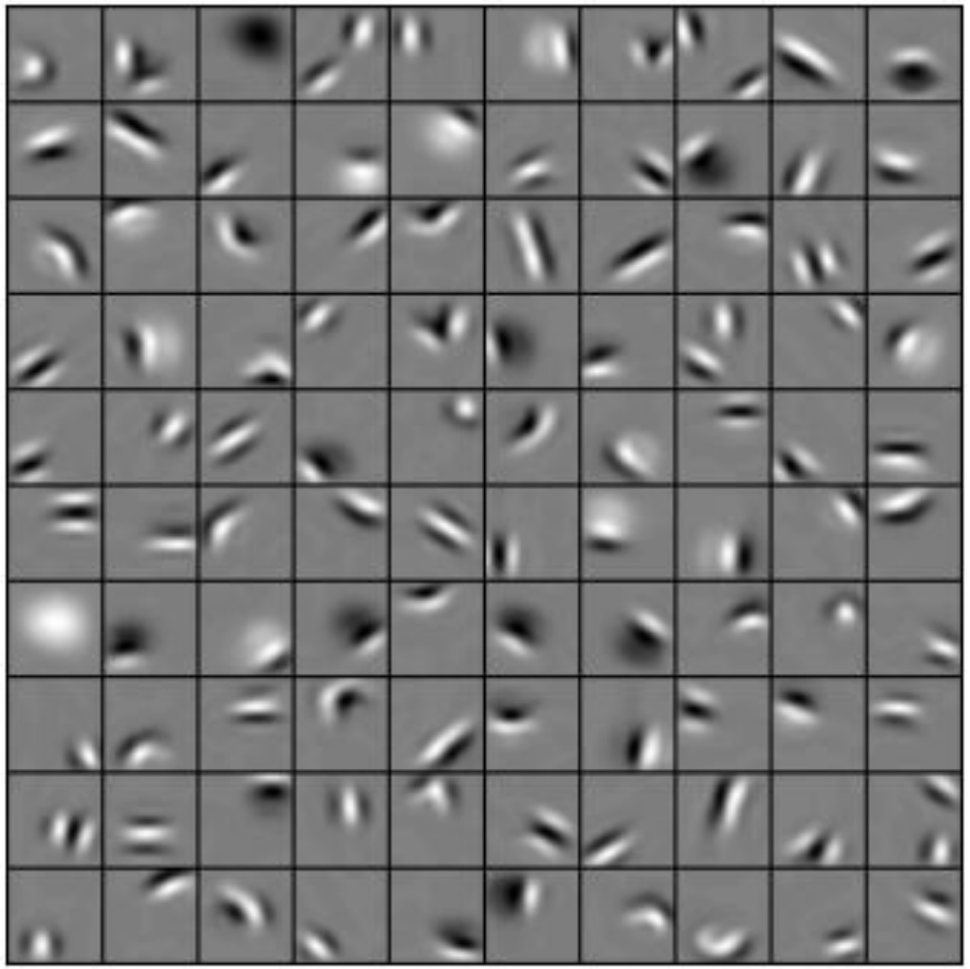
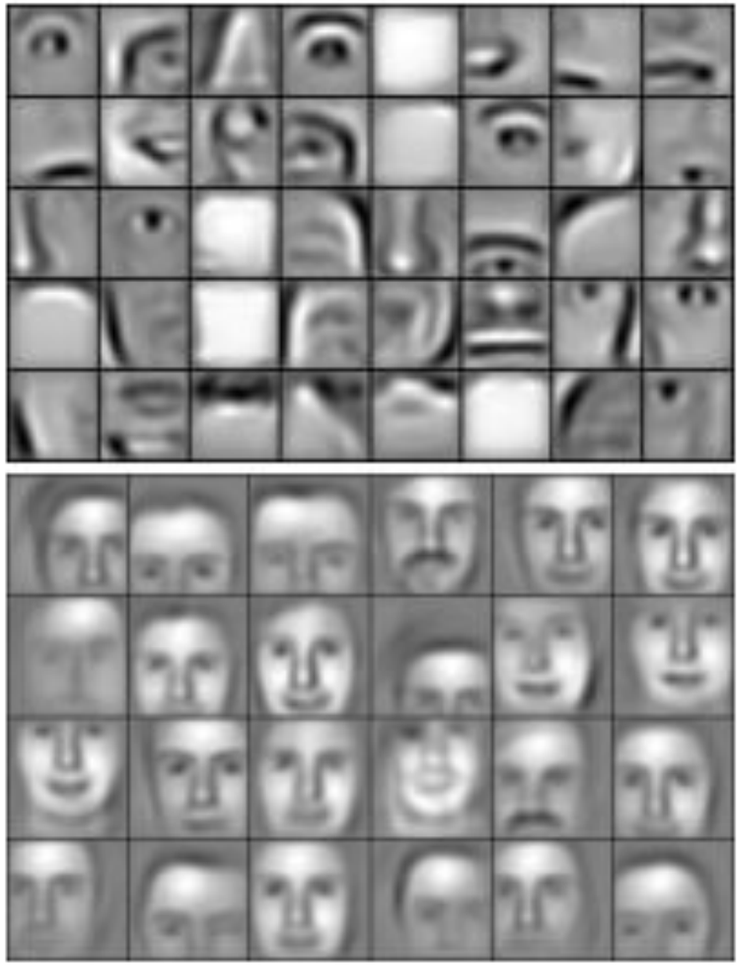
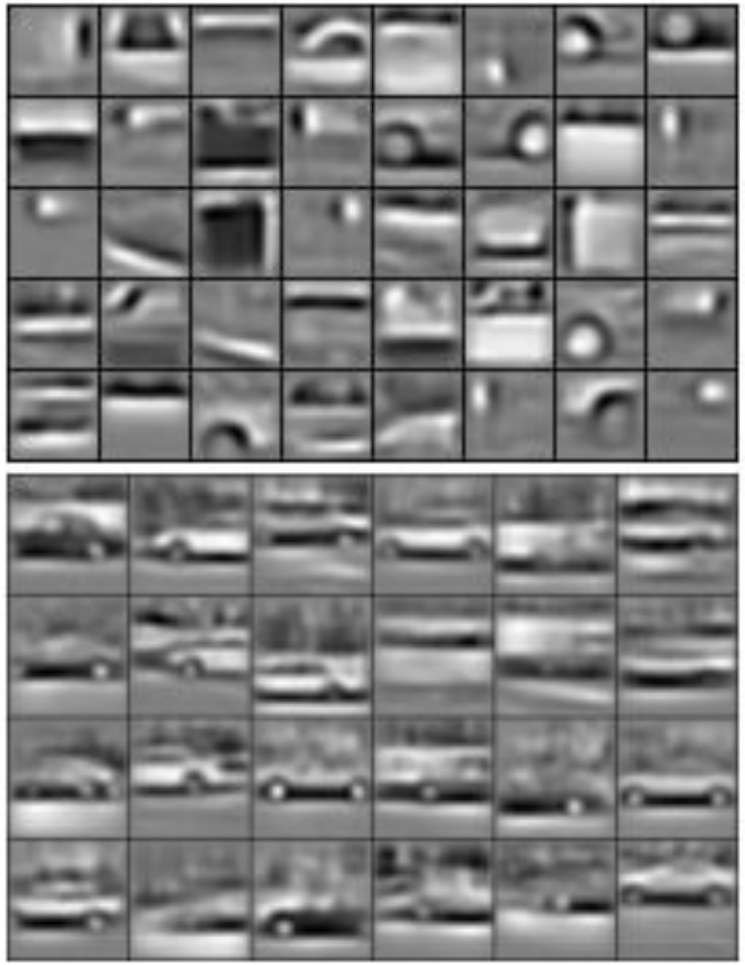
How Convolutional Neural Networks Work



faces

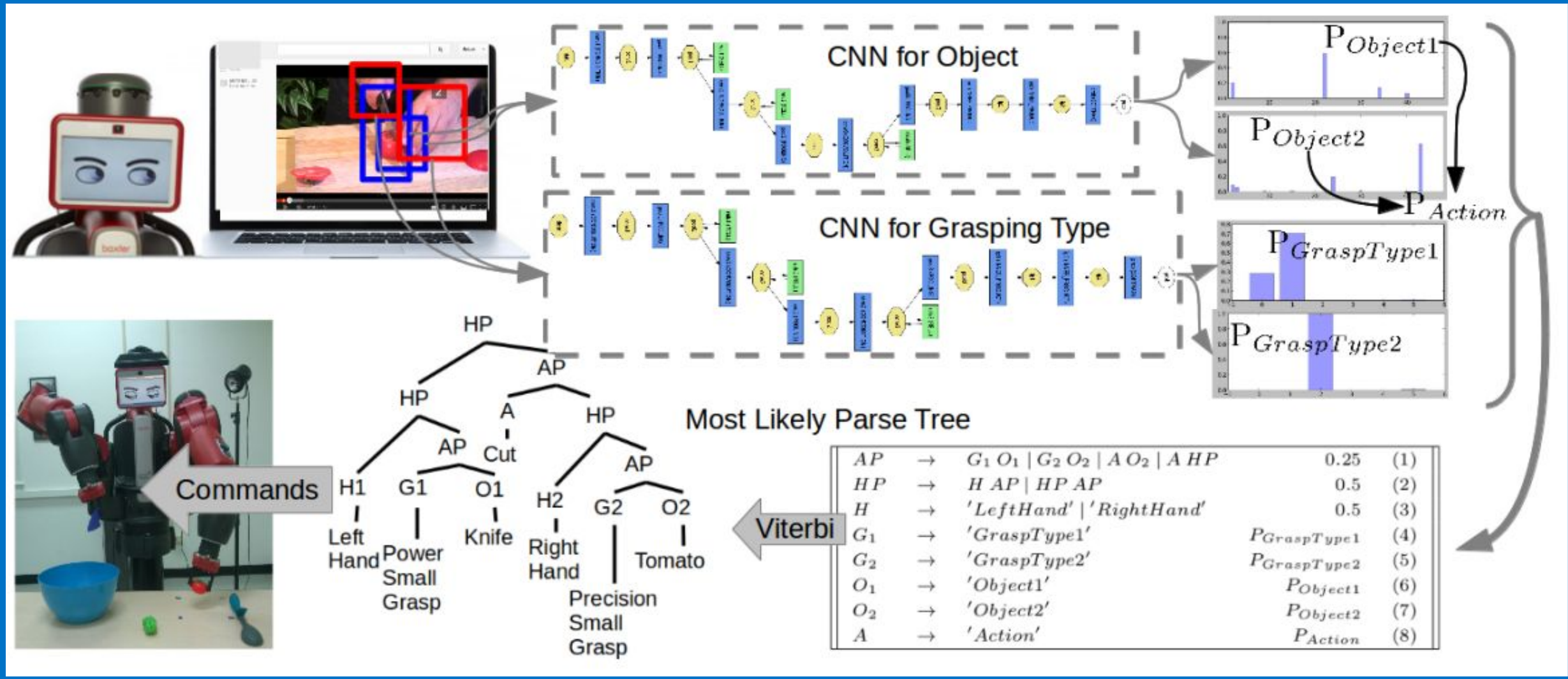


cars



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



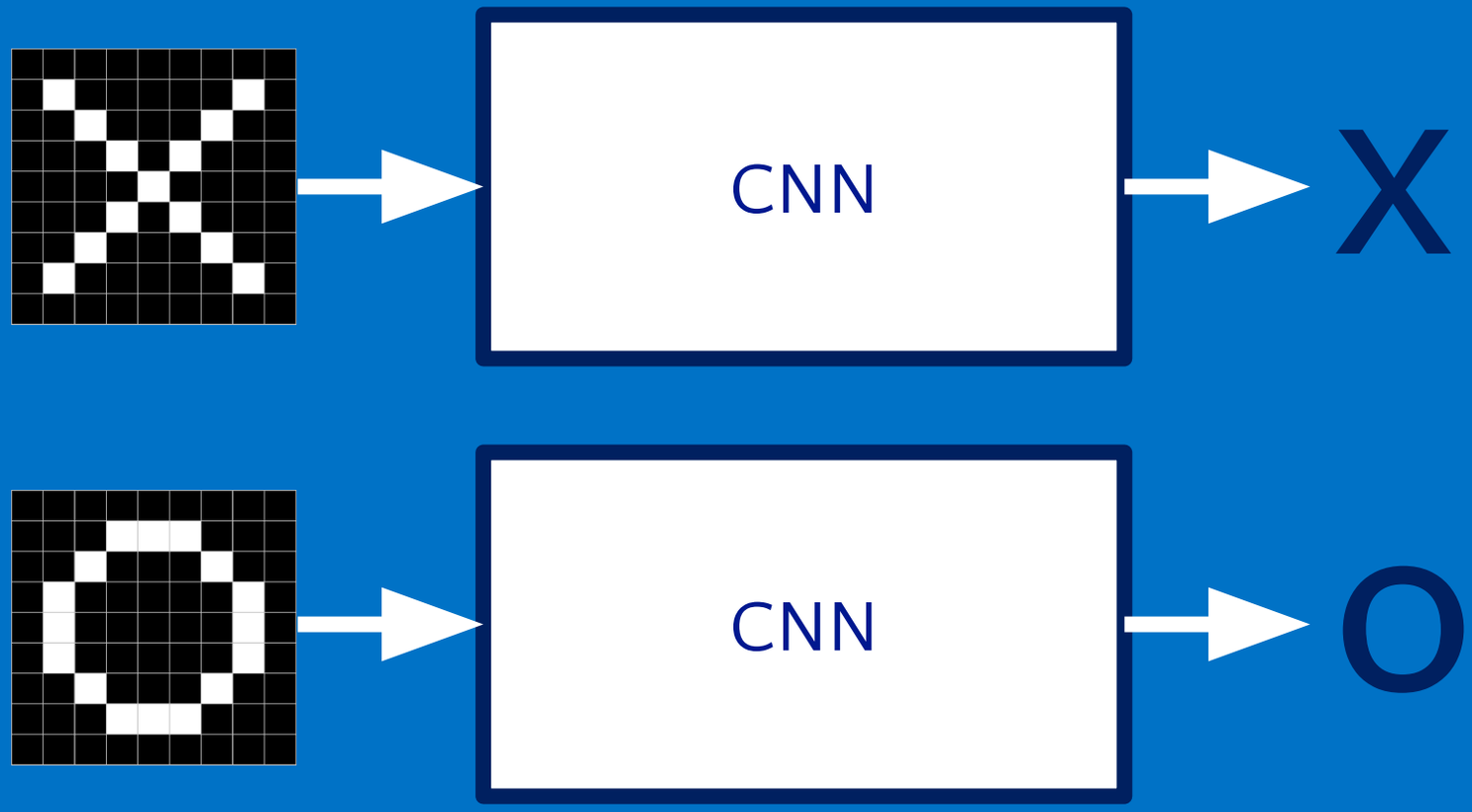
Robot Learning ManipulationAction Plans by “Watching” Unconstrained Videos from the World Wide Web.
Yezhou Yang, Cornelia Fermuller, Yiannis Aloimonos

A toy ConvNet: X's and O's

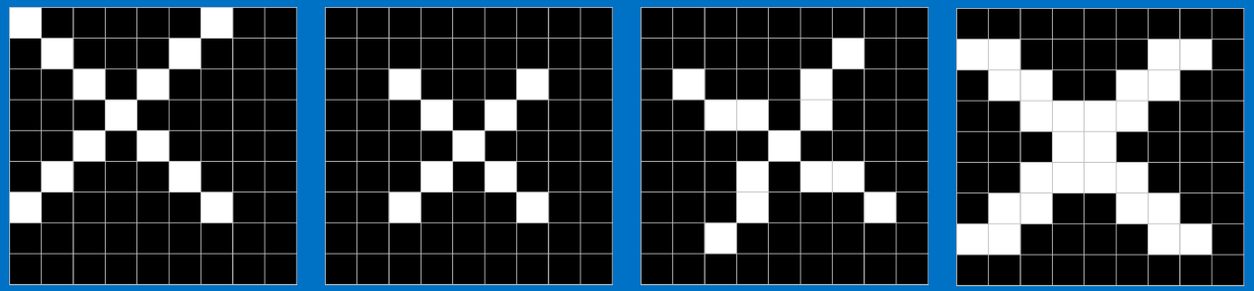
Says whether a picture is of an X or an O

A two-dimensional array of pixels

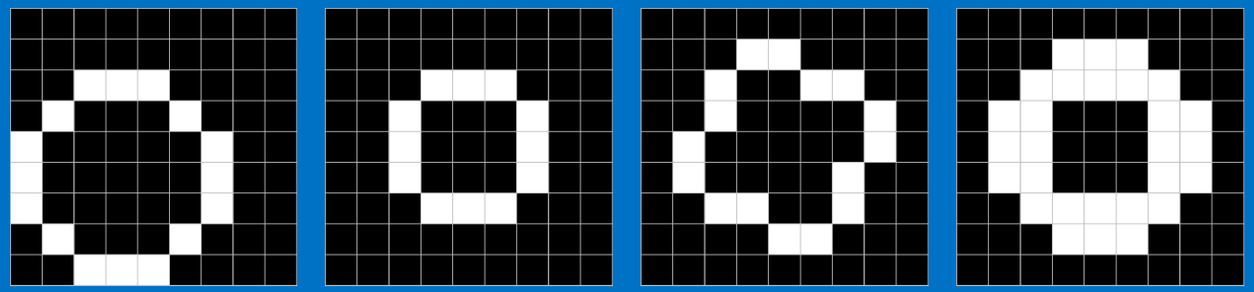
For example



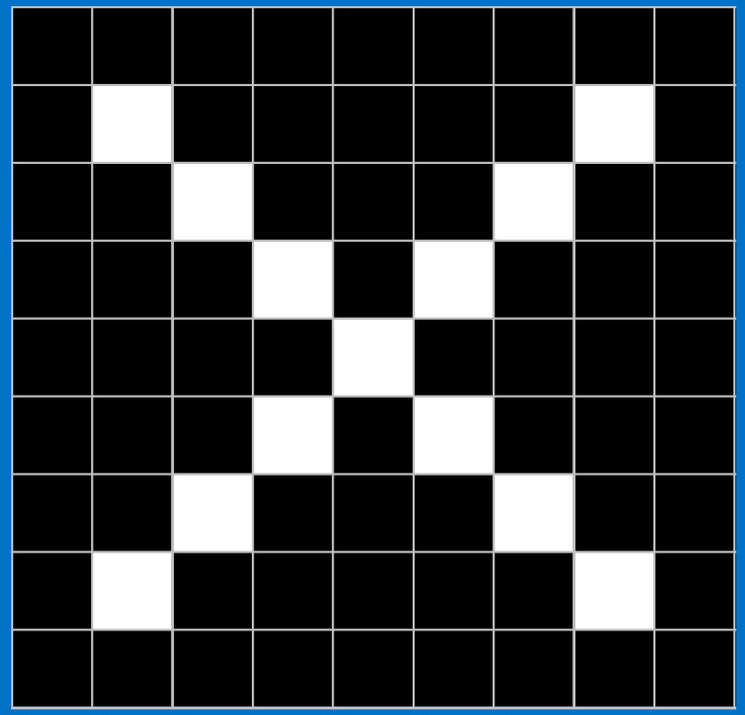
Trickier cases



translation scaling rotation weight

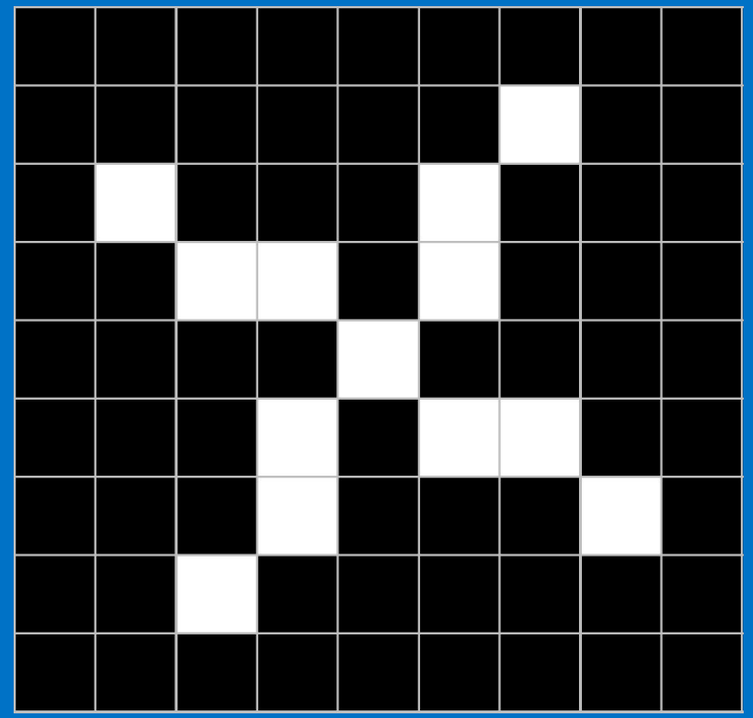


Deciding is hard



?

=

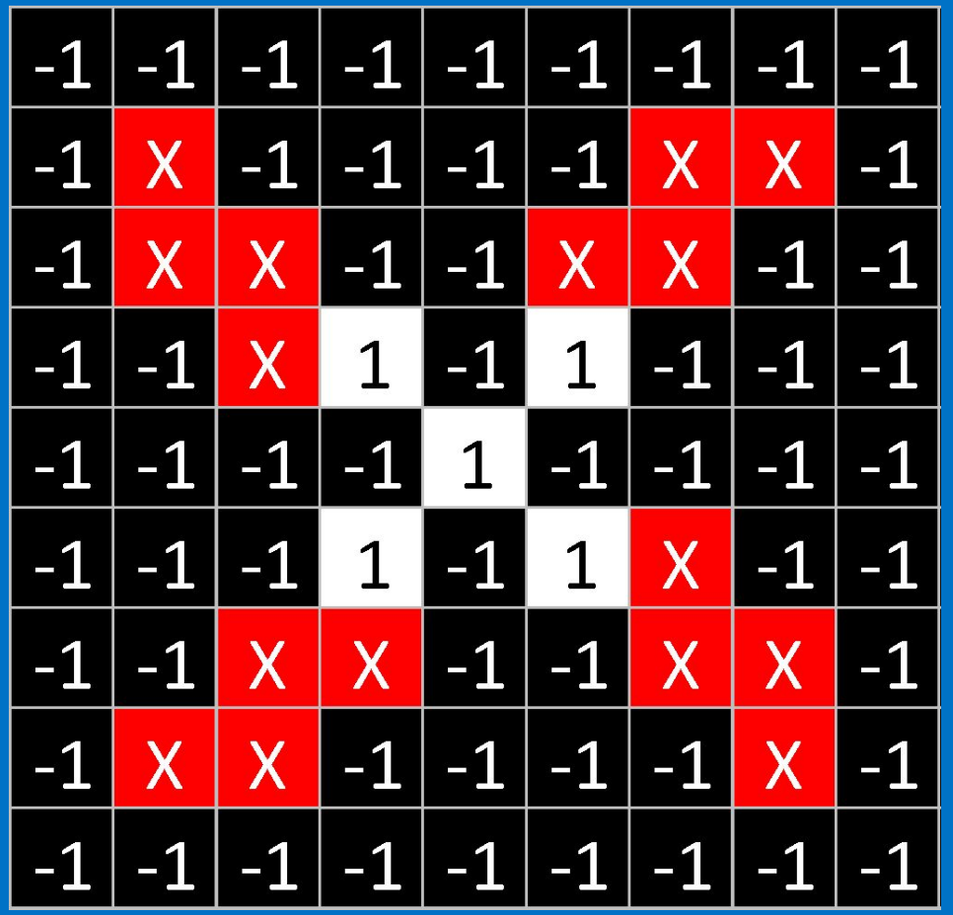


What computers see

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

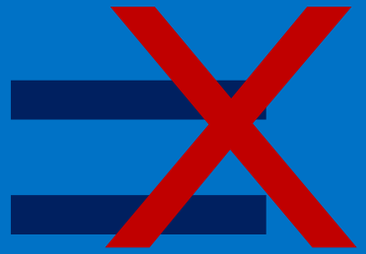
-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	1	-1	-1
-1	1	-1	-1	-1	1	-1	-1	-1
-1	-1	1	1	-1	1	-1	-1	-1
-1	-1	-1	-1	1	-1	-1	-1	-1
-1	-1	-1	1	-1	1	1	-1	-1
-1	-1	-1	1	-1	-1	-1	1	-1
-1	-1	1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1

What computers see



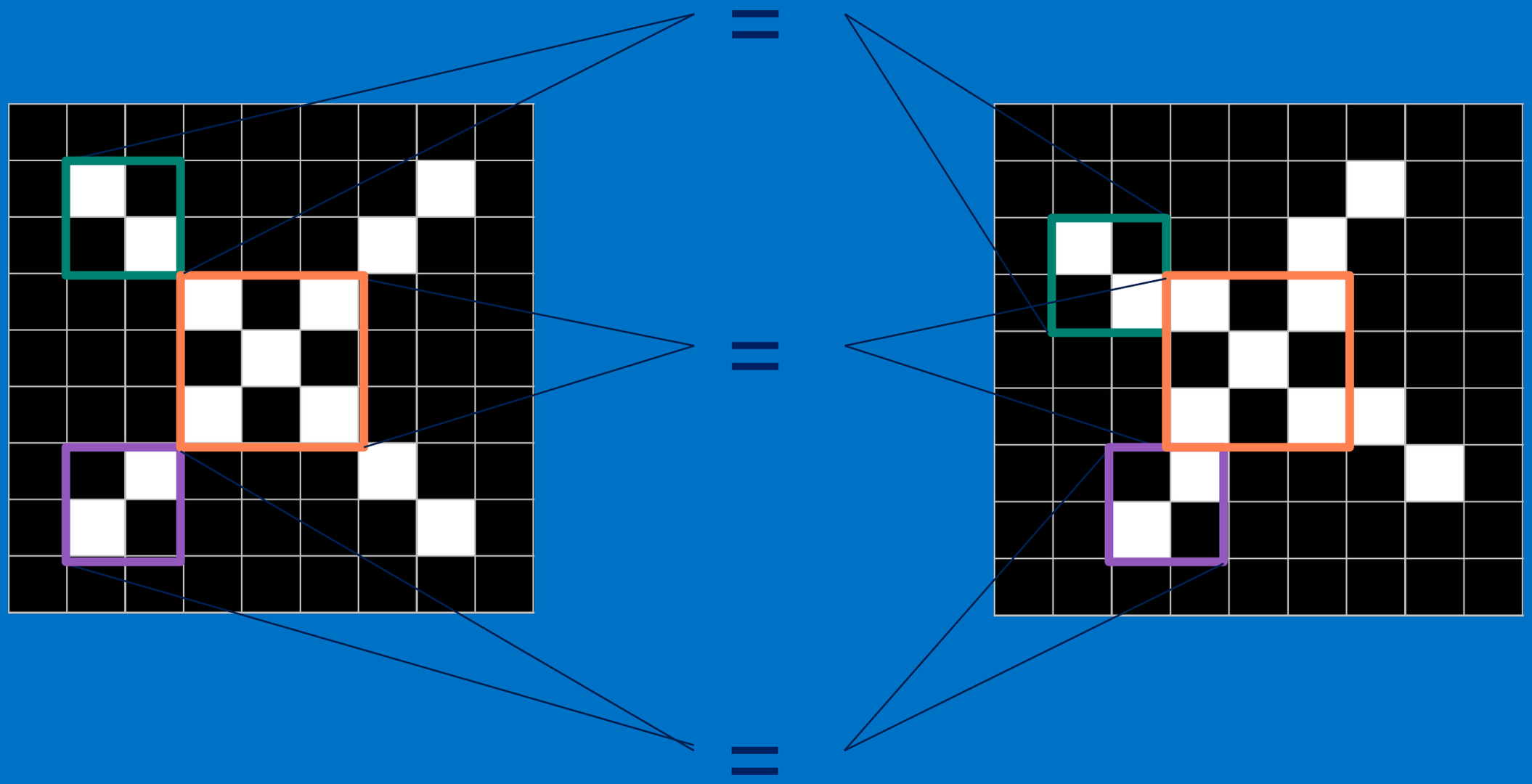
Computers are literal

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

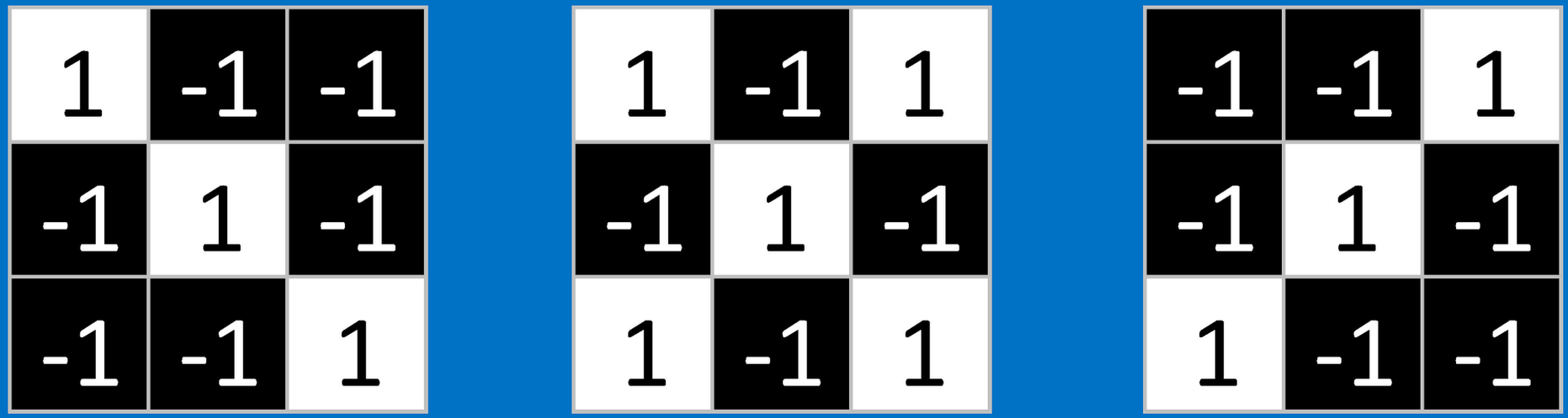


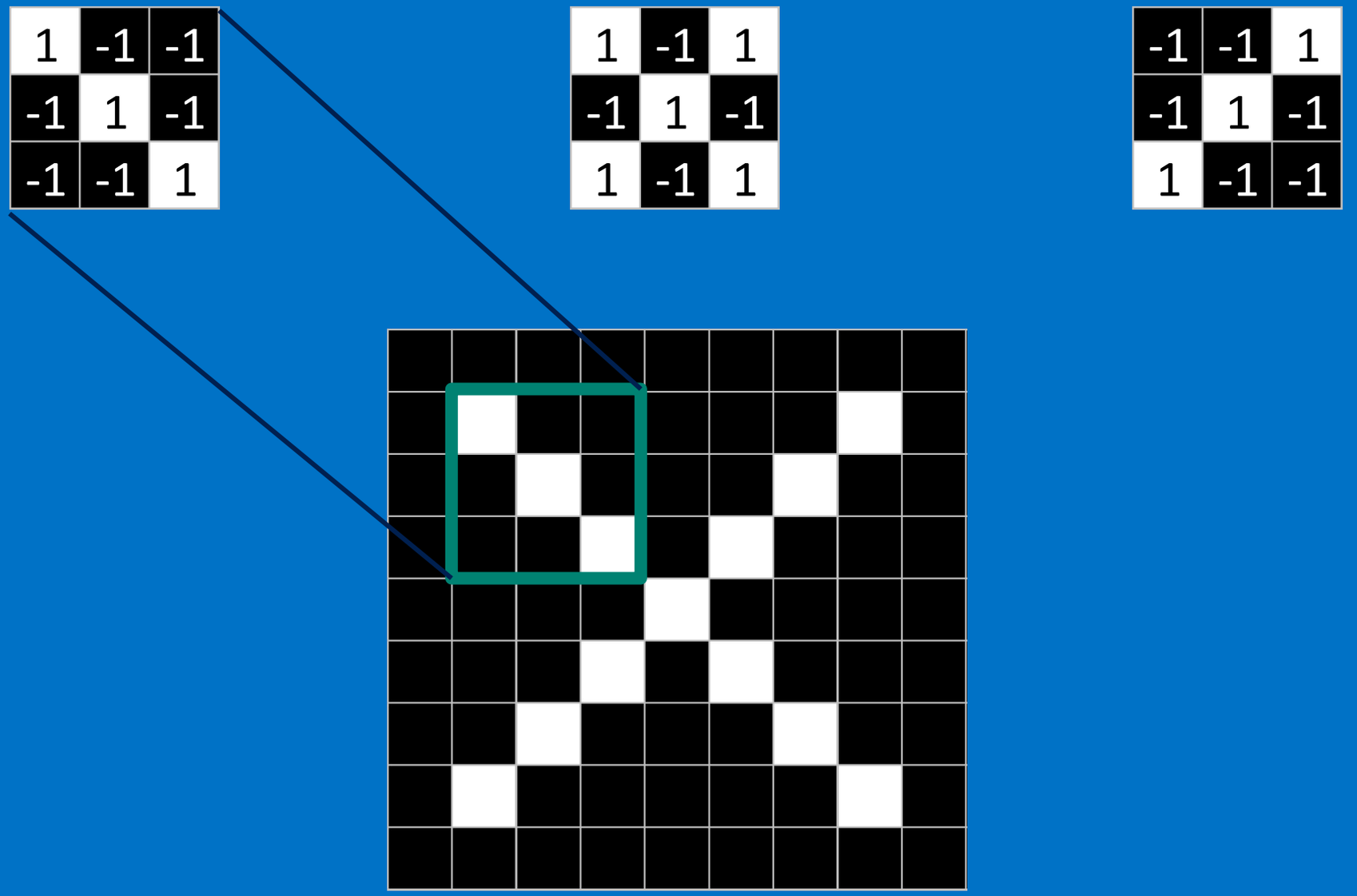
-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	1	-1	-1
-1	1	-1	-1	-1	1	-1	-1	-1
-1	-1	1	1	-1	1	-1	-1	-1
-1	-1	-1	-1	1	-1	-1	-1	-1
-1	-1	-1	1	-1	1	1	-1	-1
-1	-1	-1	1	-1	-1	-1	1	-1
-1	-1	1	-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1

