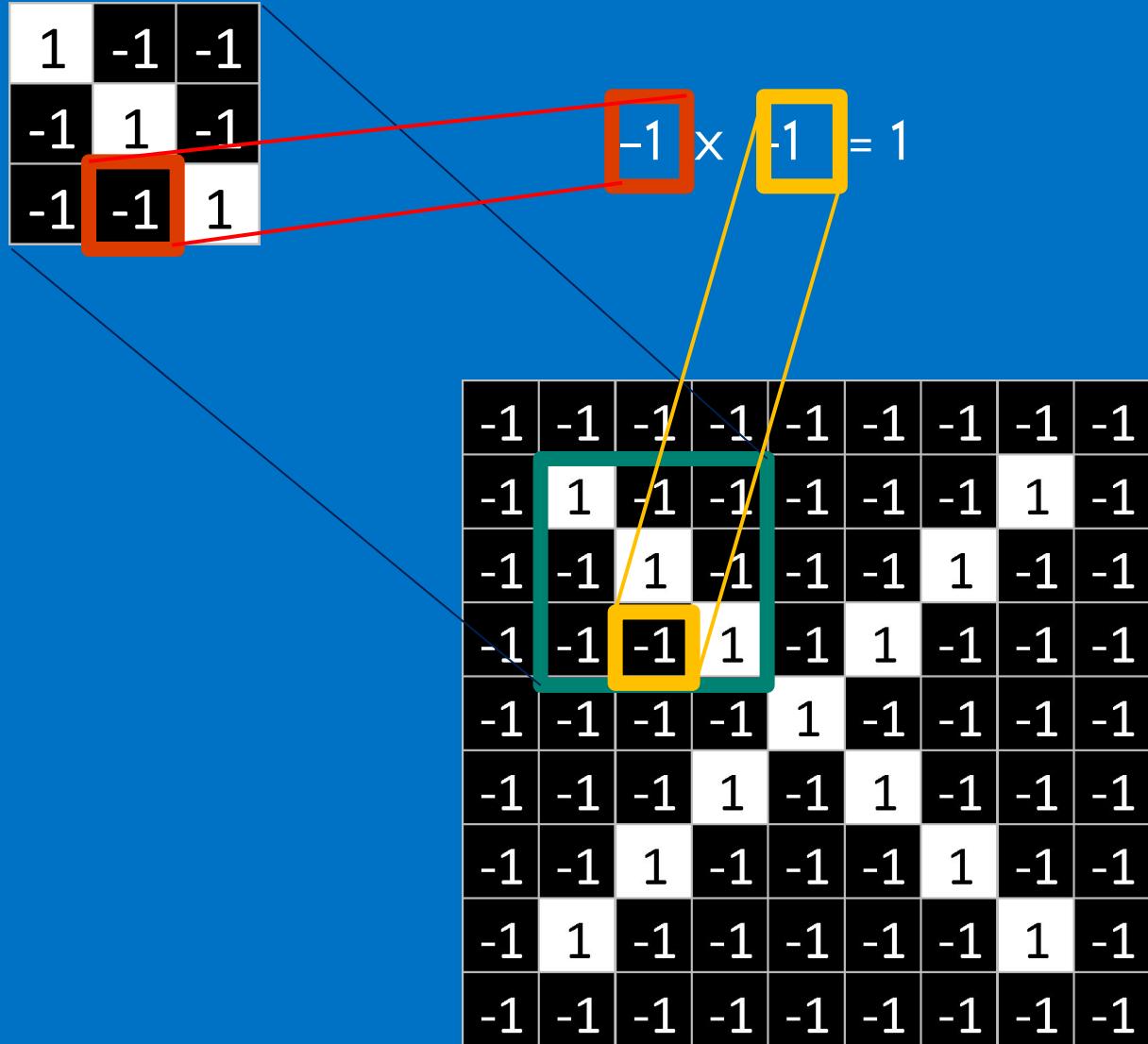
1	-1	-1
-1	1	-1
-1	-1	1

$$-1 \times -1 = 1$$

-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

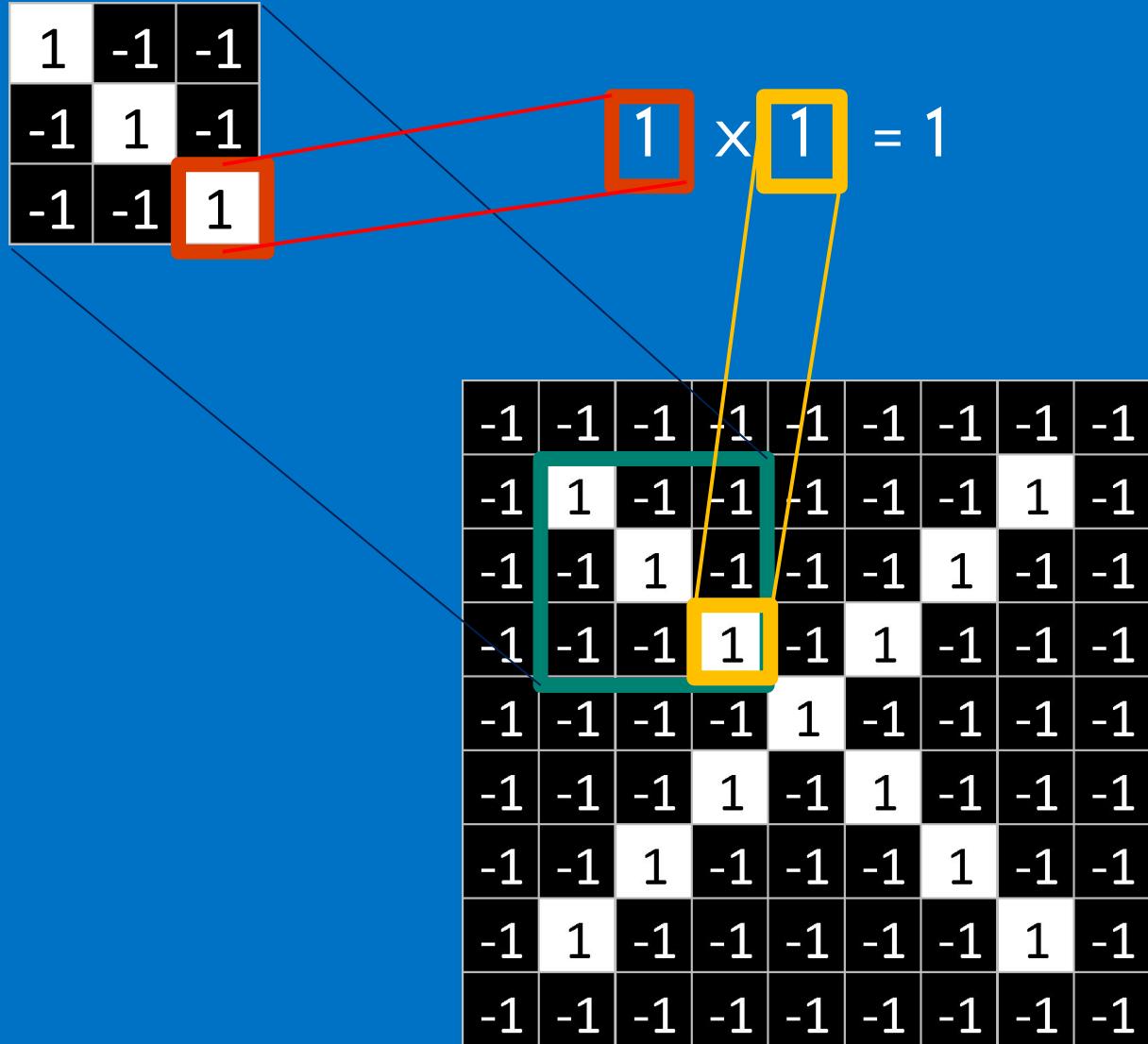
1	1	1
1	1	1
1		

Filtering: The math behind the match



1	1	1
1	1	1
1	1	

Filtering: The math behind the match



1	1	1
1	1	1
1	1	1

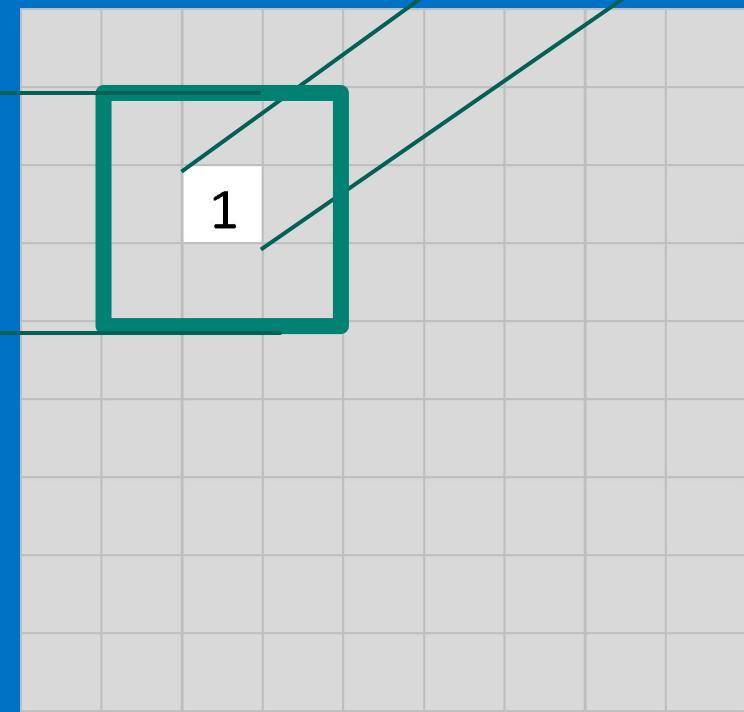
Filtering: The math behind the match

1	-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



Filtering: The math behind the match

1	-1	-1
-1	1	-1
-1	-1	1

$$1 \times 1 = 1$$

-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



Filtering: The math behind the match

1	-1	-1
-1	1	-1
-1	-1	1

$$-1 \times 1 = -1$$

-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

Filtering: The math behind the 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

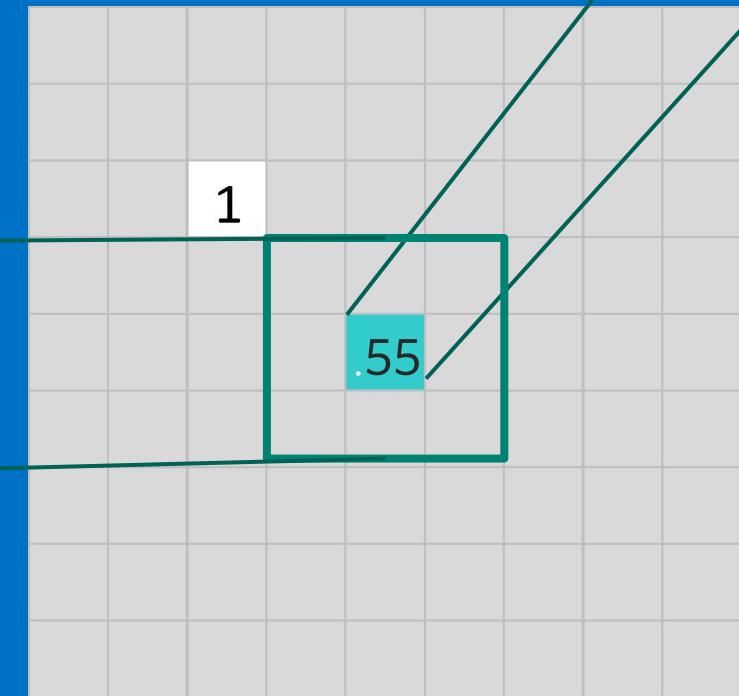
Filtering: The math behind the match

1	-1	-1
-1	1	-1
-1	-1	1

1	1	-1
1	1	1
-1	1	1

$$\frac{1 + 1 - 1 + 1 + 1 + 1 - 1 + 1 + 1}{9} = .55$$

-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	1	-1	-1	-1	-1	-1	-1	-1
-1	-1	1	-1	-1	-1	1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	-1	-1	-1	1	-1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	1	-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

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

=

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	-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	1

=

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	-1	



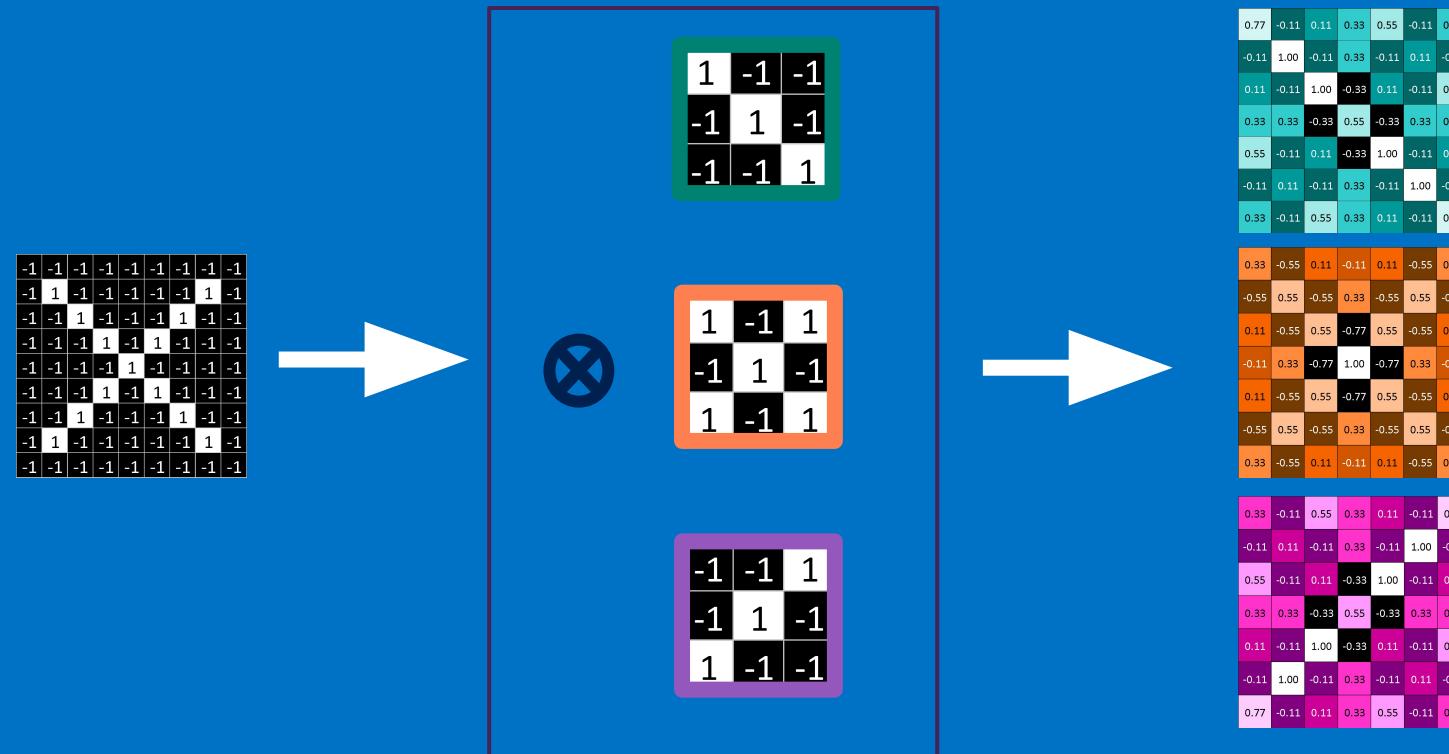
-1	-1	1
-1	1	-1
1	-1	-1

=

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

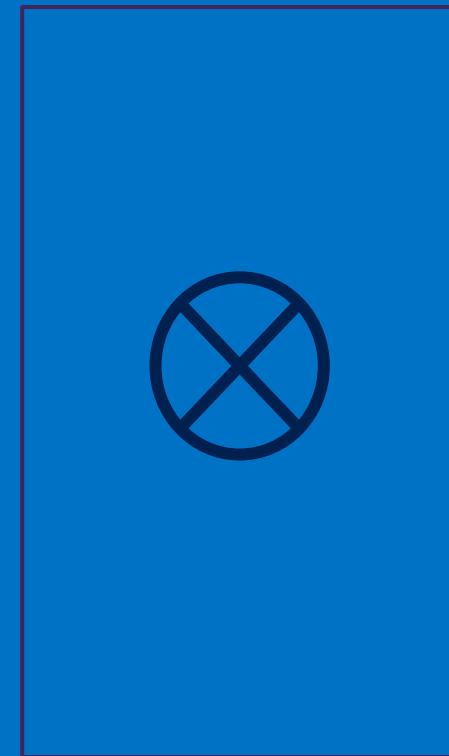
One image becomes a stack of filtered images



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	-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



Pooling

maximum

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		

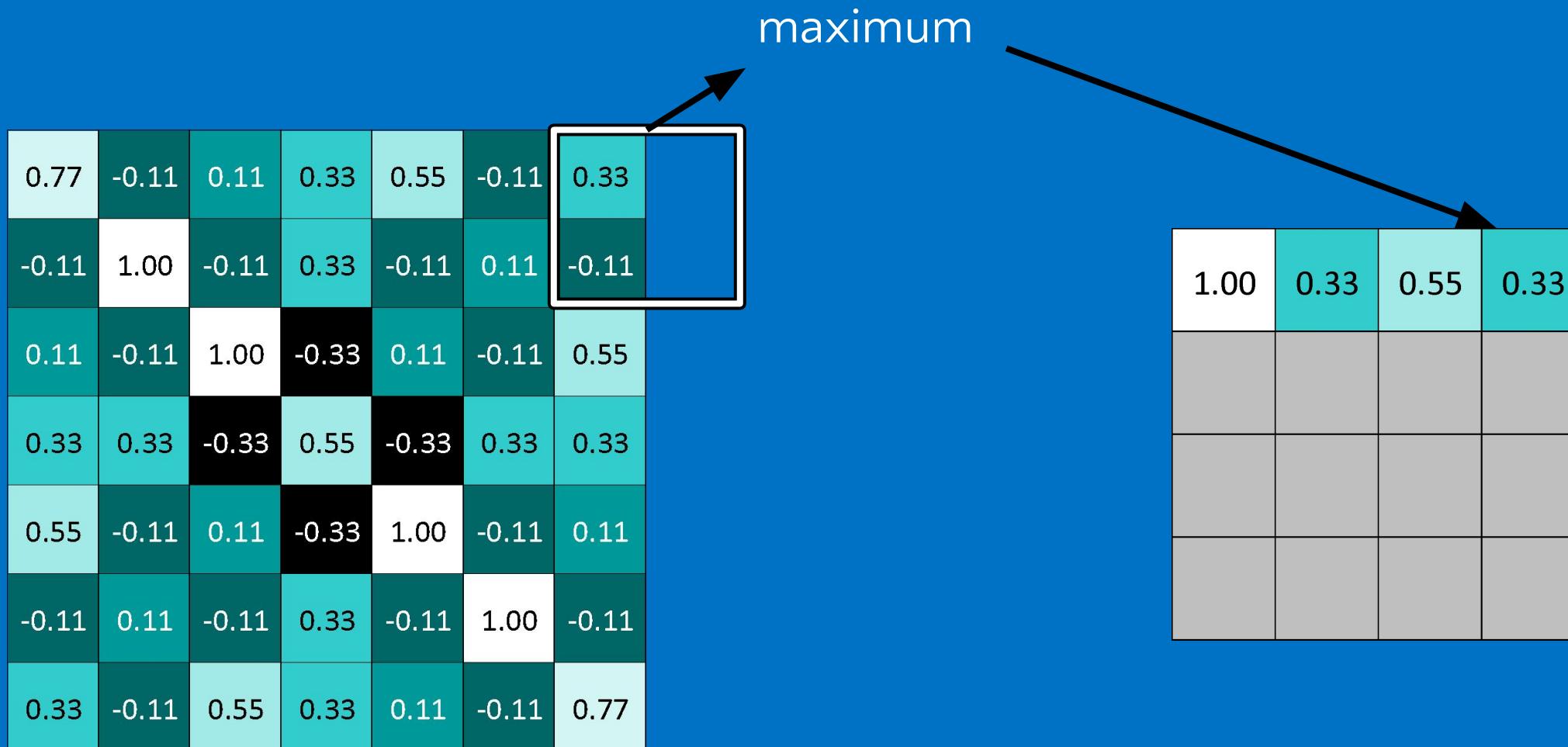
Pooling

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	0.33	0.55	

Pooling



Pooling

maximum

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			

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

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

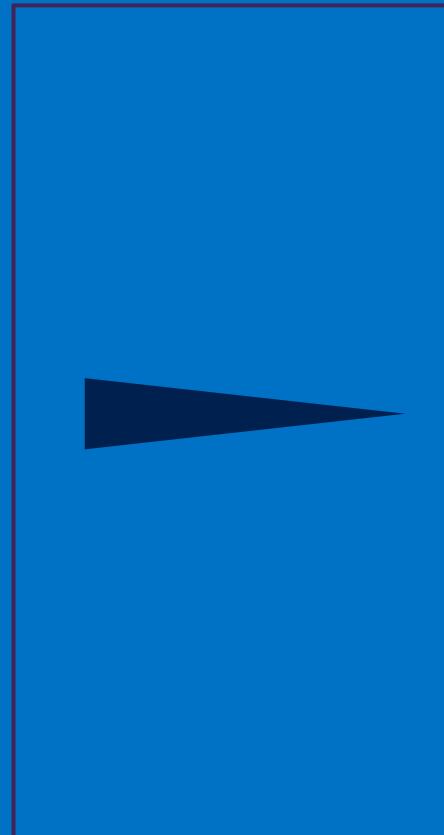
Pooling layer

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

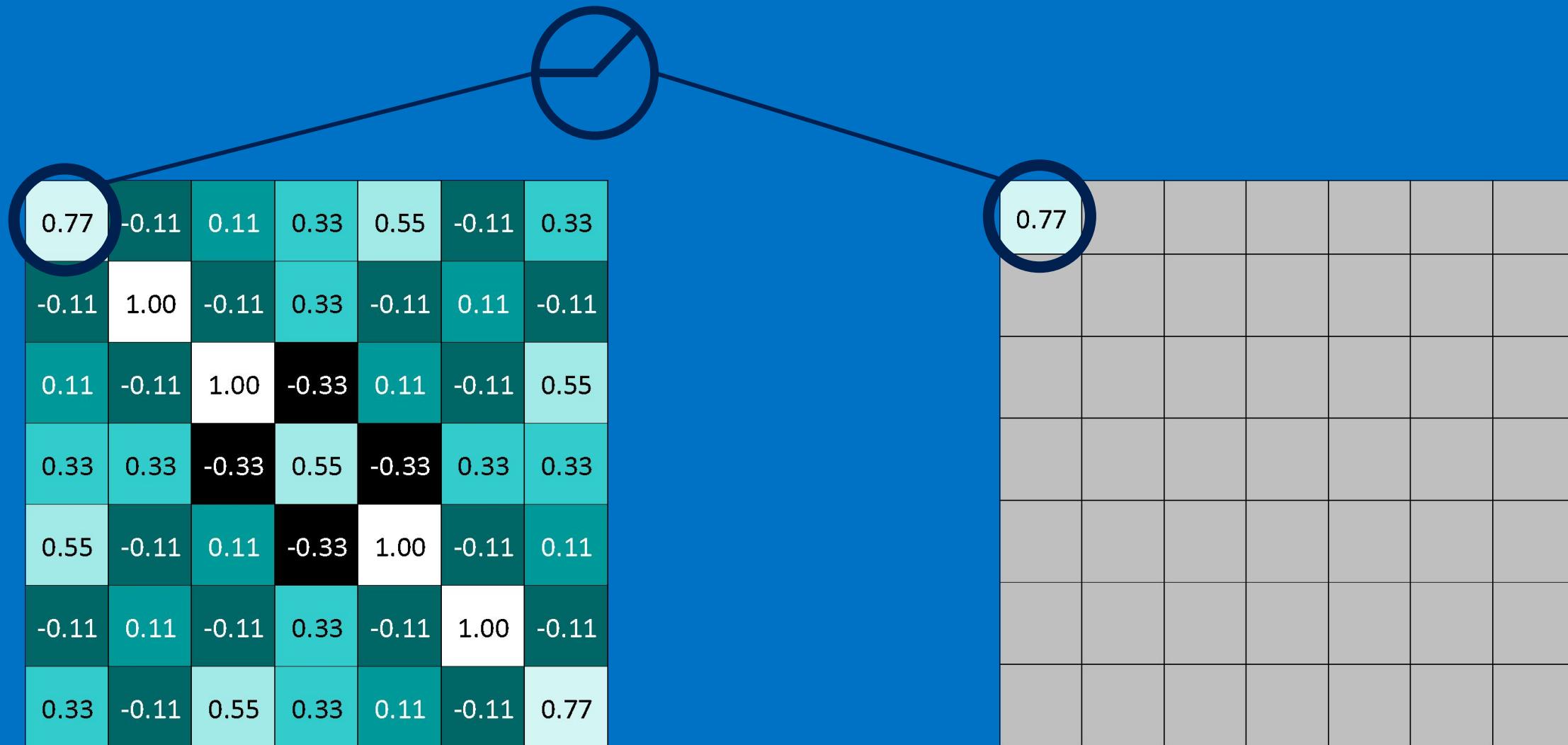
0.33	0.55	1.00	0.77
0.55	0.55	1.00	0.33
1.00	1.00	0.11	0.55
0.77	0.33	0.55	0.33

Normalization

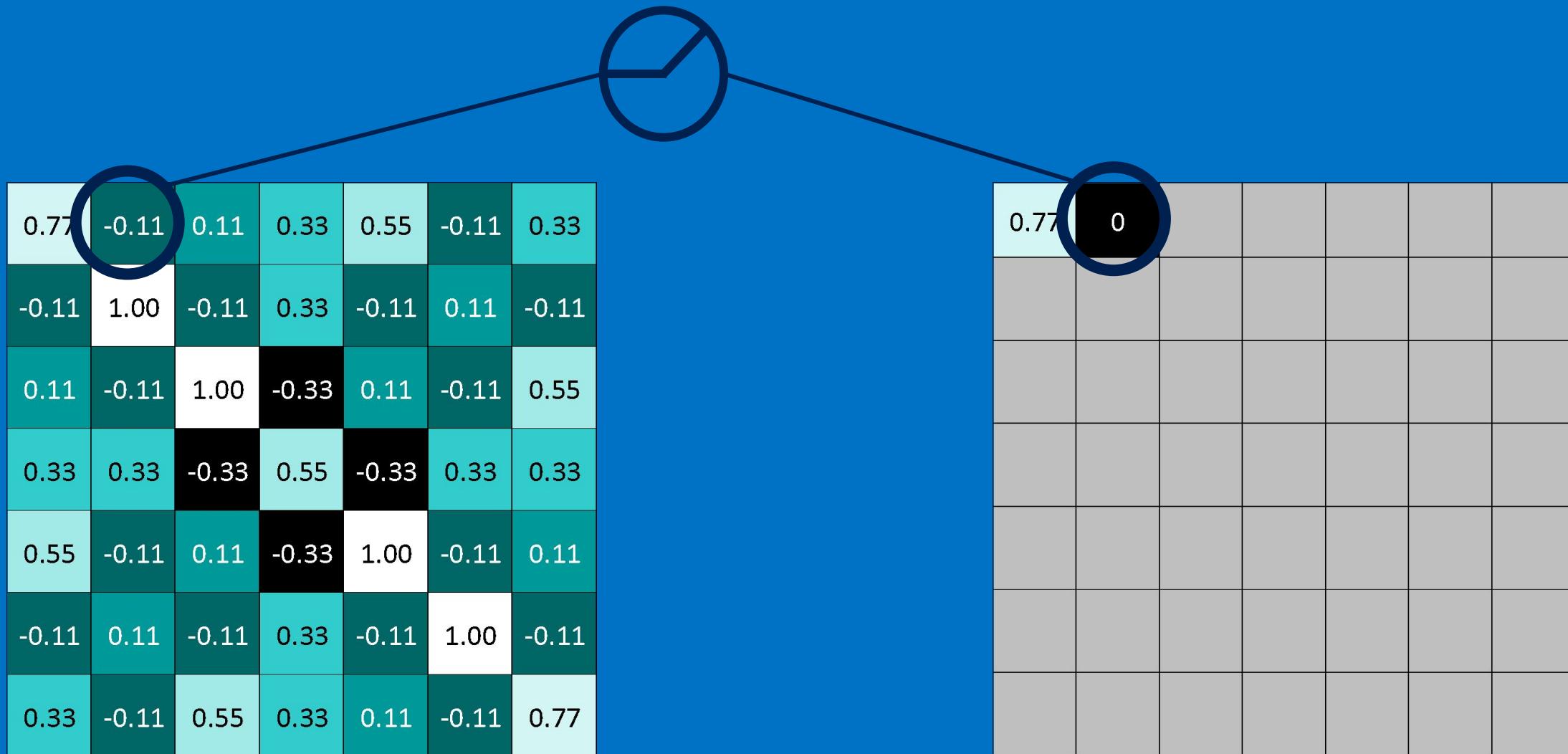
Keep the math from breaking by tweaking each of the values just a bit.

Change everything negative to zero.

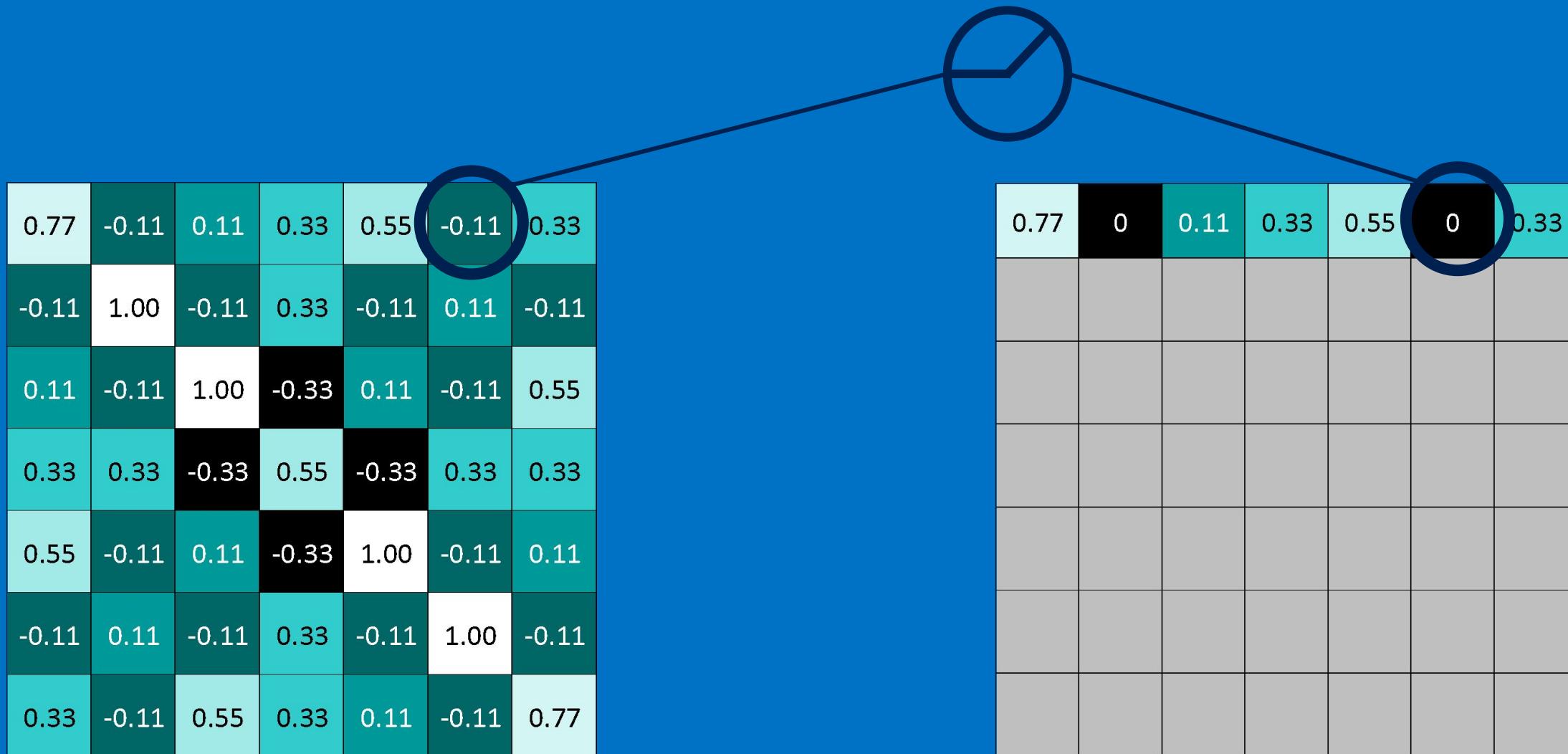
Rectified Linear Units (ReLUs)



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
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

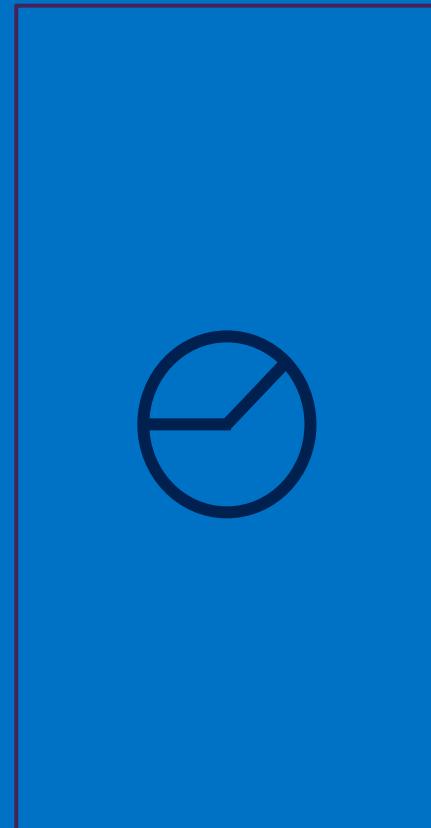
ReLU layer

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

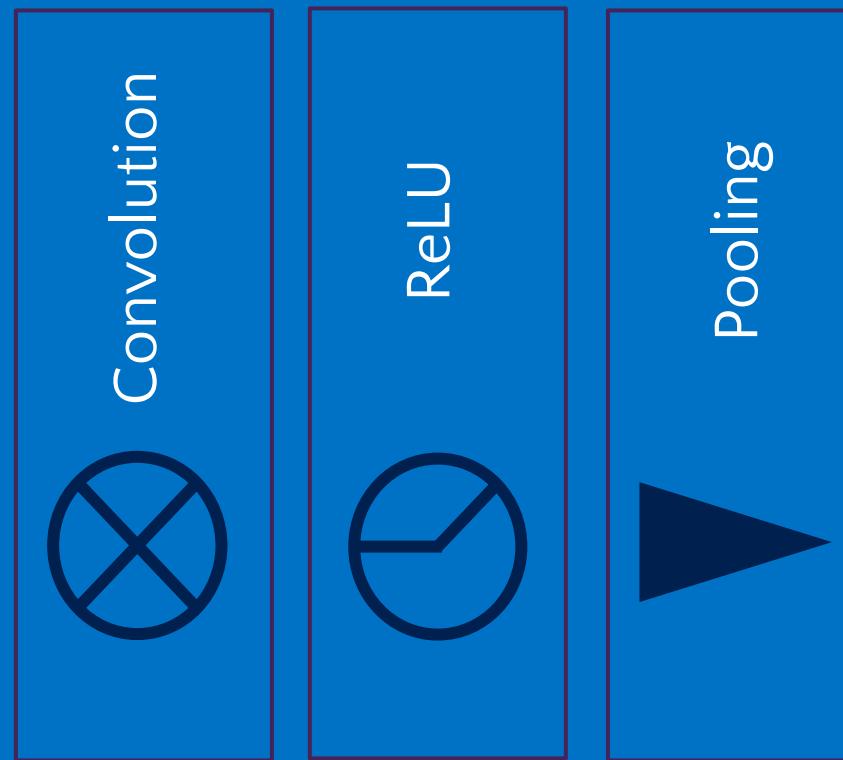
0.33	0	0.11	0	0.11	0	0.33
0	0.55	0	0.33	0	0.55	0
0.11	0	0.55	0	0.55	0	0.11
0	0.33	0	1.00	0	0.33	0
0.11	0	0.55	0	0.55	0	0.11
0	0.55	0	0.33	0	0.55	0
0.33	0	0.11	0	0.11	0	0.33

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	-1	-1	-1	-1	-1	-1	-1	-1	-1



The diagram shows the output of each layer as a 4x4 grid:

- Input:**

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
- Convolution:**

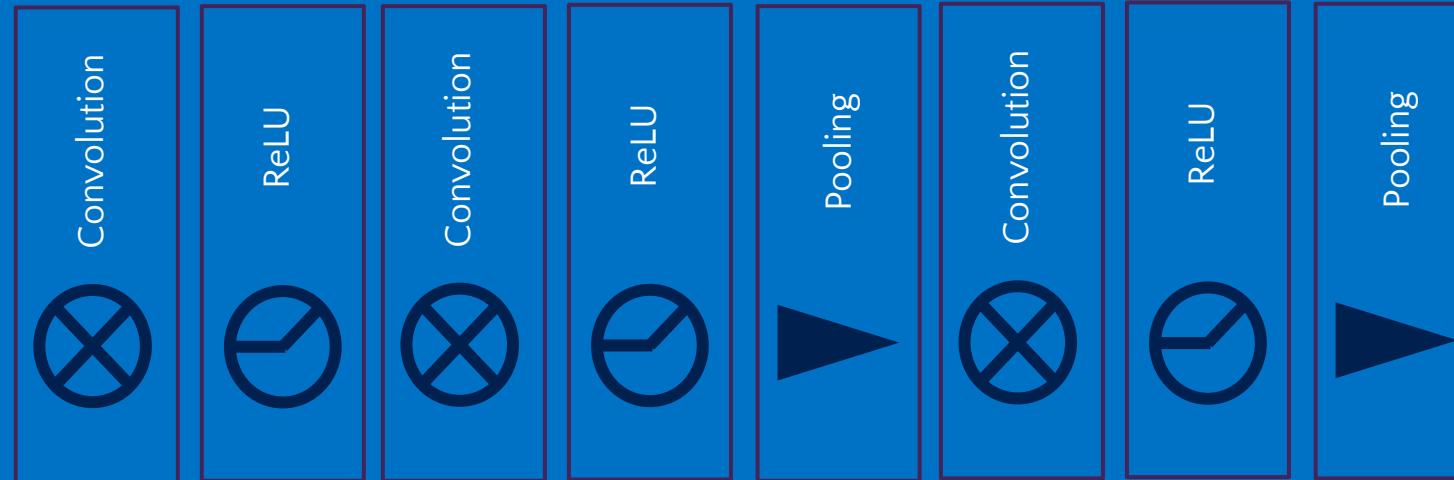
0.55	0.33	0.55	0.33
0.33	1.00	0.55	0.11
0.55	0.55	0.55	0.11
0.33	0.11	0.11	0.33
- Final Output:**

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.

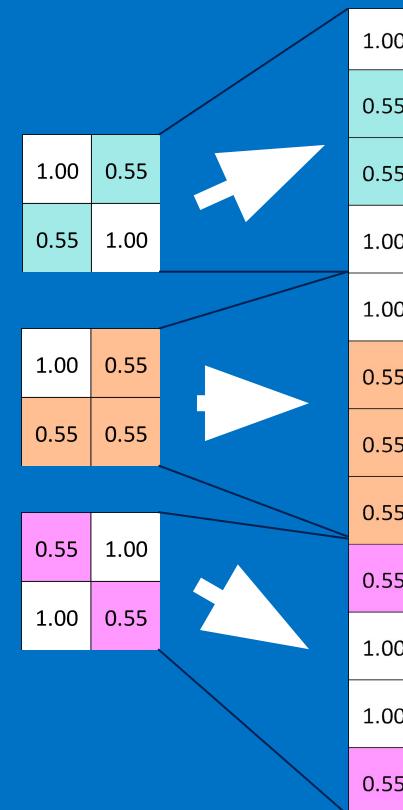
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1	-1
-1	-1	1	-1	-1	-1	1	-1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1	-1
-1	-1	-1	-1	1	-1	-1	-1	-1	-1
-1	-1	-1	-1	1	-1	-1	-1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1	-1
-1	-1	1	-1	-1	-1	1	-1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1



1.00	0.55
0.55	1.00
1.00	0.55
0.55	0.55
0.55	1.00
1.00	0.55

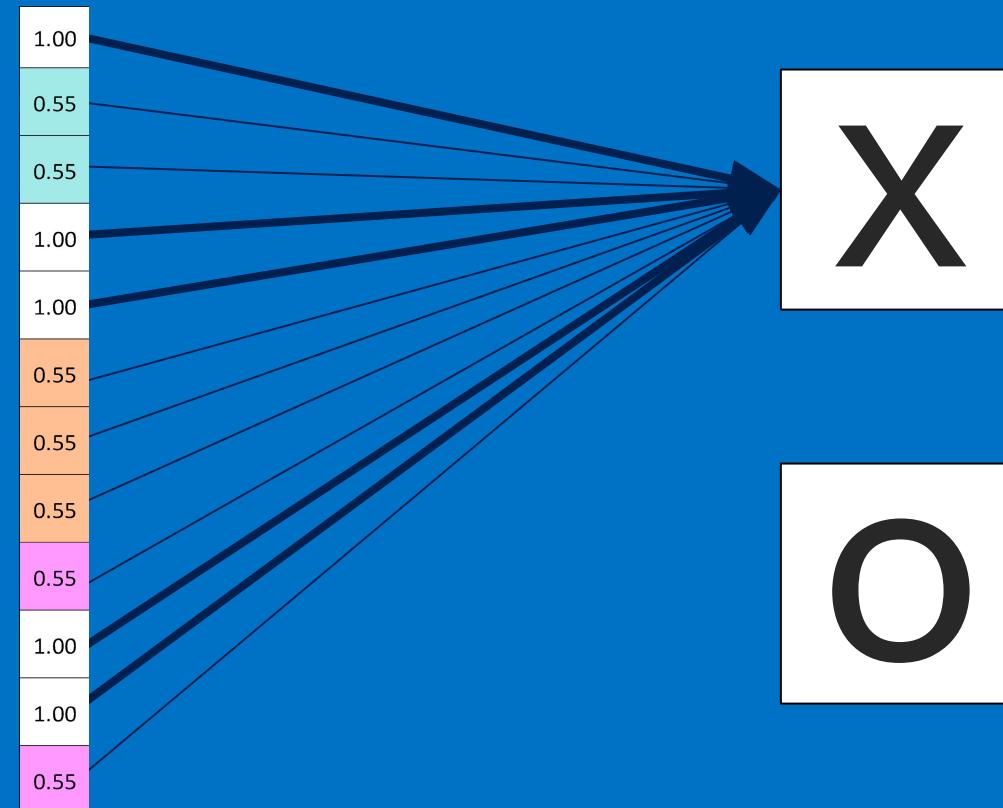
Fully connected layer

Every value gets a vote



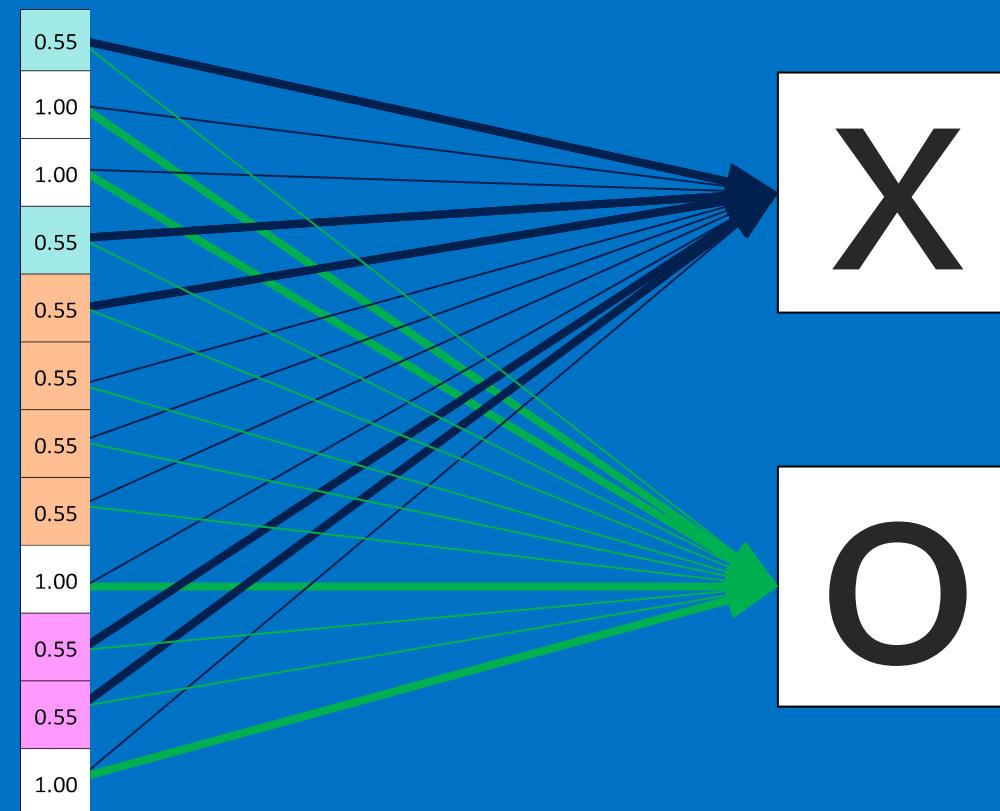
Fully connected layer

Vote depends on how strongly a value predicts X or O



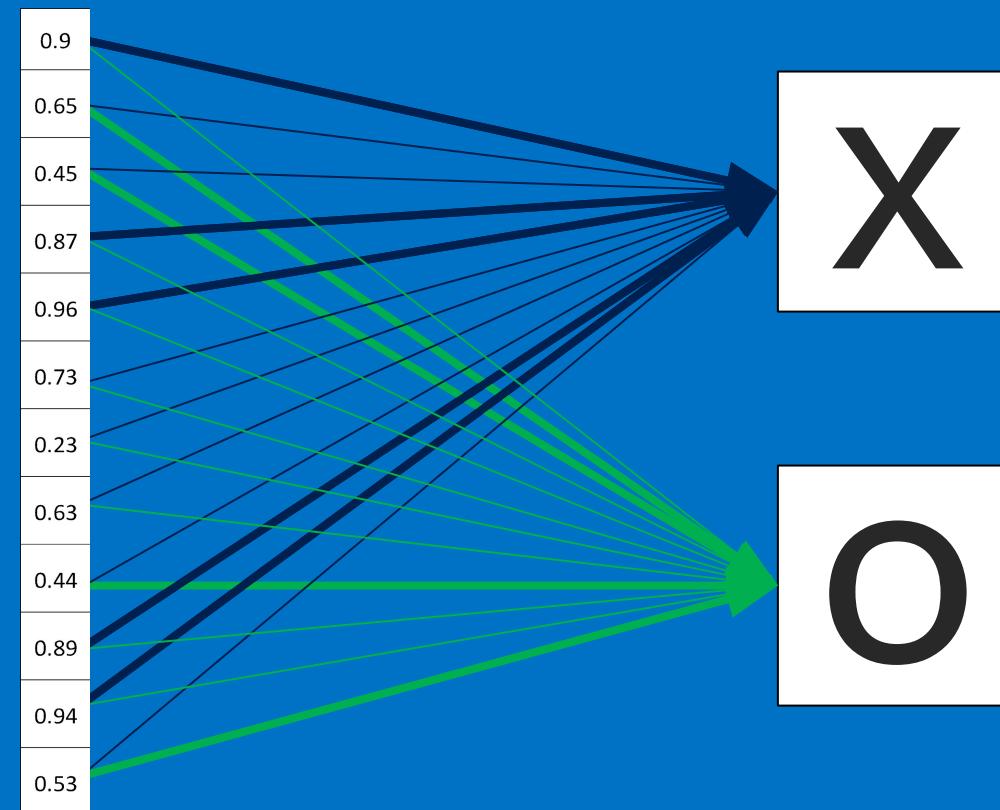
Fully connected layer

Vote depends on how strongly a value predicts X or O



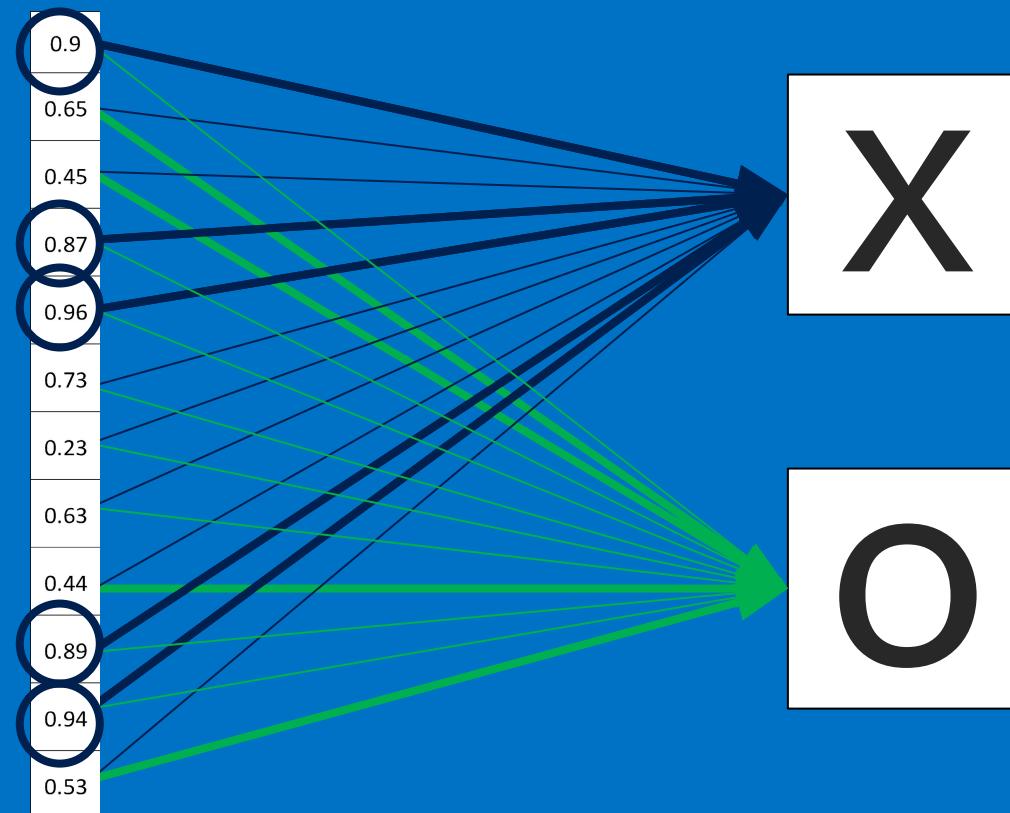
Fully connected layer

Future values vote on X or O



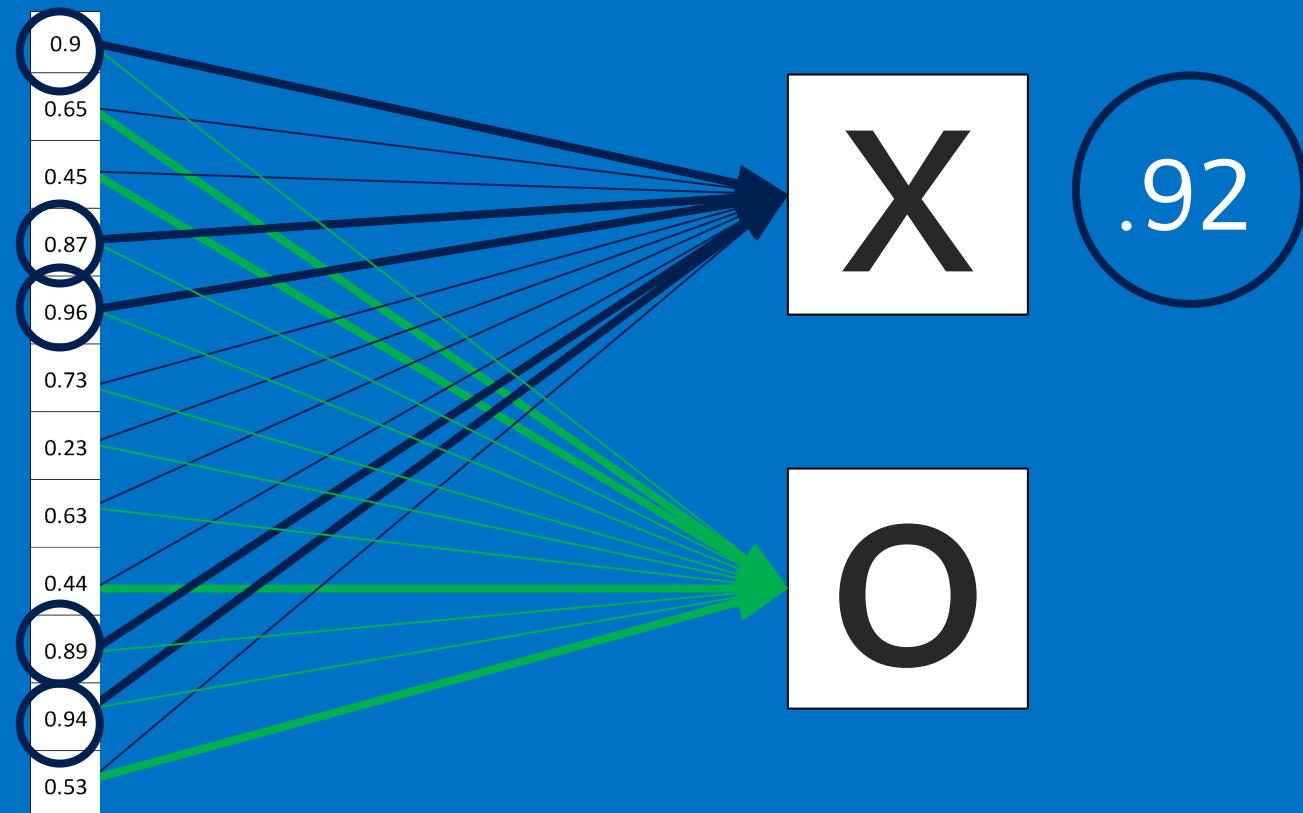
Fully connected layer

Future values vote on X or O



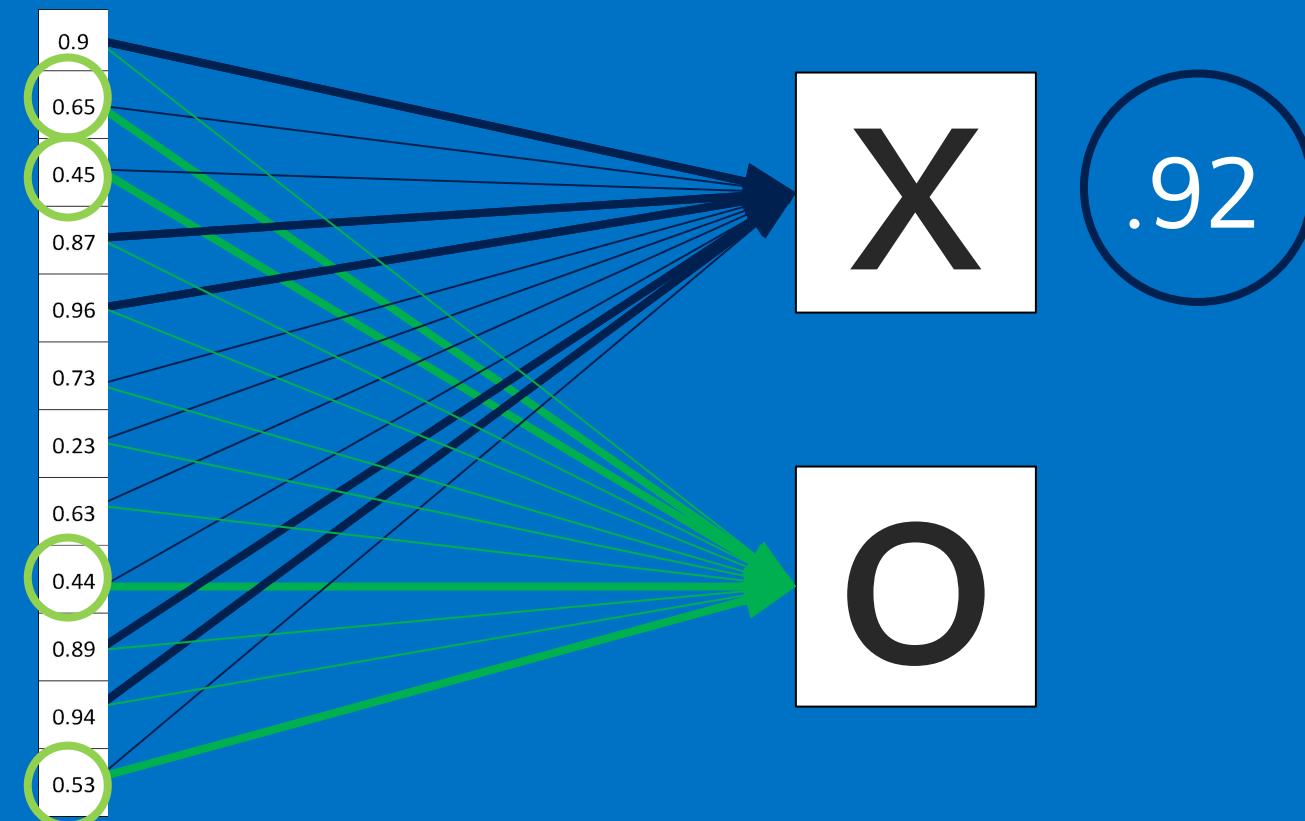
Fully connected layer

Future values vote on X or O



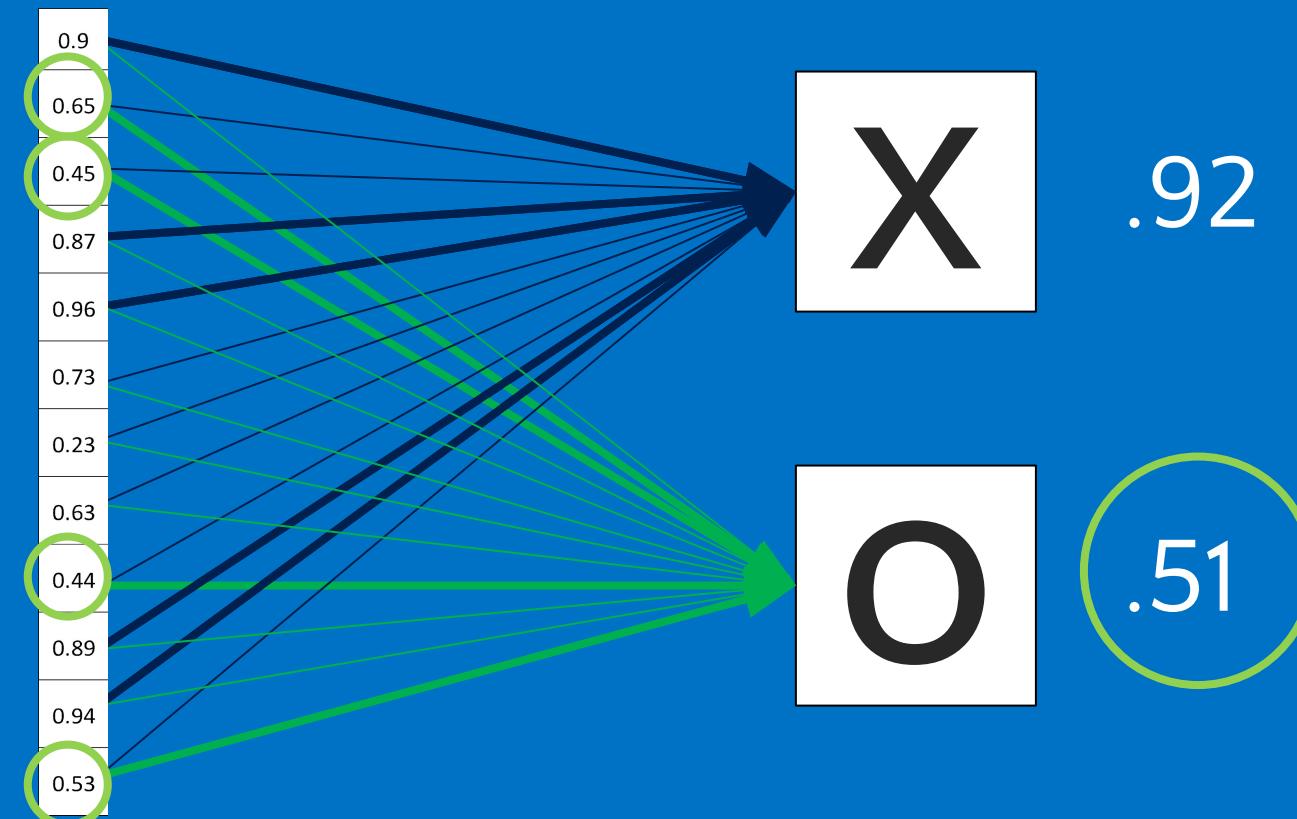
Fully connected layer

Future values vote on X or O



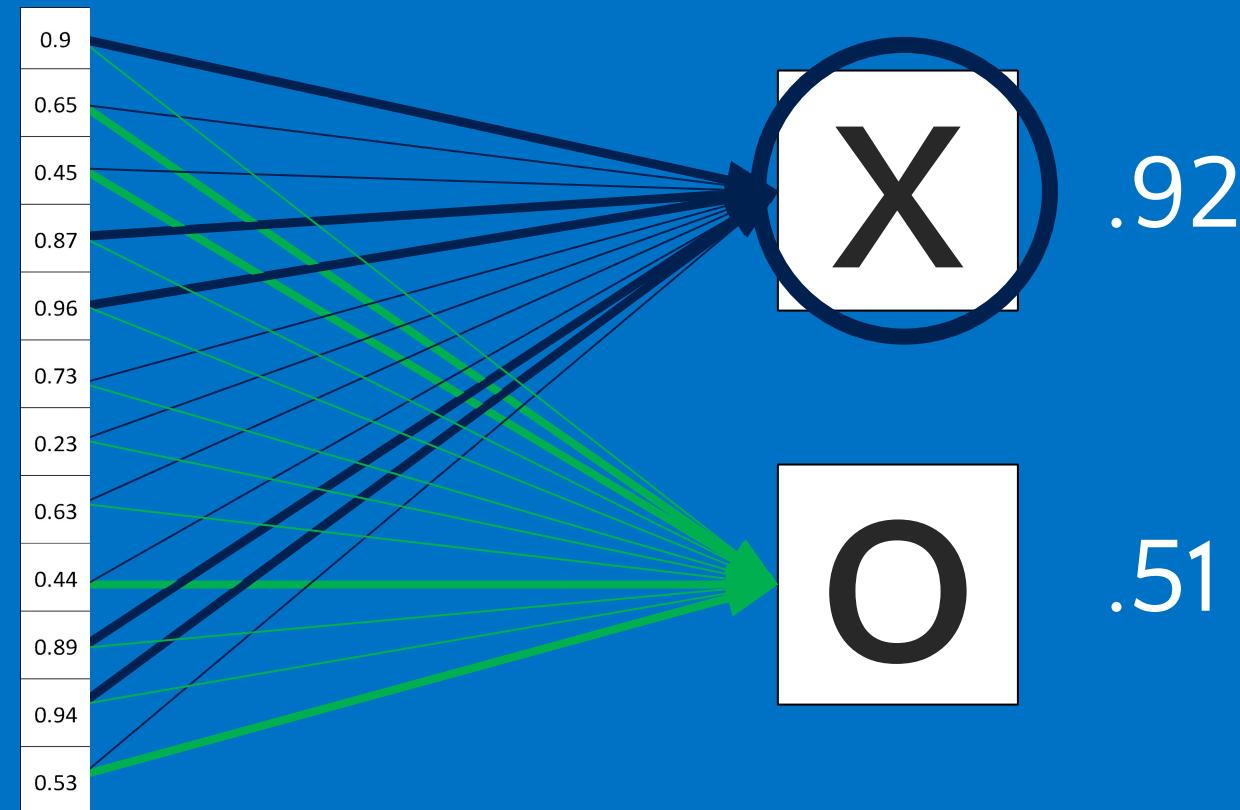
Fully connected layer

Future values vote on X or O



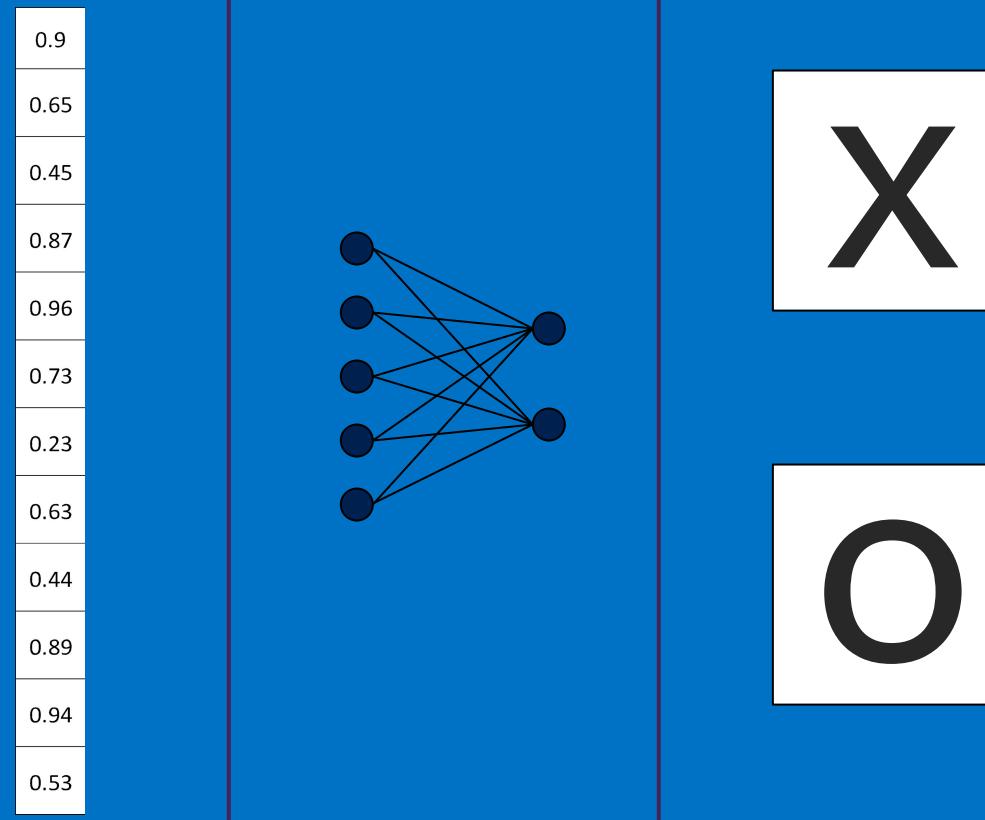
Fully connected layer

Future values vote on X or O



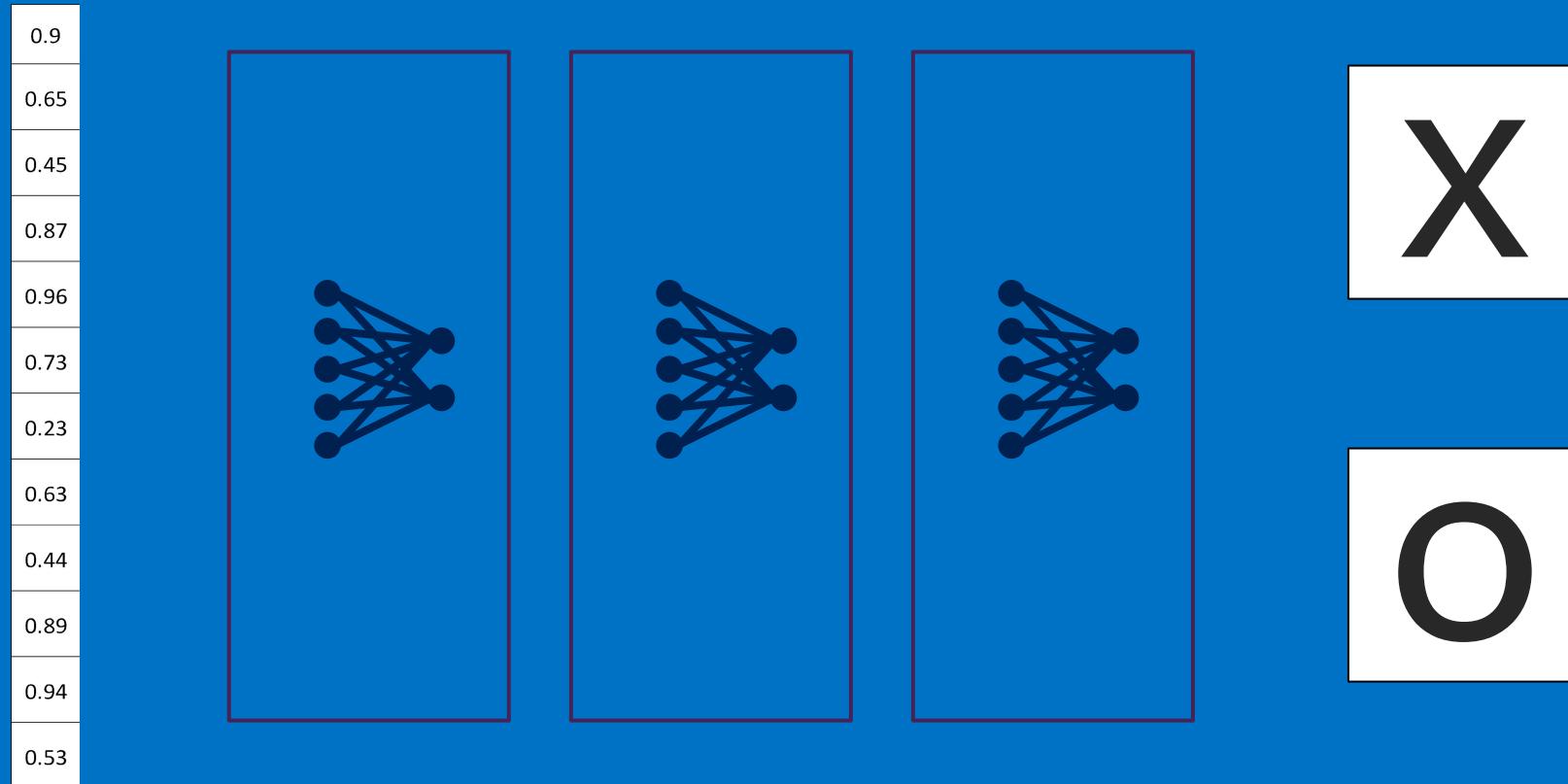
Fully connected layer

A list of feature values becomes a list of votes.