ConvNets match pieces of the image



Features match pieces of the image

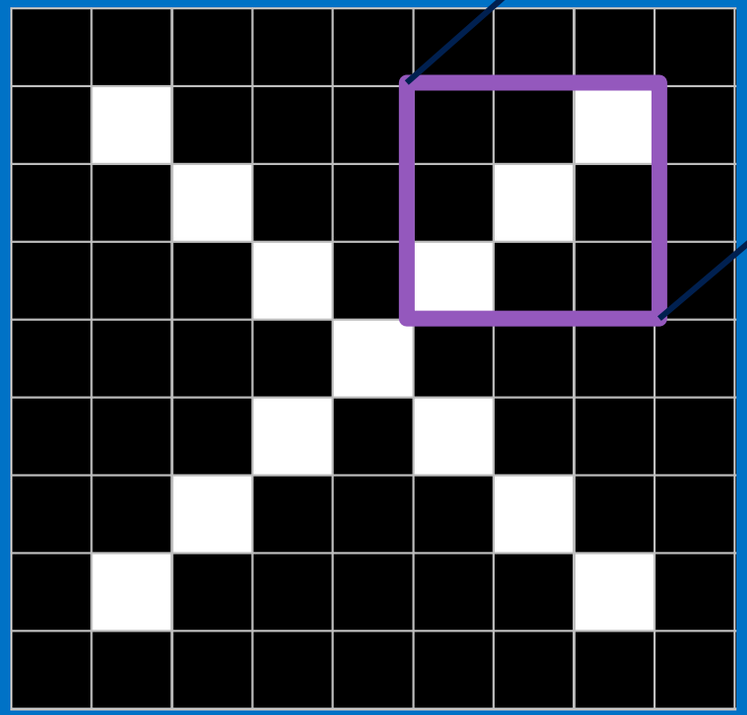



1 -1 -1
-1 1 -1
-1 -1 1
1 -1 1
-1 1 -1
1 -1 1
-1 -1 1
-1 1 -1
1 -1 -1

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

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

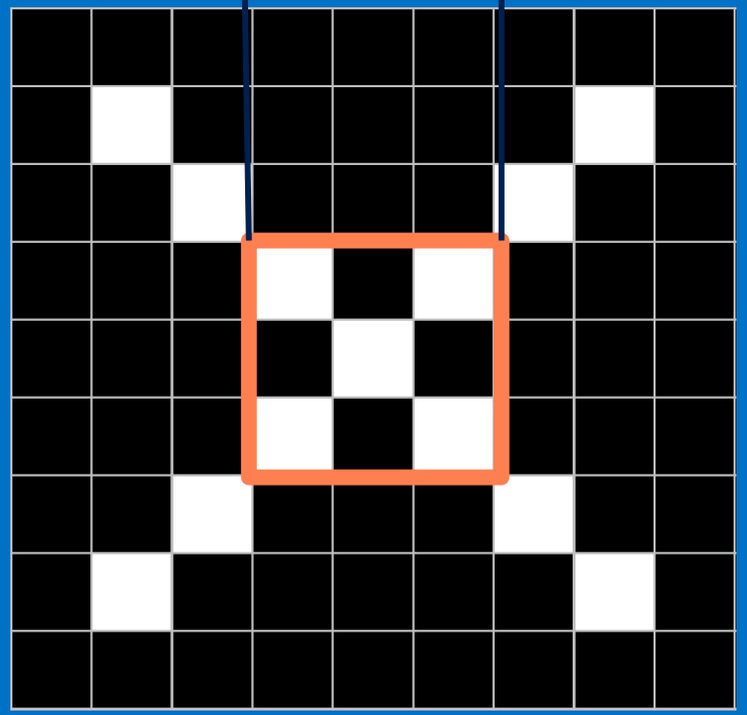
-1	-1	1
-1	1	-1
1	-1	-1

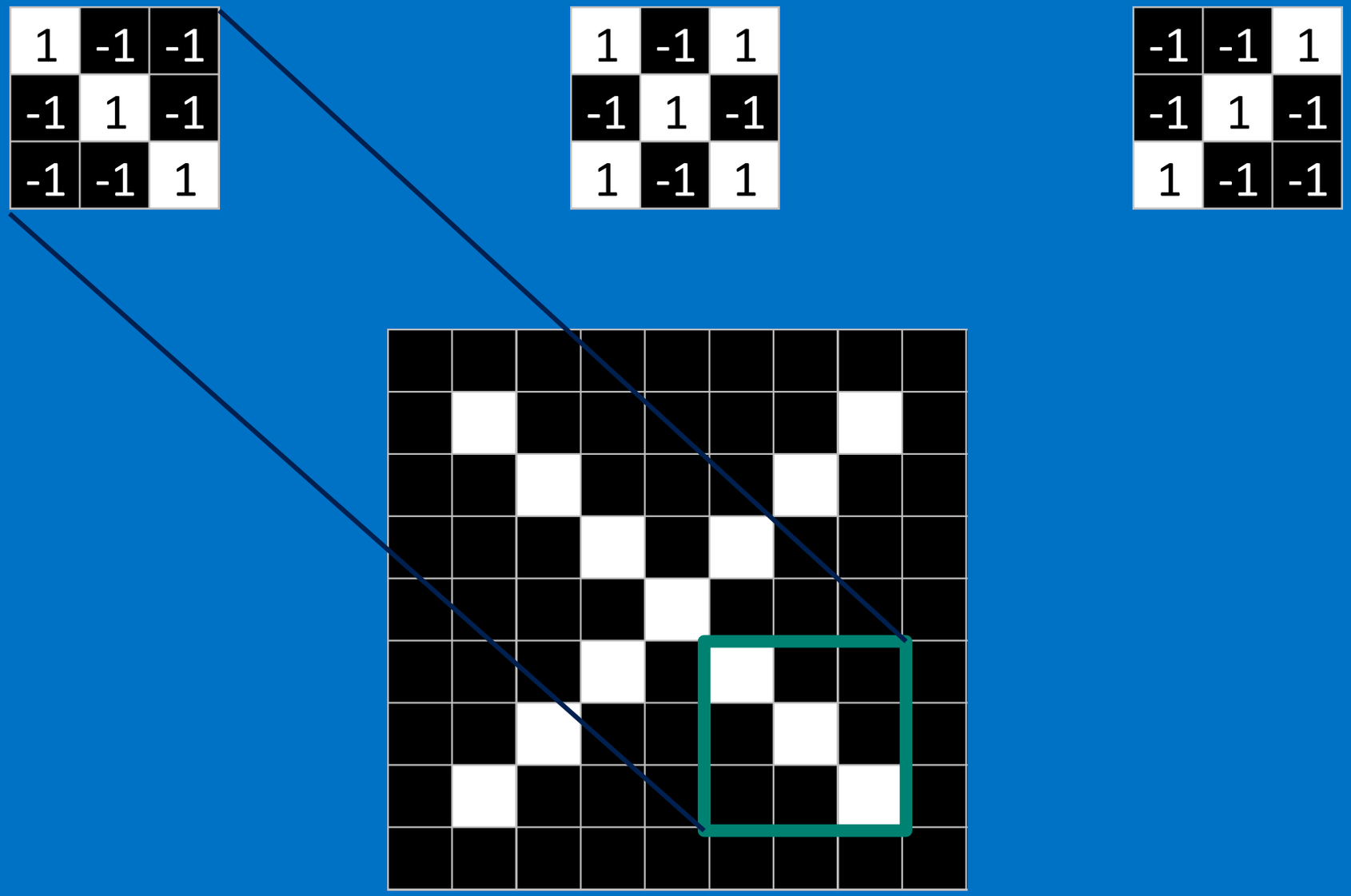


1	-1	-1
-1	1	-1
-1	-1	1

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

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

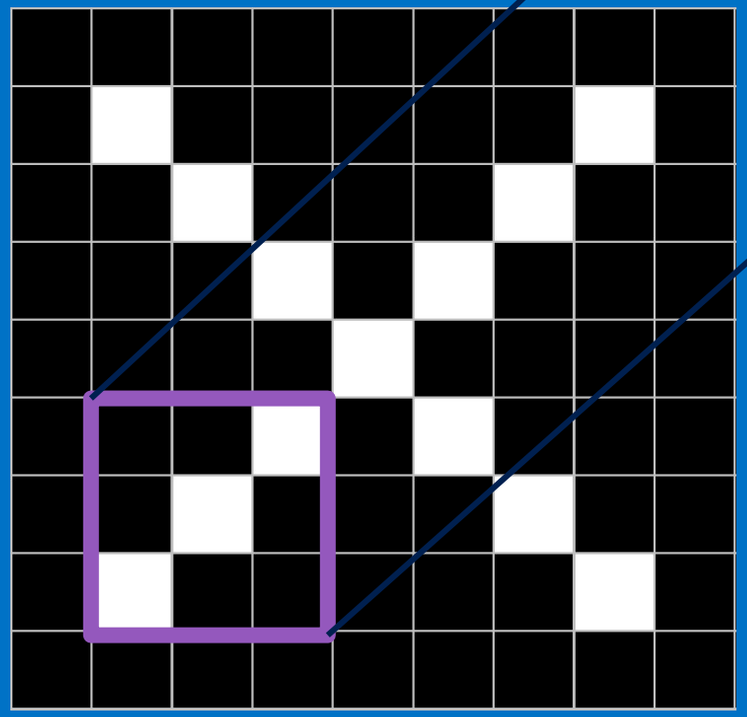



1 -1 -1
-1 1 -1
-1 -1 1
1 -1 1
-1 1 -1
1 -1 1
-1 -1 1
-1 1 -1
1 -1 -1

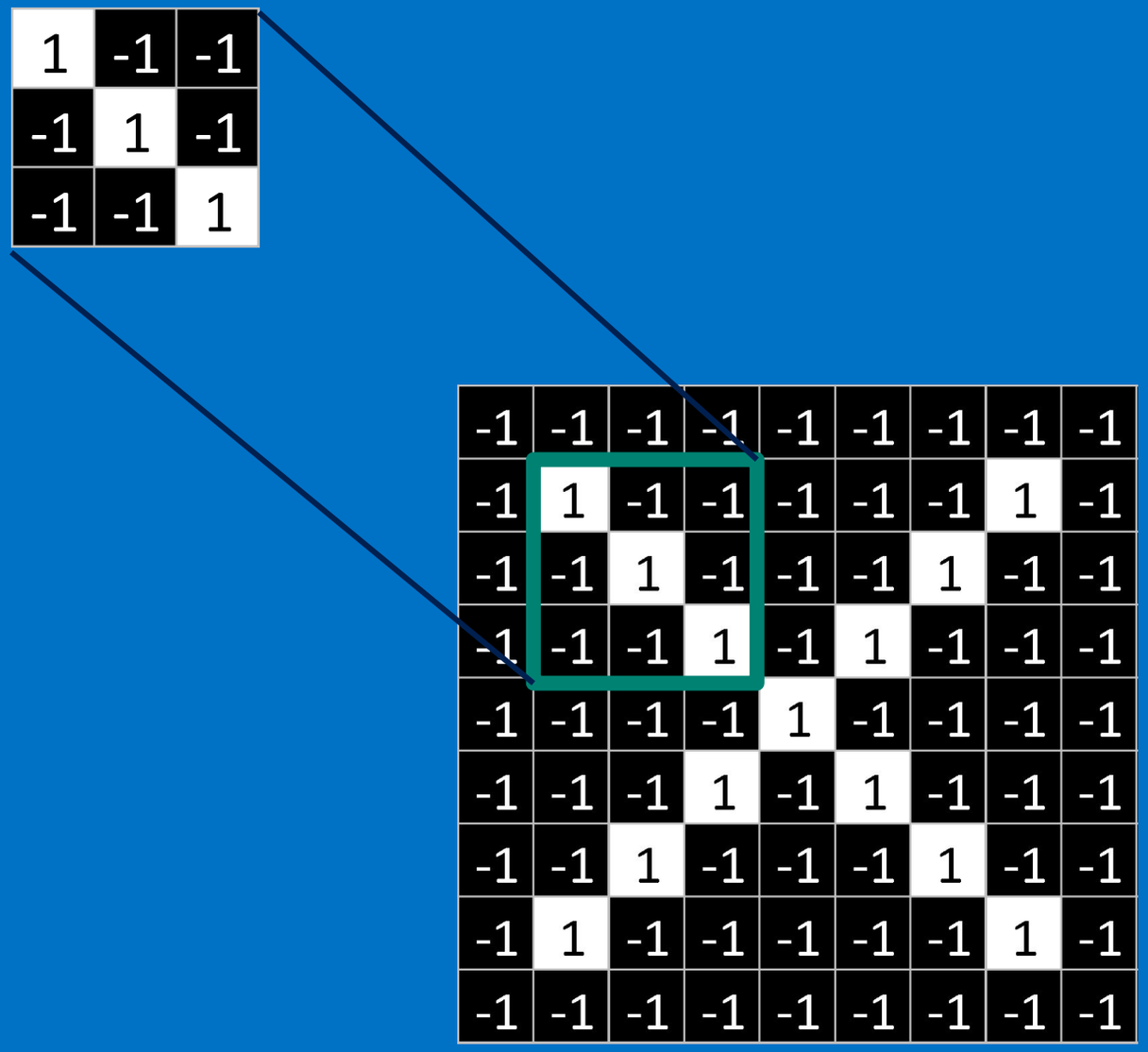
1	-1	-1
-1	1	-1
-1	-1	1

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

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



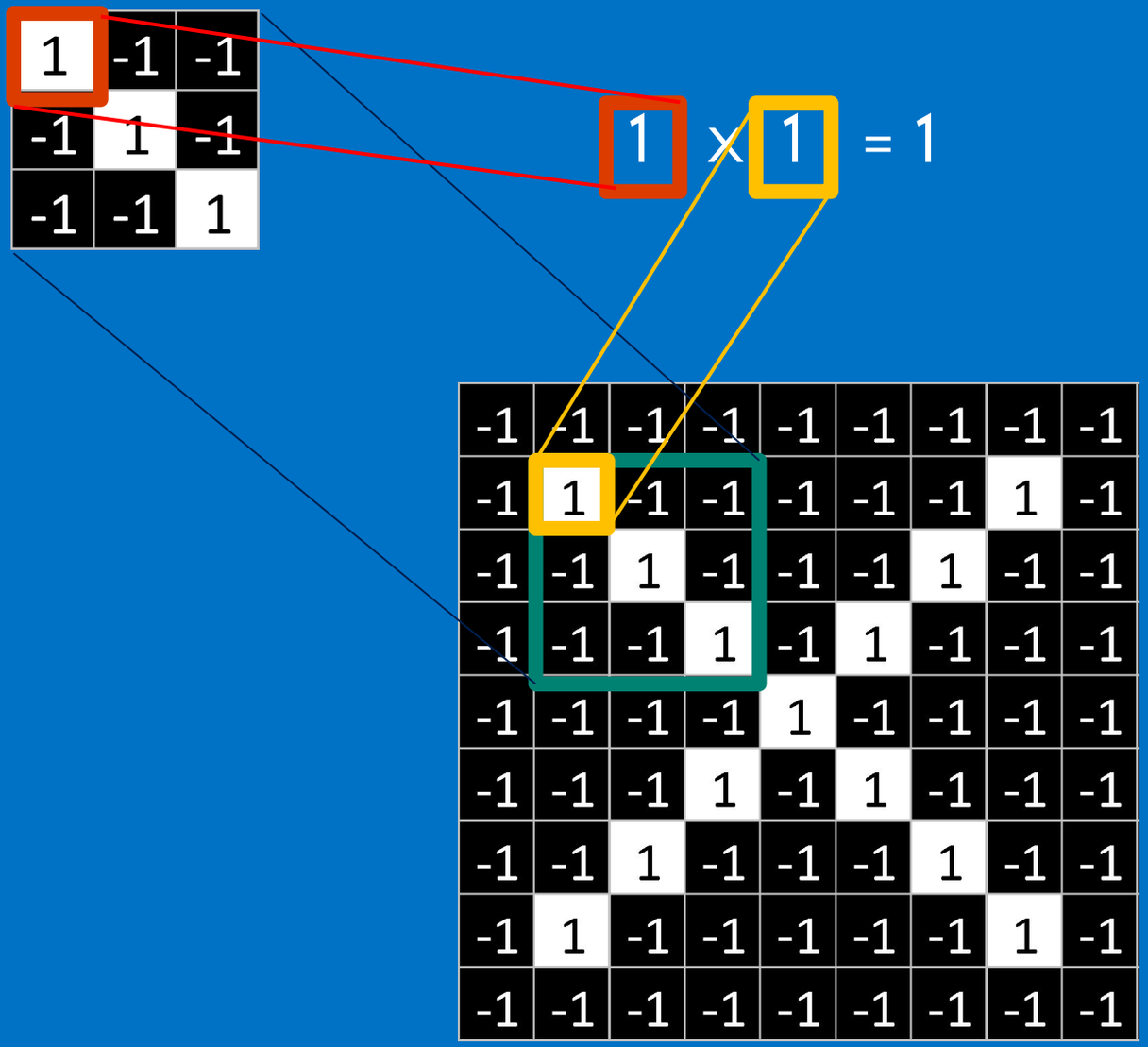
Filtering: The math behind the match



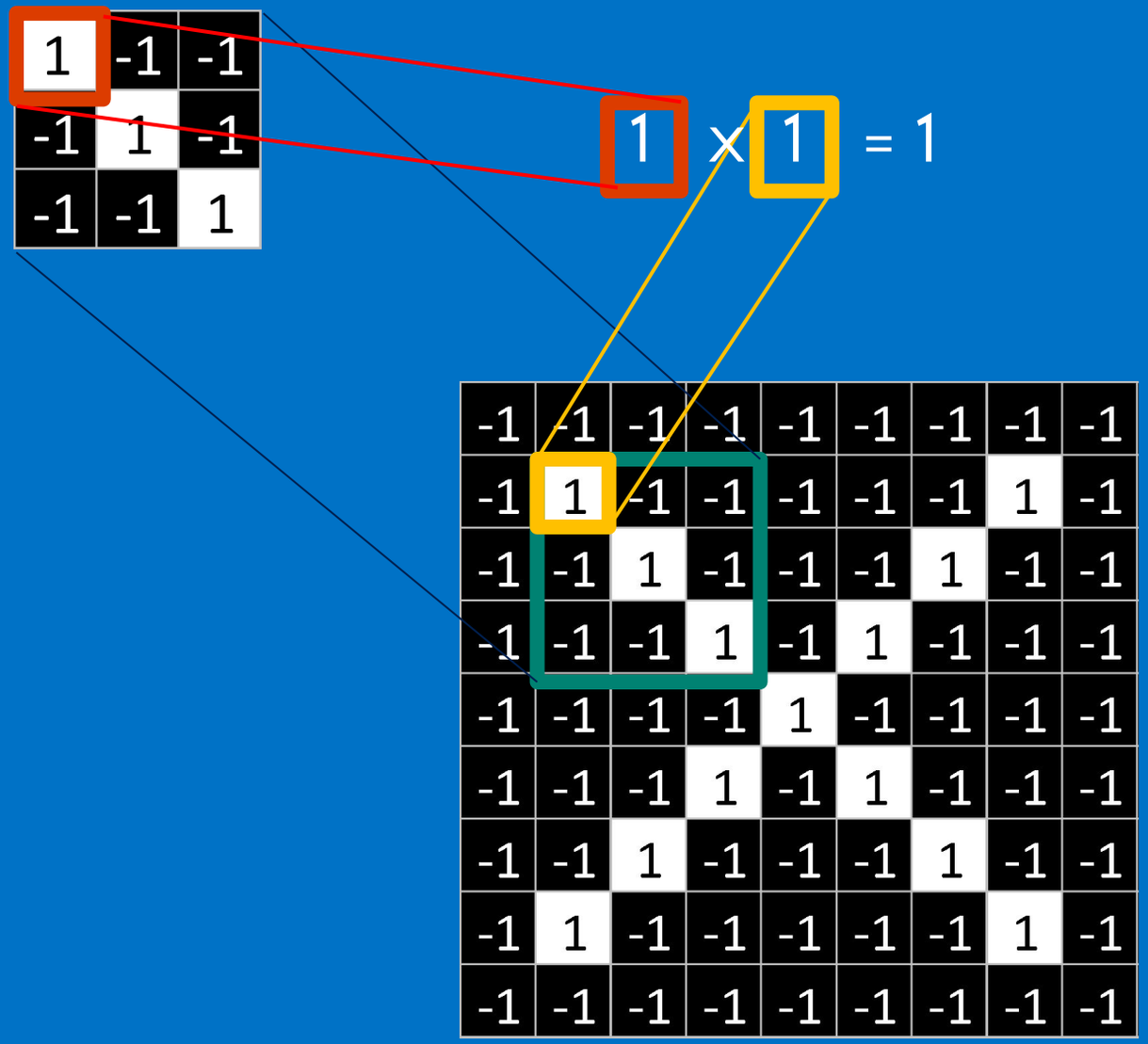
Filtering: The math behind the match

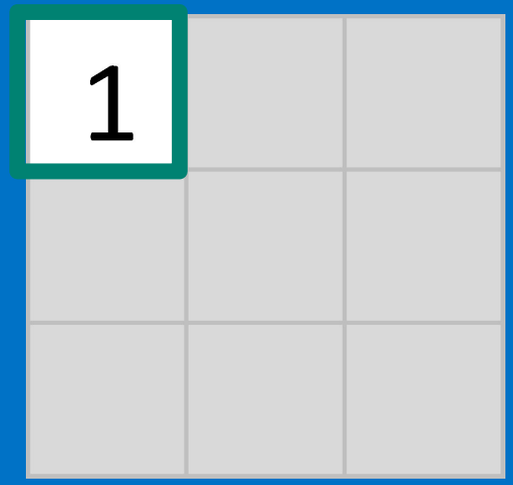
1. Line up the feature and the image patch.
2. Multiply each image pixel by the corresponding feature pixel.
3. Add them up.
4. Divide by the total number of pixels in the feature.