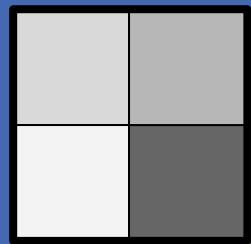
Fully connected layer

These can also be stacked.

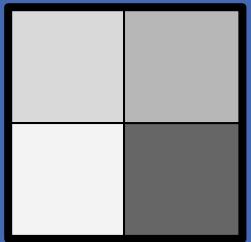


How neural networks work

A four pixel camera



Categorize images



solid



vertical



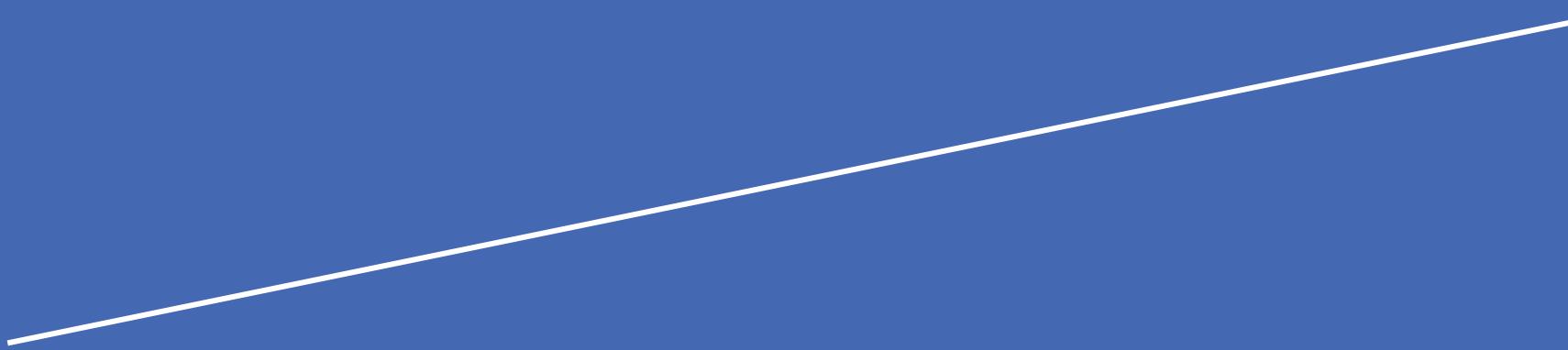
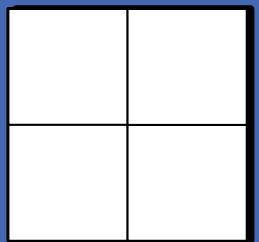
diagonal



horizontal



Categorize images



solid

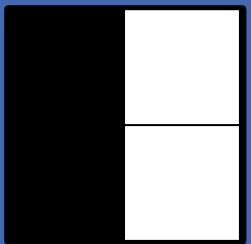
vertical

diagonal

horizontal



Categorize images



solid



vertical



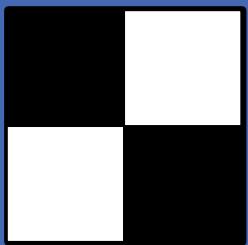
diagonal



horizontal



Categorize images



solid



vertical



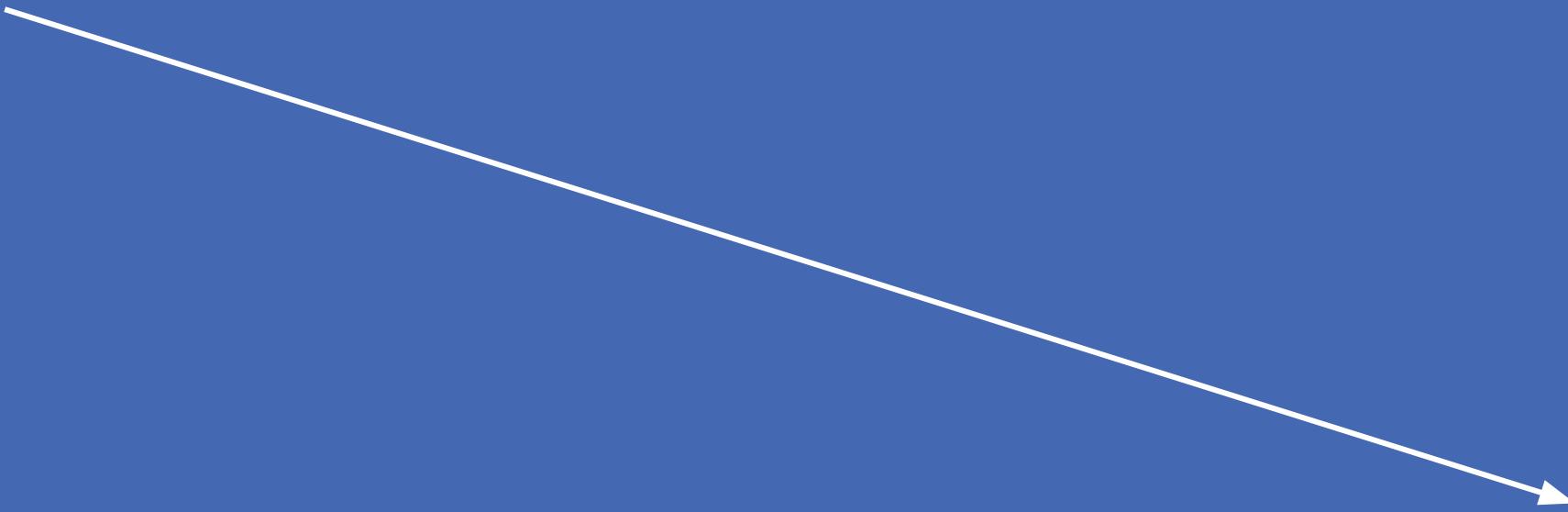
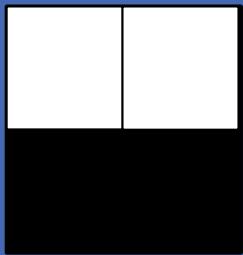
diagonal



horizontal



Simple rules can't do it



solid



vertical



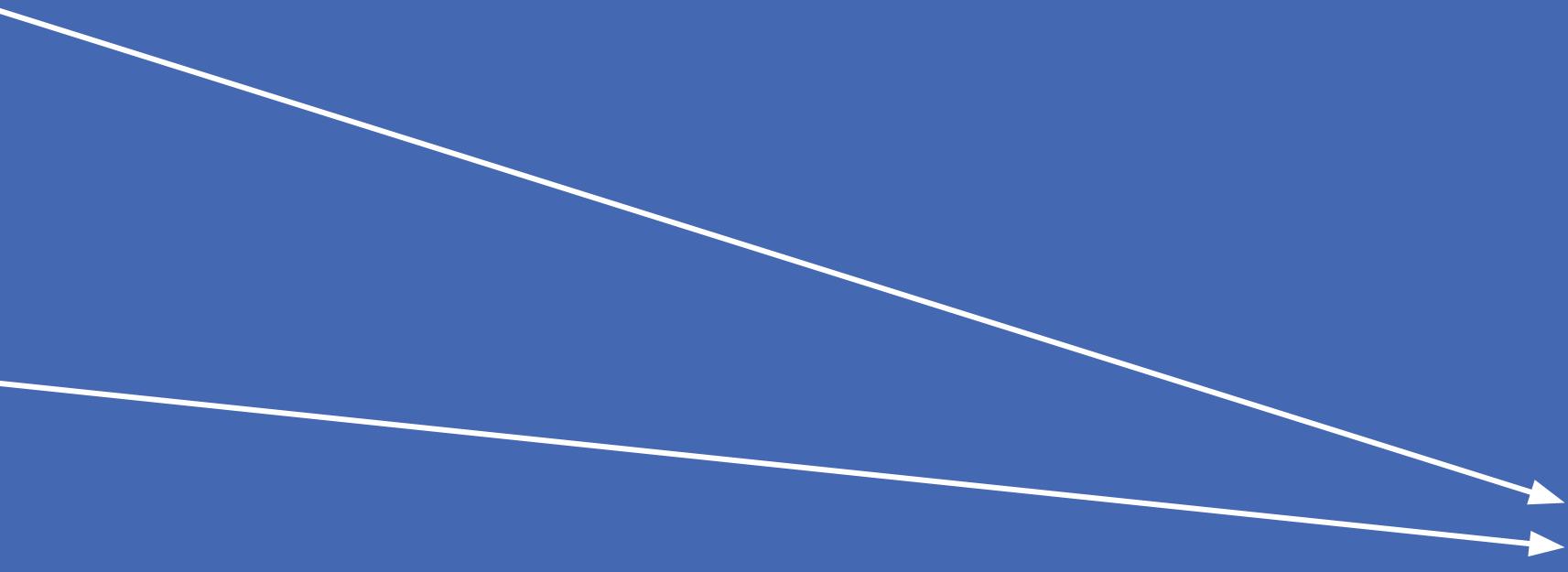
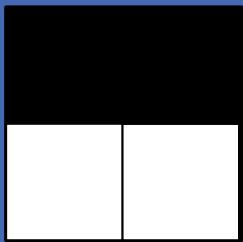
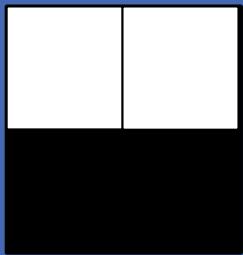
diagonal



horizontal



Simple rules can't do it



solid



vertical



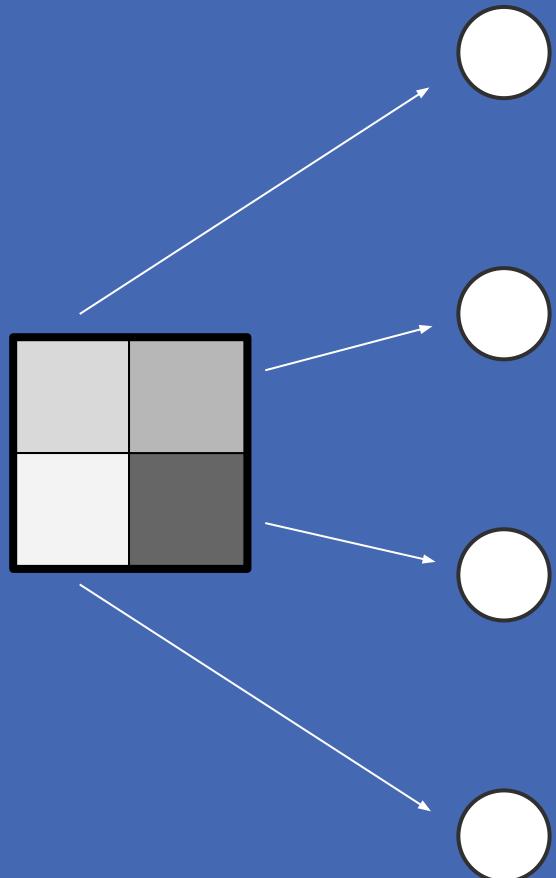
diagonal



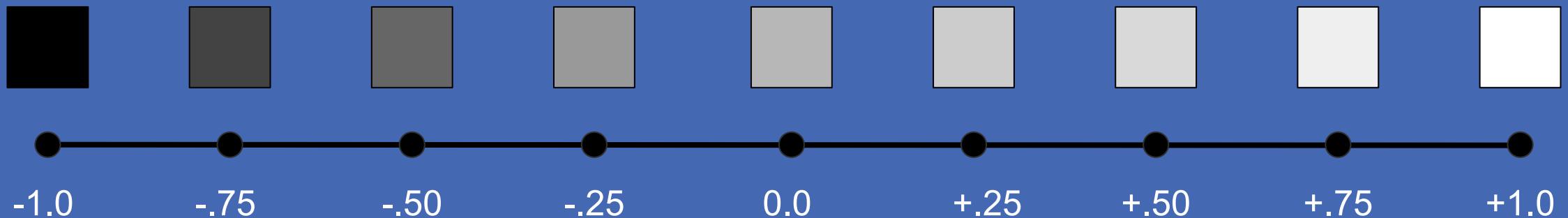
horizontal



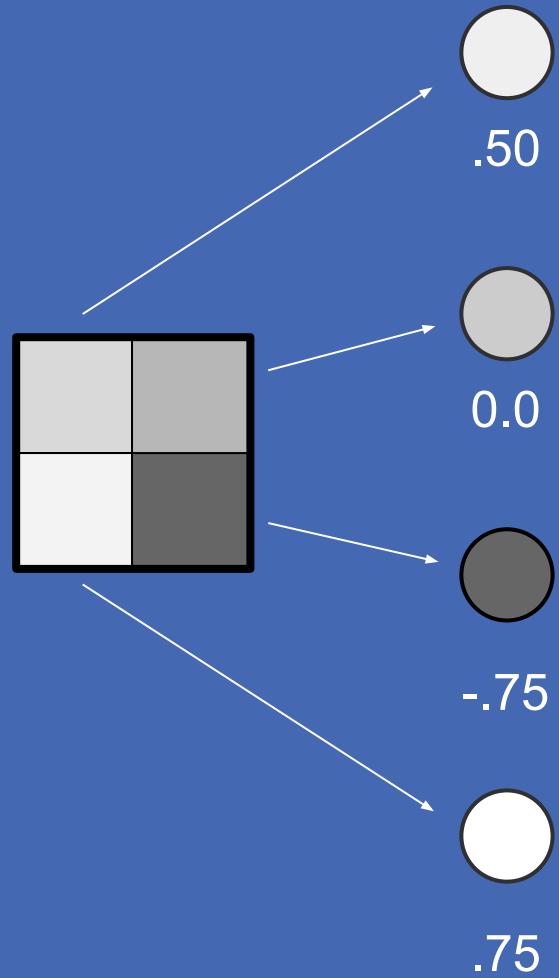
Input neurons



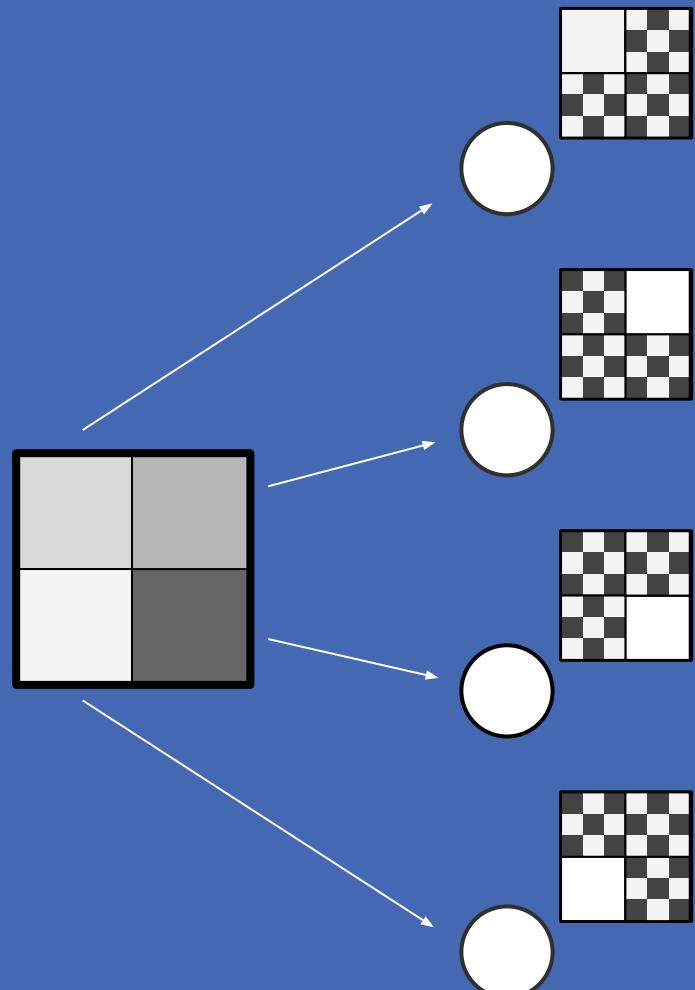
Pixel brightness



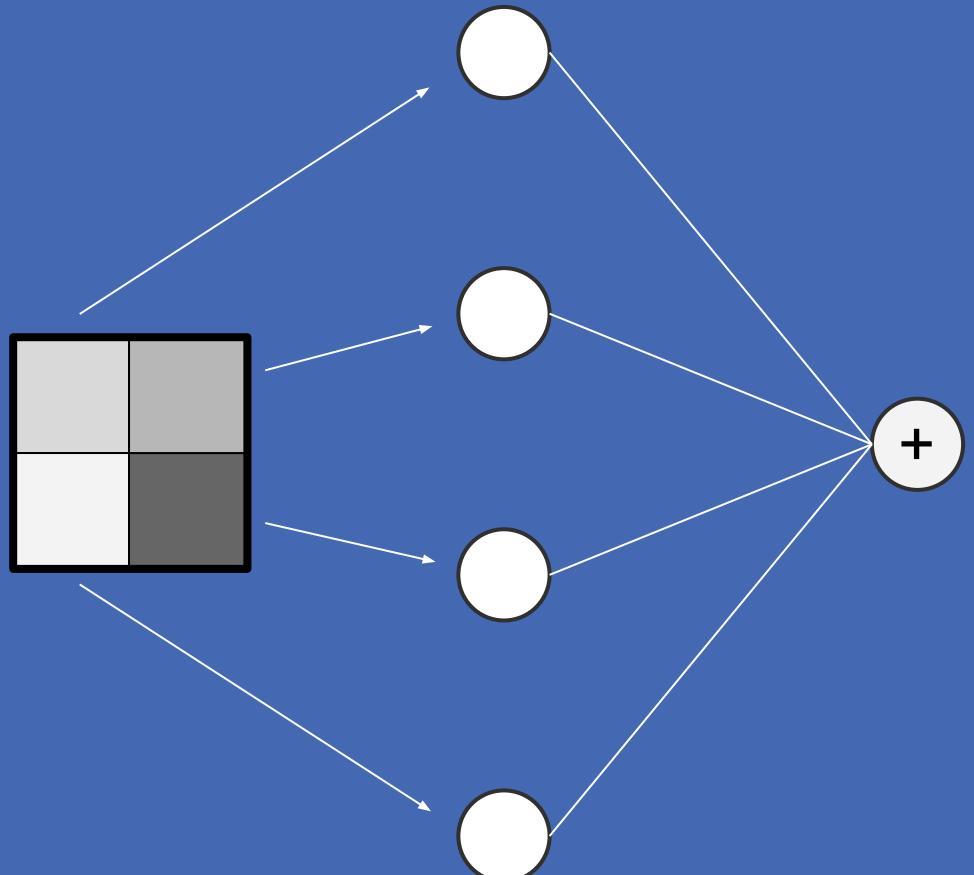
Input vector



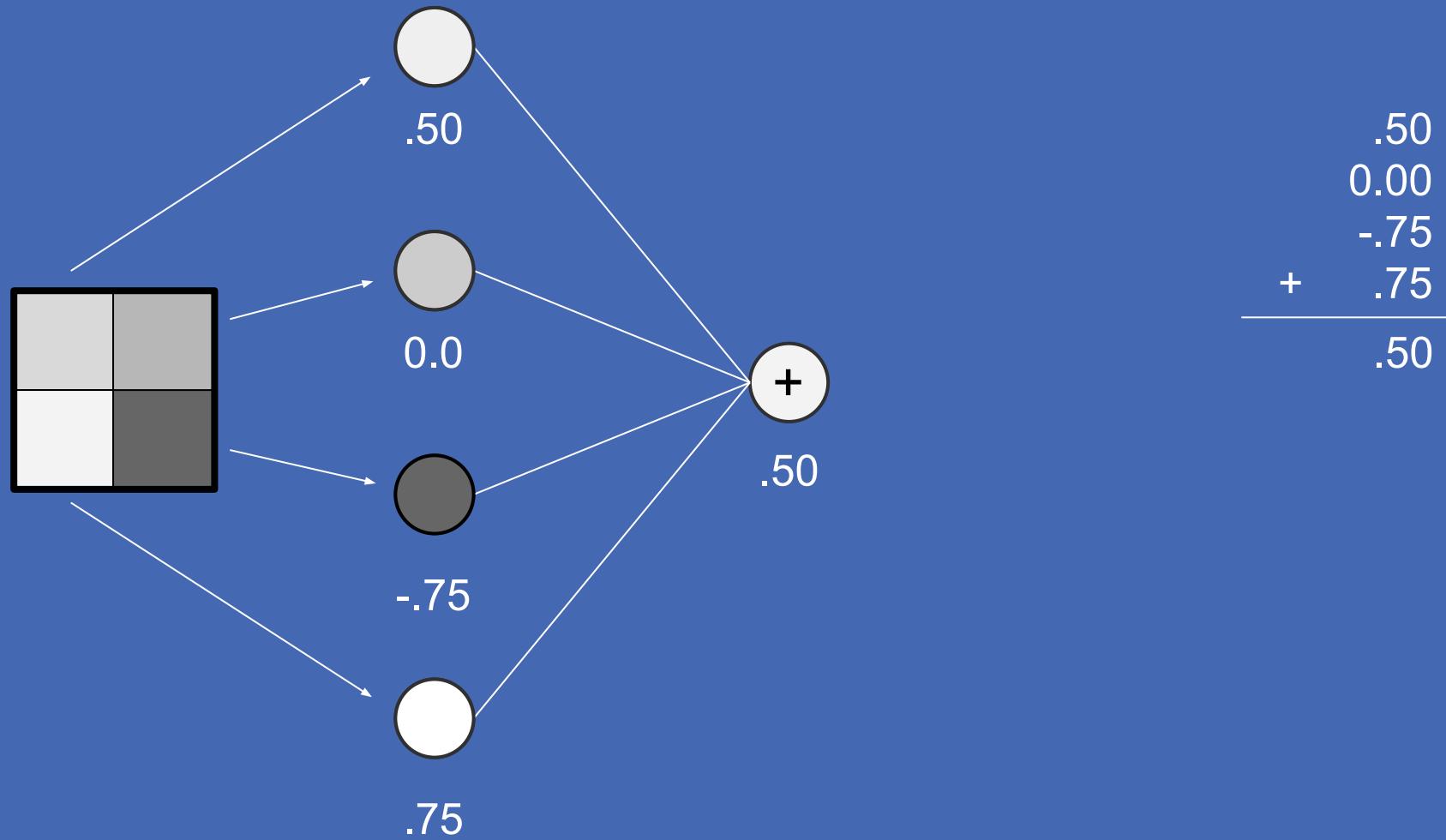
Receptive fields



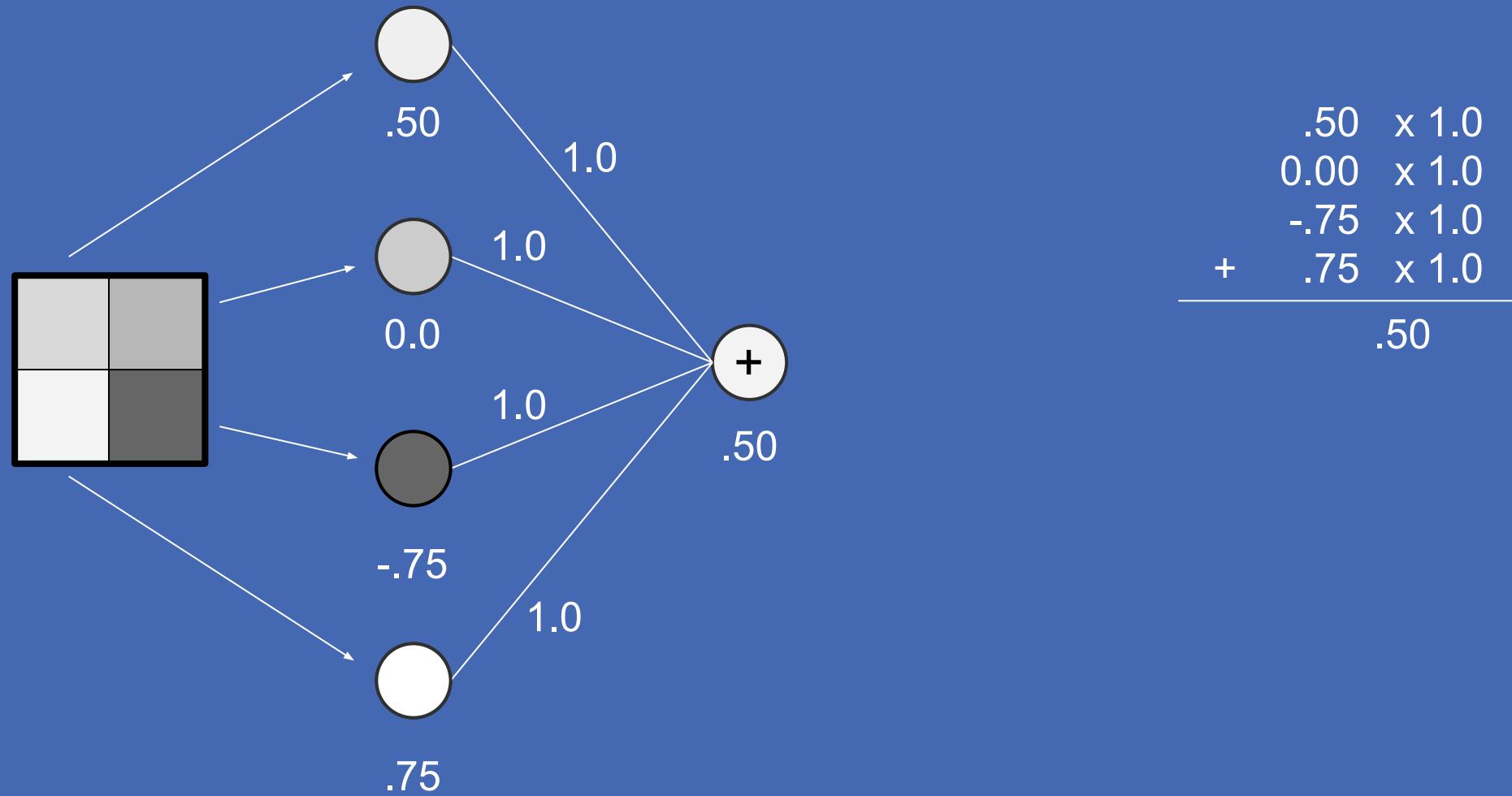
A neuron



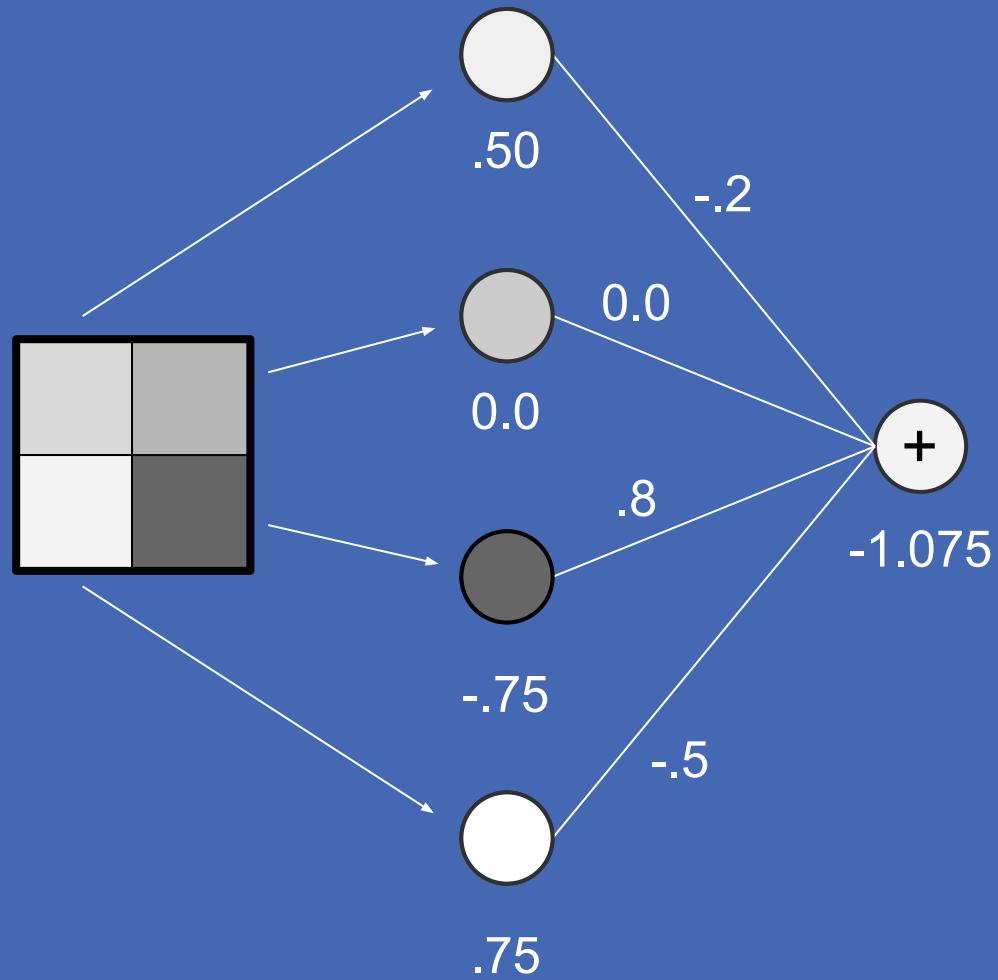
Sum all the inputs



Weights

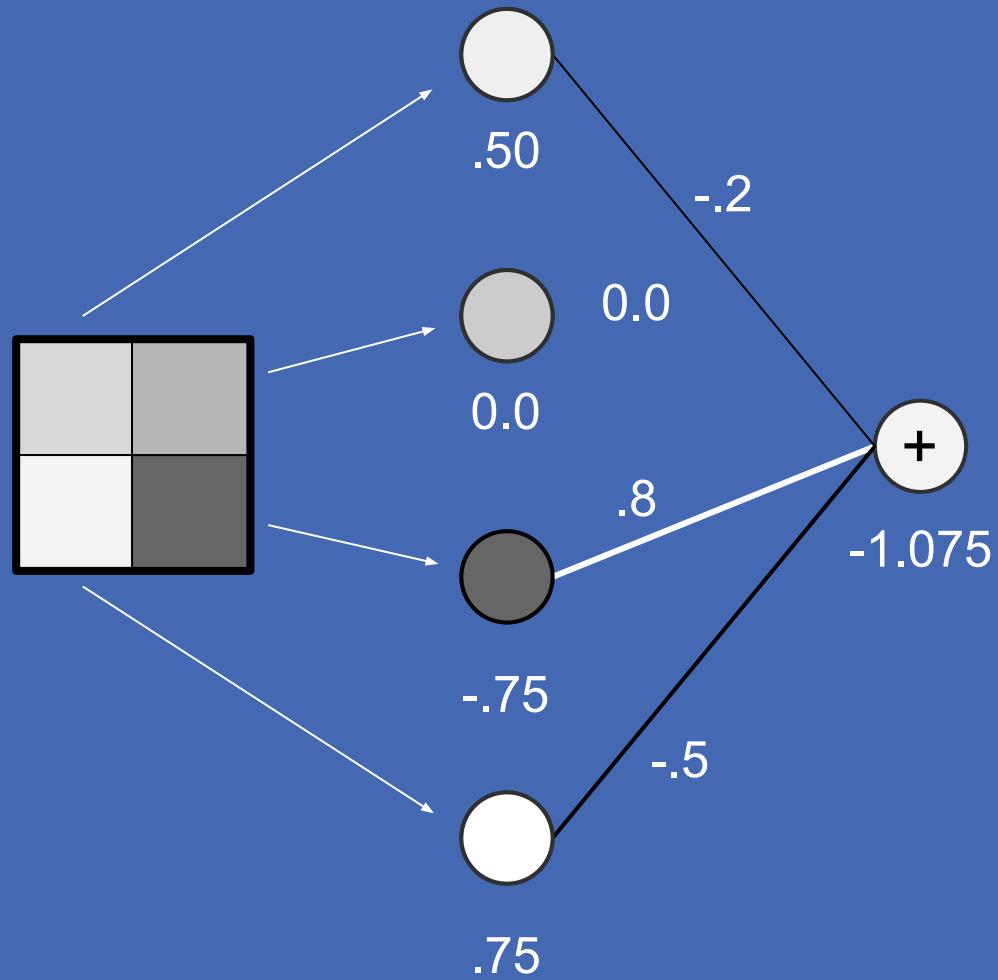


Weights



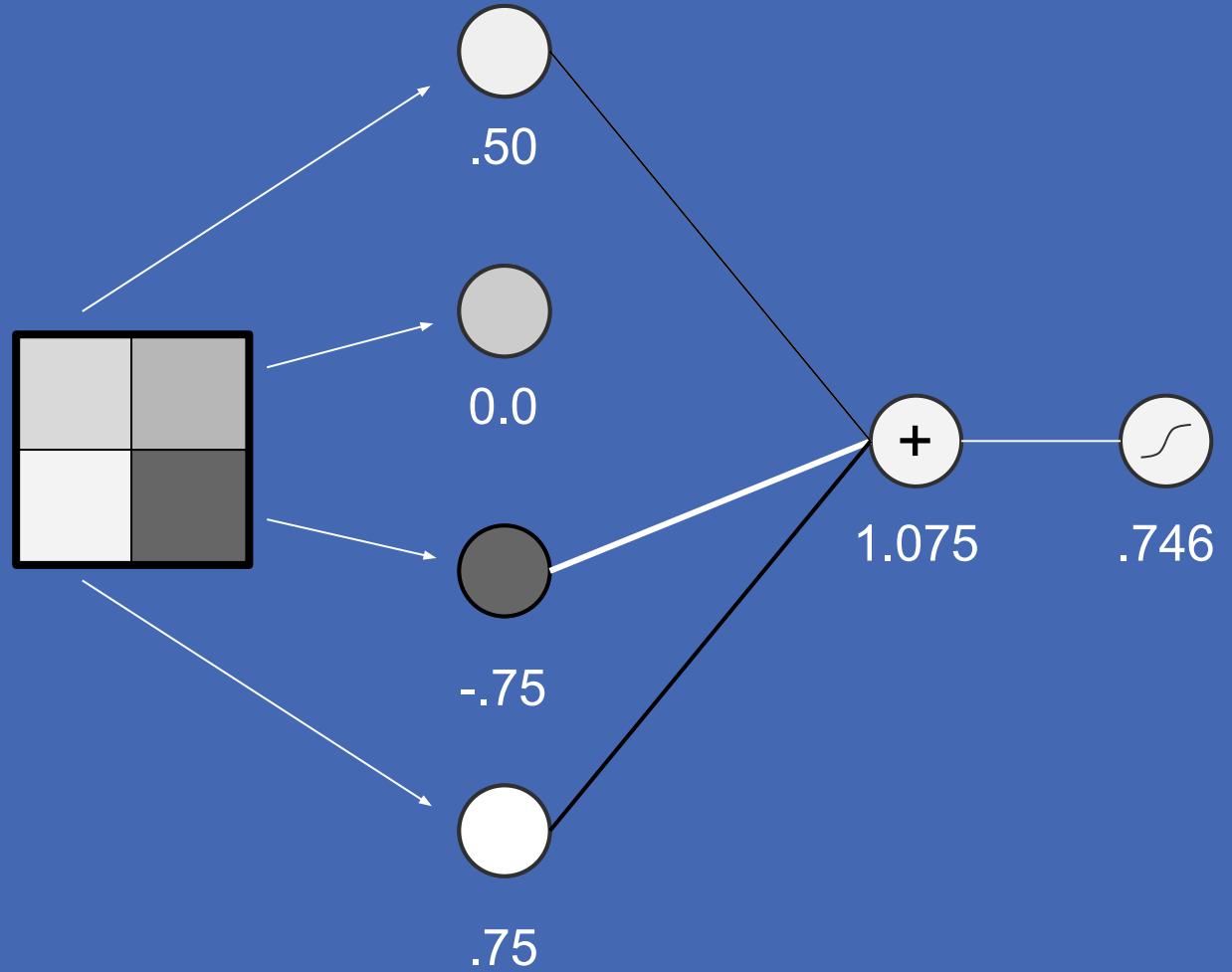
$$\begin{array}{r} .50 \times -.2 \\ 0.00 \times 0.0 \\ -.75 \times .8 \\ + .75 \times -.5 \\ \hline -1.075 \end{array}$$

Weights

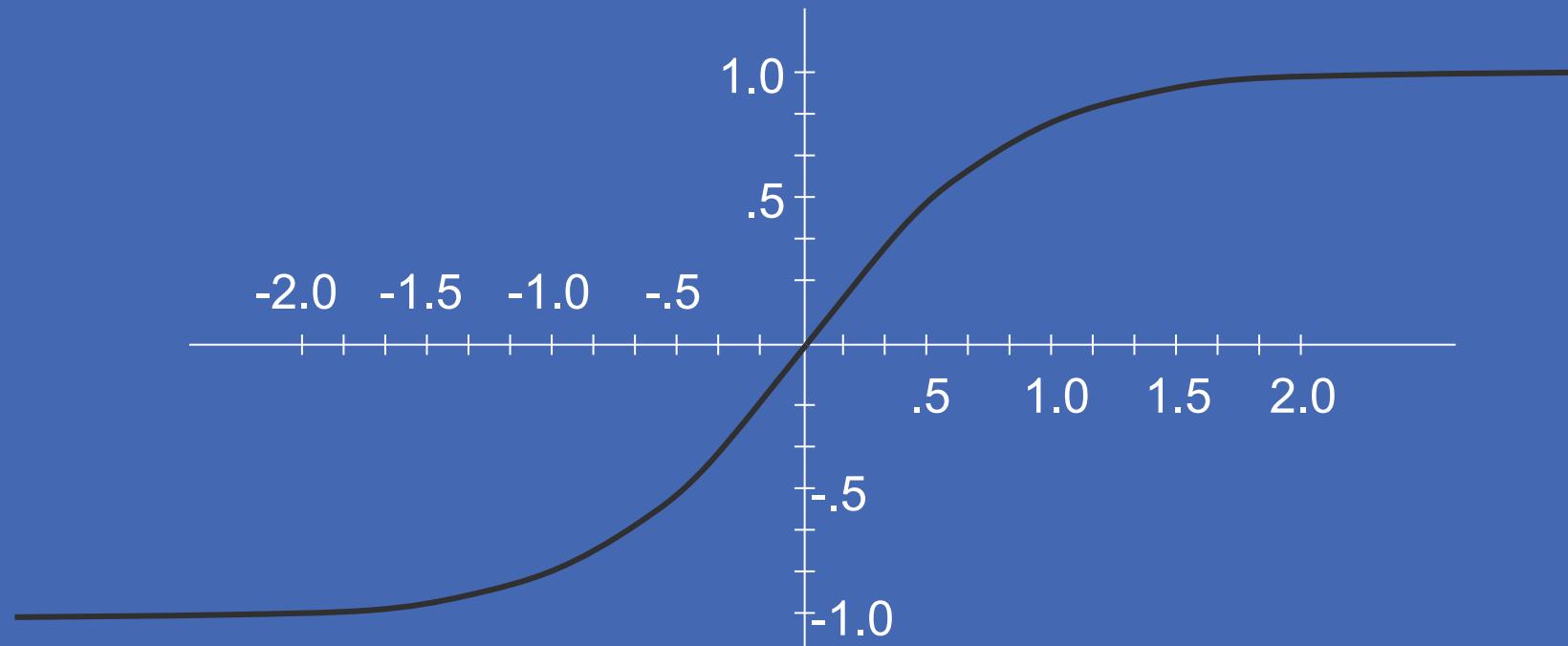


$$\begin{array}{r} .50 \times -.2 \\ 0.00 \times 0.0 \\ -.75 \times .8 \\ + .75 \times -.5 \\ \hline -1.075 \end{array}$$

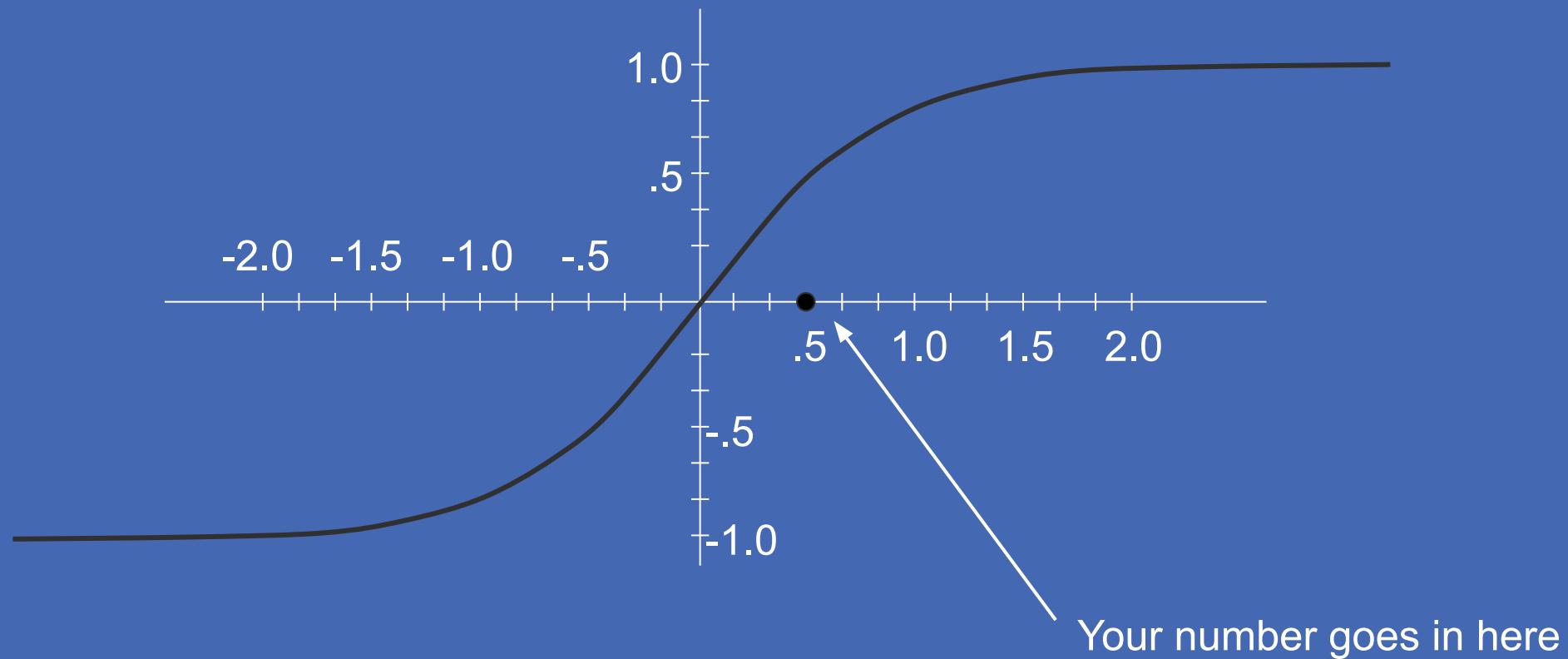
Squash the result



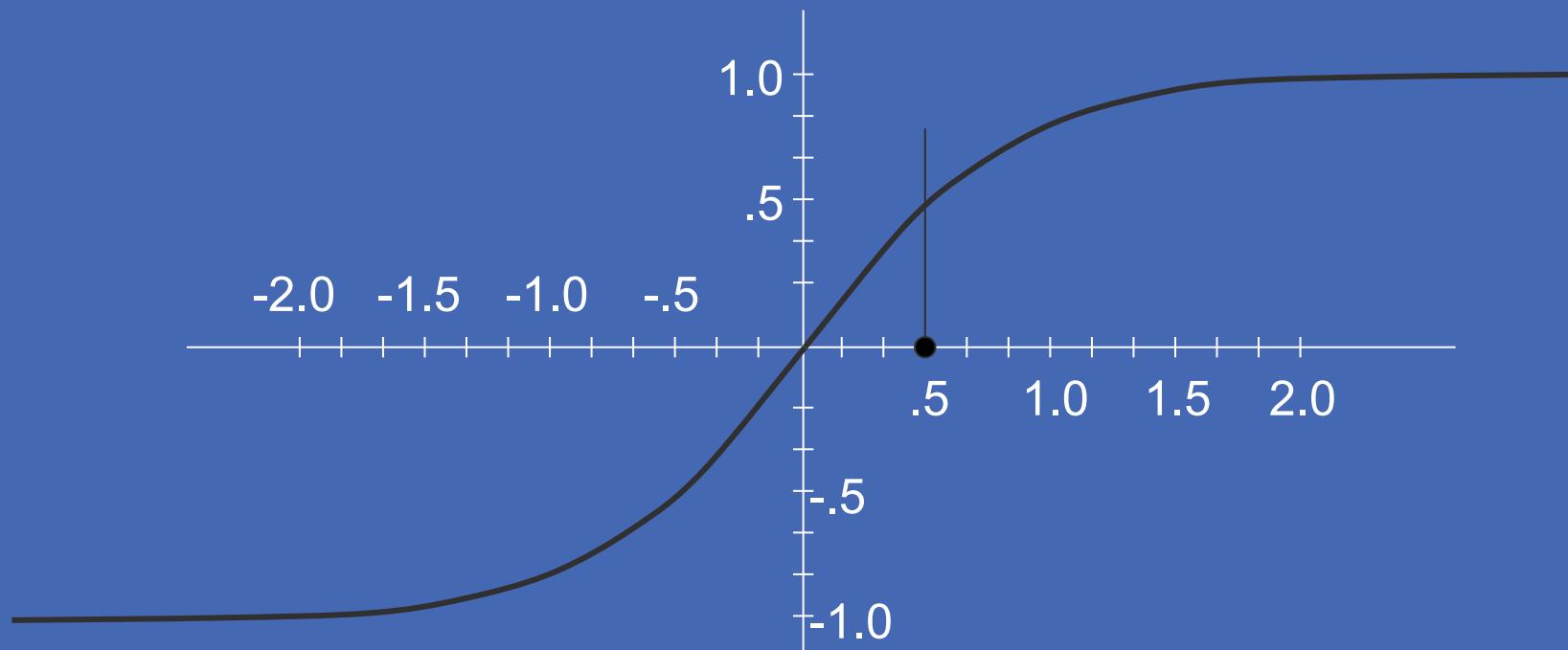
Sigmoid squashing function



Sigmoid squashing function



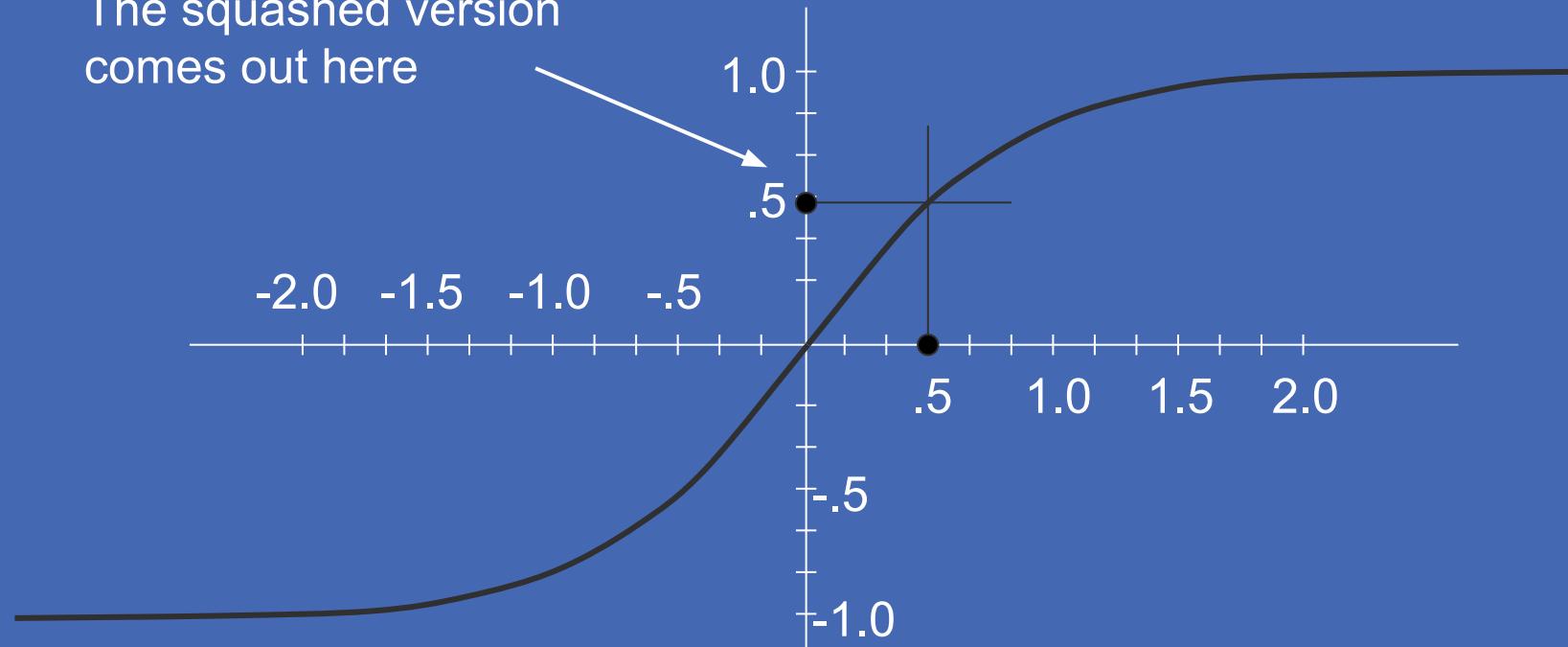
Sigmoid squashing function



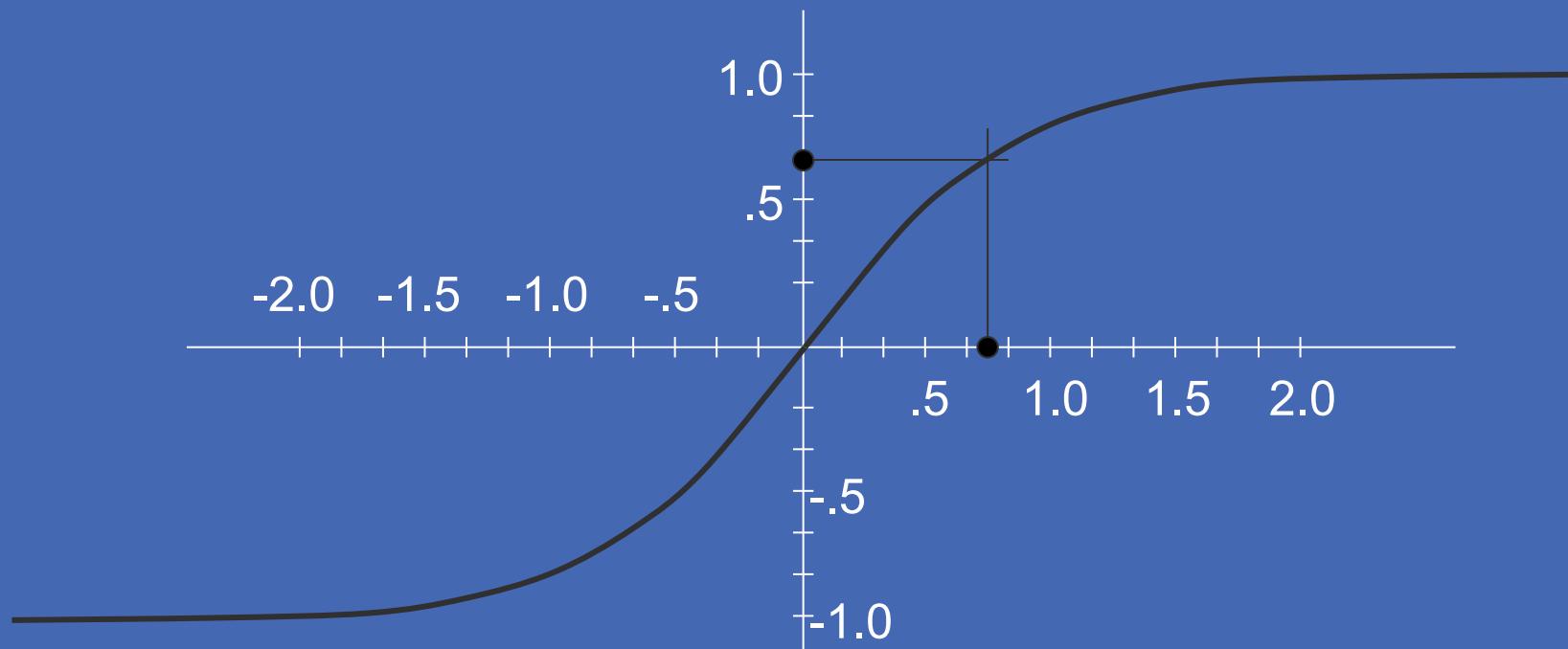
Sigmoid squashing function



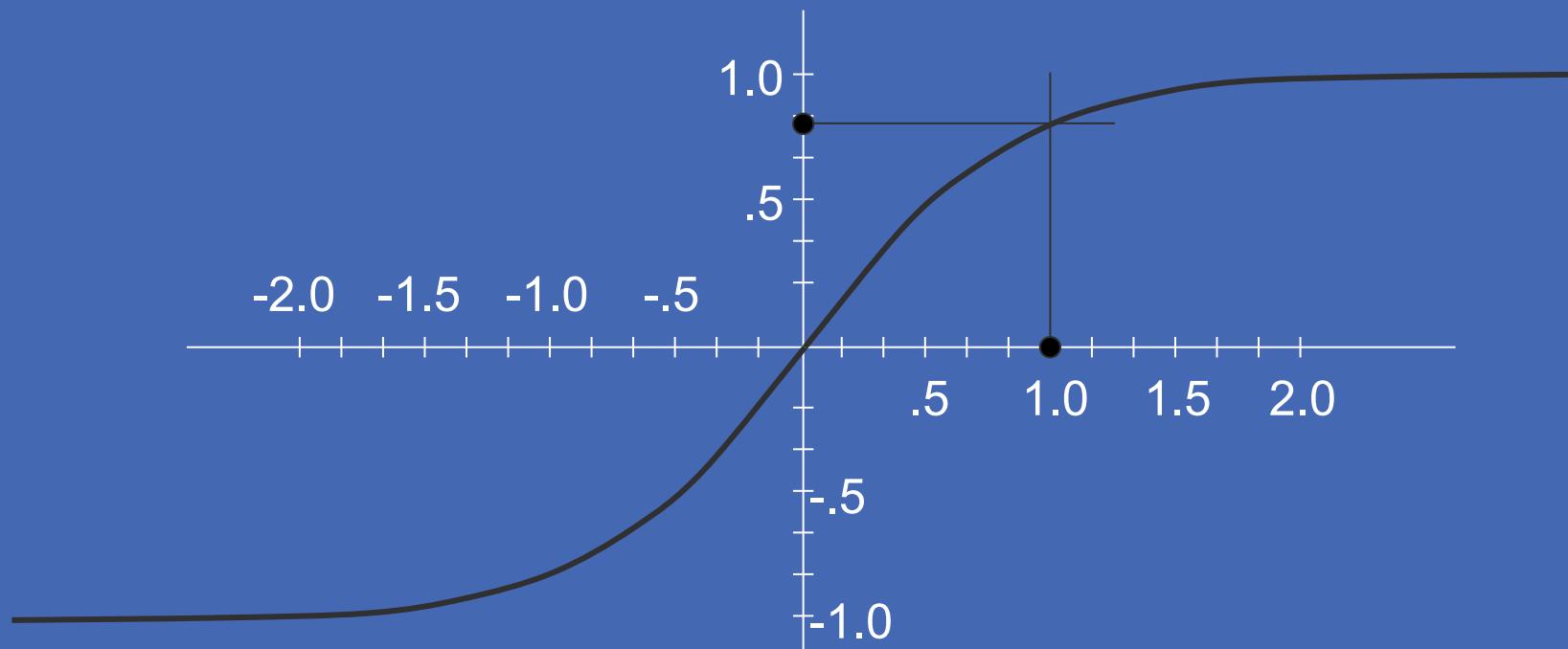
The squashed version
comes out here



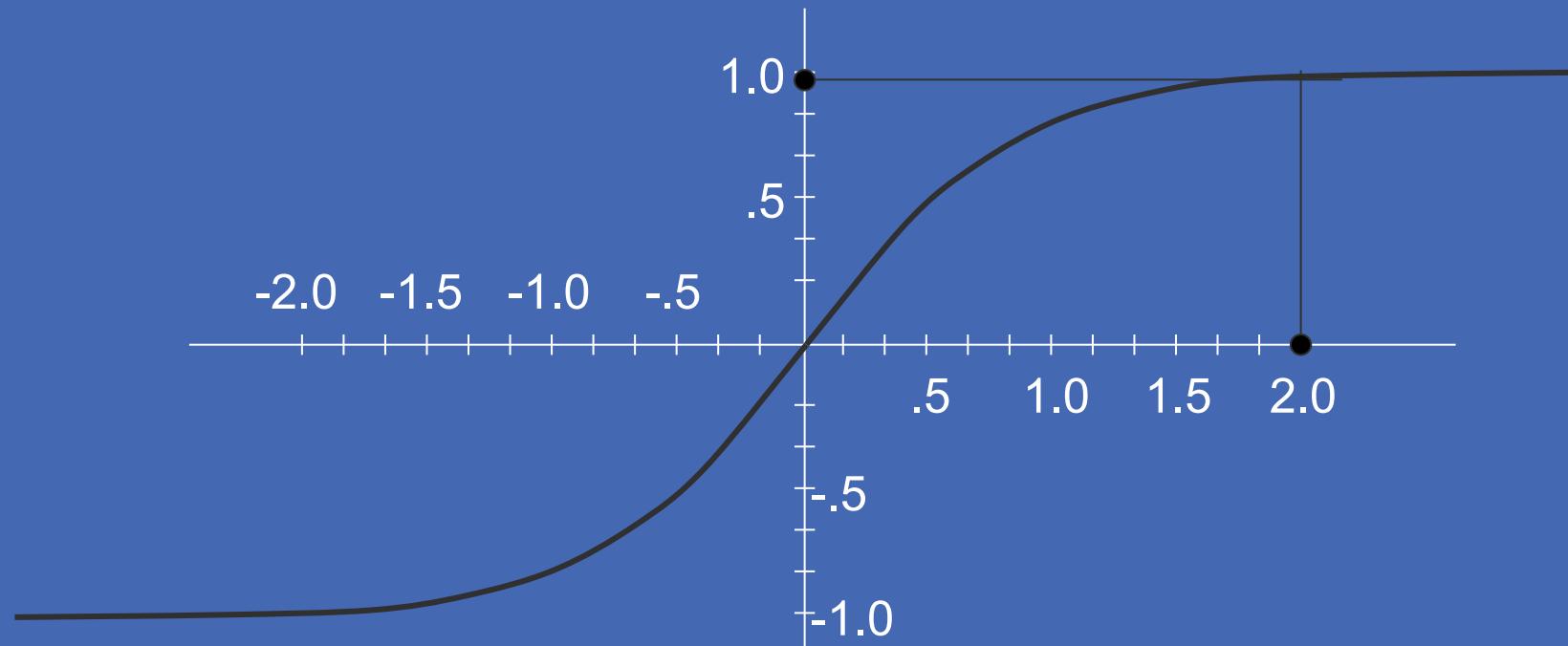
Sigmoid squashing function



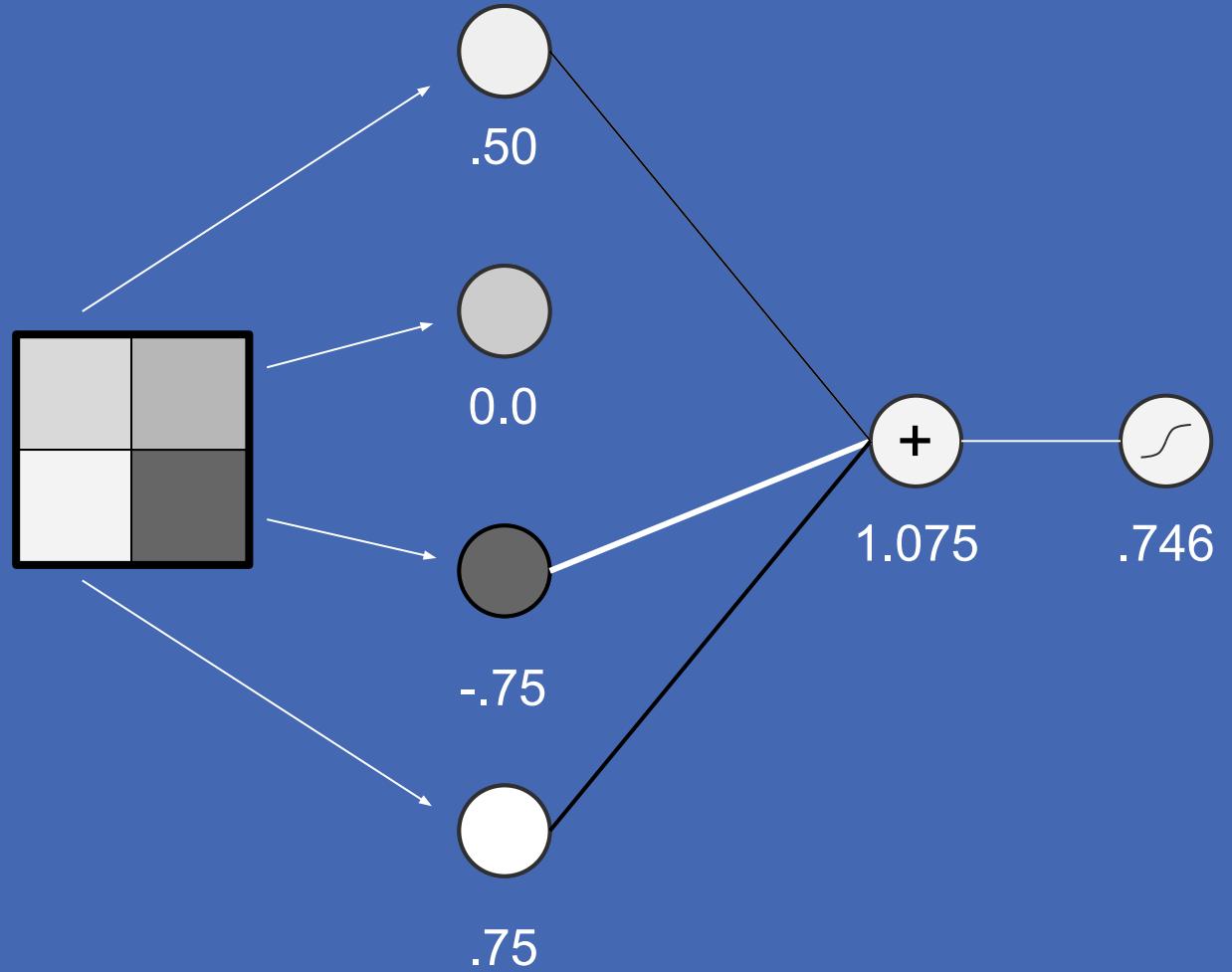
Sigmoid squashing function



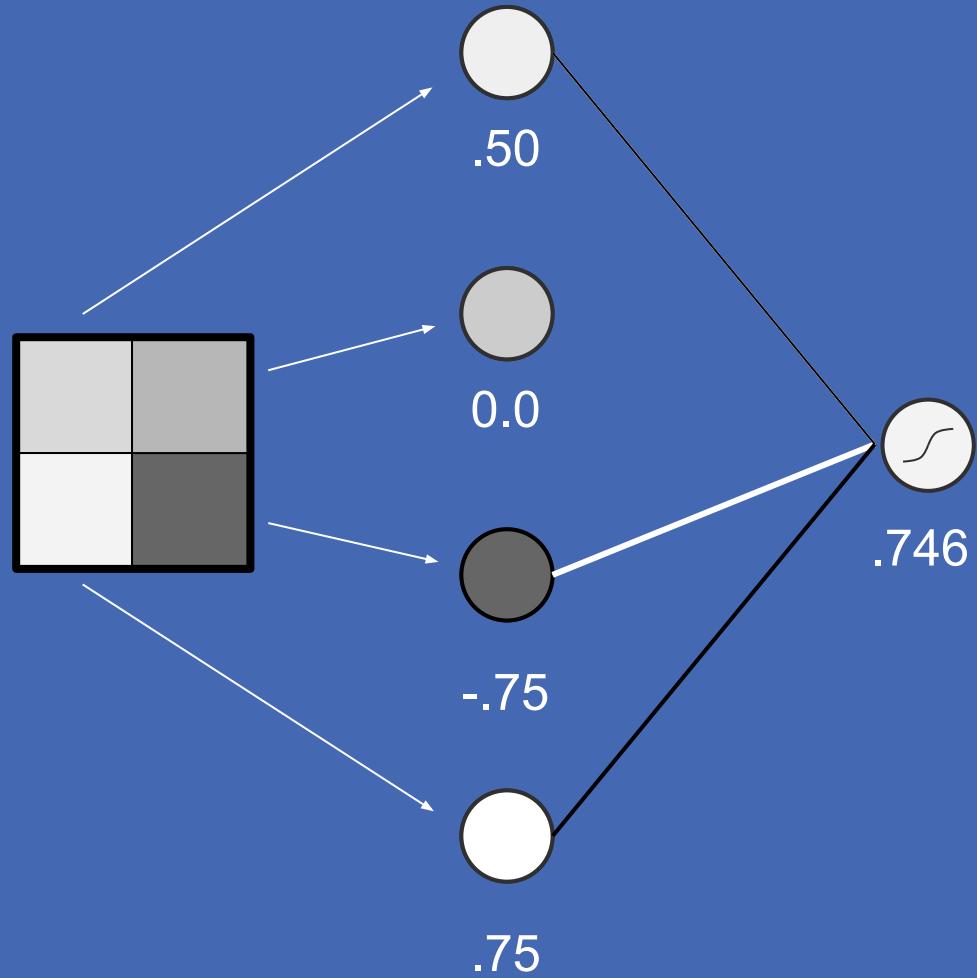
No matter what you start with, the answer stays between -1 and 1.