Filtering: The math behind the match

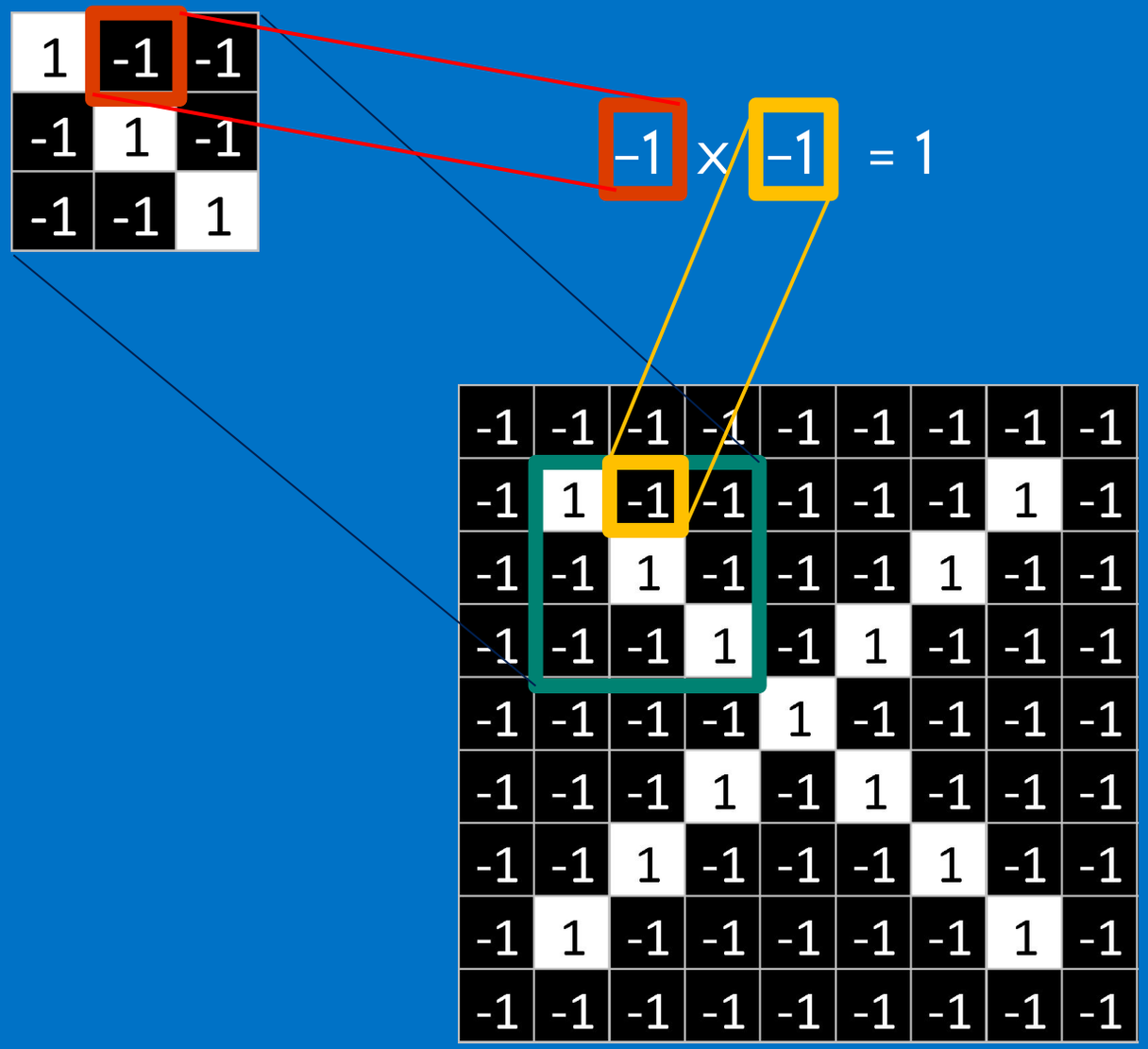


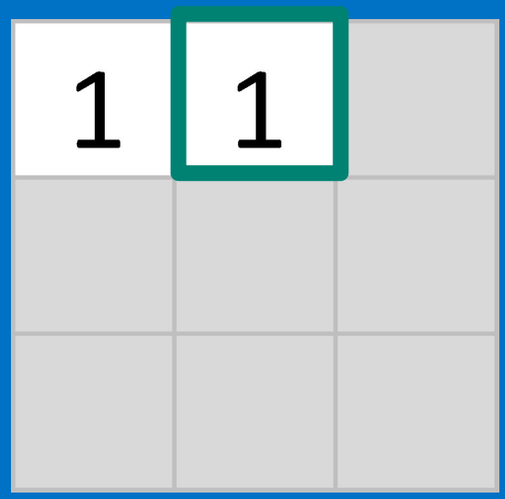
Filtering: The math behind the match



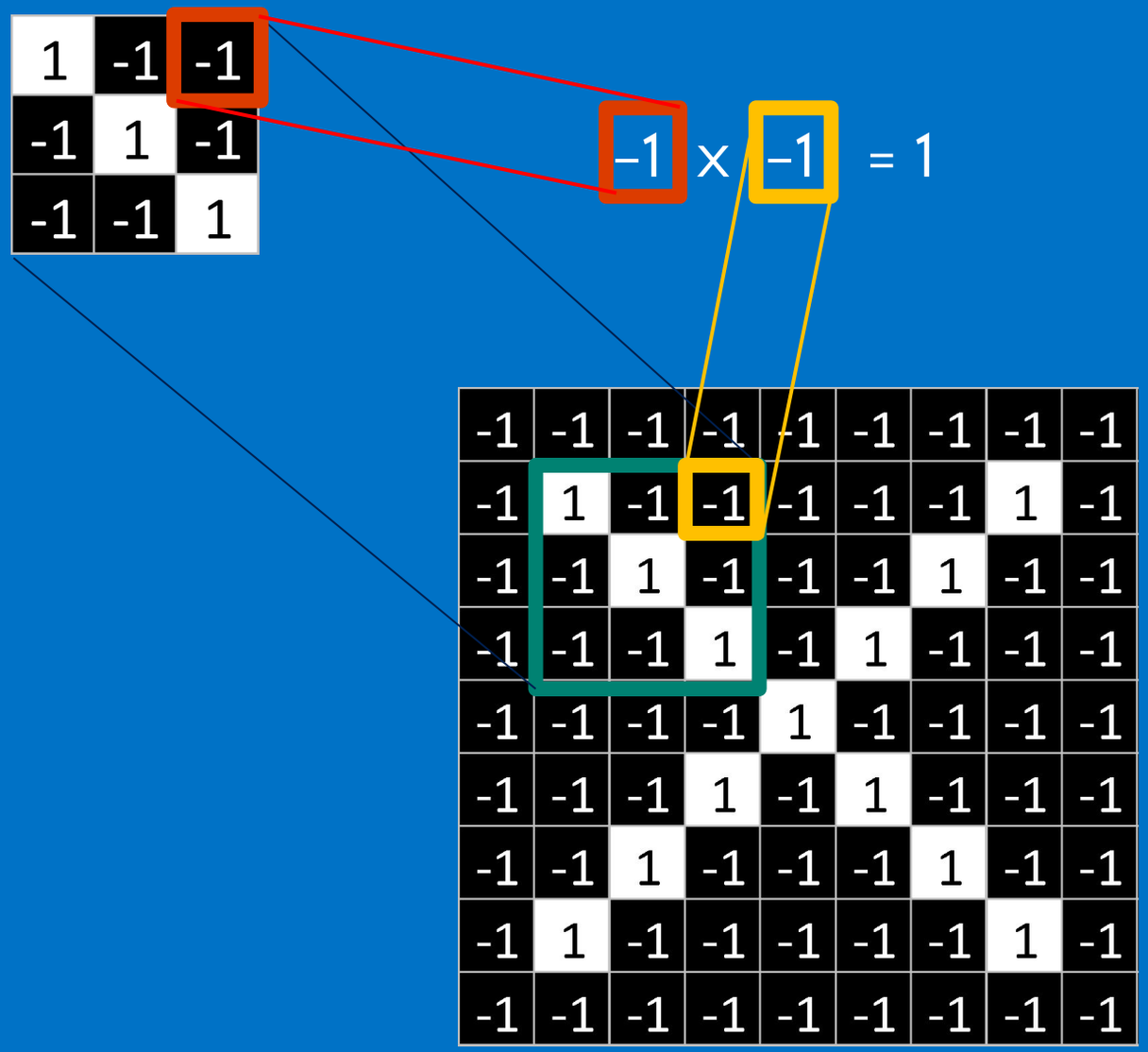


Filtering: The math behind the match

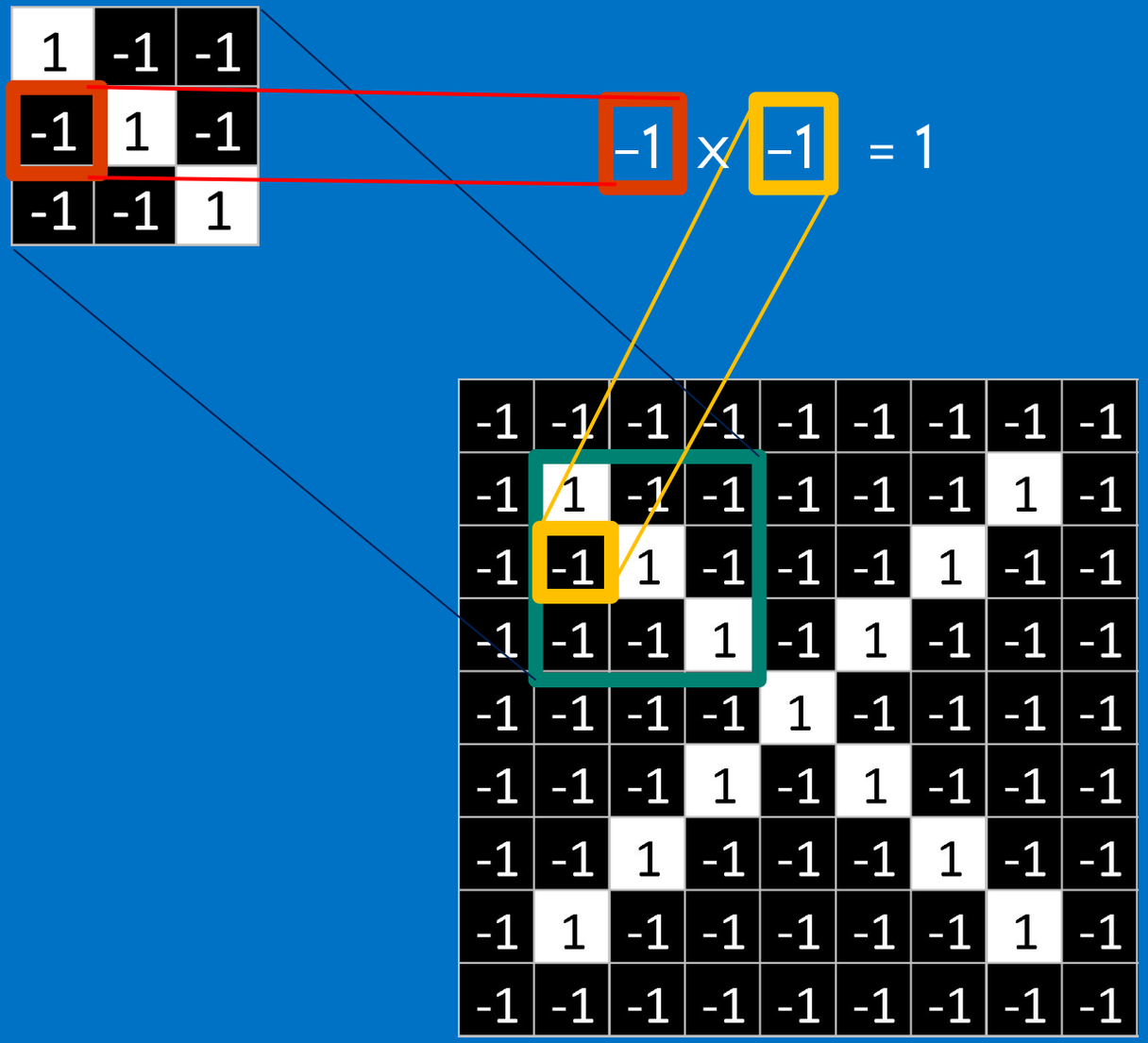


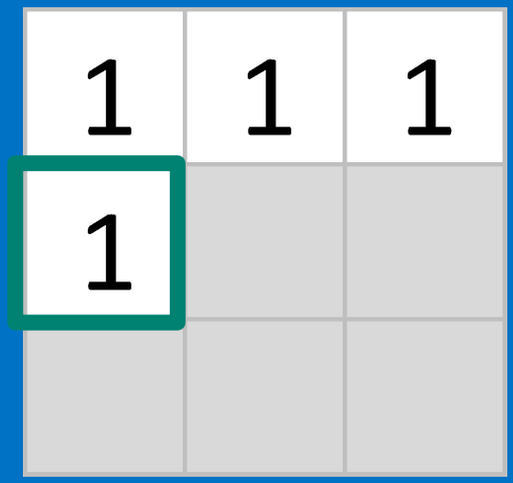


Filtering: The math behind the match

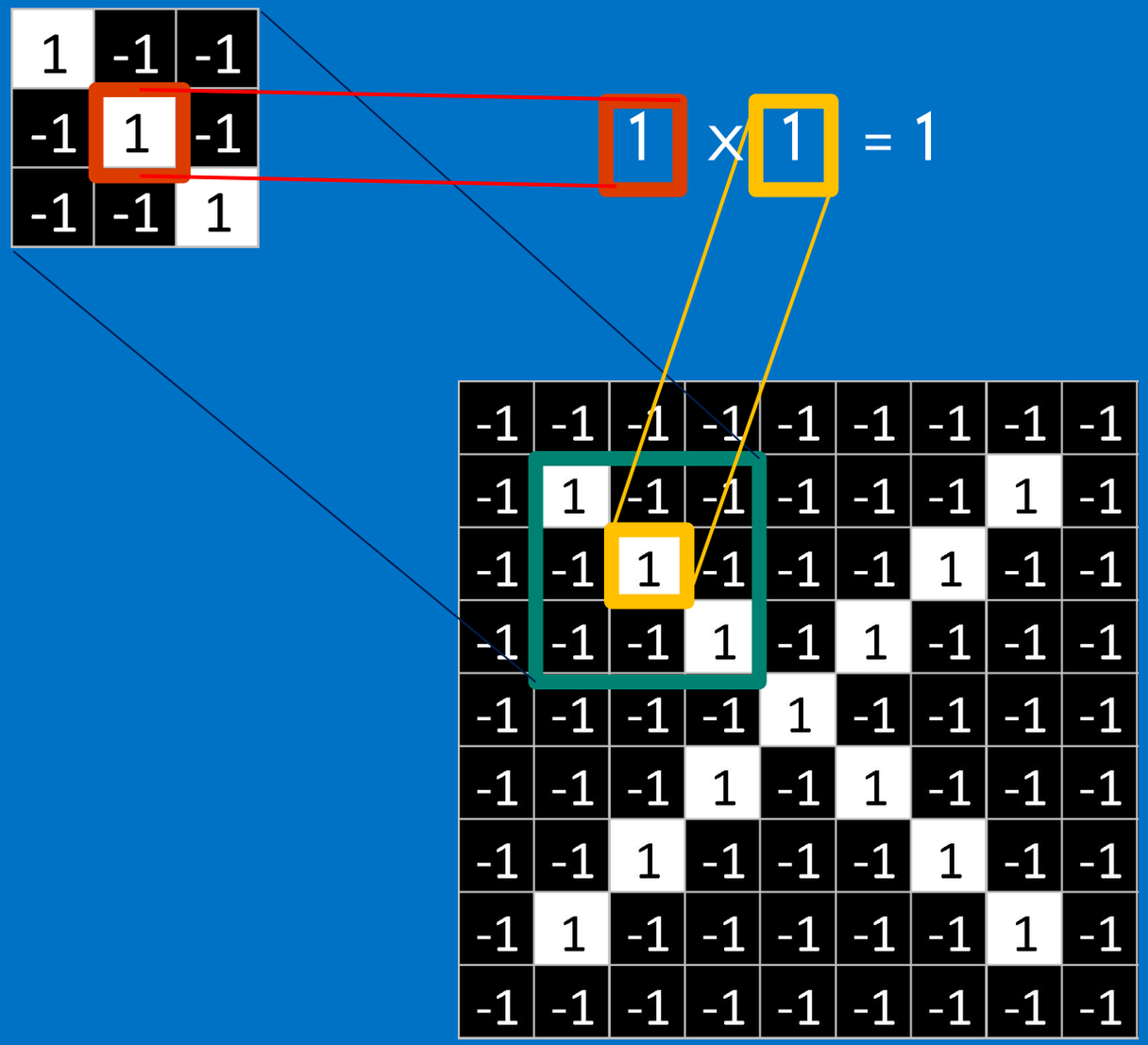


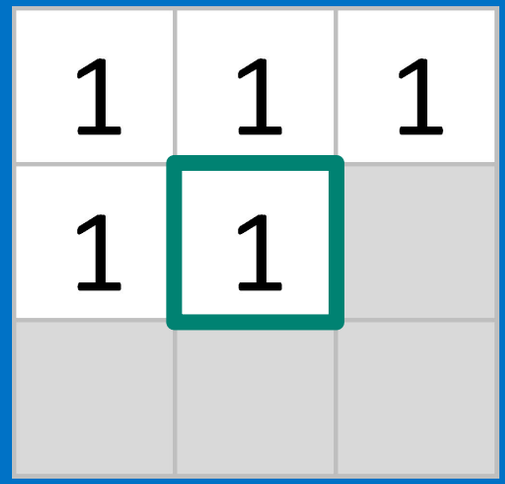
Filtering: The math behind the match



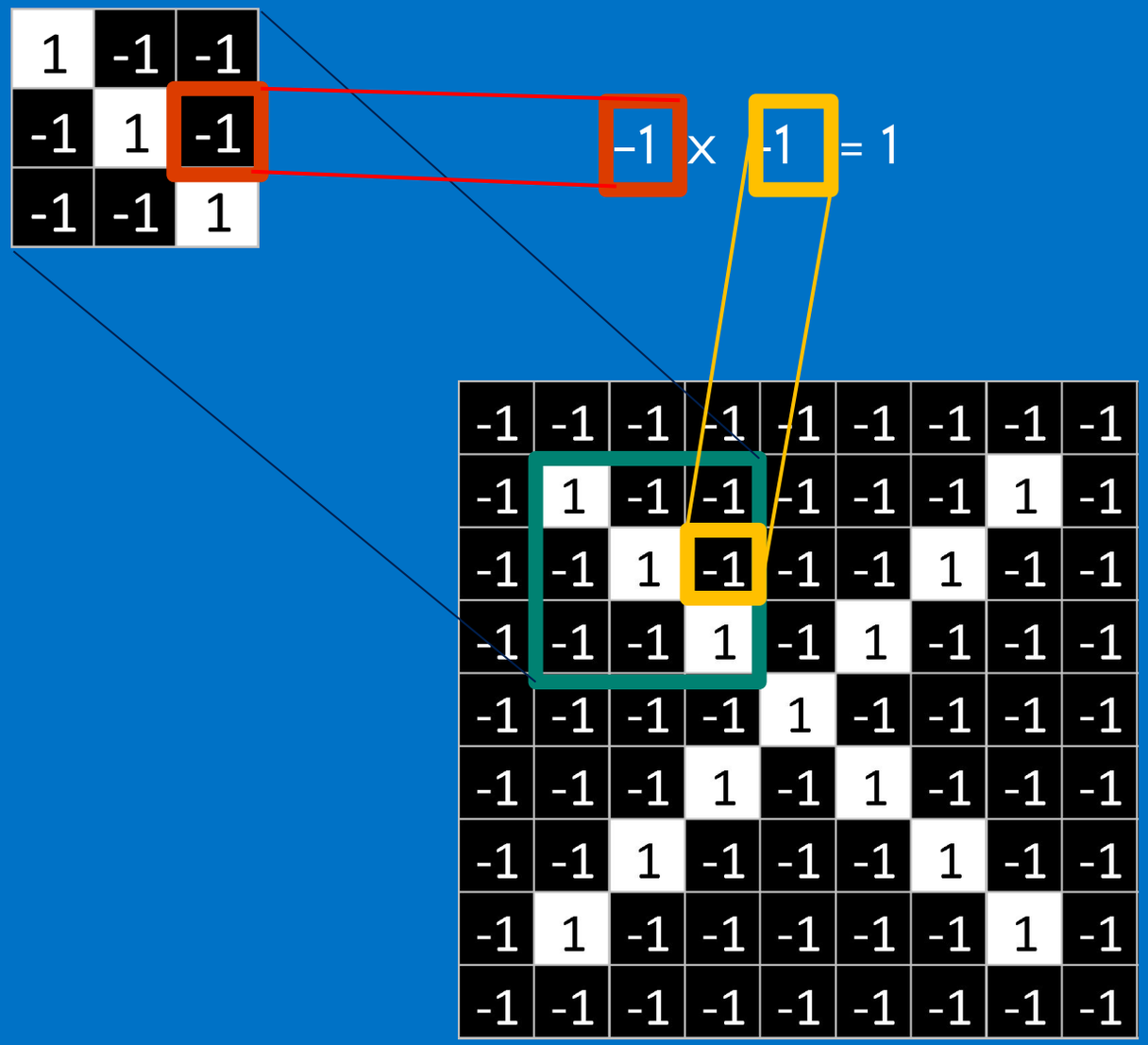


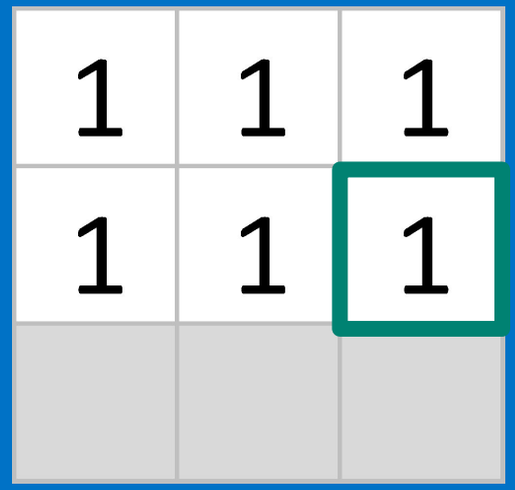
Filtering: The math behind the match



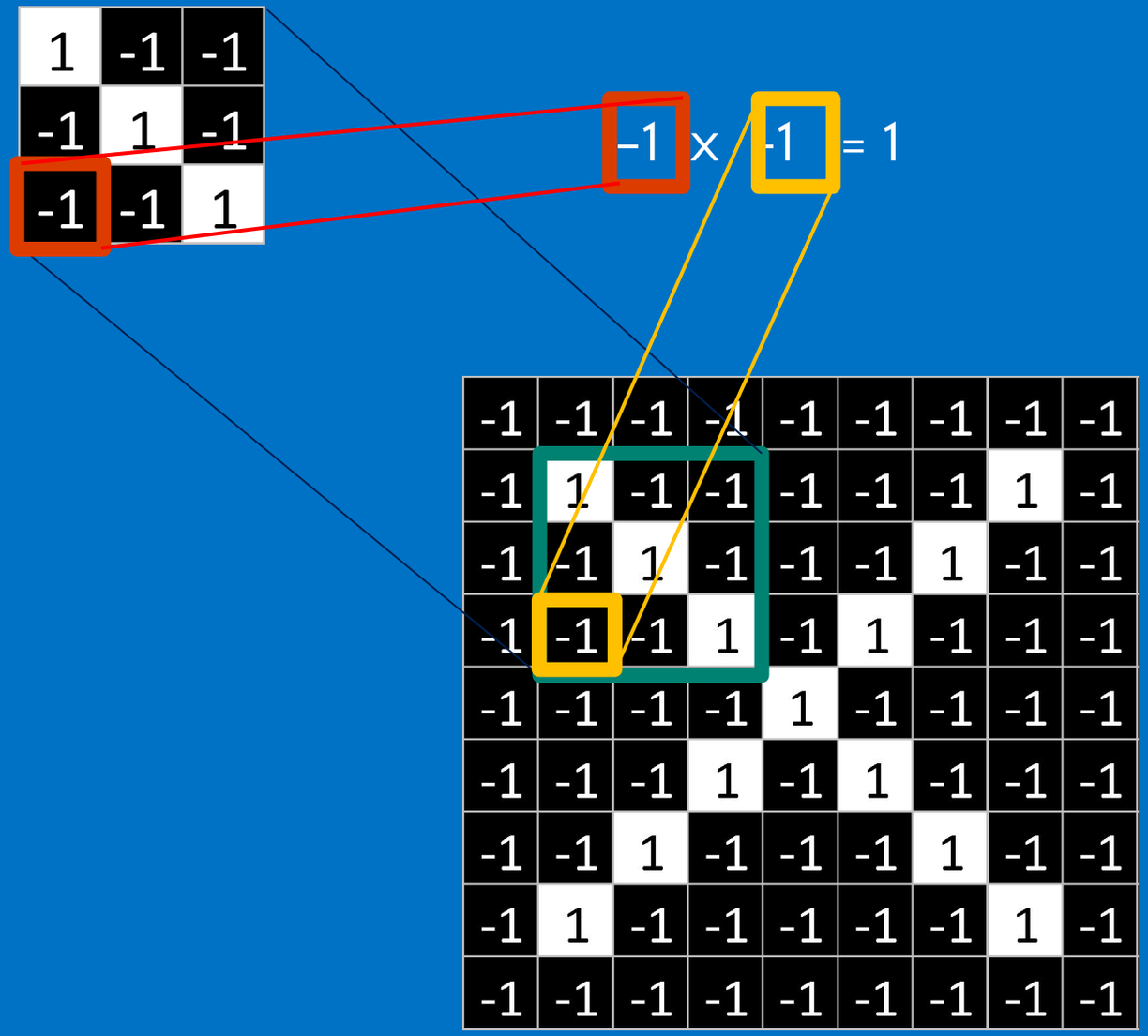


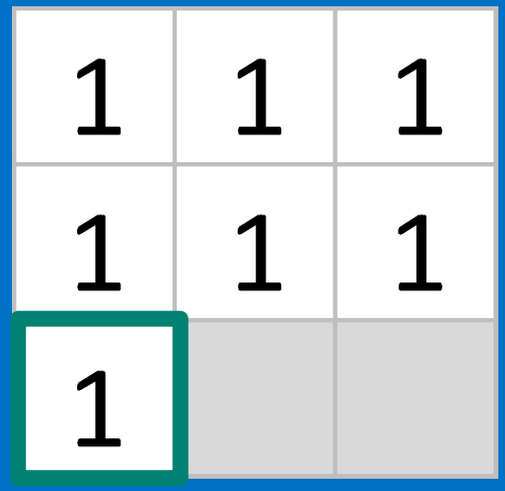
Filtering: The math behind the match



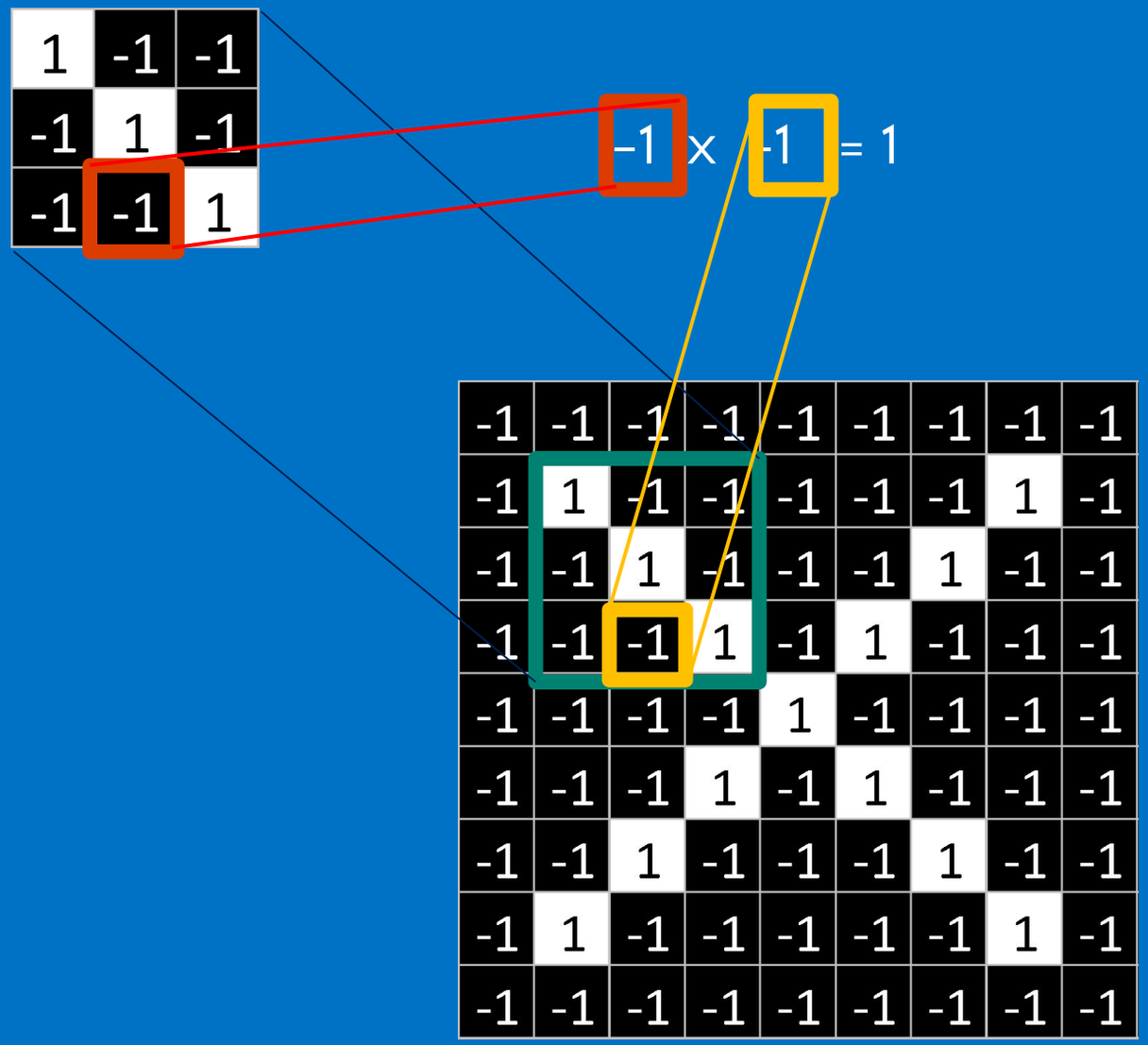


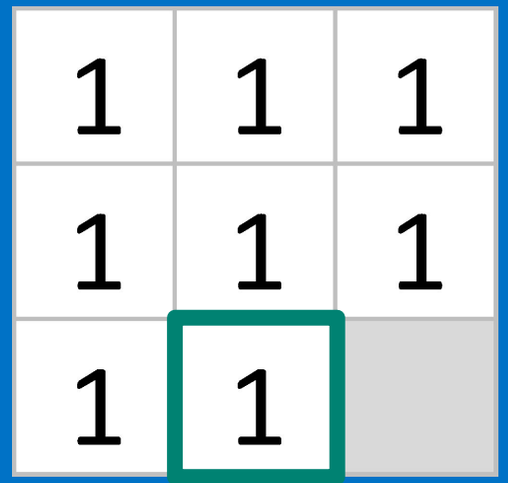
Filtering: The math behind the match



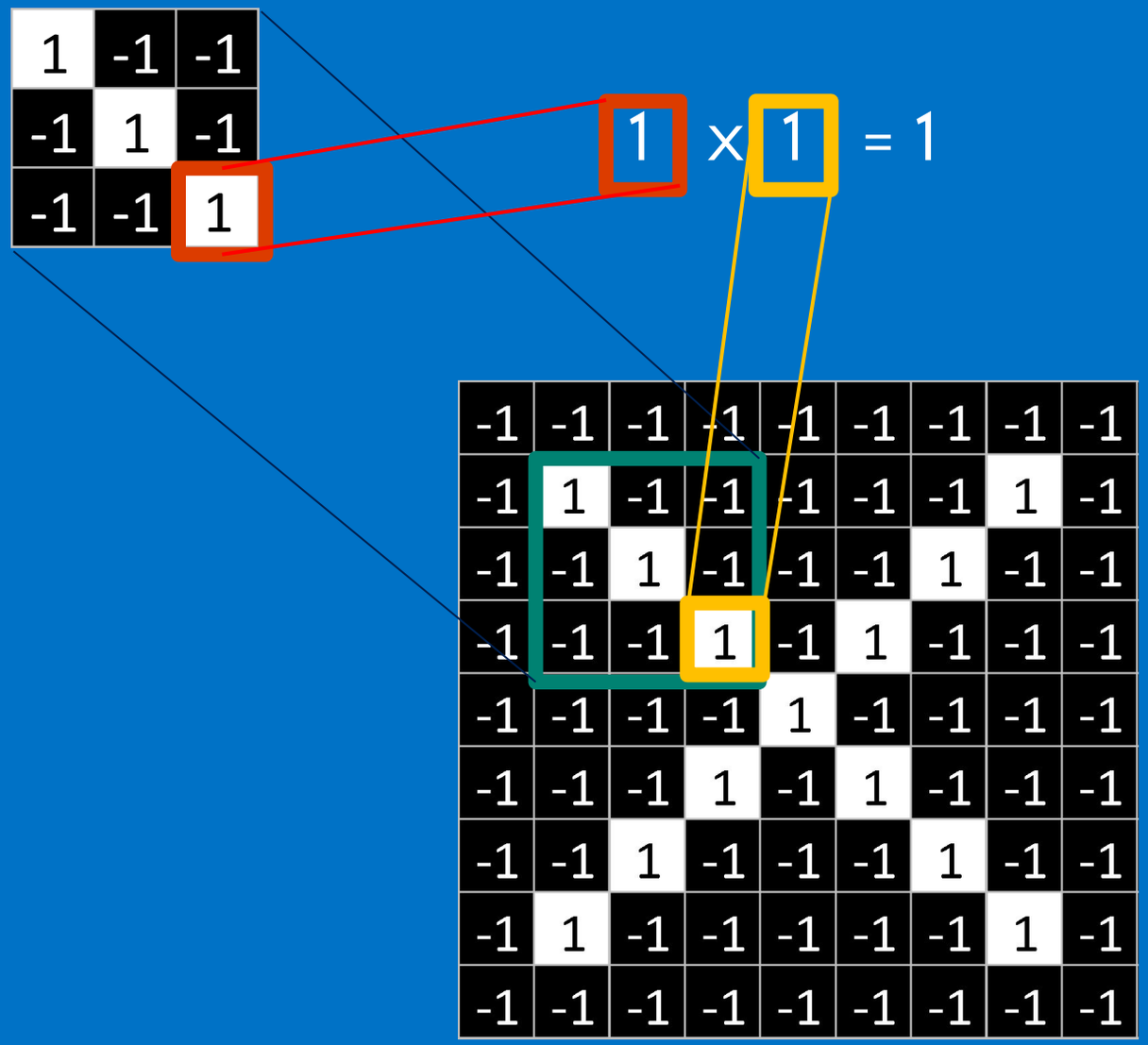


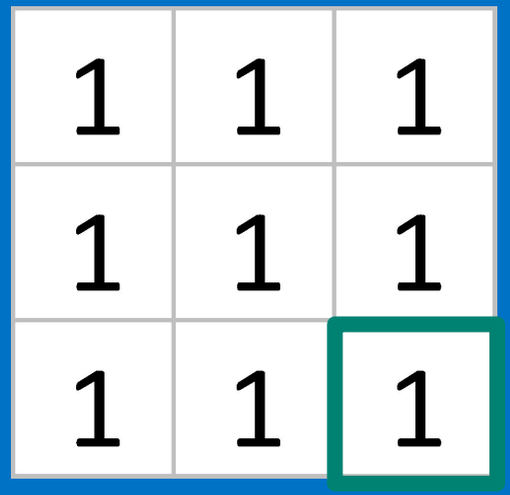
Filtering: The math behind the match



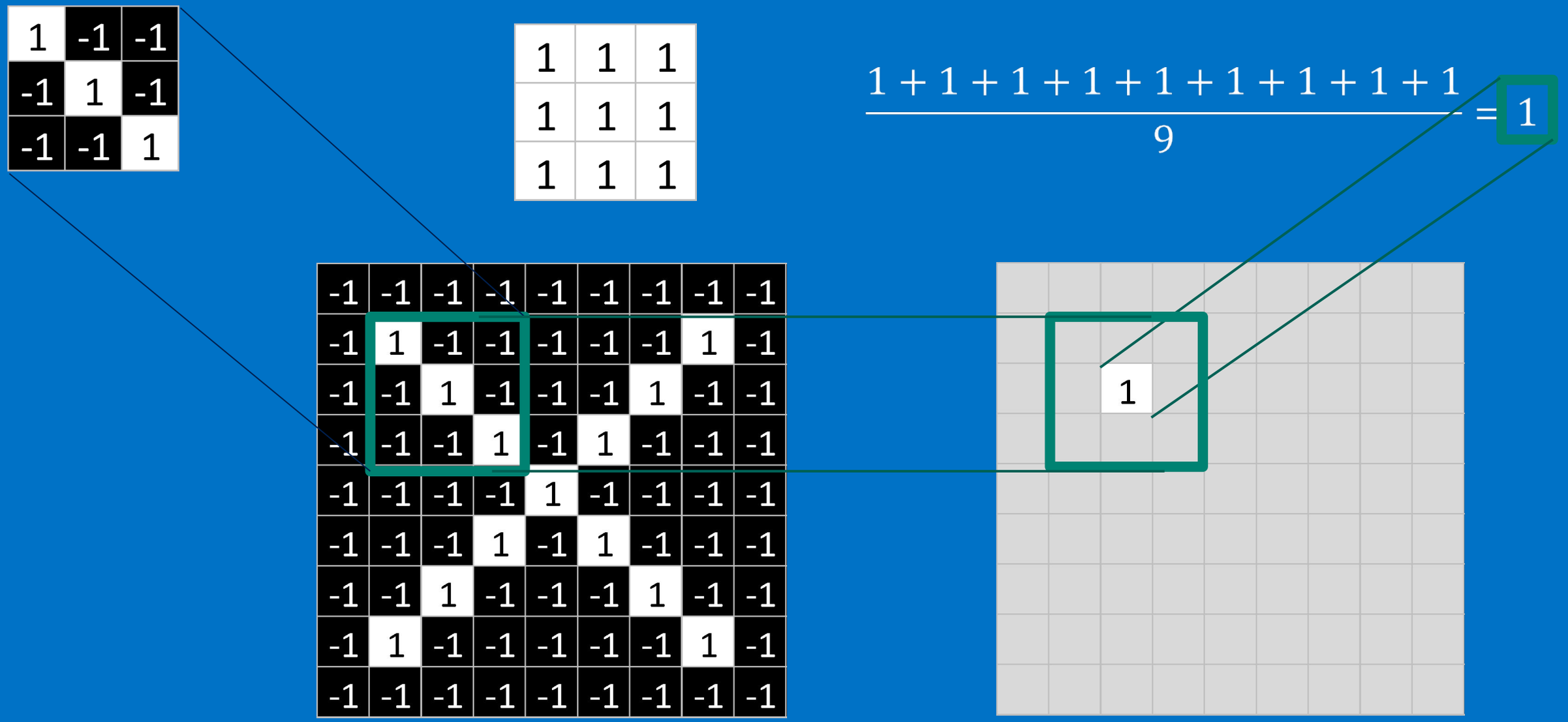


Filtering: The math behind the match

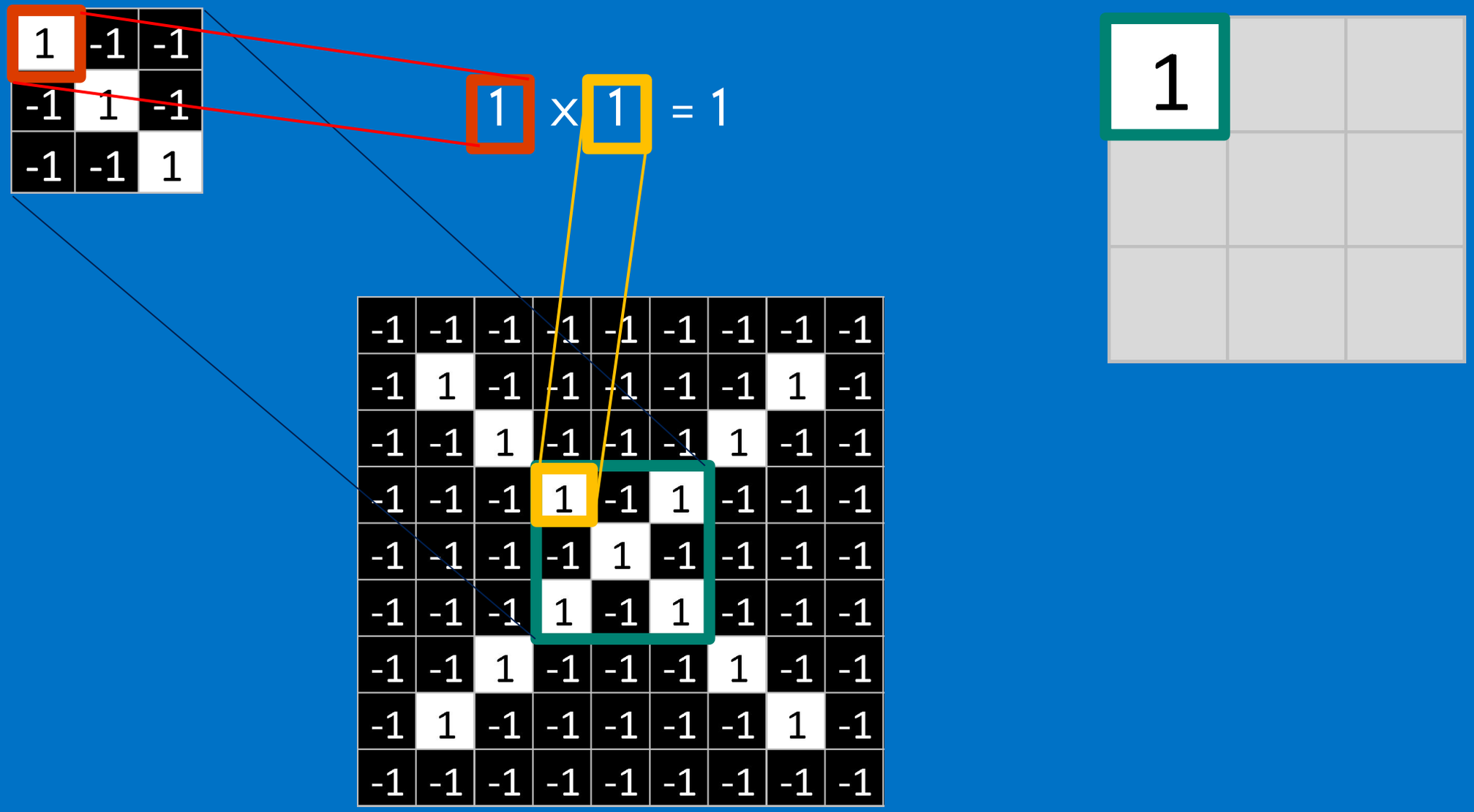




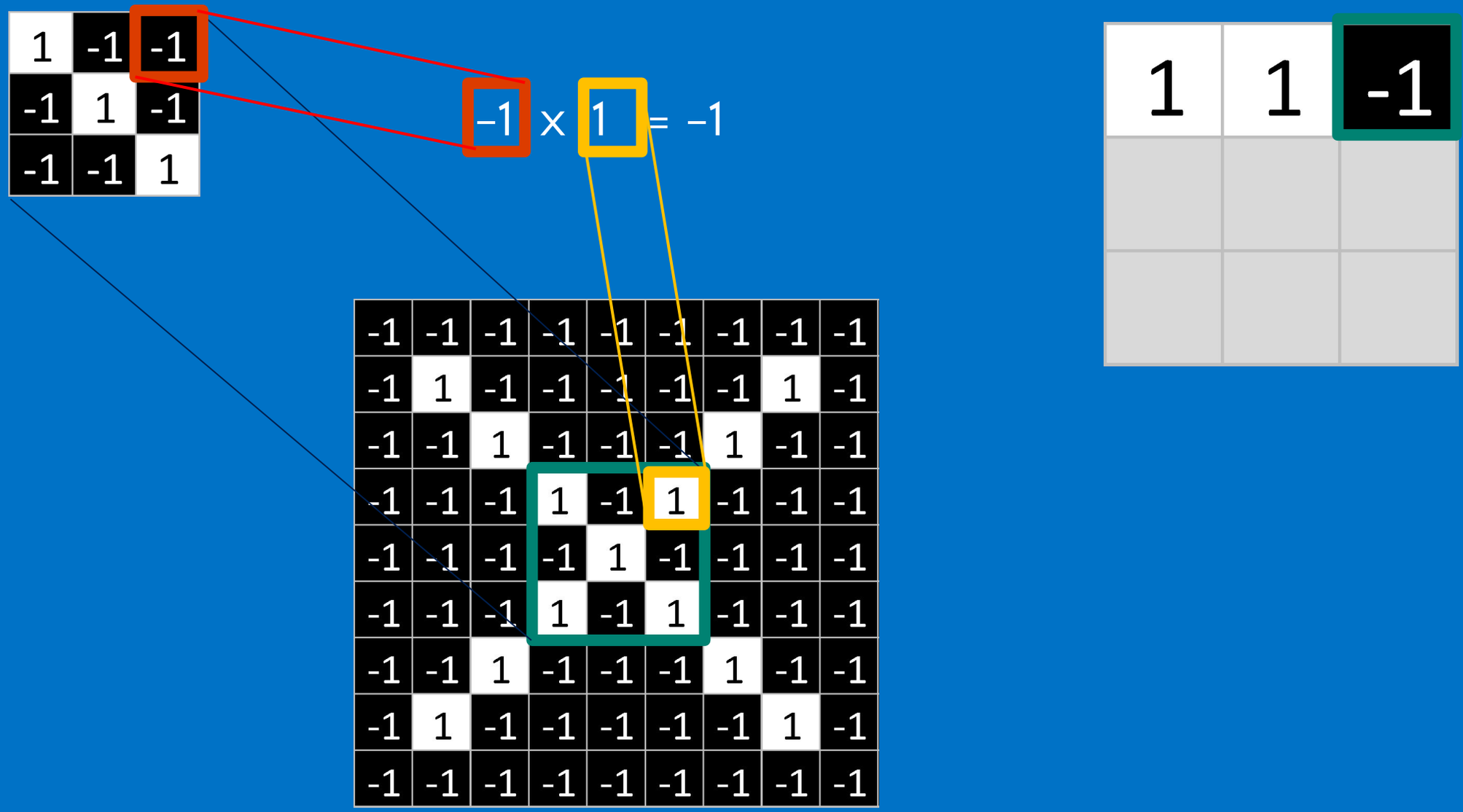
Filtering: The math behind the match



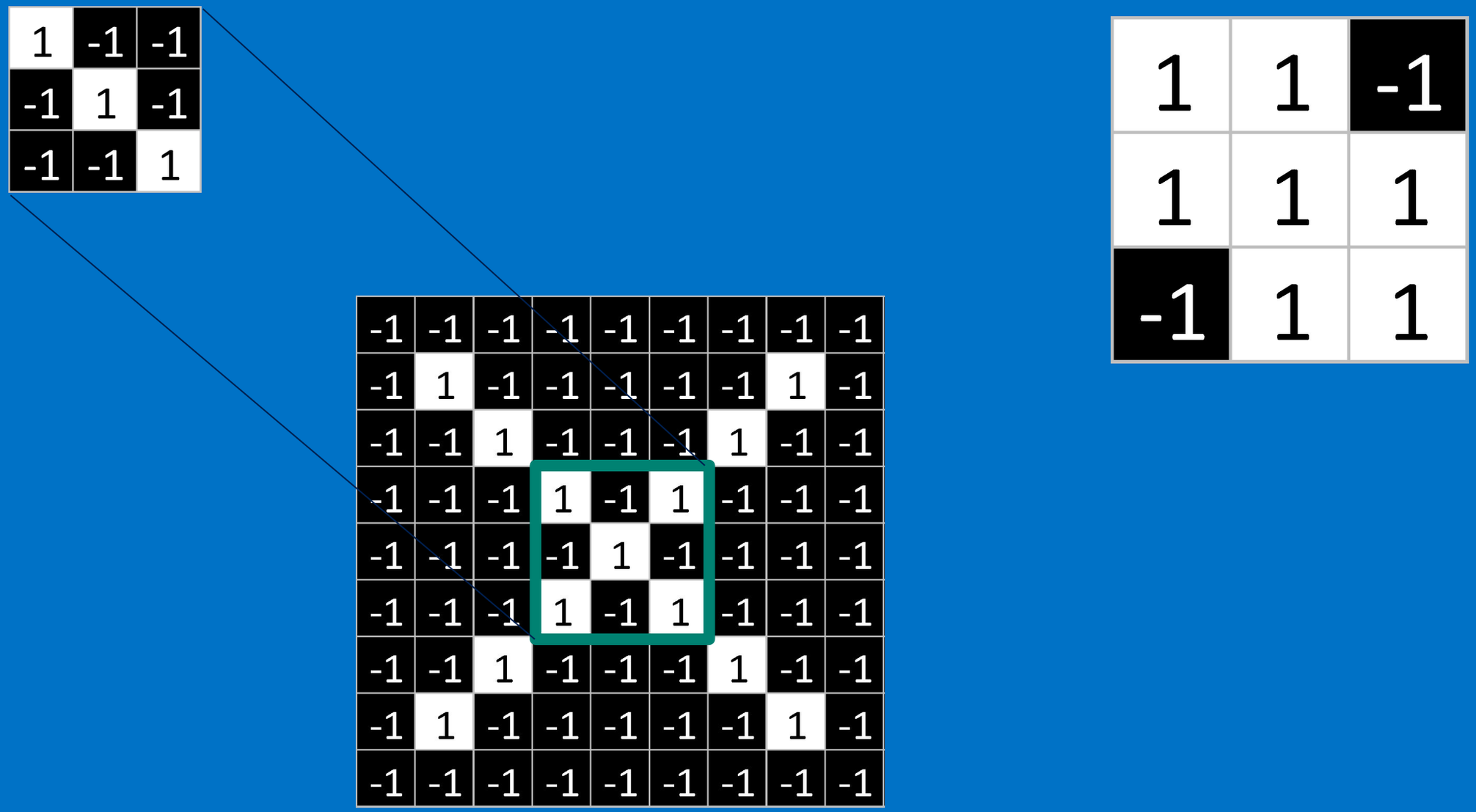
Filtering: The math behind the match



Filtering: The math behind the match



Filtering: The math behind the match



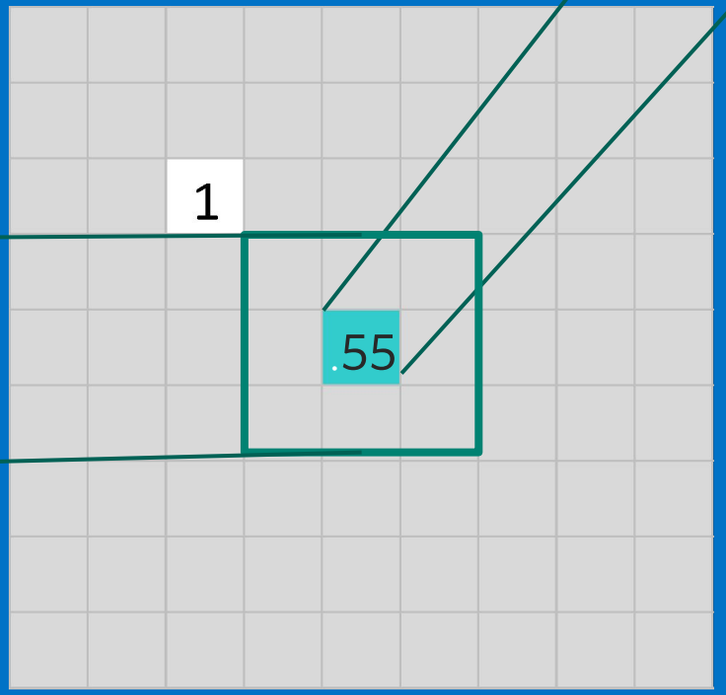
Filtering: The math behind the match

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

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

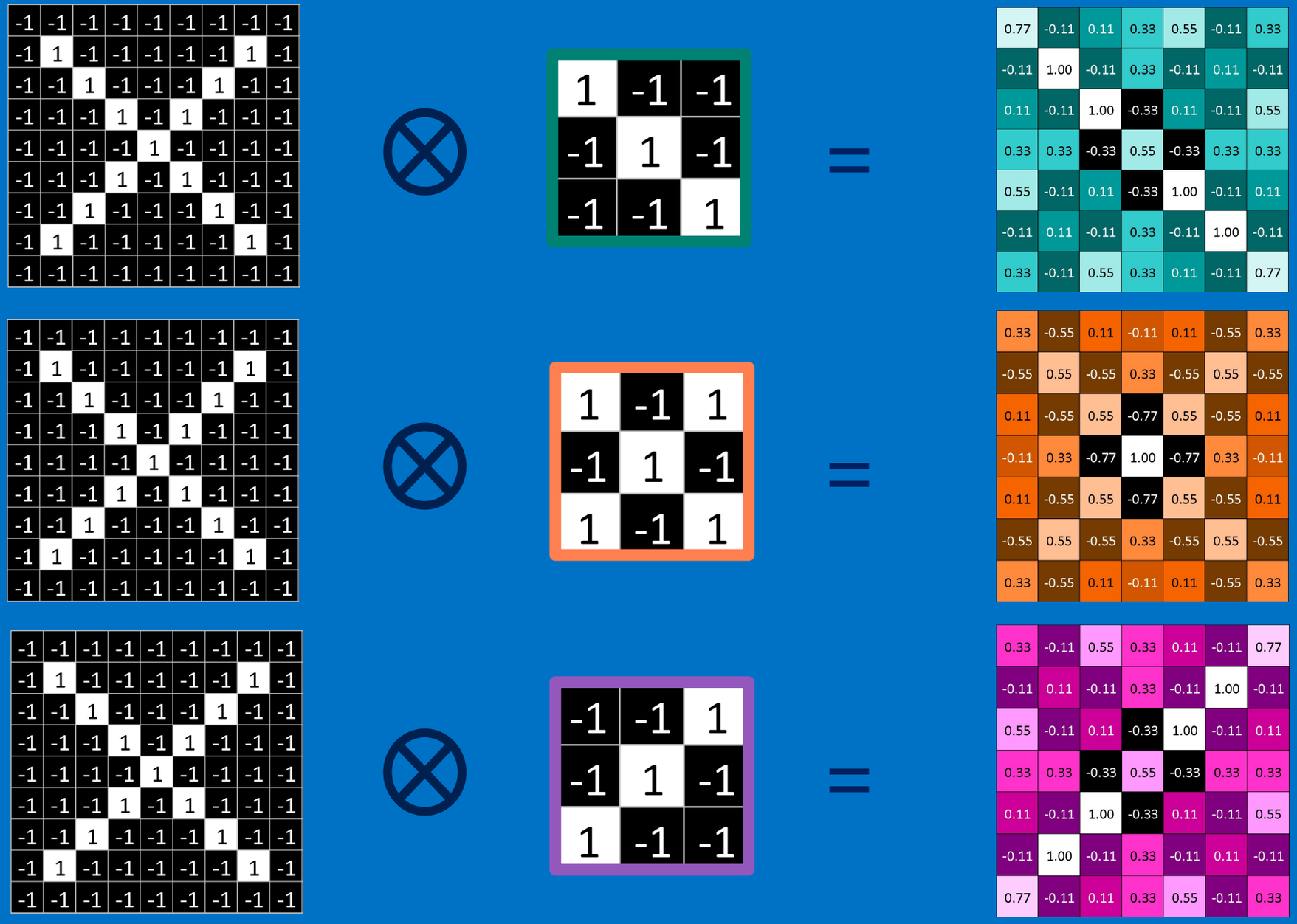
-1	-1	-1	-1	-1	-1	-1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1
-1	-1	1	-1	-1	-1	1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	-1	-1	1	-1	-1	-1	-1
-1	-1	-1	1	-1	1	-1	-1	-1
-1	-1	1	-1	-1	-1	1	-1	-1
-1	1	-1	-1	-1	-1	-1	1	-1
-1	-1	-1	-1	-1	-1	-1	-1	-1

$\otimes$

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

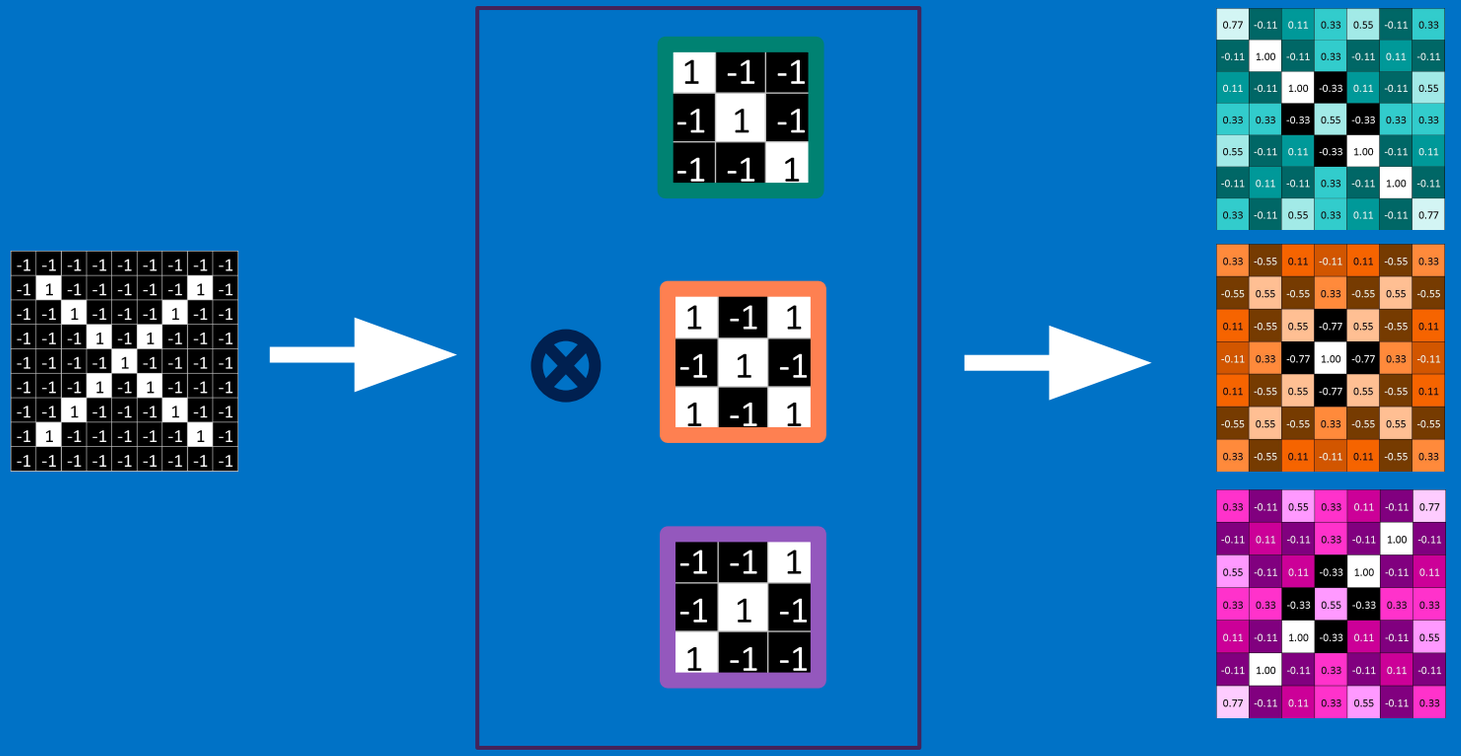
$=$

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
=
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
=
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
=
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

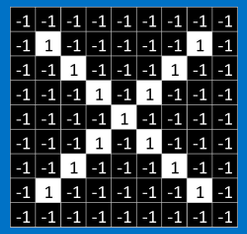
Convolution layer

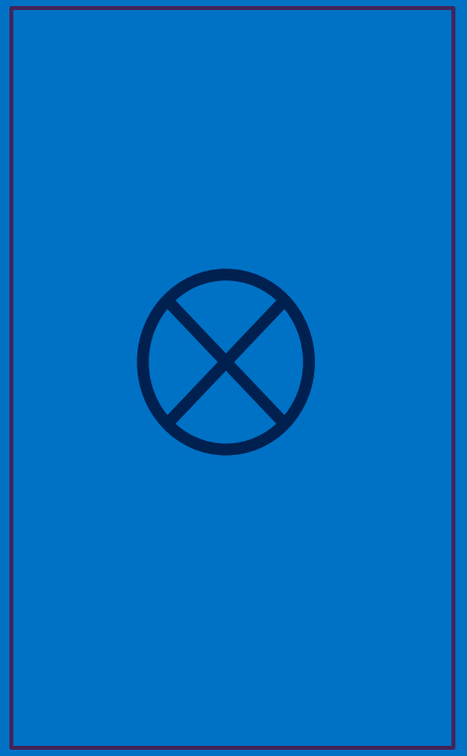
One image becomes a stack of filtered images

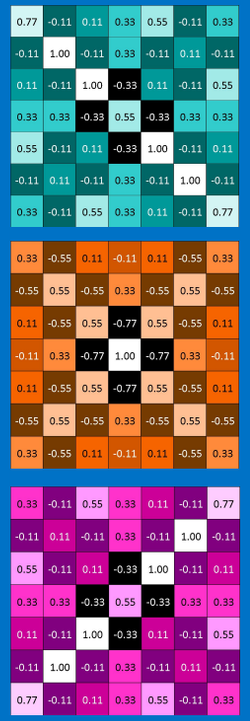


Convolution layer

One image becomes a stack of filtered images



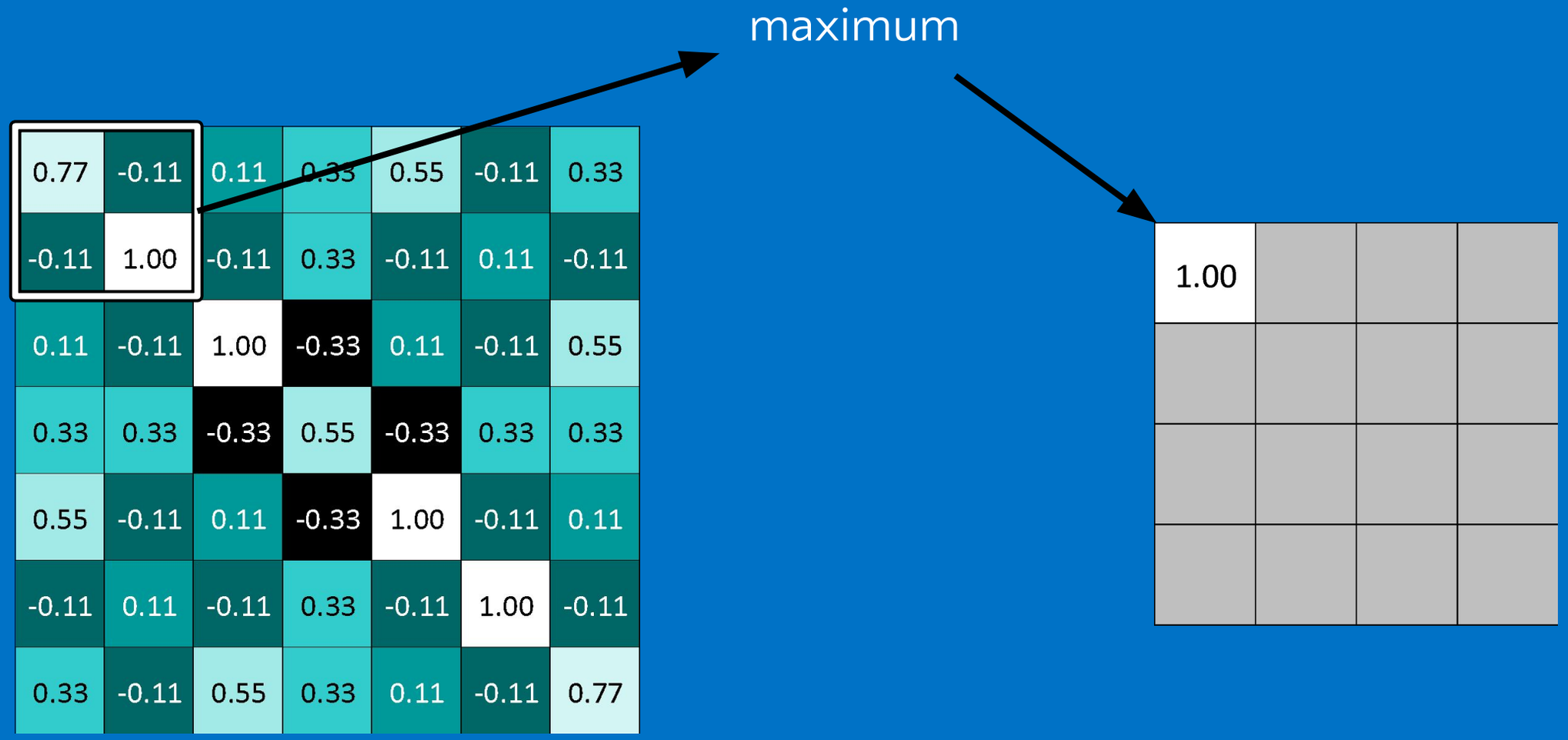




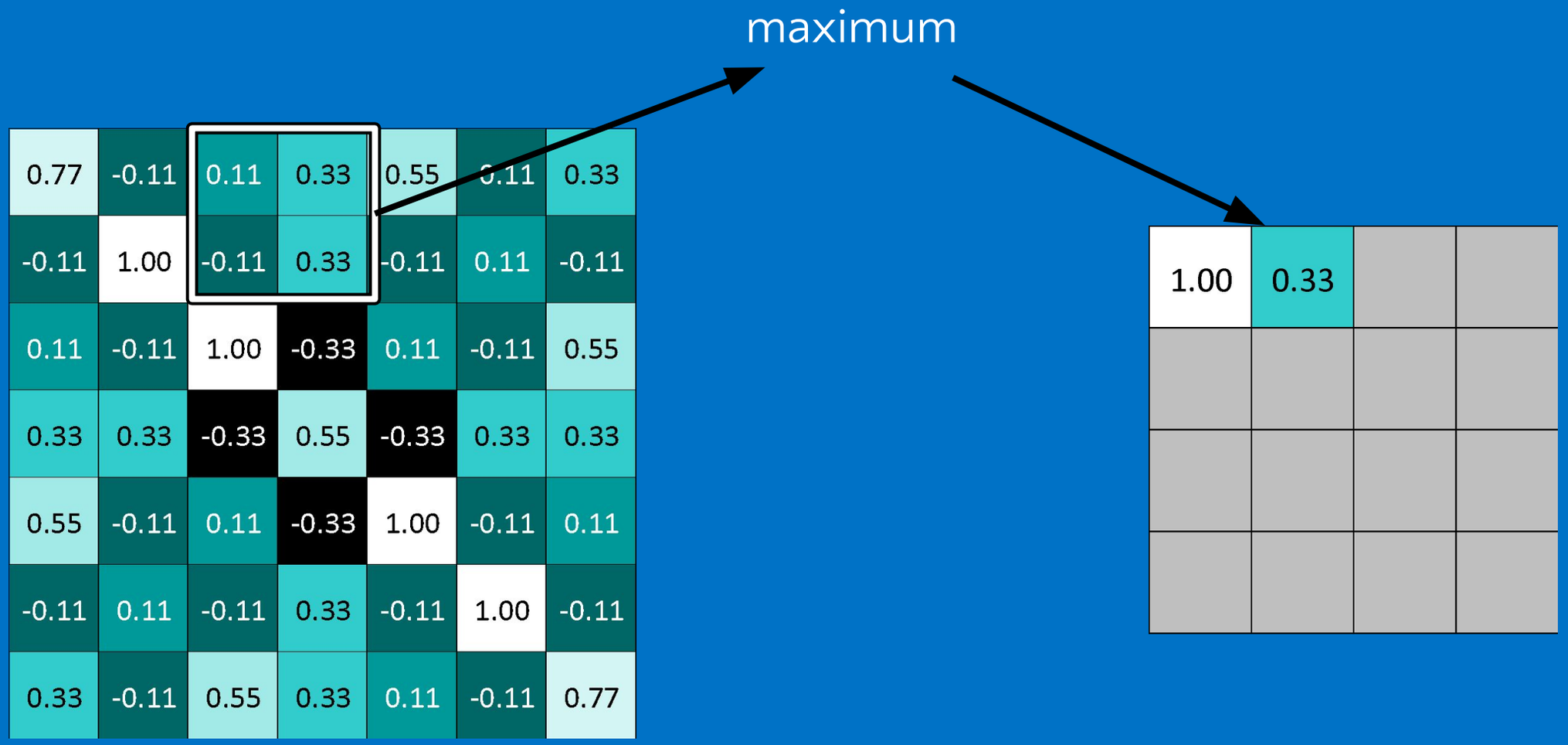
Pooling: Shrinking the image stack

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

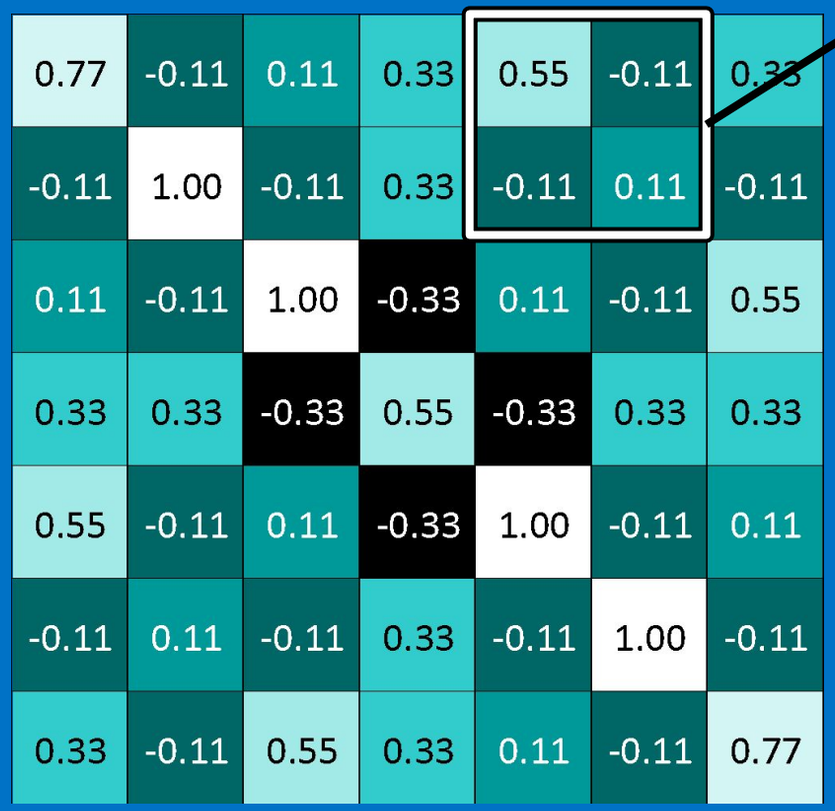

maximum
0.77 -0.11 0.11 0.33 0.55 -0.11 0.33
-0.11 1.00 -0.11 0.33 -0.11 0.11 -0.11
0.11 -0.11 1.00 -0.33 0.11 -0.11 0.55
0.33 0.33 -0.33 0.55 -0.33 0.33 0.33
0.55 -0.11 0.11 -0.33 1.00 -0.11 0.11
-0.11 0.11 -0.11 0.33 -0.11 1.00 -0.11
0.33 -0.11 0.55 0.33 0.11 -0.11 0.77
1.00

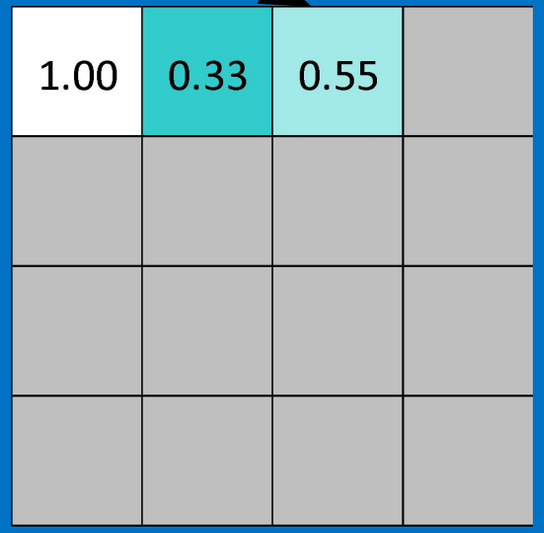
Pooling


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

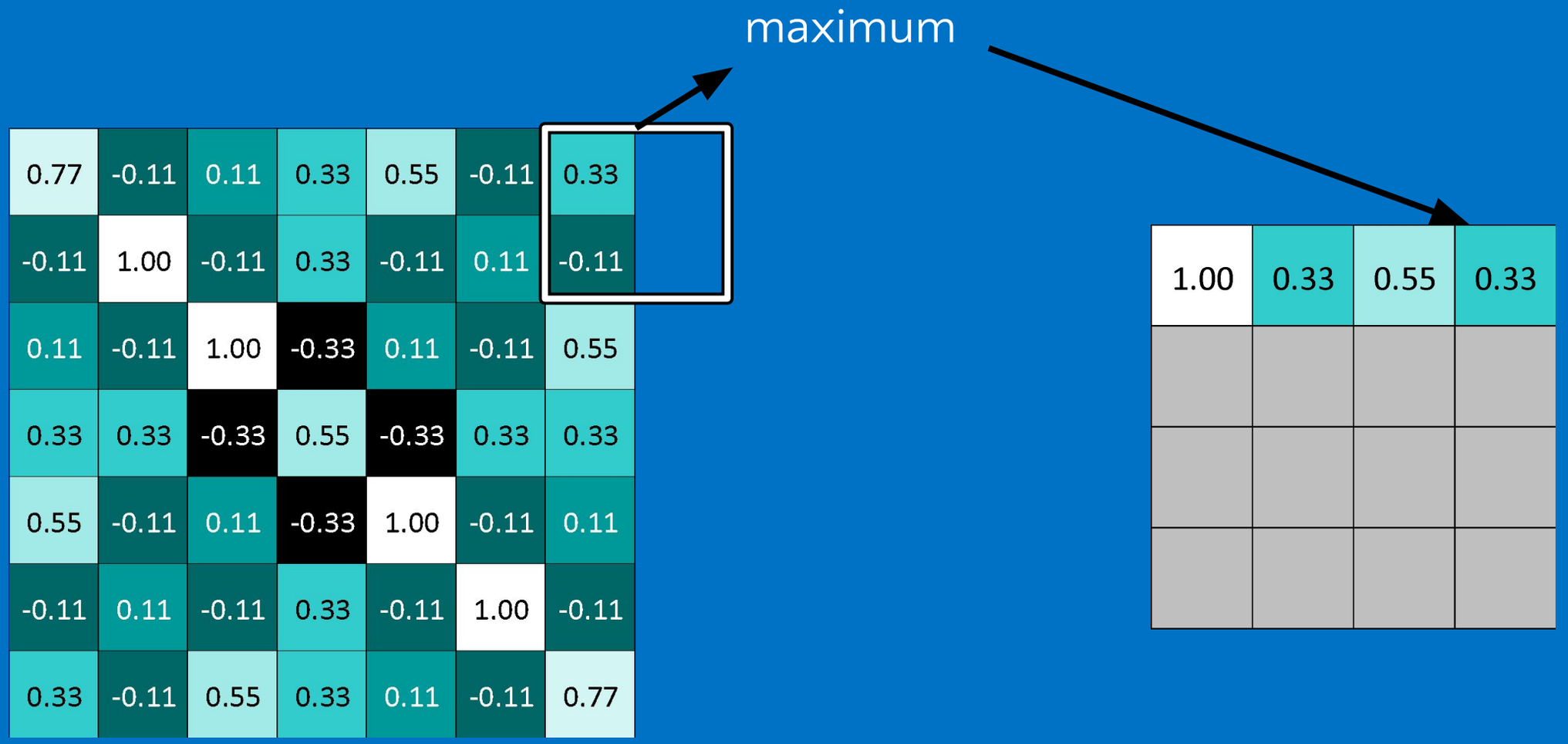
Pooling

maximum

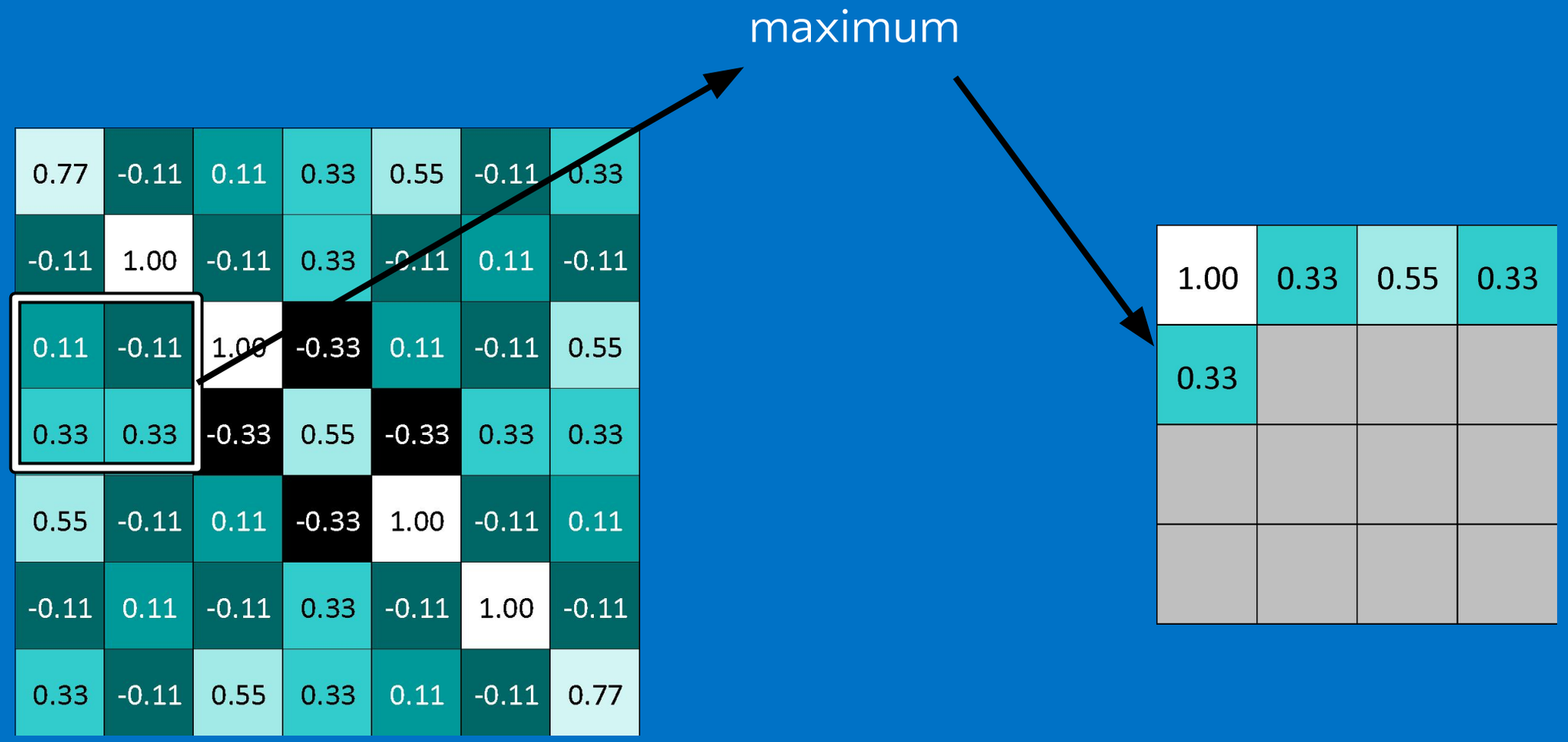




Pooling


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

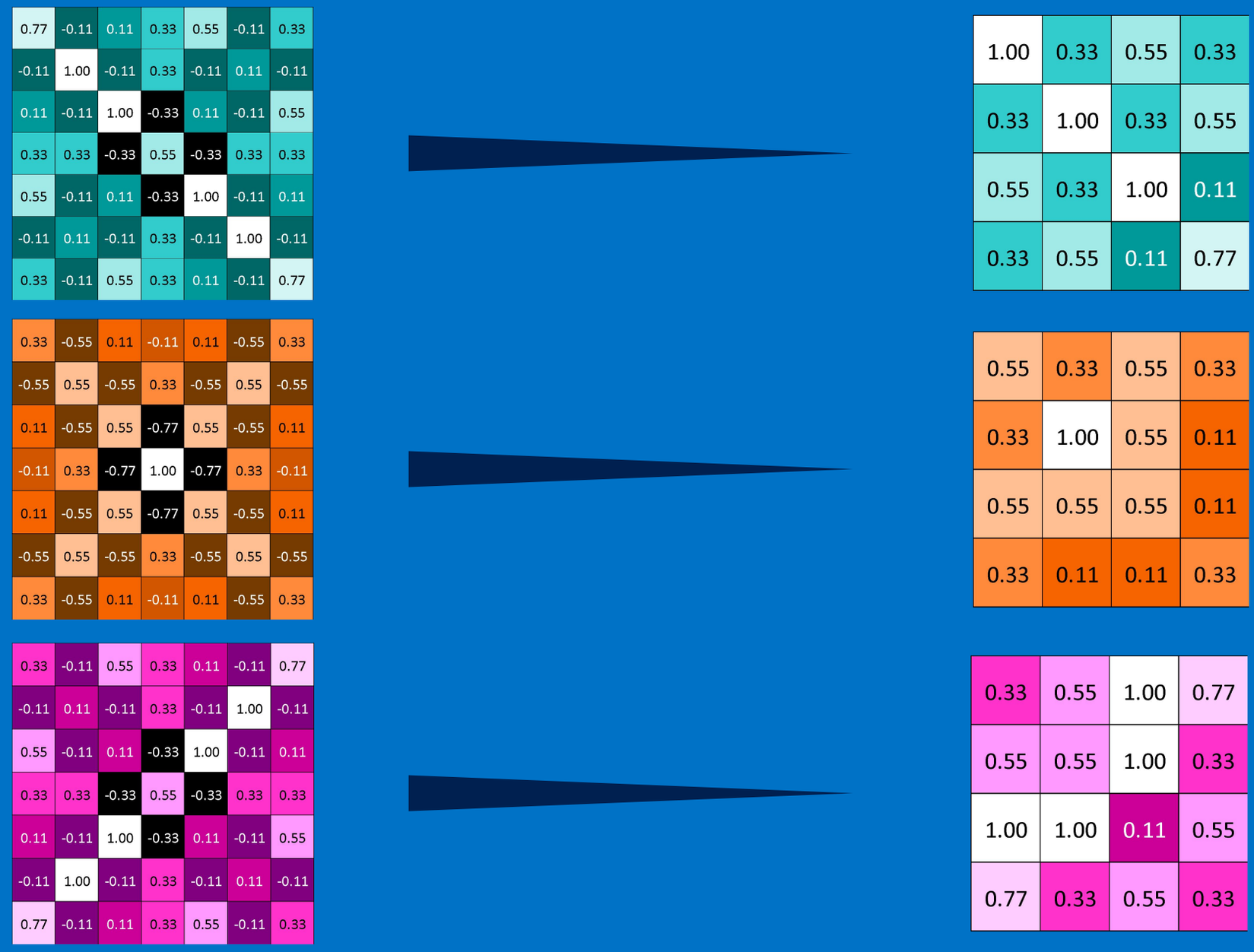
Pooling


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

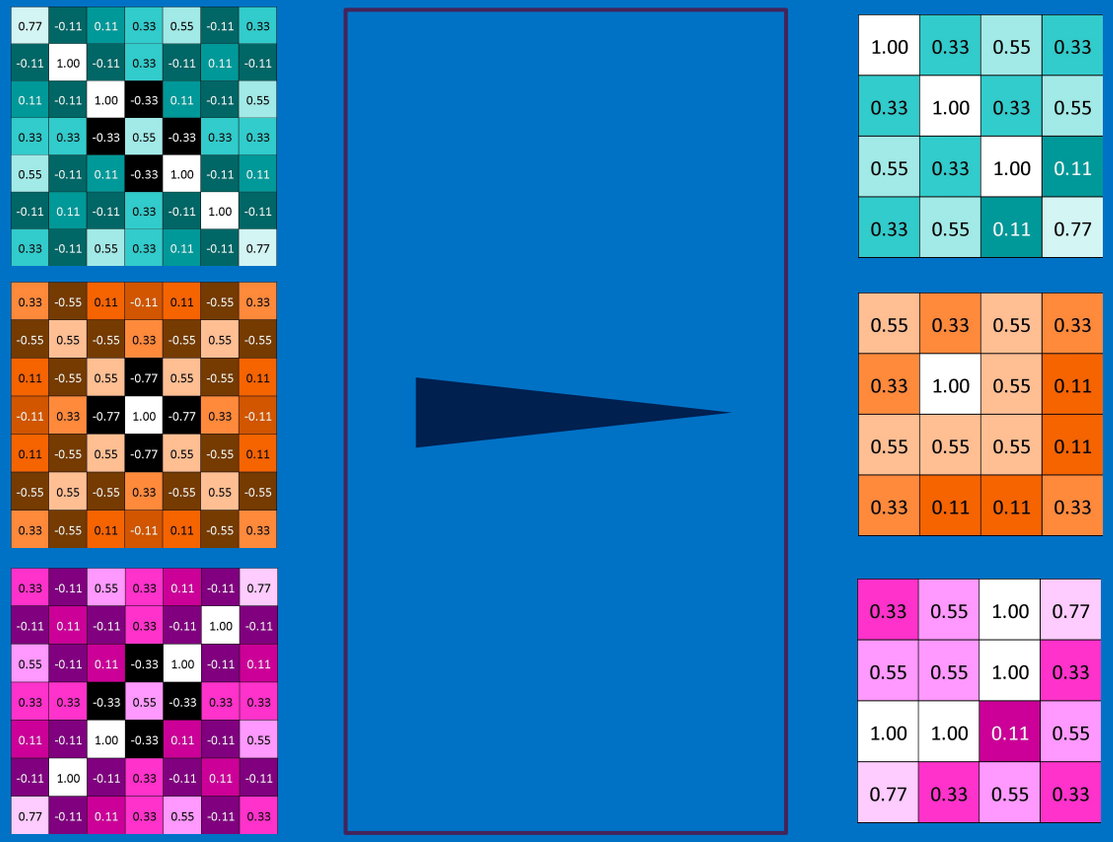
0.77	-0.11	0.11	0.33	0.55	-0.11	0.33
-0.11	1.00	-0.11	0.33	-0.11	0.11	-0.11
0.11	-0.11	1.00	-0.33	0.11	-0.11	0.55
0.33	0.33	-0.33	0.55	-0.33	0.33	0.33
0.55	-0.11	0.11	-0.33	1.00	-0.11	0.11
-0.11	0.11	-0.11	0.33	-0.11	1.00	-0.11
0.33	-0.11	0.55	0.33	0.11	-0.11	0.77

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
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.

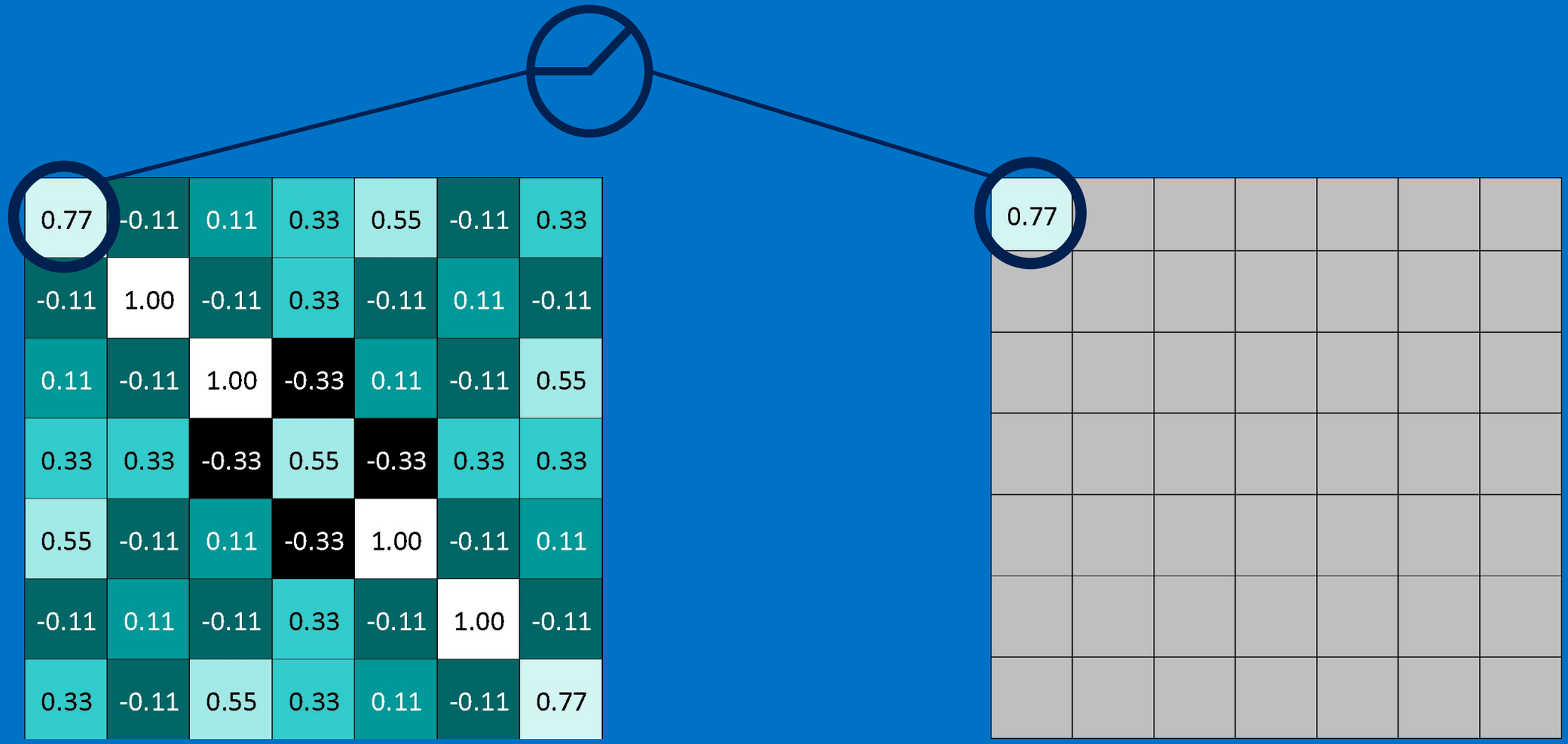


Normalization

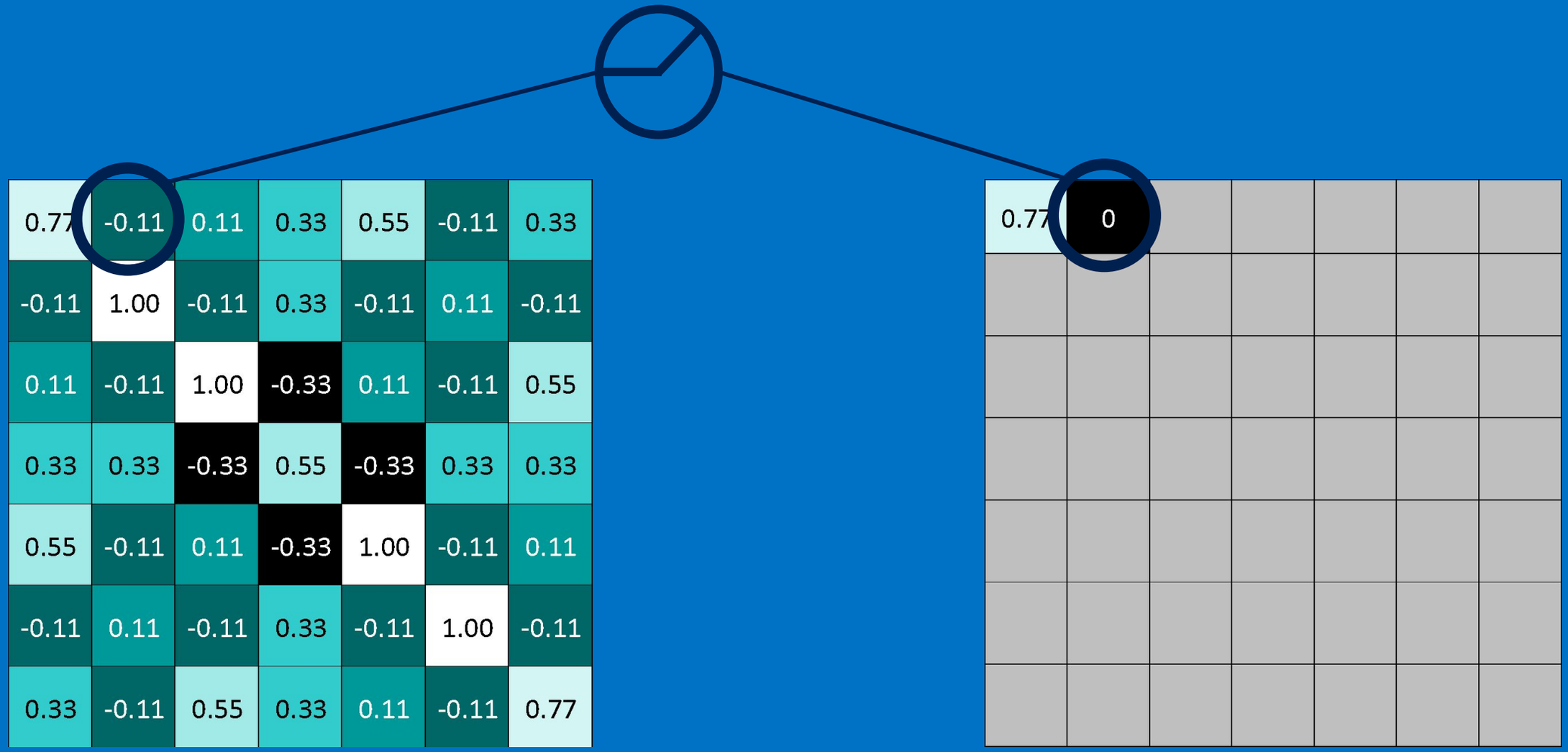
Keep the math from breaking by tweaking each of the values just a bit.

Change everything negative to zero.

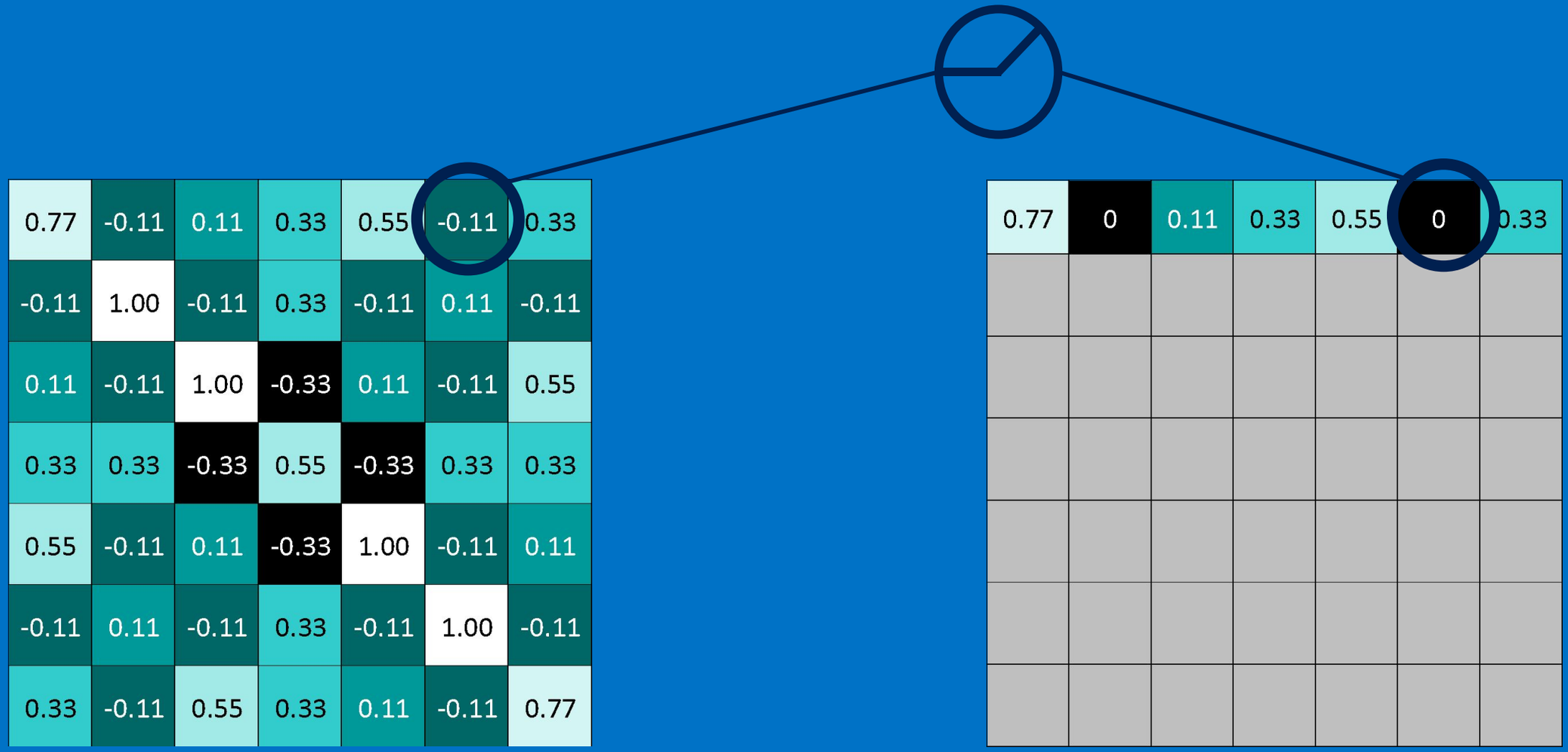
Rectified Linear Units (ReLUs)



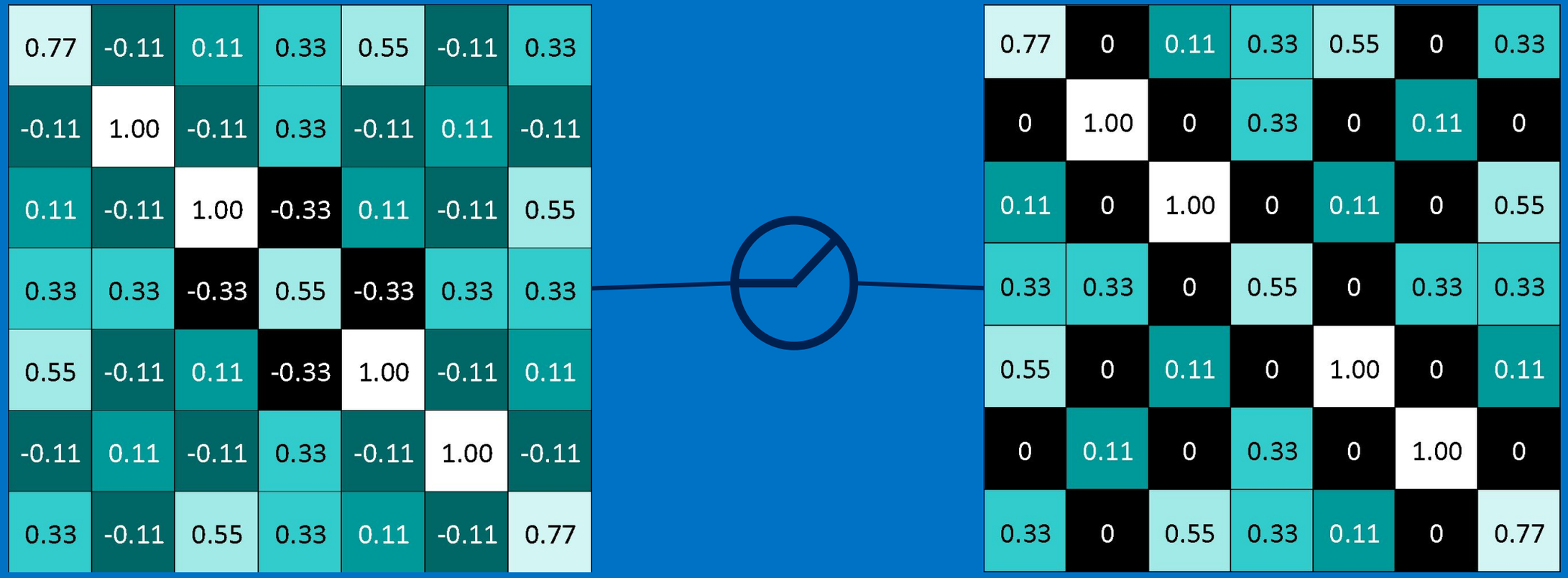
Rectified Linear Units (ReLUs)



Rectified Linear Units (ReLUs)

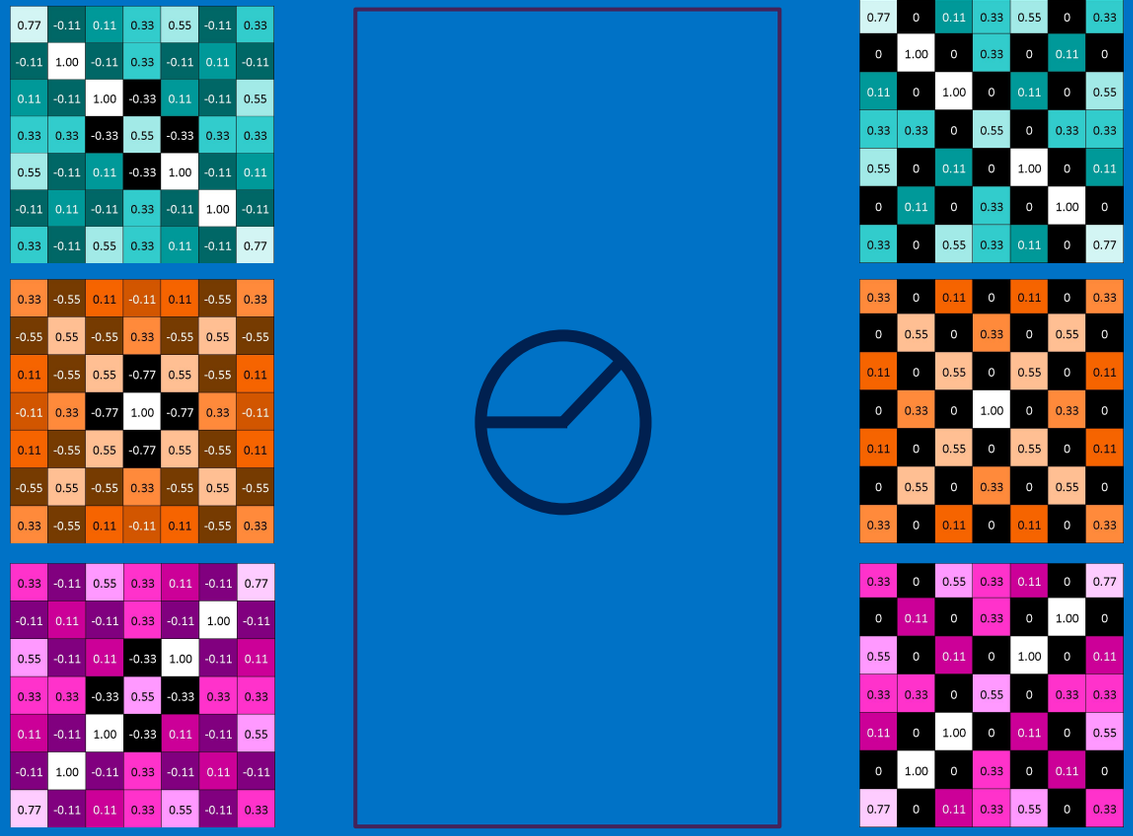


Rectified Linear Units (ReLUs)



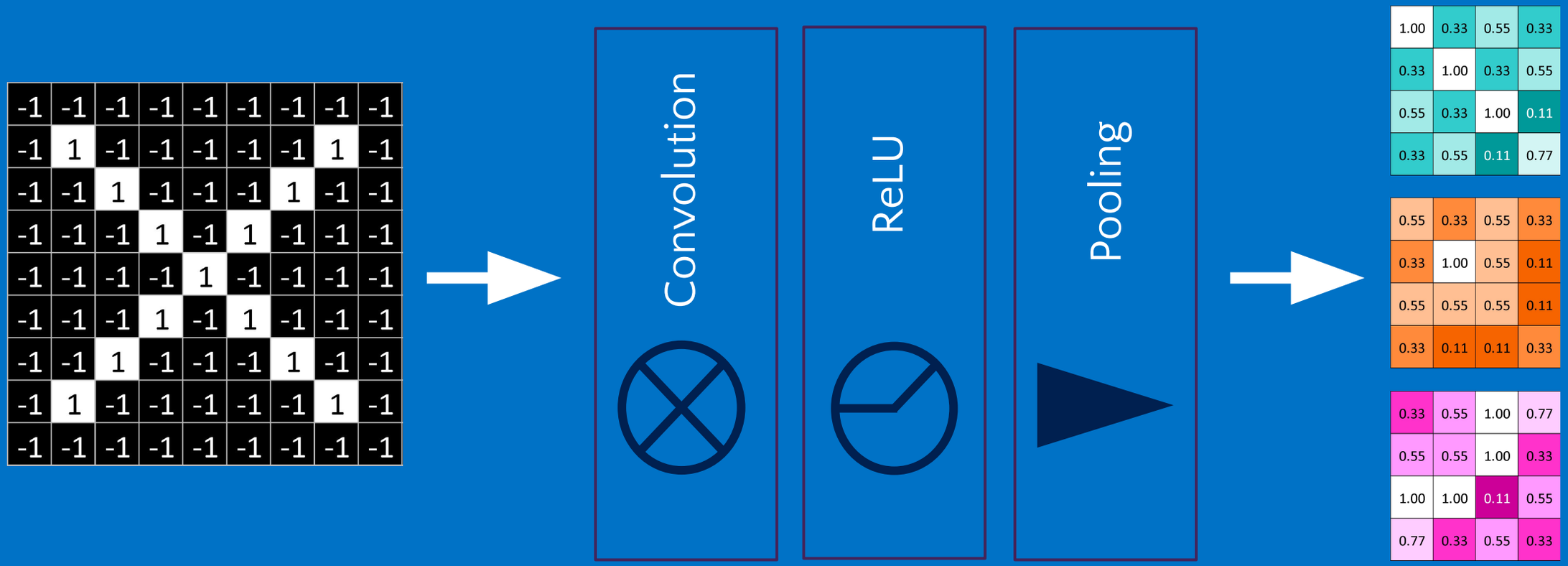
ReLU layer

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



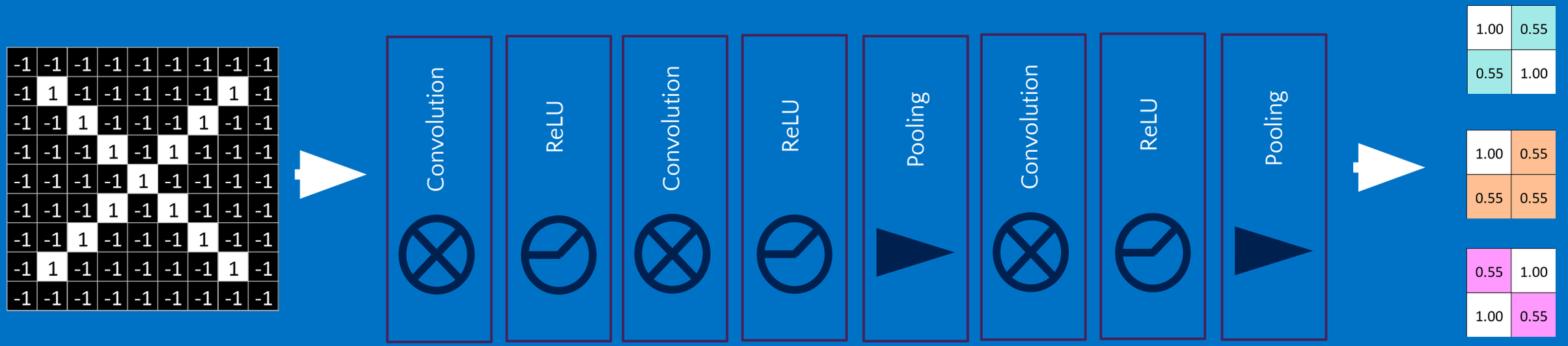
Layers get stacked

The output of one becomes the input of the next.



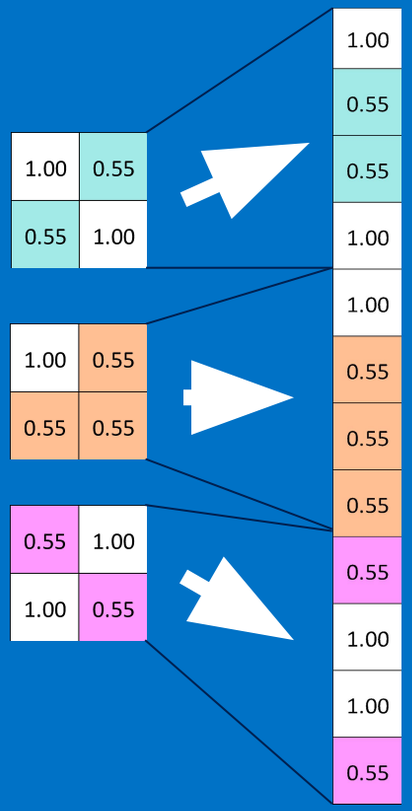
Deep stacking

Layers can be repeated several (or many) times.