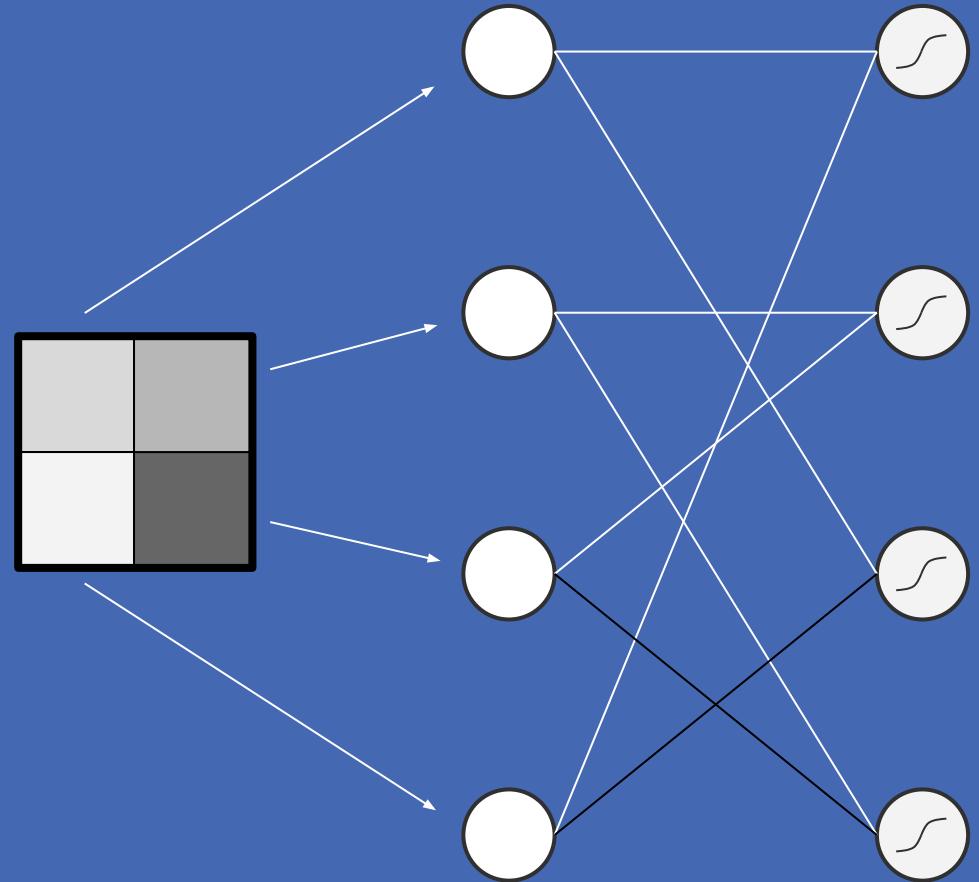
Squash the result



Weighted sum-and-squash neuron

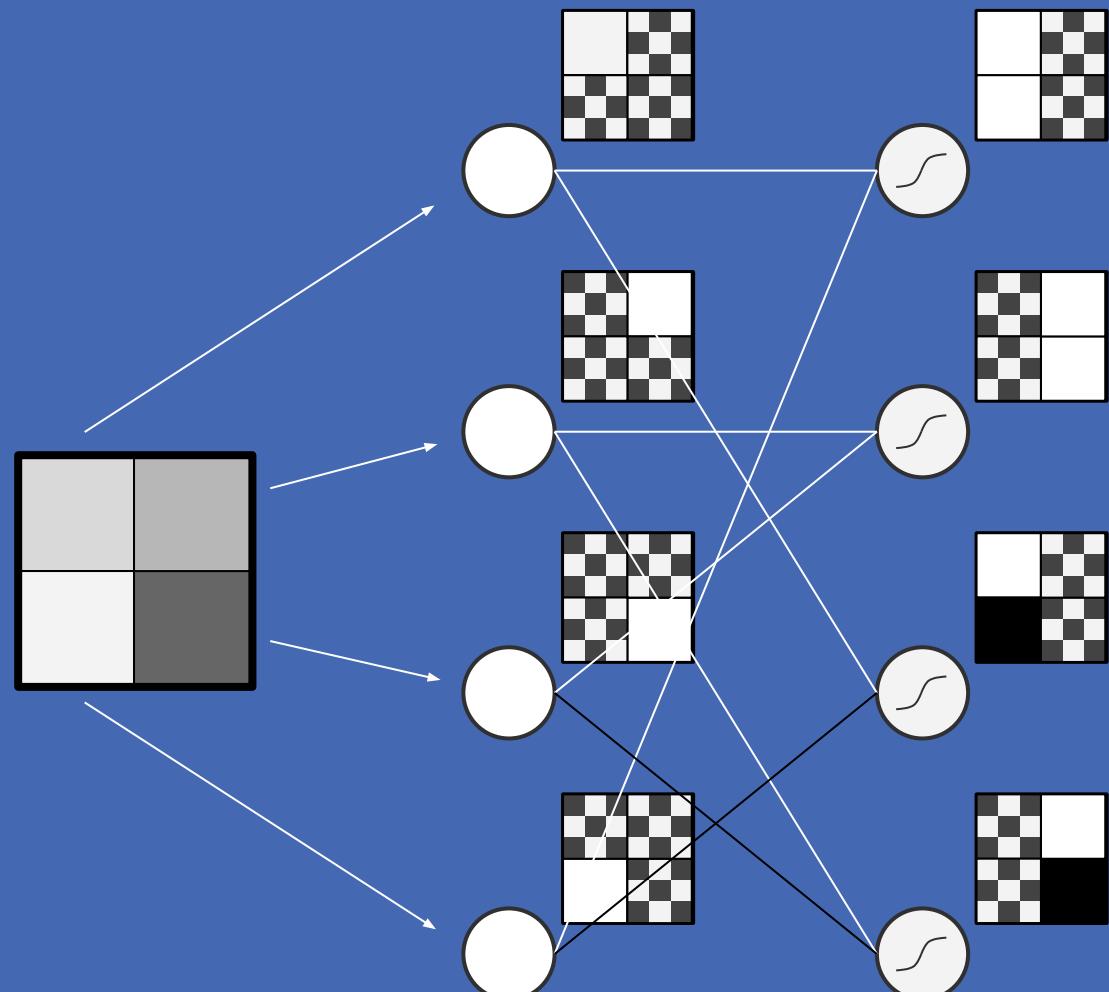


Make lots of neurons, identical except for weights

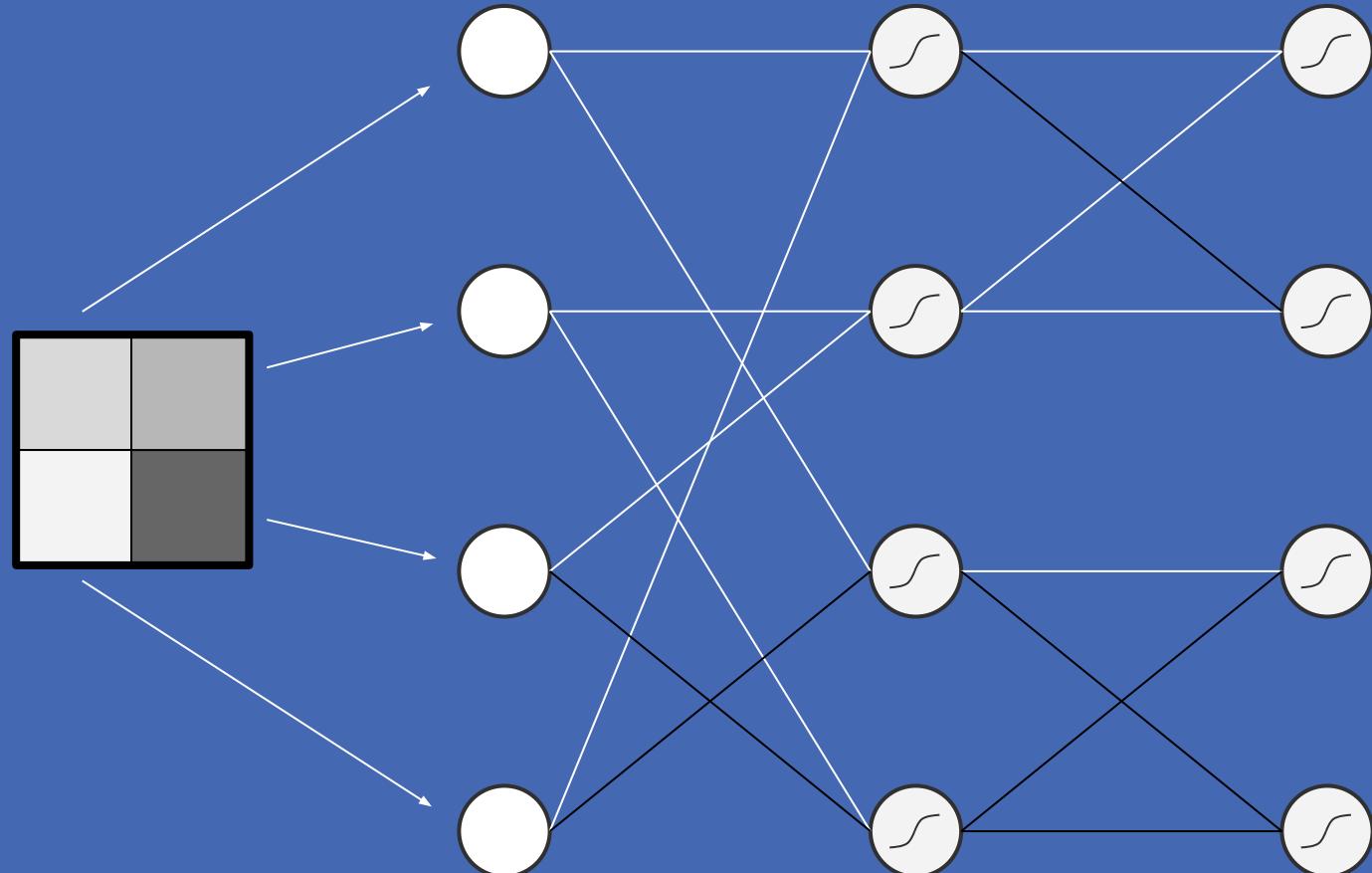


To keep our picture clear,
weights will either be
1.0 (white)
-1.0 (black) or
0.0 (missing)

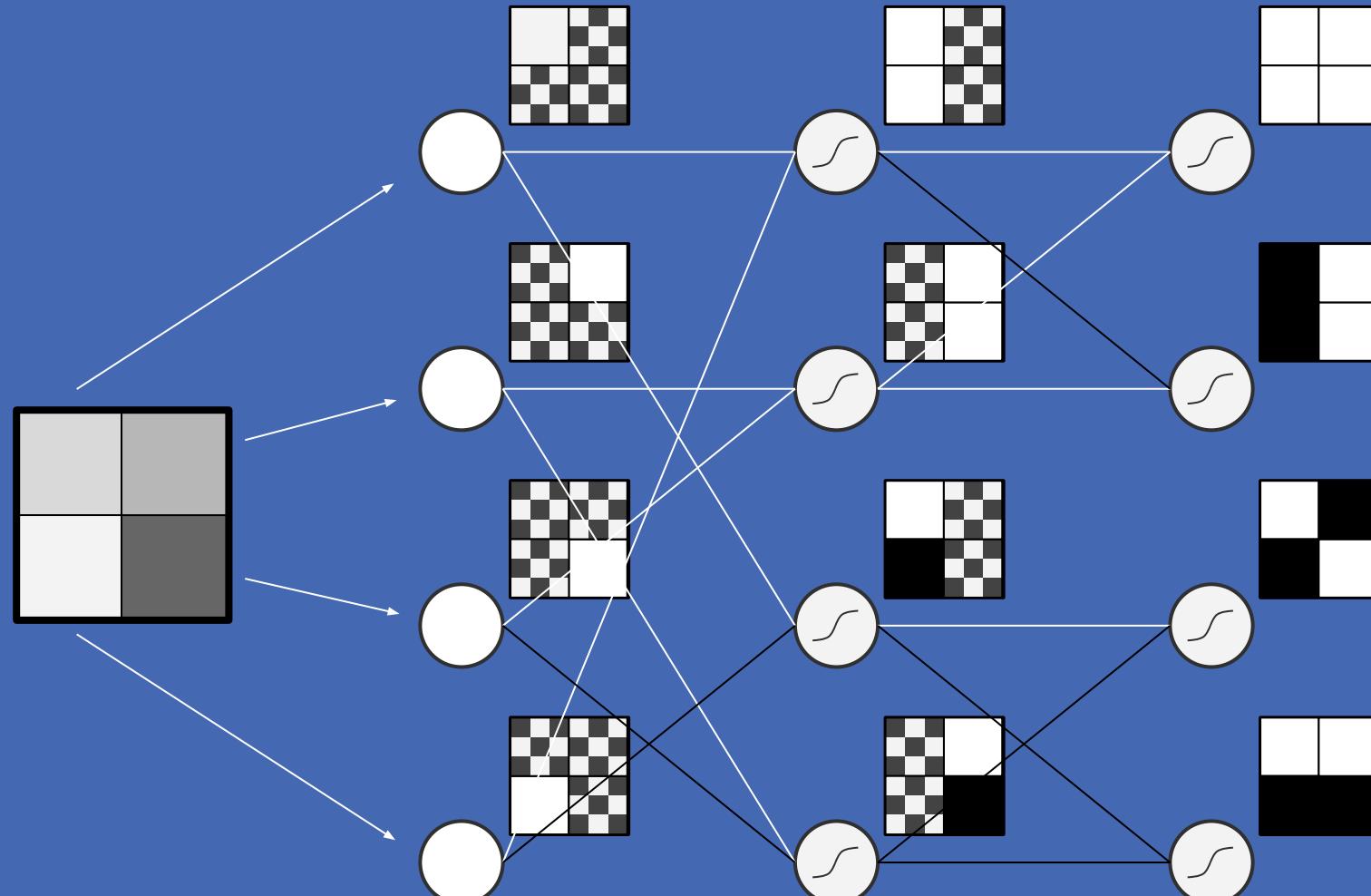
Receptive fields get more complex



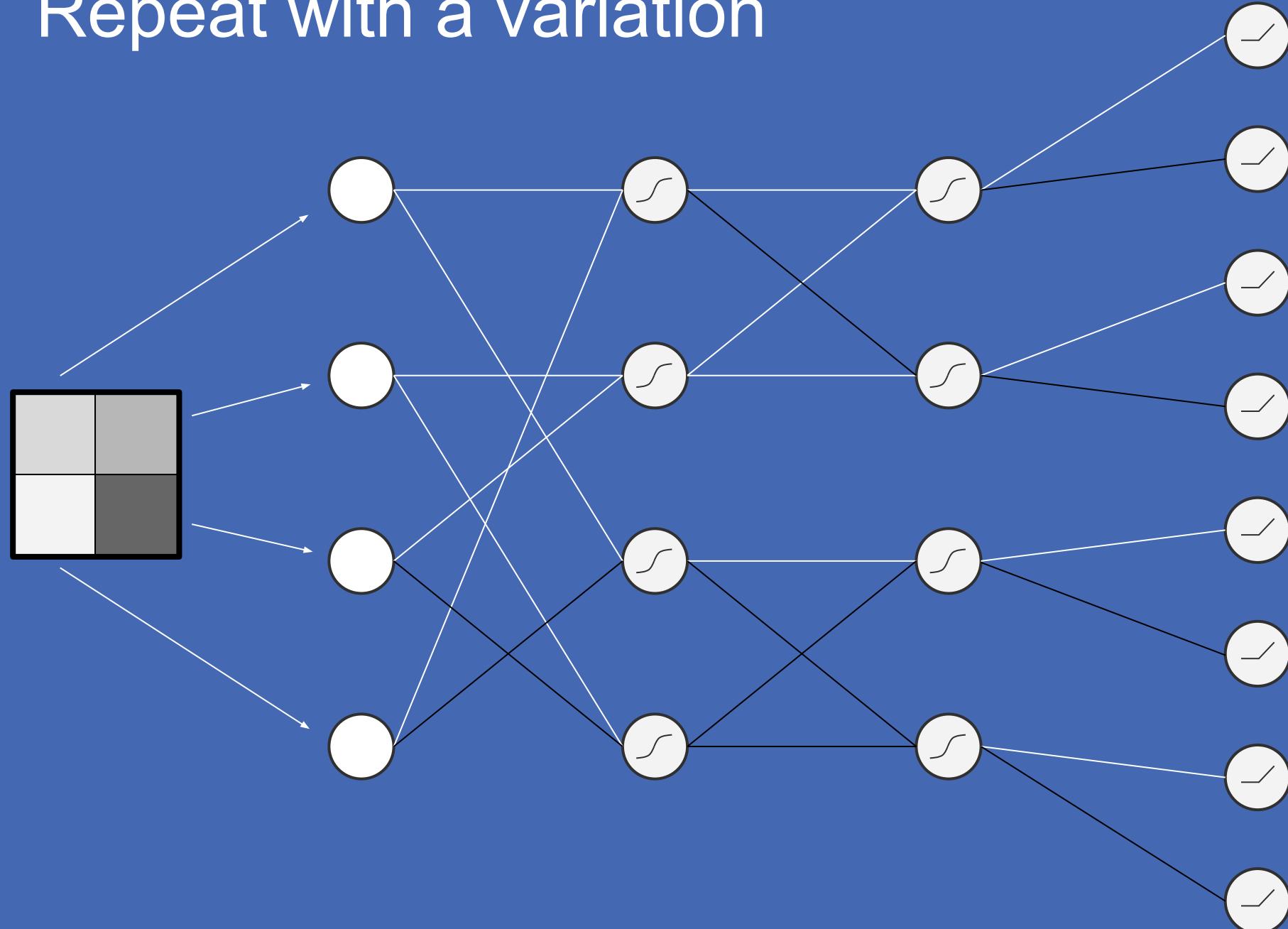
Repeat for additional layers



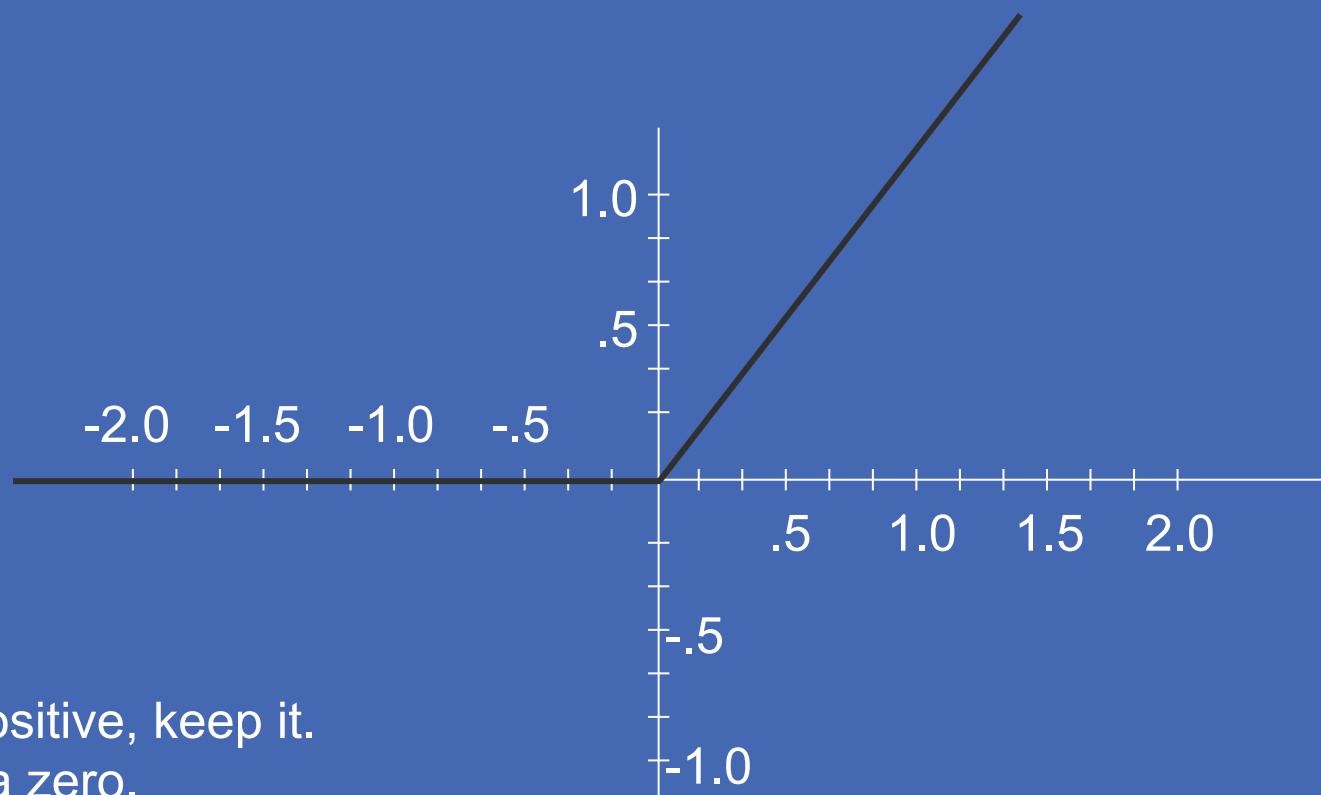
Receptive fields get still more complex



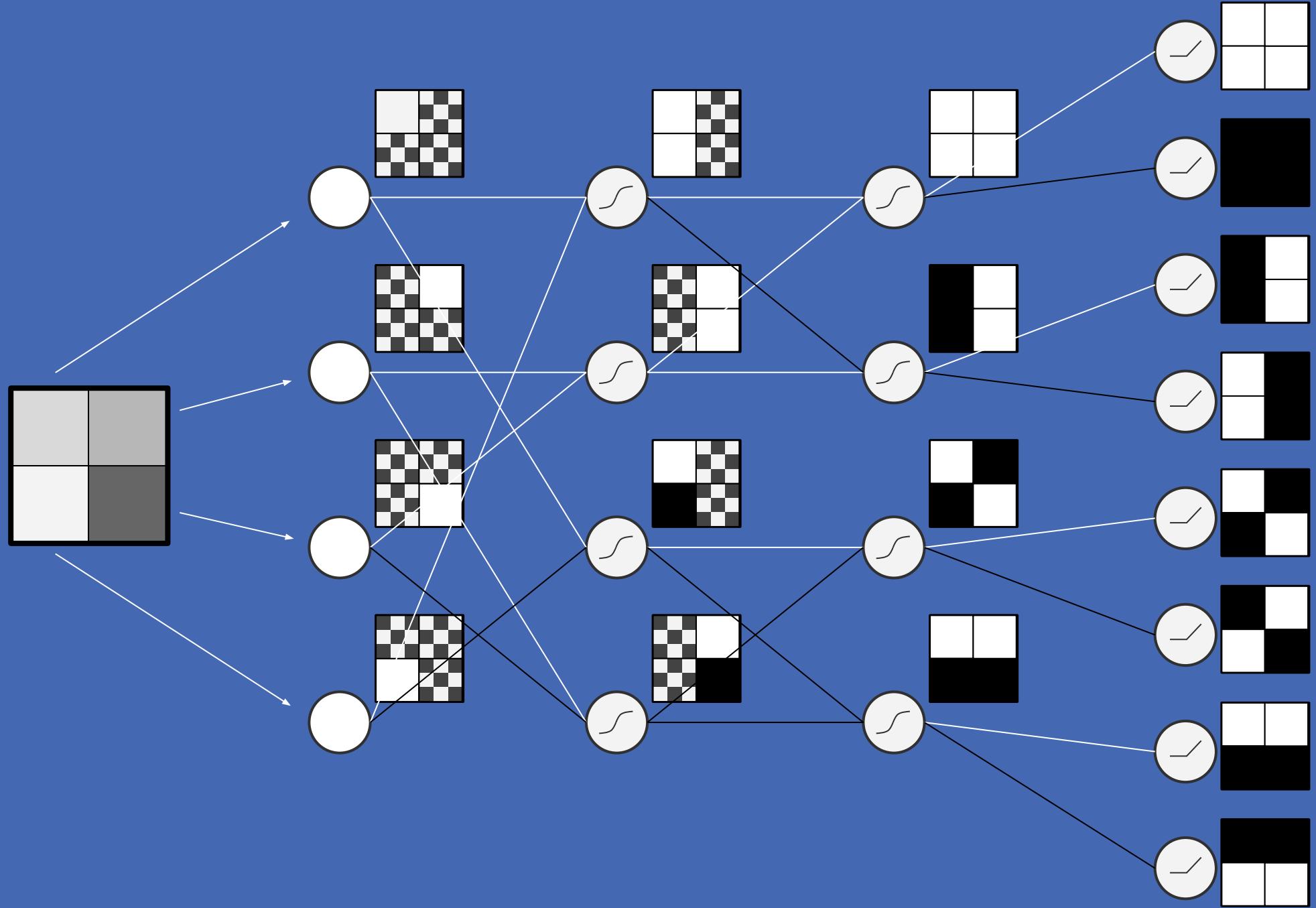
Repeat with a variation



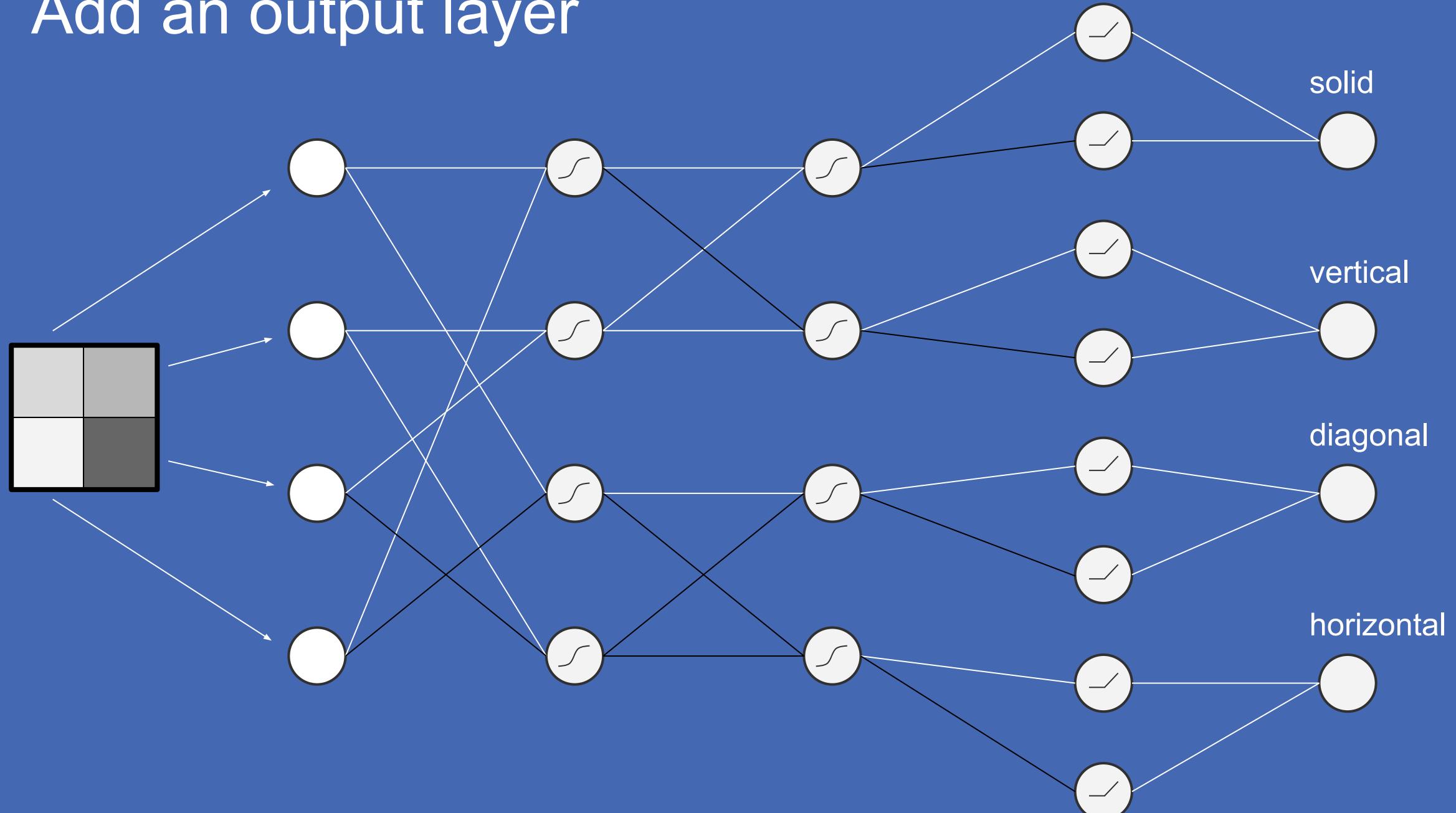
Rectified linear units (ReLUs)

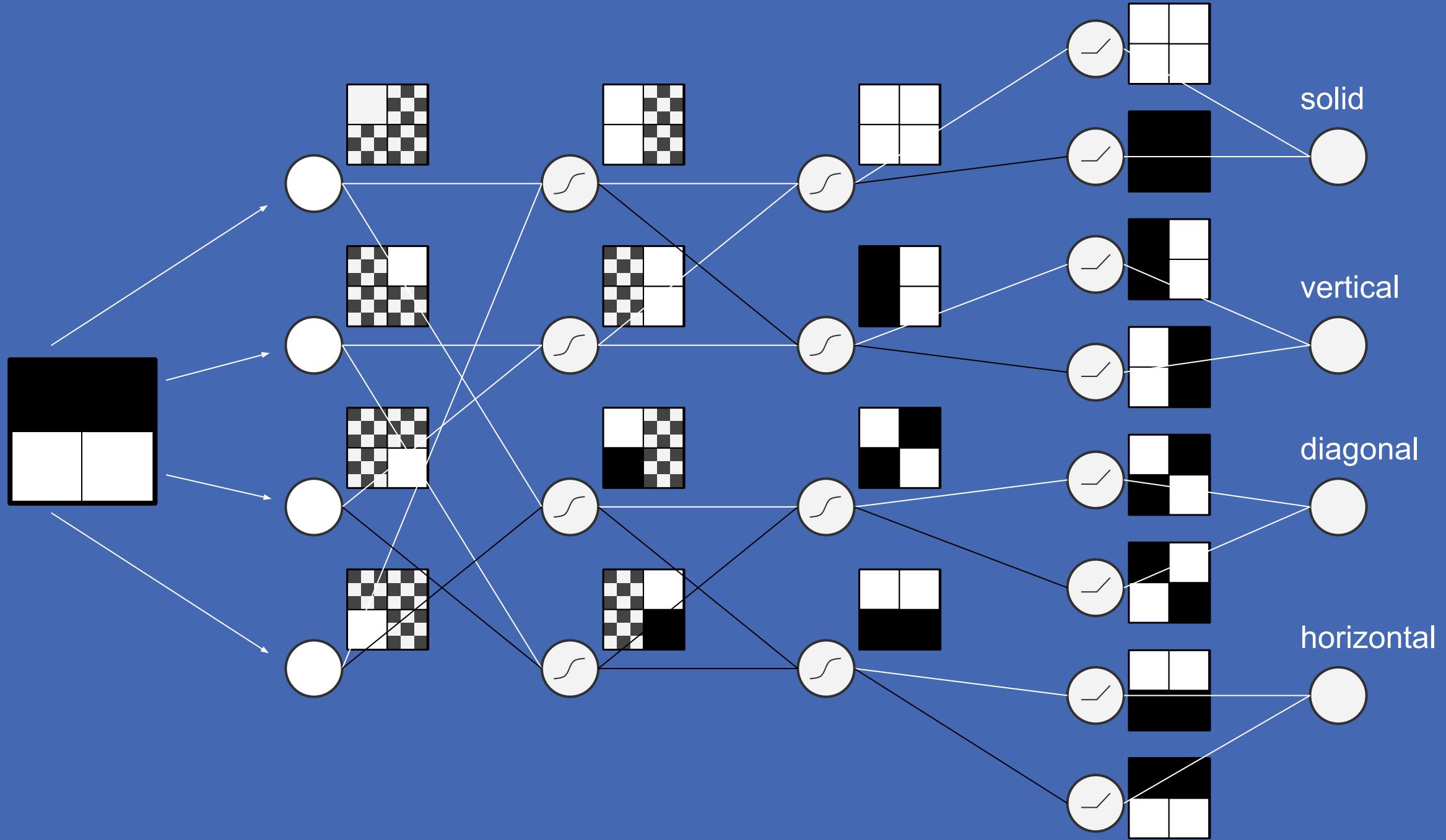


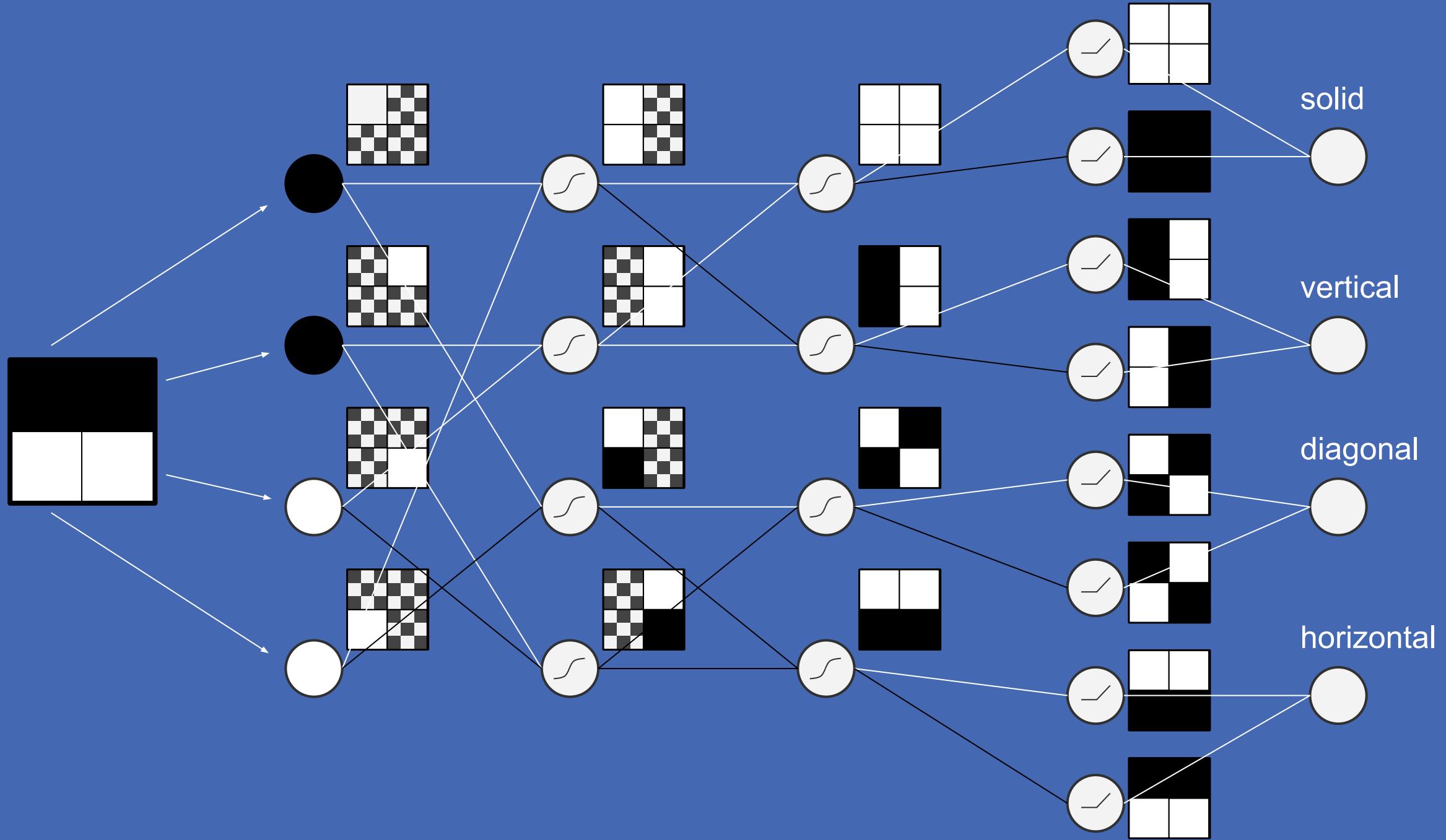
If your number is positive, keep it.
Otherwise you get a zero.

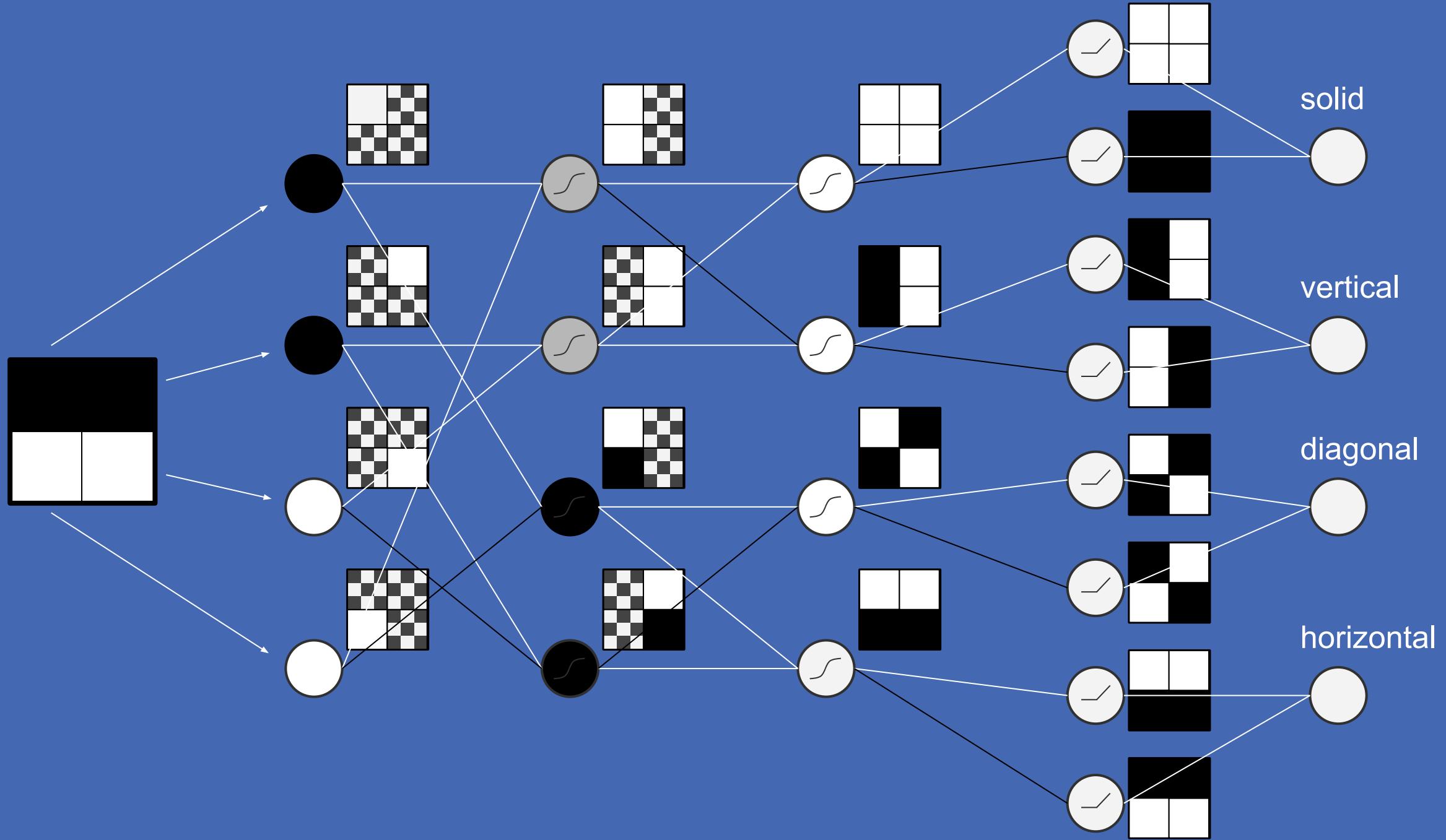


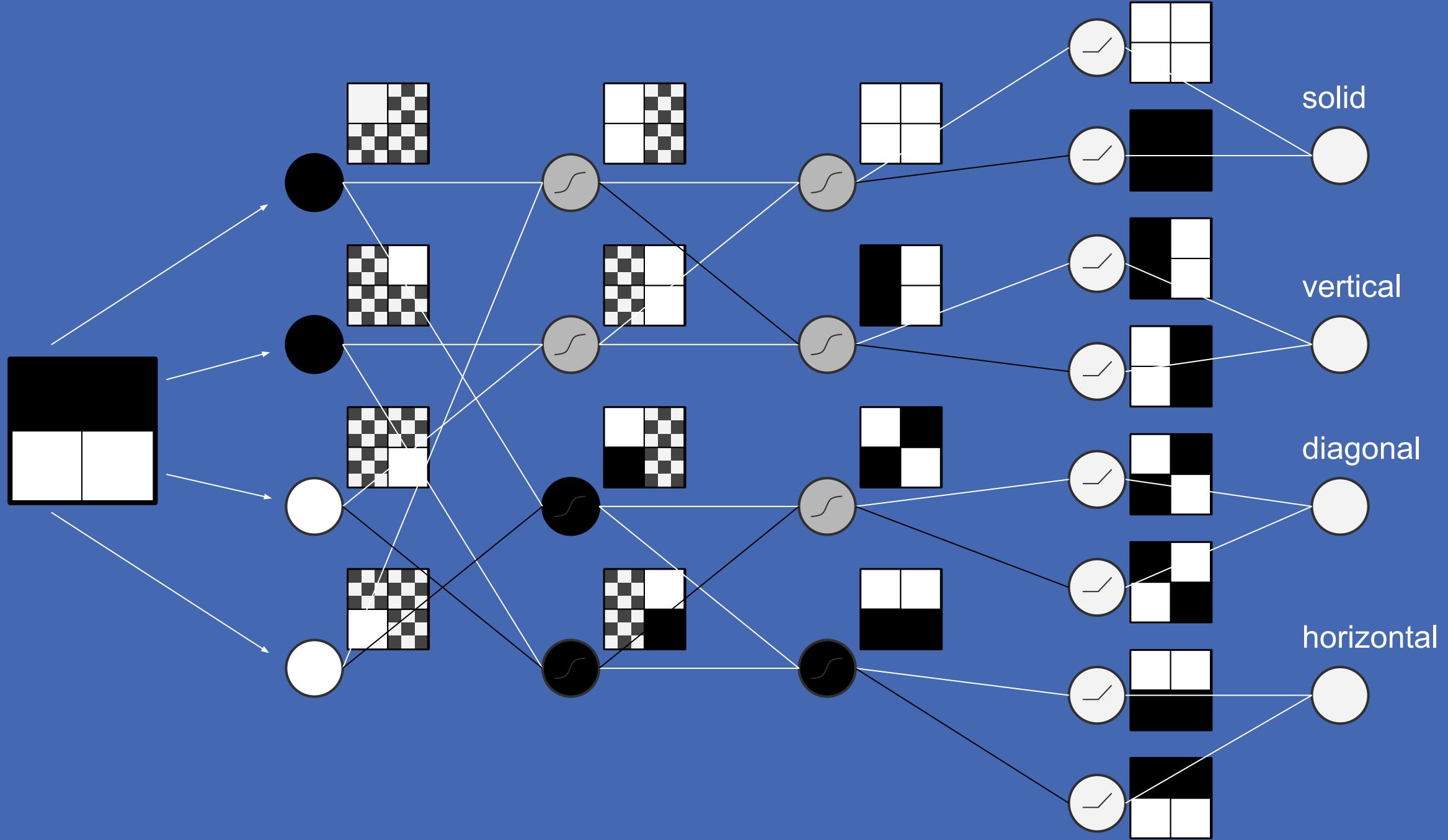
Add an output layer

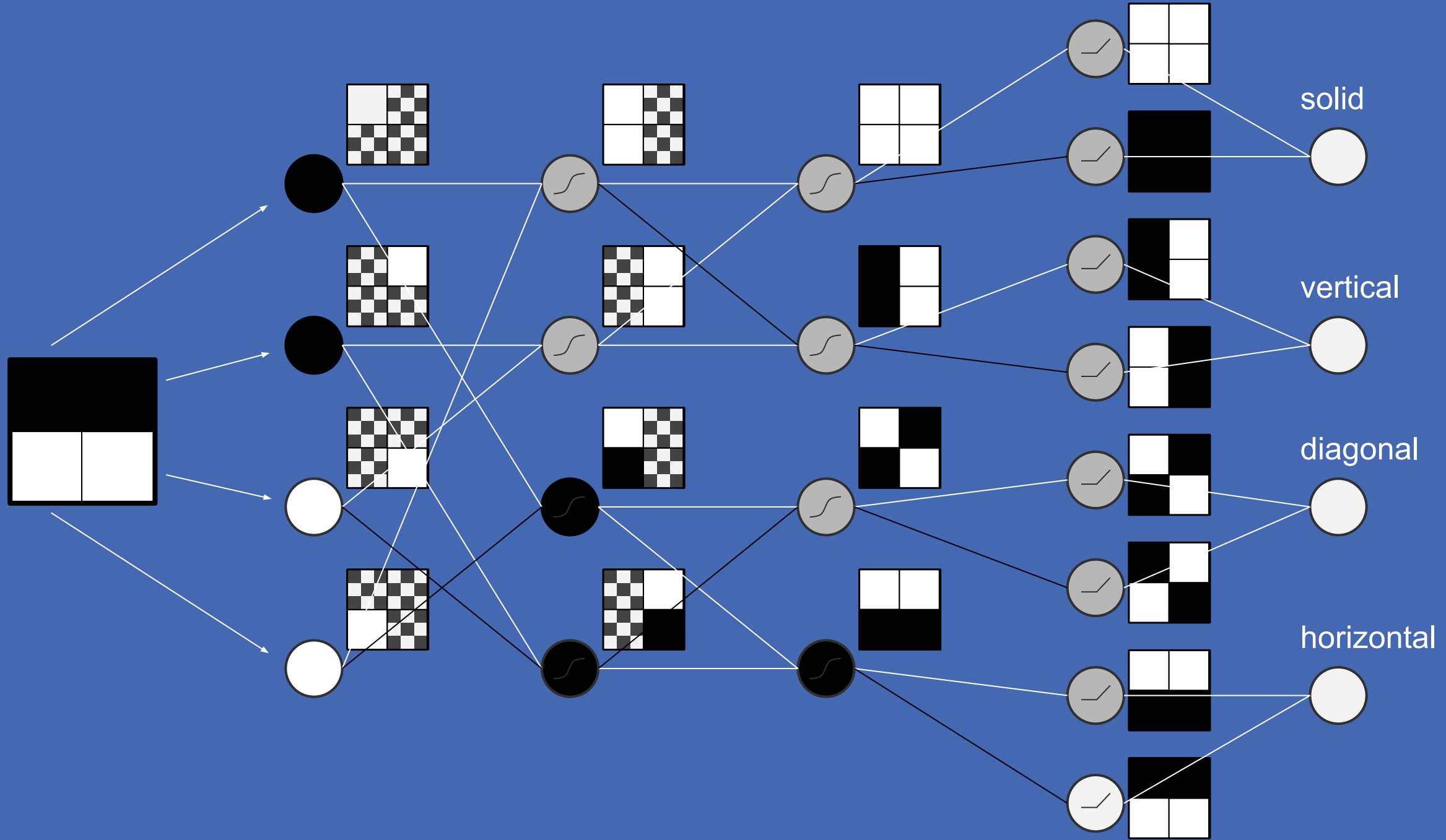


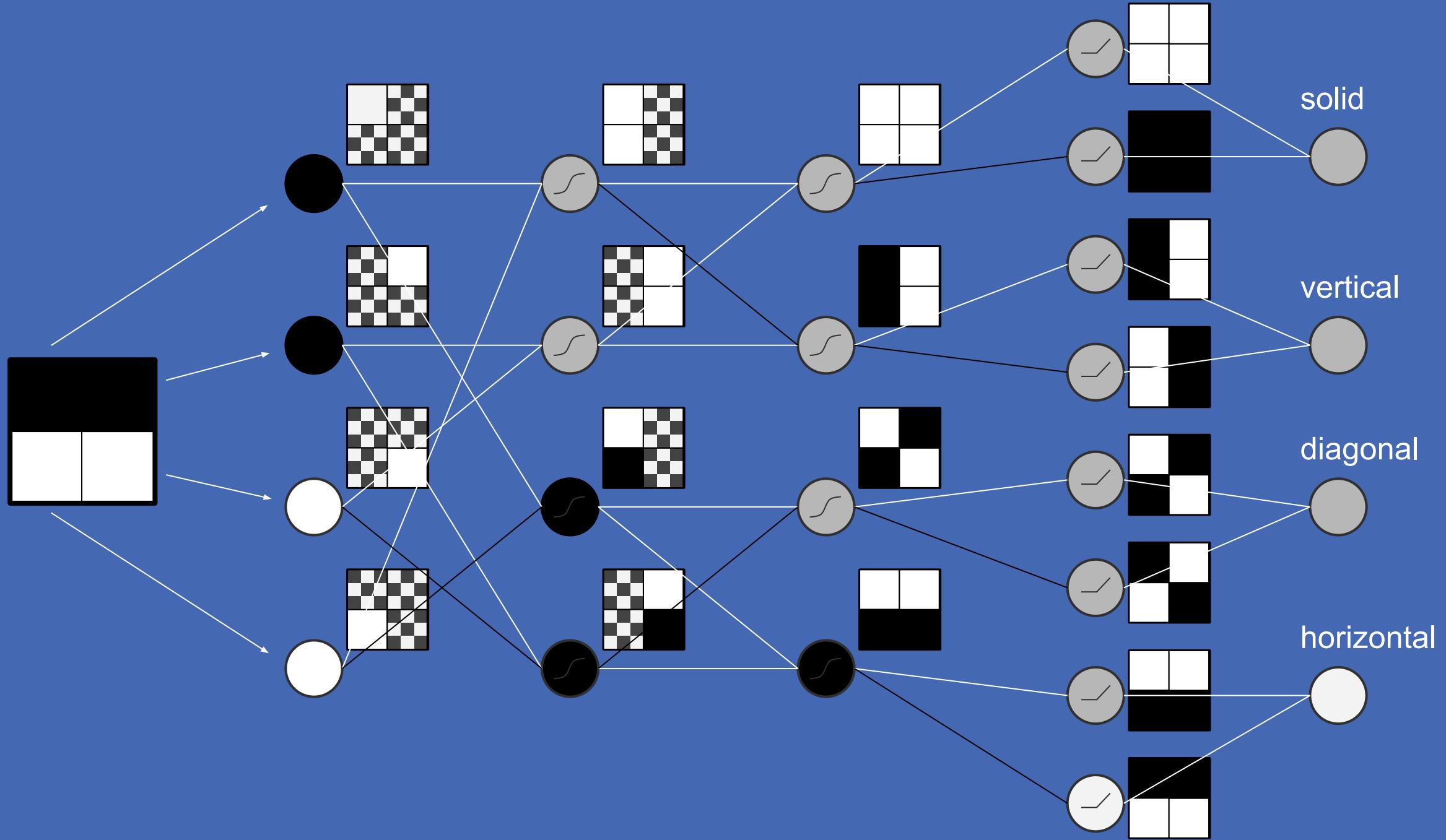


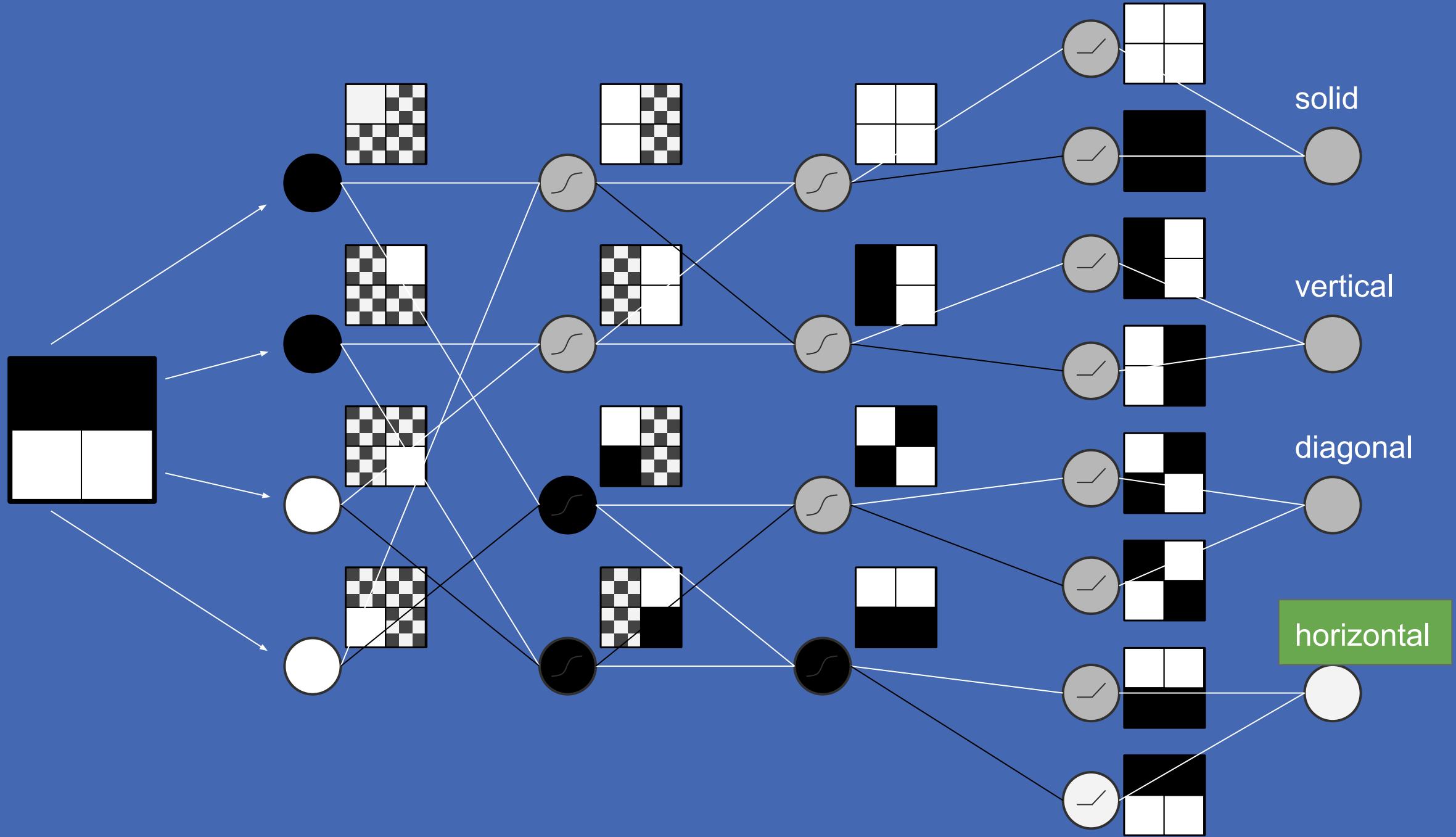






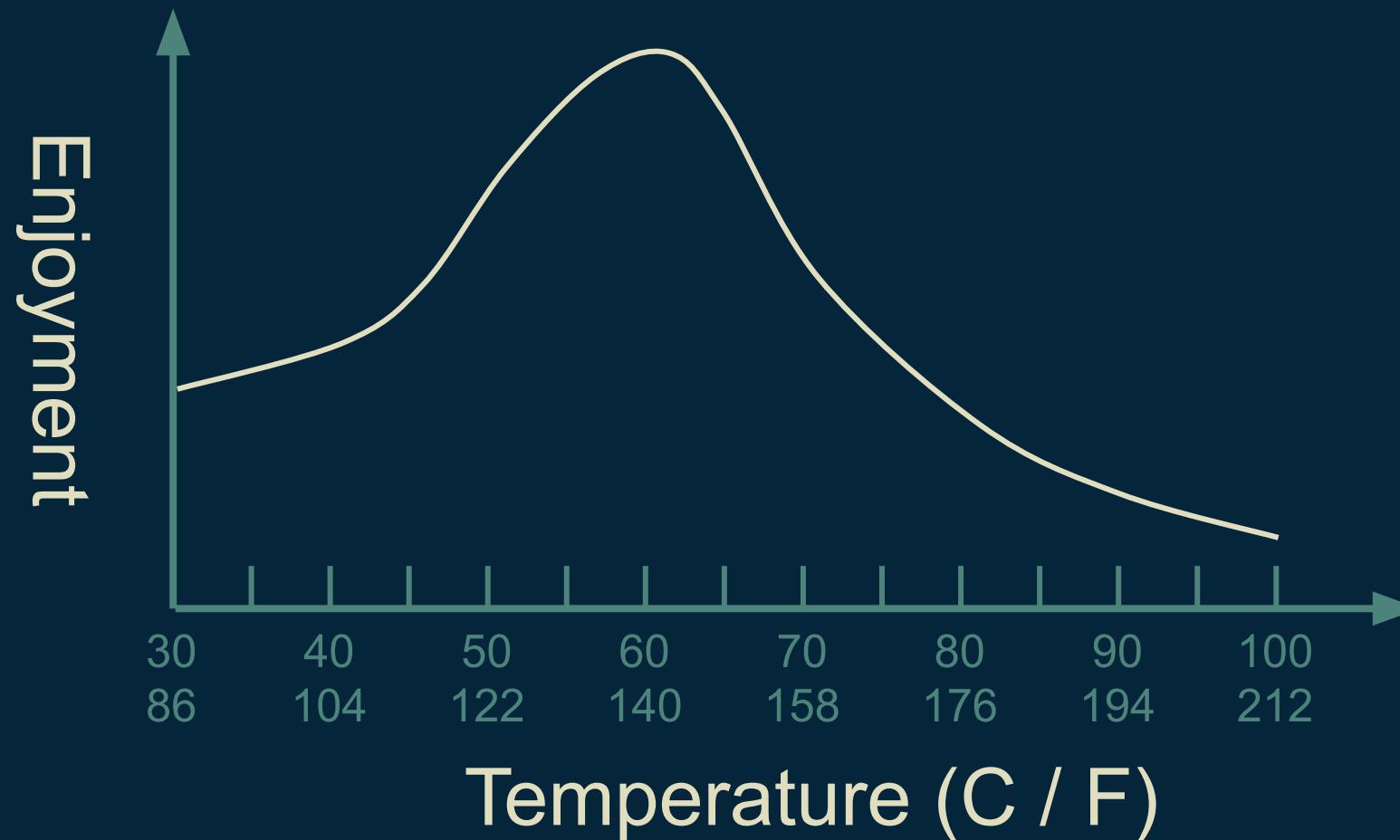




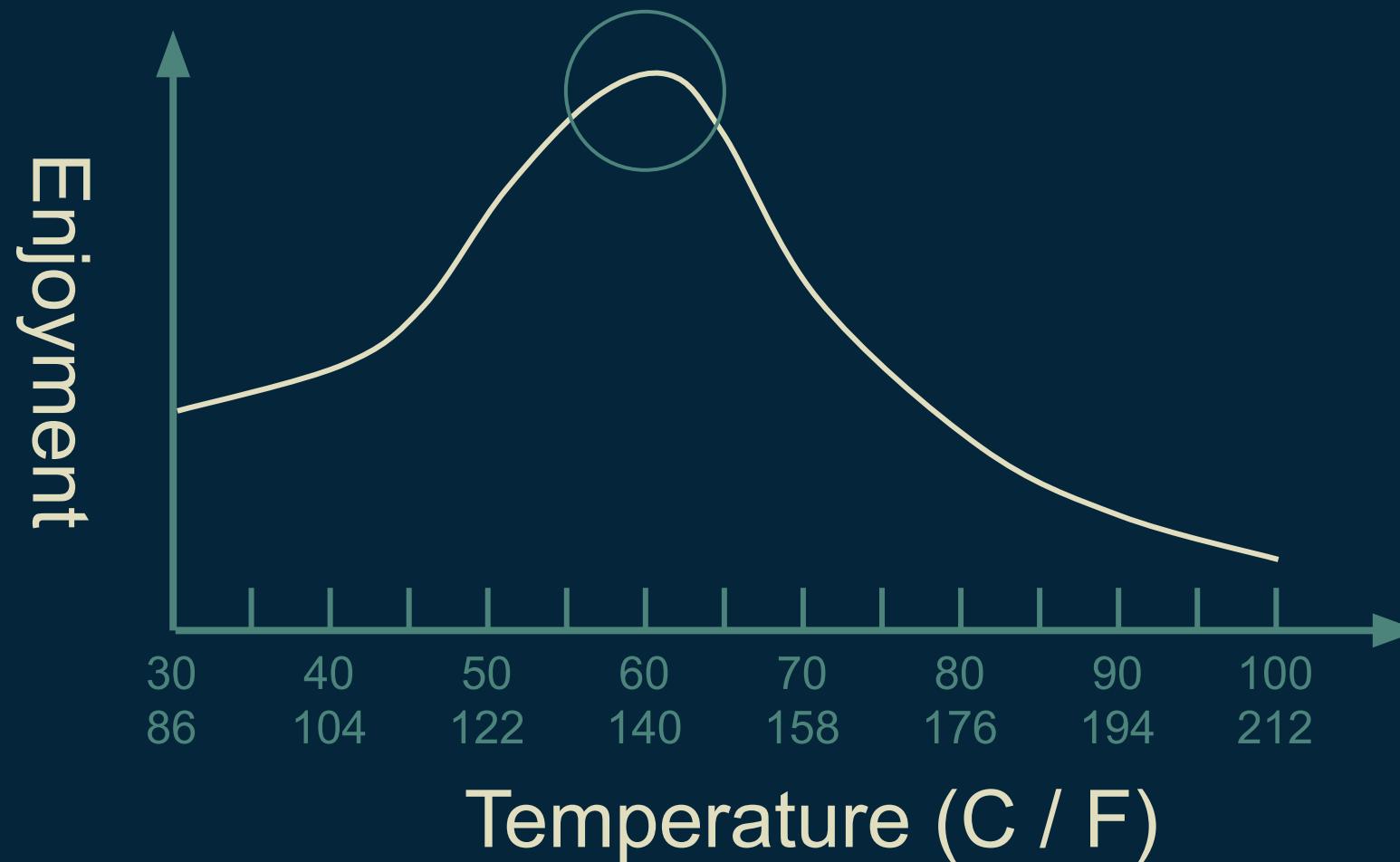


How Optimization for Machine Learning Works

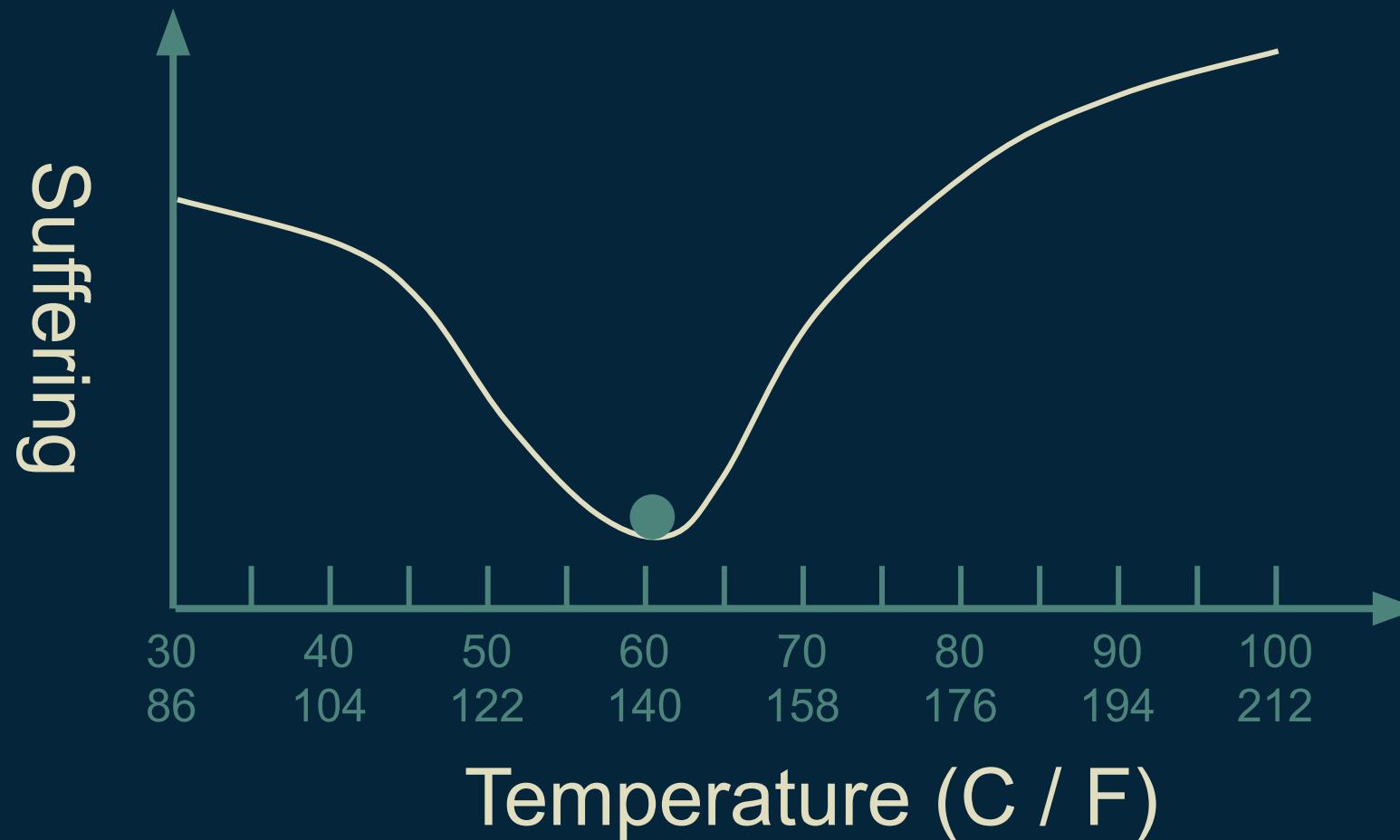
Tea drinking temperature



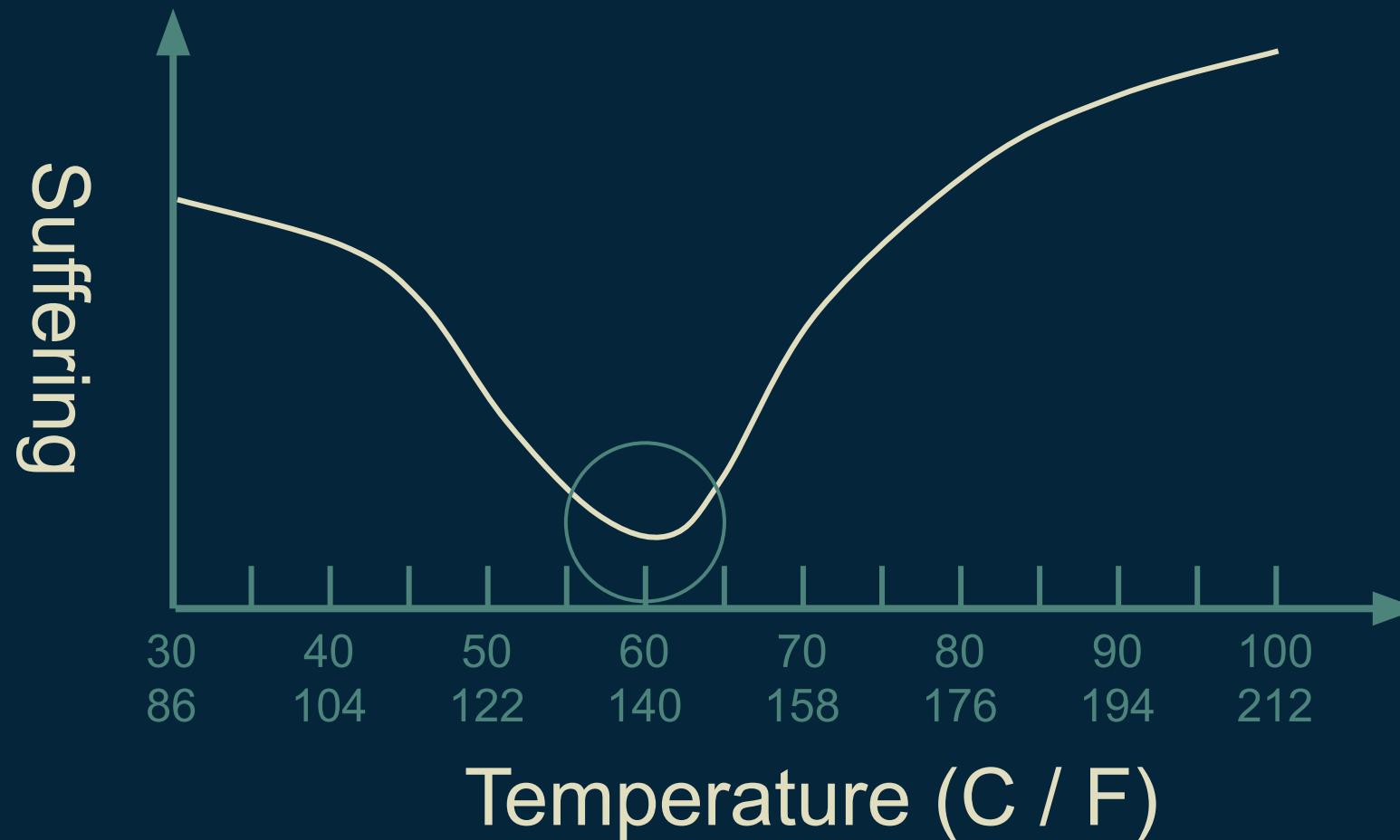
Tea drinking temperature



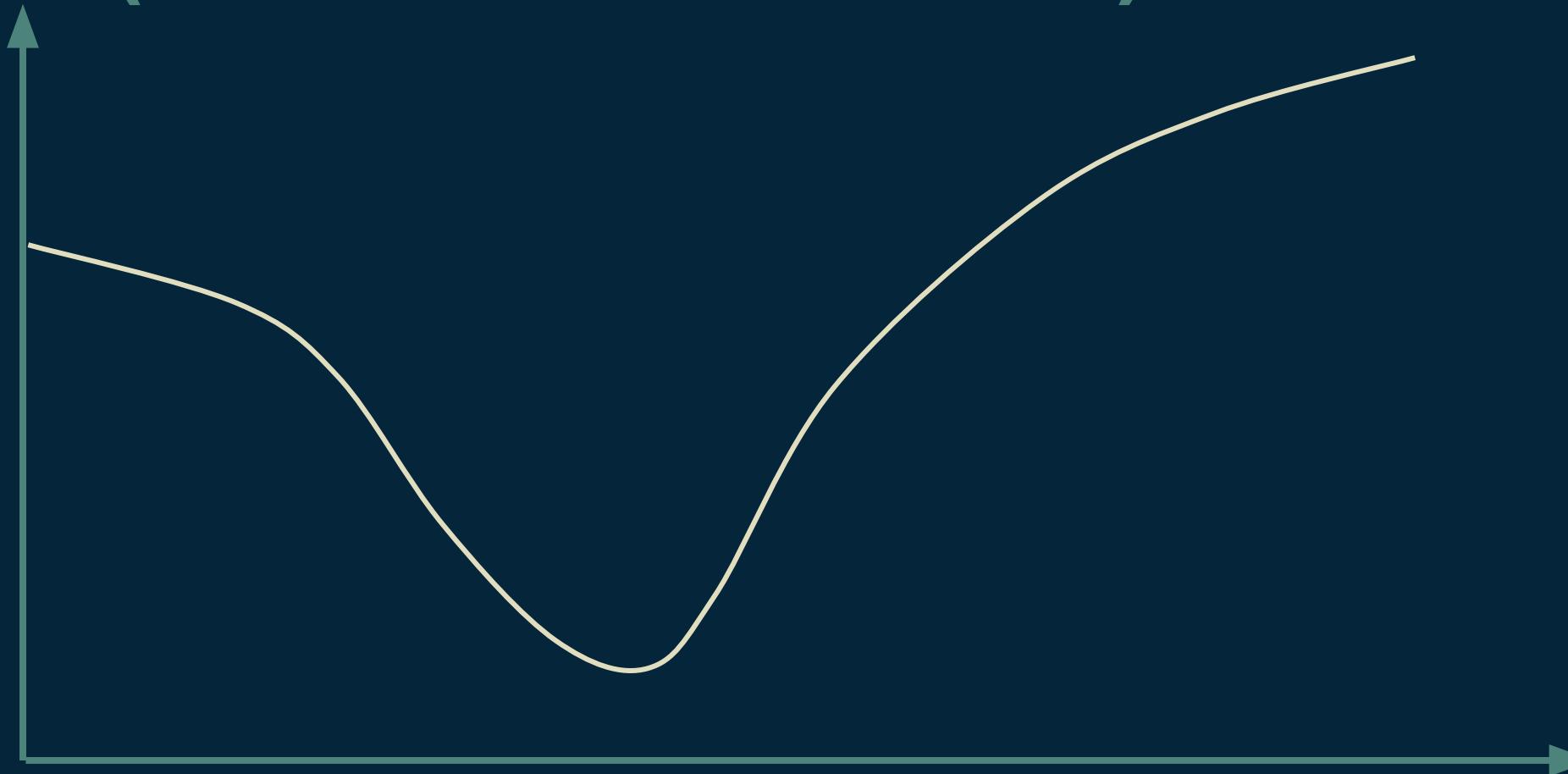
Tea drinking temperature



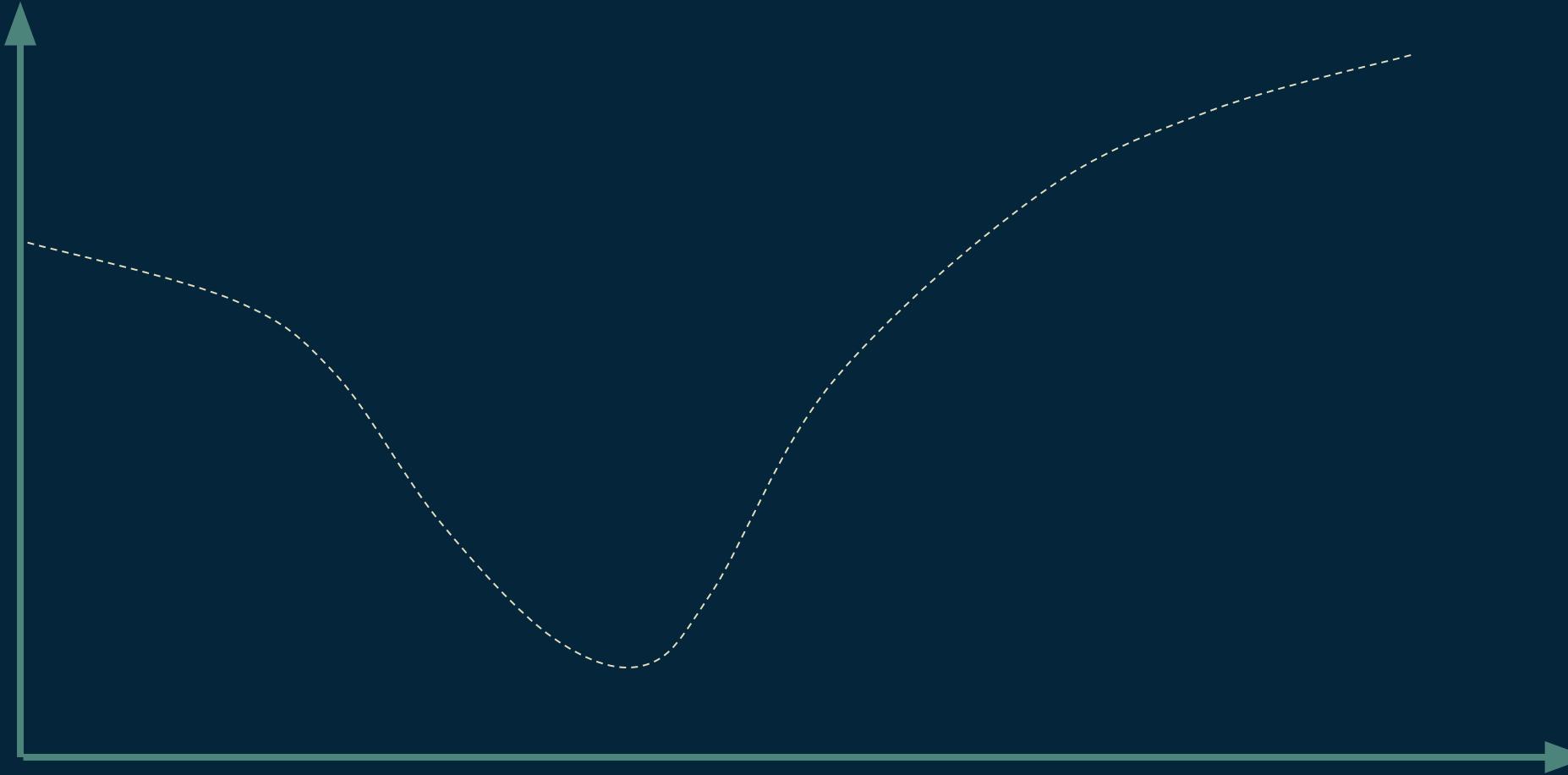
Tea drinking temperature



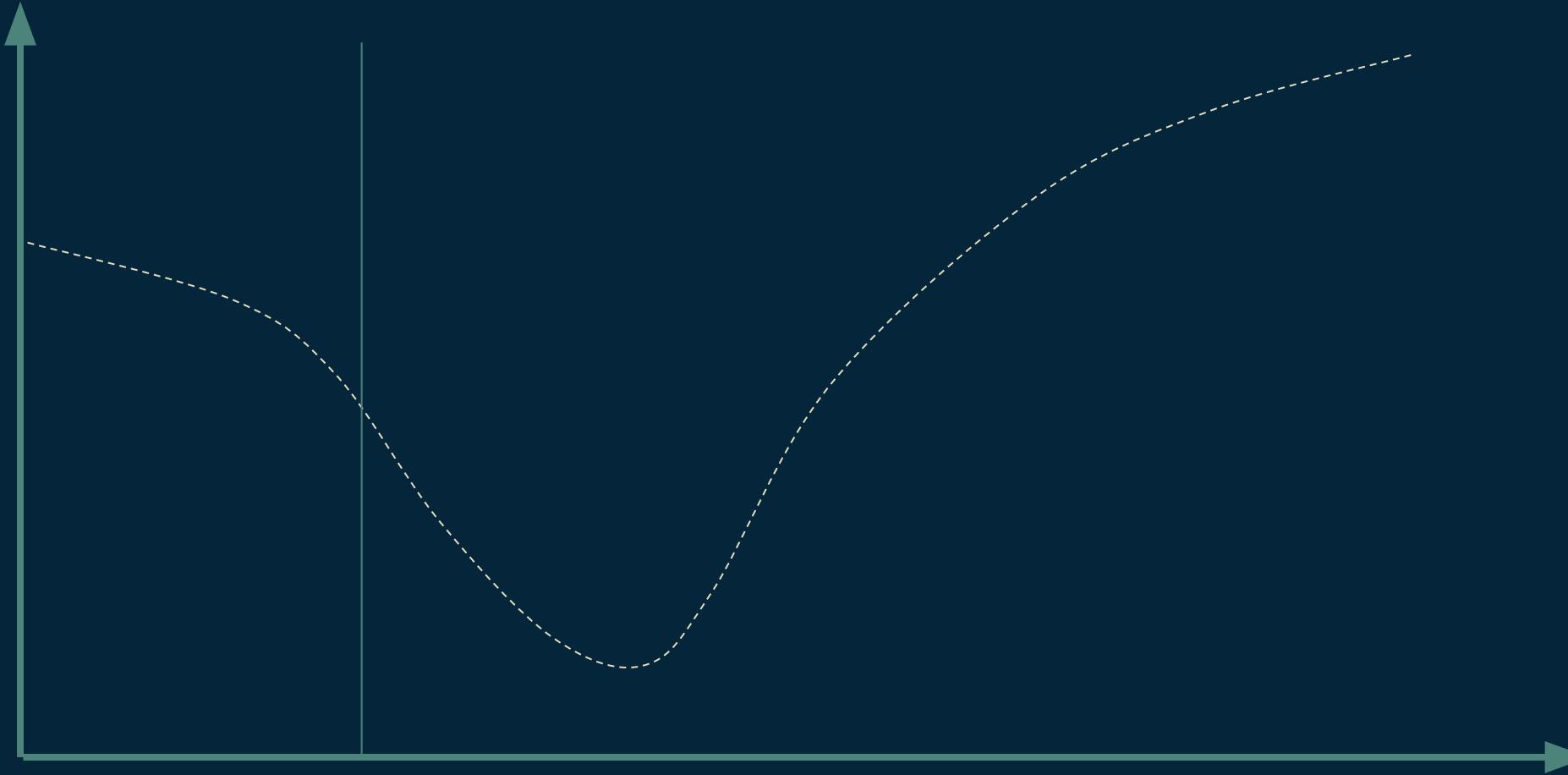
Pick the lowest point (Exhaustive search)



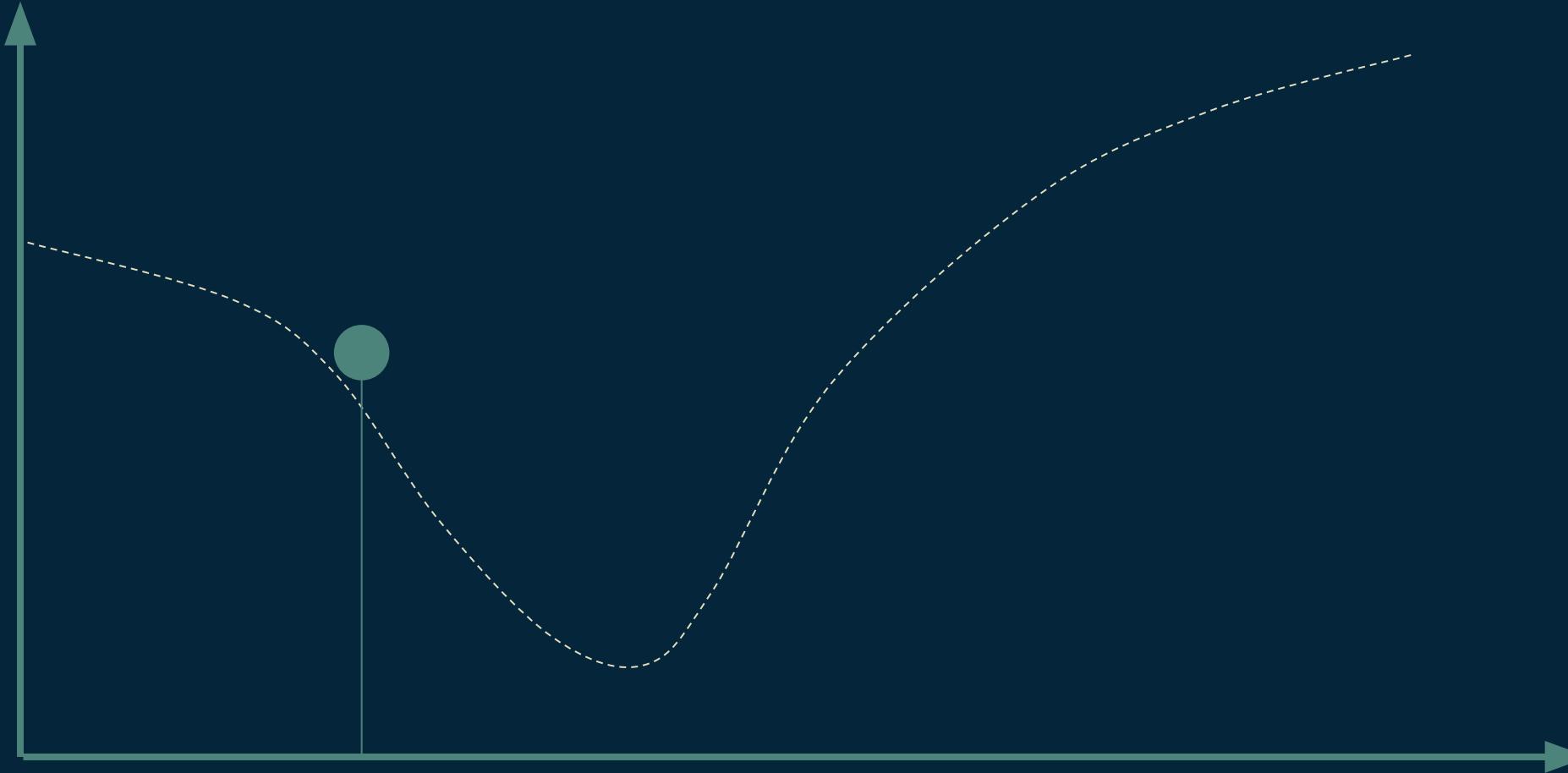
Exhaustive search



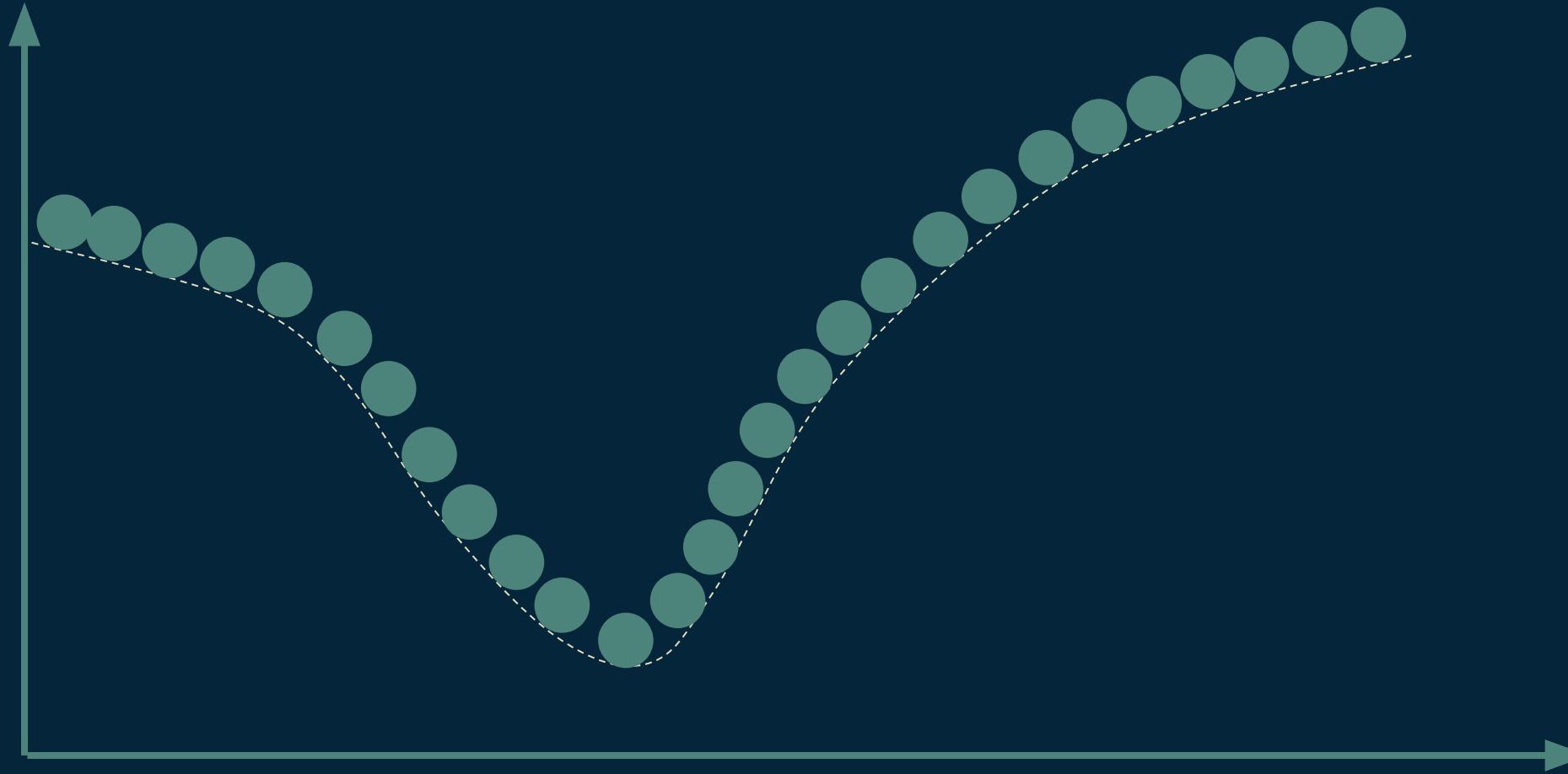
Exhaustive search



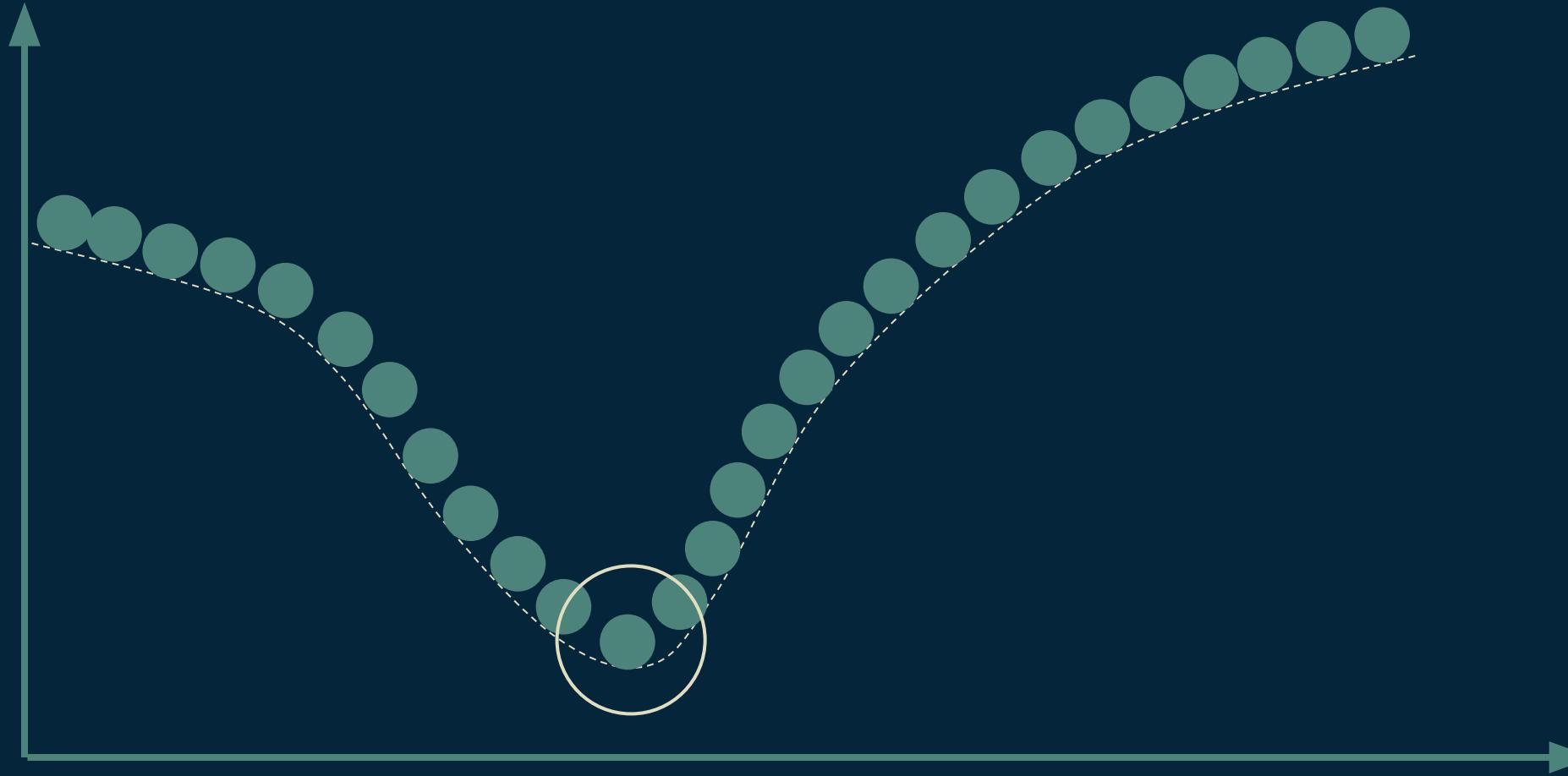
Exhaustive search



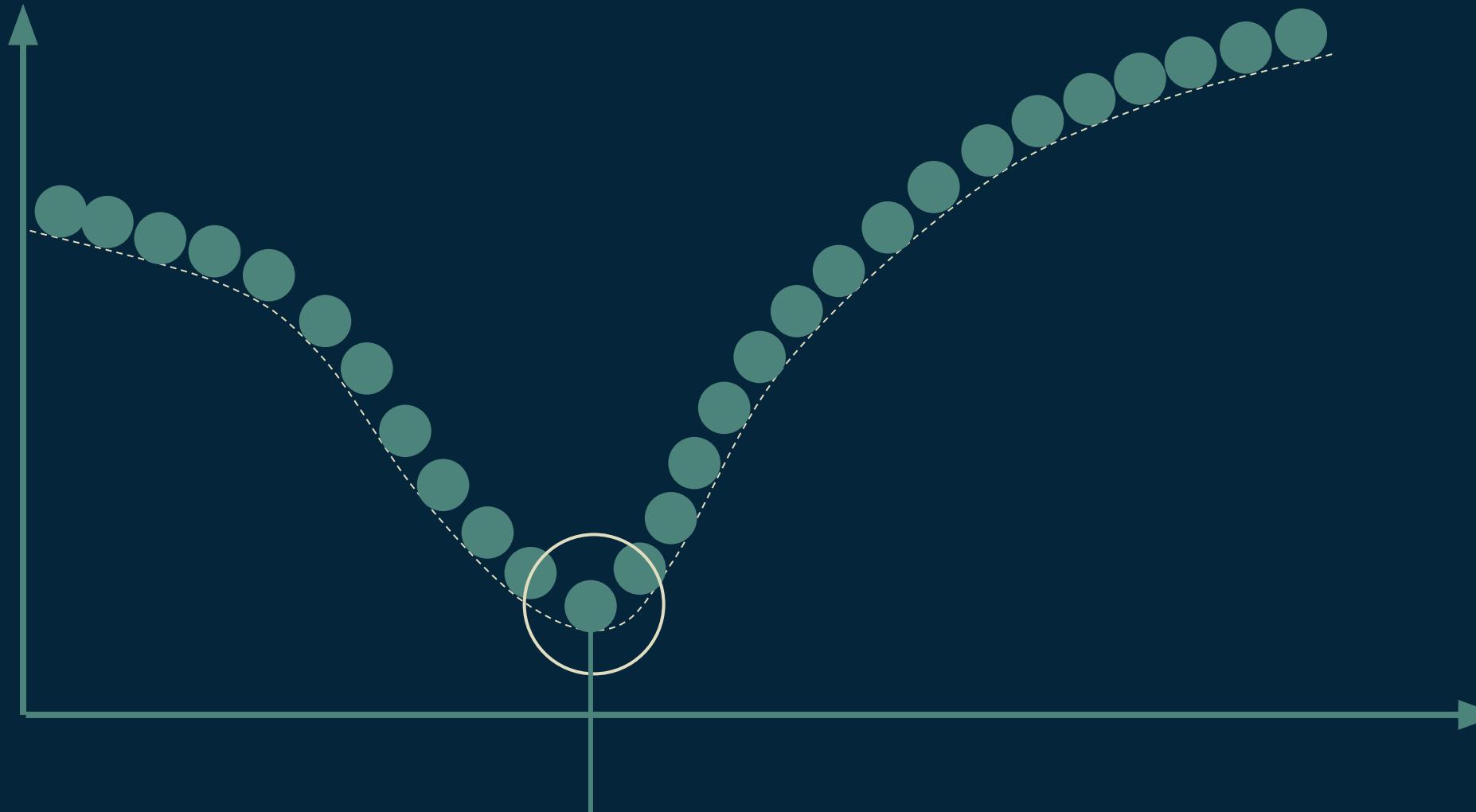
Exhaustive search



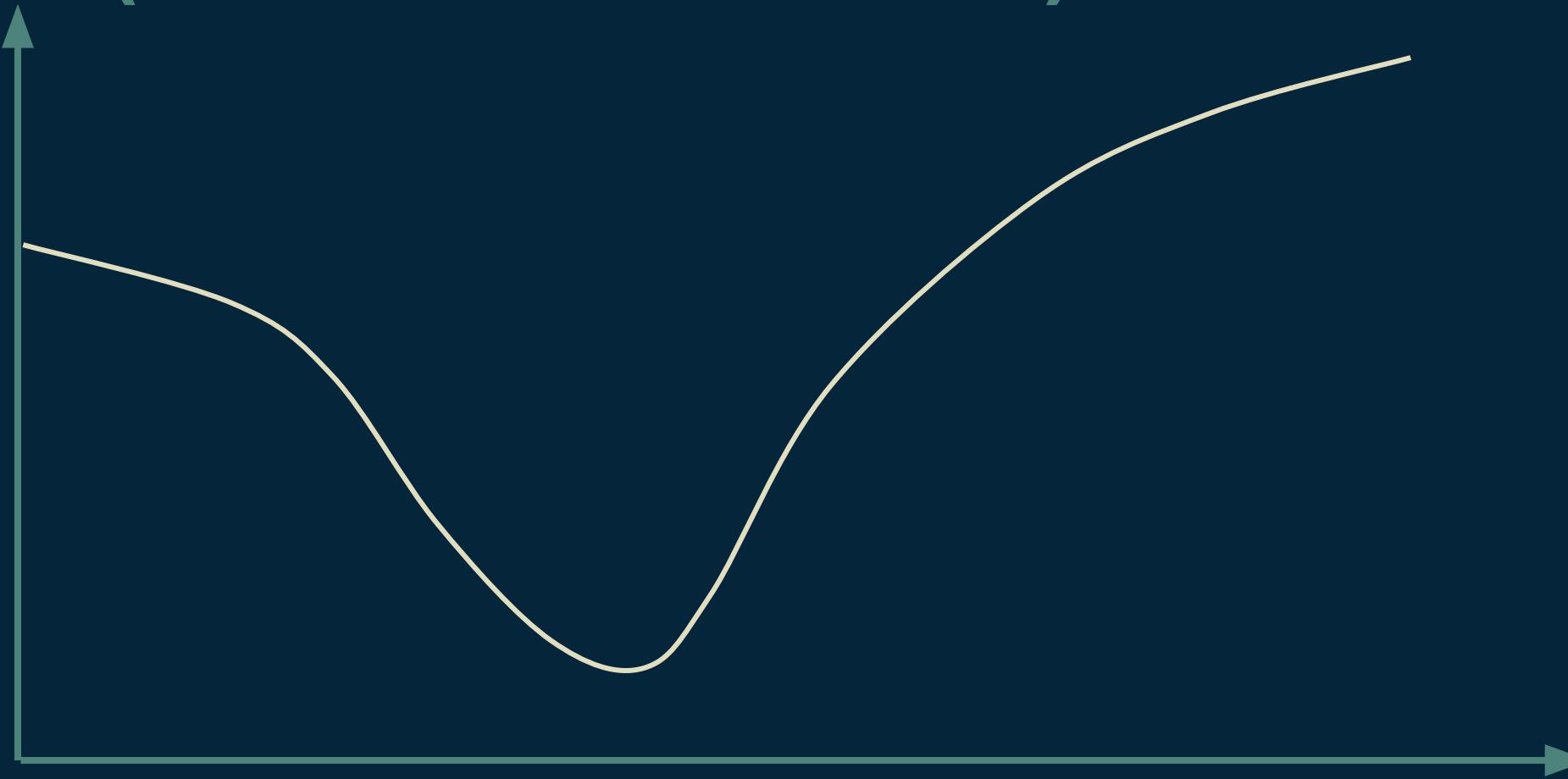
Exhaustive search



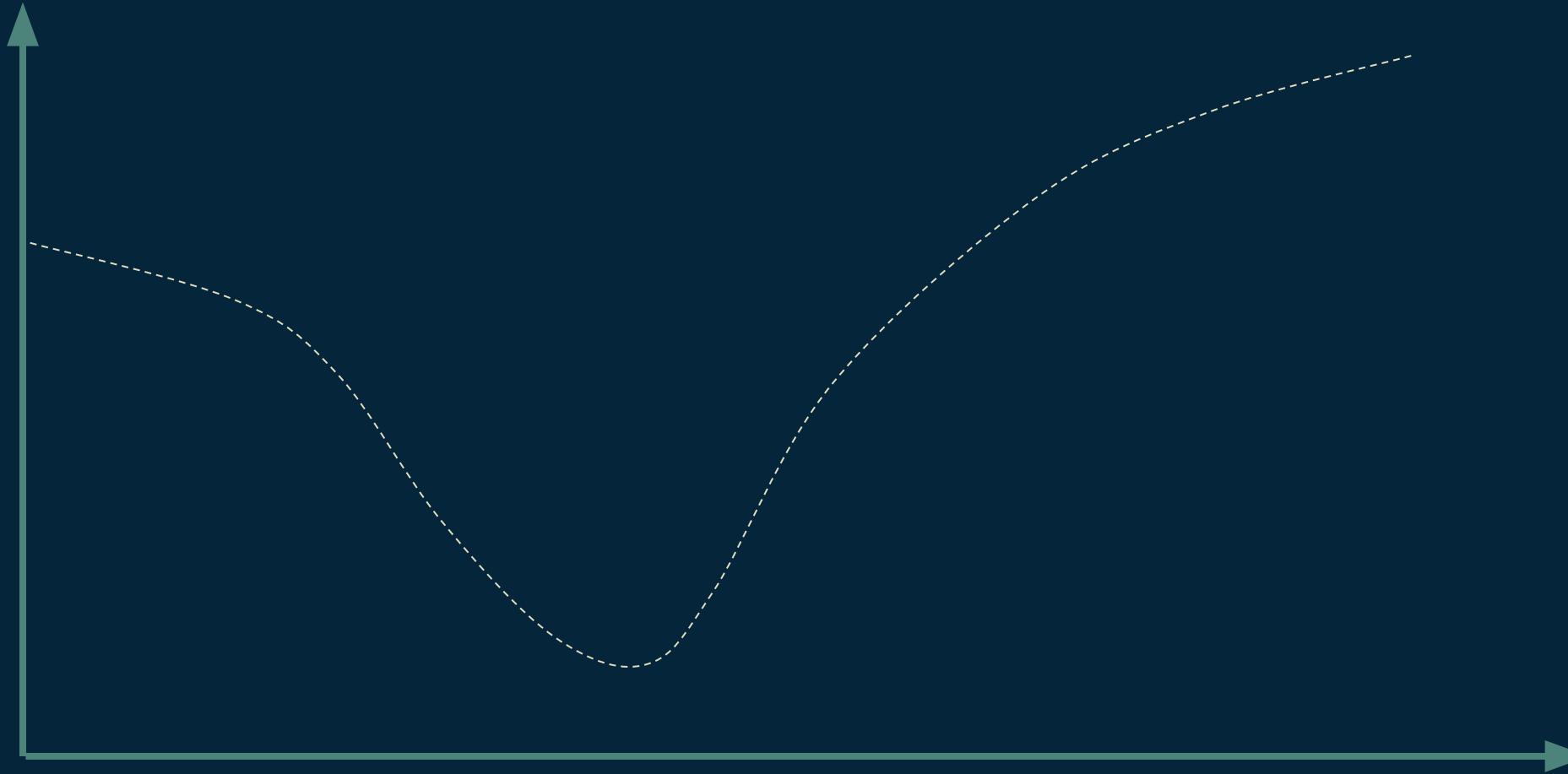
Exhaustive search



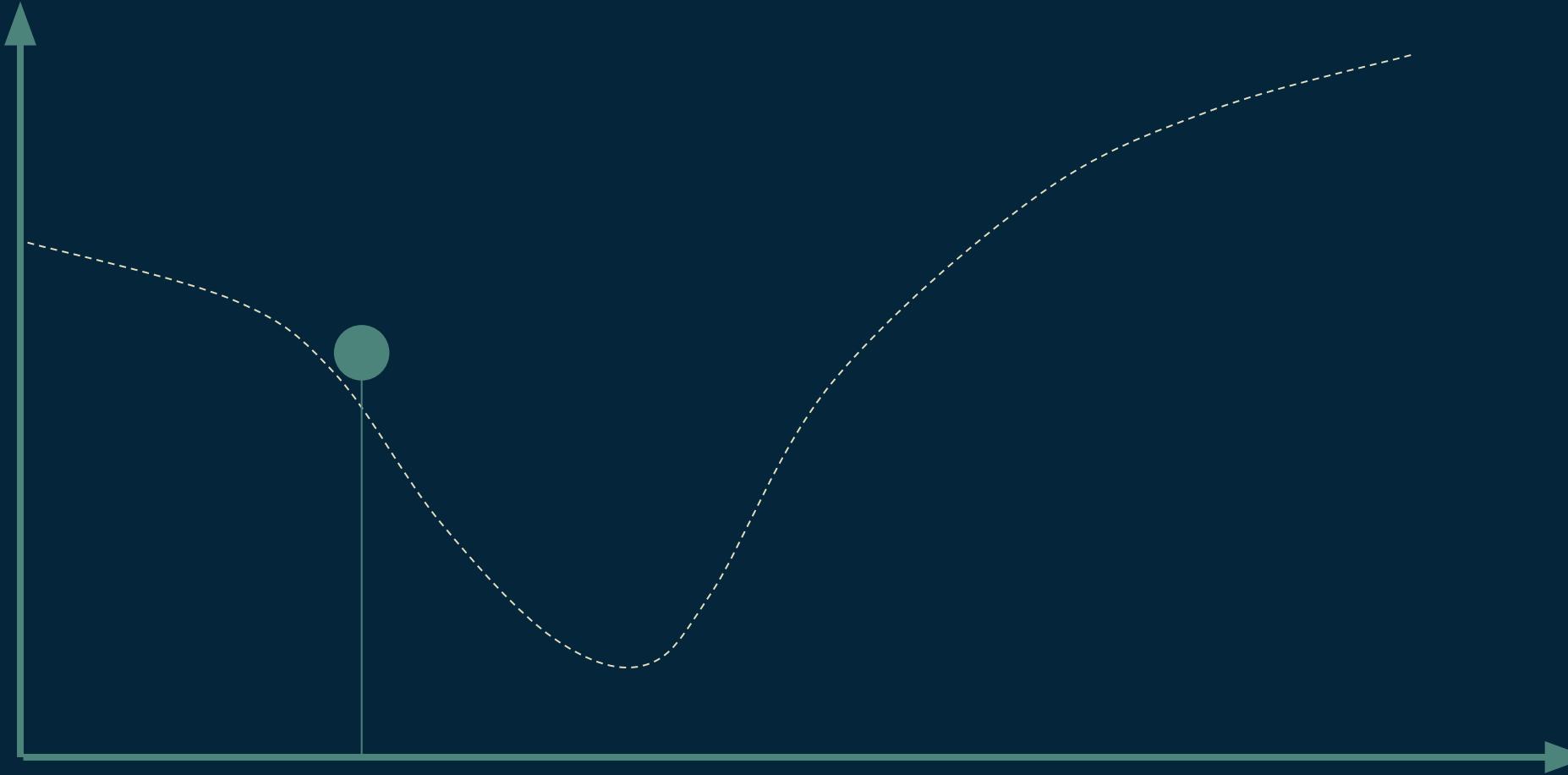
Let the marble roll downhill (Gradient descent)