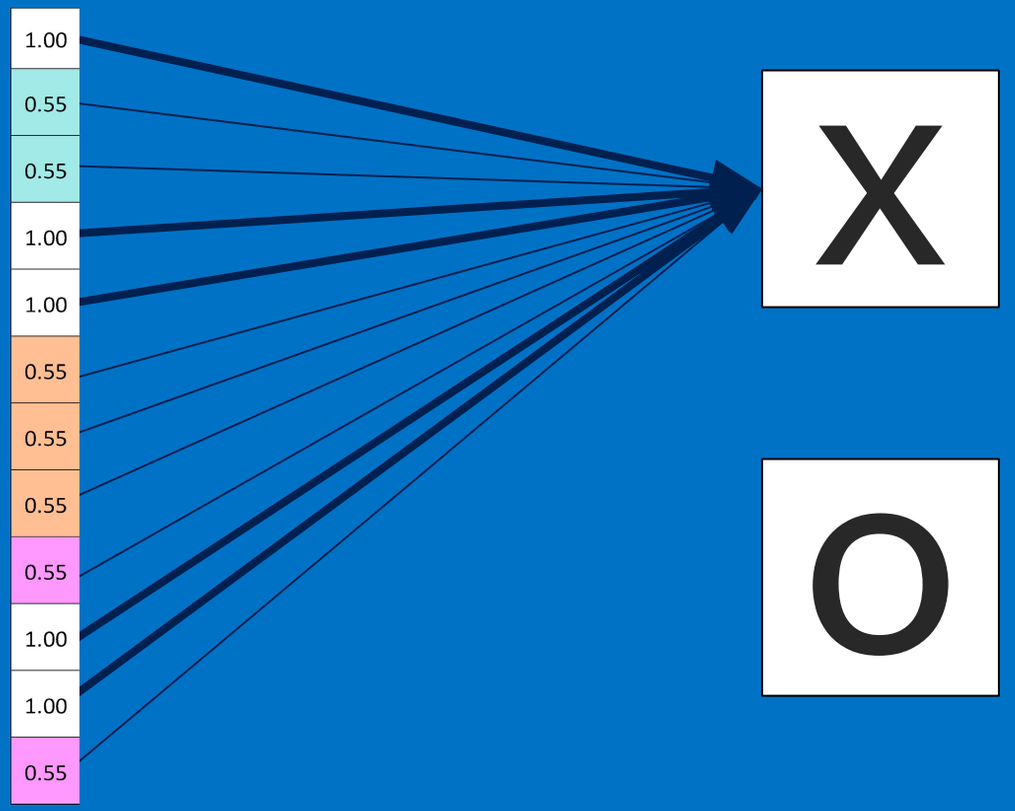
Fully connected layer

Every value gets a vote



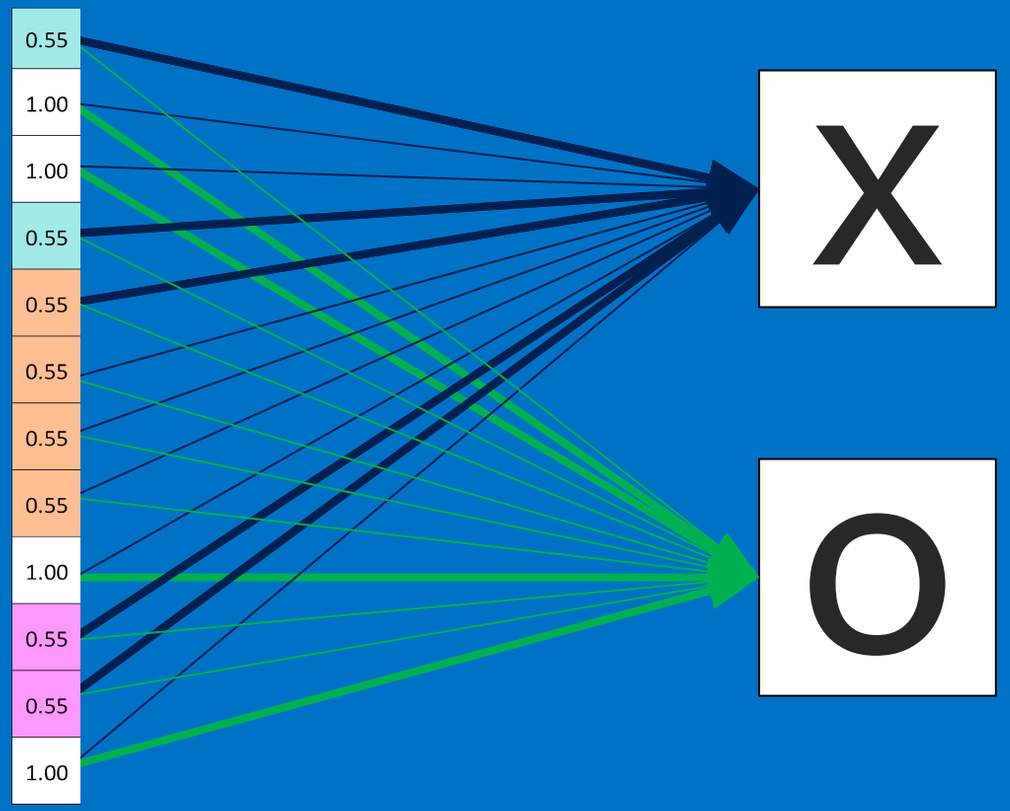
Fully connected layer

Vote depends on how strongly a value predicts X or O



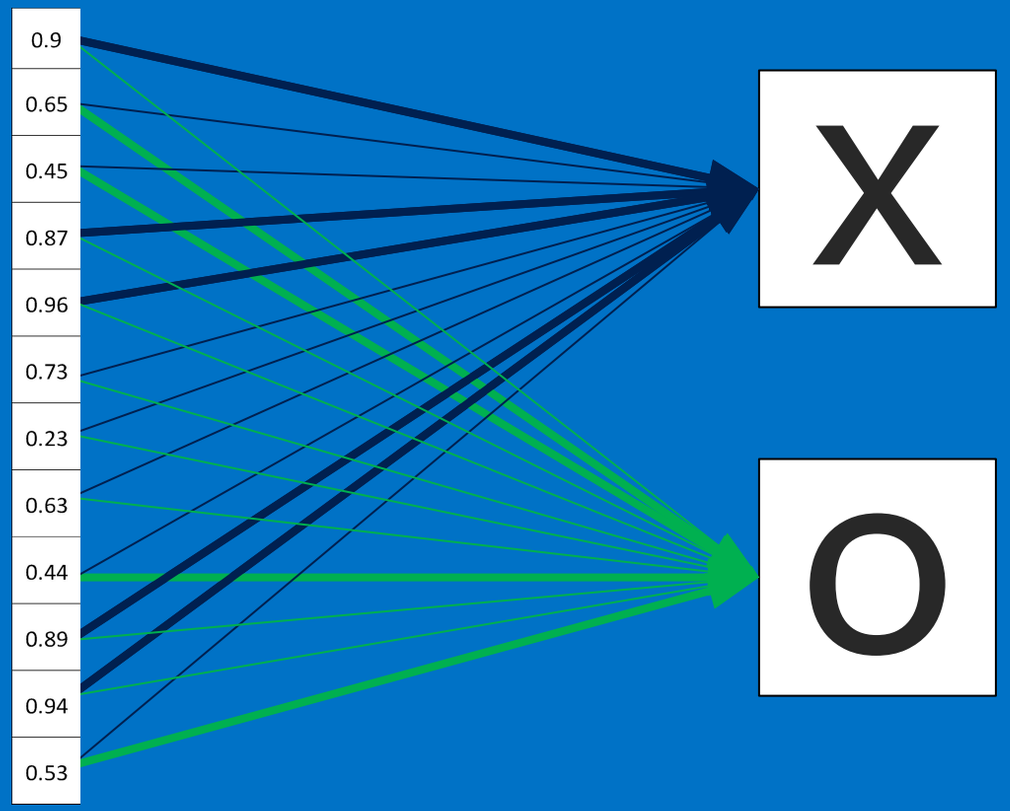
Fully connected layer

Vote depends on how strongly a value predicts X or O



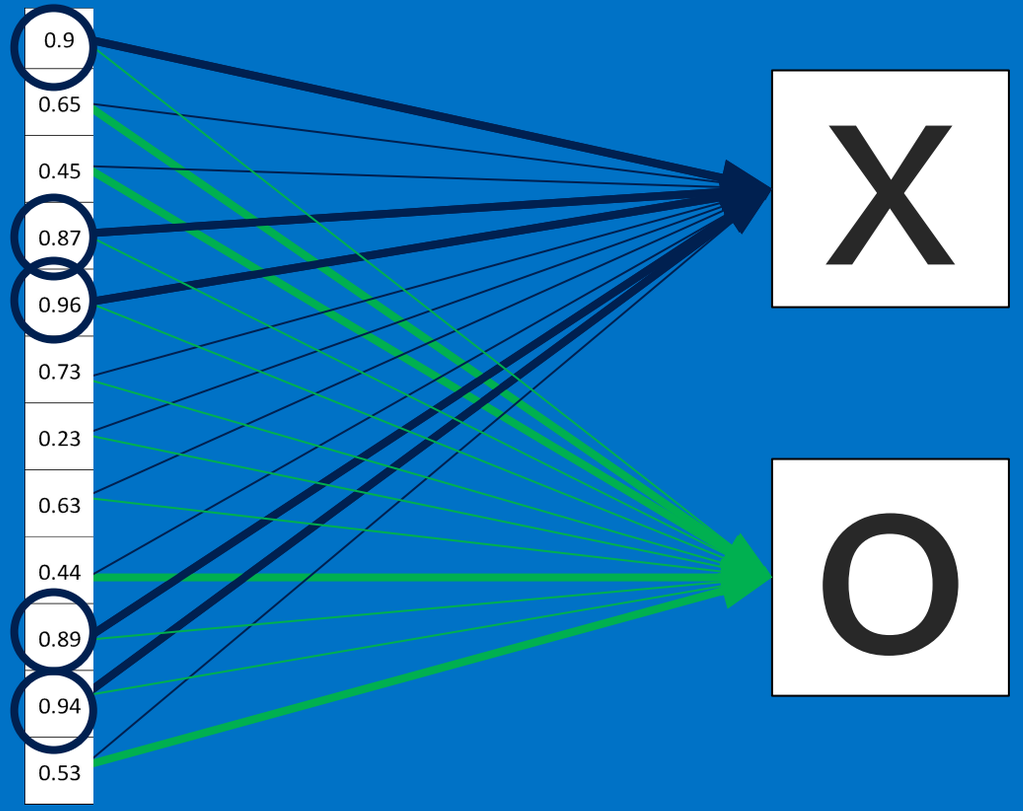
Fully connected layer

Future values vote on X or O



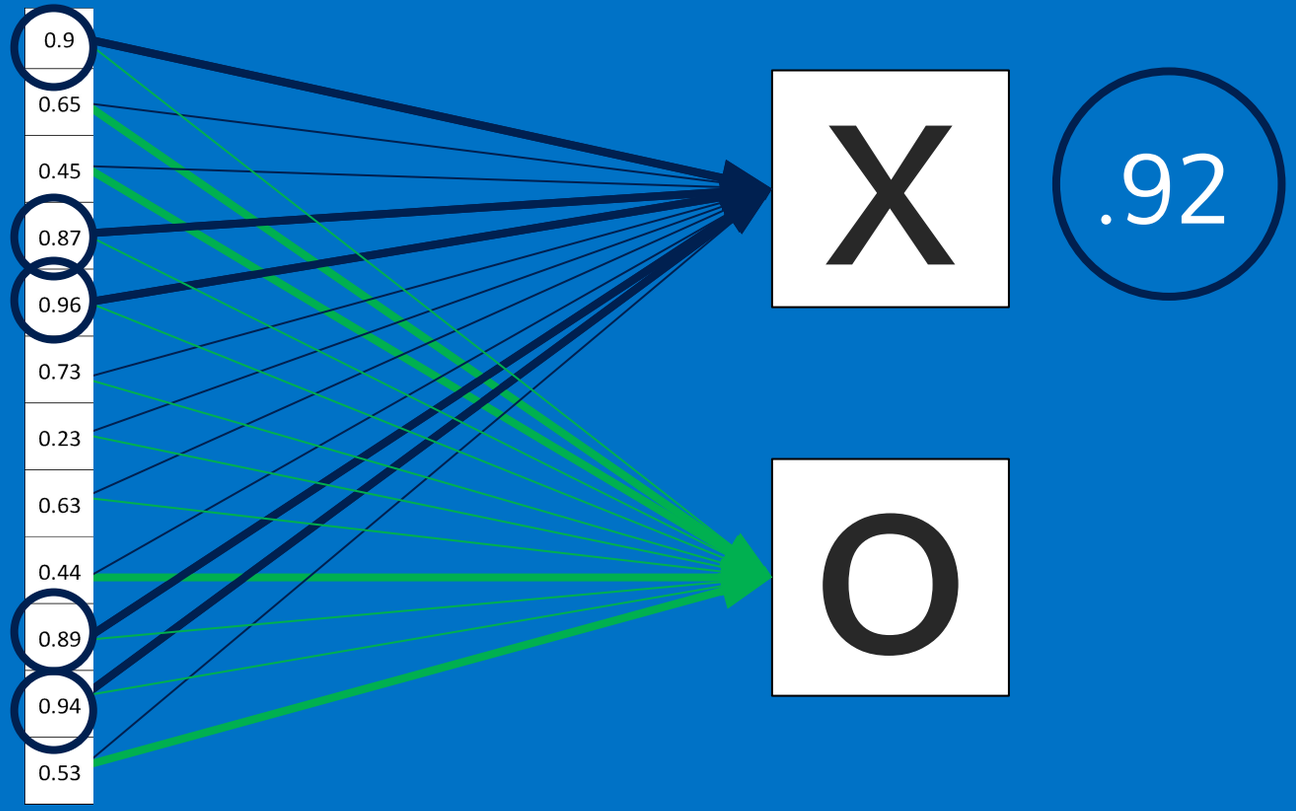
Fully connected layer

Future values vote on X or O



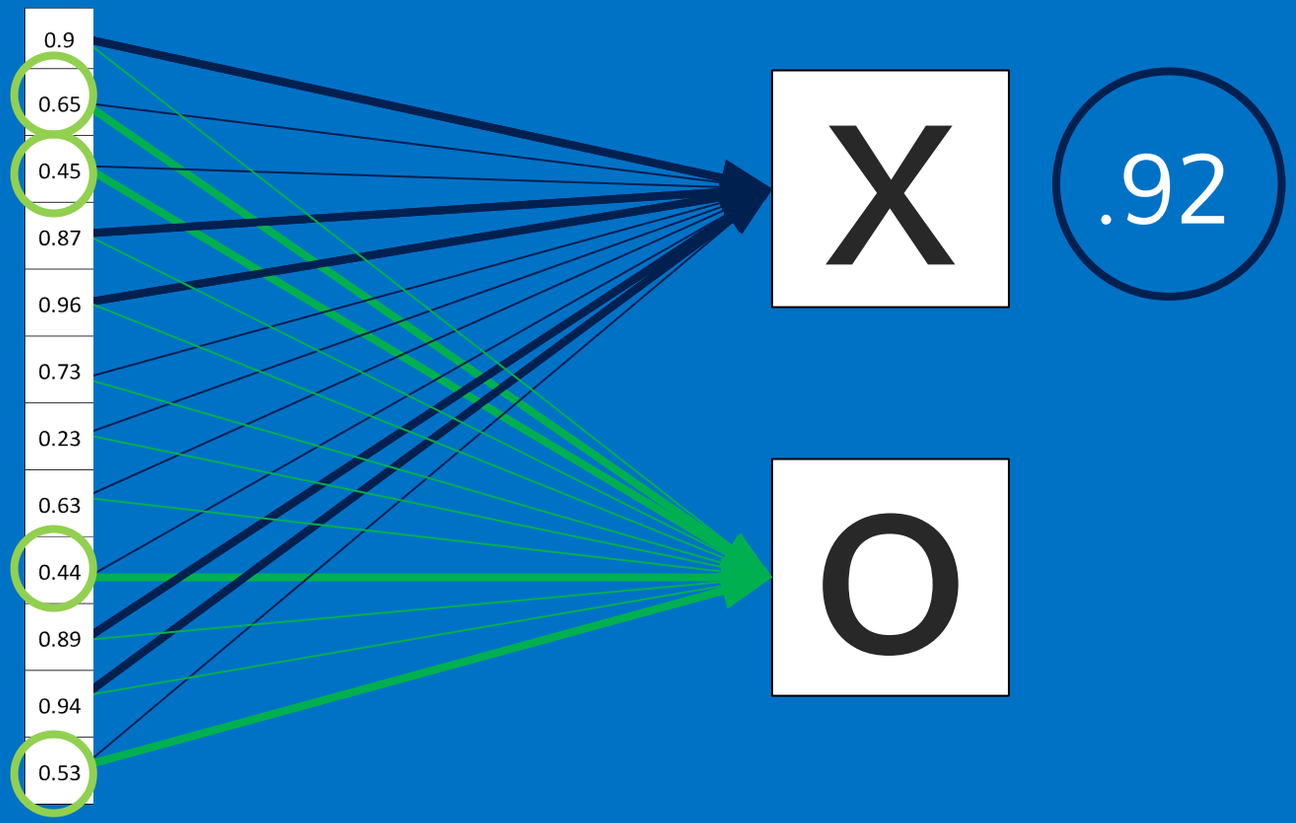
Fully connected layer

Future values vote on X or O



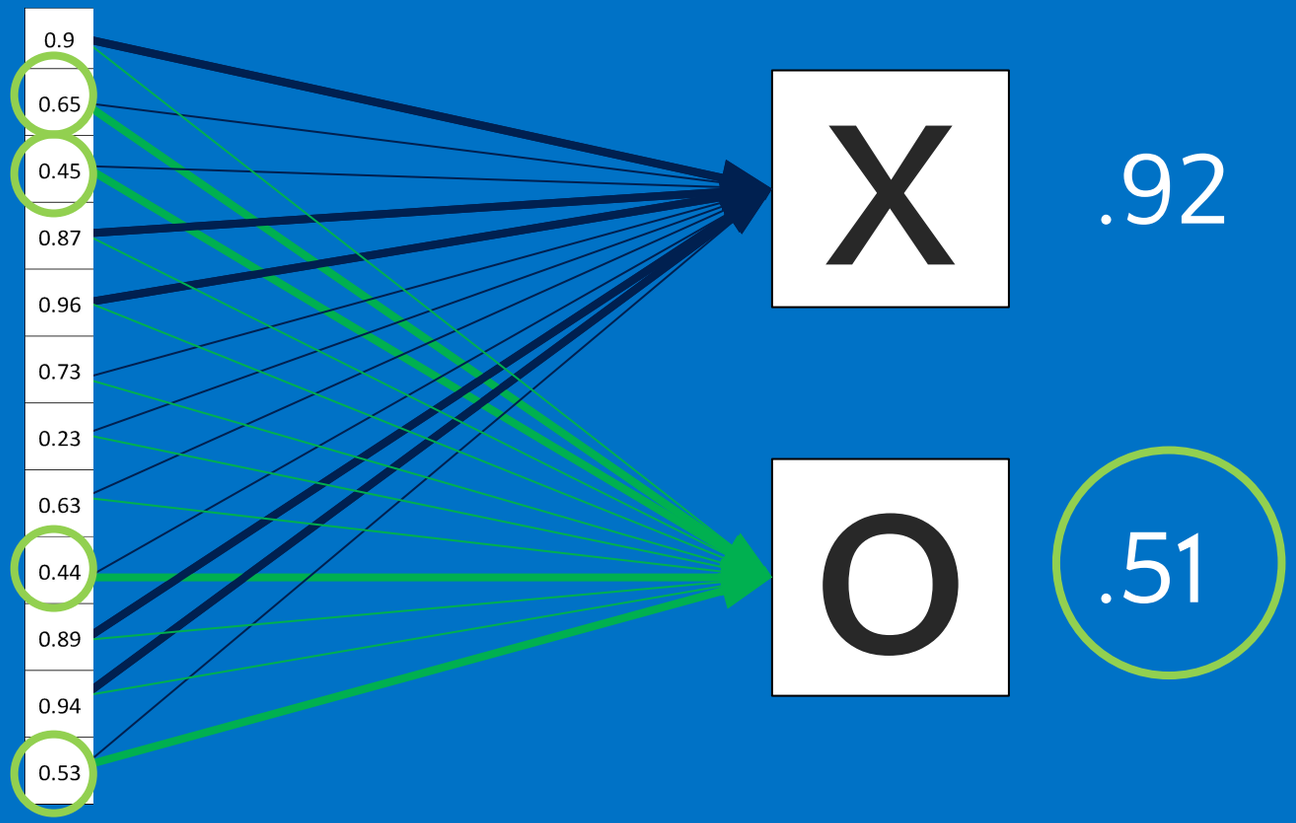
Fully connected layer

Future values vote on X or O



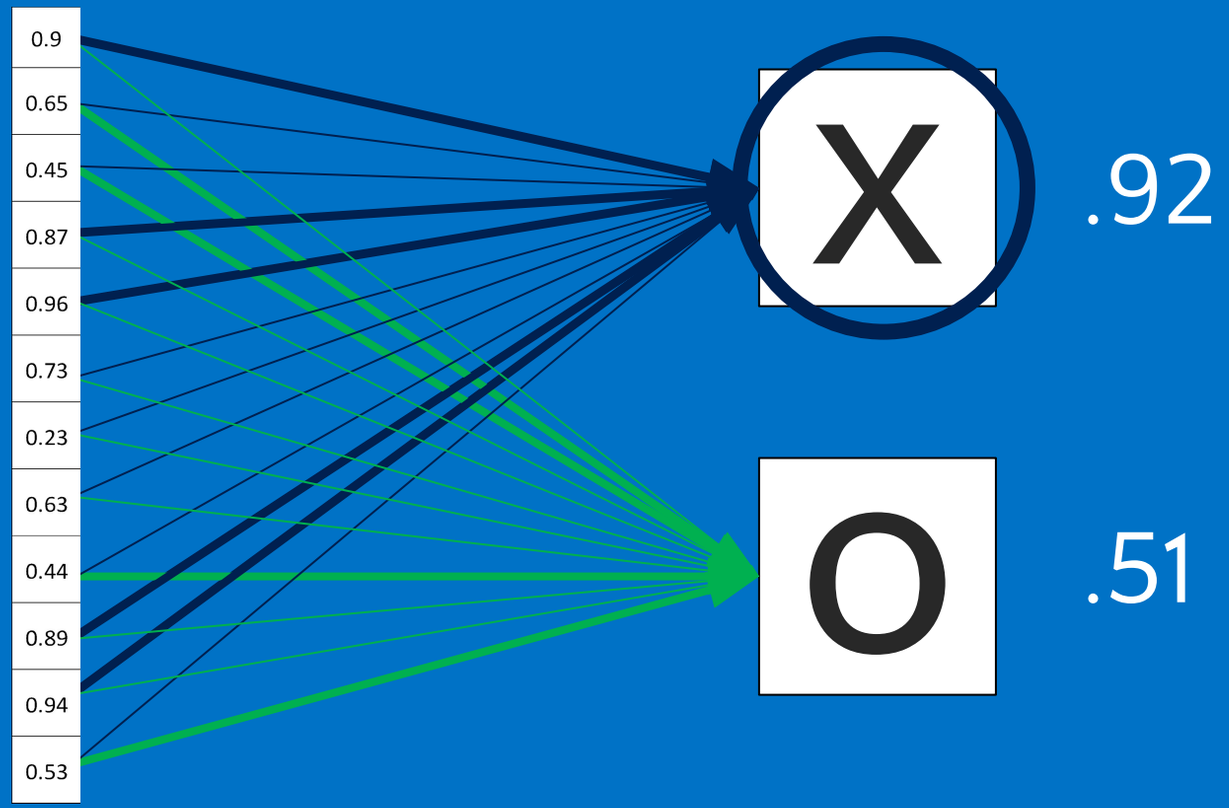
Fully connected layer

Future values vote on X or O



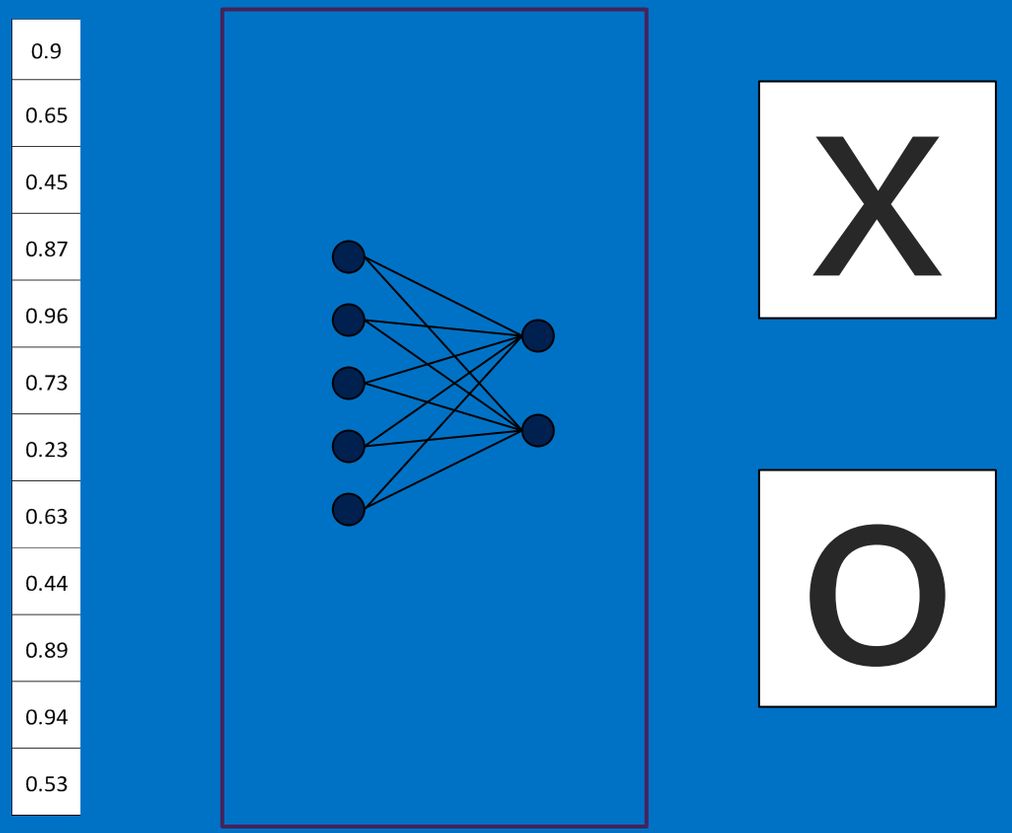
Fully connected layer

Future values vote on X or O



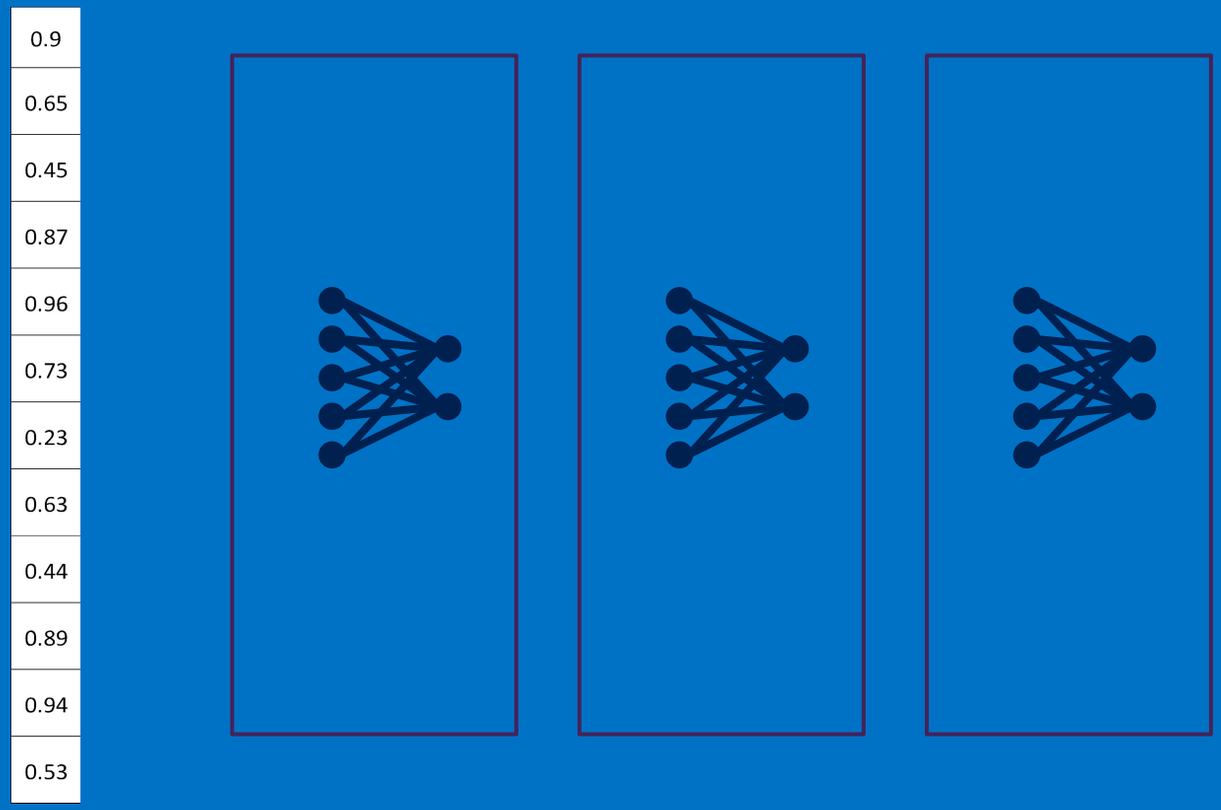
Fully connected layer

A list of feature values becomes a list of votes.



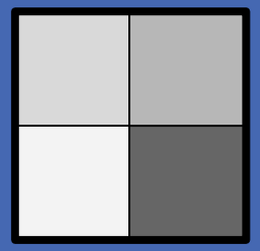
Fully connected layer

These can also be stacked.

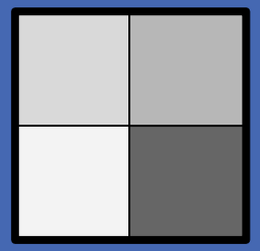


How neural networks work

A four pixel camera



Categorize images



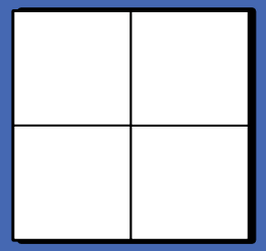
solid

vertical

diagonal

horizontal

Categorize images



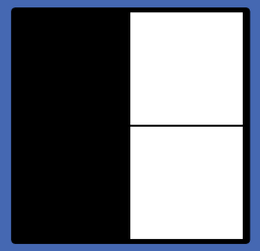
solid

vertical

diagonal

horizontal

Categorize images



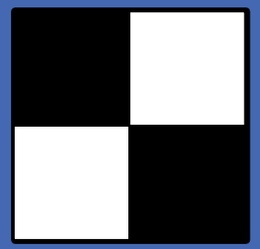
solid

vertical

diagonal

horizontal

Categorize images



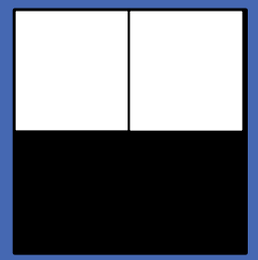
solid

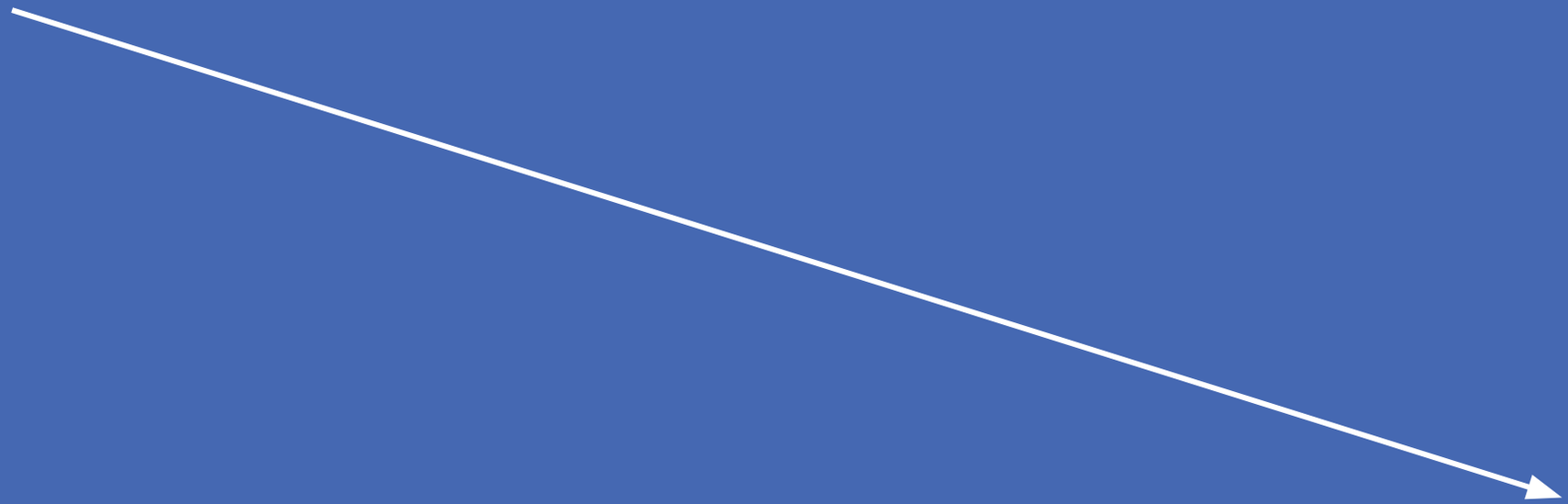
vertical

diagonal

horizontal

Simple rules can’t do it





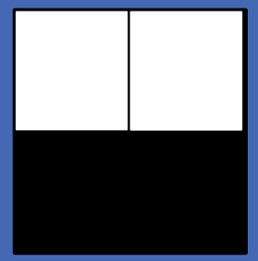
solid

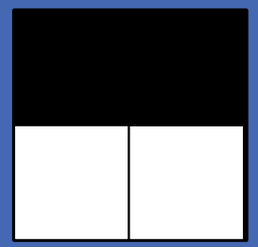
vertical

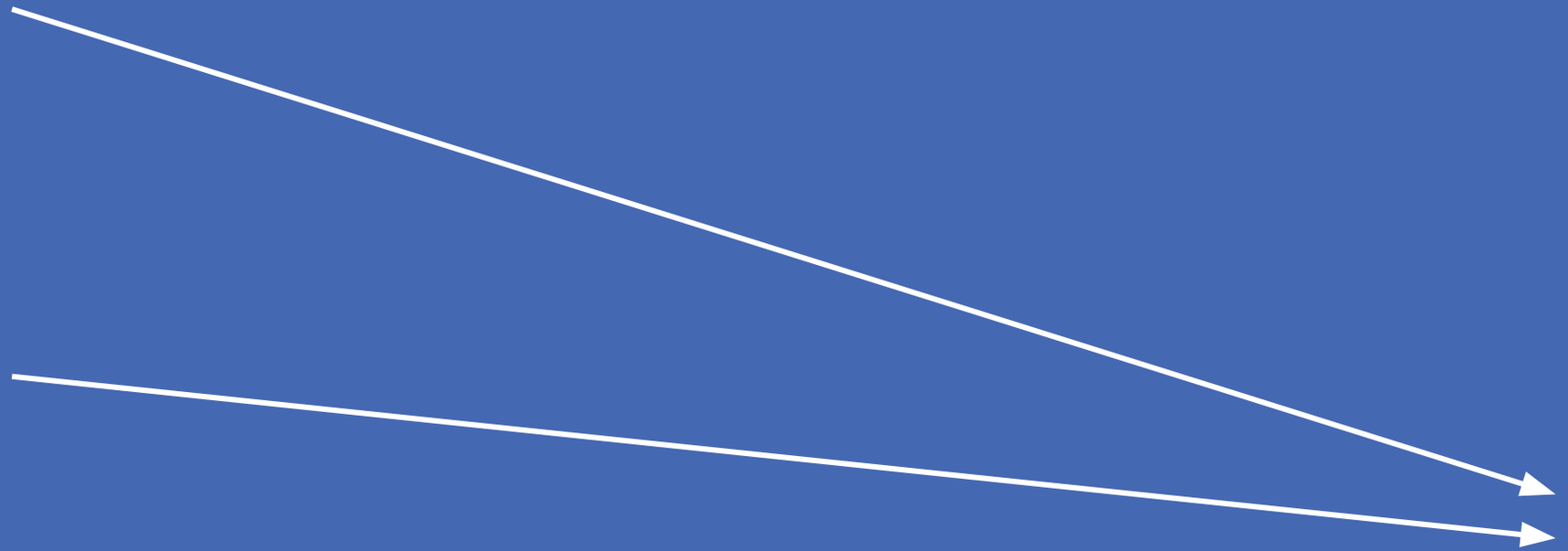
diagonal

horizontal

Simple rules can’t do it







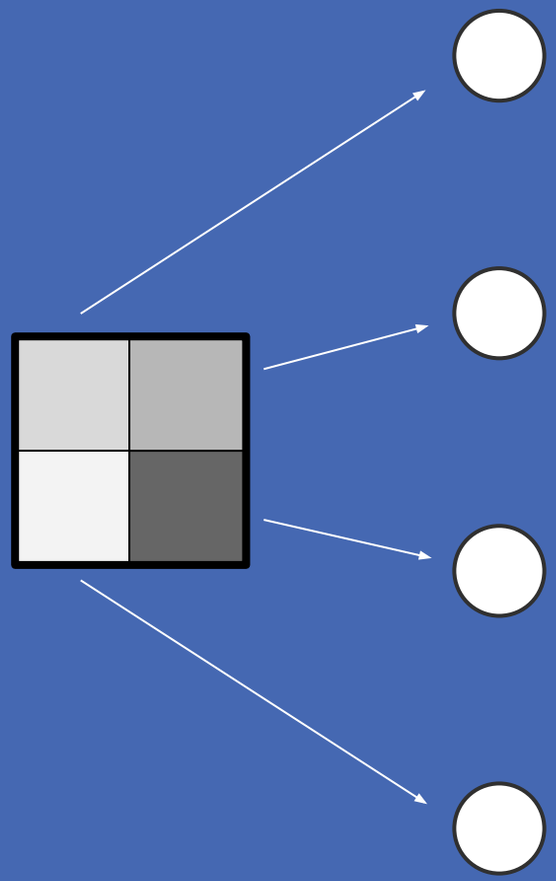
solid

vertical

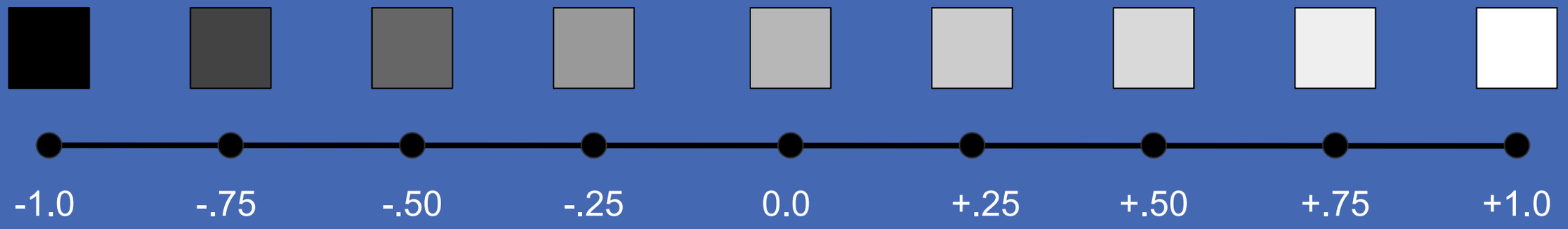
diagonal

horizontal

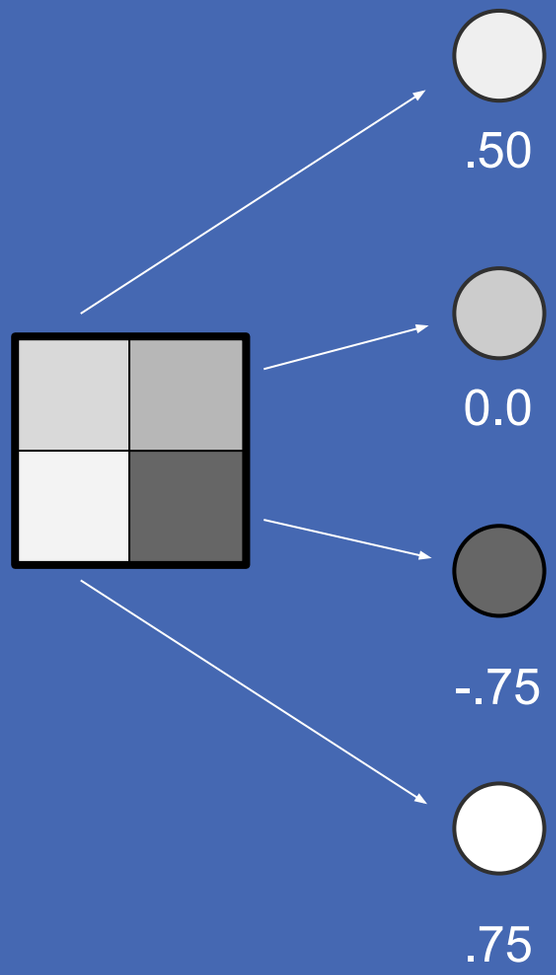
Input neurons



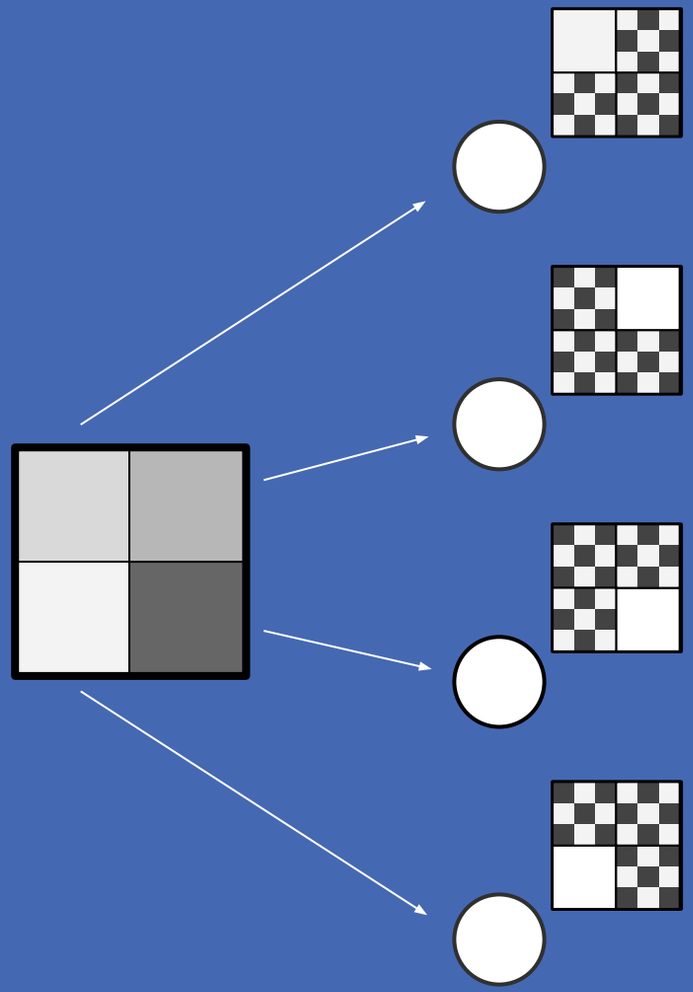
Pixel brightness



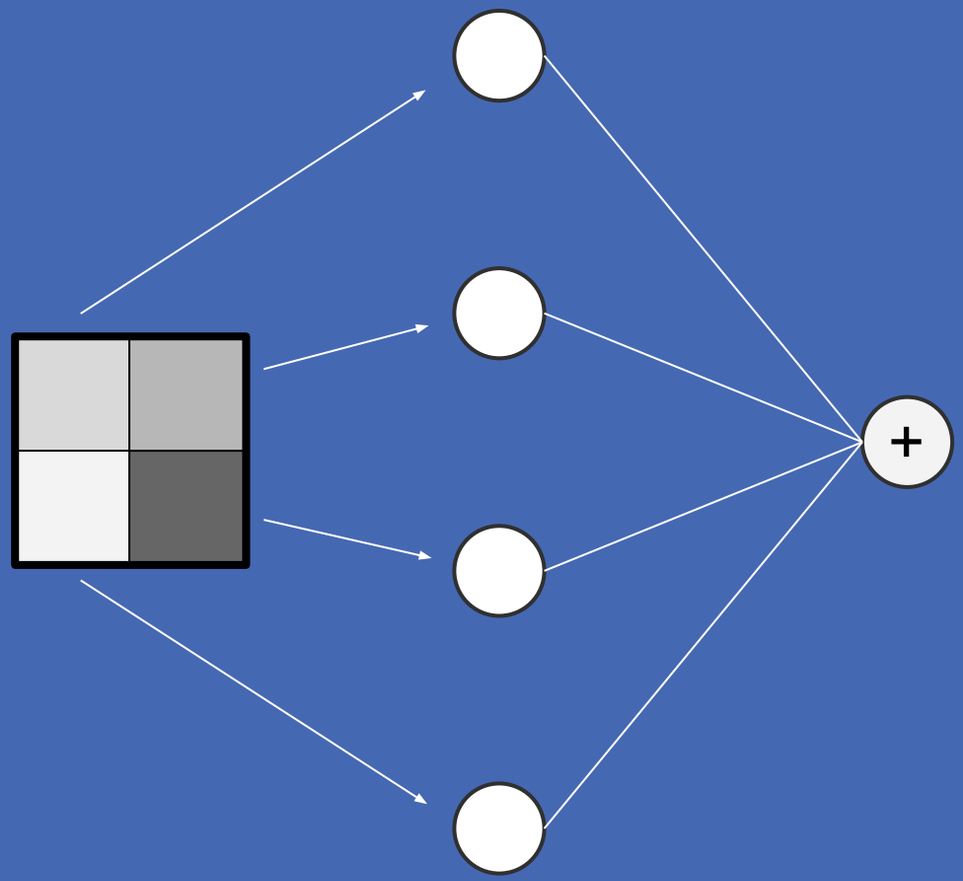
Input vector



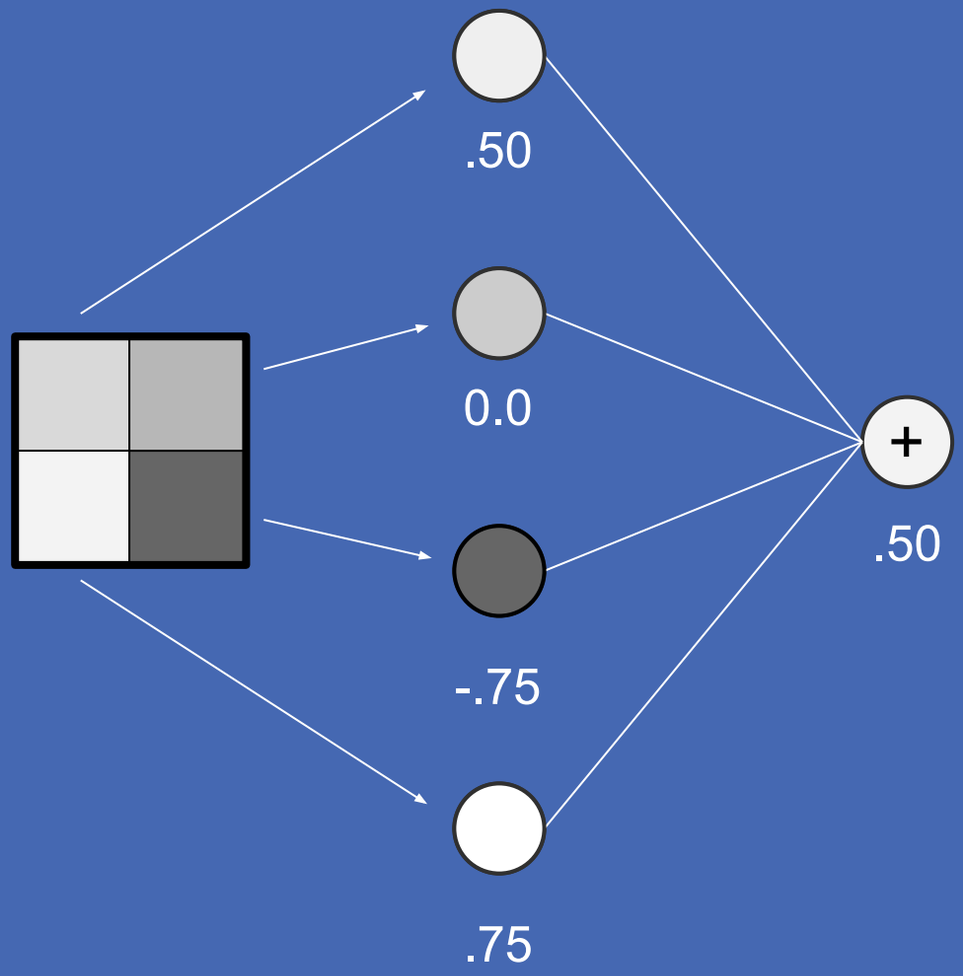
Receptive fields



A neuron

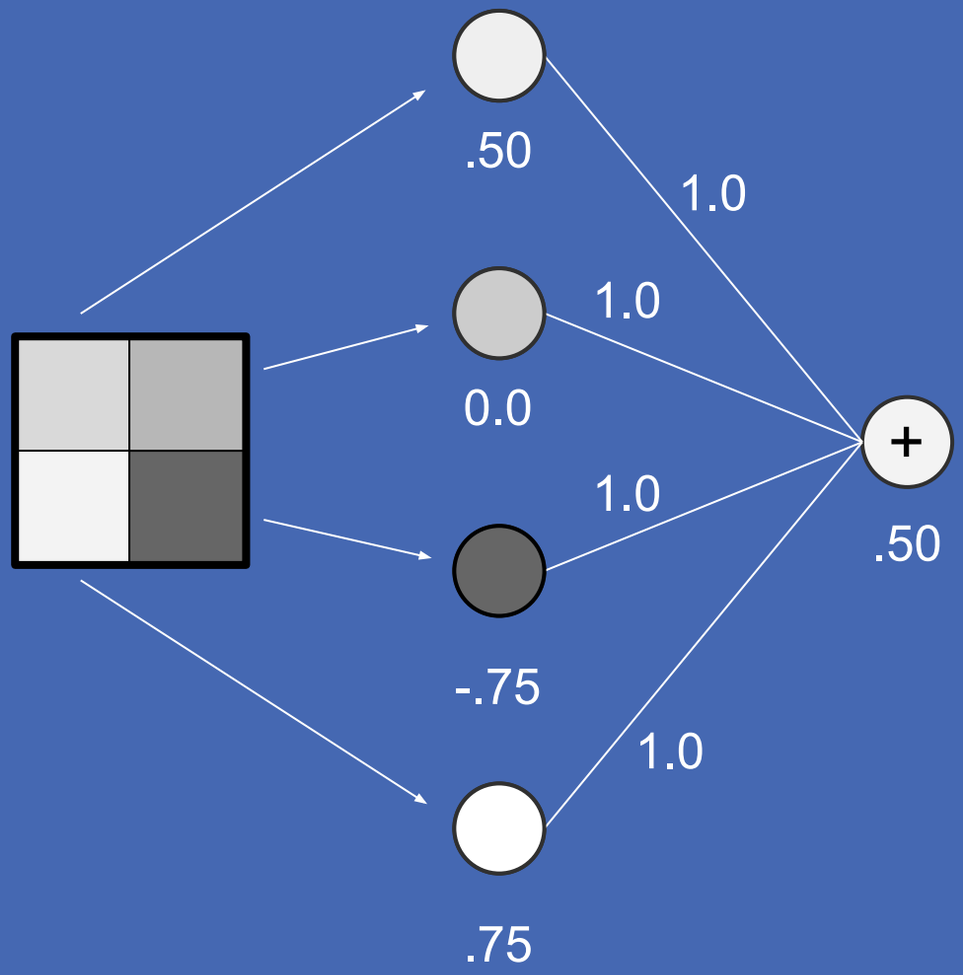


Sum all the inputs



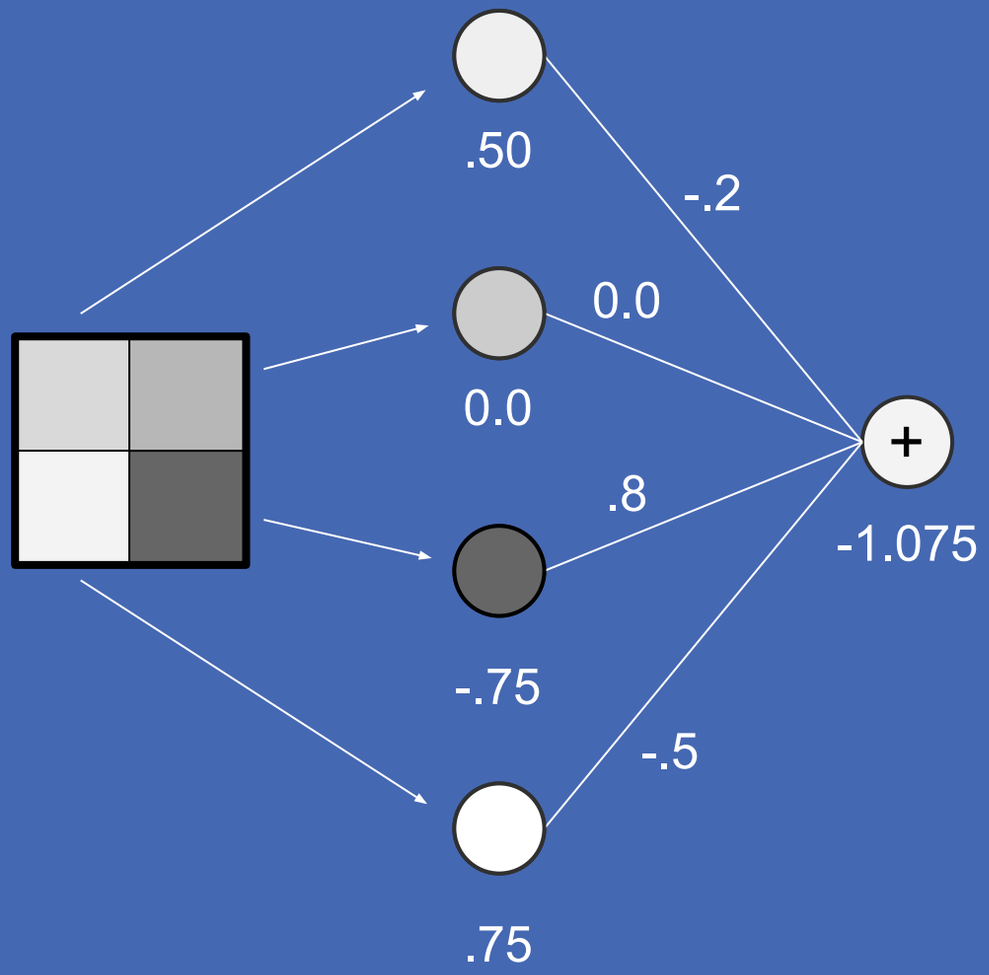

```
   .50
  0.00
  -.75
+  .75
------
   .50
```

Weights



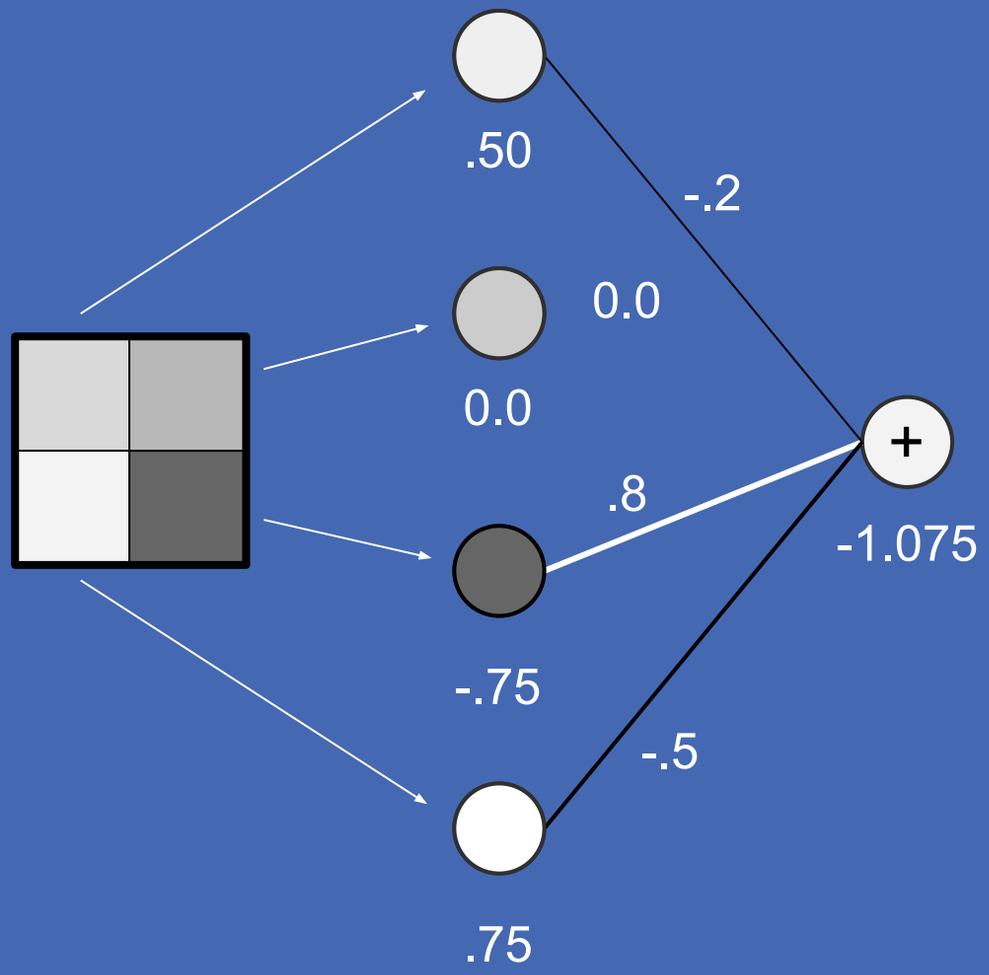
	.50	x 1.0
	0.00	x 1.0
	-.75	x 1.0
+	.75	x 1.0
	.50	