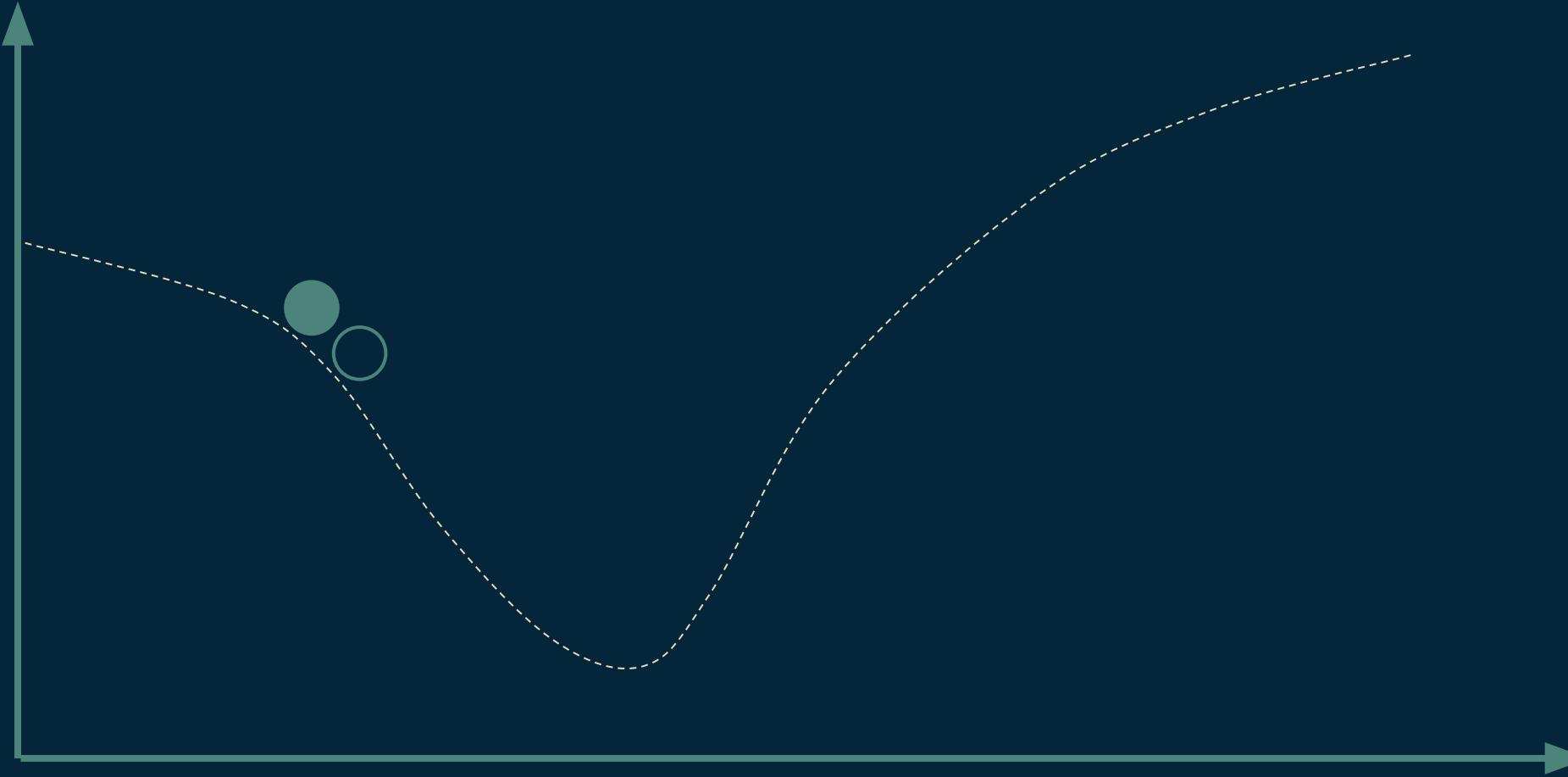
Gradient descent



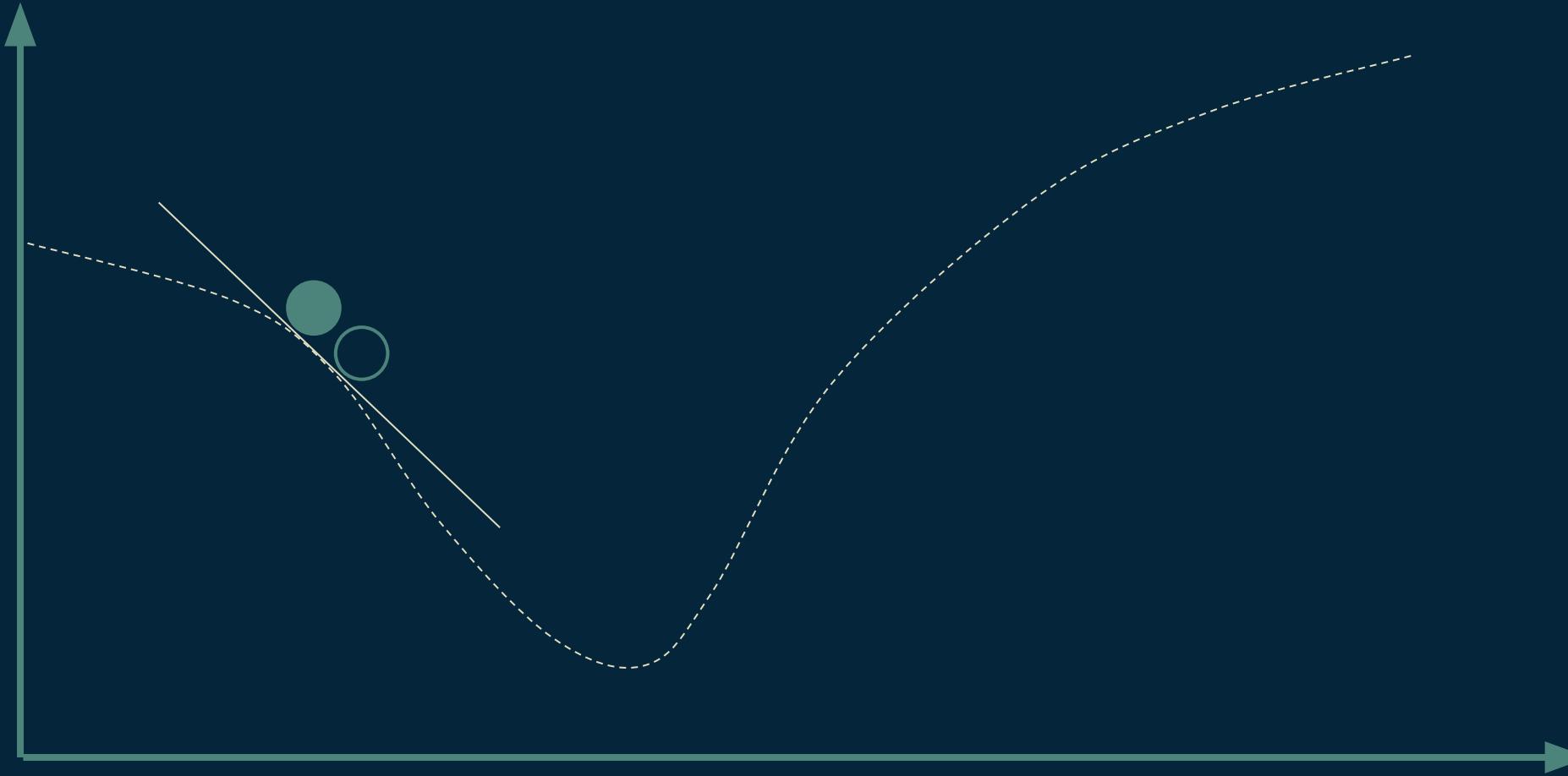
Gradient descent



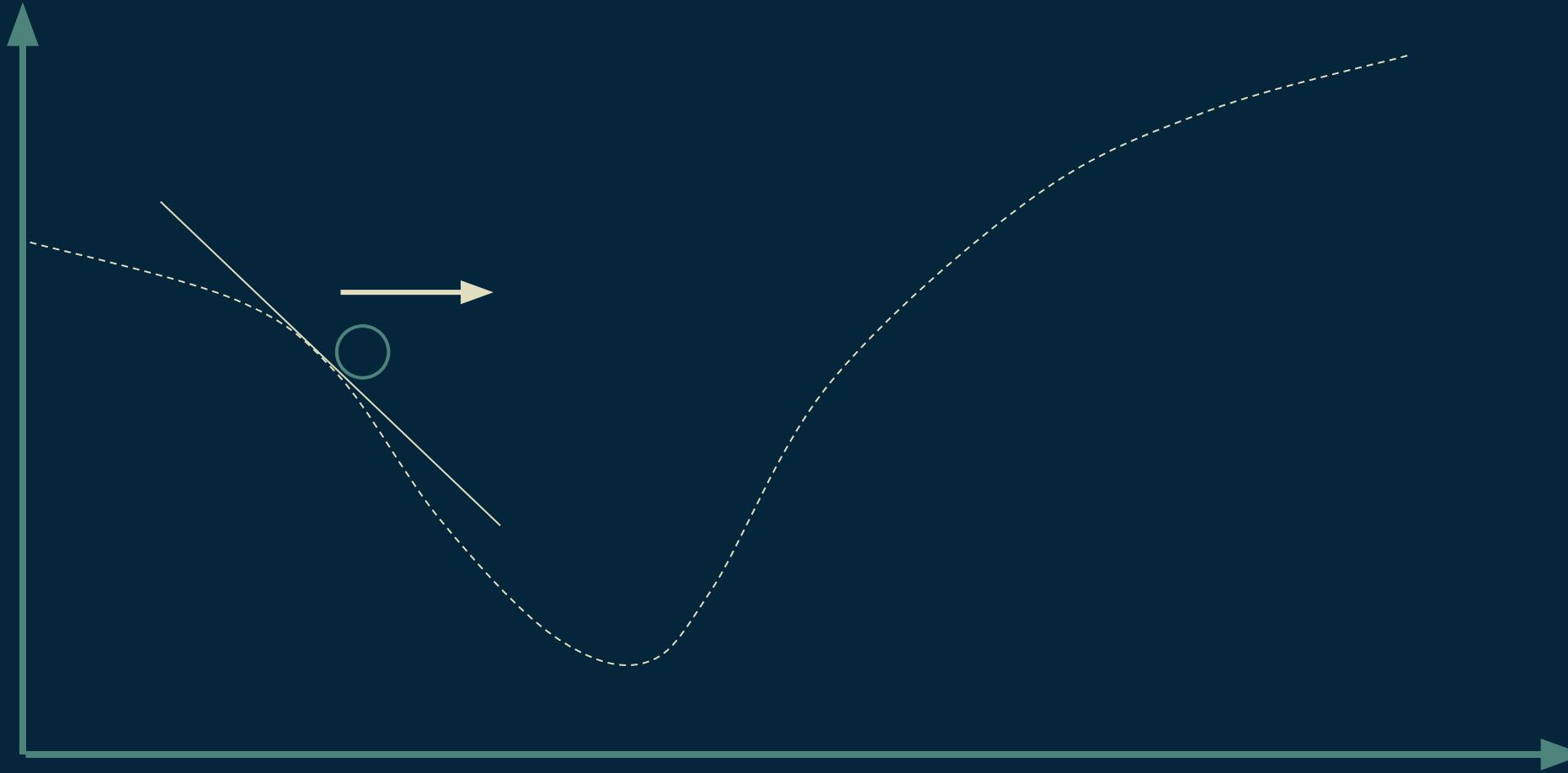
Gradient descent



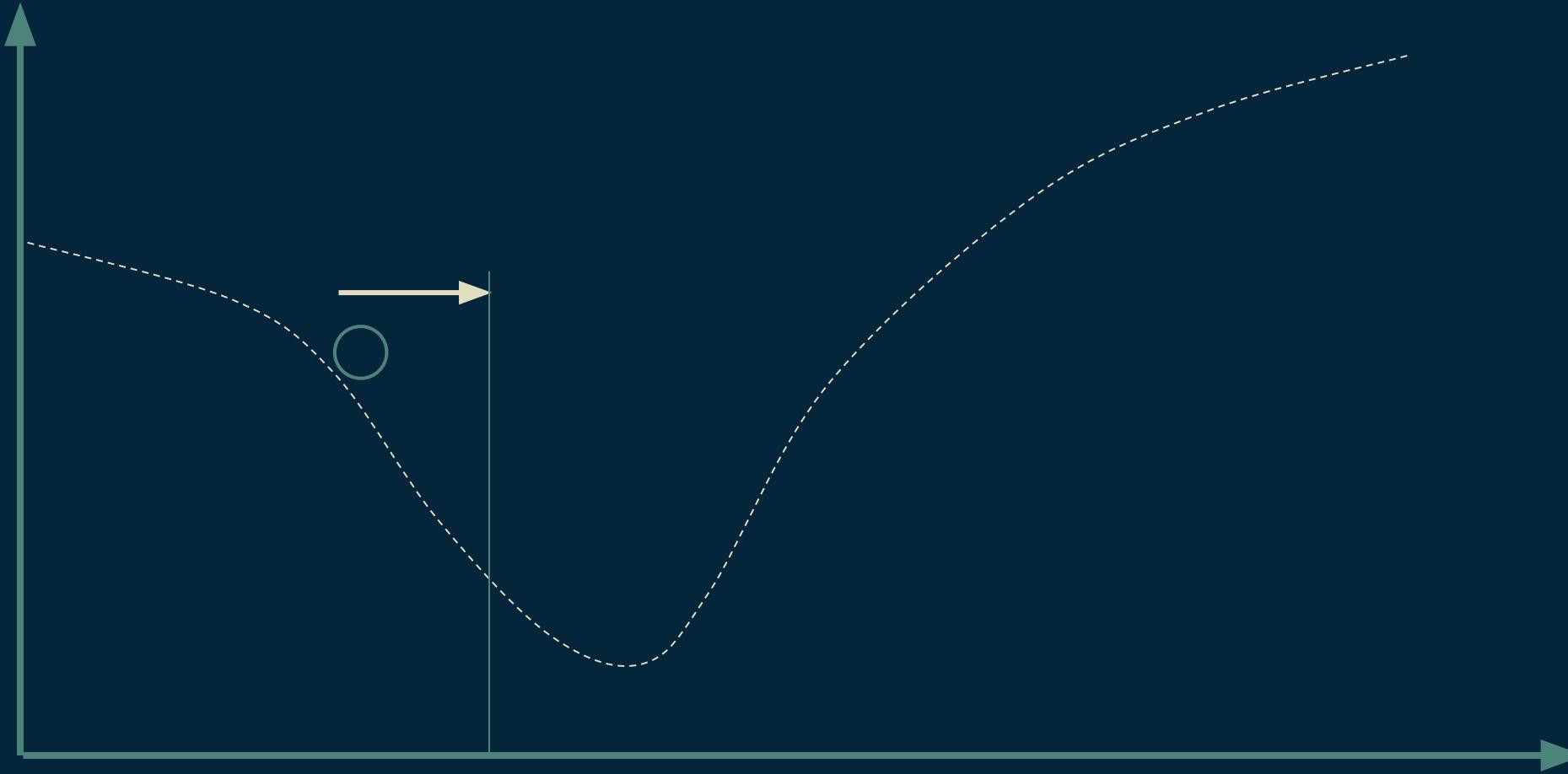
Gradient descent



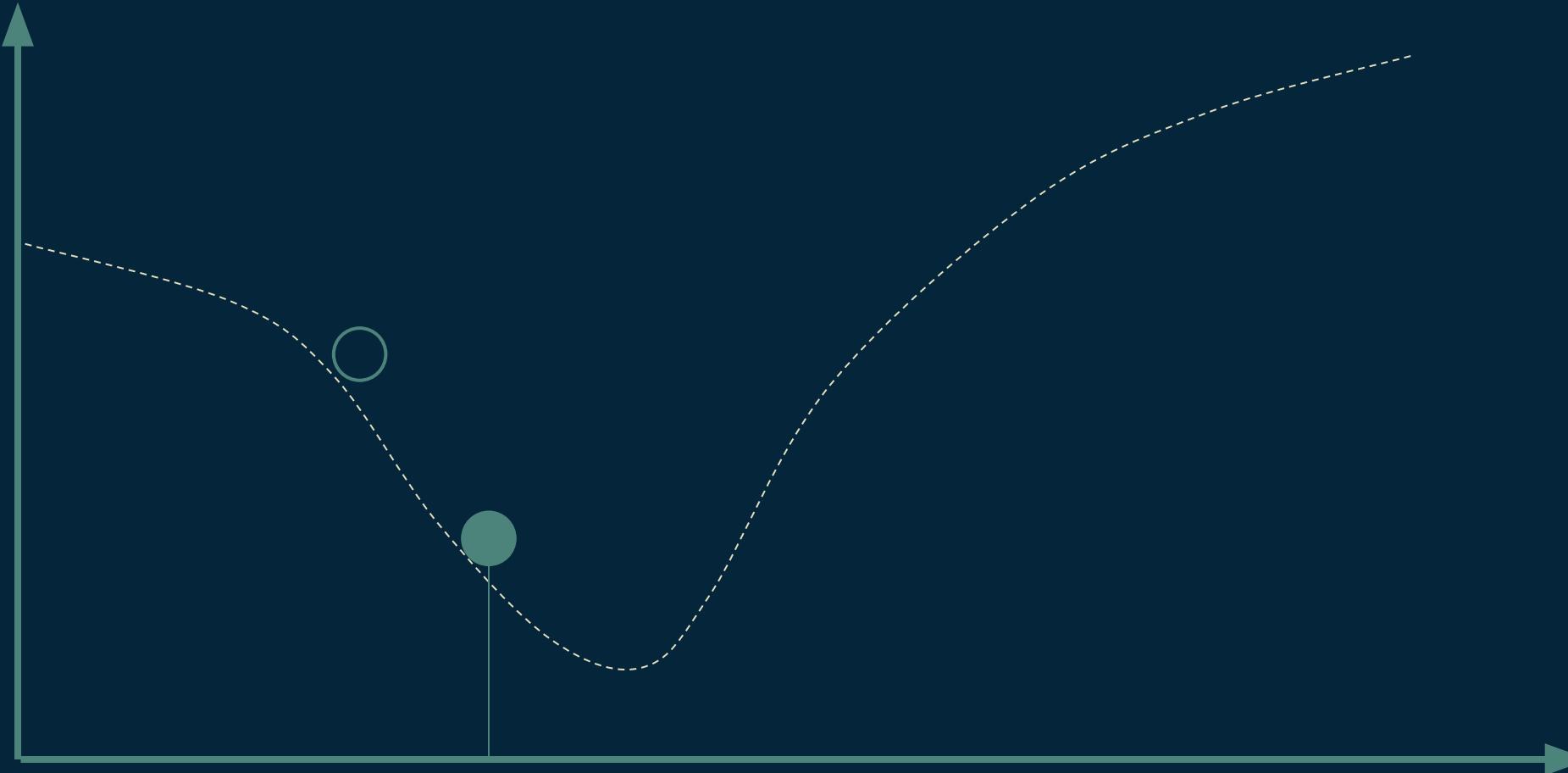
Gradient descent



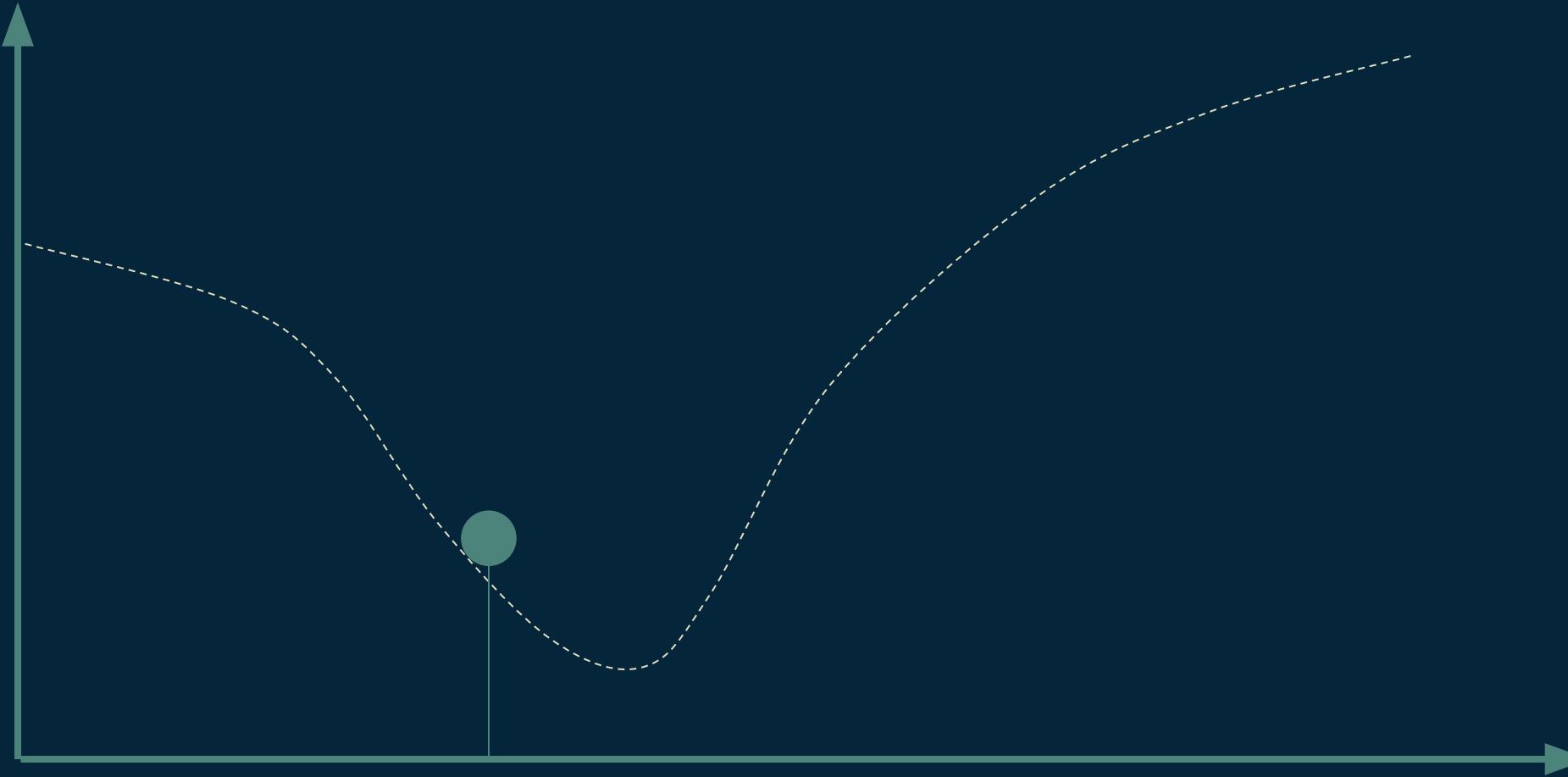
Gradient descent



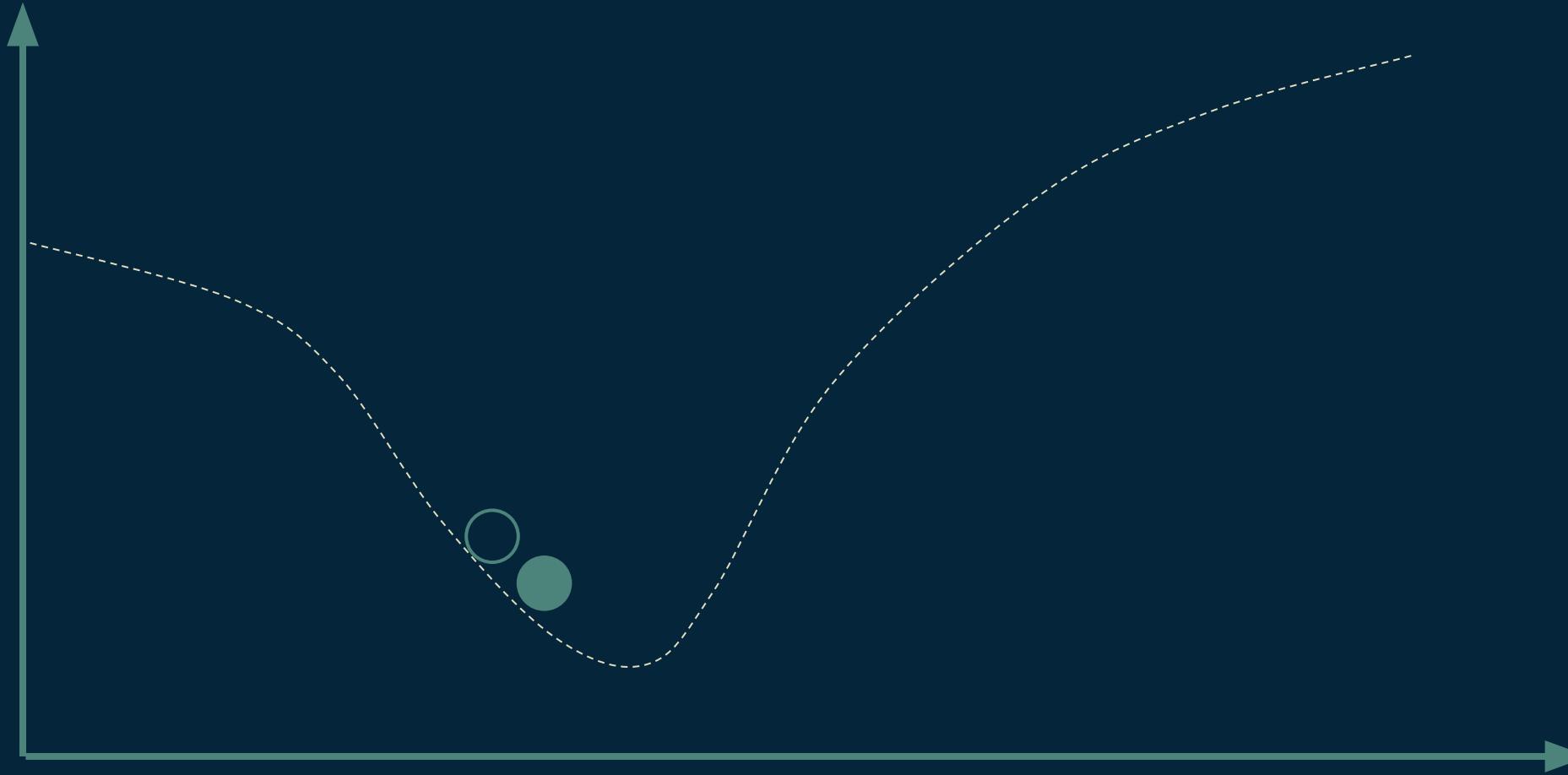
Gradient descent



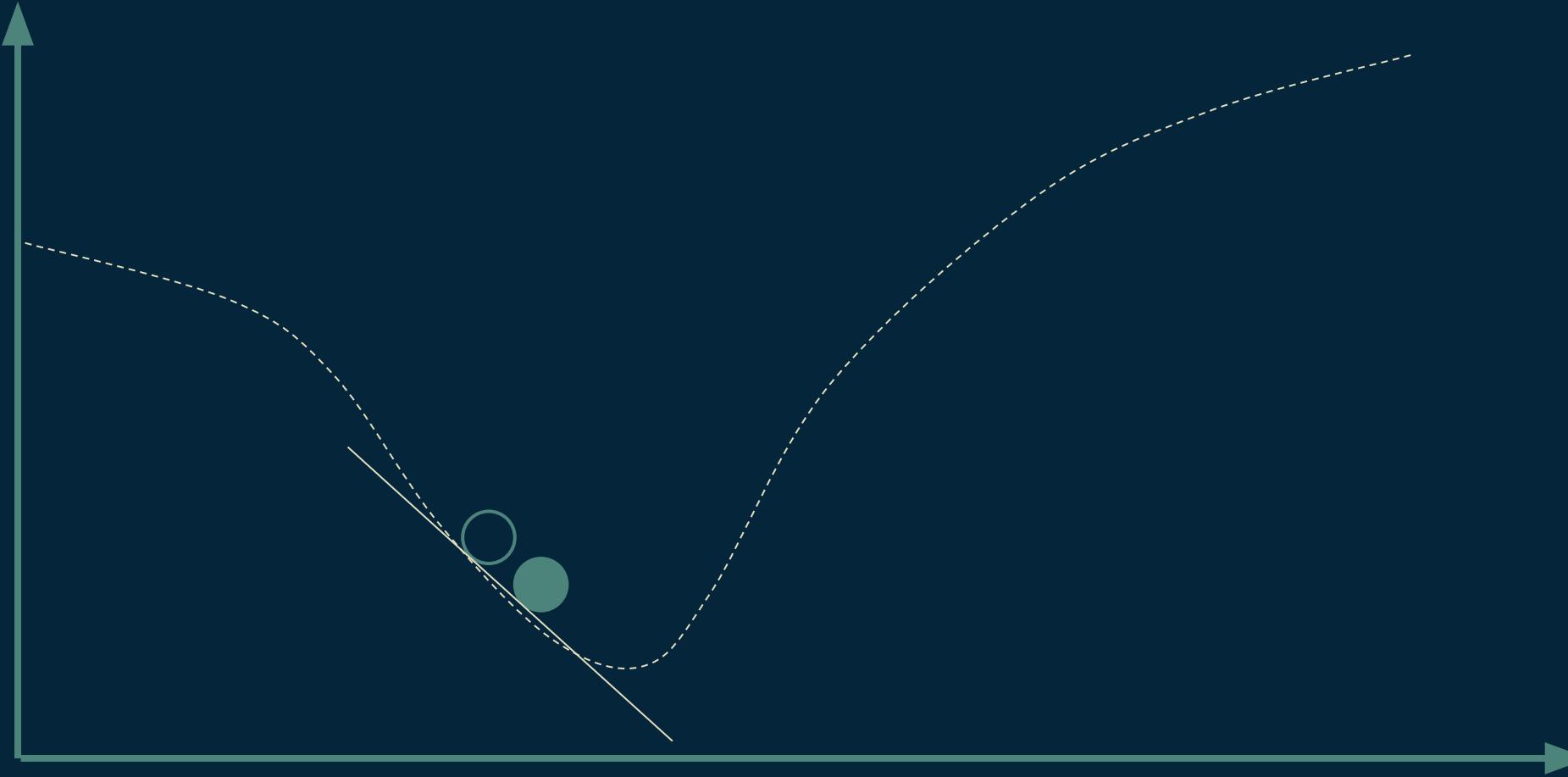
Gradient descent



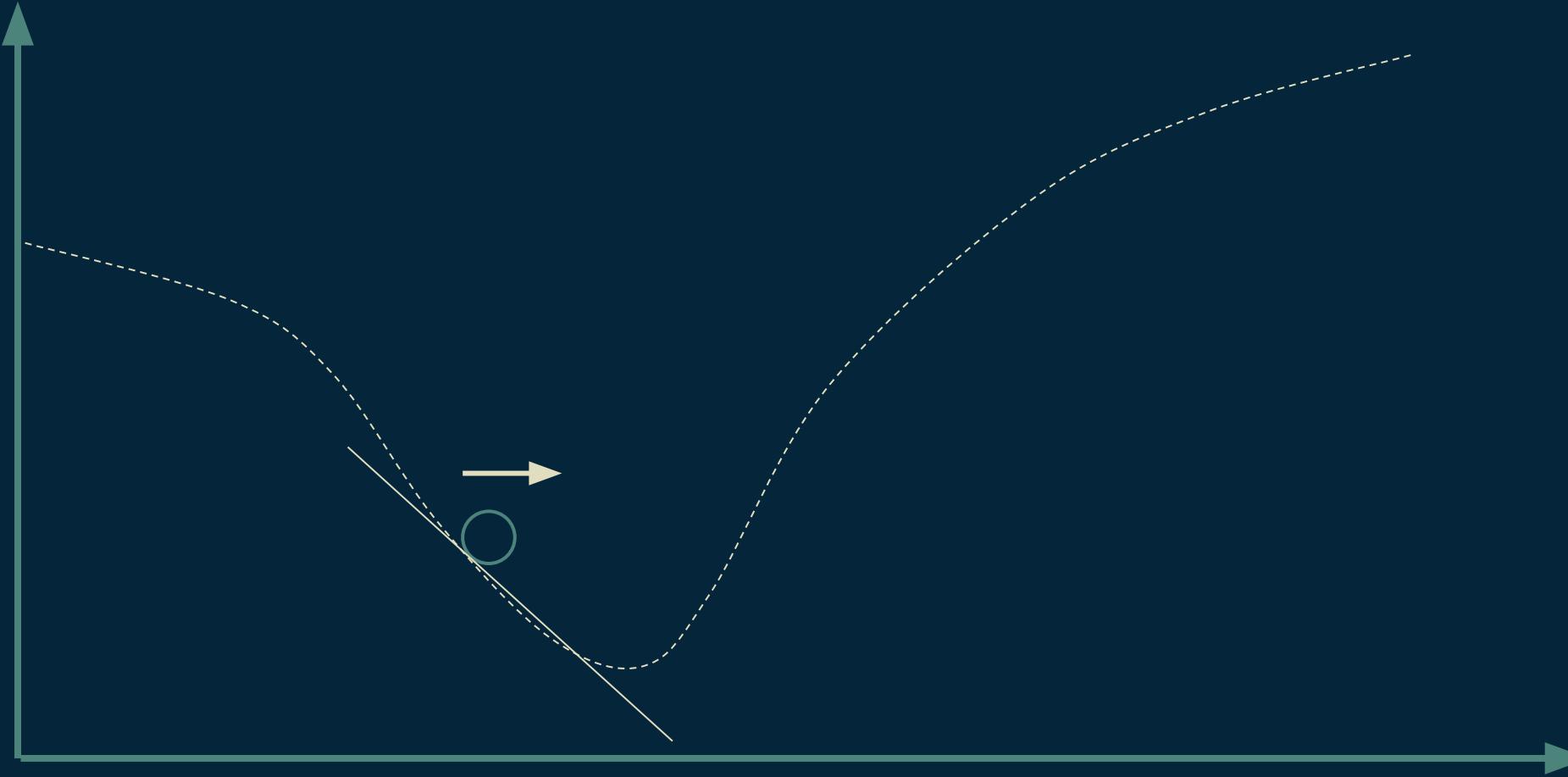
Gradient descent



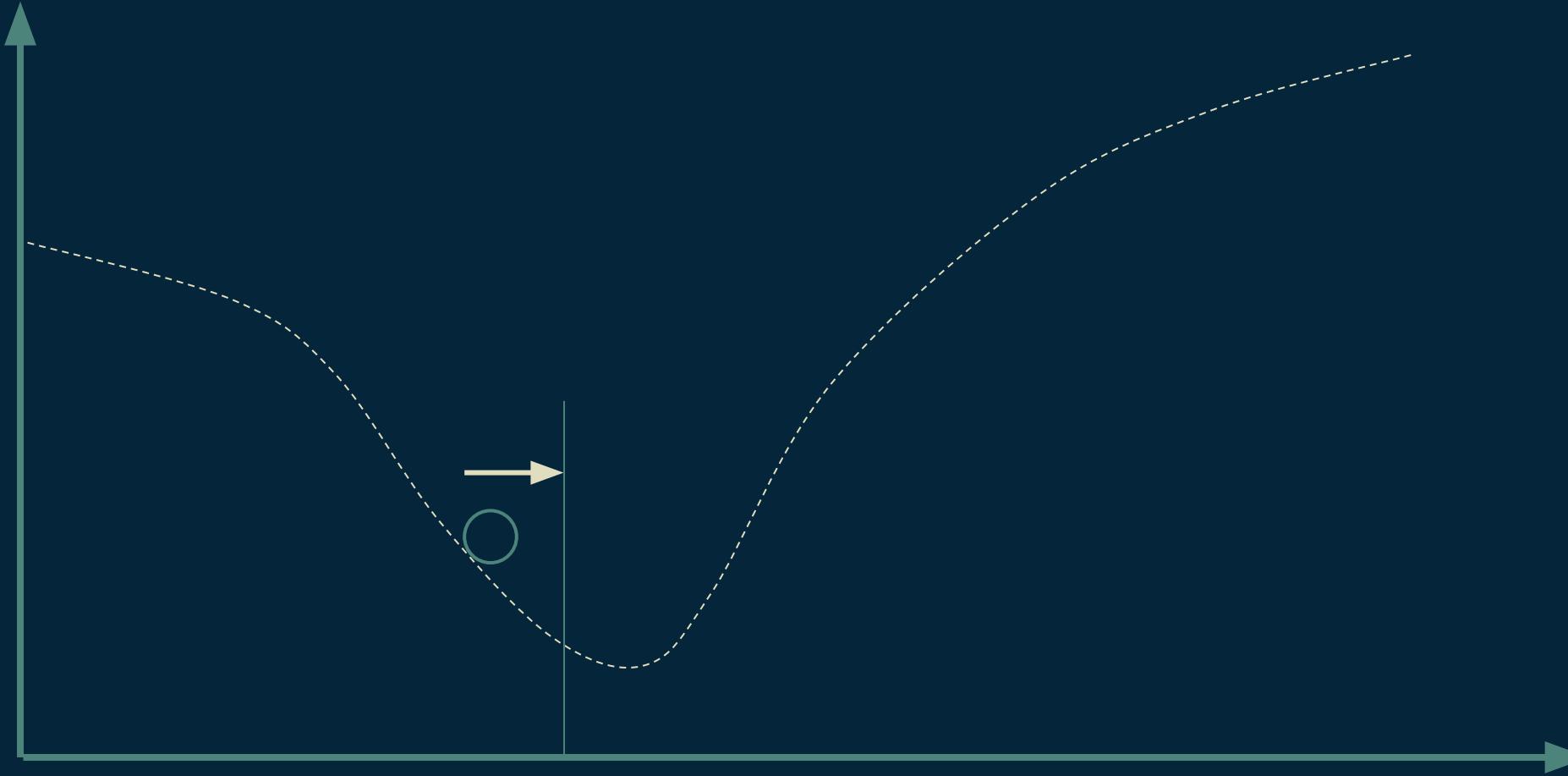
Gradient descent



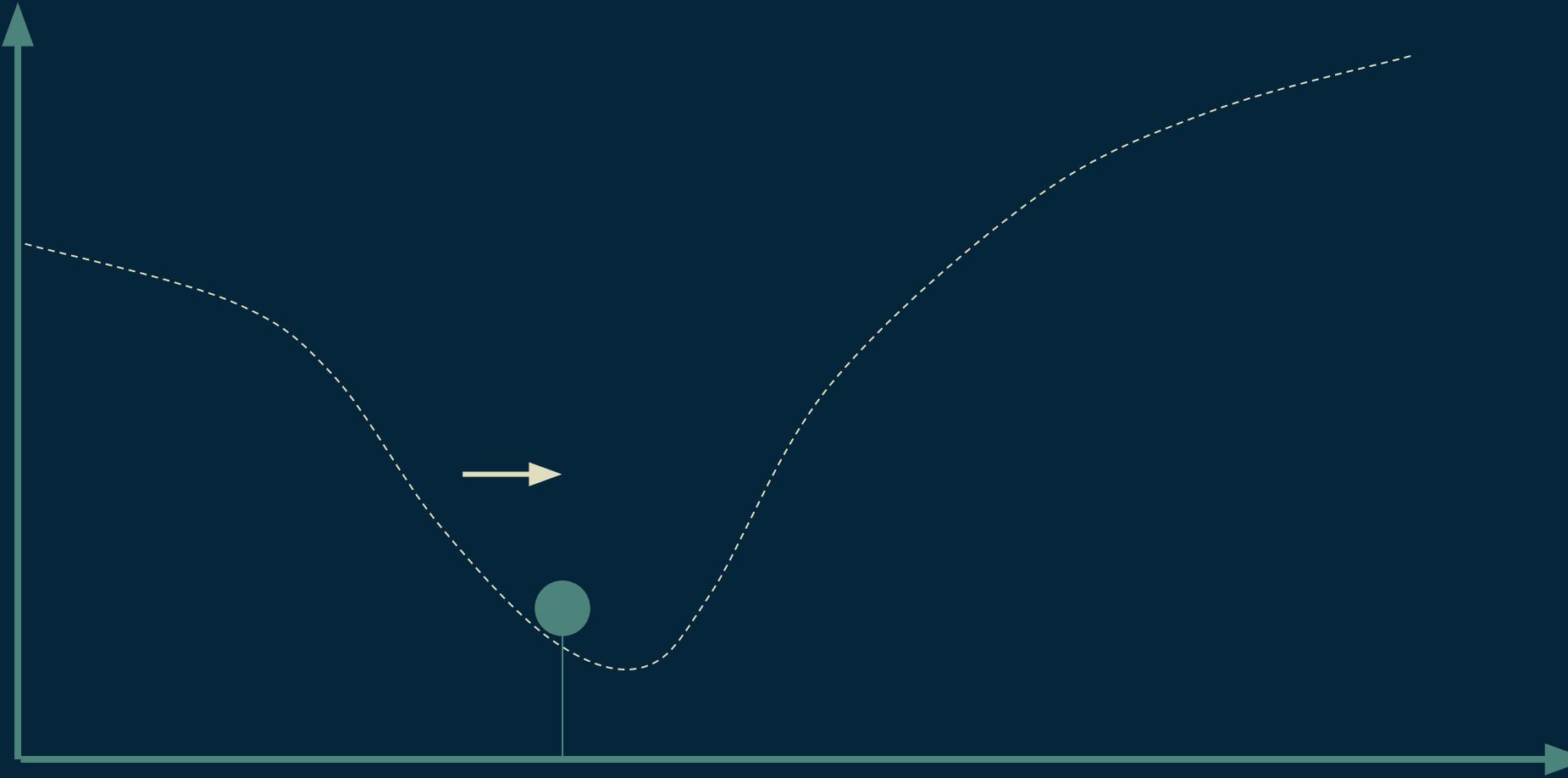
Gradient descent



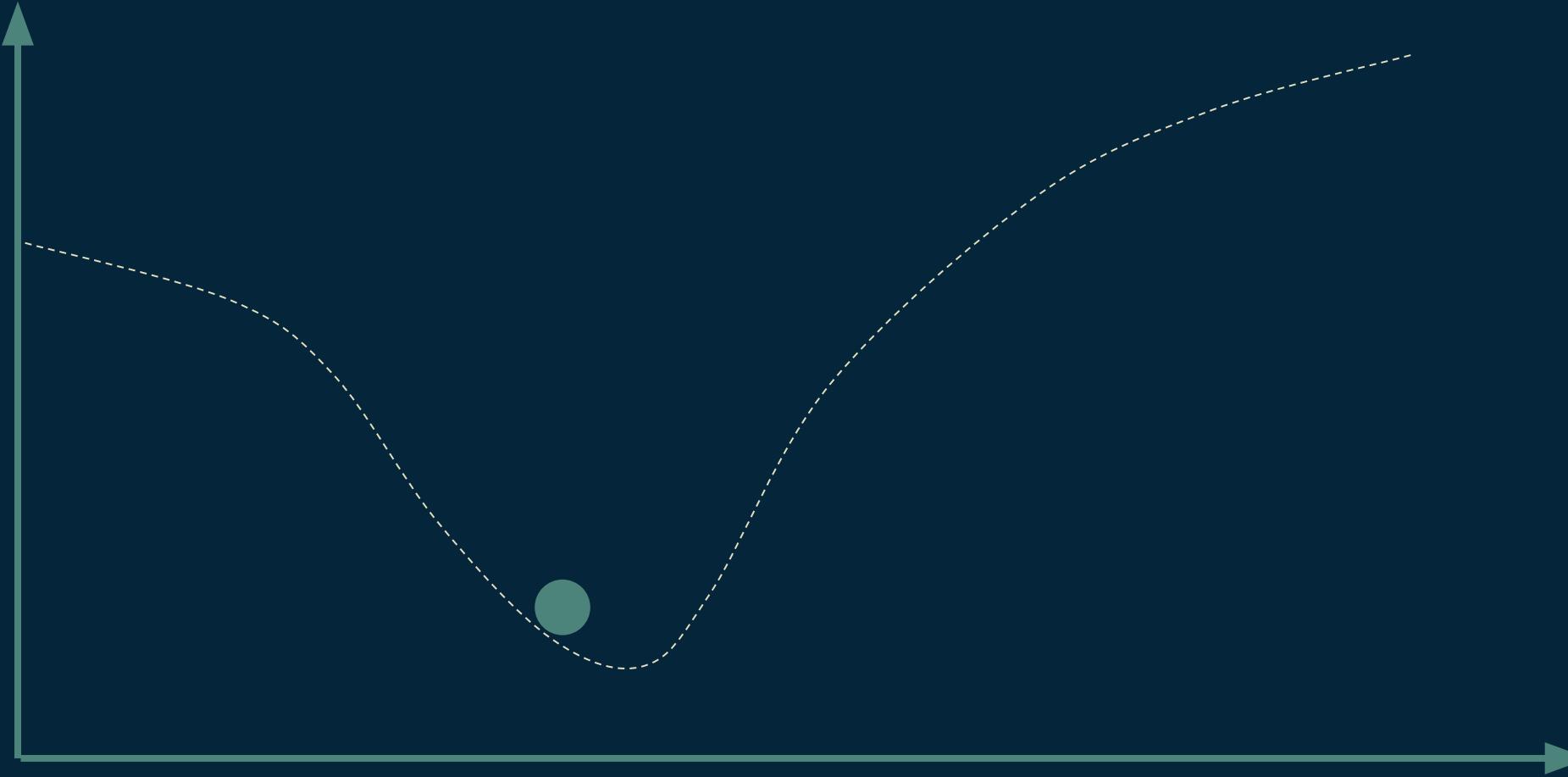
Gradient descent



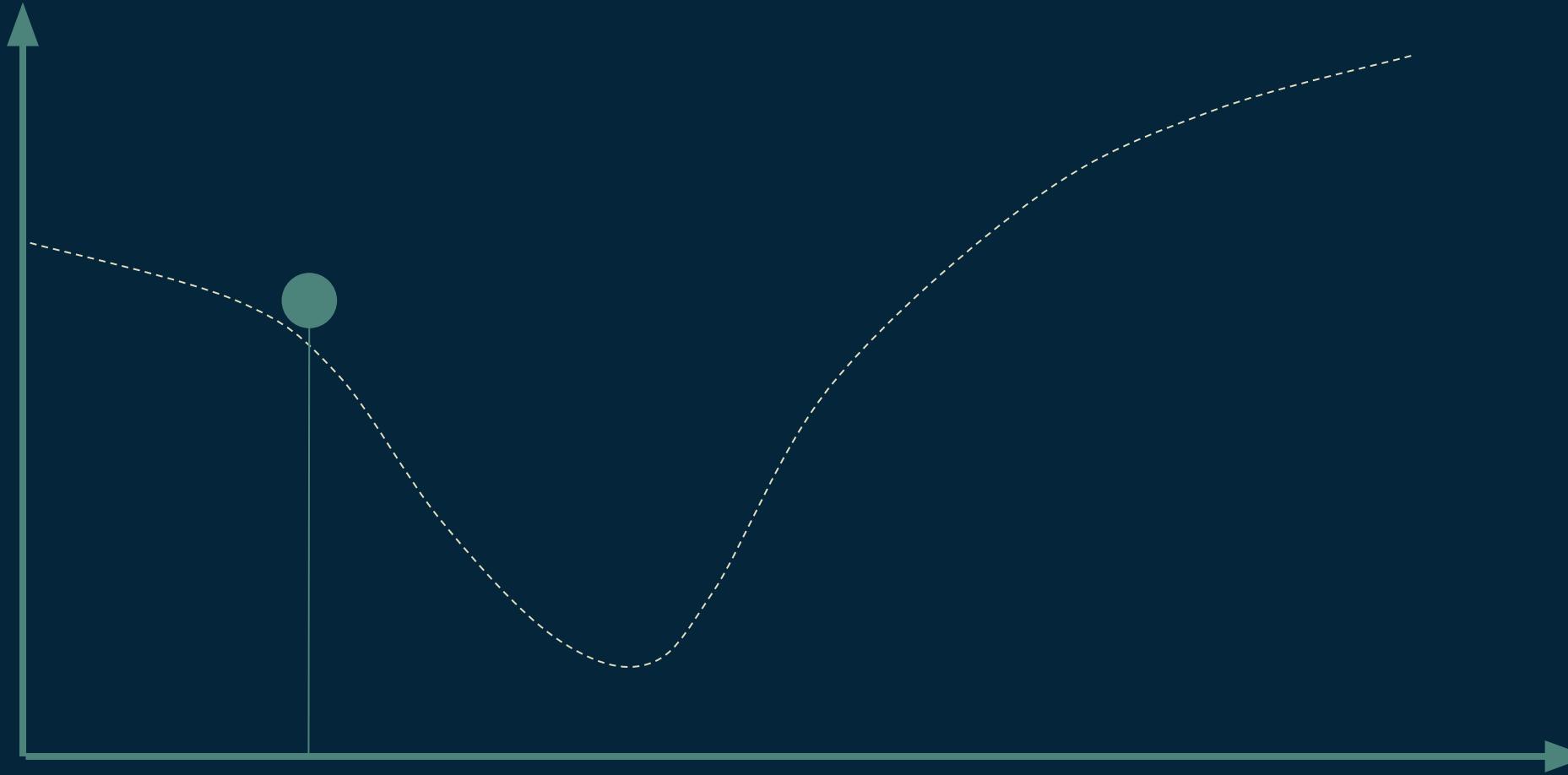
Gradient descent



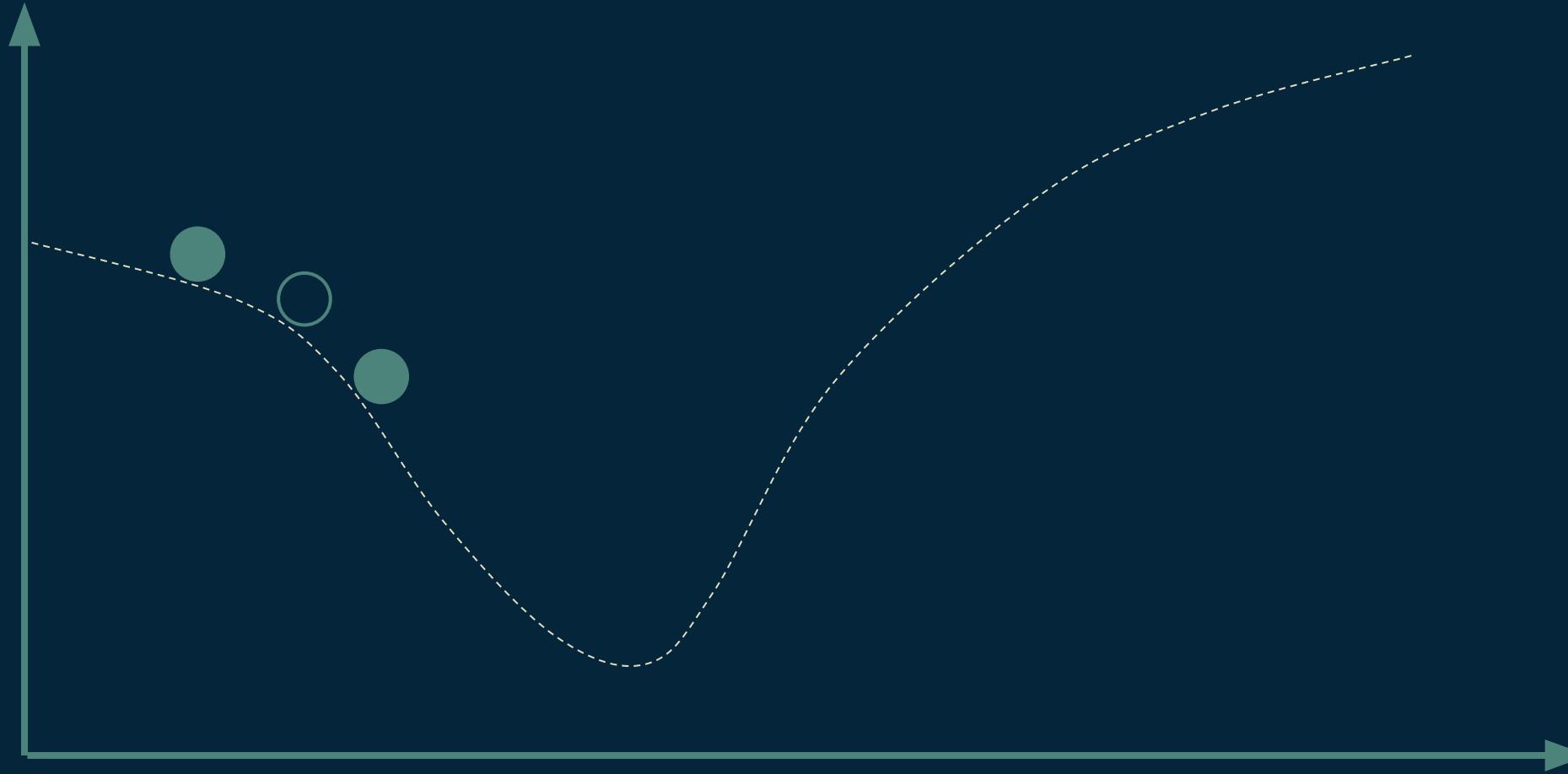
Gradient descent



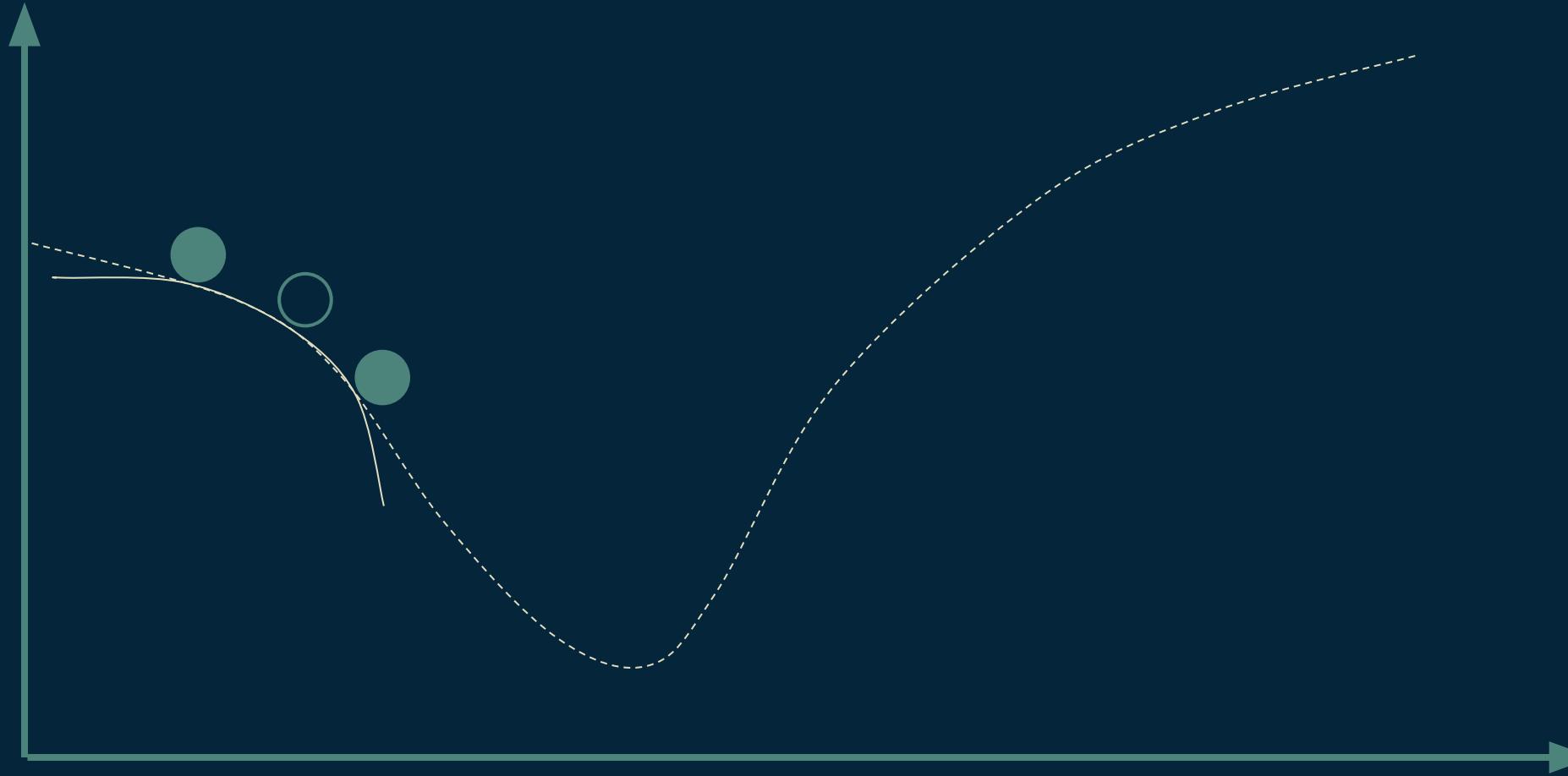
Gradient descent with curvature



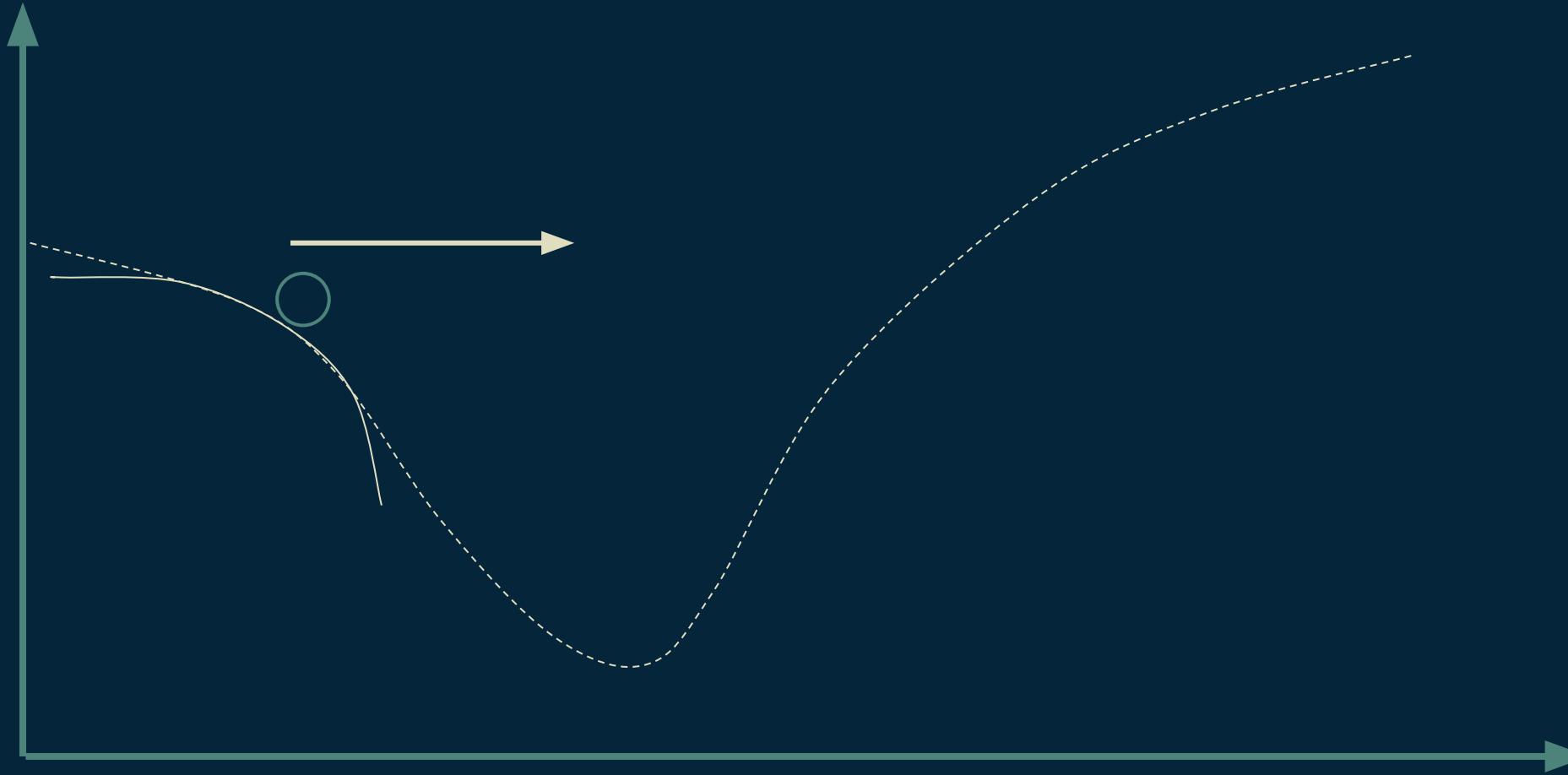
Gradient descent with curvature



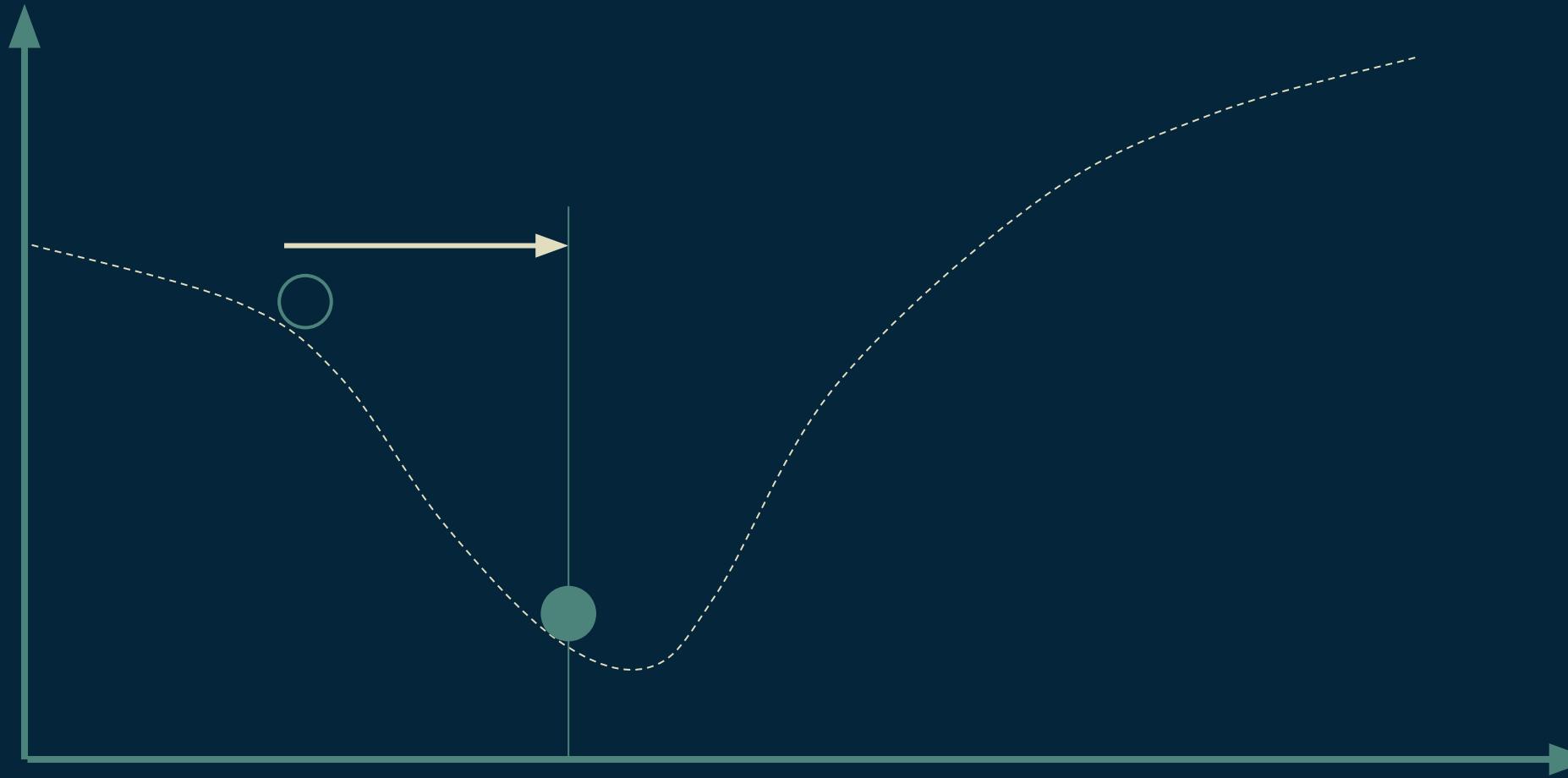
Gradient descent with curvature



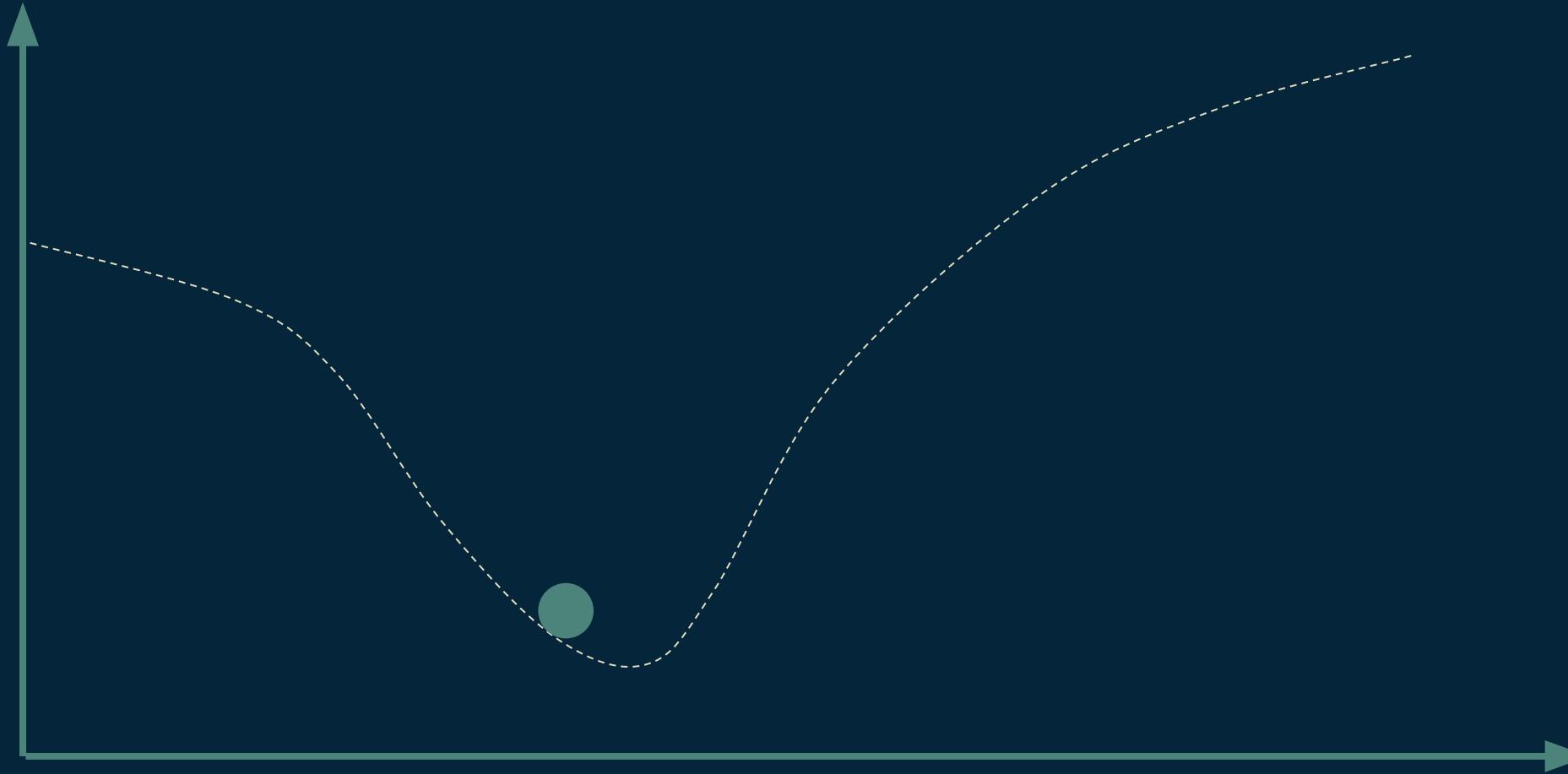
Gradient descent with curvature



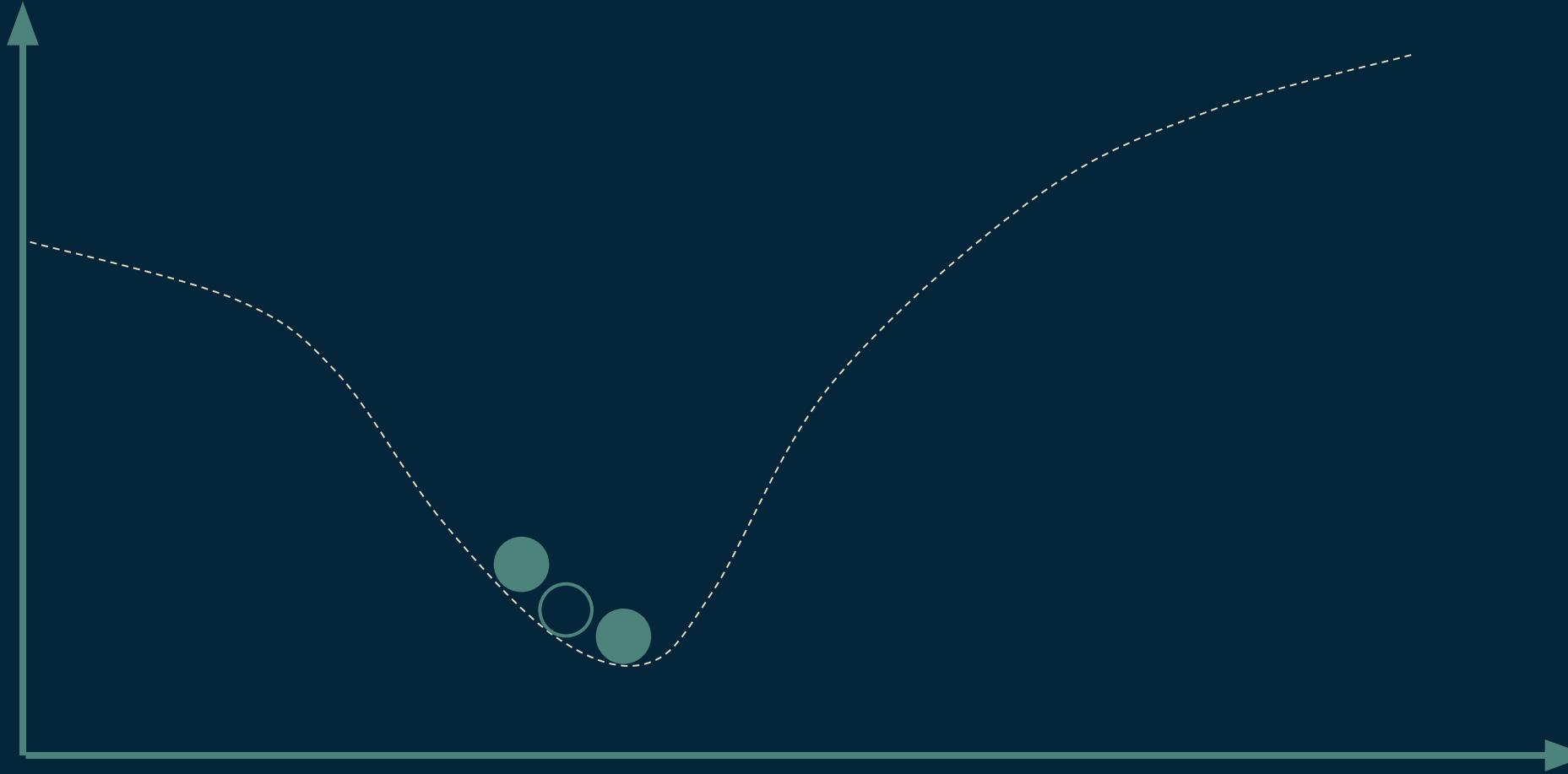
Gradient descent with curvature



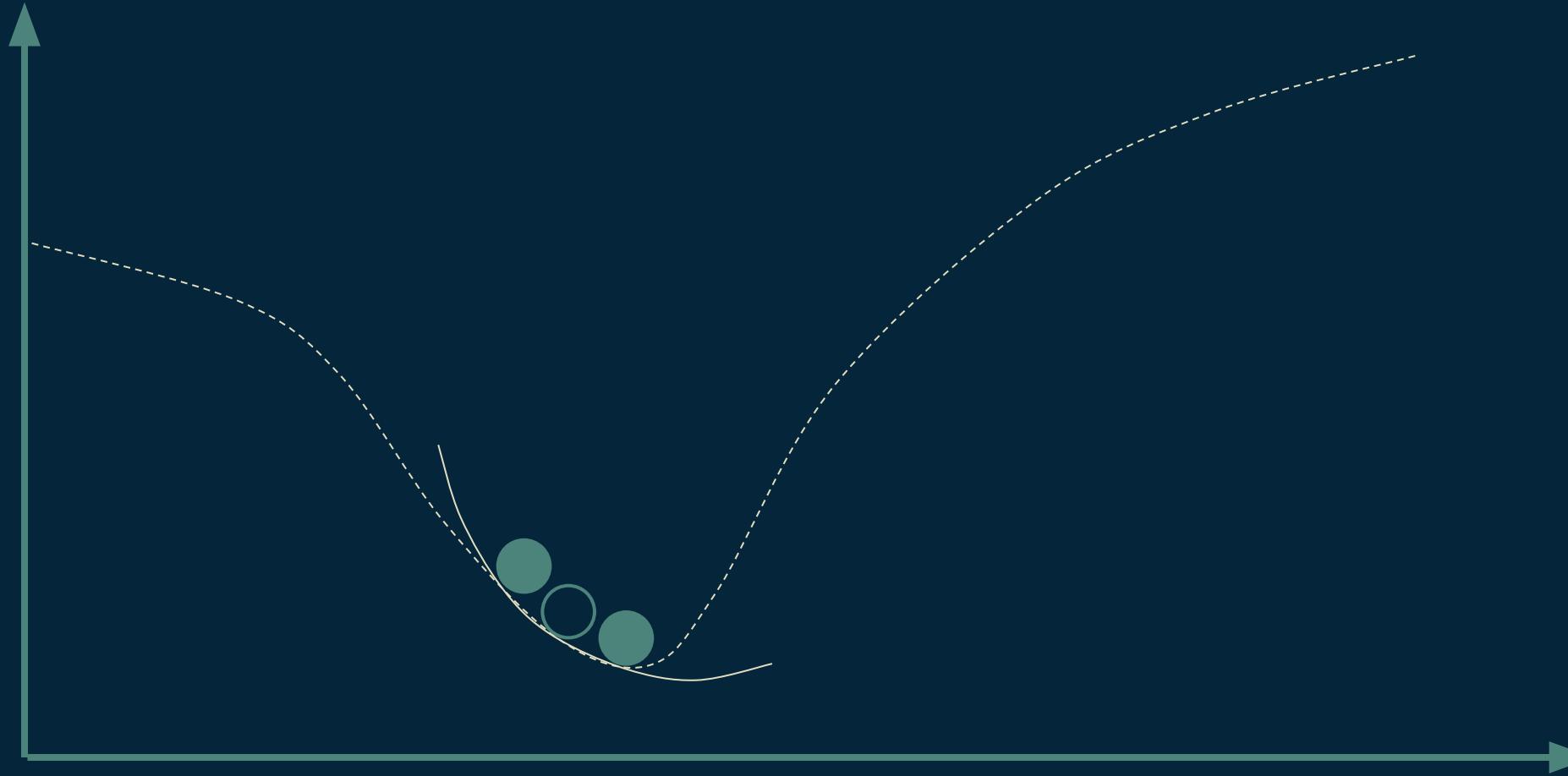
Gradient descent with curvature



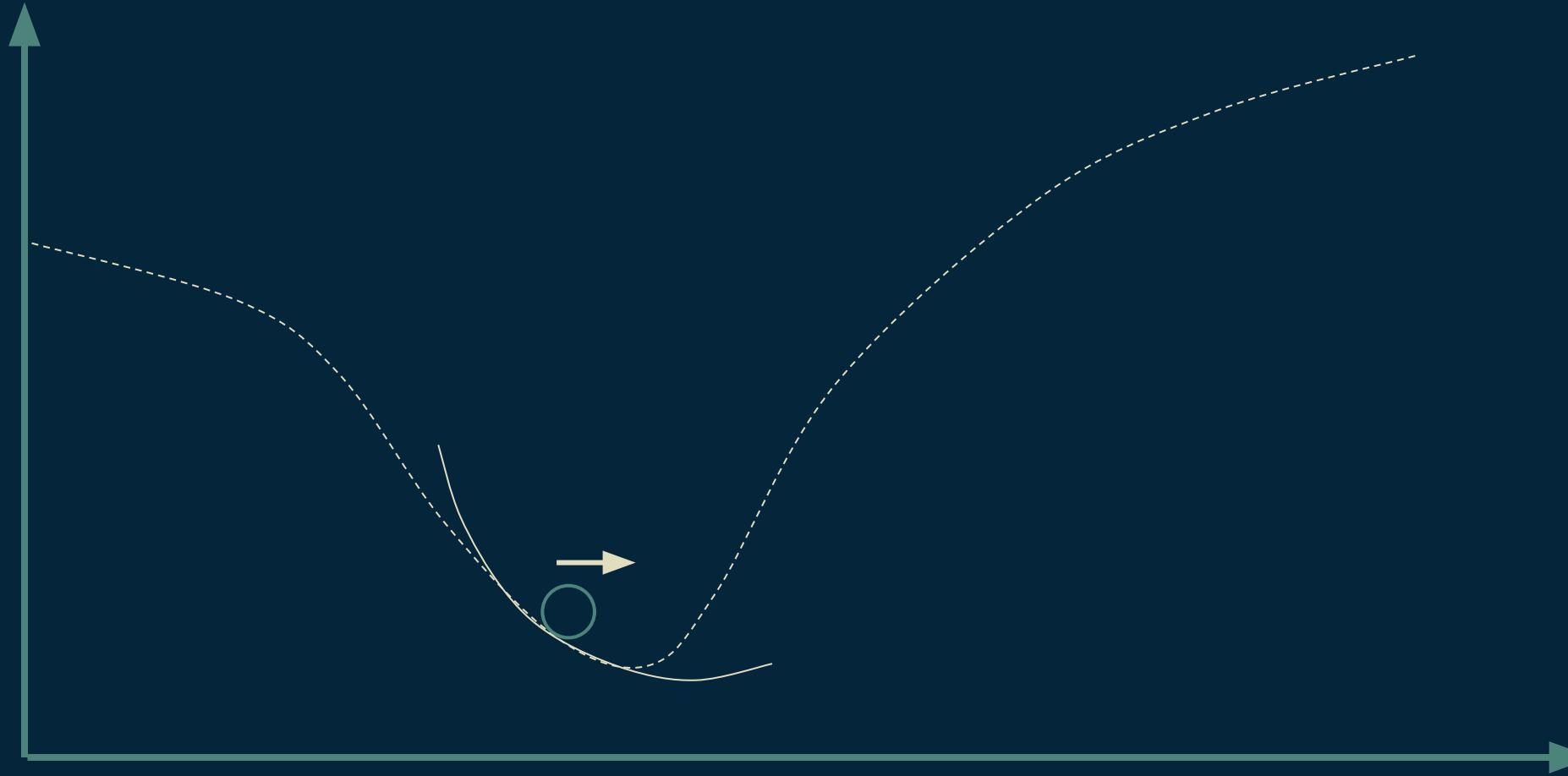
Gradient descent with curvature



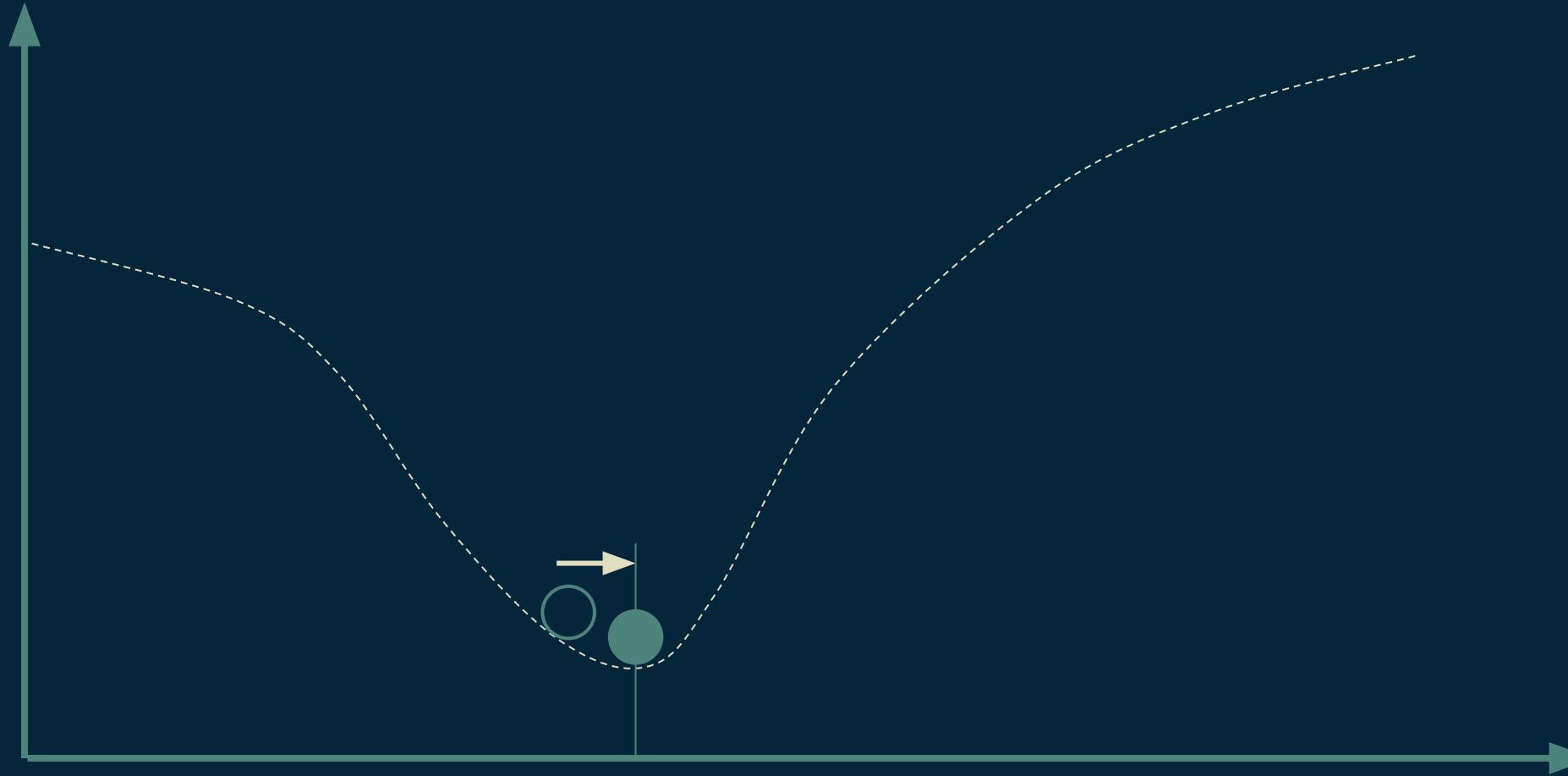
Gradient descent with curvature



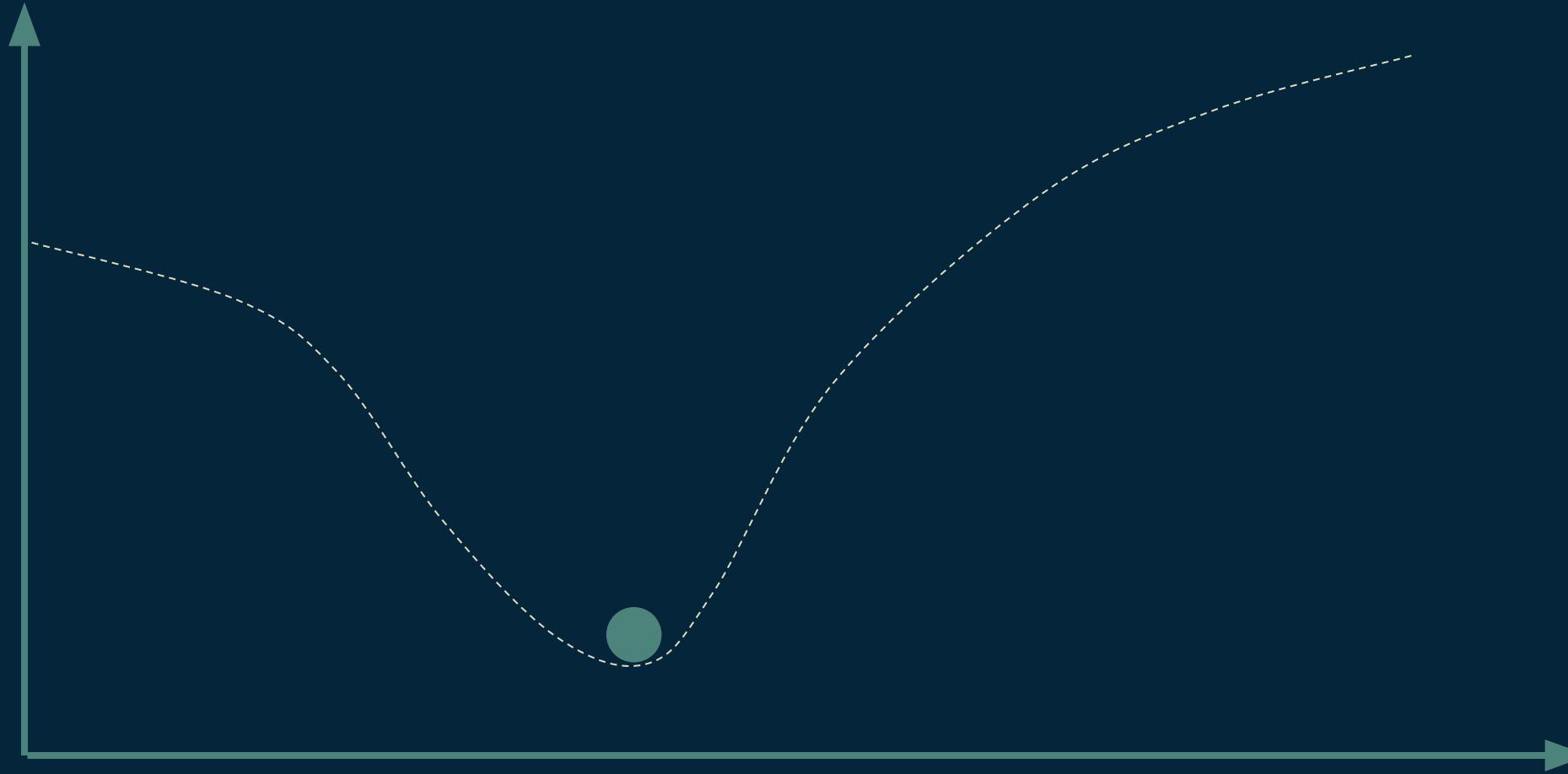
Gradient descent with curvature



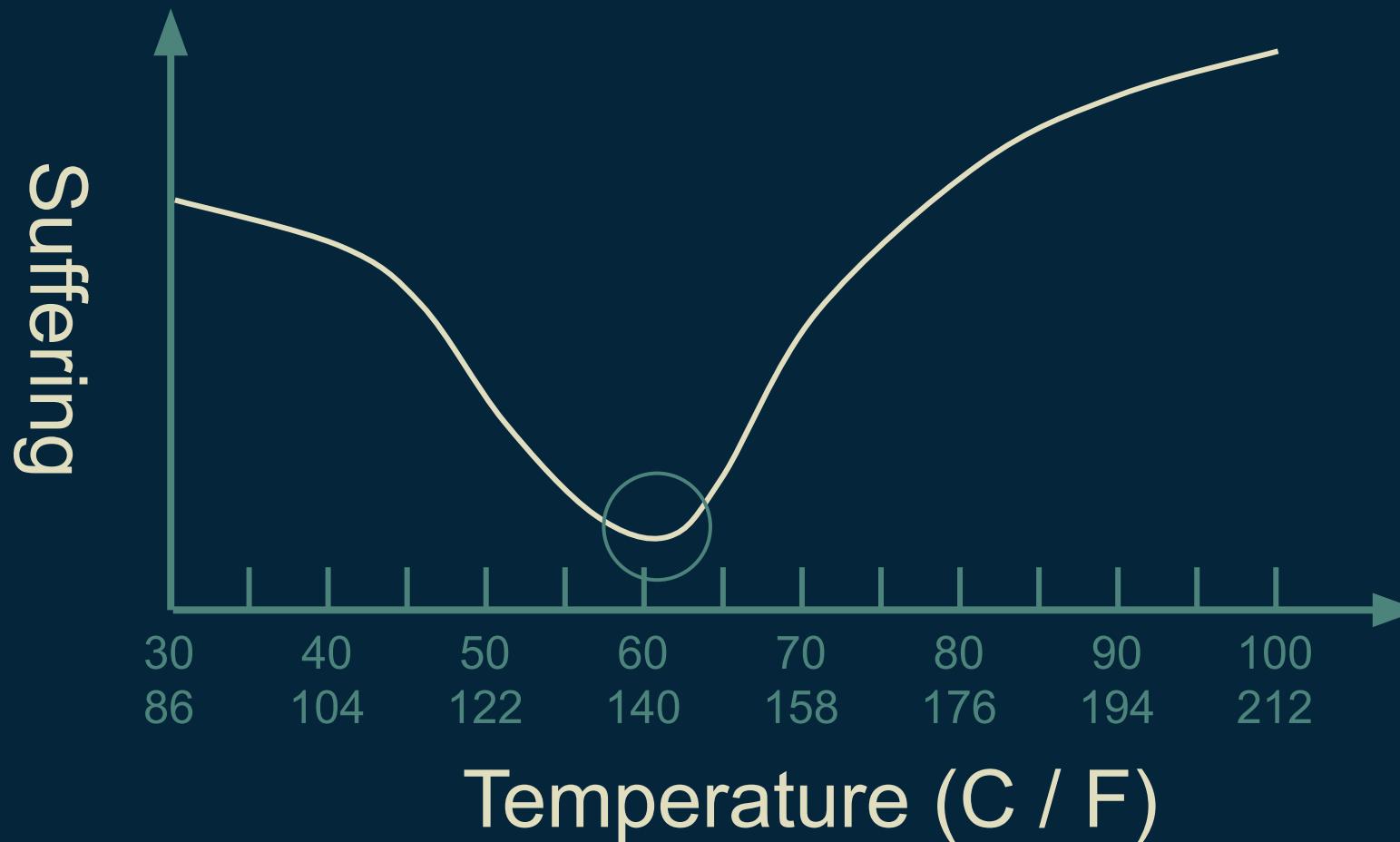
Gradient descent with curvature



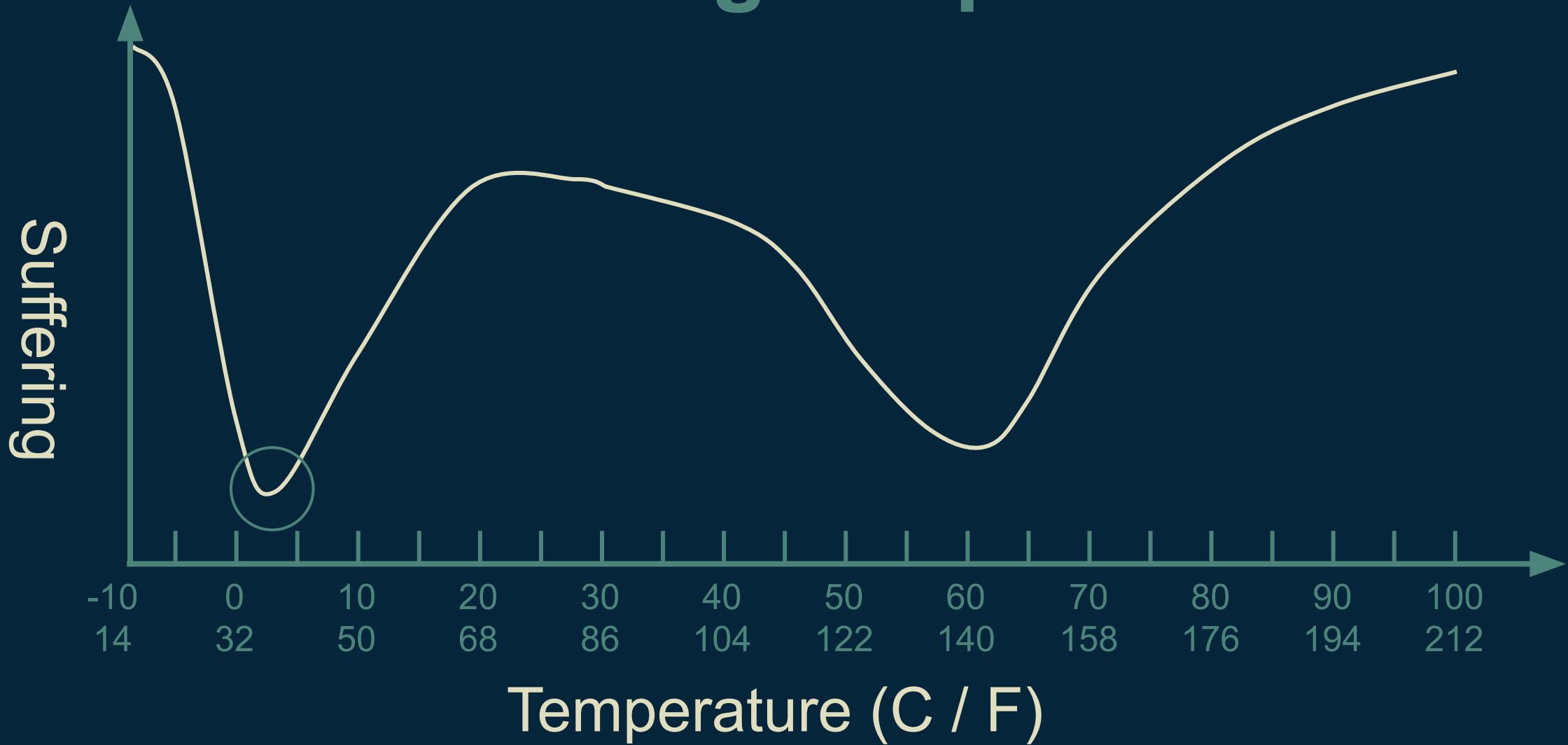
Gradient descent with curvature



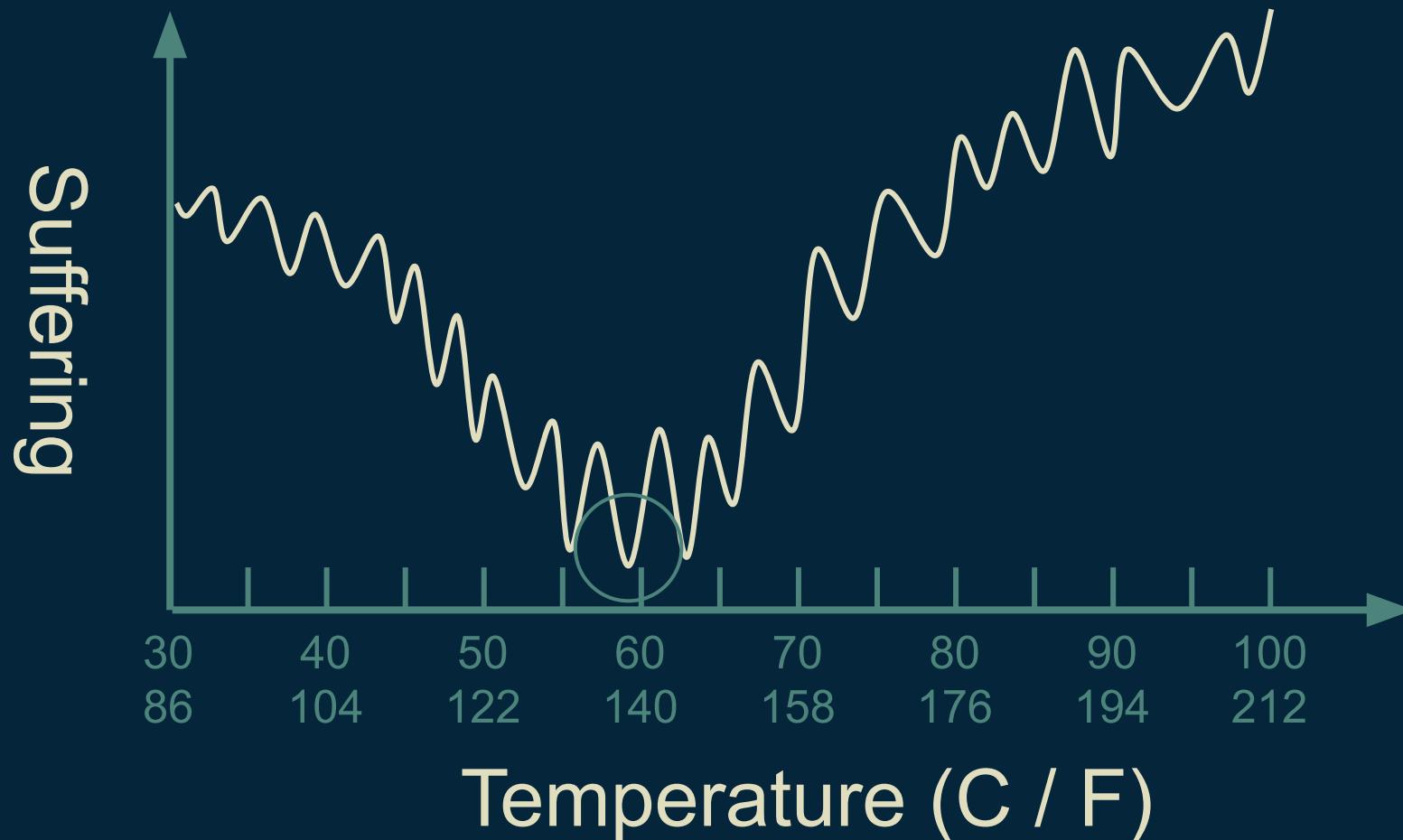
Tea drinking temperature



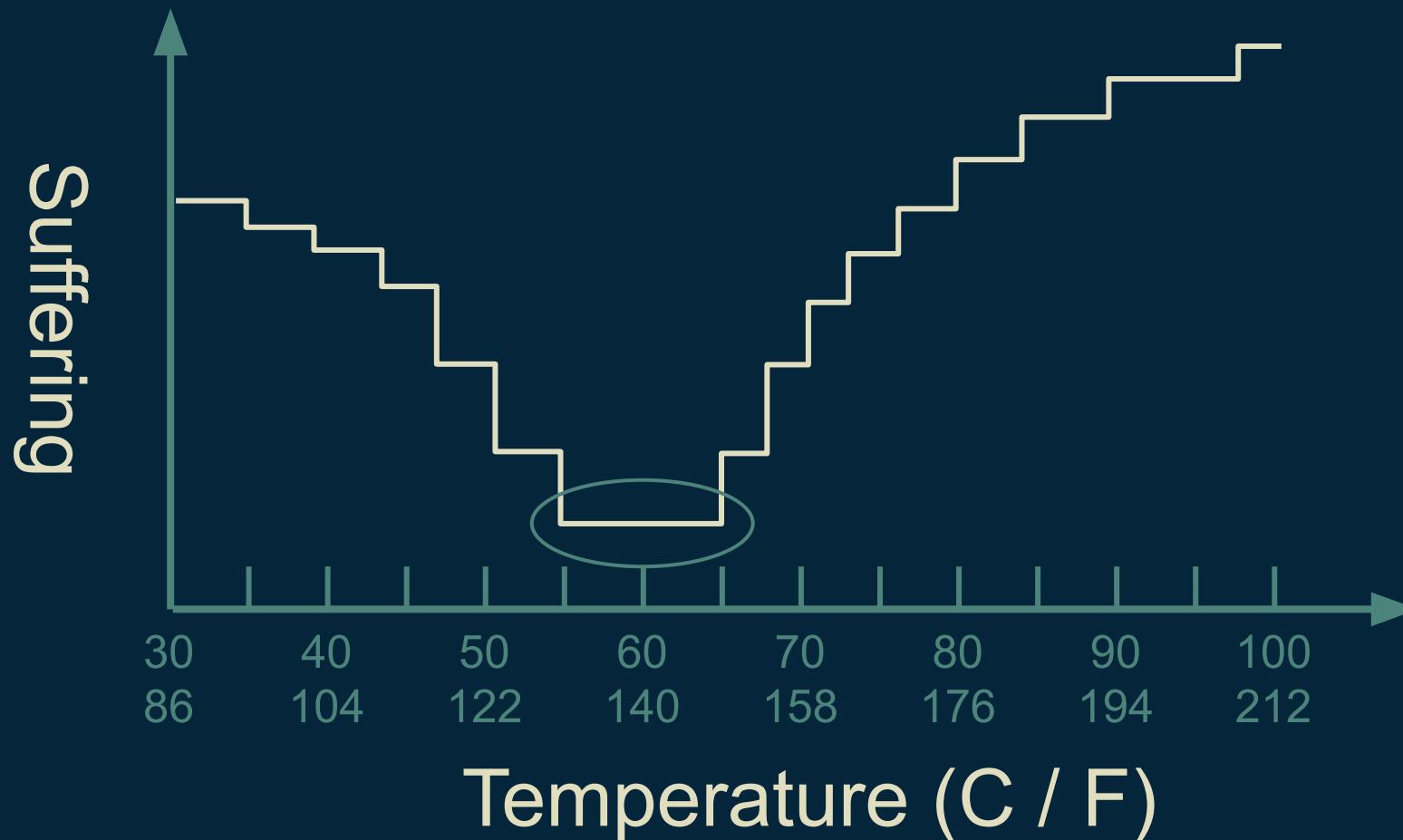
Tea drinking temperature



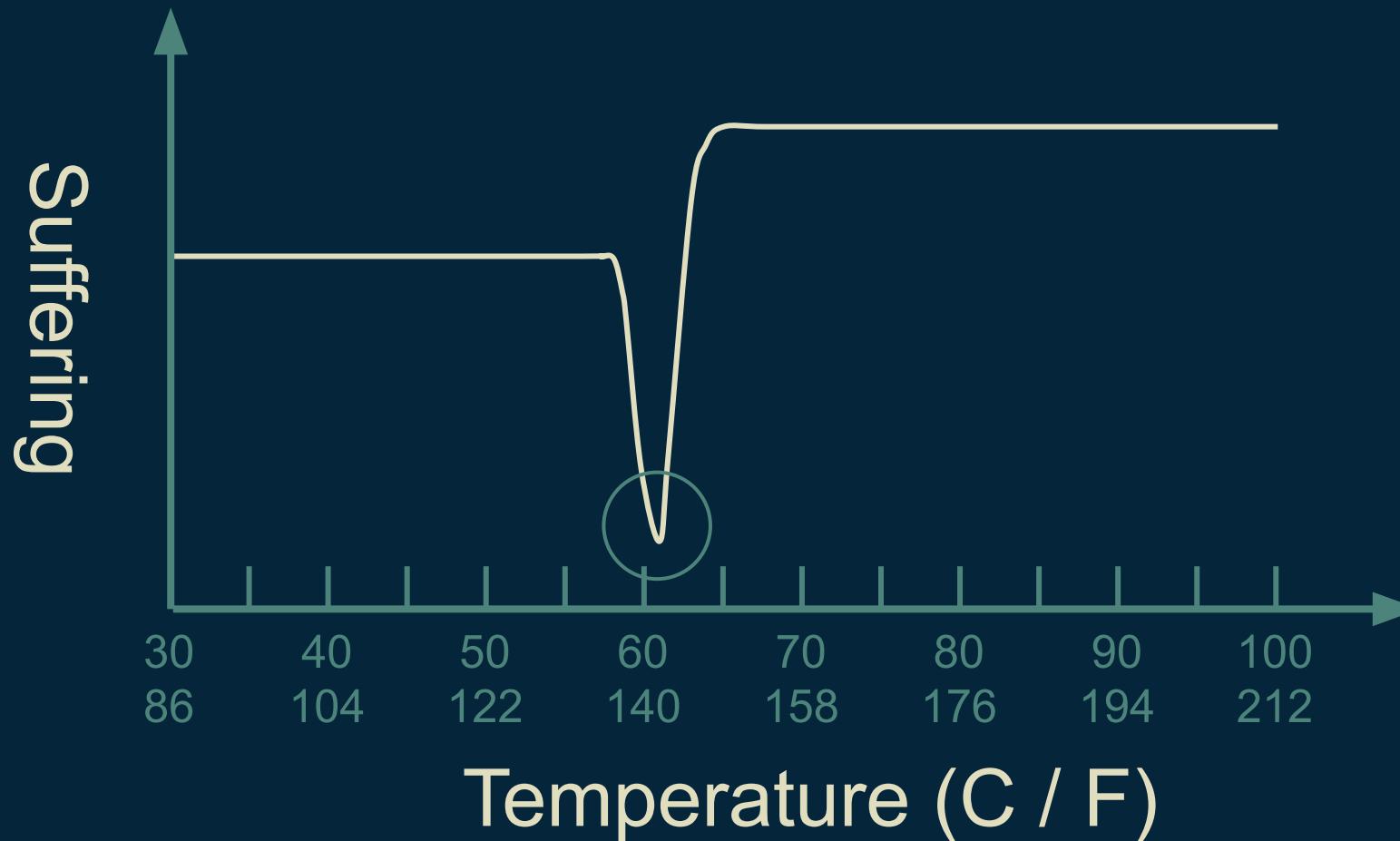
Tea drinking temperature



Tea drinking temperature



Tea drinking temperature



Assumptions vs. performance



Assumptions vs. performance

Exhaustive
exploration



Assumptions vs. performance

Exhaustive
exploration

Gradient
descent



Assumptions vs. performance

Exhaustive
exploration

Genetic algorithms,
simulated annealing, etc.

Gradient
descent



Assumptions vs. performance

Exhaustive
exploration

Genetic algorithms,
simulated annealing, etc.

Gradient
descent



Few assumptions

More assumptions

Assumptions vs. performance

Exhaustive
exploration

Genetic algorithms,
simulated annealing, etc.

Gradient
descent



Few assumptions
Robust

More assumptions
Sensitive

Assumptions vs. performance

Exhaustive
exploration

Genetic algorithms,
simulated annealing, etc.

Gradient
descent



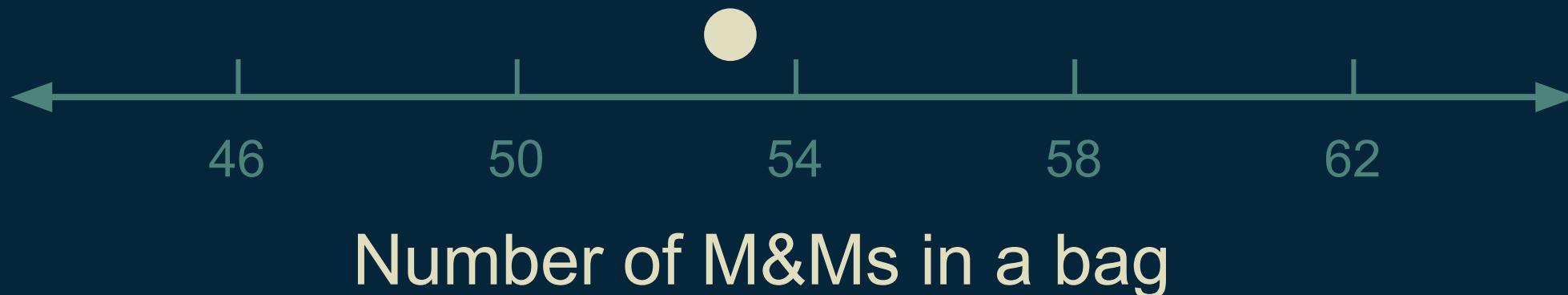
Few assumptions
Robust
Expensive to compute

More assumptions
Sensitive
Efficient to compute

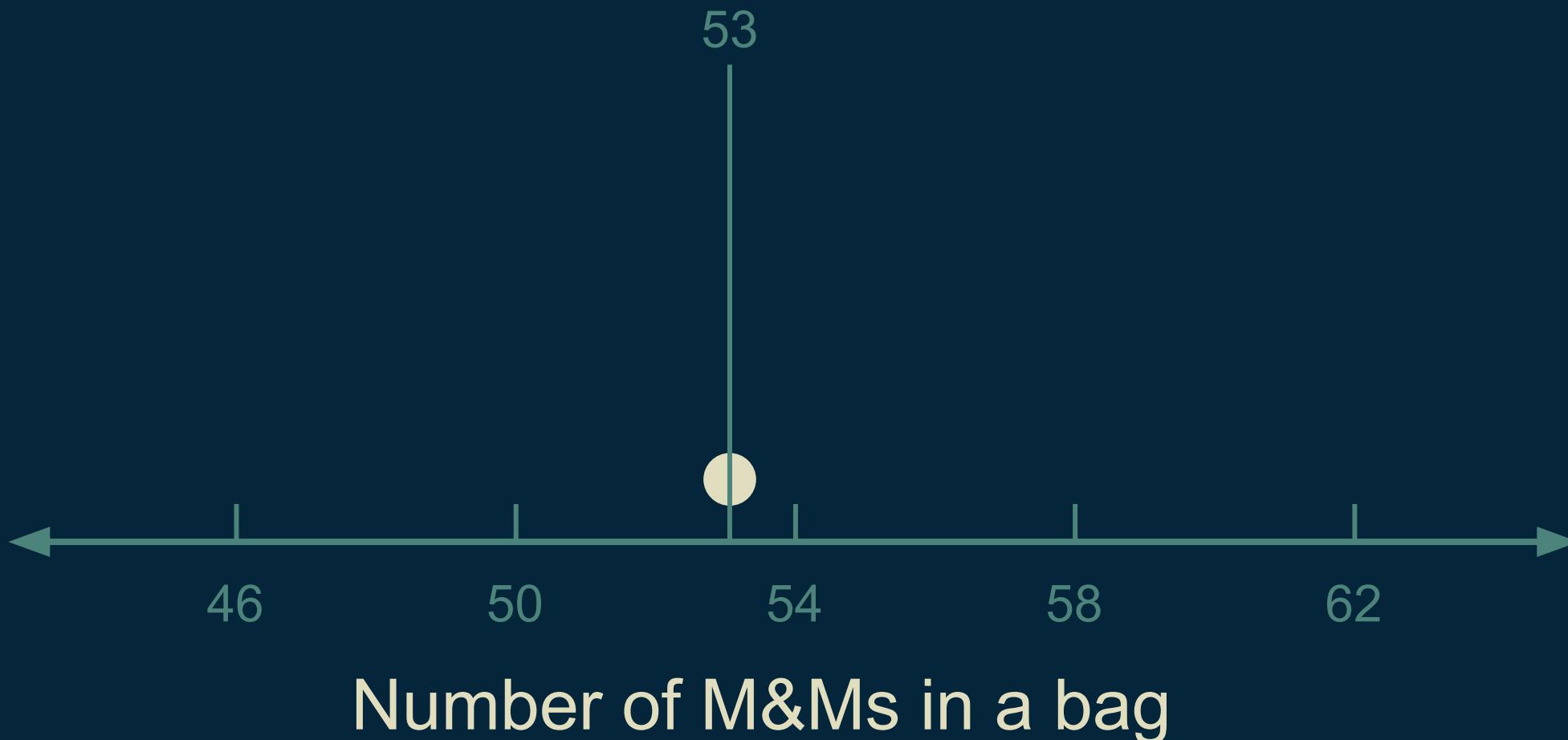
From data to model: How many M&Ms in a bag?



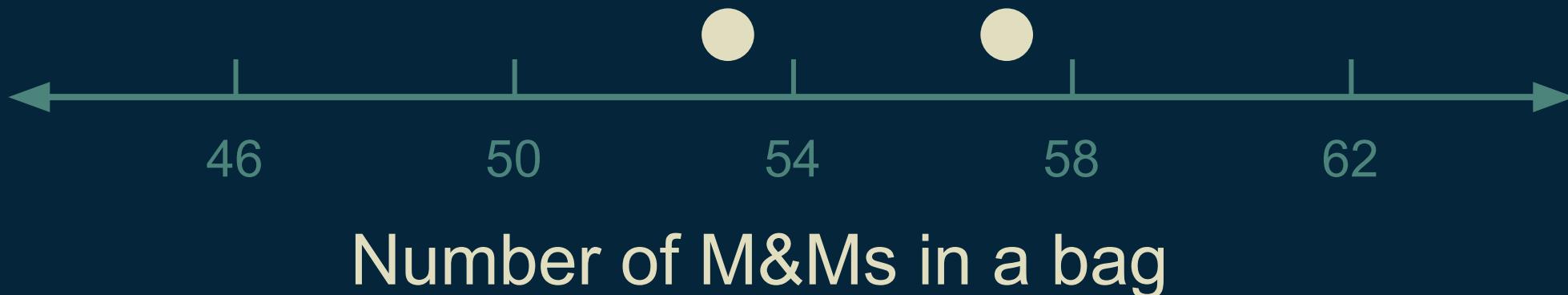
How many M&Ms in a bag?



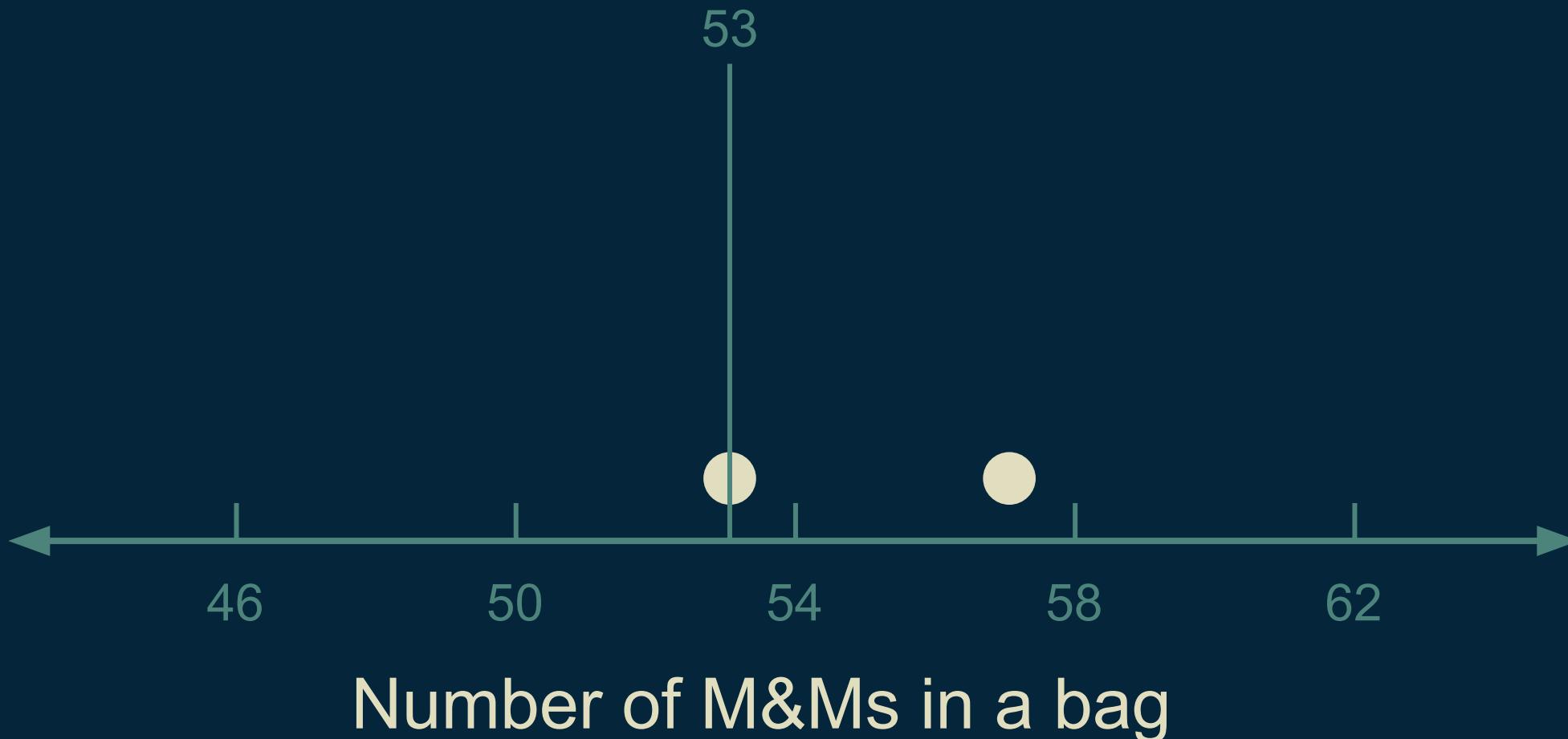
How many M&Ms in a bag?



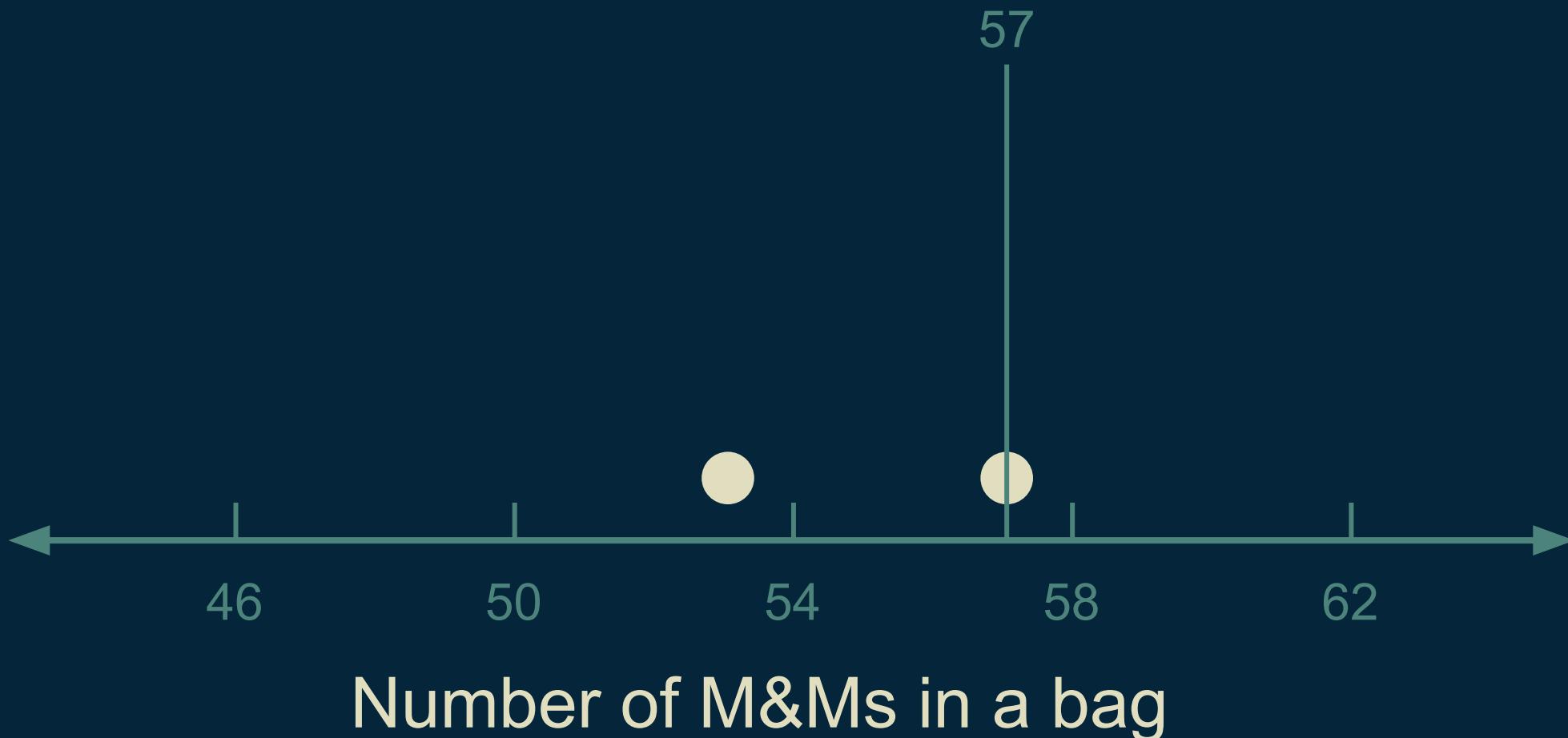
How many M&Ms in a bag?



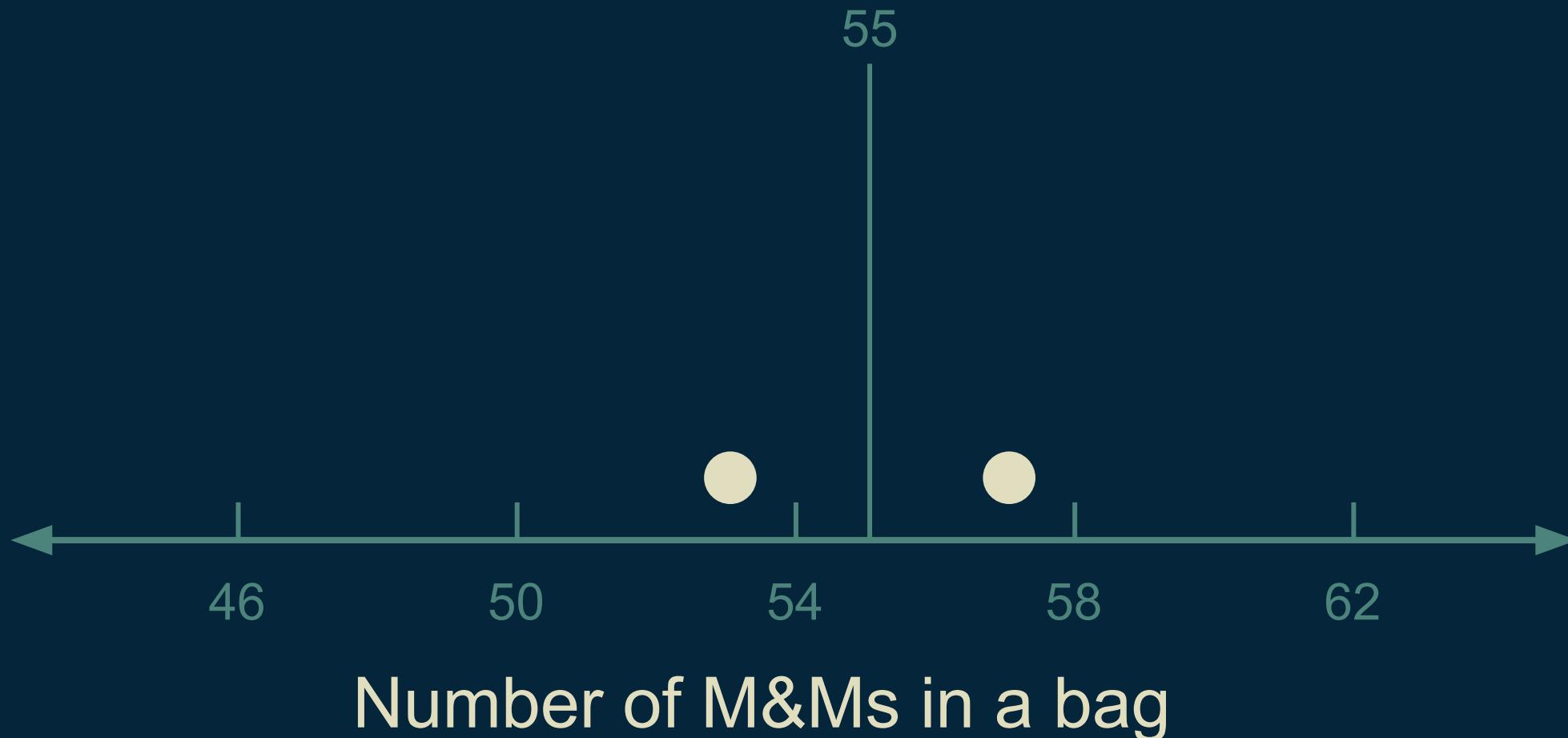
How many M&Ms in a bag?



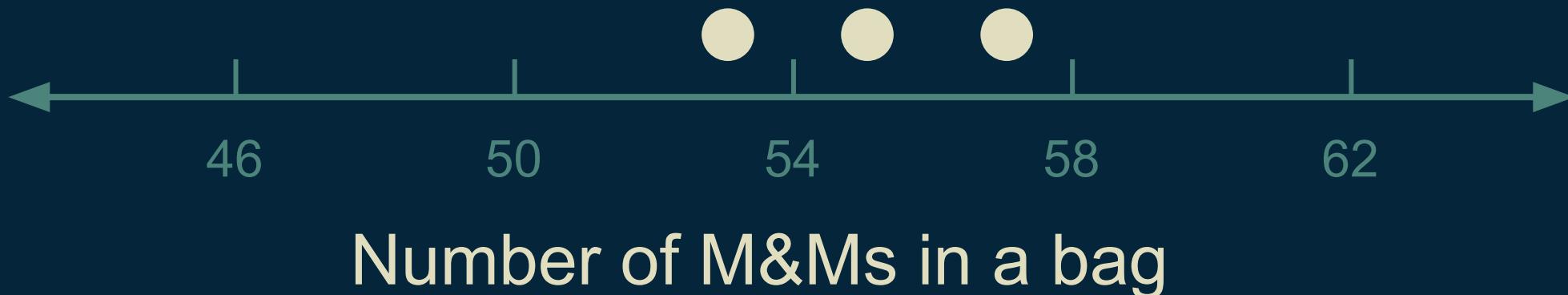
How many M&Ms in a bag?



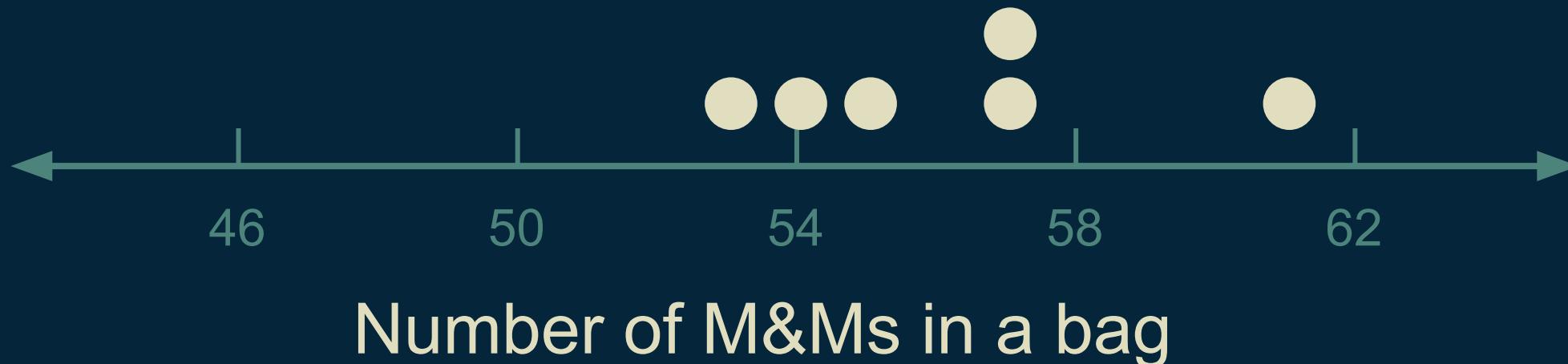
How many M&Ms in a bag?



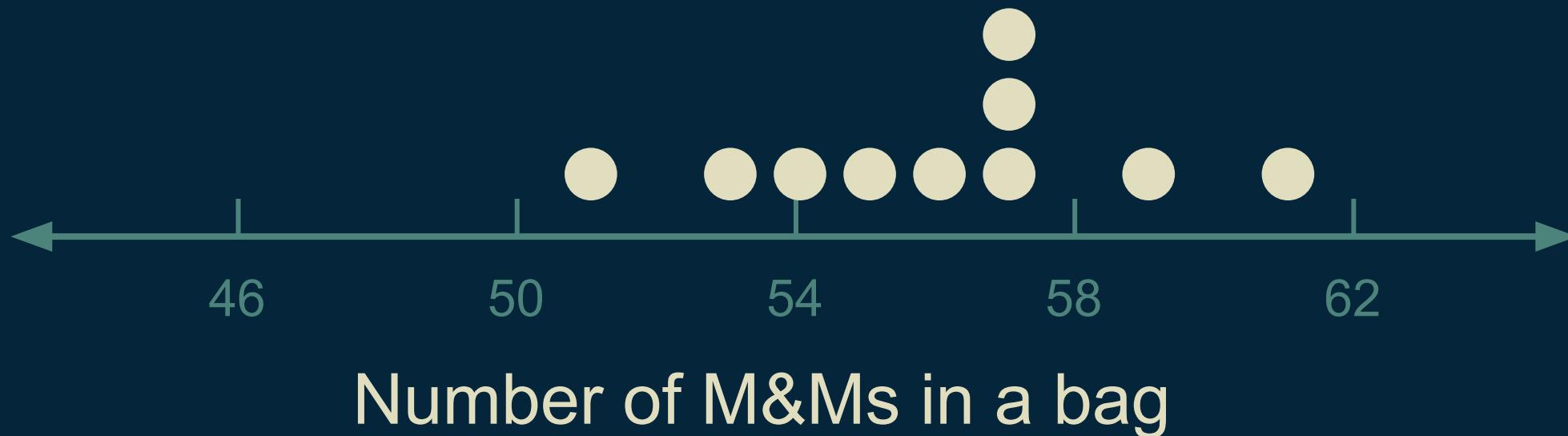
How many M&Ms in a bag?



How many M&Ms in a bag?



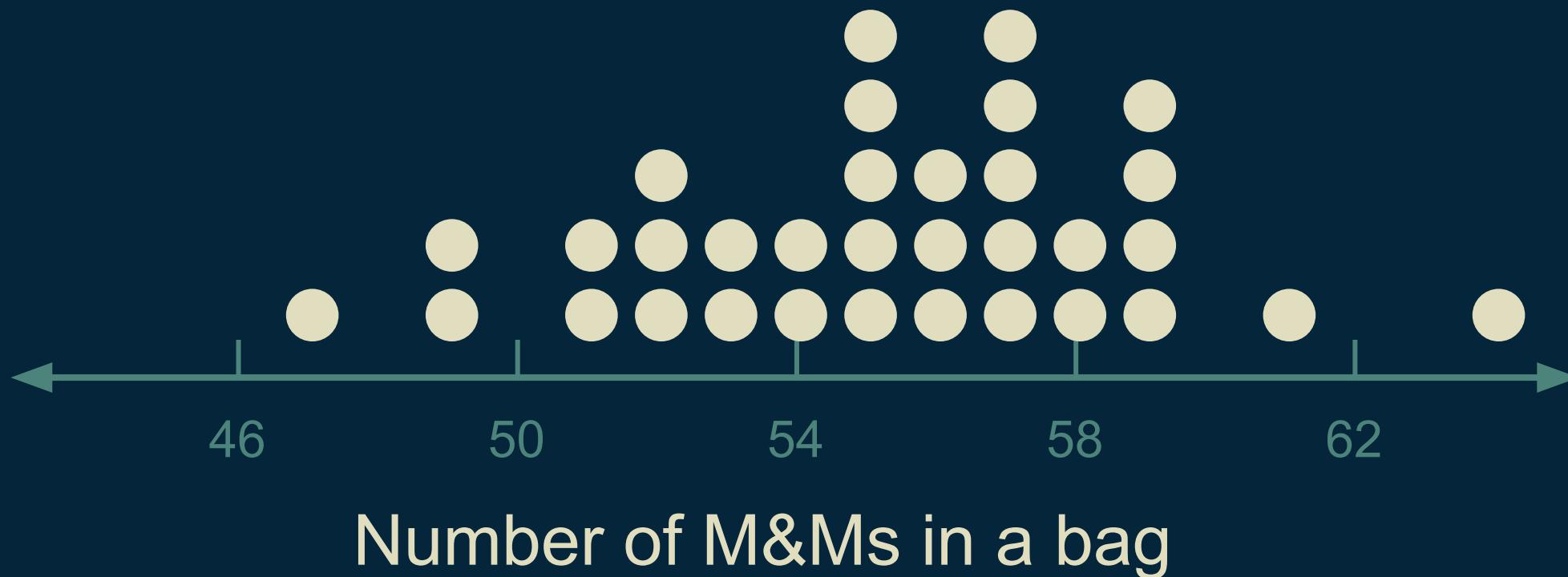
How many M&Ms in a bag?



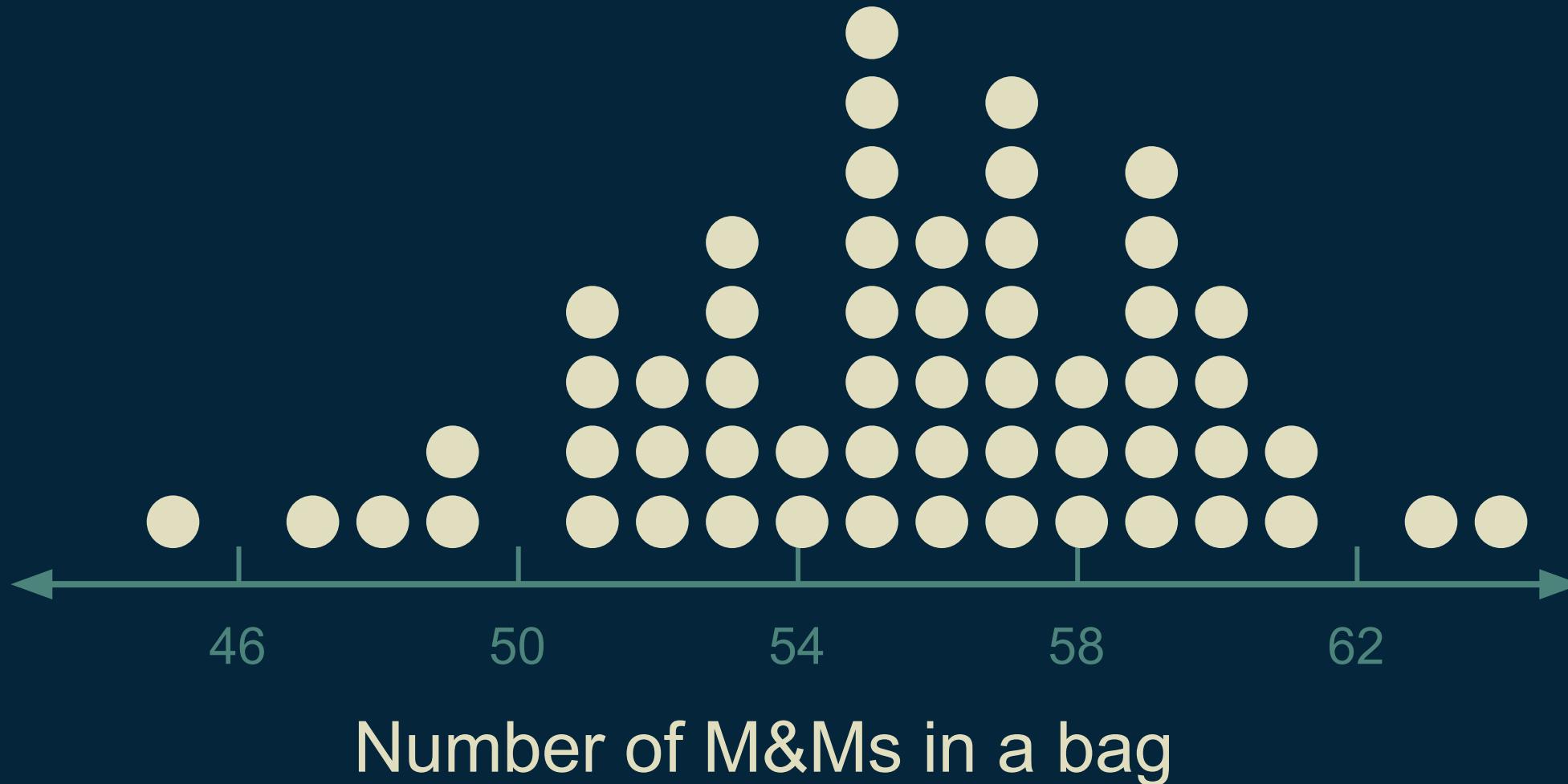
How many M&Ms in a bag?



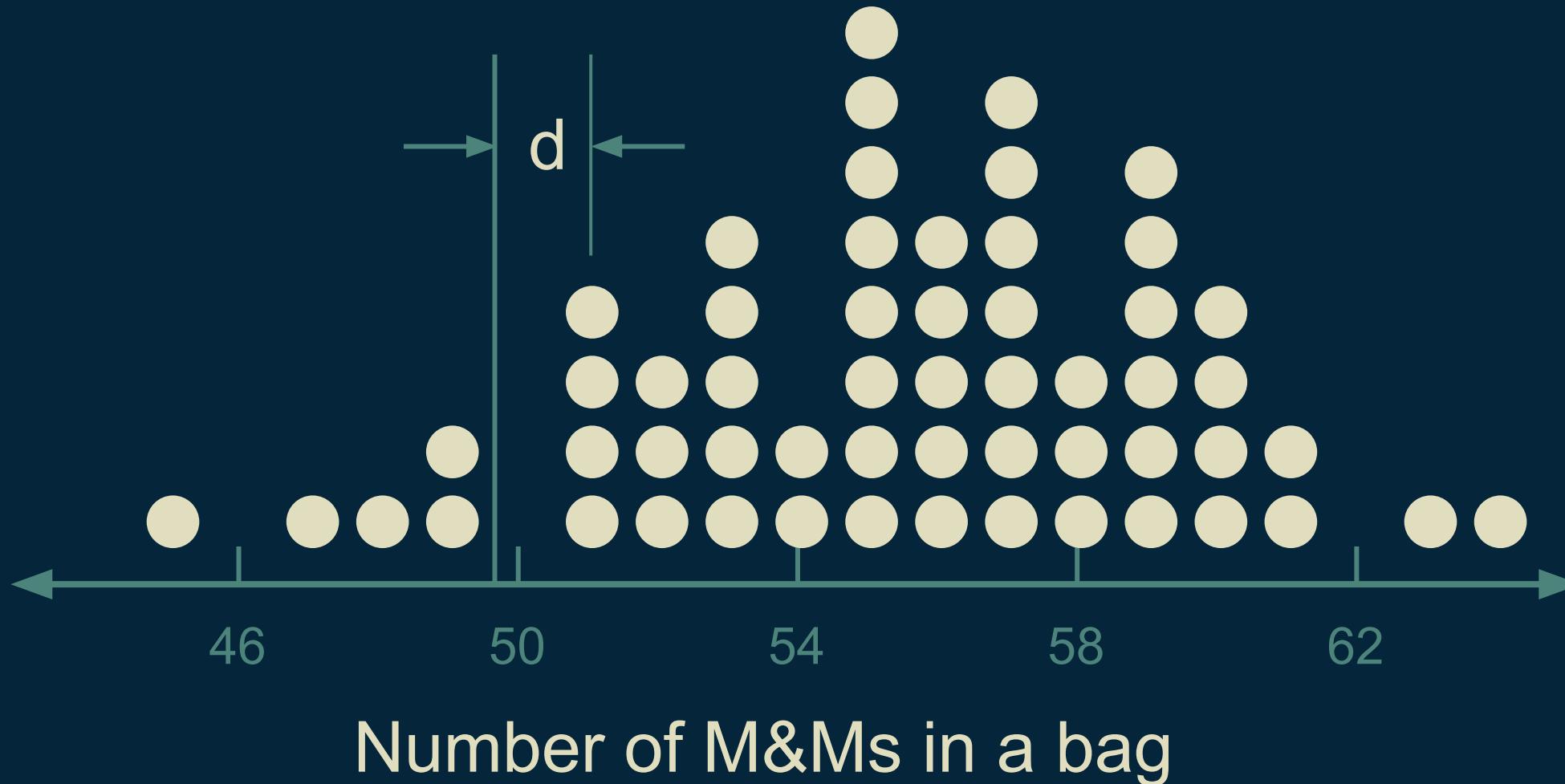
How many M&Ms in a bag?



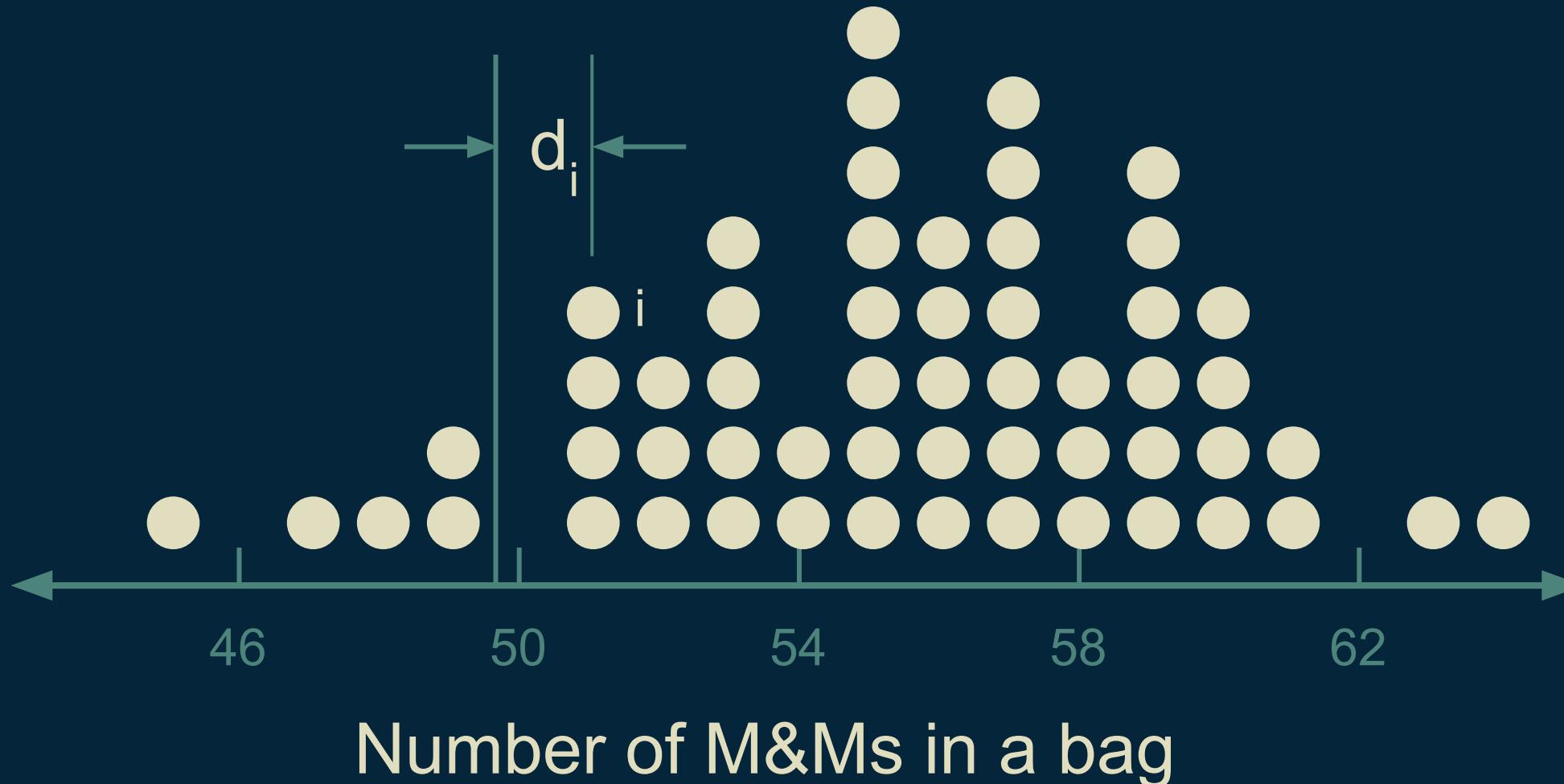
How many M&Ms in a bag?



How wrong is any answer?



How wrong is an answer?



What is the cost of being off by d?

$$d = n_{\text{actual}} - n_{\text{guess}}$$

What is the cost of being off by d?

$$d = n_{\text{actual}} - n_{\text{guess}}$$

cost
candidate

$$d^2$$

What is the cost of being off by d ?

$$d = n_{\text{actual}} - n_{\text{guess}}$$

cost candidate	cost of being off by			
	1	2	4	8
d^2				

What is the cost of being off by d ?

$$d = n_{\text{actual}} - n_{\text{guess}}$$

cost candidate	cost of being off by			
	1	2	4	8
d^2	1	4	16	64

What is the cost of being off by d ?

$$d = n_{\text{actual}} - n_{\text{guess}}$$

cost candidate	cost of being off by			
	1	2	4	8
$ d $	1	2	4	8
d^2	1	4	16	64

What is the cost of being off by d ?

$$d = n_{\text{actual}} - n_{\text{guess}}$$

cost candidate	cost of being off by			
	1	2	4	8
$\sqrt{ d }$	1	1.41	2	2.83
$ d $	1	2	4	8
d^2	1	4	16	64

What is the cost of being off by d ?

$$d = n_{\text{actual}} - n_{\text{guess}}$$

cost candidate	cost of being off by			
	1	2	4	8
$\sqrt{ d }$	1	1.41	2	2.83
$ d $	1	2	4	8
d^2	1	4	16	64
$10^{ d -1}$	1	10	1000	10,000,000

What is the cost of being off by d ?

$$d = n_{\text{actual}} - n_{\text{guess}}$$

cost candidate	cost of being off by			
	1	2	4	8
$\sqrt{ d }$	1	1.41	2	2.83
$ d $	1	2	4	8
d^2	1	4	16	64
$10^{ d -1}$	1	10	1000	10,000,000

What is the total cost of any guess?

For guess n_{est} ,

What is the total cost of any guess?

For guess n_{est} ,

$$\mathcal{L}(n_{\text{est}})$$

What is the total cost of any guess?

For guess n_{est} ,

$$\mathcal{L}(n_{\text{est}}) = d_1^2 + d_2^2 + d_3^2 + \dots + d_m^2$$

What is the total cost of any guess?

For guess n_{est} ,

$$\mathcal{L}(n_{\text{est}}) = d_1^2 + d_2^2 + d_3^2 + \dots + d_m^2$$

$$\mathcal{L}(n_{\text{est}}) = (n_1 - n_{\text{est}})^2 + (n_2 - n_{\text{est}})^2 + (n_3 - n_{\text{est}})^2 + \dots + (n_m - n_{\text{est}})^2$$

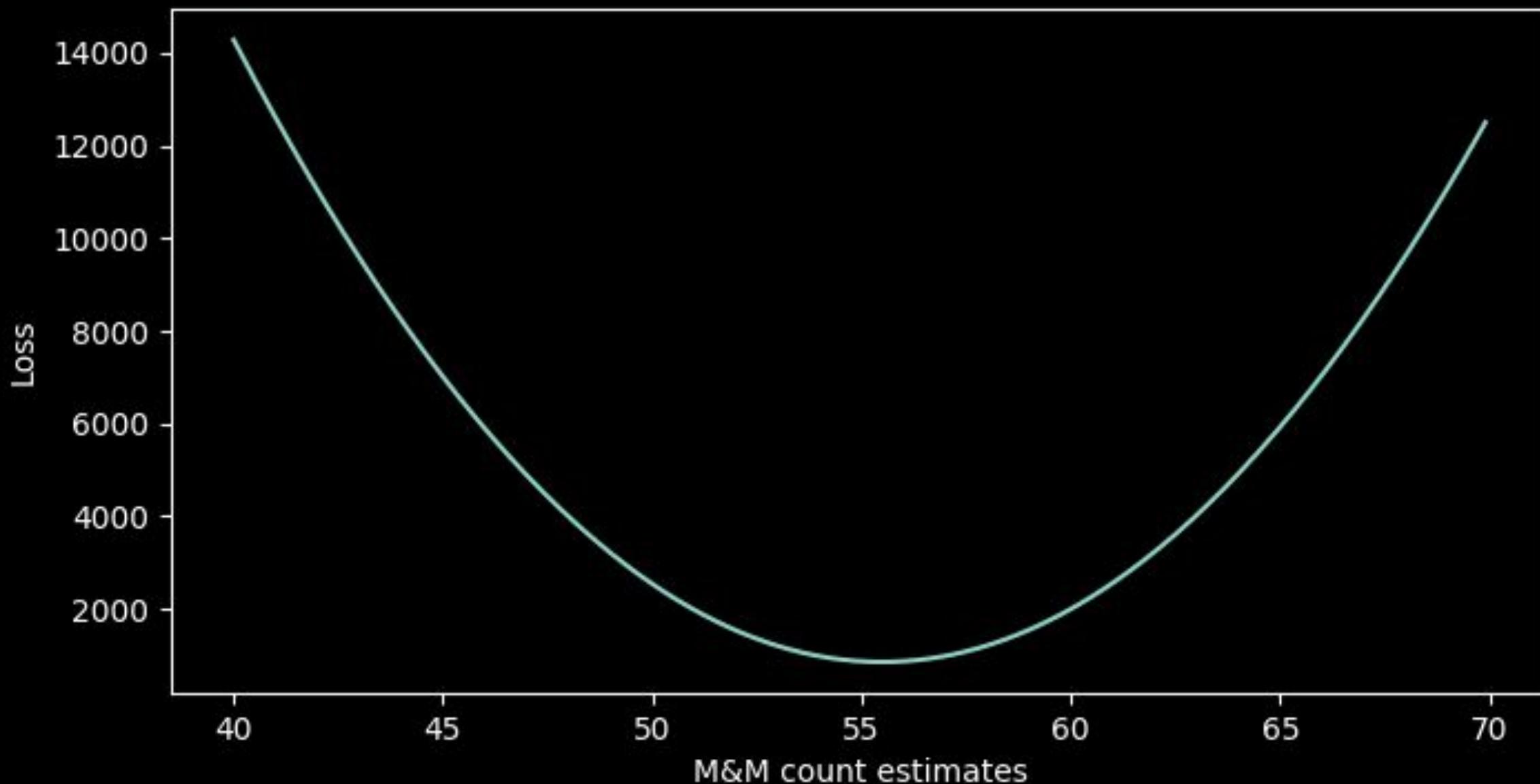
What is the total cost of any guess?