Weights



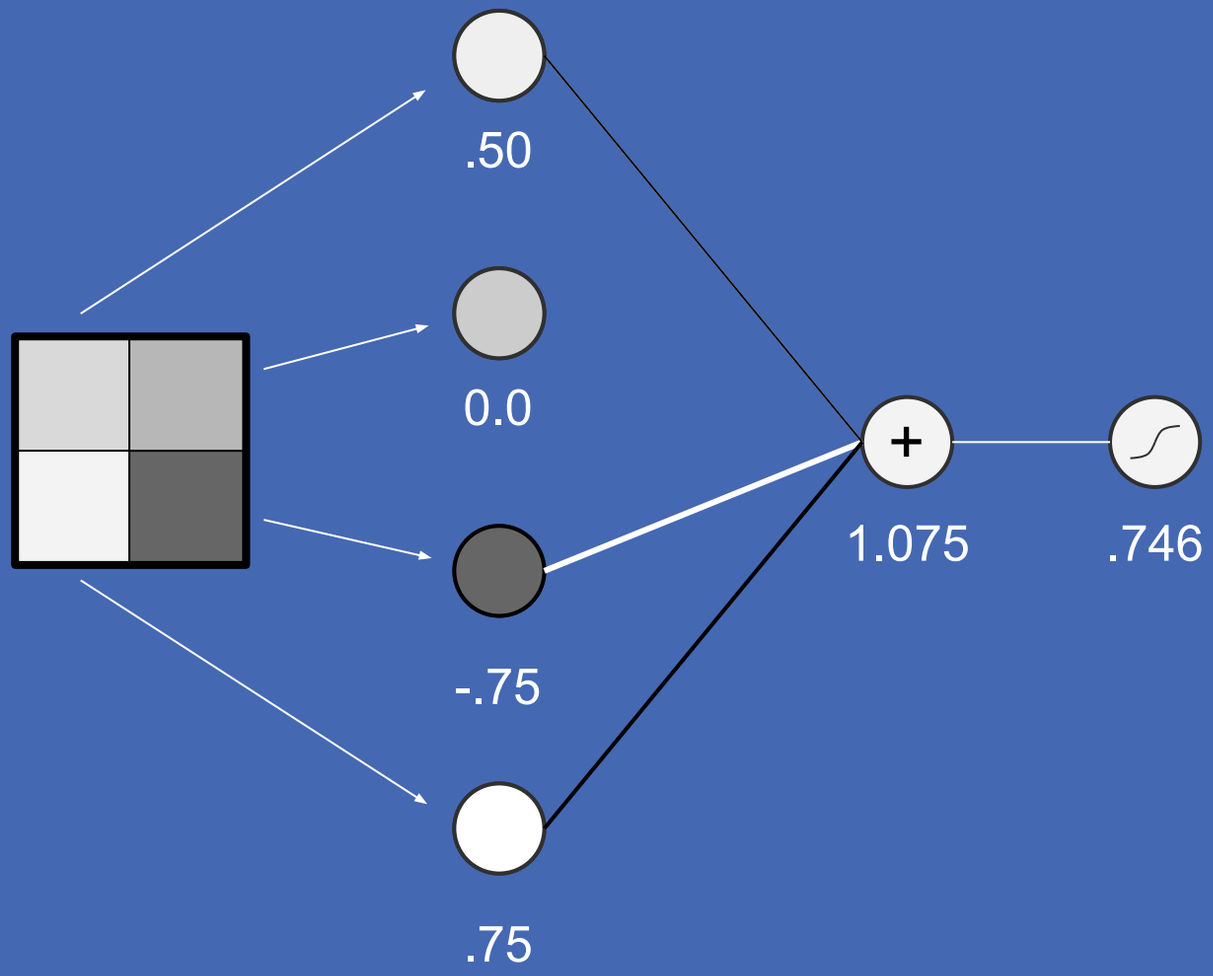
	.50	x	-.2
	0.00	x	0.0
	-.75	x	.8
+	.75	x	-.5
	-1.075		

Weights

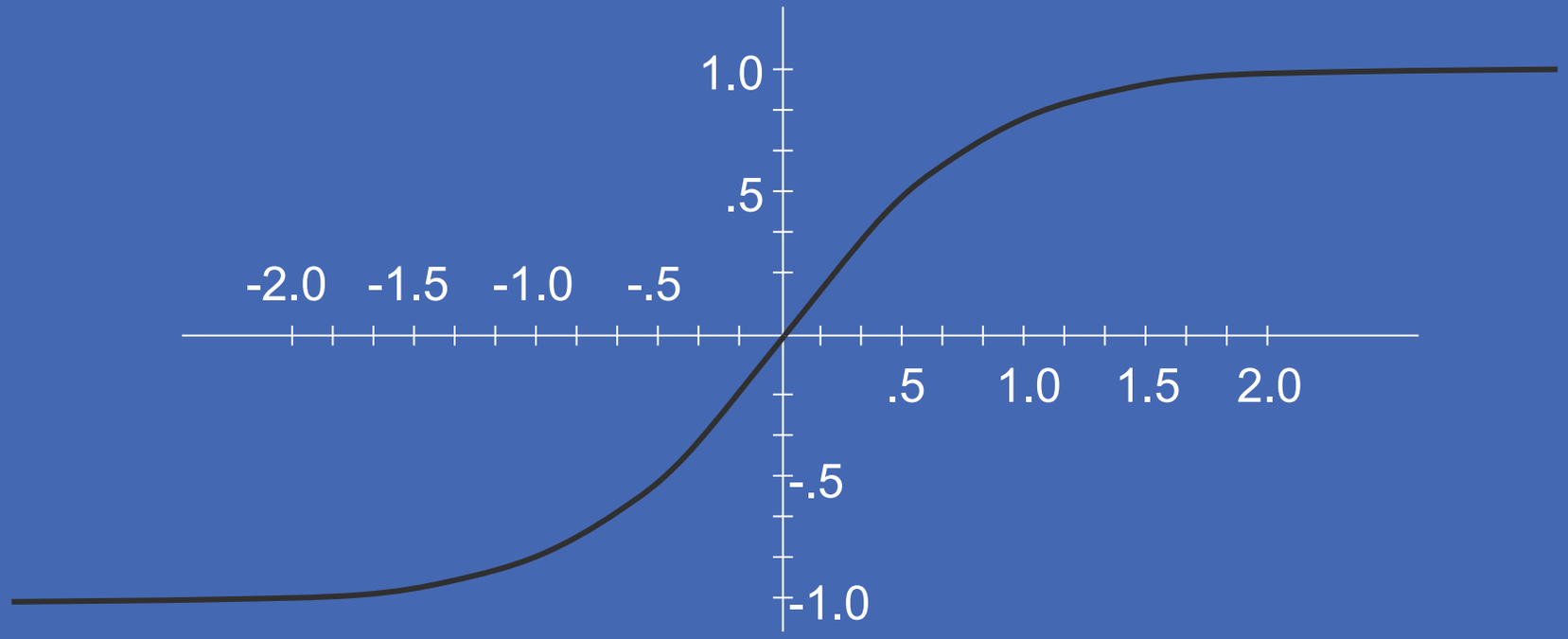


	.50	x	-.2
	0.00	x	0.0
	-.75	x	.8
+	.75	x	-.5
	-1.075		

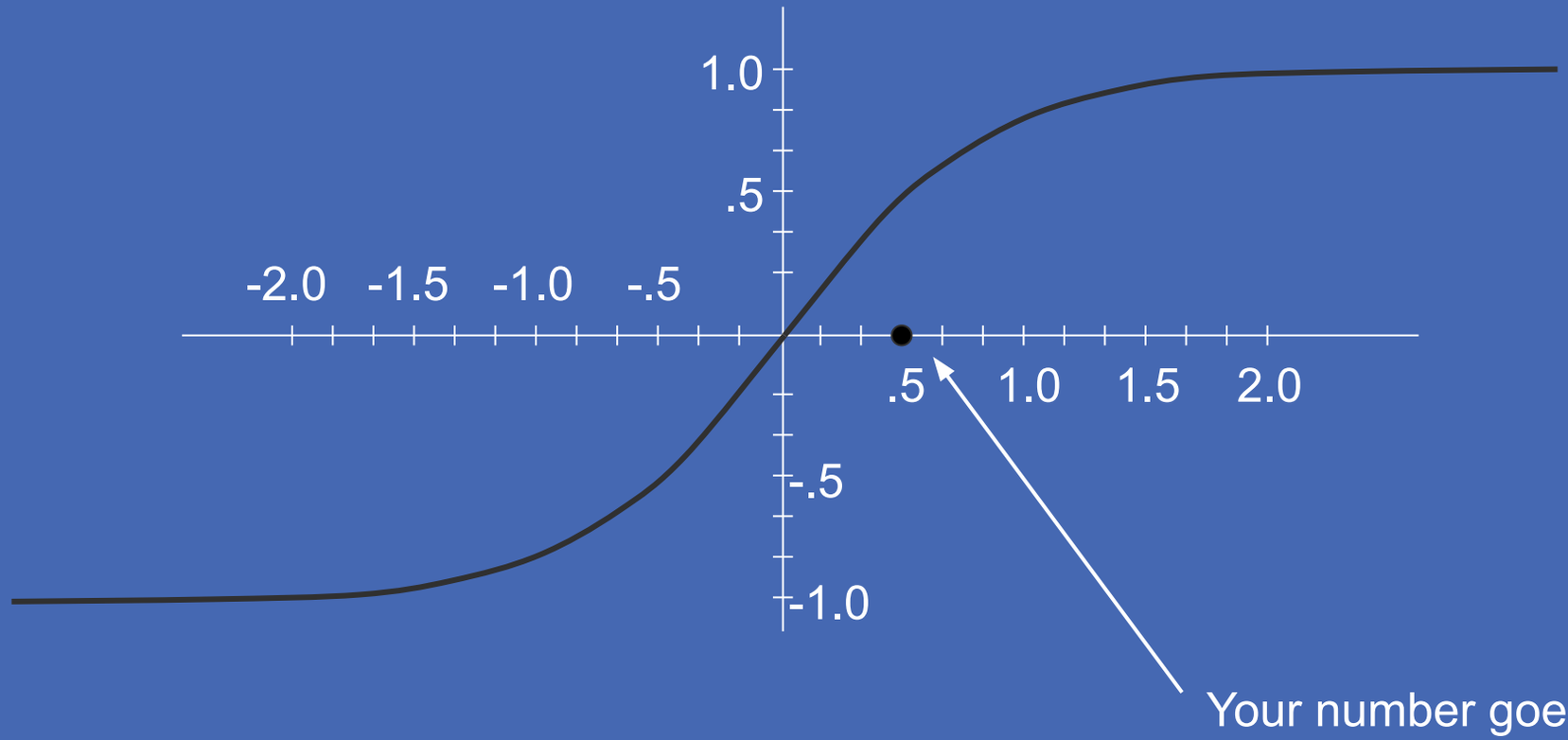
Squash the result



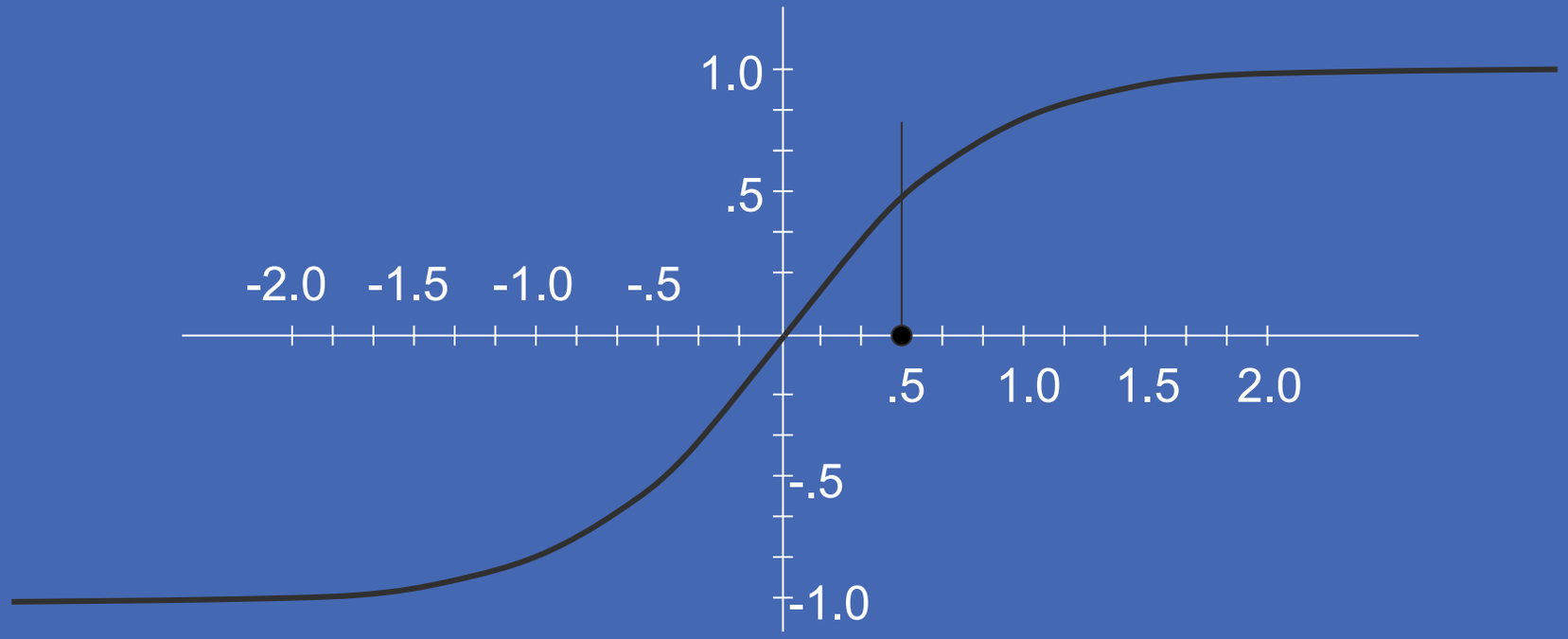
Sigmoid squashing function



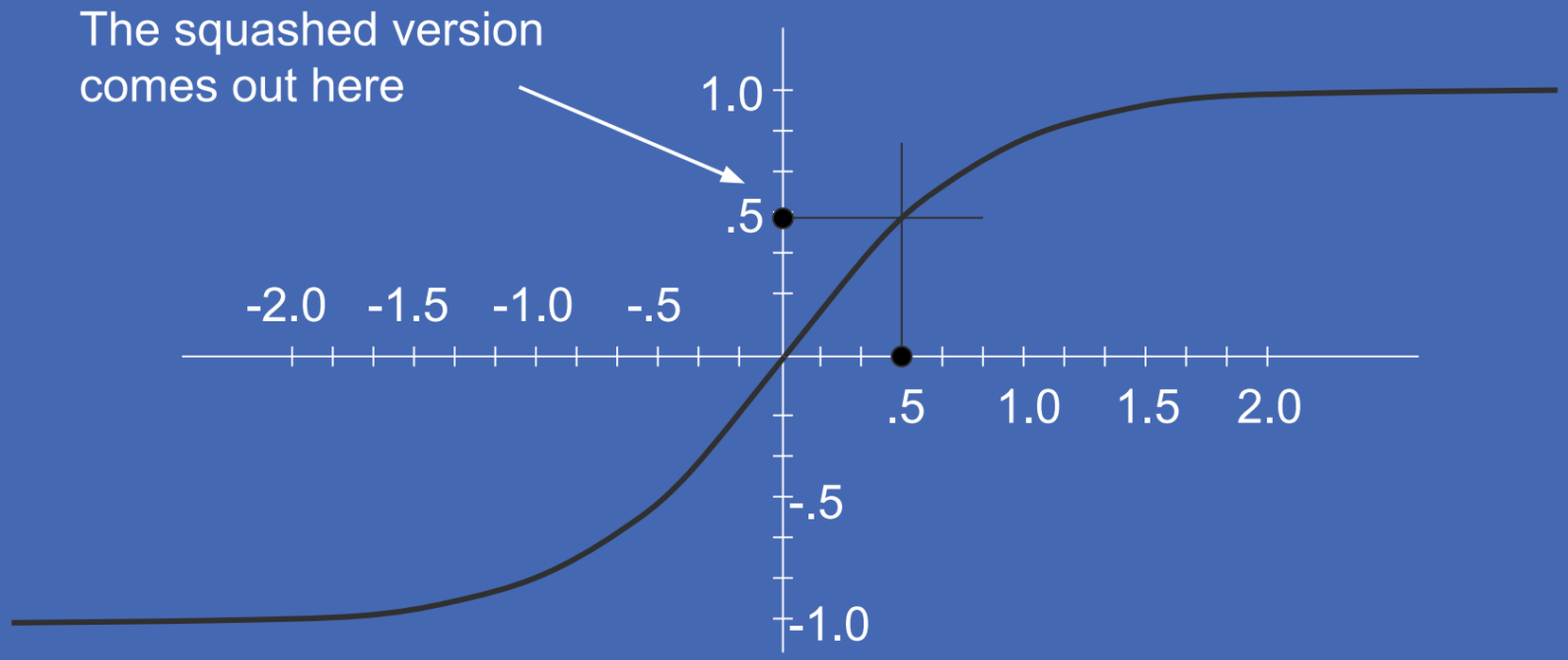
Sigmoid squashing function



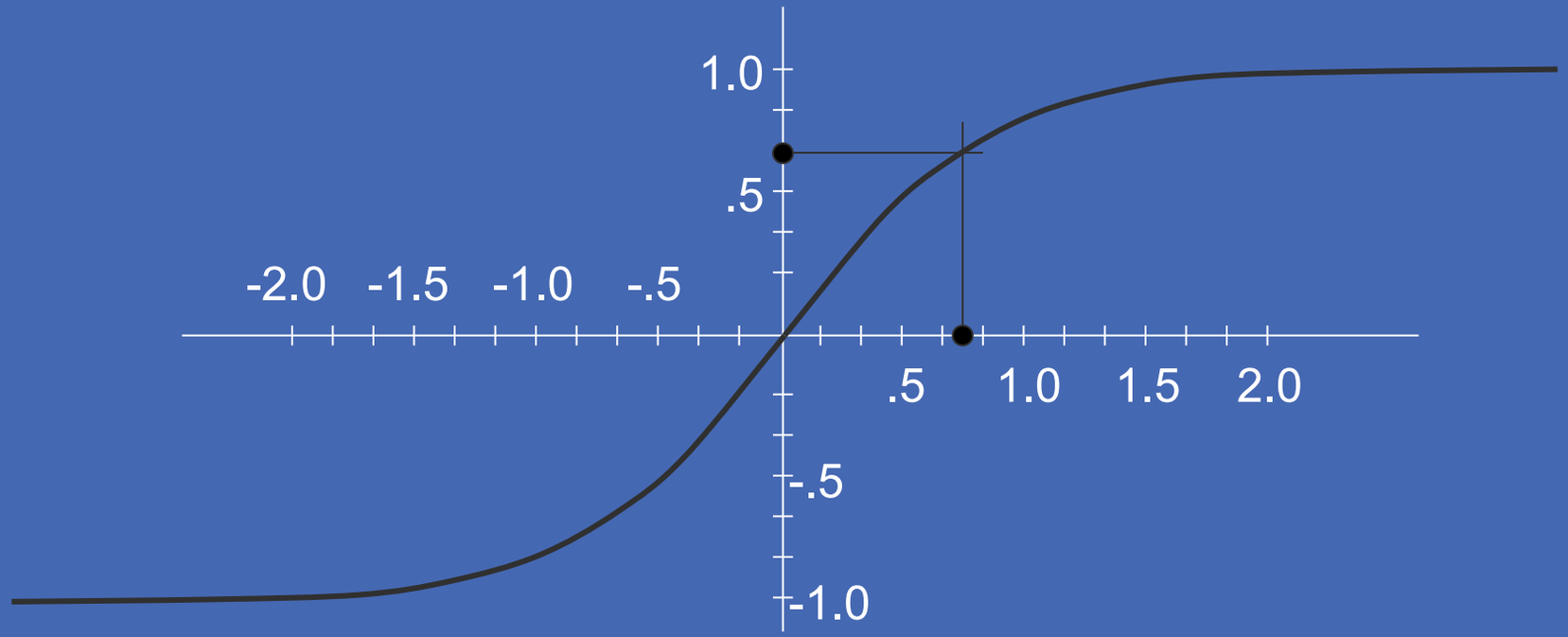
Sigmoid squashing function



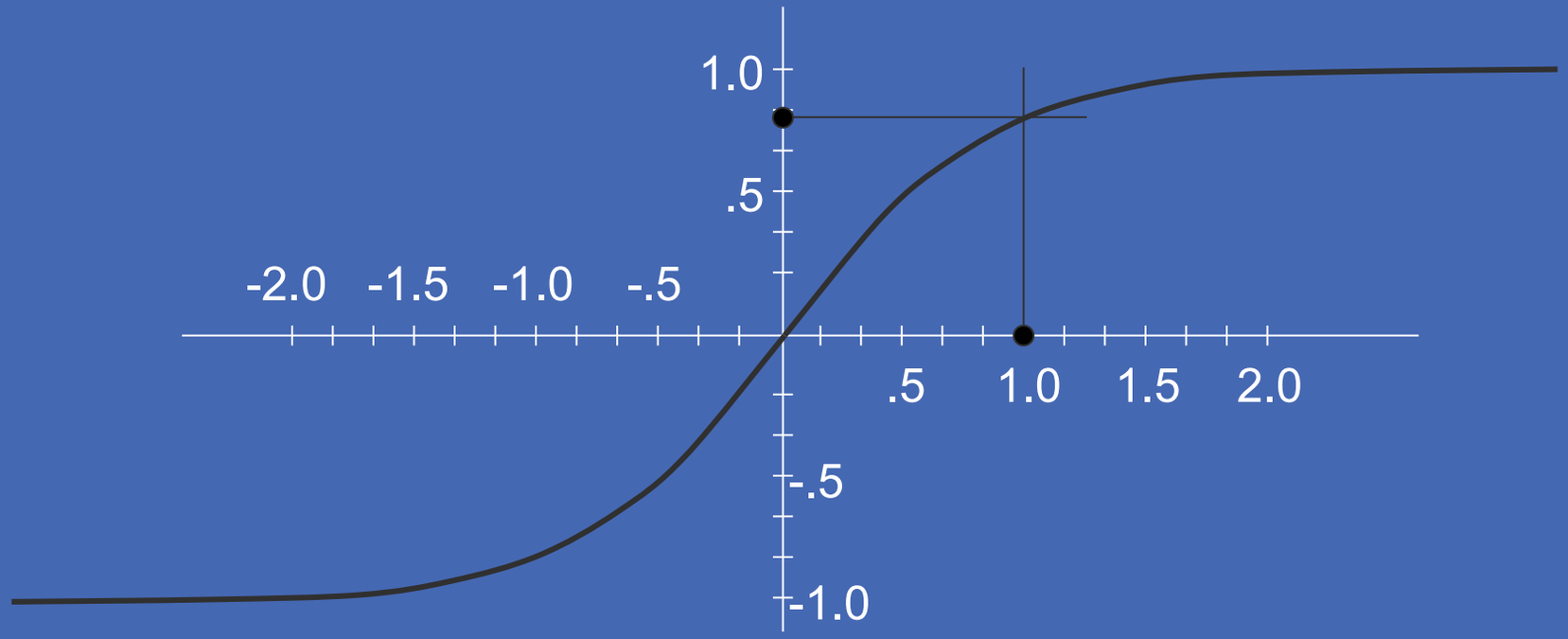
Sigmoid squashing function



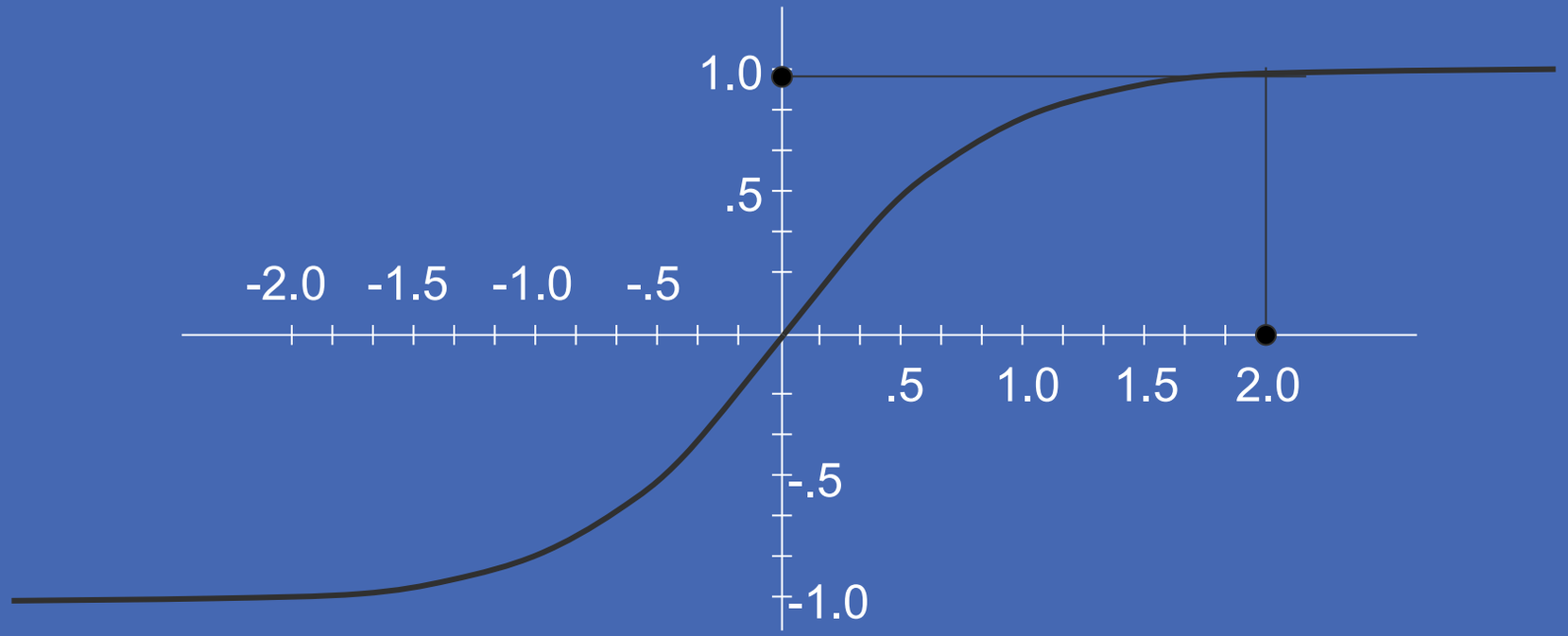
Sigmoid squashing function



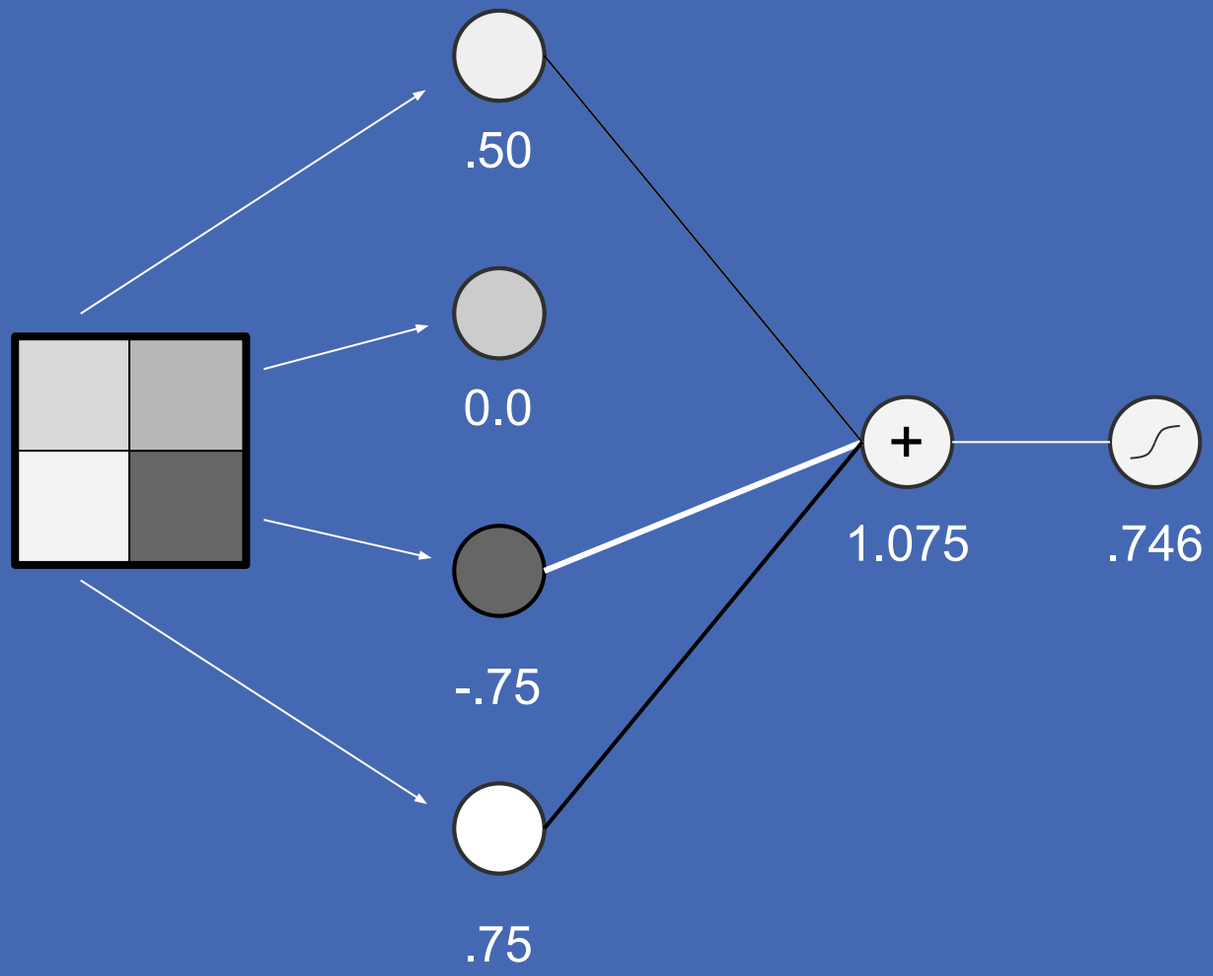
Sigmoid squashing function



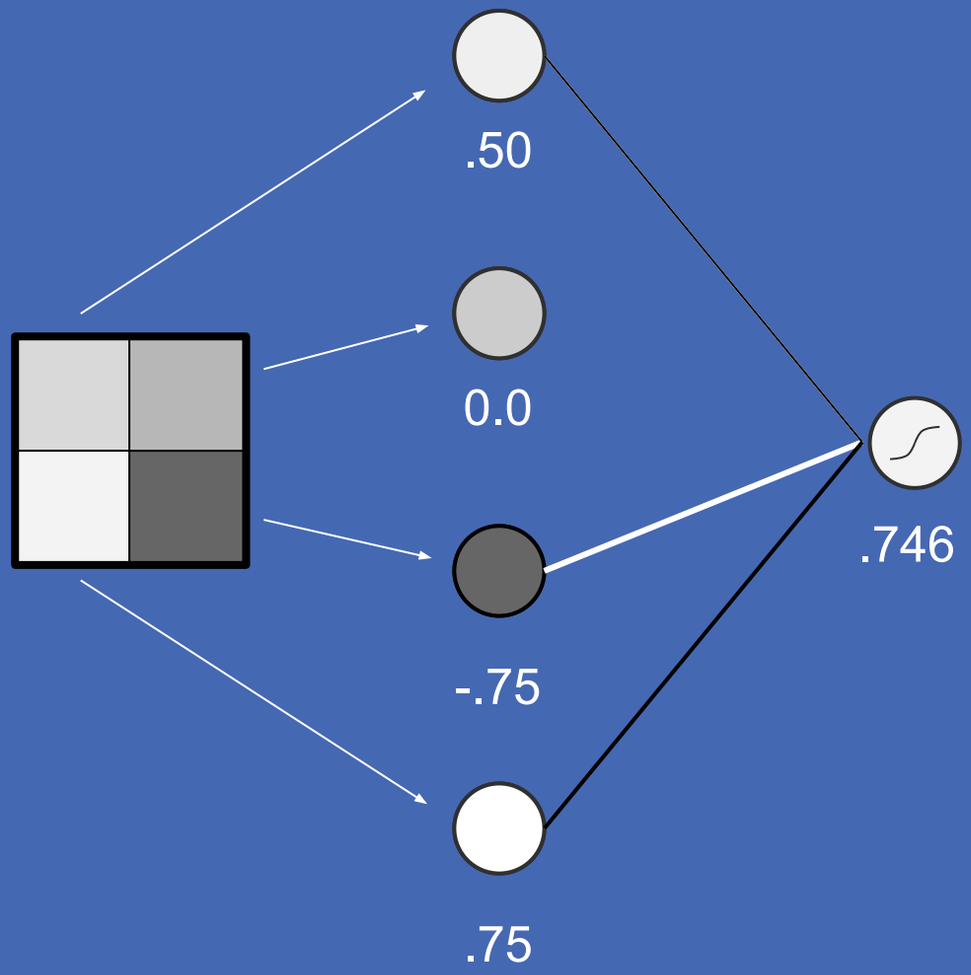
No matter what you start with, the answer stays between -1 and 1.



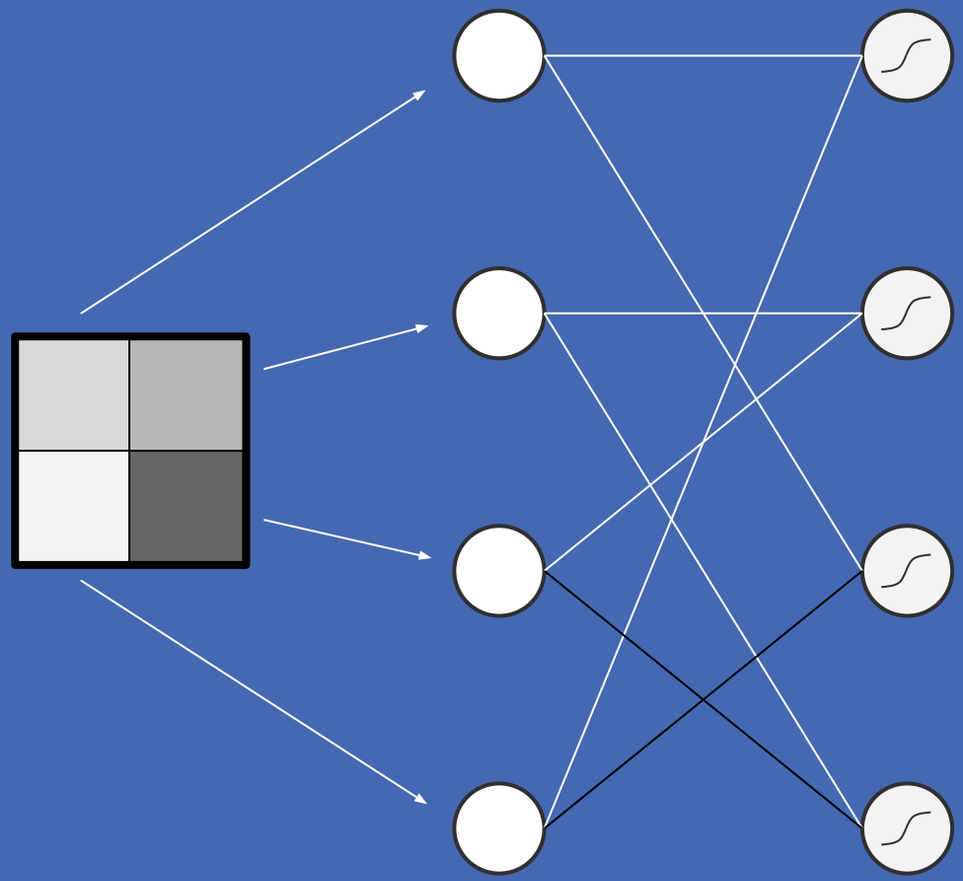
Squash the result



Weighted sum-and-squash neuron

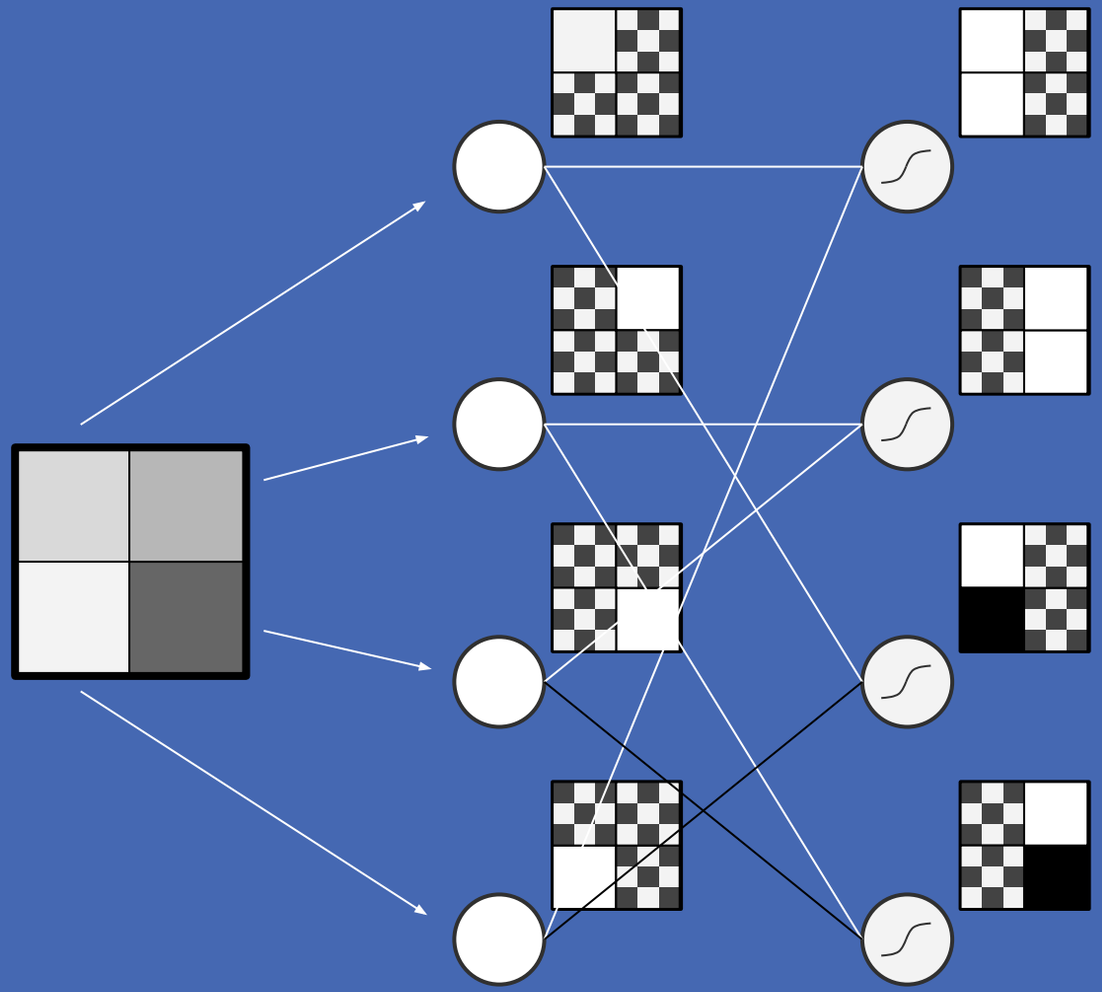


Make lots of neurons, identical except for weights

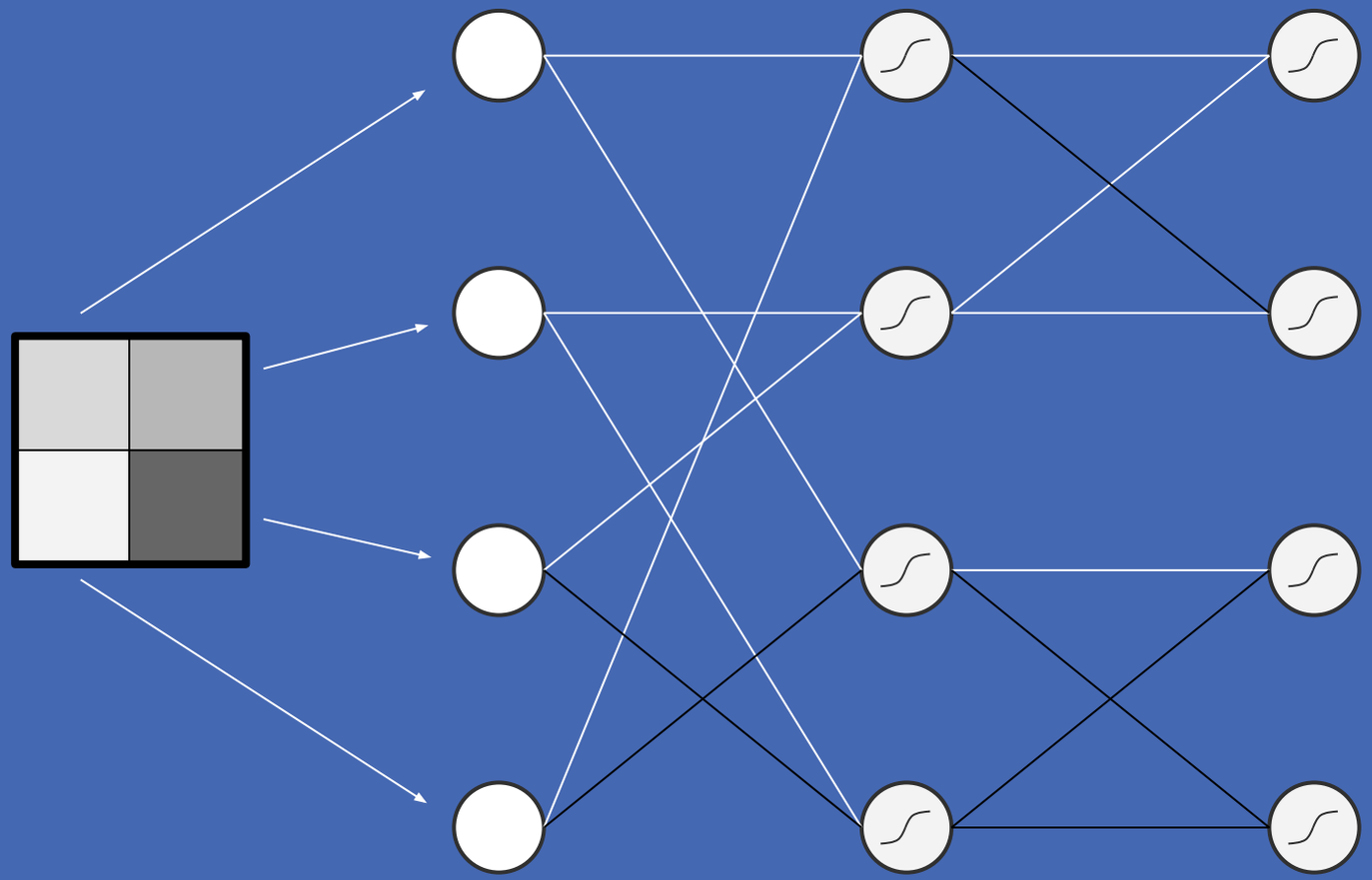


To keep our picture clear,
weights will either be
1.0 (white)
-1.0 (black) or
0.0 (missing)