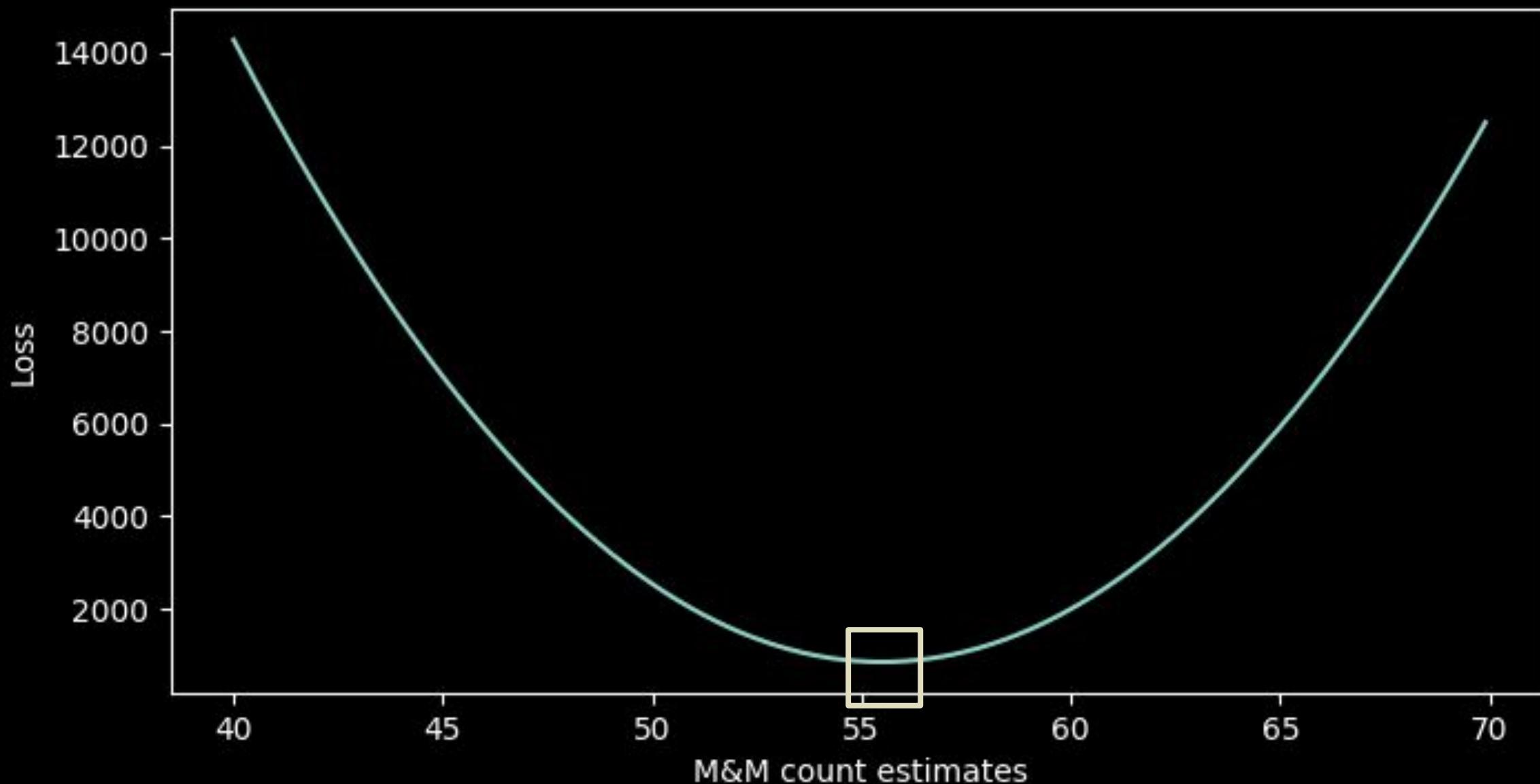
For guess n_{est} ,

$$\mathcal{L}(n_{\text{est}}) = d_1^2 + d_2^2 + d_3^2 + \dots + d_m^2$$

$$\mathcal{L}(n_{\text{est}}) = (n_1 - n_{\text{est}})^2 + (n_2 - n_{\text{est}})^2 + (n_3 - n_{\text{est}})^2 + \dots + (n_m - n_{\text{est}})^2$$

$$\mathcal{L}(n_{\text{est}}) = \sum_i (n_i - n_{\text{est}})^2$$





What is the total cost of any guess?

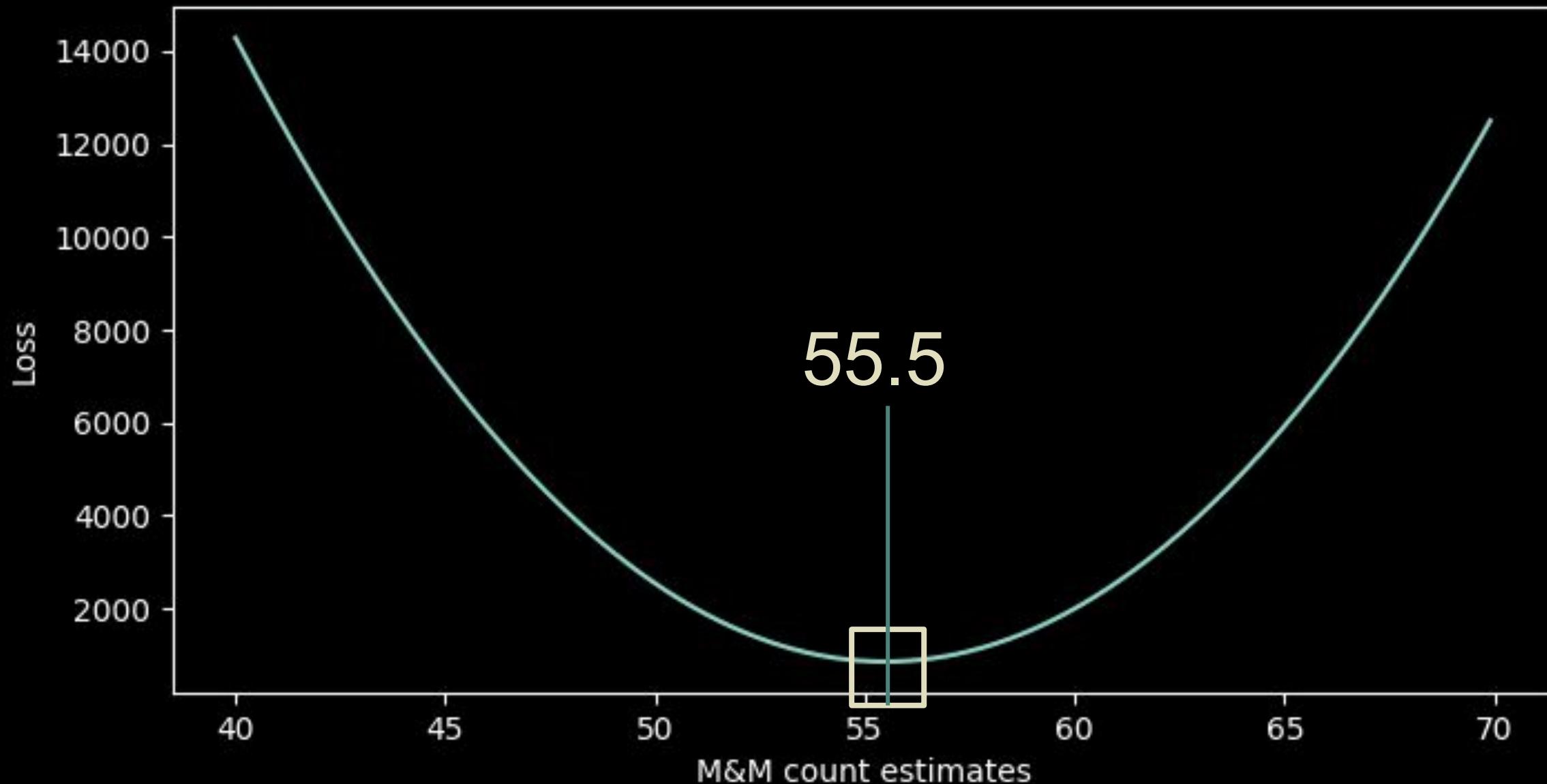
For guess n_{est} ,

$$\mathcal{L}(n_{\text{est}}) = d_1^2 + d_2^2 + d_3^2 + \dots + d_m^2$$

$$\mathcal{L}(n_{\text{est}}) = (n_1 - n_{\text{est}})^2 + (n_2 - n_{\text{est}})^2 + (n_3 - n_{\text{est}})^2 + \dots + (n_m - n_{\text{est}})^2$$

$$\mathcal{L}(n_{\text{est}}) = \sum_i (n_i - n_{\text{est}})^2$$

$$n_{\text{best}} = \underset{n_{\text{est}}}{\operatorname{argmin}} (\mathcal{L}(n_{\text{est}}))$$



Another way: Find where slope = 0

$$\frac{d}{dn}(\mathcal{L}(n_{\text{est}}))$$

Another way: Find where slope = 0

$$\frac{d}{dn}(\mathcal{L}(n_{\text{est}})) = 0$$

Another way: Find where slope = 0

$$\frac{d}{dn}(\mathcal{L}(n_{\text{est}})) = 0$$

$$= \frac{d}{dn}(\sum_i (n_i - n_{\text{est}})^2)$$

Another way: Find where slope = 0

$$\frac{d}{dn}(\mathcal{L}(n_{\text{est}})) = 0$$

$$= \frac{d}{dn}(\sum_i (n_i - n_{\text{est}})^2)$$

$$= \sum_i \frac{d}{dn}((n_i - n_{\text{est}})^2)$$

Another way: Find where slope = 0

$$\frac{d}{dn}(\mathcal{L}(n_{\text{est}})) = 0$$

$$= \frac{d}{dn}(\sum_i (n_i - n_{\text{est}})^2)$$

$$= \sum_i \frac{d}{dn}((n_i - n_{\text{est}})^2)$$

$$= \sum_i 2(n_i - n_{\text{est}})$$

Another way: Find where slope = 0

$$\frac{d}{dn}(\mathcal{L}(n_{\text{est}})) = 0$$

$$= \frac{d}{dn}(\sum_i (n_i - n_{\text{est}})^2)$$

$$= \sum_i \frac{d}{dn}((n_i - n_{\text{est}})^2)$$

$$= \sum_i 2(n_i - n_{\text{est}})$$

$$= \sum_i (n_i - n_{\text{est}})$$

Another way: Find where slope = 0

$$\sum_i (n_i - n_{\text{est}}) = 0$$

Another way: Find where slope = 0

$$\sum_i (n_i - n_{\text{est}}) = 0$$

$$\sum_i n_i - \sum_i n_{\text{est}} = 0$$

Another way: Find where slope = 0

$$\sum_i (n_i - n_{\text{est}}) = 0$$

$$\sum_i n_i - \sum_i n_{\text{est}} = 0$$

$$\sum_i n_i - m n_{\text{est}} = 0$$

Another way: Find where slope = 0

$$\sum_i (n_i - n_{\text{est}}) = 0$$

$$\sum_i n_i - \sum_i n_{\text{est}} = 0$$

$$\sum_i n_i - m n_{\text{est}} = 0$$

$$m n_{\text{est}} = \sum_i n_i$$

Another way: Find where slope = 0

$$\sum_i (n_i - n_{\text{est}}) = 0$$

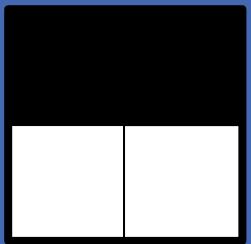
$$\sum_i n_i - \sum_i n_{\text{est}} = 0$$

$$\sum_i n_i - m n_{\text{est}} = 0$$

$$m n_{\text{est}} = \sum_i n_i$$

$$n_{\text{est}} = \sum_i n_i / m = \bar{n}$$

Errors



truth

0.

solid



vertical

0.



diagonal

0.

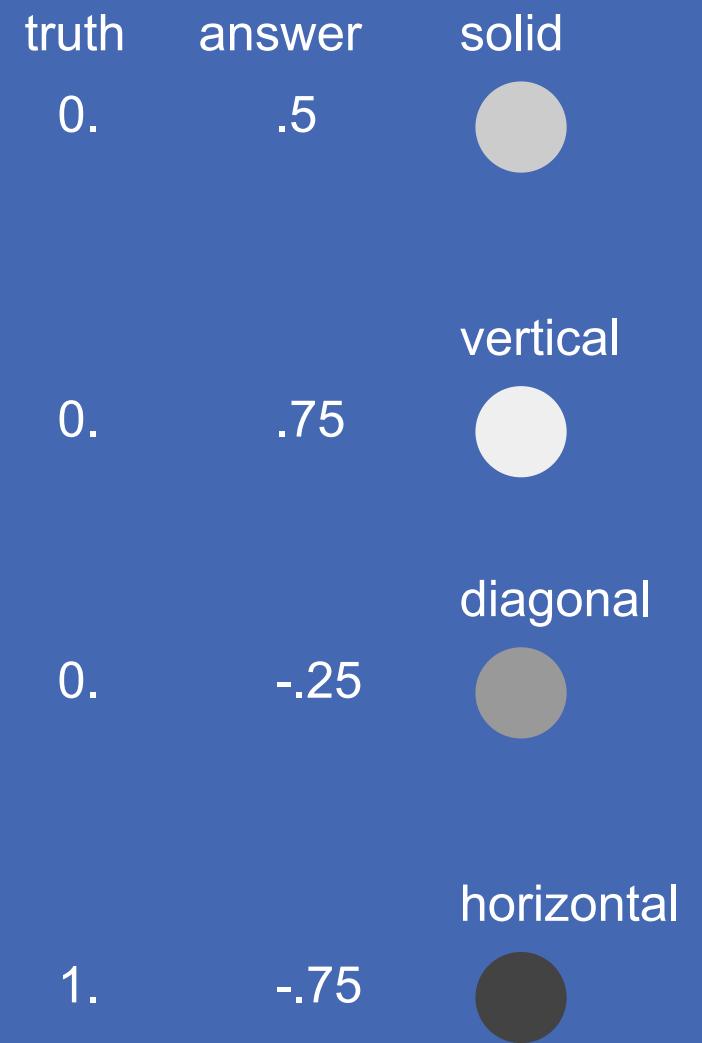
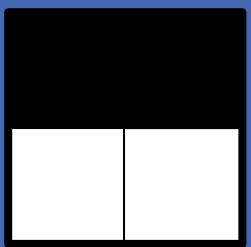


horizontal

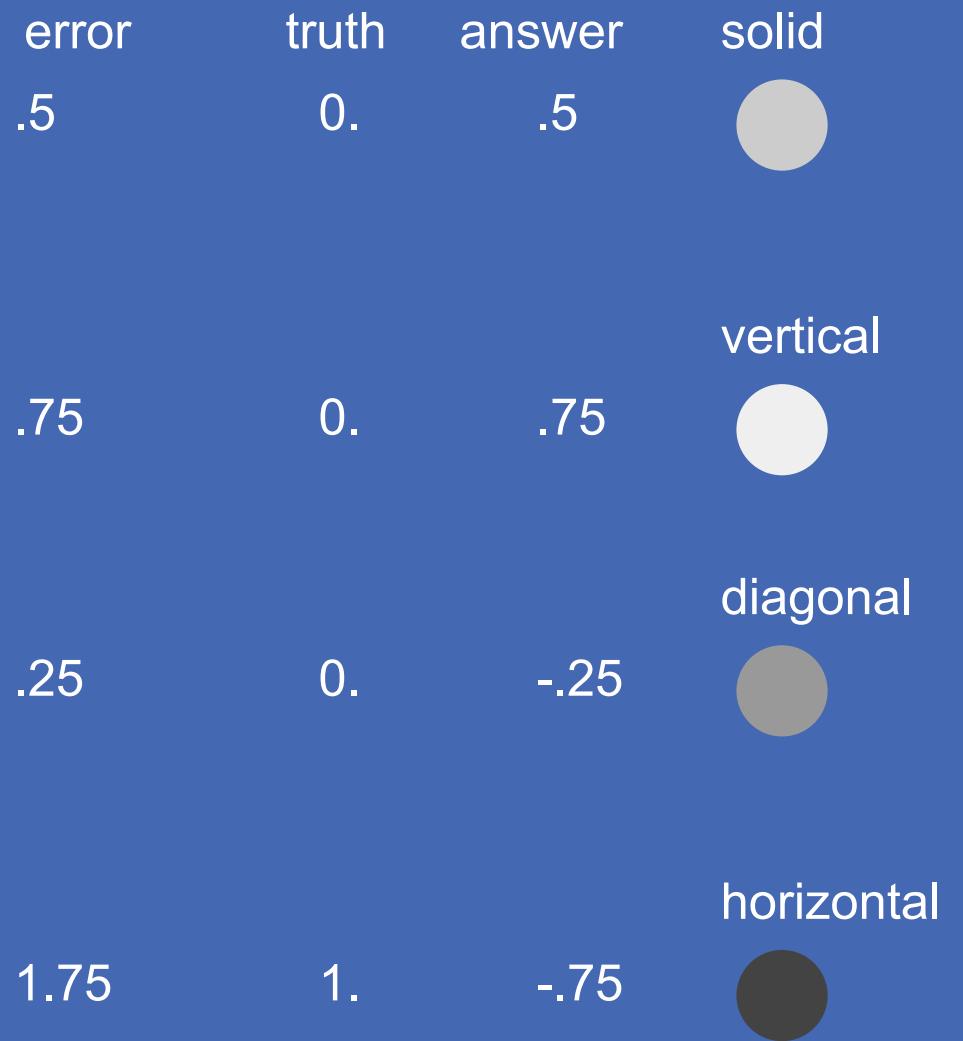
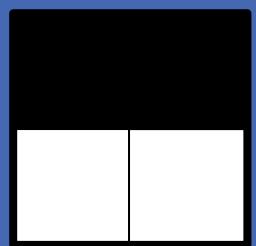
1.



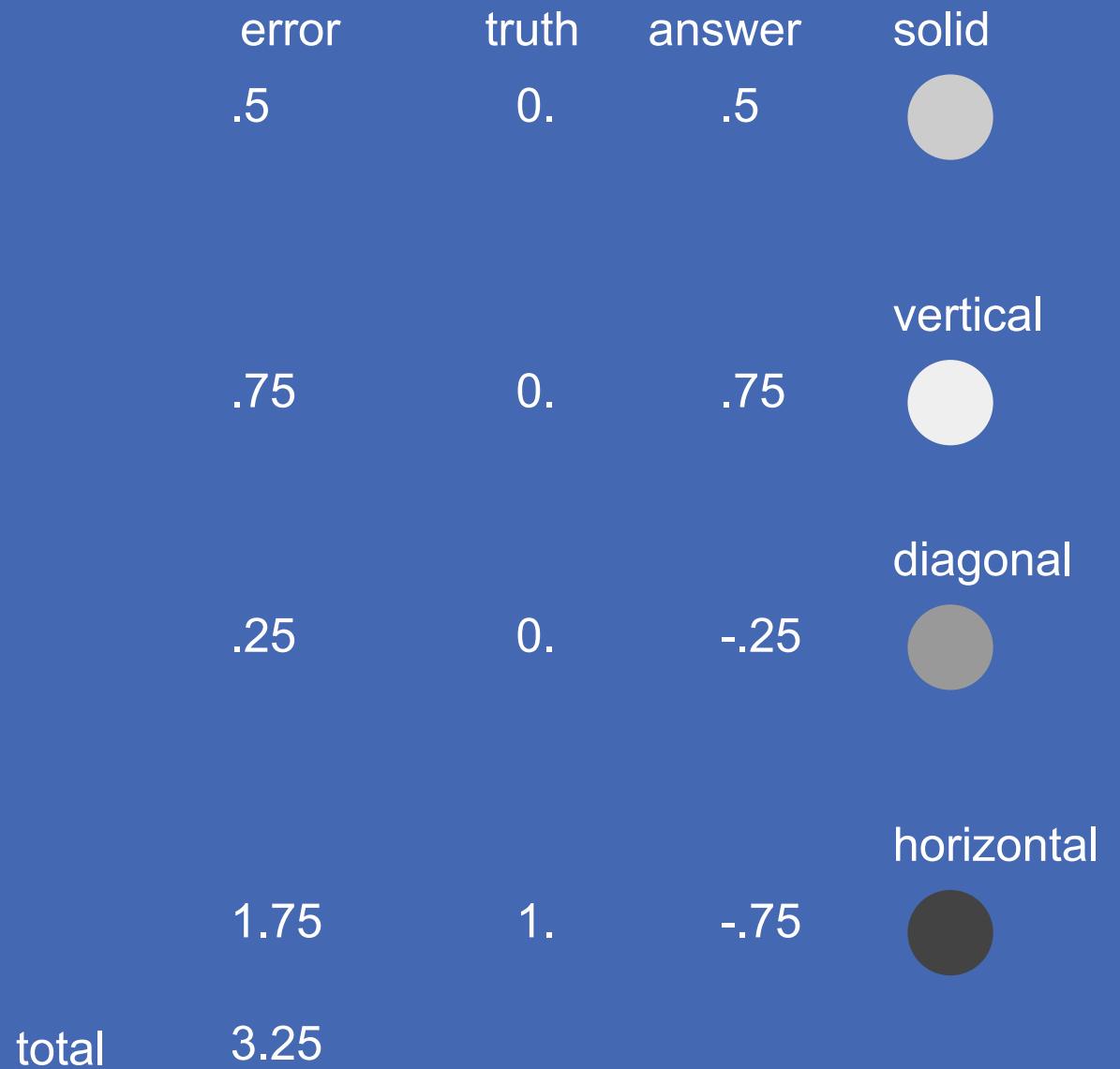
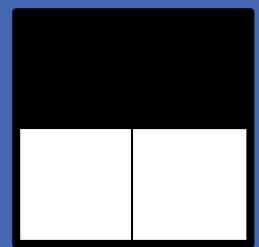
Errors



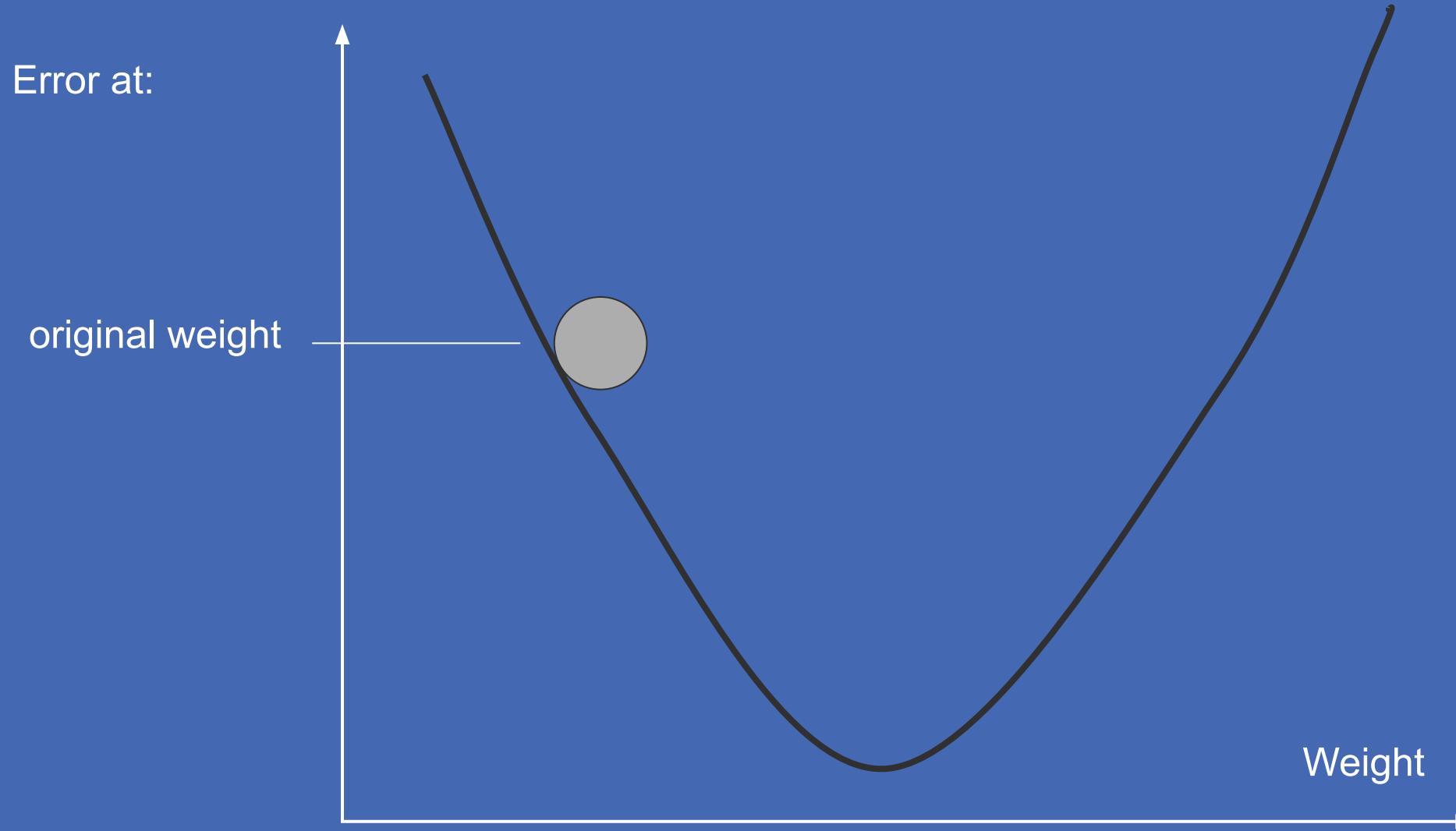
Errors



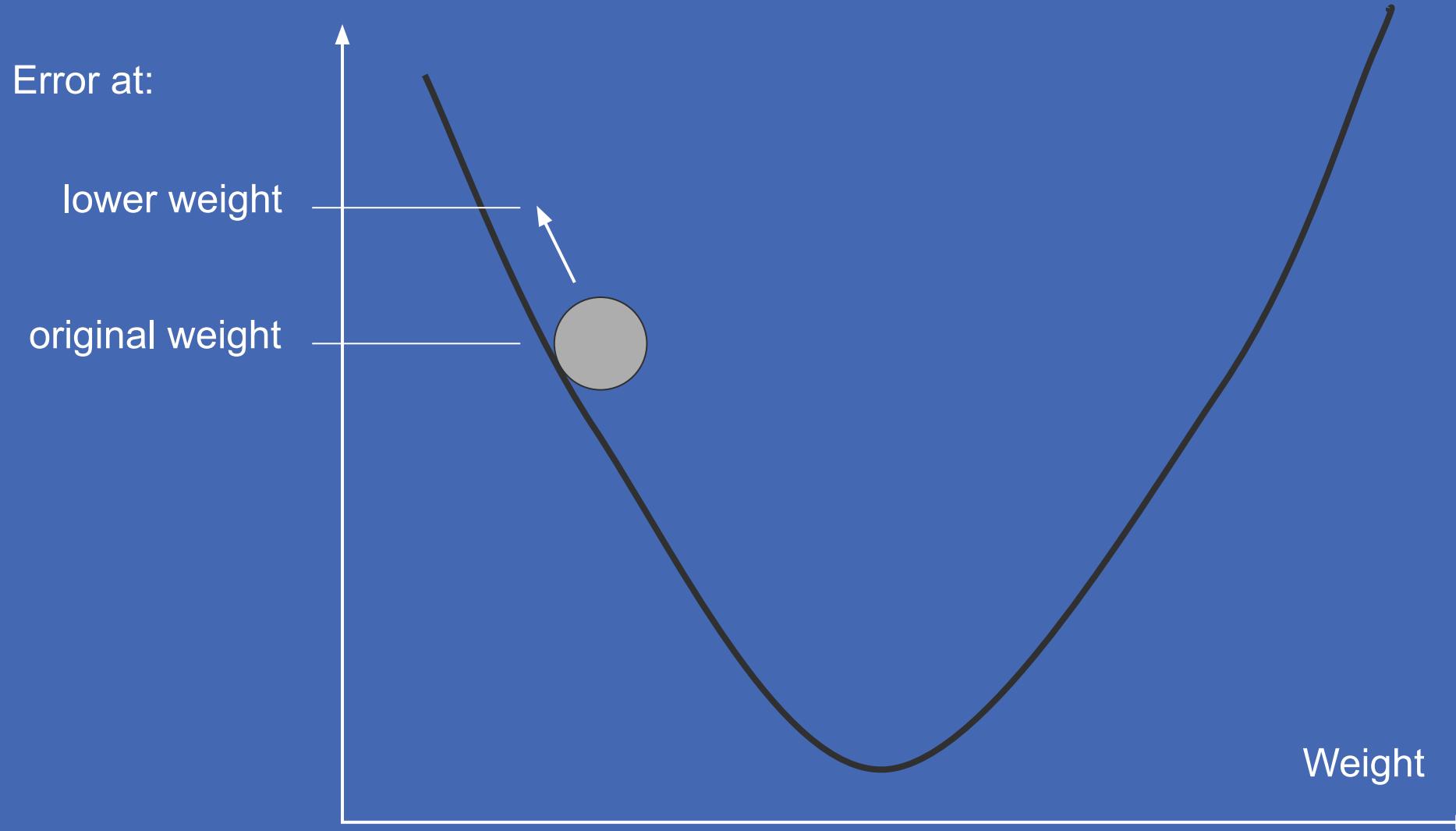
Errors



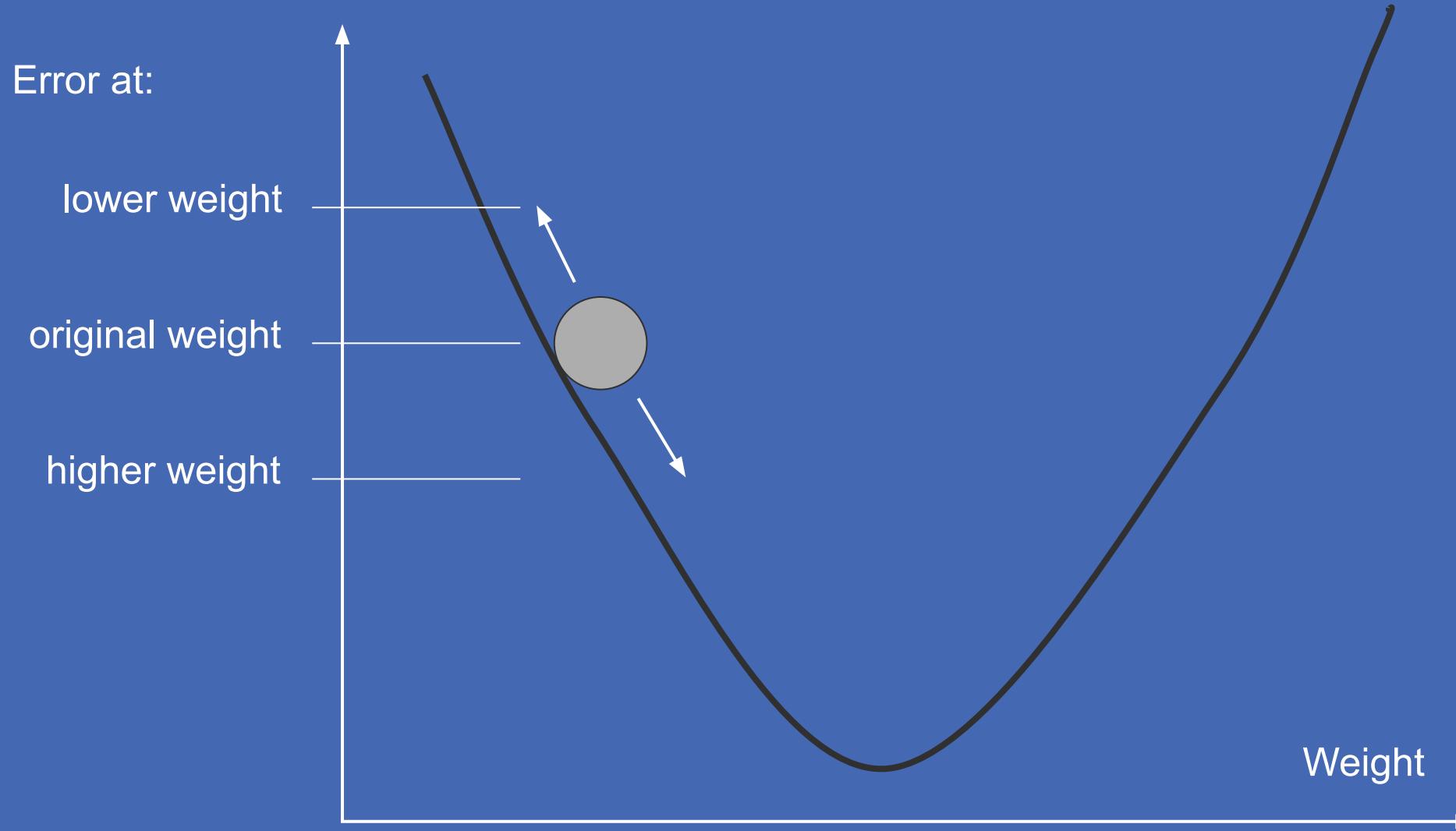
Learn all the weights: Gradient descent



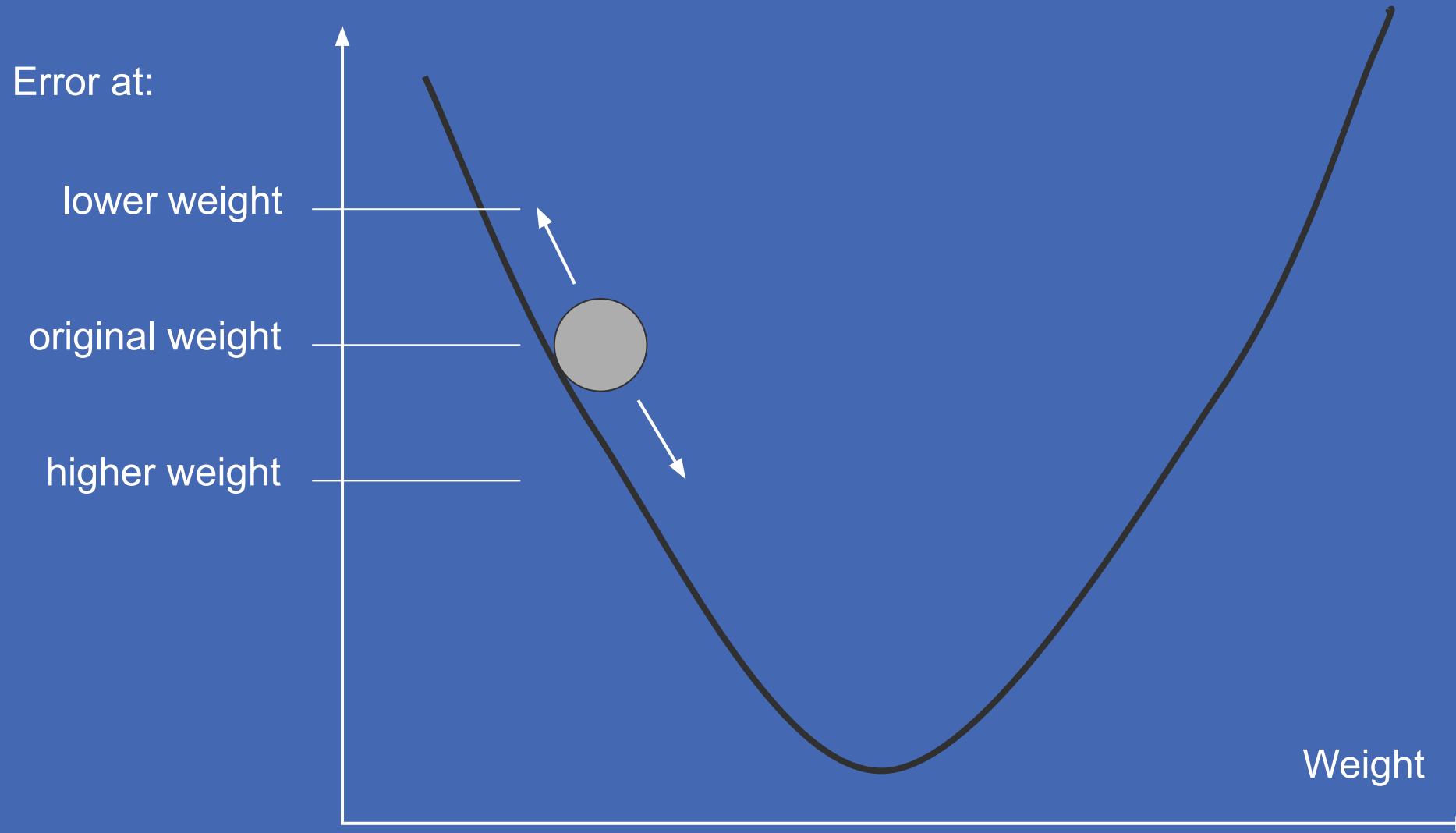
Learn all the weights: Gradient descent



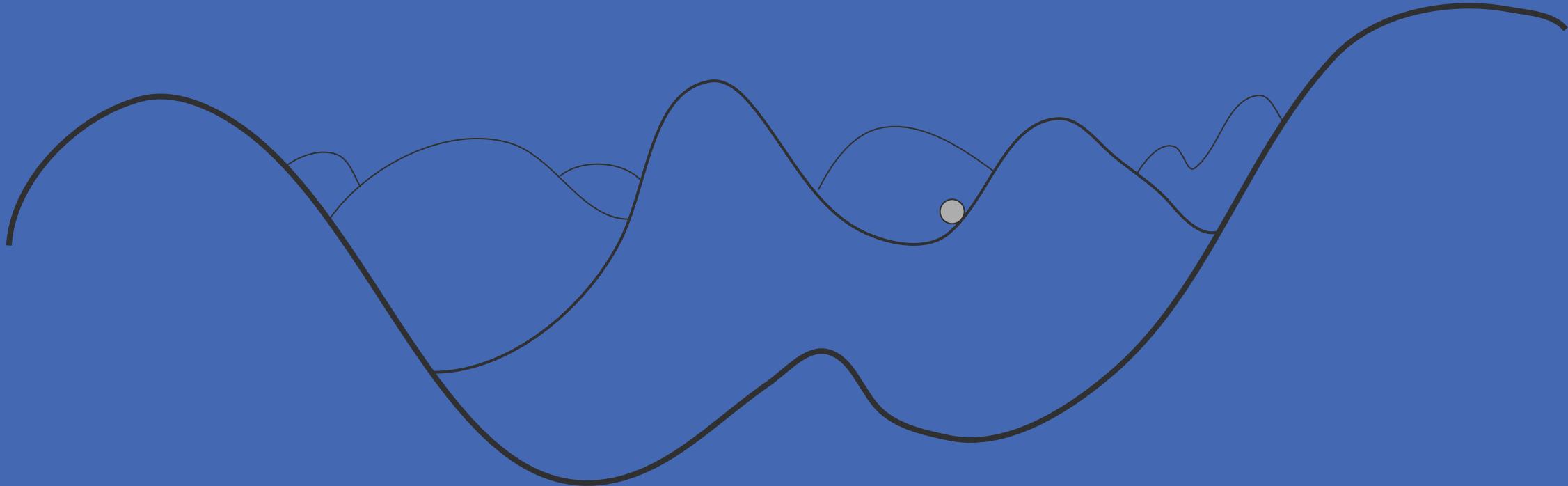
Learn all the weights: Gradient descent



Numerically calculating the gradient is expensive

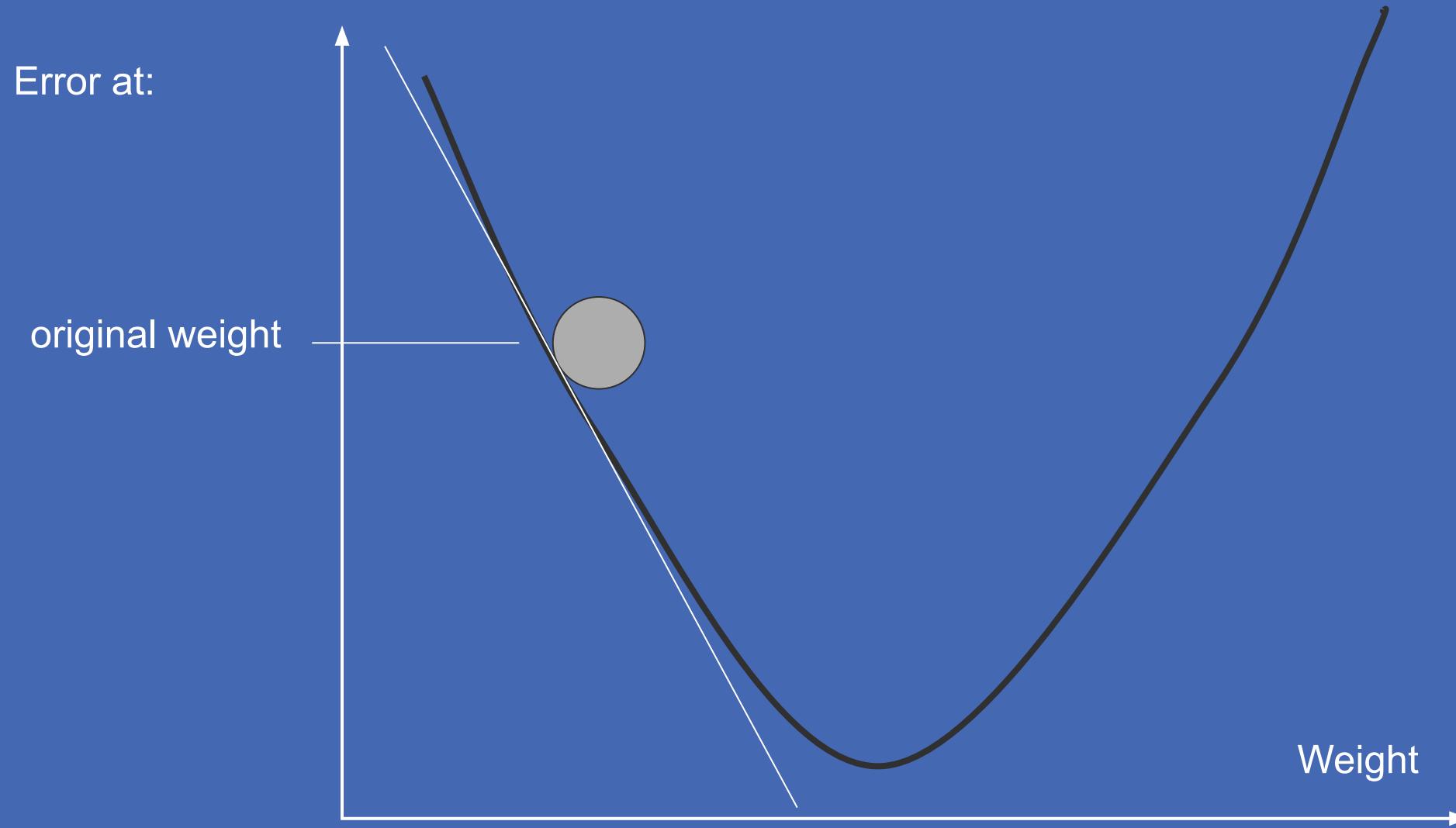


Numerically calculating the gradient is very expensive

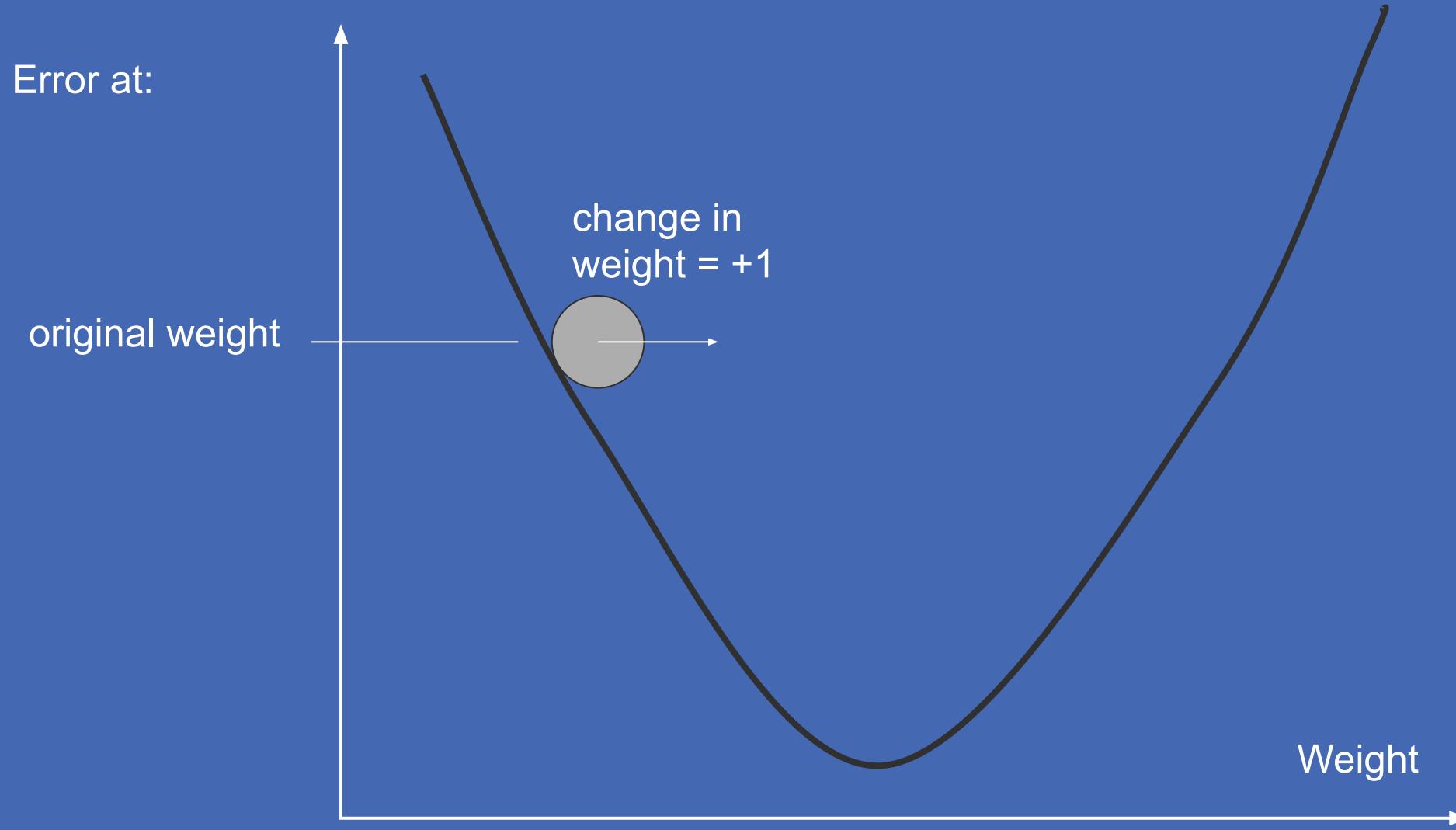


How backpropagation works

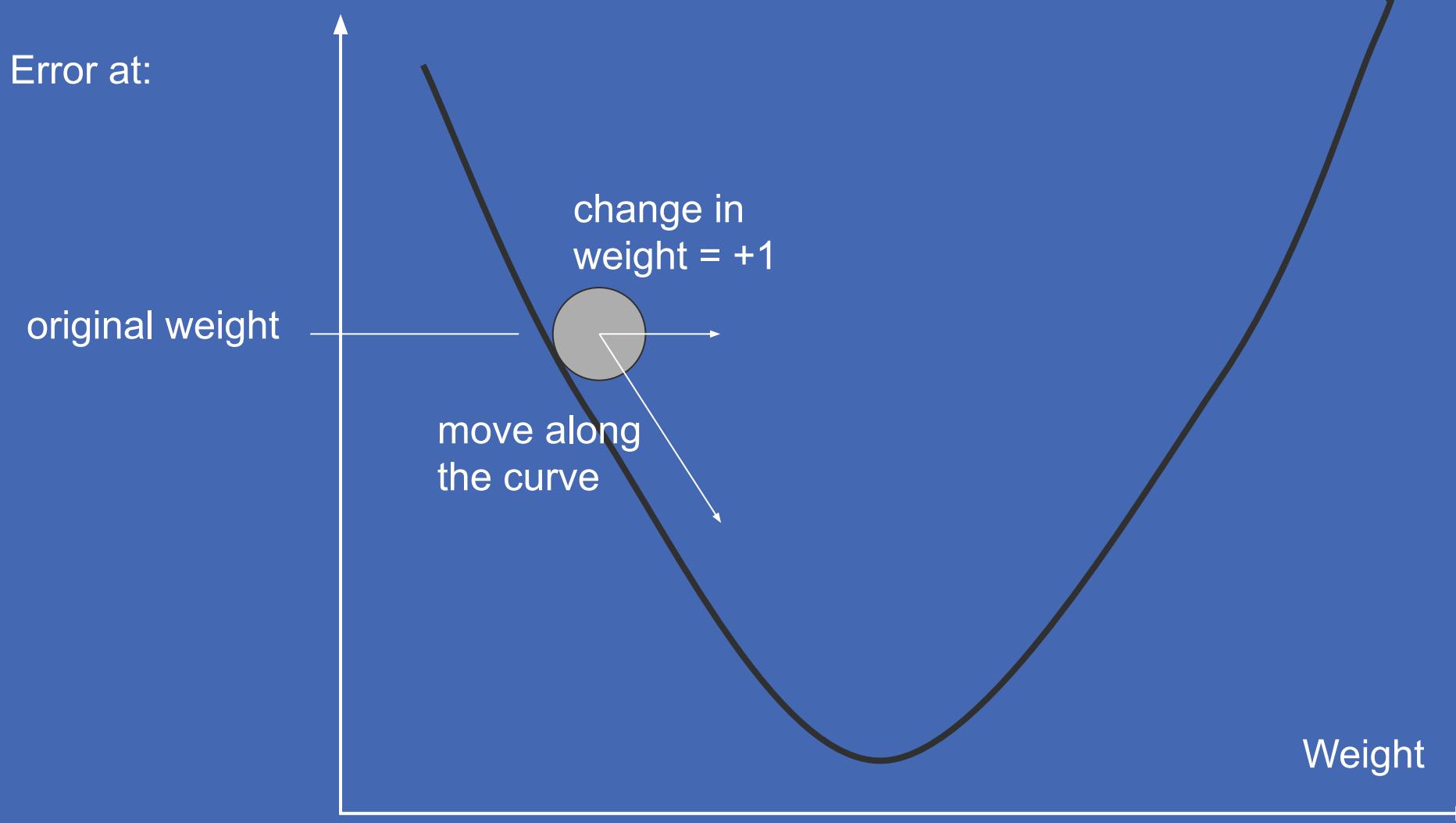
Calculate the gradient (slope) directly



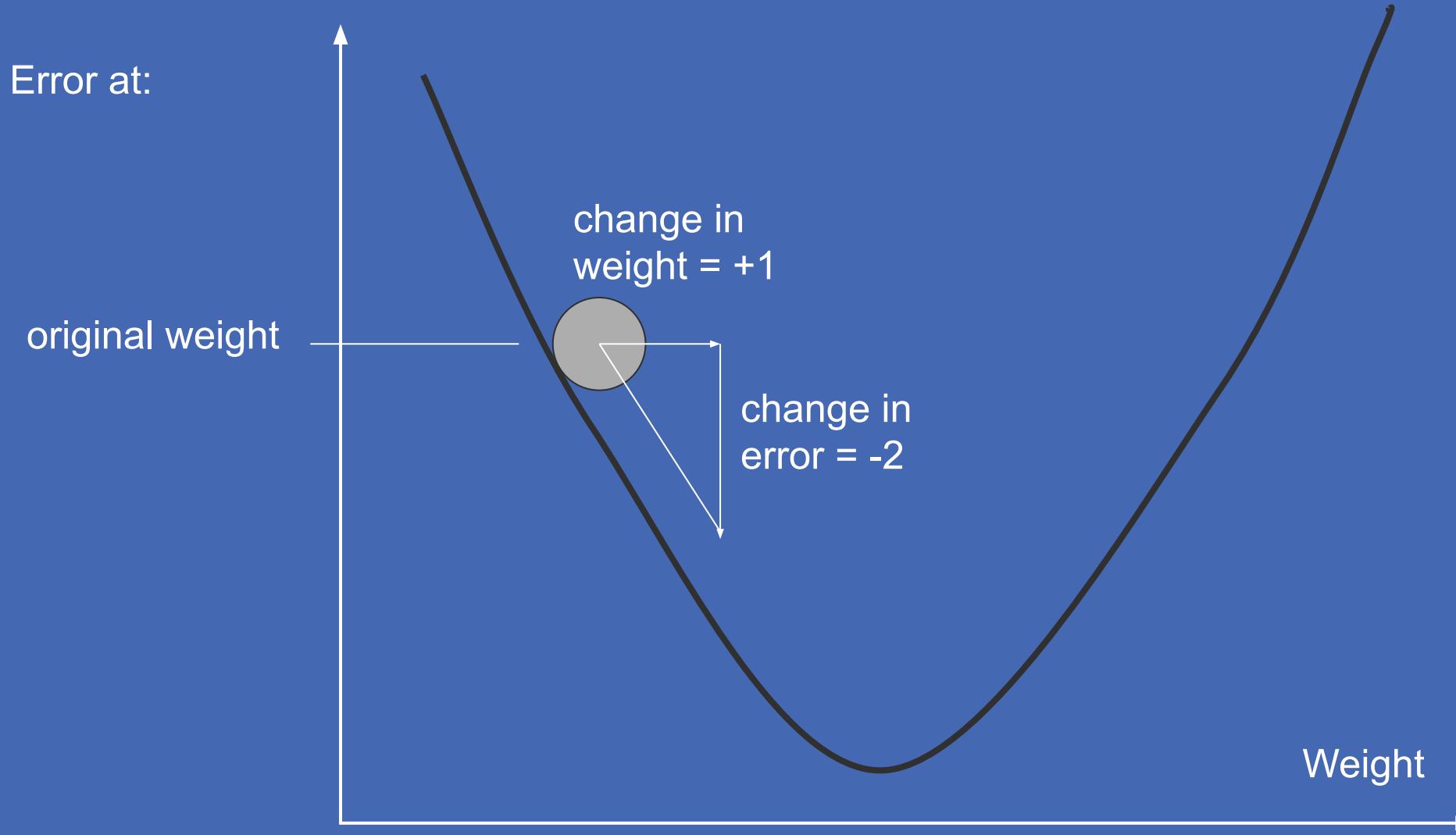
Slope



Slope

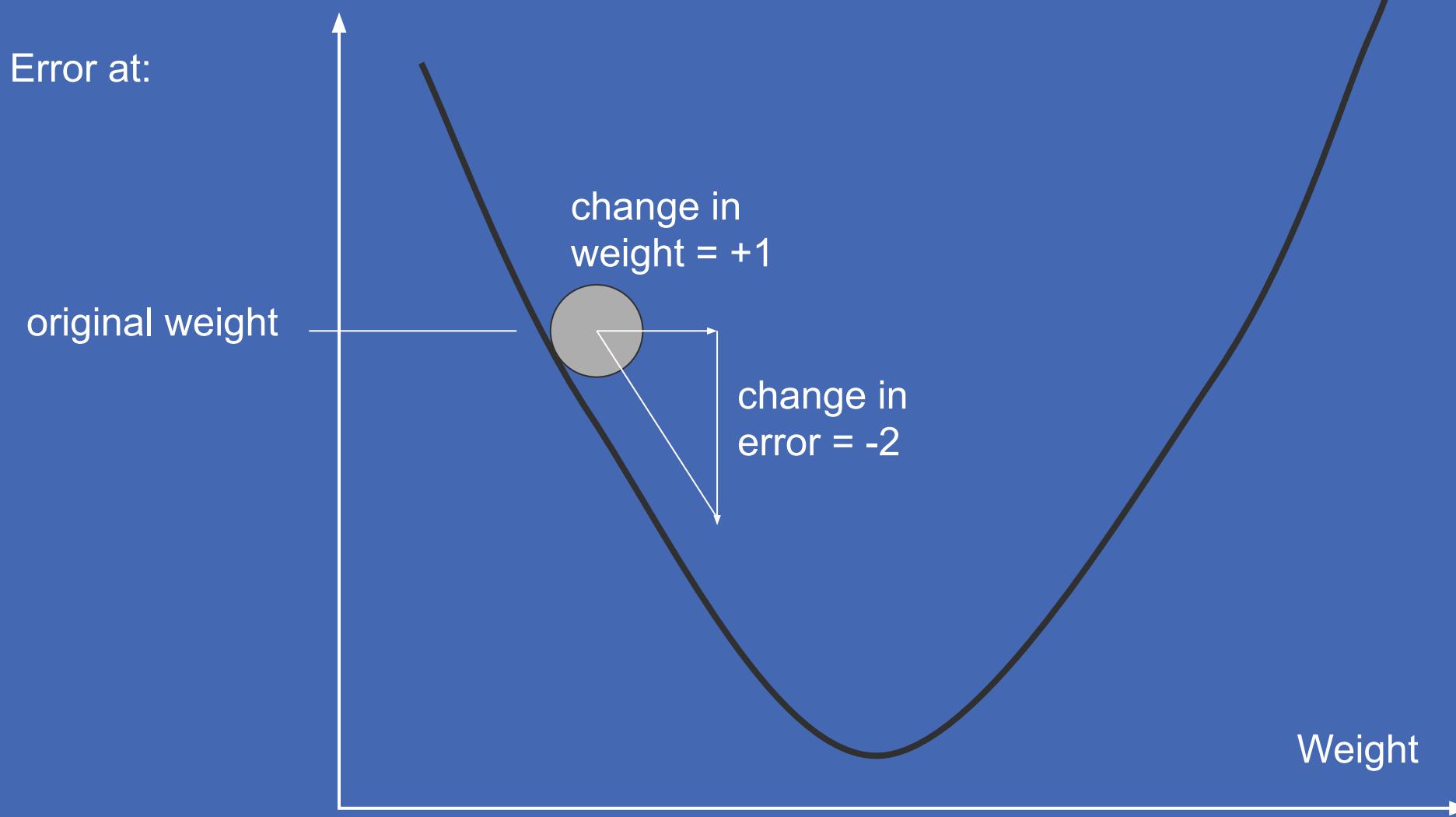


Slope

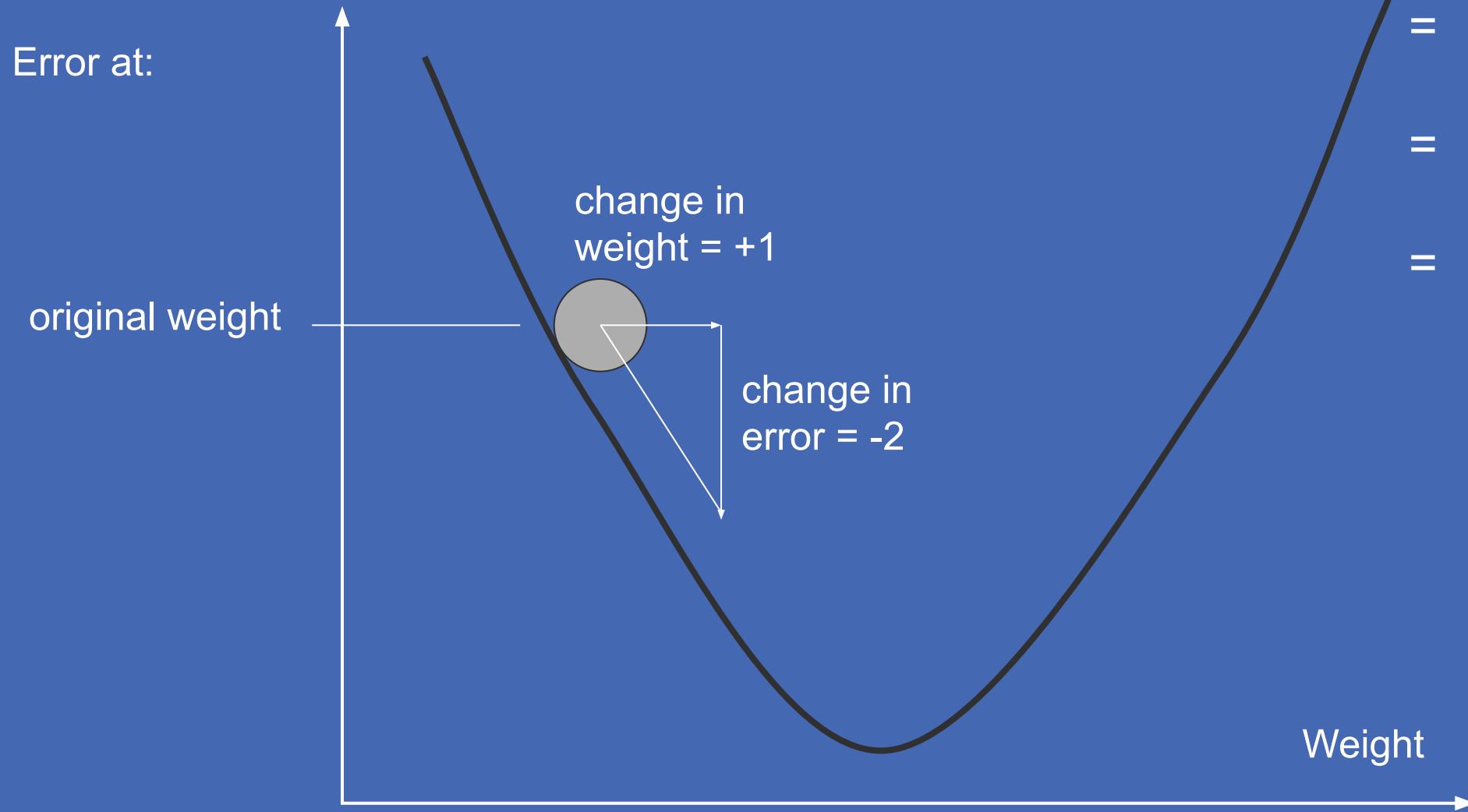


Slope

$$\text{slope} = \frac{\text{change in error}}{\text{change in weight}}$$



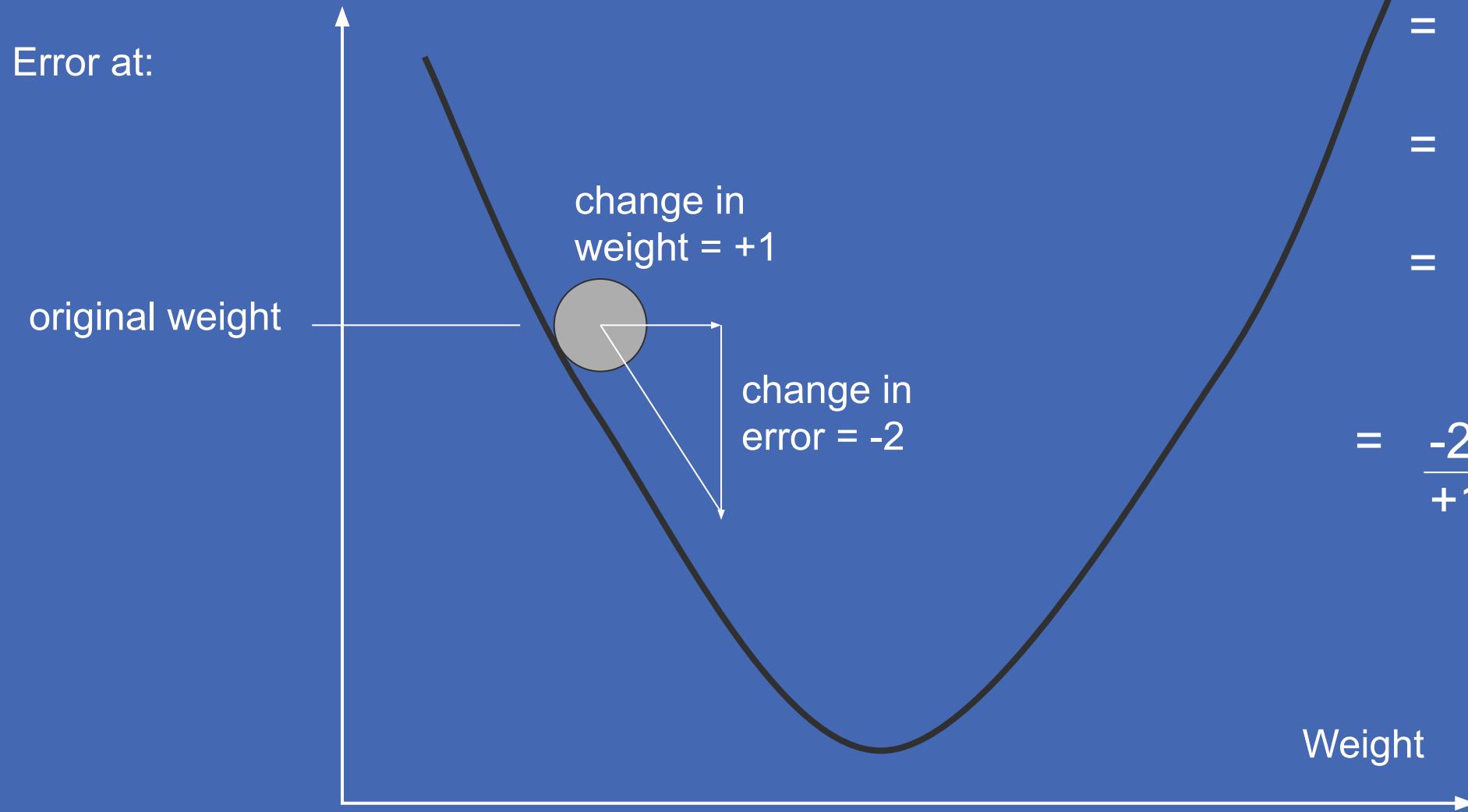
Slope



$$\text{slope} = \frac{\text{change in error}}{\text{change in weight}}$$

$$\begin{aligned} &= \frac{\Delta \text{error}}{\Delta \text{weight}} \\ &= \frac{d(\text{error})}{d(\text{weight})} \\ &= \frac{\partial e}{\partial w} \end{aligned}$$

Slope



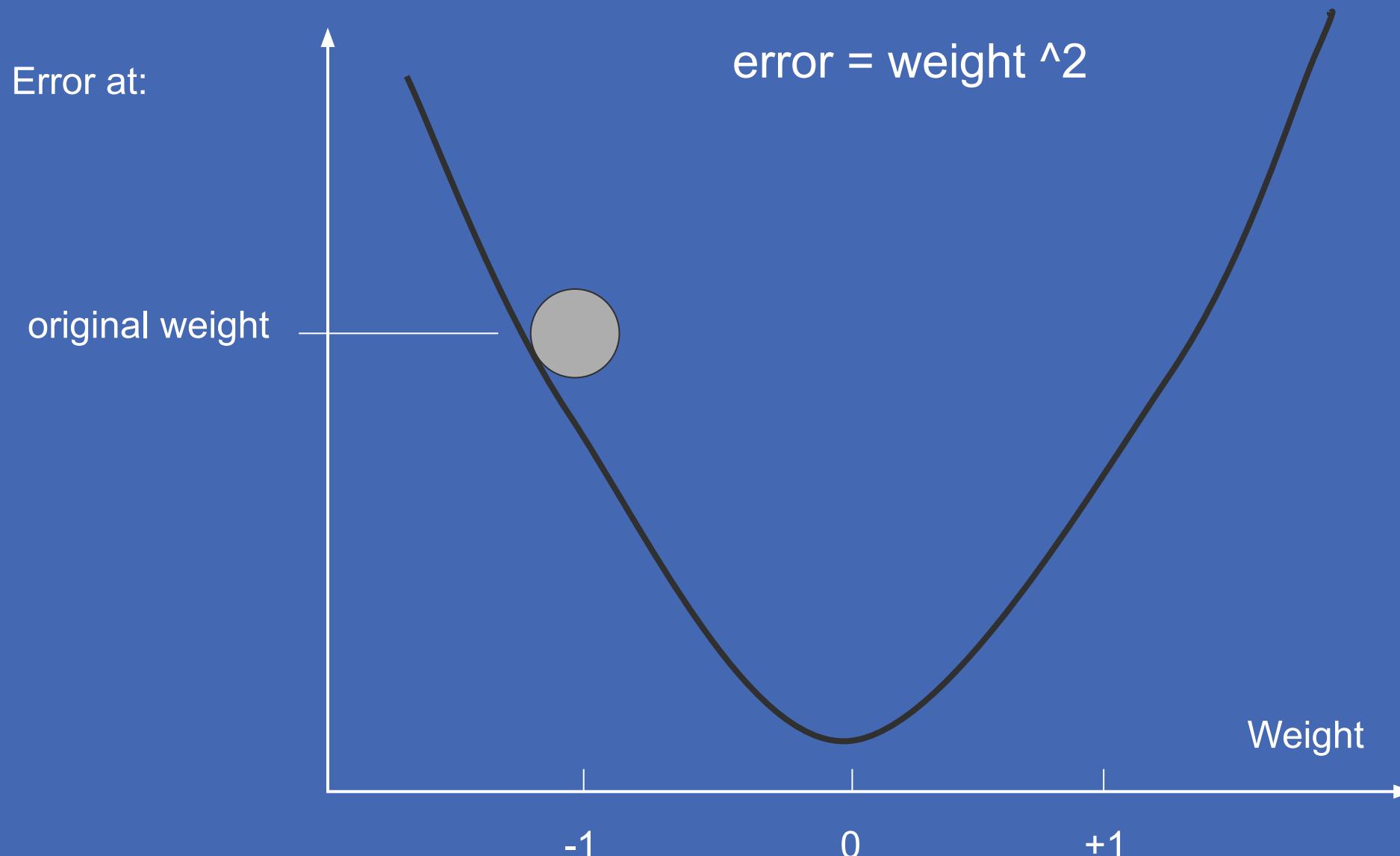
$$\text{slope} = \frac{\text{change in error}}{\text{change in weight}}$$

$$\begin{aligned} &= \frac{\Delta \text{error}}{\Delta \text{weight}} \\ &= \frac{d(\text{error})}{d(\text{weight})} \\ &= \frac{\partial e}{\partial w} \end{aligned}$$

$$= \frac{-2}{+1} = -2$$

Slope

You have to know your error function.
For example:

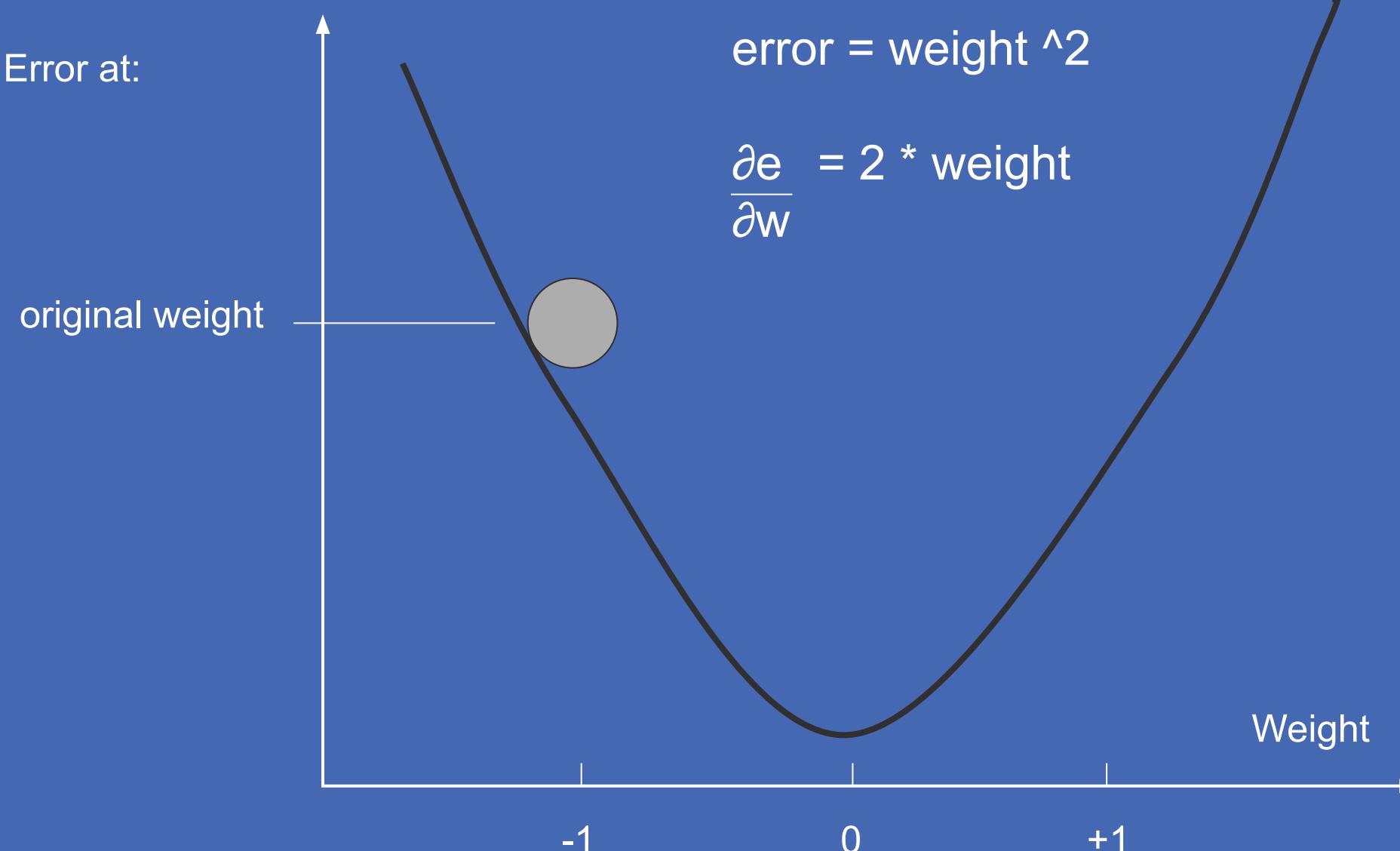


Slope

You have to know your error function.
For example:

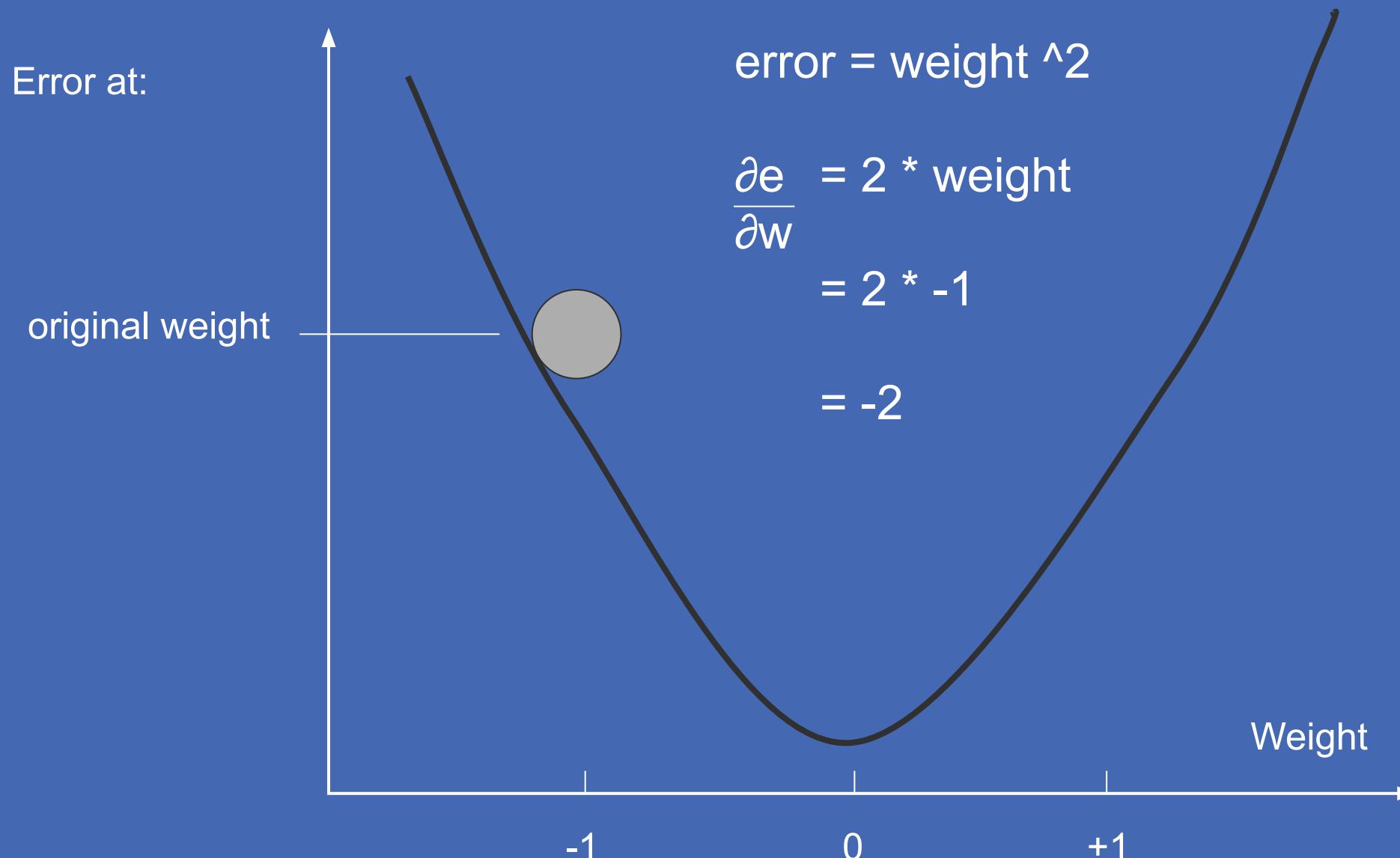
$$\text{error} = \text{weight}^2$$

$$\frac{\partial e}{\partial w} = 2 * \text{weight}$$



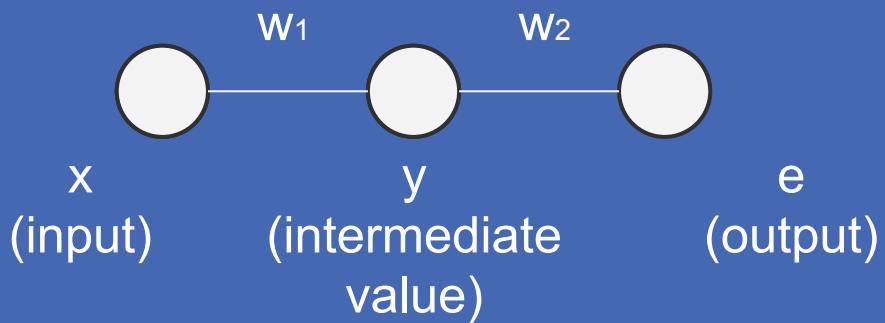
Slope

You have to know your error function.
For example:



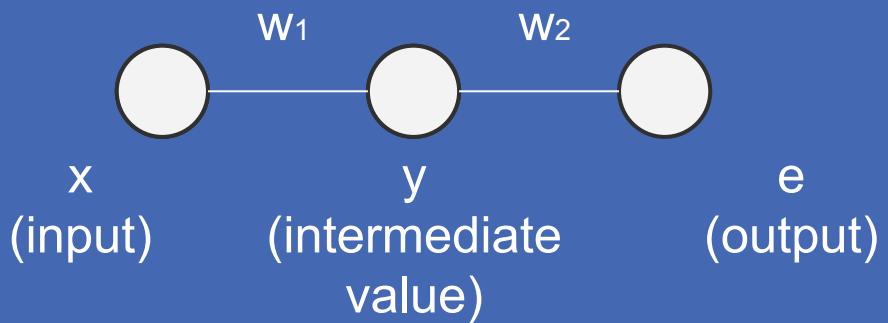
Chaining

$$y = x * w_1$$



Chaining

$$\begin{aligned}y &= x * w_1 \\ \frac{\partial y}{\partial w_1} &= x\end{aligned}$$



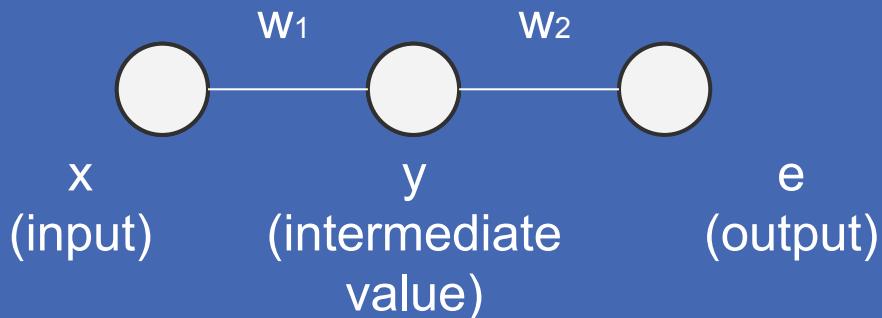
Chaining

$$y = x * w_1$$

$$\frac{\partial y}{\partial w_1} = x$$

$$e = y * w_2$$

$$\frac{\partial e}{\partial y} = w_2$$



Chaining

$$y = x * w_1$$

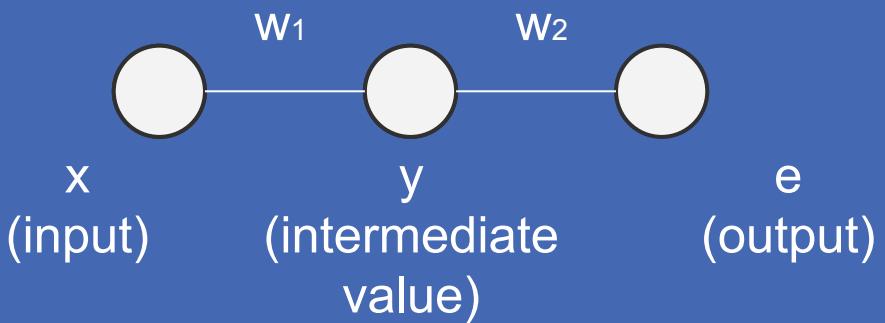
$$\frac{\partial y}{\partial w_1} = x$$

$$e = y * w_2$$

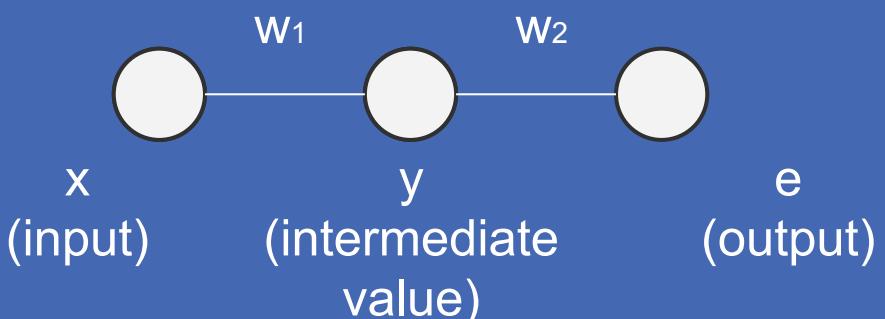
$$\frac{\partial e}{\partial y} = w_2$$

$$e = x * w_1 * w_2$$

$$\frac{\partial e}{\partial w_1} = x * w_2$$



Chaining



$$\frac{\partial y}{\partial w_1} = x$$

$$\frac{\partial e}{\partial y} = w_2$$

$$\frac{\partial e}{\partial w_1} = x * w_2$$

$$\frac{\partial e}{\partial w_1} = \left(\frac{\partial y}{\partial w_1} * \frac{\partial e}{\partial y} \right)$$

Chaining

$$y = x * w_1$$

$$\frac{\partial y}{\partial w_1} = x$$

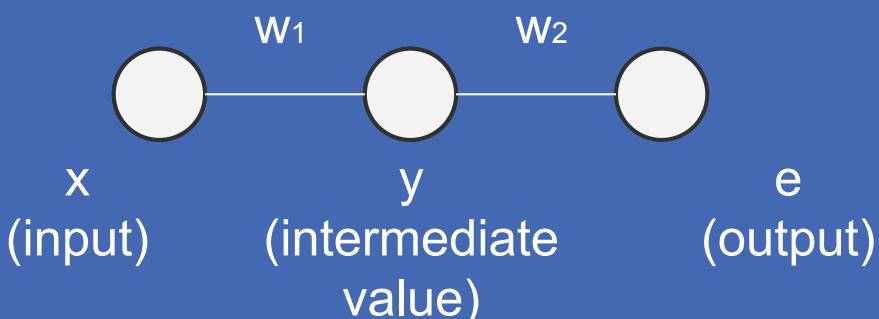
$$e = y * w_2$$

$$\frac{\partial e}{\partial y} = w_2$$

$$e = x * w_1 * w_2$$

$$\frac{\partial e}{\partial w_1} = x * w_2$$

$$\frac{\partial e}{\partial w_1} = \frac{\partial y}{\partial w_1} * \frac{\partial e}{\partial y}$$



Chaining

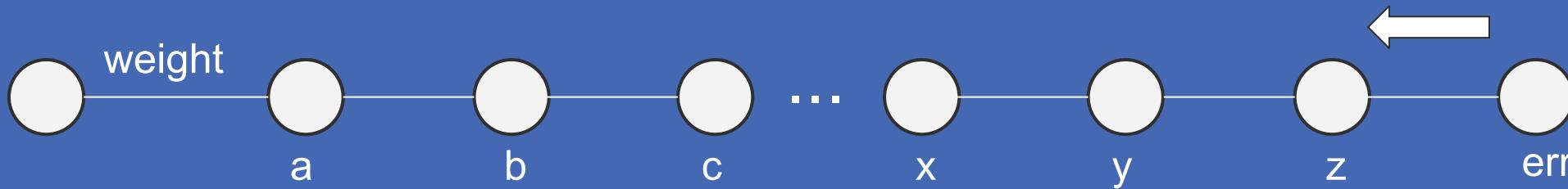
$$\frac{\partial \text{err}}{\partial \text{weight}} = \frac{\partial a}{\partial \text{weight}} * \frac{\partial b}{\partial a} * \frac{\partial c}{\partial b} * \frac{\partial d}{\partial c} * \dots * \frac{\partial y}{\partial x} * \frac{\partial z}{\partial y} * \frac{\partial \text{err}}{\partial z}$$



Chaining

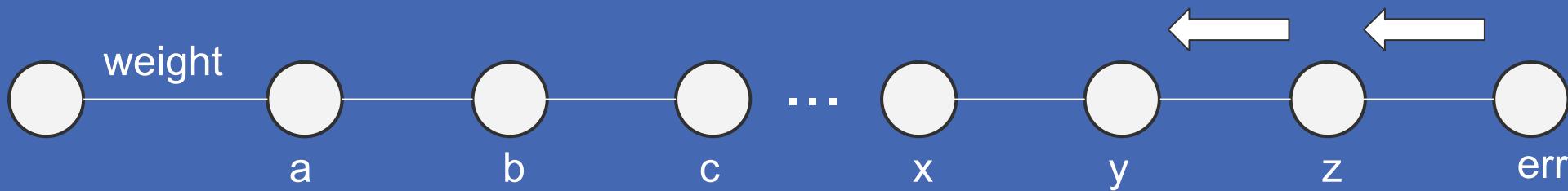


$$\frac{\partial \text{err}}{\partial \text{weight}} = \frac{\partial a}{\partial \text{weight}} * \frac{\partial b}{\partial a} * \frac{\partial c}{\partial b} * \frac{\partial d}{\partial c} * \dots * \frac{\partial y}{\partial x} * \frac{\partial z}{\partial y} * \frac{\partial \text{err}}{\partial z}$$



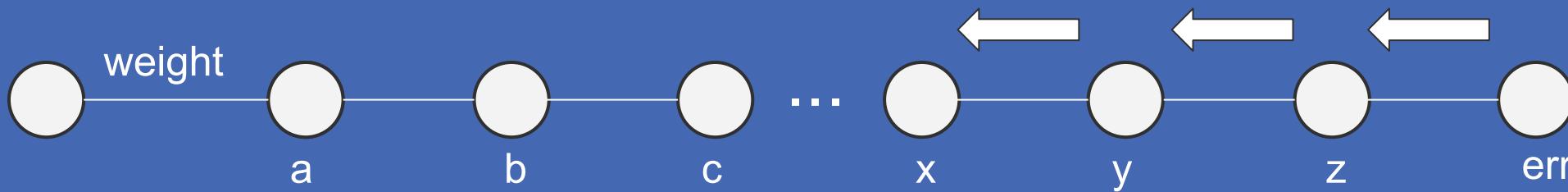
Chaining

$$\frac{\partial \text{err}}{\partial \text{weight}} = \frac{\partial a}{\partial \text{weight}} * \frac{\partial b}{\partial a} * \frac{\partial c}{\partial b} * \frac{\partial d}{\partial c} * \dots * \frac{\partial y}{\partial x} * \frac{\partial z}{\partial y} * \frac{\partial \text{err}}{\partial z}$$



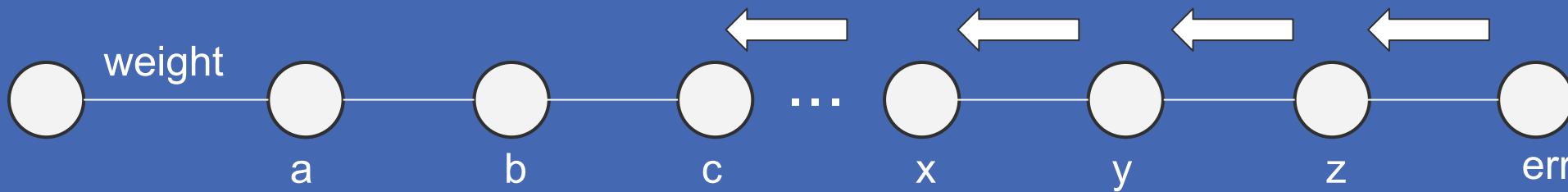
Chaining

$$\frac{\partial \text{err}}{\partial \text{weight}} = \frac{\partial a}{\partial \text{weight}} * \frac{\partial b}{\partial a} * \frac{\partial c}{\partial b} * \frac{\partial d}{\partial c} * \dots * \frac{\partial y}{\partial x} * \frac{\partial z}{\partial y} * \frac{\partial \text{err}}{\partial z}$$



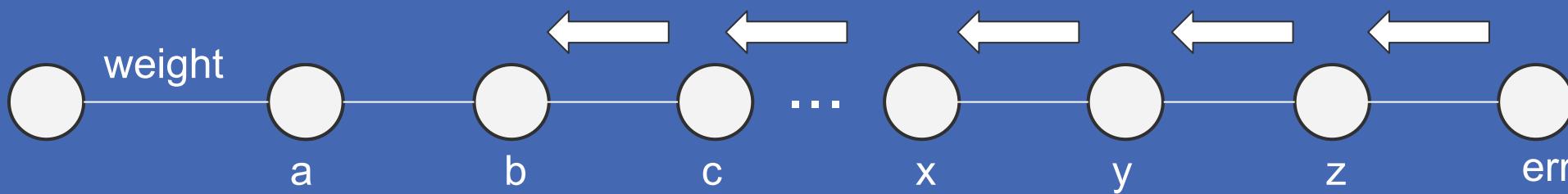
Chaining

$$\frac{\partial \text{err}}{\partial \text{weight}} = \frac{\partial a}{\partial \text{weight}} * \frac{\partial b}{\partial a} * \frac{\partial c}{\partial b} * \frac{\partial d}{\partial c} * \dots * \frac{\partial y}{\partial x} * \frac{\partial z}{\partial y} * \frac{\partial \text{err}}{\partial z}$$



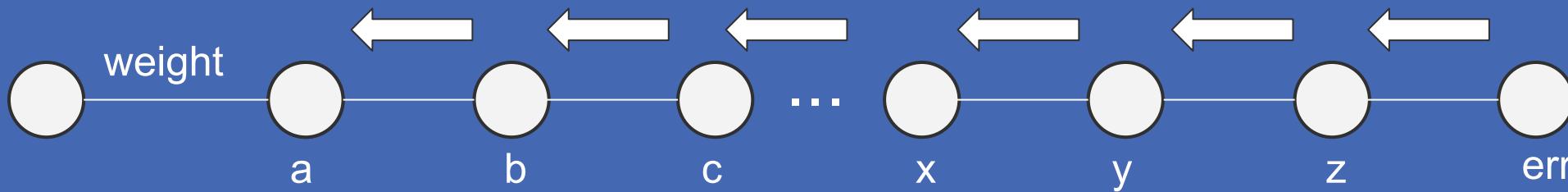
Chaining

$$\frac{\partial \text{err}}{\partial \text{weight}} = \frac{\partial a}{\partial \text{weight}} * \frac{\partial b}{\partial a} * \frac{\partial c}{\partial b} * \frac{\partial d}{\partial c} * \dots * \frac{\partial y}{\partial x} * \frac{\partial z}{\partial y} * \frac{\partial \text{err}}{\partial z}$$



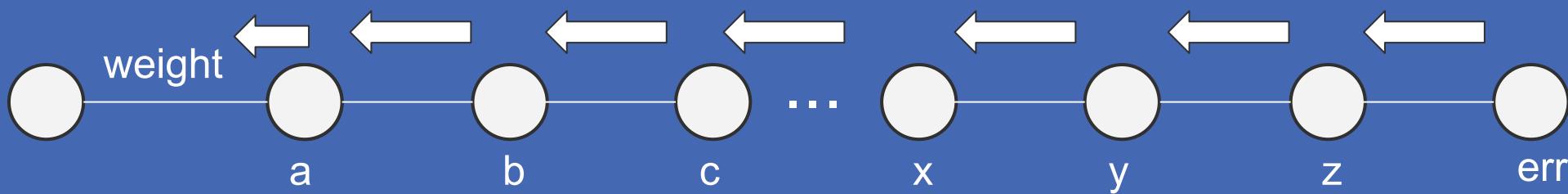
Chaining

$$\frac{\partial \text{err}}{\partial \text{weight}} = \frac{\partial a}{\partial \text{weight}} * \frac{\partial b}{\partial a} * \frac{\partial c}{\partial b} * \frac{\partial d}{\partial c} * \dots * \frac{\partial y}{\partial x} * \frac{\partial z}{\partial y} * \frac{\partial \text{err}}{\partial z}$$

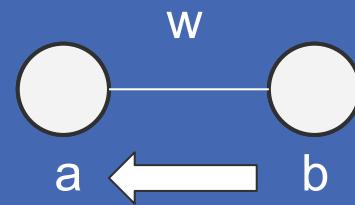


Chaining

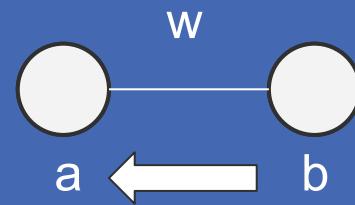
$$\frac{\partial \text{err}}{\partial \text{weight}} = \frac{\partial a}{\partial \text{weight}} * \frac{\partial b}{\partial a} * \frac{\partial c}{\partial b} * \frac{\partial d}{\partial c} * \dots * \frac{\partial y}{\partial x} * \frac{\partial z}{\partial y} * \frac{\partial \text{err}}{\partial z}$$



Backpropagation challenge: weights

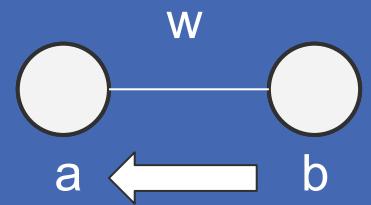


Backpropagation challenge: weights



$$\frac{\partial \text{err}}{\partial a} = \frac{\partial b}{\partial a} * \frac{\partial \text{err}}{\partial b}$$

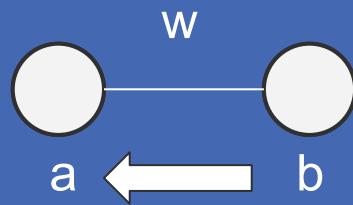
Backpropagation challenge: weights



$$b = wa$$

$$\frac{\partial \text{err}}{\partial a} = \frac{\partial b}{\partial a} * \frac{\partial \text{err}}{\partial b}$$

Backpropagation challenge: weights

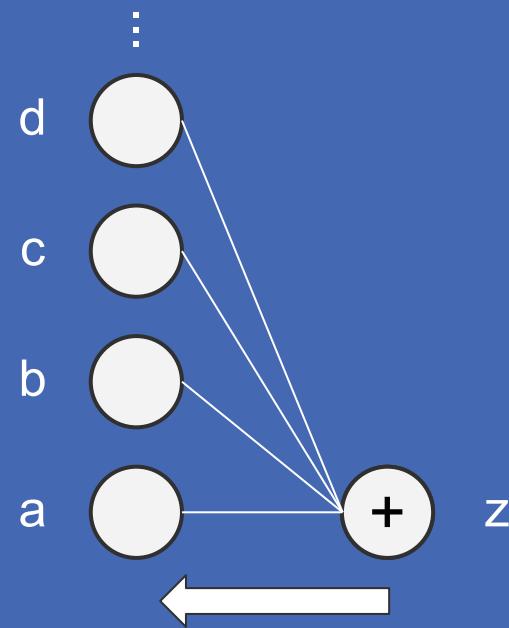


$$\frac{\partial \text{err}}{\partial a} = \frac{\partial b}{\partial a} * \frac{\partial \text{err}}{\partial b}$$

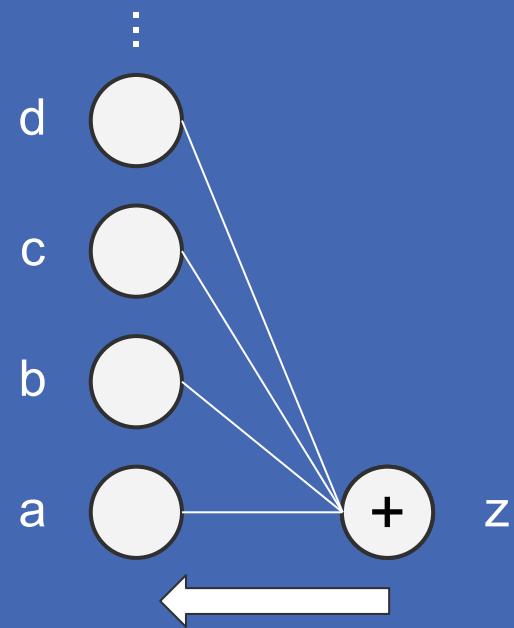
$$b = wa$$

$$\frac{\partial b}{\partial a} = w$$

Backpropagation challenge: sums

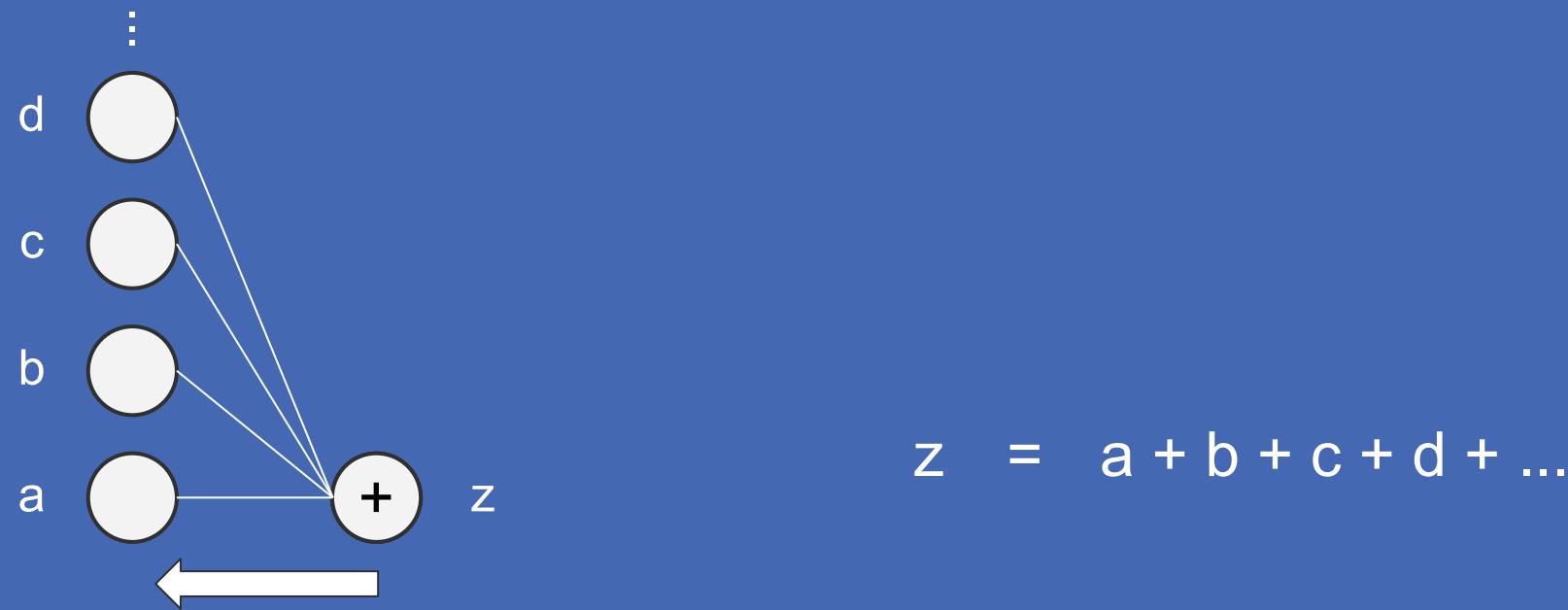


Backpropagation challenge: sums



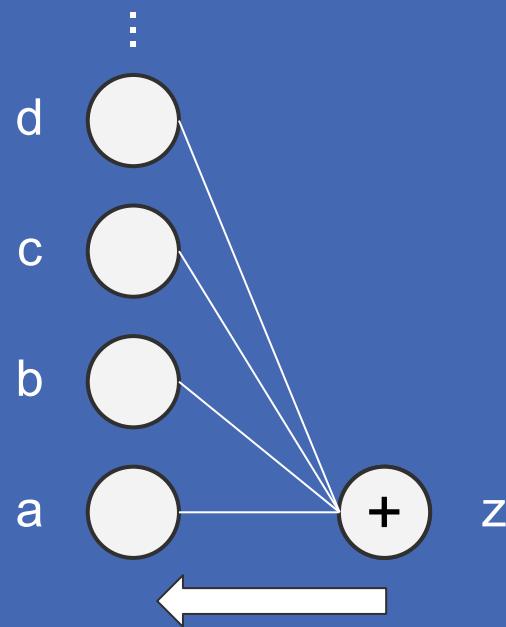
$$\frac{\partial \text{err}}{\partial a} = \frac{\partial z}{\partial a} * \frac{\partial \text{err}}{\partial z}$$

Backpropagation challenge: sums



$$\frac{\partial \text{err}}{\partial a} = \frac{\partial z}{\partial a} * \frac{\partial \text{err}}{\partial z}$$

Backpropagation challenge: sums

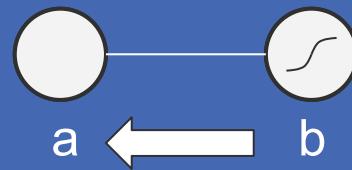


$$\frac{\partial \text{err}}{\partial a} = \frac{\partial z}{\partial a} * \frac{\partial \text{err}}{\partial z}$$

$$z = a + b + c + d + \dots$$

$$\frac{\partial z}{\partial a} = 1$$

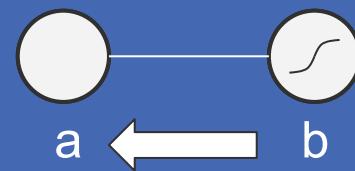
Backpropagation challenge: sigmoid



$$\frac{\partial \text{err}}{\partial a} = \frac{\partial b}{\partial a} * \frac{\partial \text{err}}{\partial b}$$

Backpropagation challenge: sigmoid

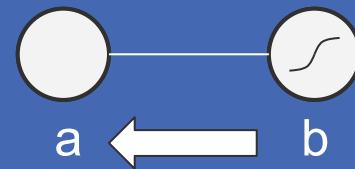
$$b = \frac{1}{1 + e^{-a}}$$



$$\frac{\partial \text{err}}{\partial a} = \frac{\partial b}{\partial a} * \frac{\partial \text{err}}{\partial b}$$

Backpropagation challenge: sigmoid

$$\begin{aligned} b &= \frac{1}{1 + e^{-a}} \\ &= \sigma(a) \end{aligned}$$

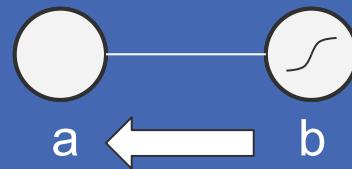


$$\frac{\partial \text{err}}{\partial a} = \frac{\partial b}{\partial a} * \frac{\partial \text{err}}{\partial b}$$

Backpropagation challenge: sigmoid

$$\begin{aligned} b &= \frac{1}{1 + e^{-a}} \\ &= \sigma(a) \end{aligned}$$

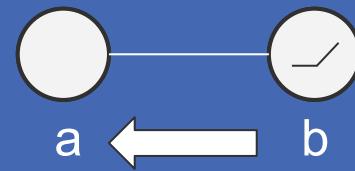
Because math is beautiful /
dumb luck:



$$\frac{\partial \text{err}}{\partial a} = \frac{\partial b}{\partial a} * \frac{\partial \text{err}}{\partial b}$$

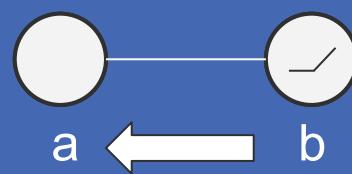
$$\frac{\partial b}{\partial a} = \sigma(a) * (1 - \sigma(a))$$

Backpropagation challenge: ReLU



$$\frac{\partial \text{err}}{\partial a} = \frac{\partial b}{\partial a} * \frac{\partial \text{err}}{\partial b}$$

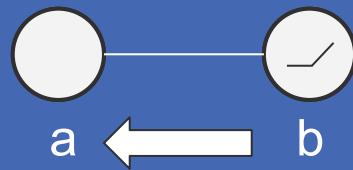
Backpropagation challenge: ReLU



$$\begin{aligned} b &= a, \quad a > 0 \\ &= 0, \quad \text{otherwise} \end{aligned}$$

$$\frac{\partial \text{err}}{\partial a} = \frac{\partial b}{\partial a} * \frac{\partial \text{err}}{\partial b}$$

Backpropagation challenge: ReLU

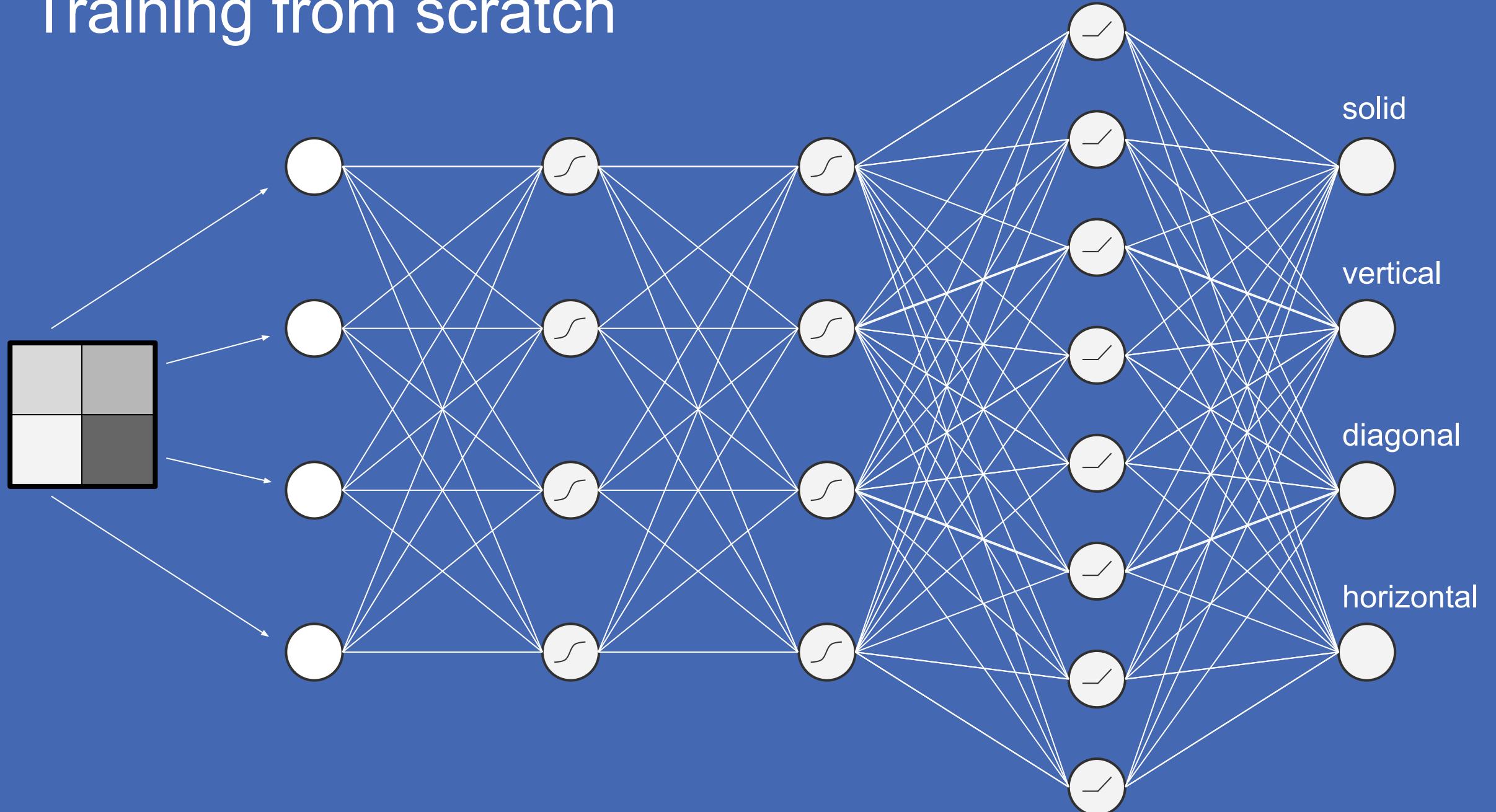


$$\frac{\partial \text{err}}{\partial a} = \frac{\partial b}{\partial a} * \frac{\partial \text{err}}{\partial b}$$

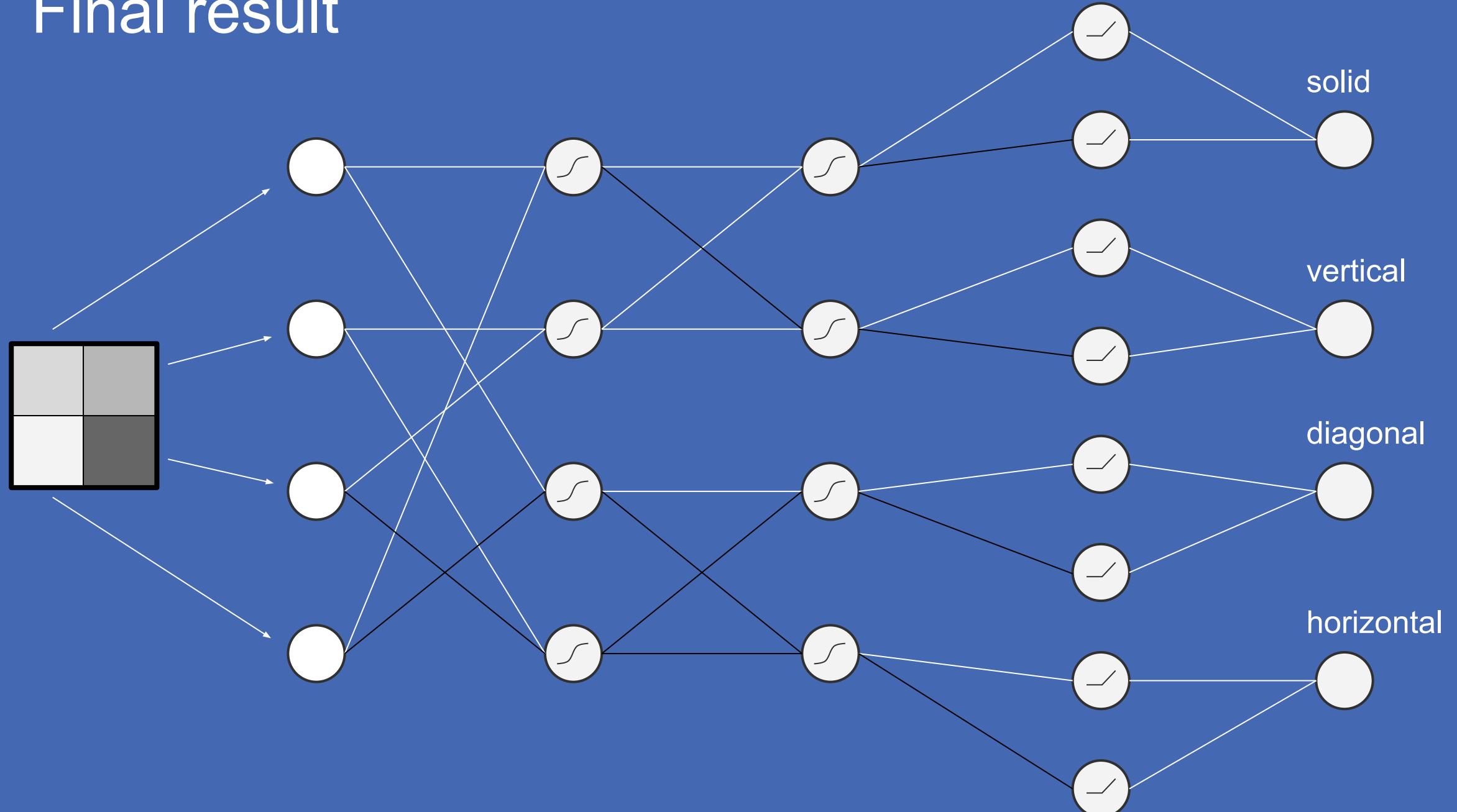
$$\begin{aligned} b &= a, \quad a > 0 \\ &= 0, \quad \text{otherwise} \end{aligned}$$

$$\begin{aligned} \frac{\partial b}{\partial a} &= 1, \quad a > 0 \\ &= 0, \quad \text{otherwise} \end{aligned}$$

Training from scratch

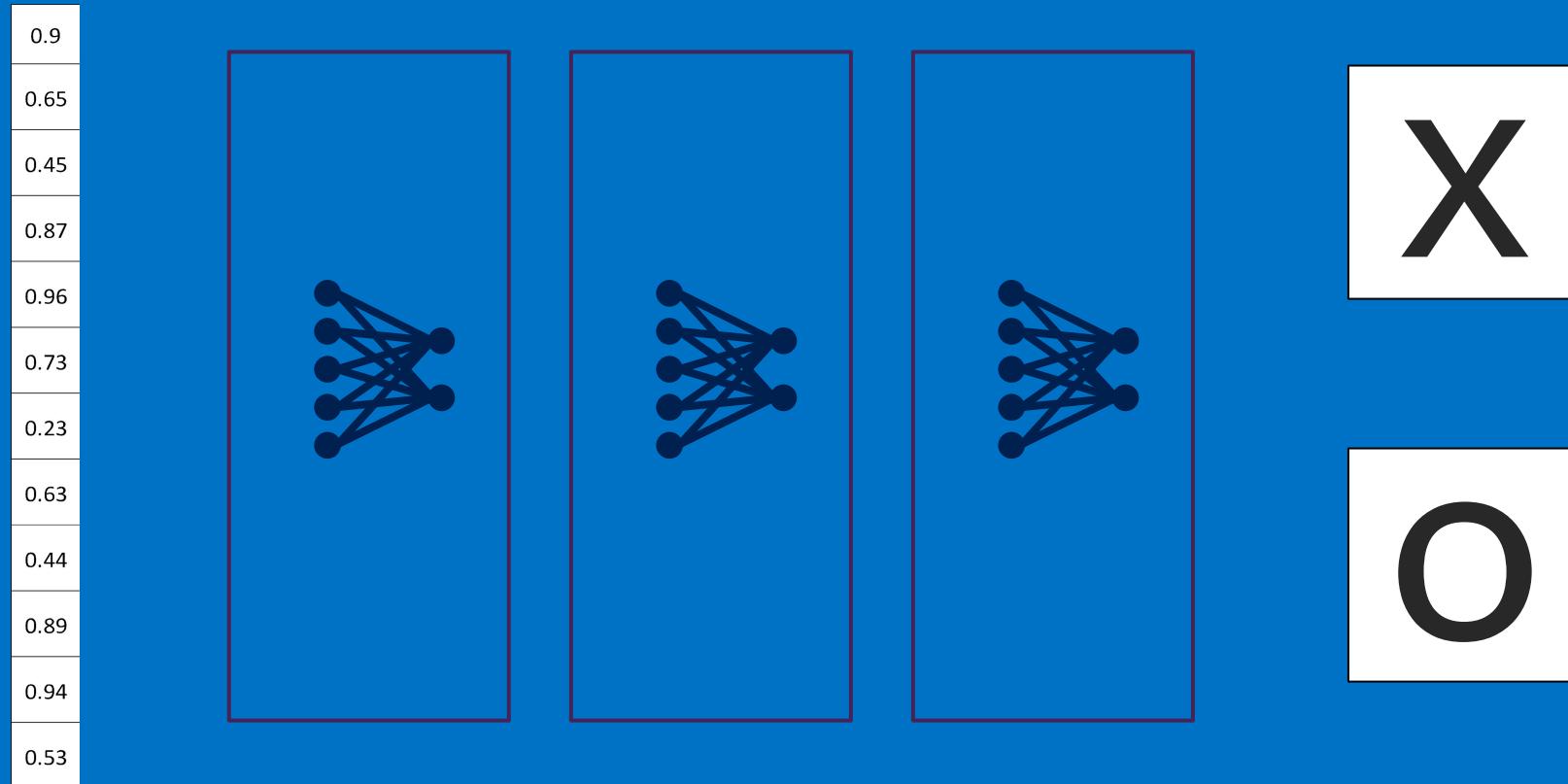


Final result



Fully connected layer

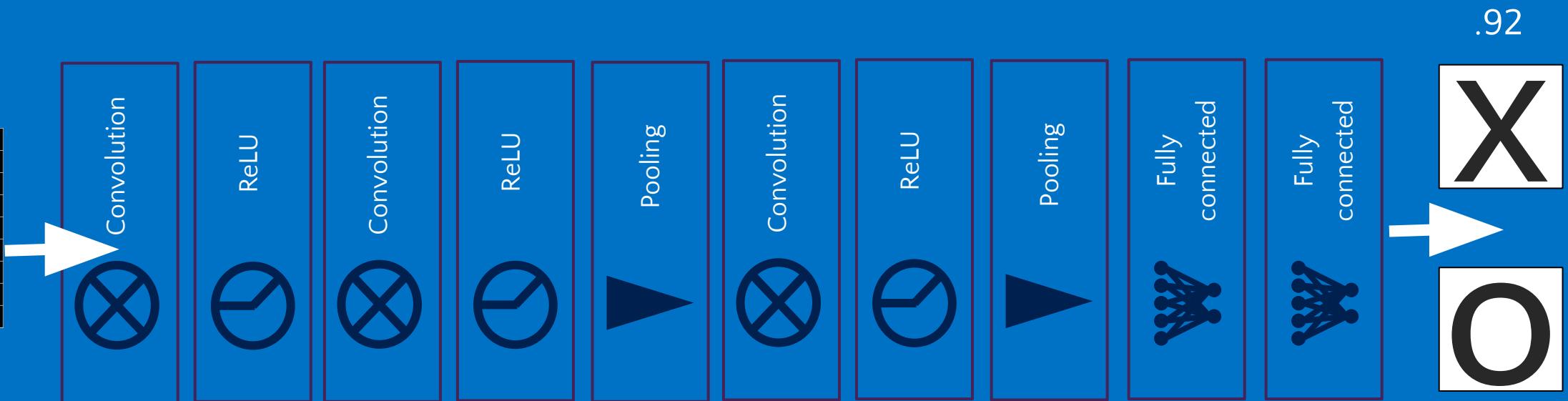
These can also be stacked.



Putting it all together

A set of pixels becomes a set of votes.

-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1	-1	-1	-1
-1	-1	1	-1	-1	-1	1	-1	-1	-1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	1	-1	1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	1	-1	-1	-1
-1	1	-1	-1	-1	-1	-1	-1	-1	1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	1	-1



.92

.51

Hyperparameters (knobs)

Convolution

- Number of features

- Size of features

Pooling

- Window size

- Window stride

Fully Connected

- Number of neurons

Architecture

How many of each type of layer?

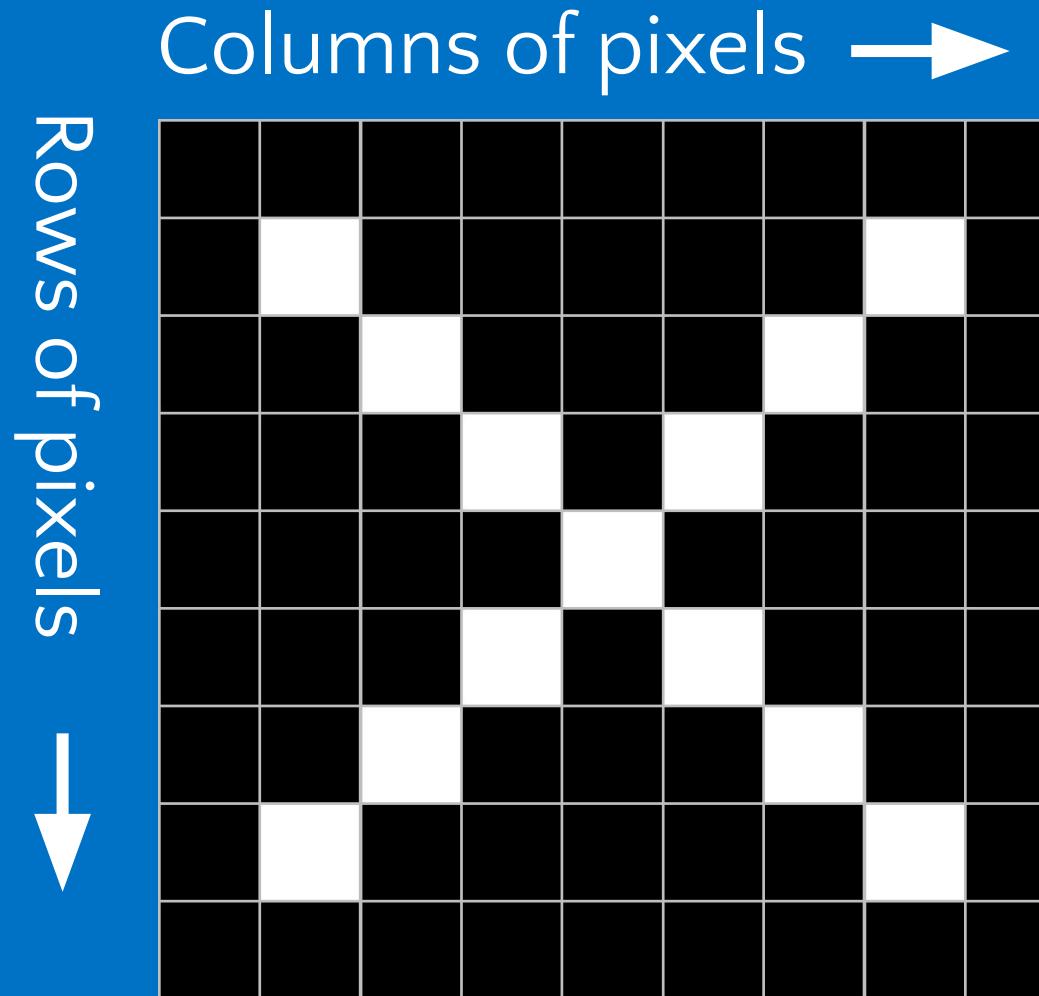
In what order?

Not just images

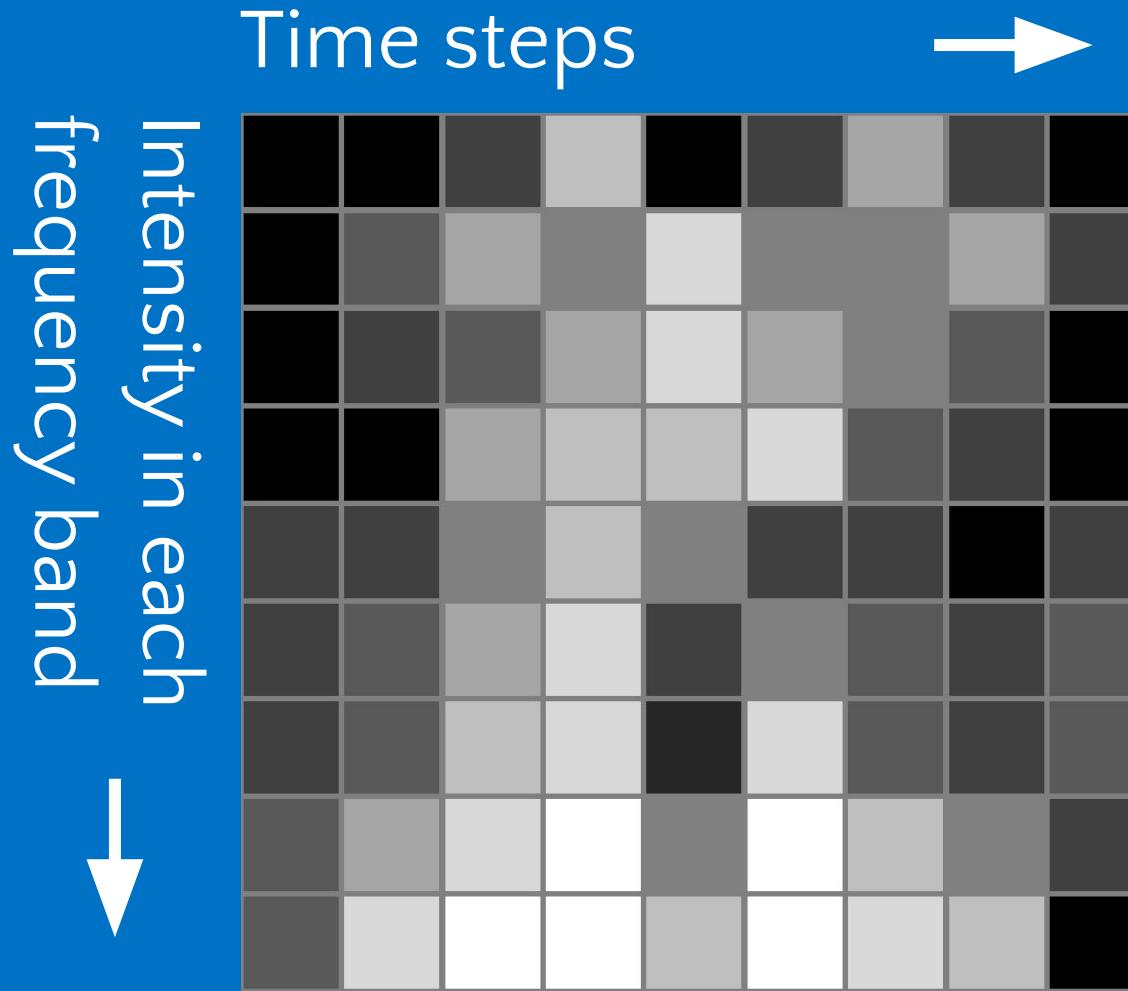
Any 2D (or 3D) data.

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

Images



Sound

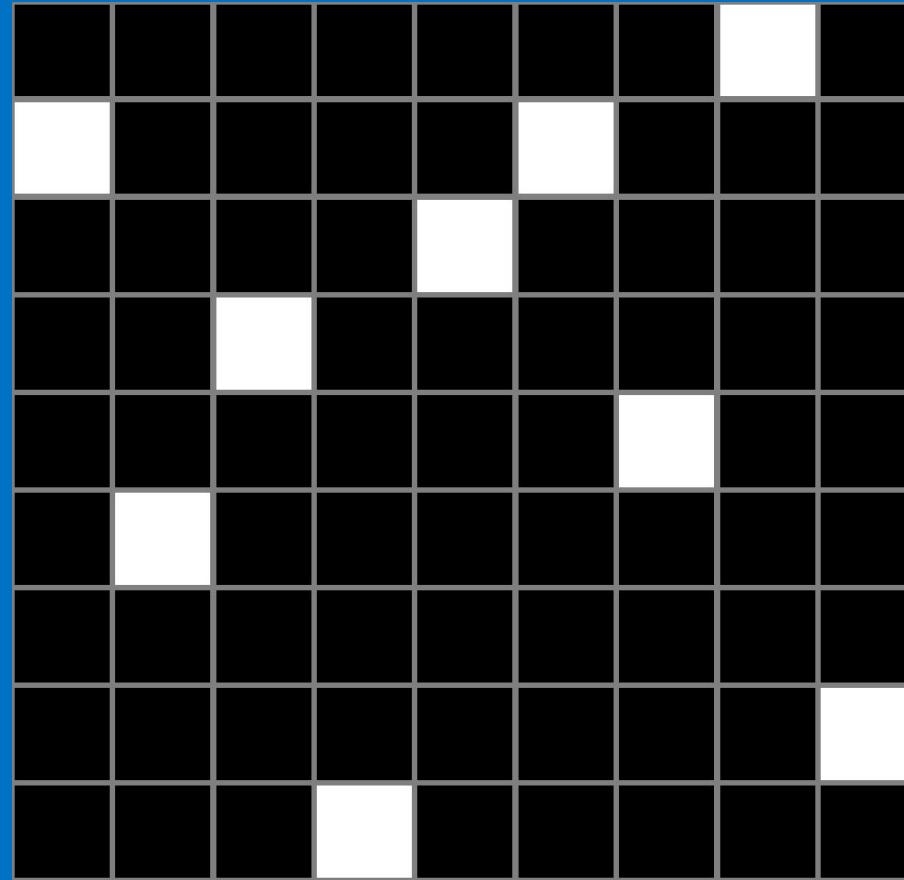


Text

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

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



Customers



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.

Some ConvNet/DNN toolkits

Caffe (Berkeley Vision and Learning Center)

CNTK (Microsoft)

Deeplearning4j (Skymind)

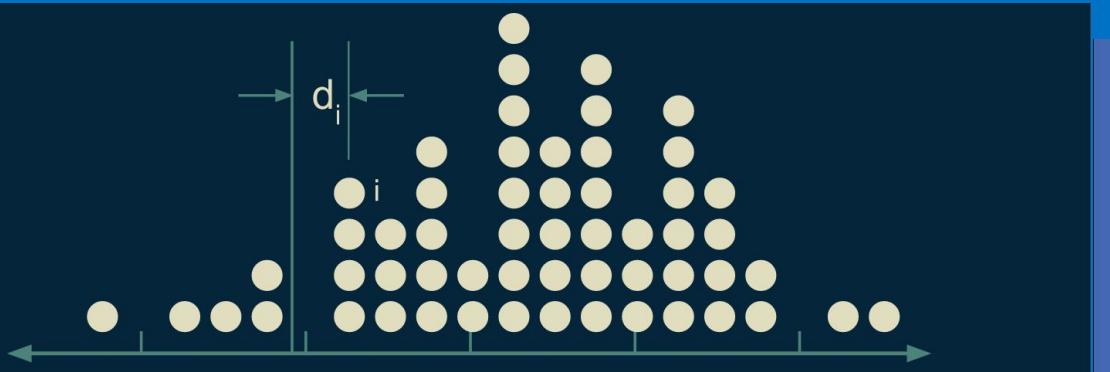
TensorFlow (Google)

Theano (University of Montreal + broad community)

Torch (Ronan Collobert)

Pytorch (Facebook)

Many others



$$\frac{\partial \text{err}}{\partial \text{weight}} = \frac{\partial a}{\partial \text{weight}} * \frac{\partial b}{\partial a} * \frac{\partial c}{\partial b} * \frac{\partial d}{\partial c} * \dots * \frac{\partial y}{\partial x} * \frac{\partial z}{\partial y} * \frac{\partial \text{err}}{\partial z}$$

Diagram illustrating backpropagation through a chain of neurons (a, b, c, ..., x, y, z) to calculate the error gradient with respect to a weight.