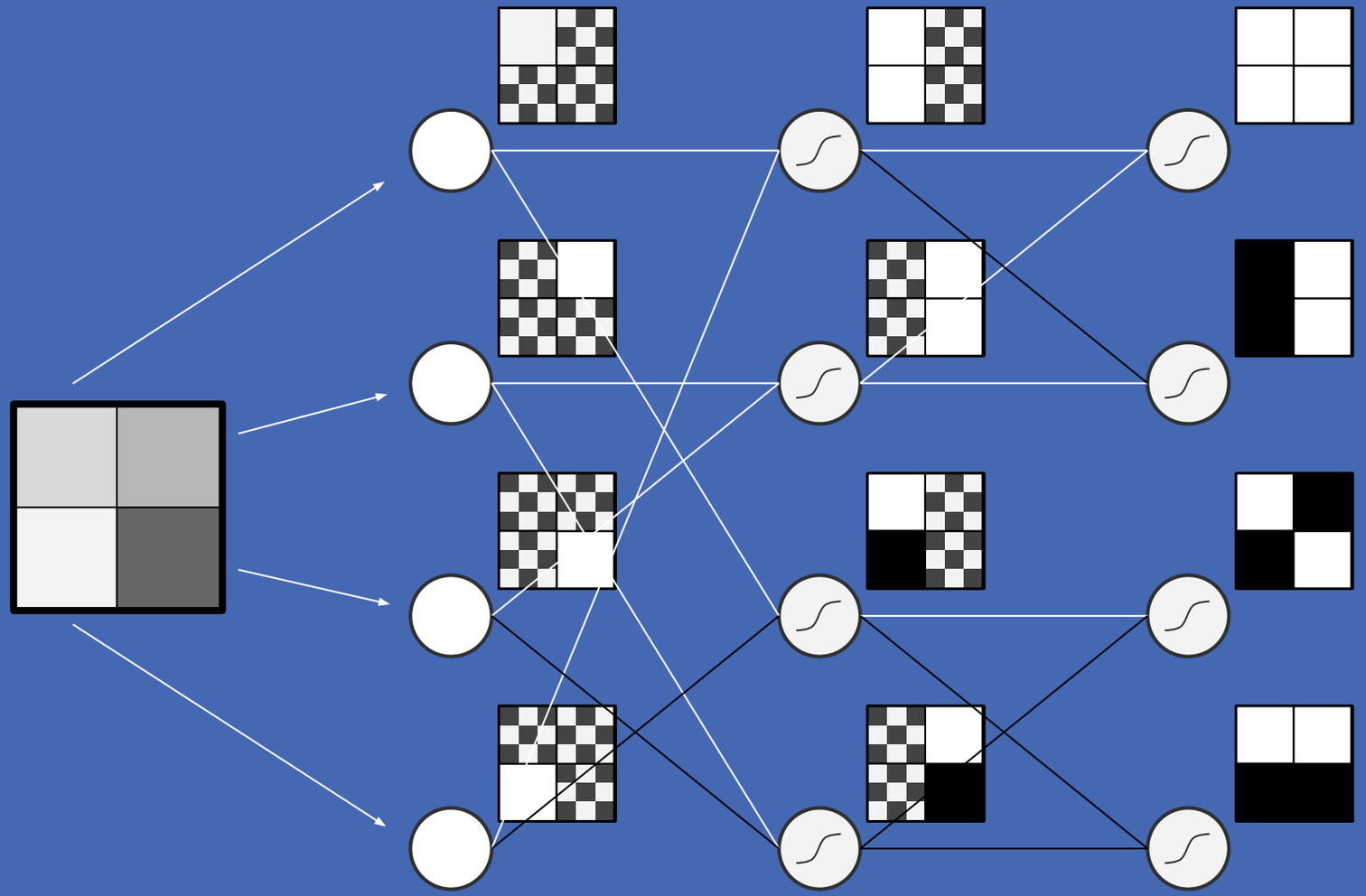
Receptive fields get more complex



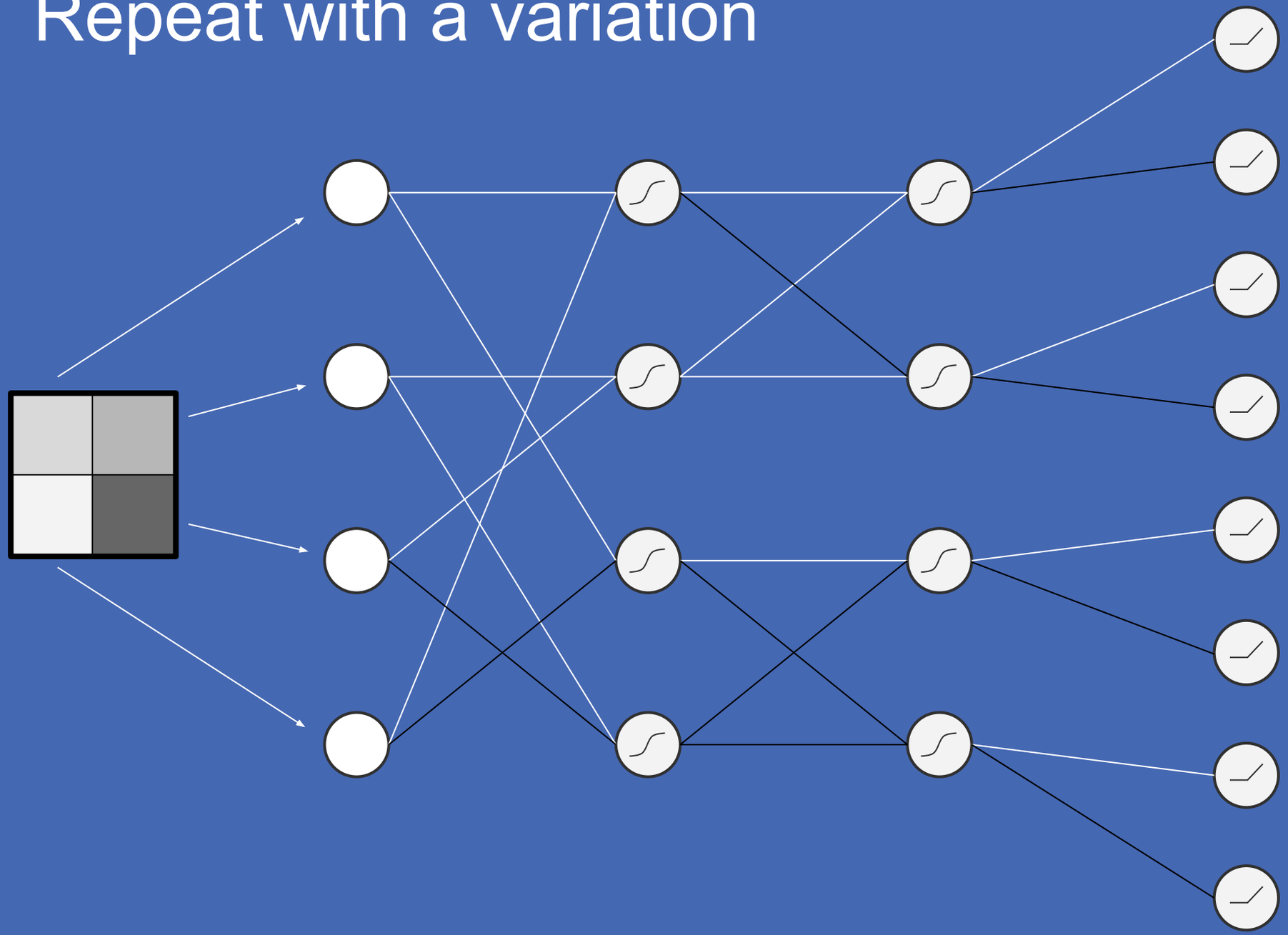
Repeat for additional layers



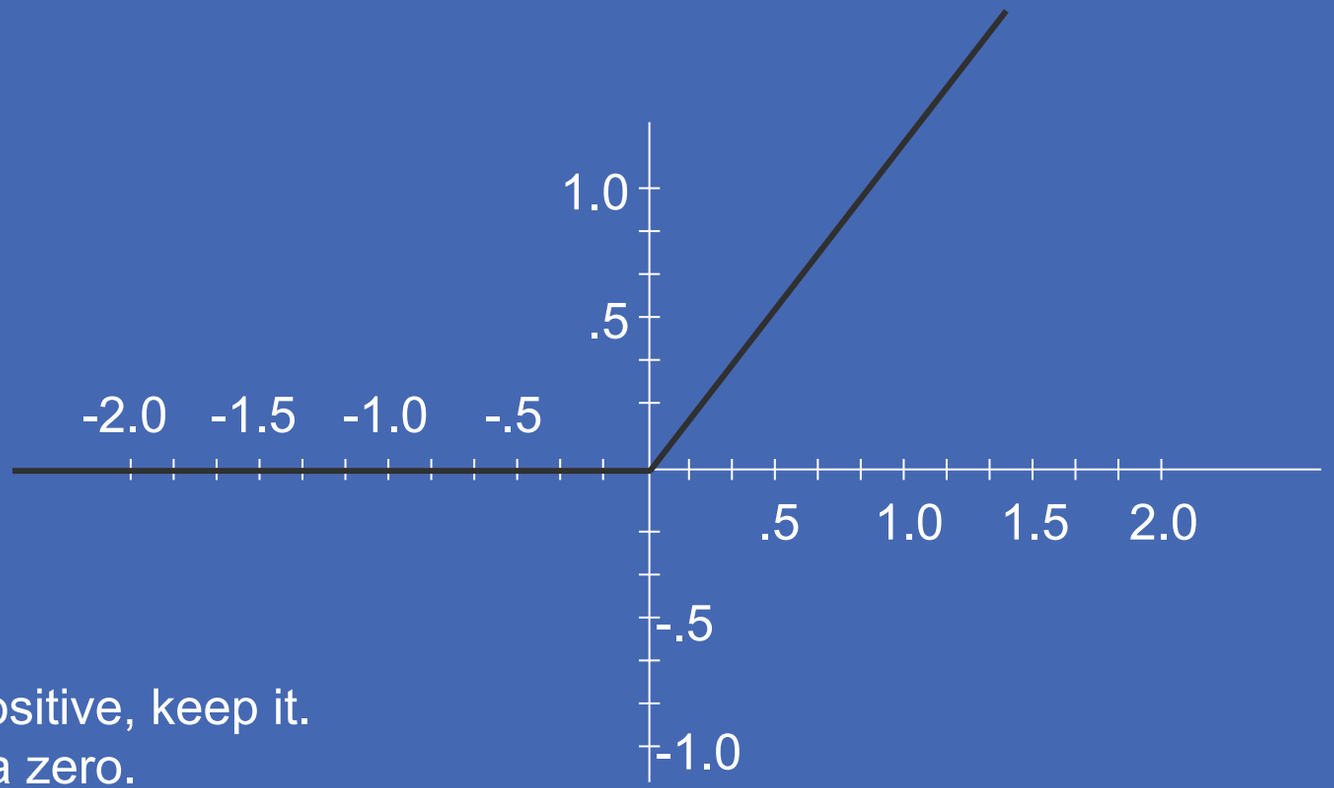
Receptive fields get still more complex



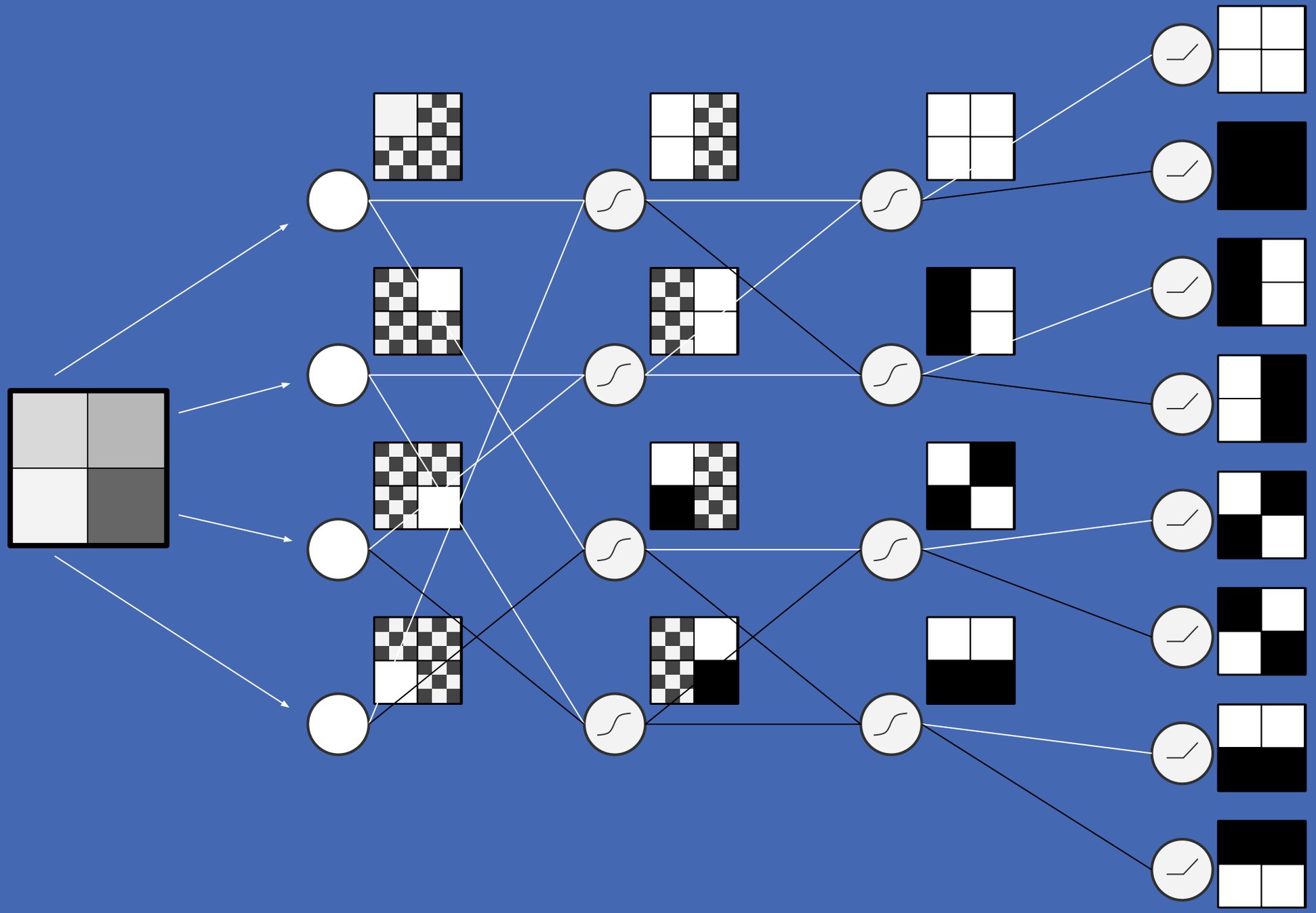
Repeat with a variation



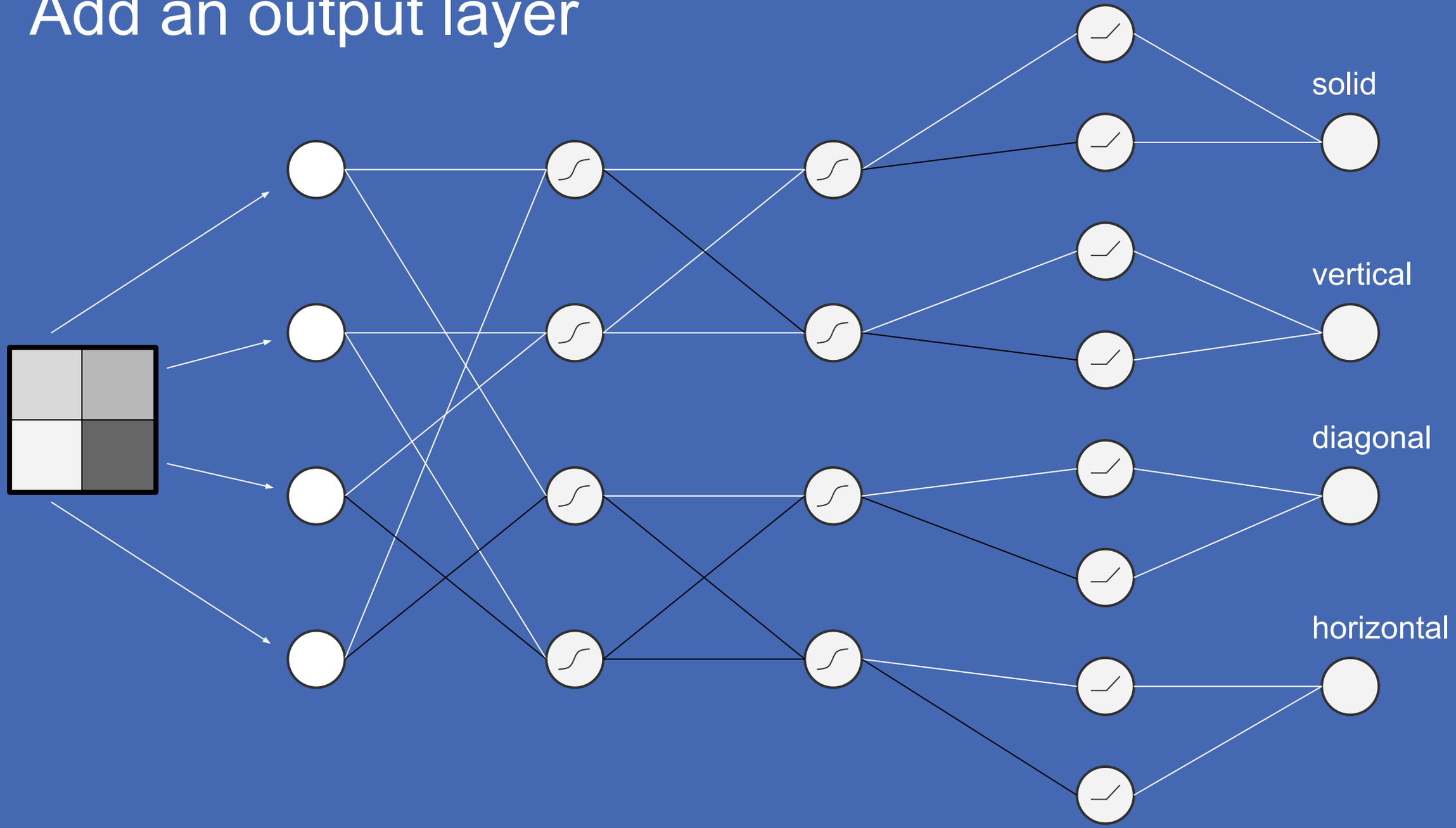
Rectified linear units (ReLUs)

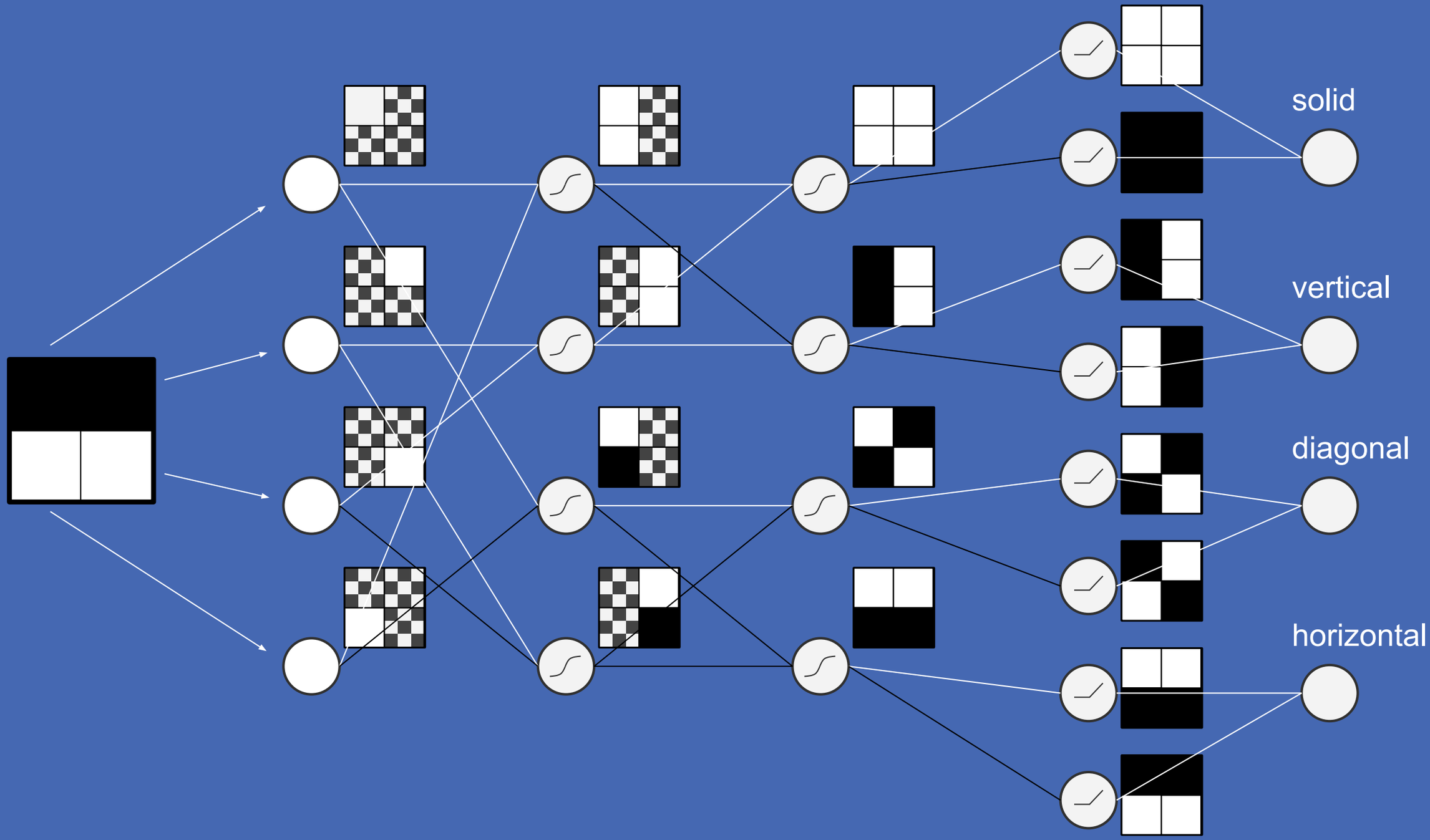


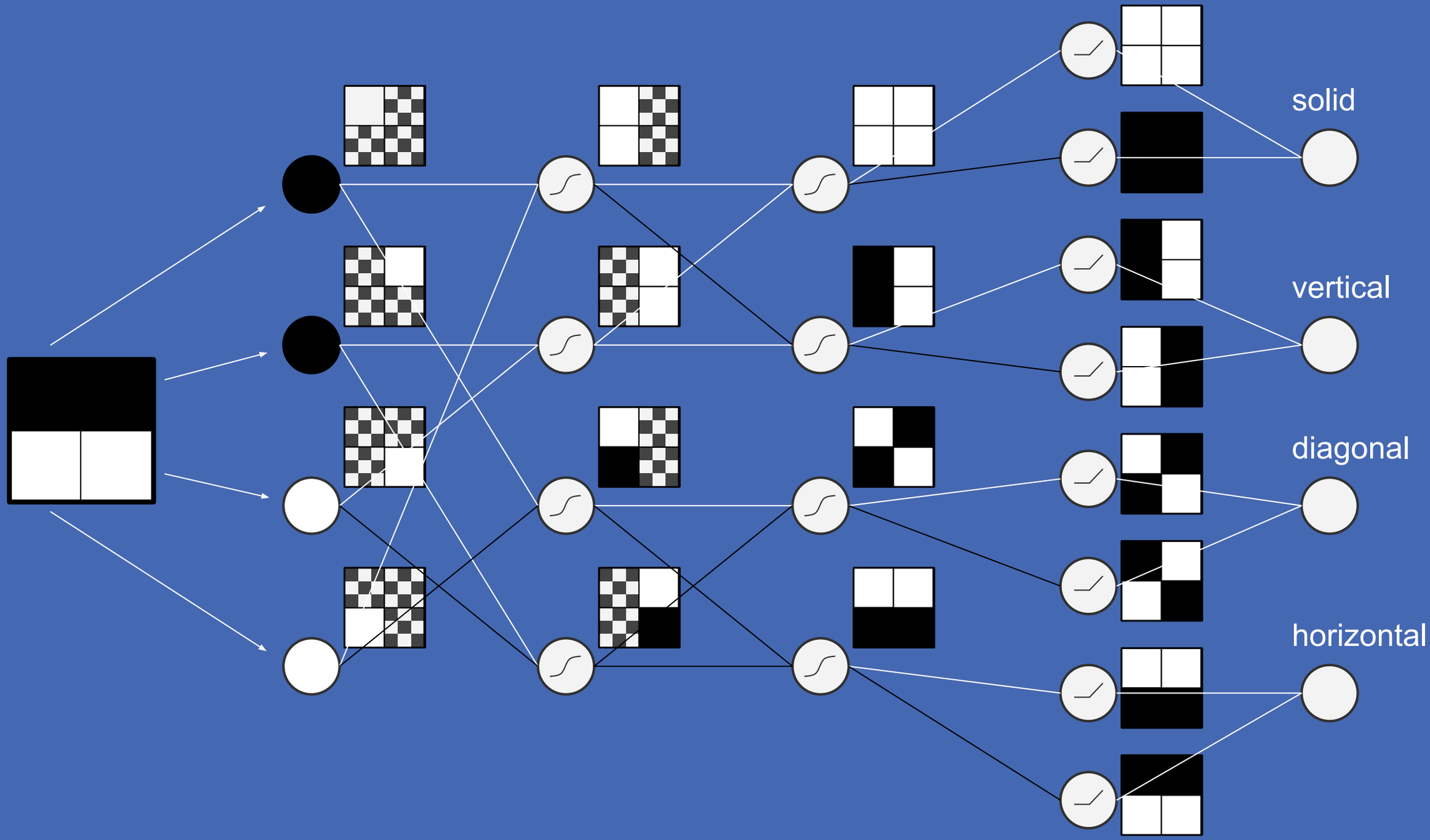
If your number is positive, keep it.
Otherwise you get a zero.

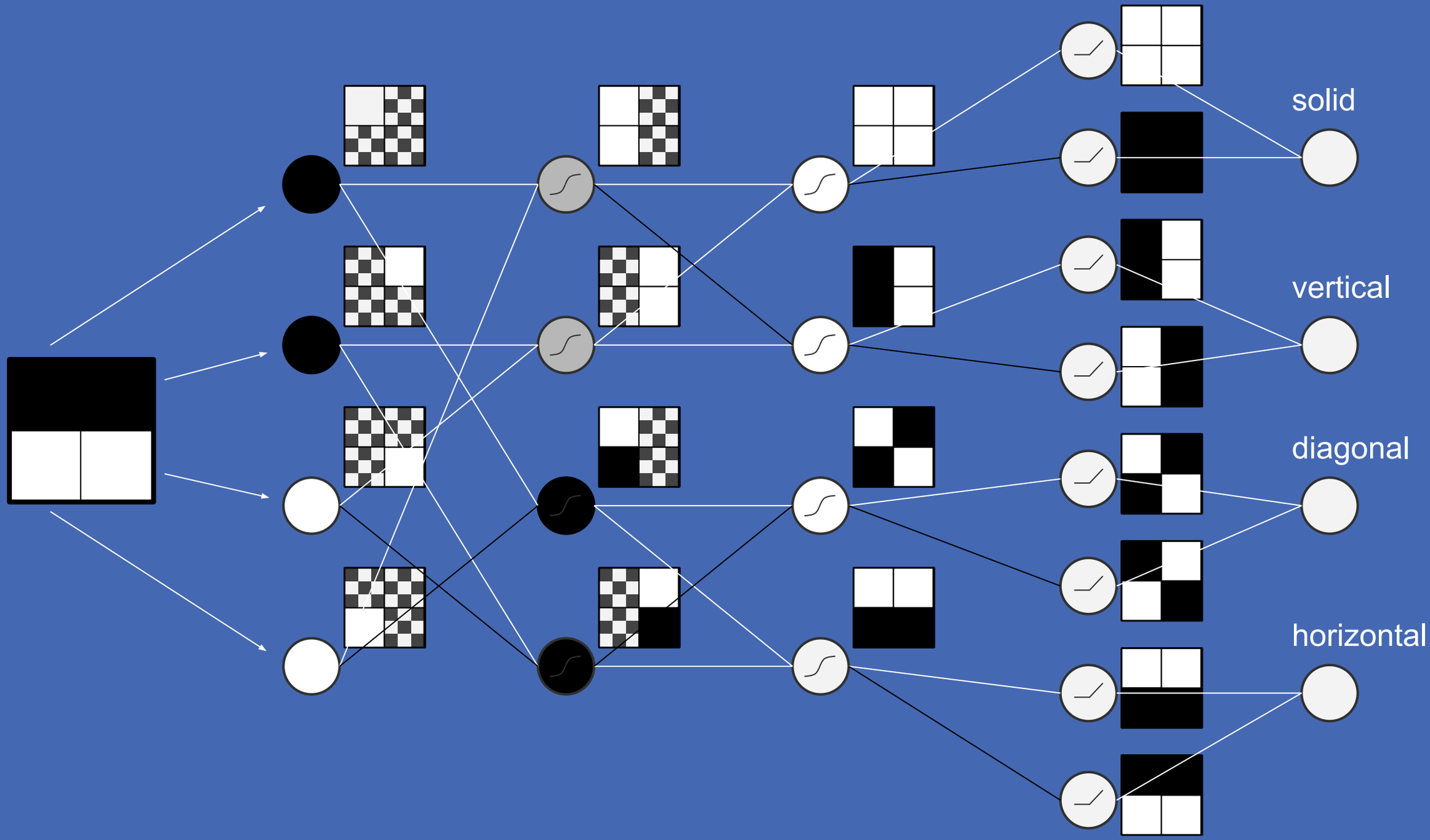


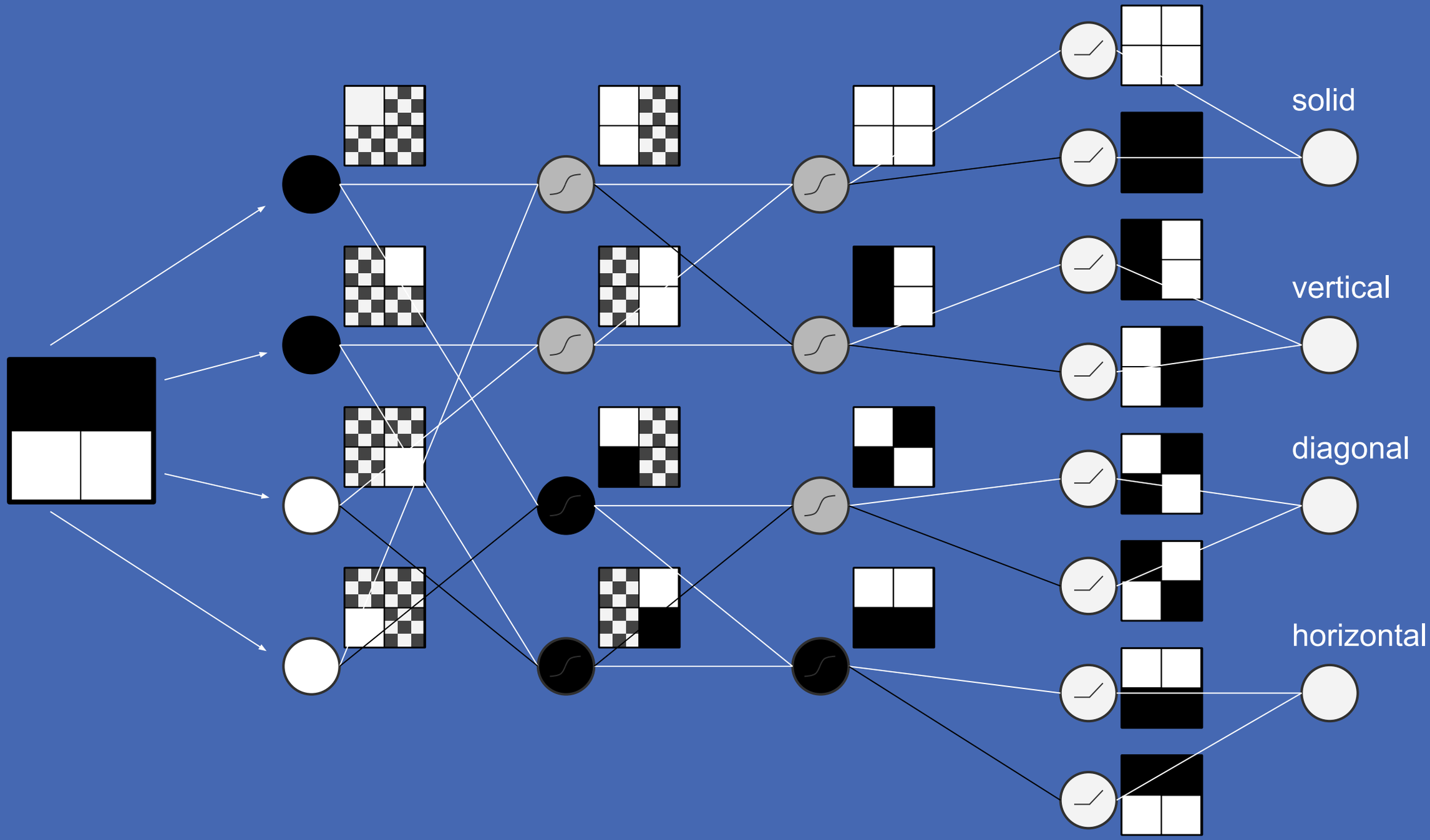
Add an output layer

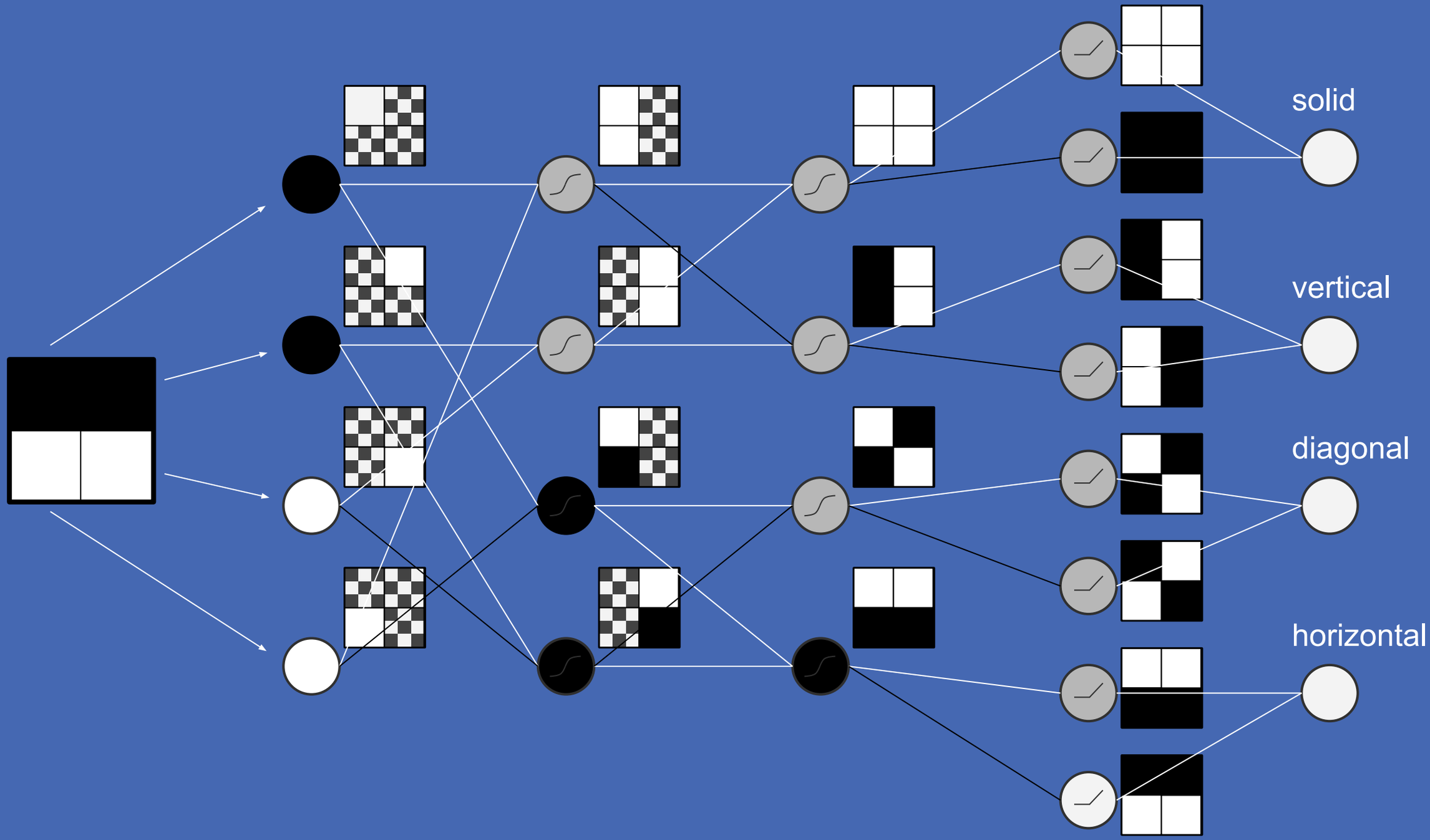


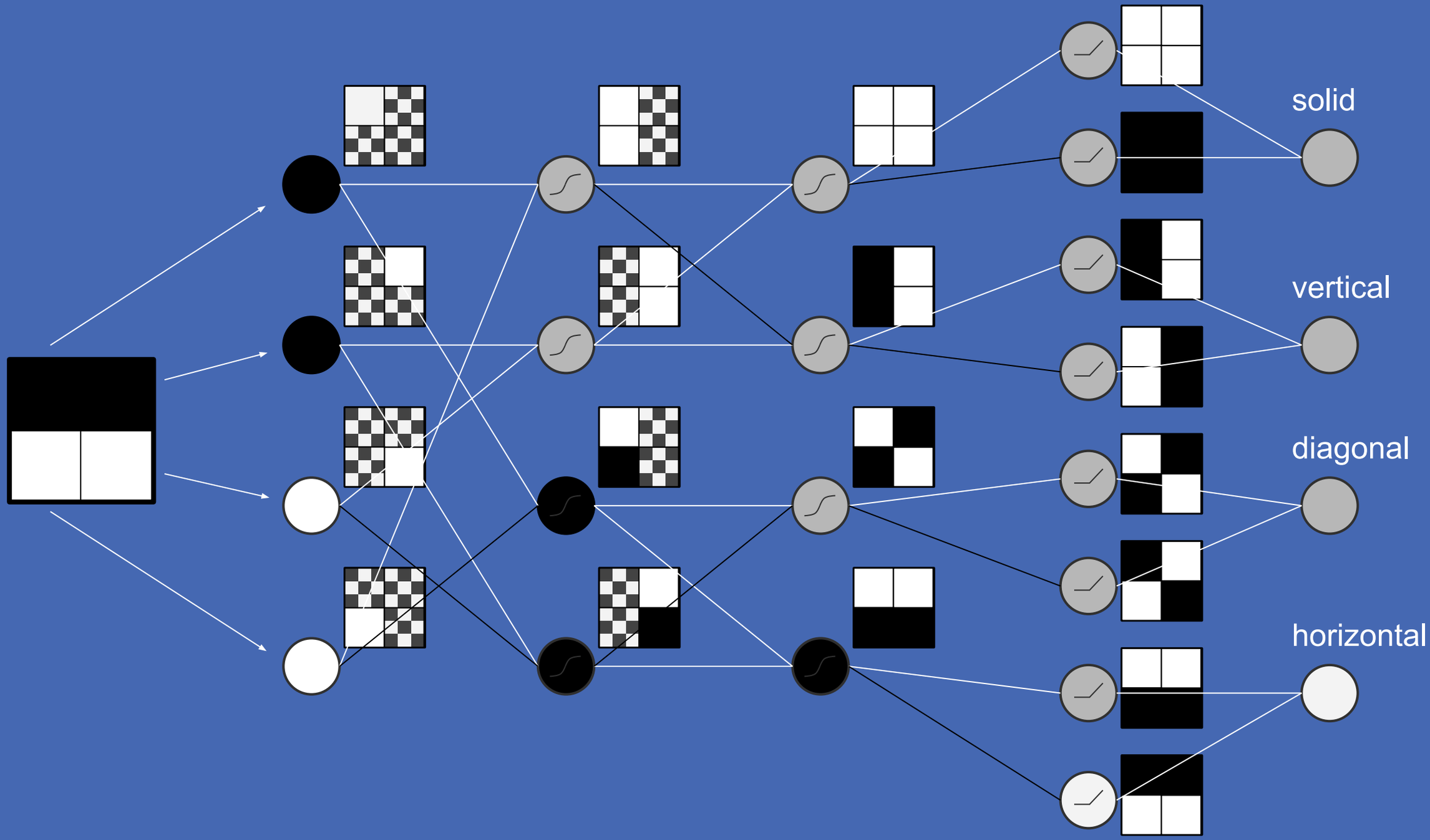

solid
vertical
diagonal
horizontal

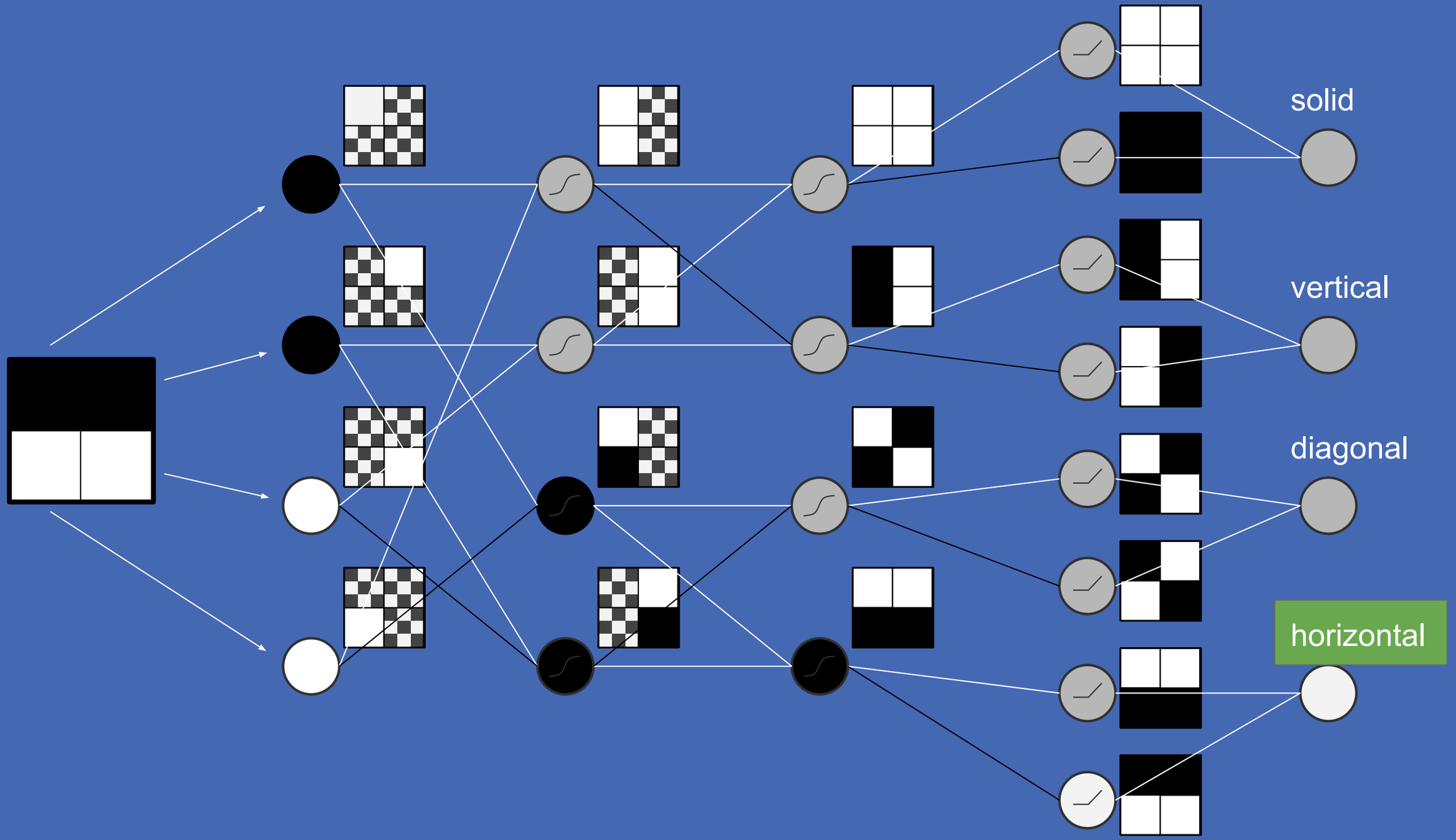

solid
vertical
diagonal
horizontal


solid
vertical
diagonal
horizontal


solid
vertical
diagonal
horizontal

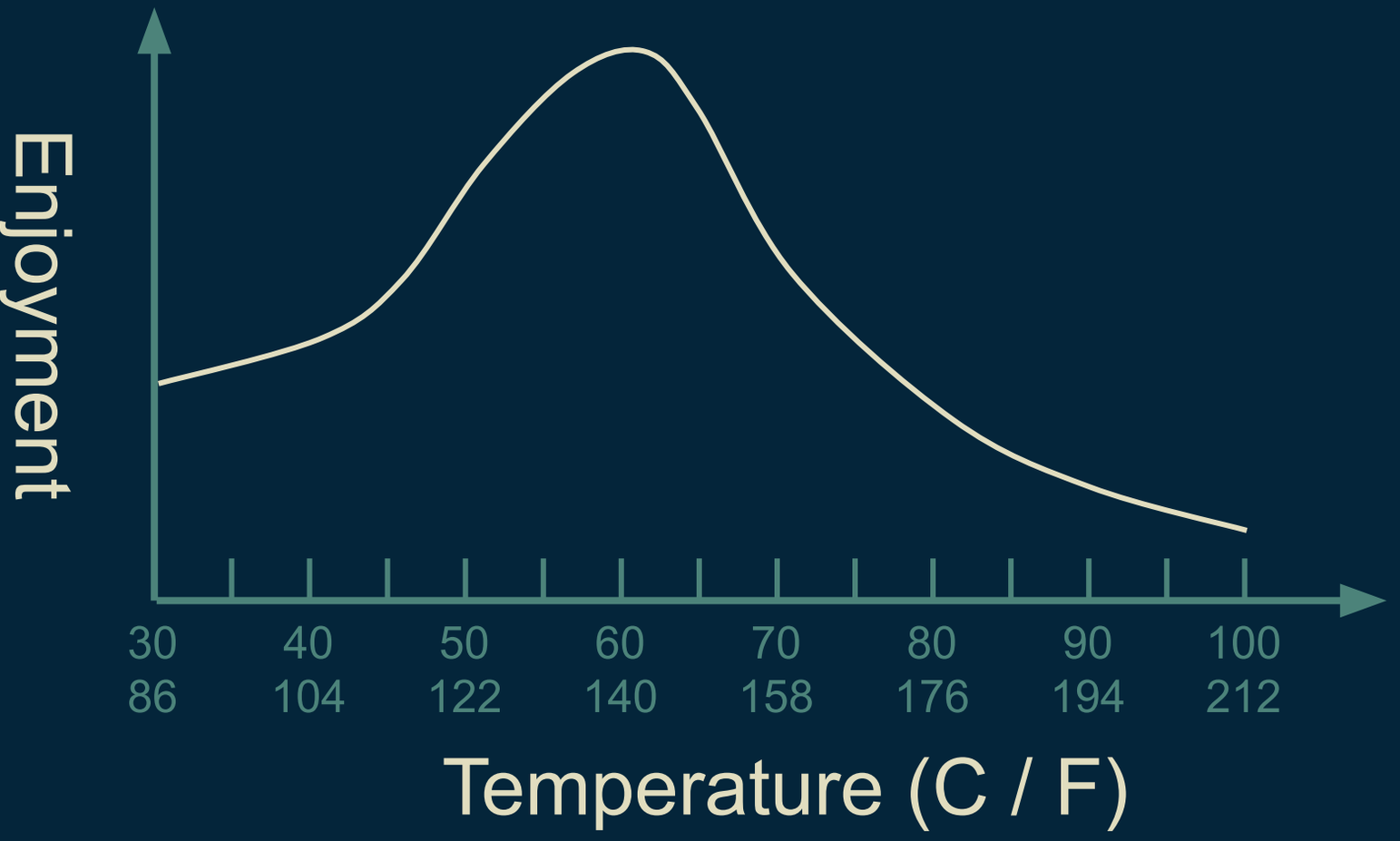

solid
vertical
diagonal
horizontal


solid
vertical
diagonal
horizontal

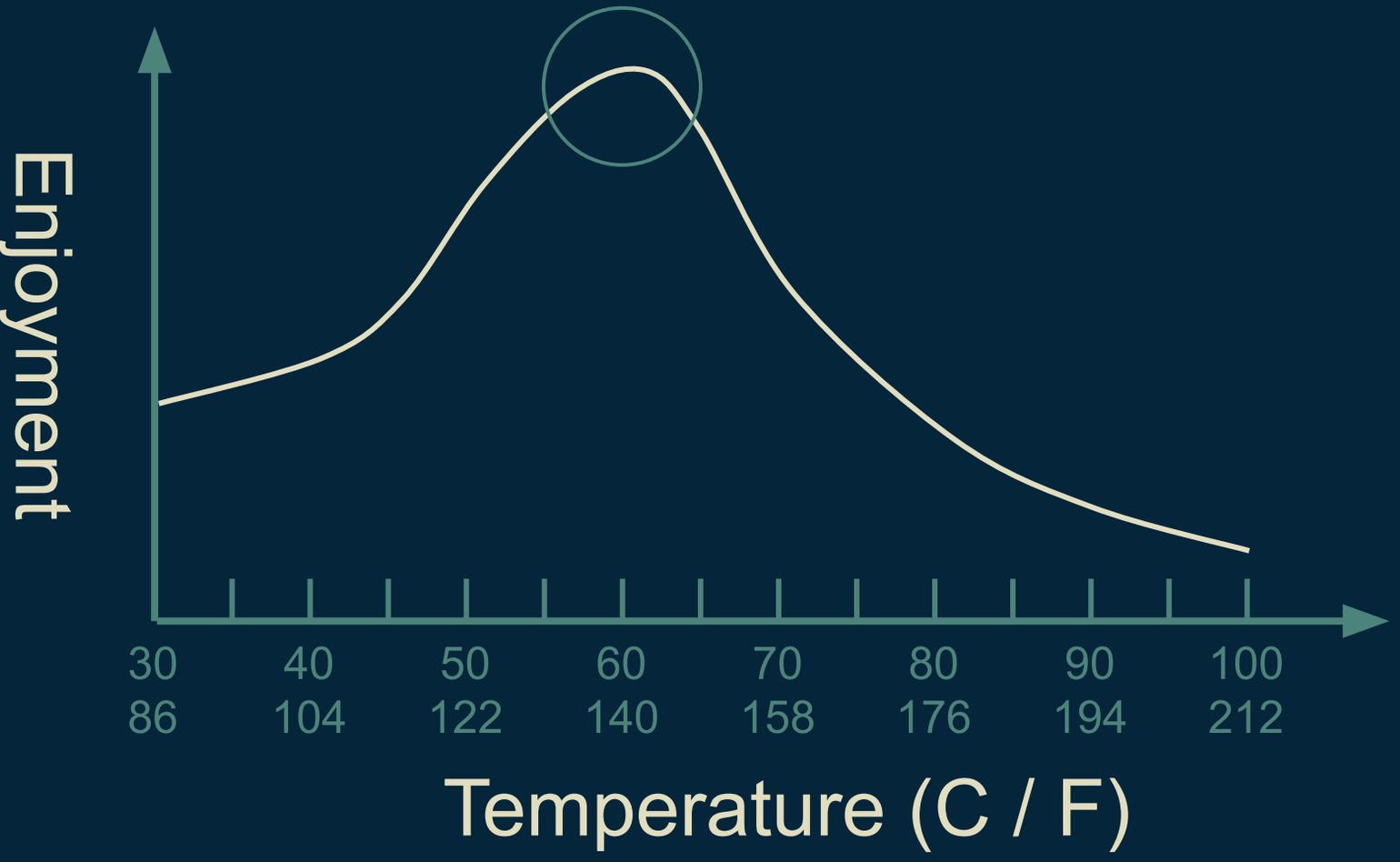

solid
vertical
diagonal
horizontal

How Optimization for Machine Learning Works

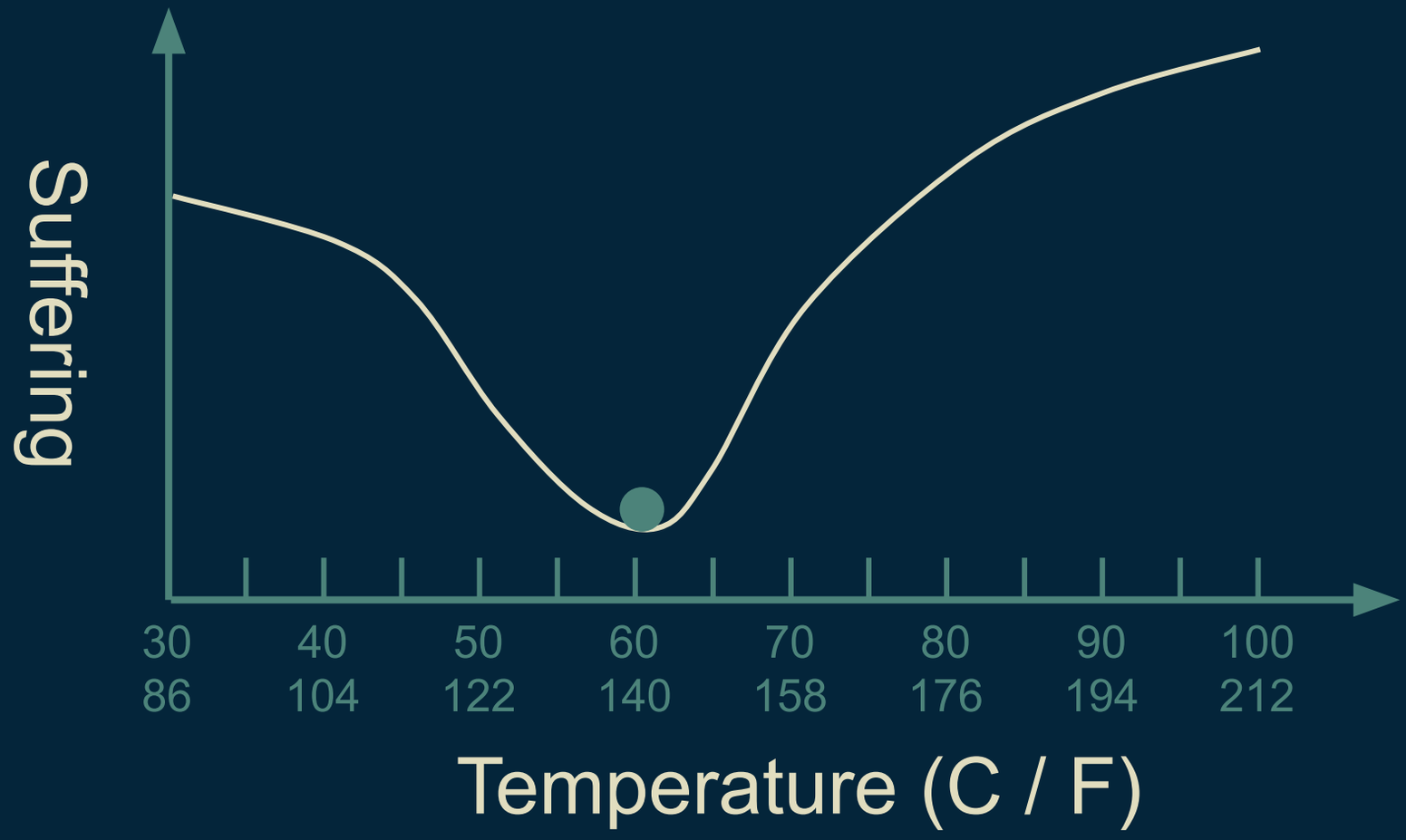
Tea drinking temperature



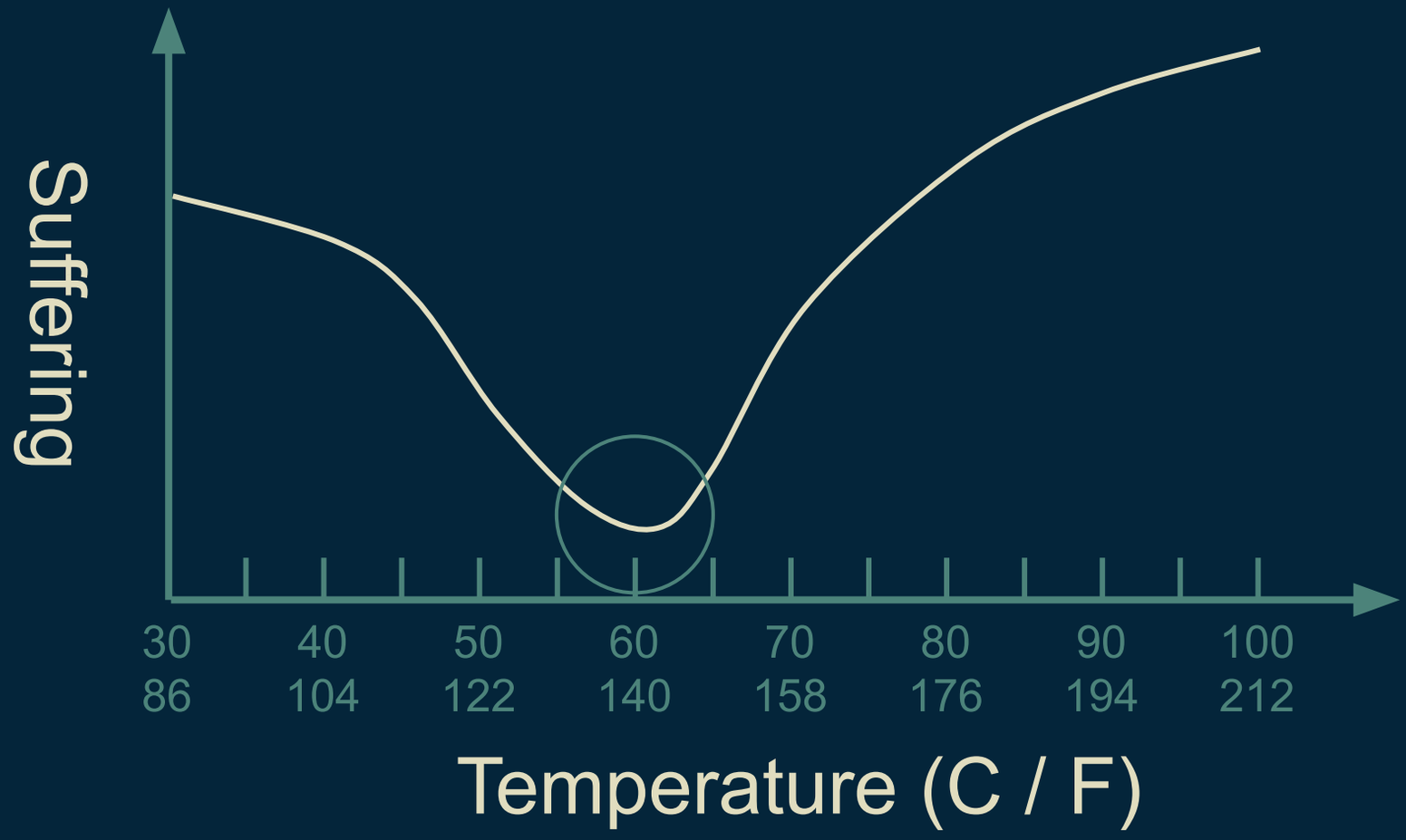
Tea drinking temperature



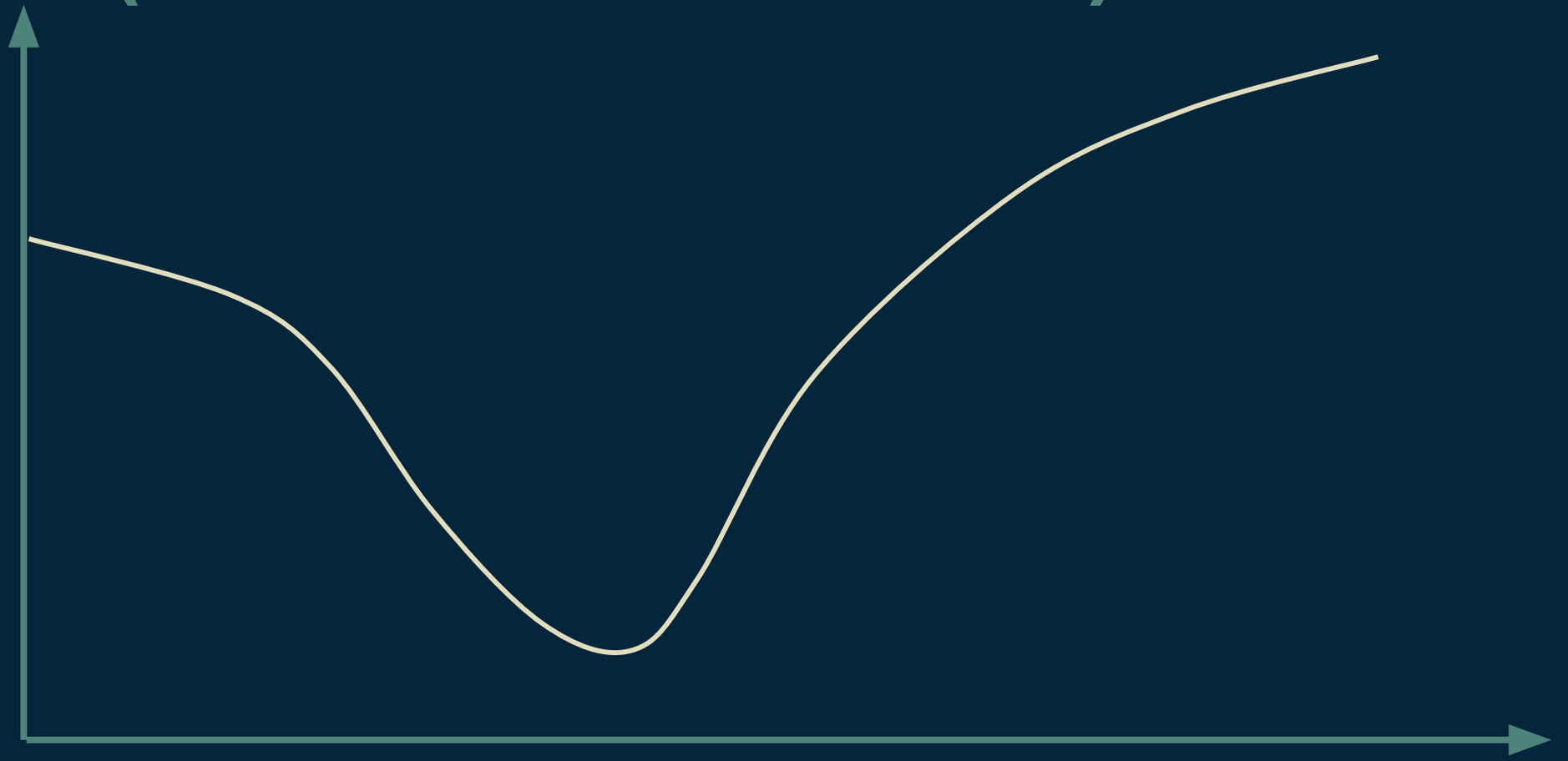
Tea drinking temperature



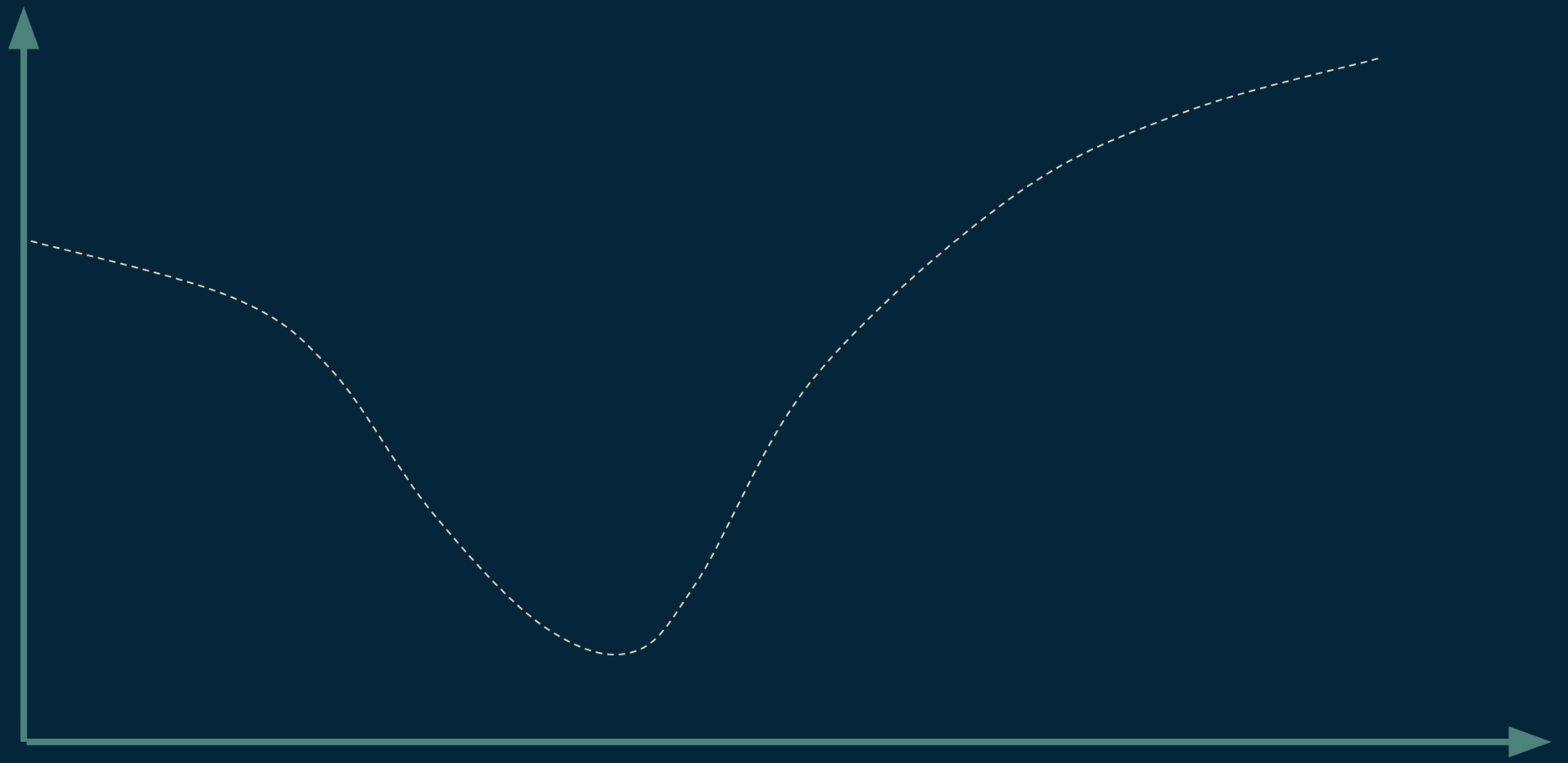
Tea drinking temperature



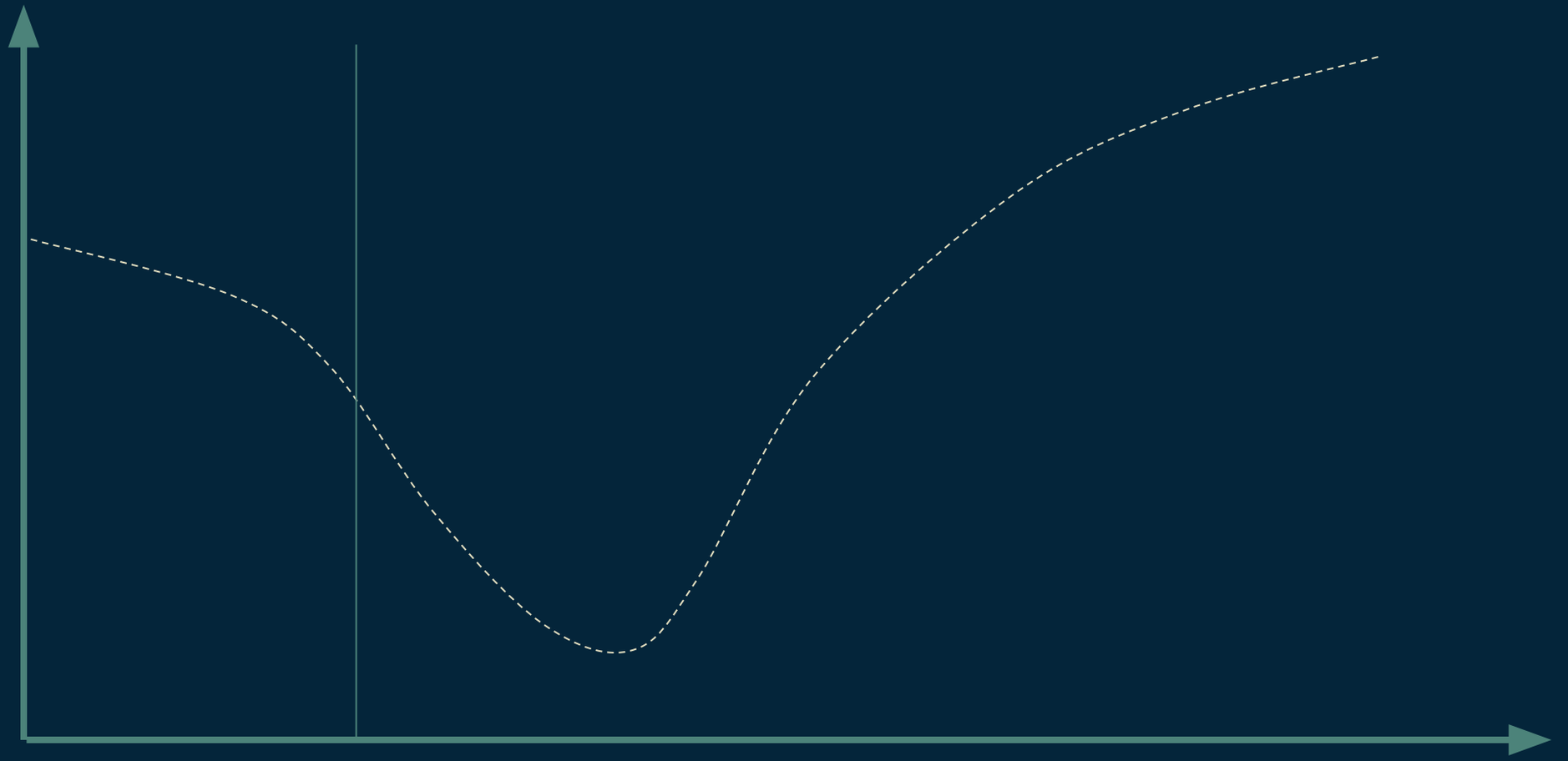
Pick the lowest point (Exhaustive search)



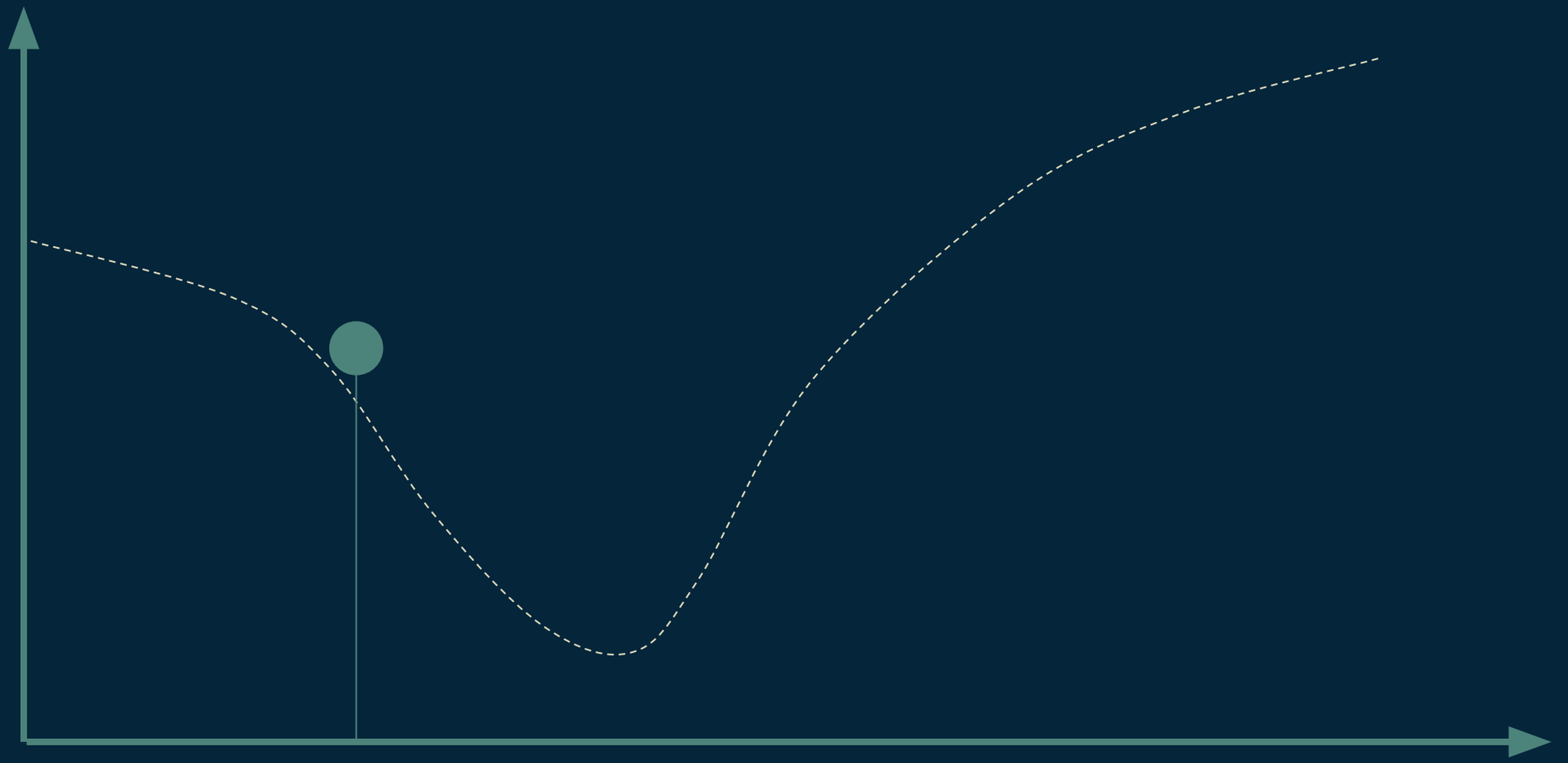
Exhaustive search



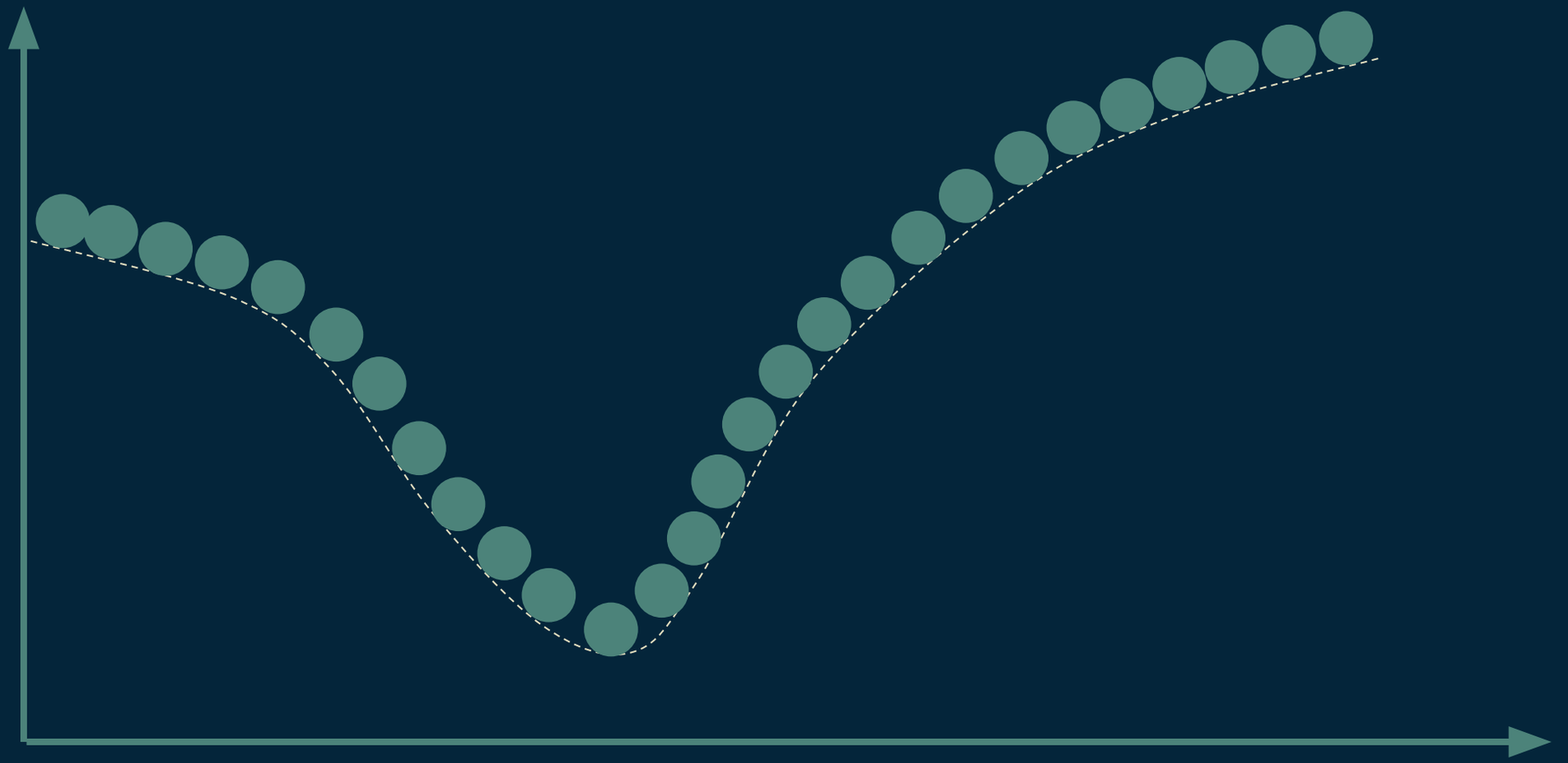
Exhaustive search



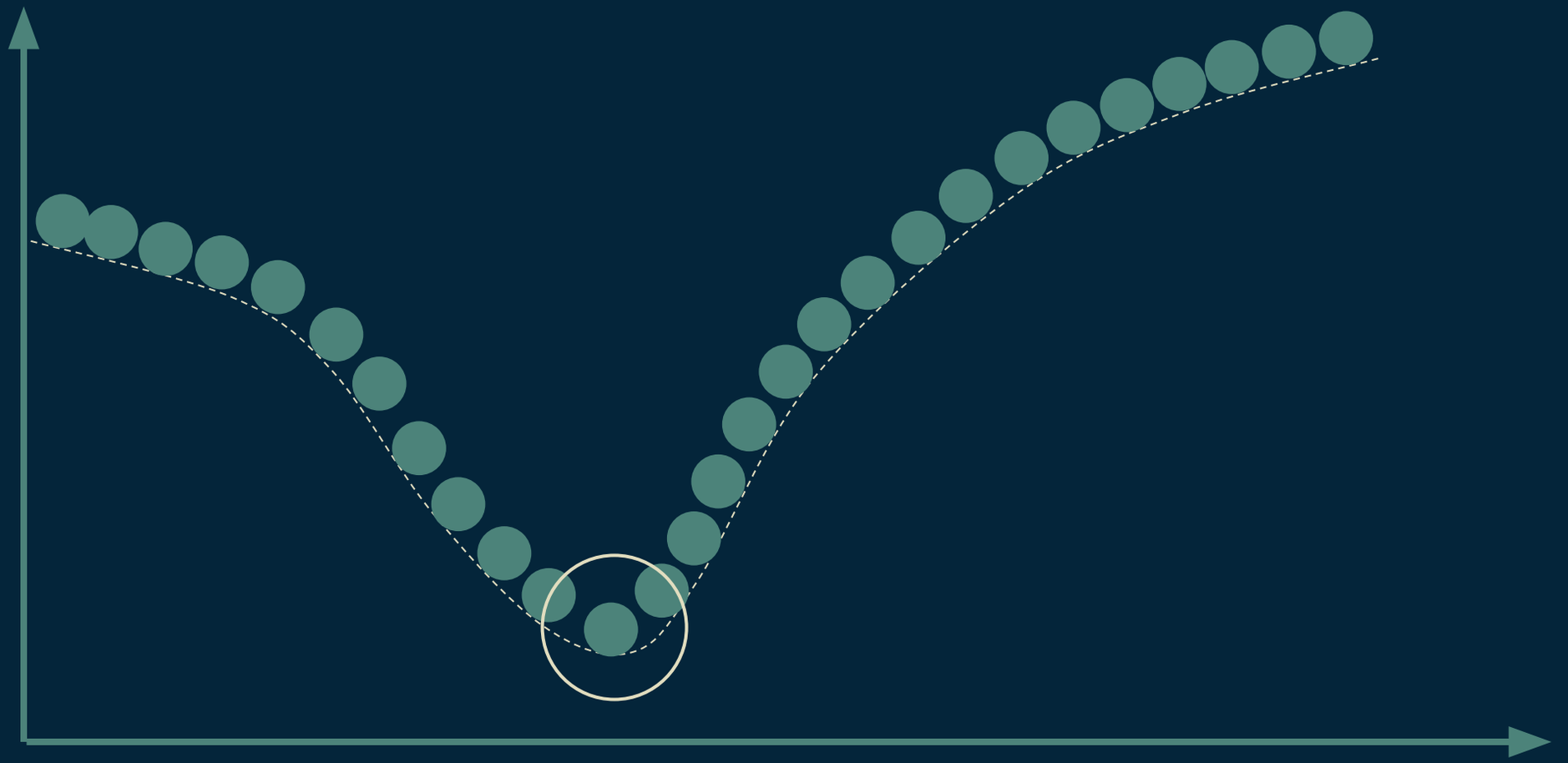
Exhaustive search



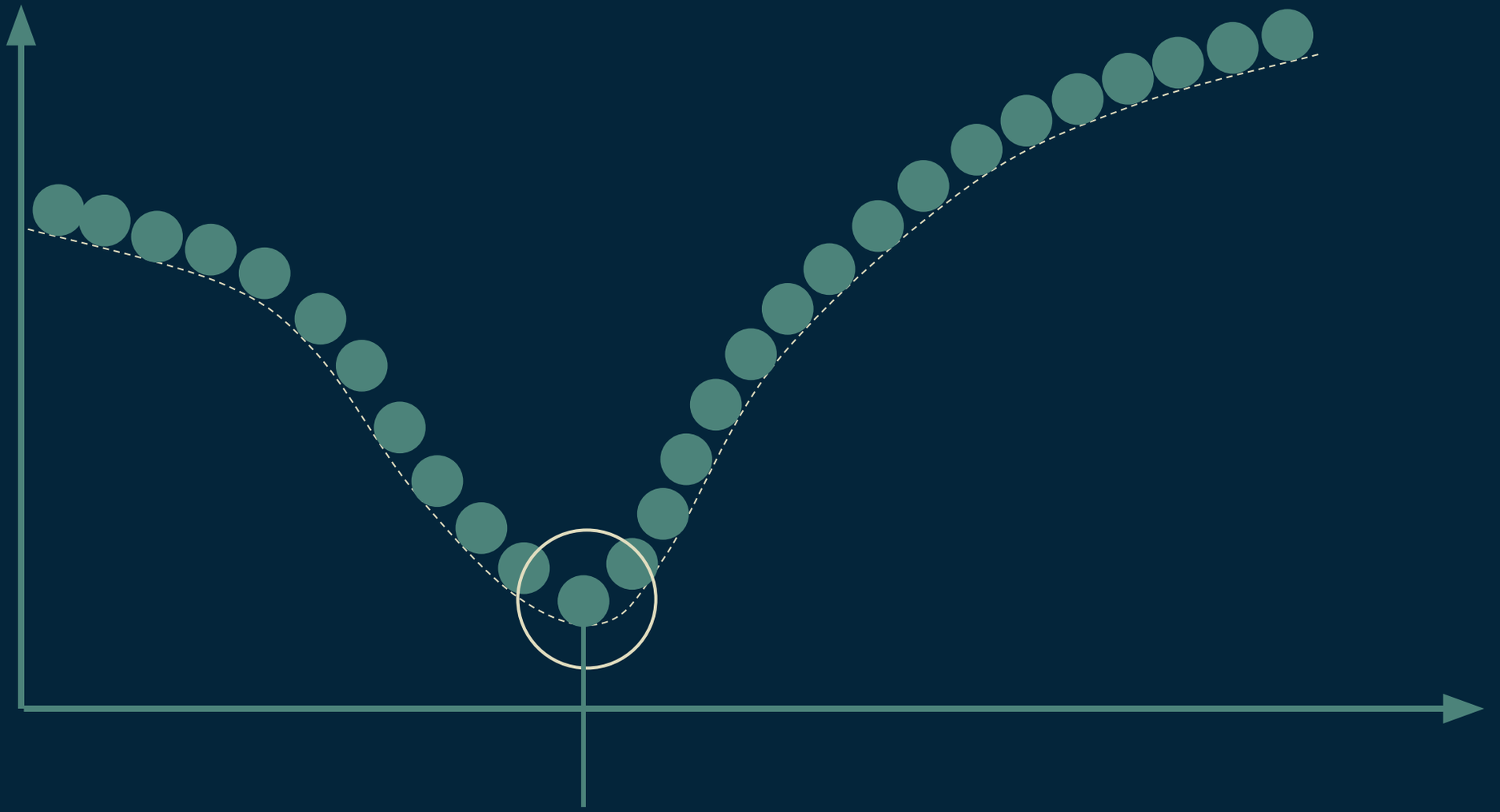
Exhaustive search



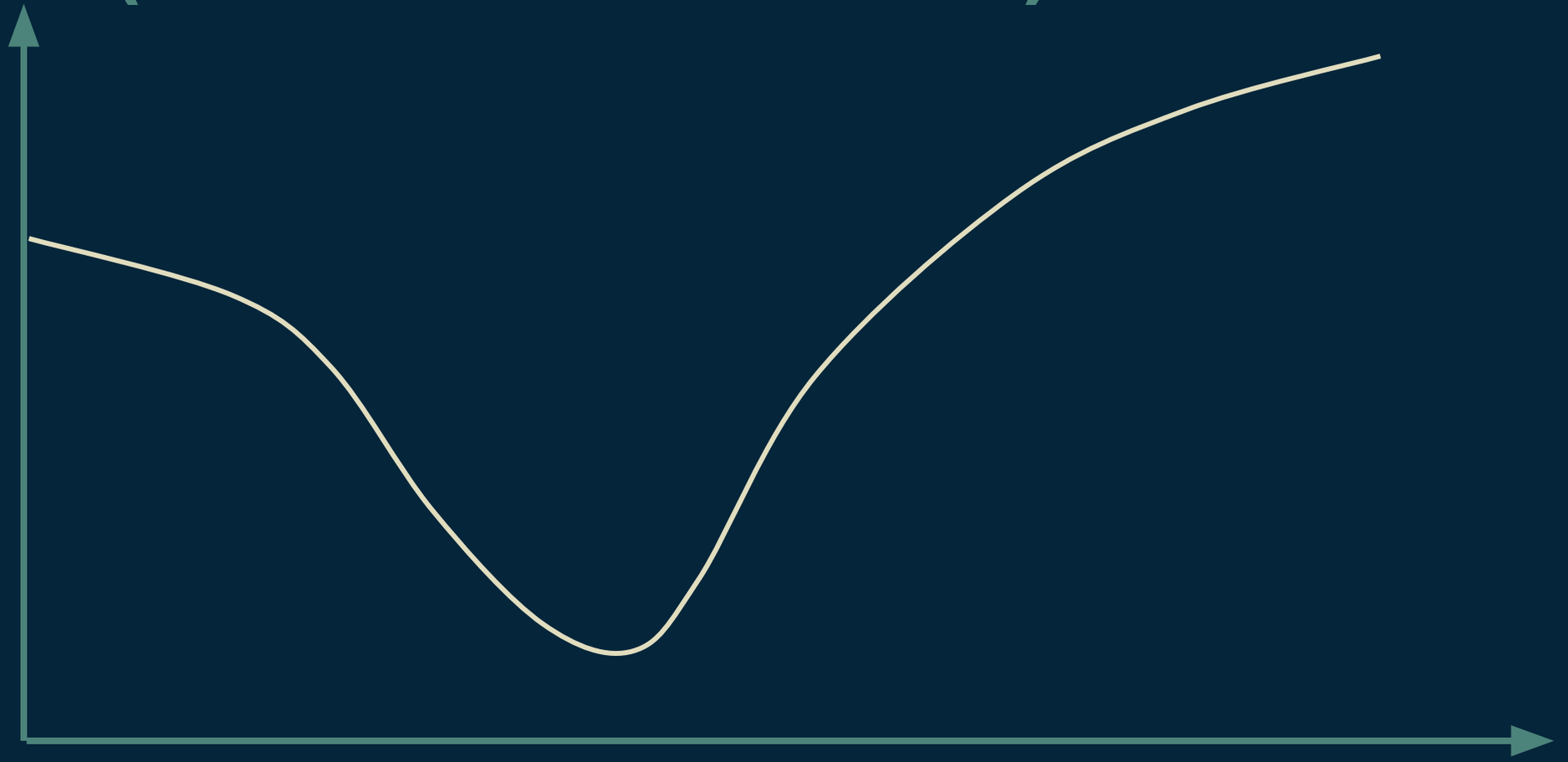
Exhaustive search



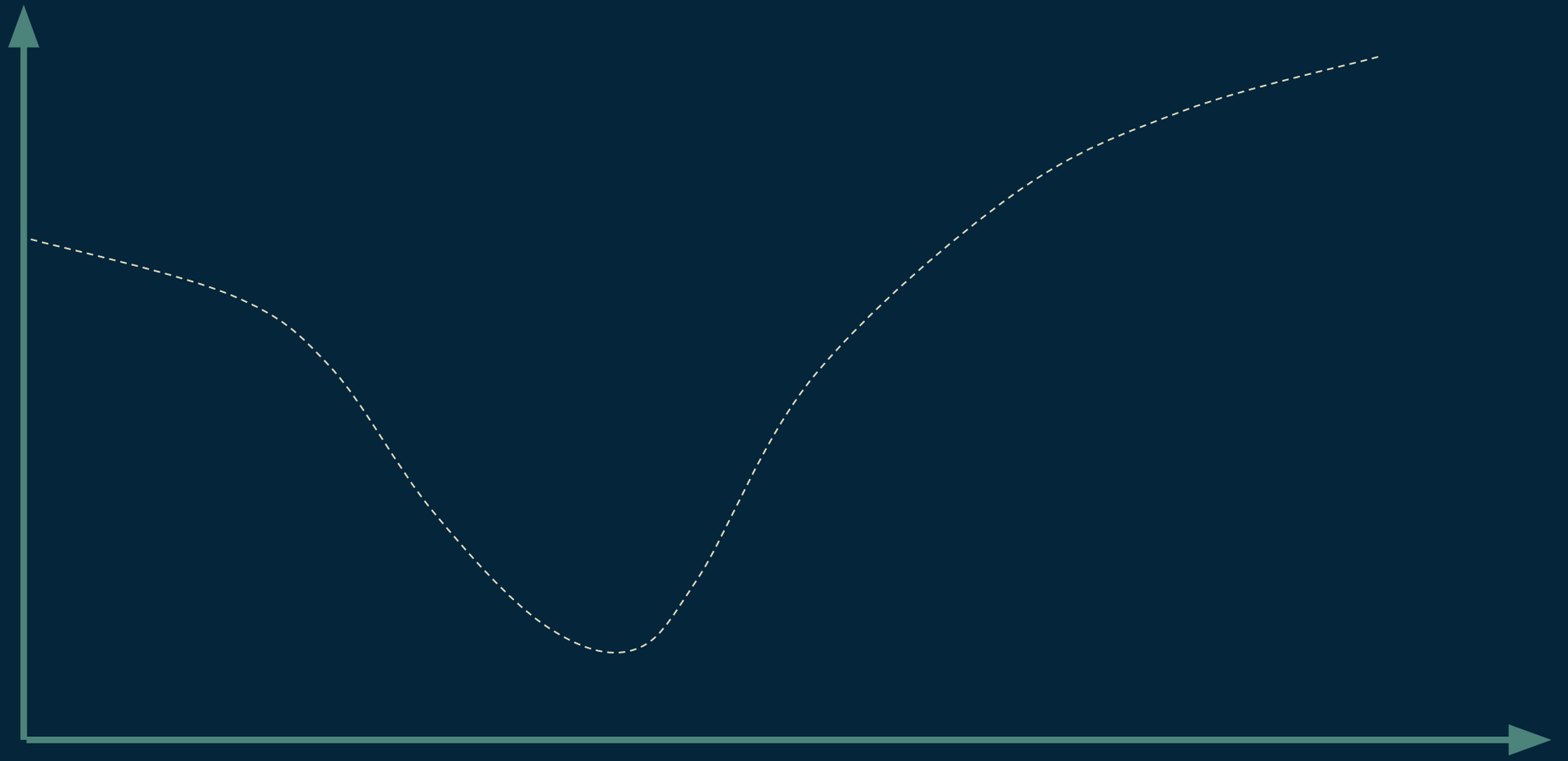
Exhaustive search



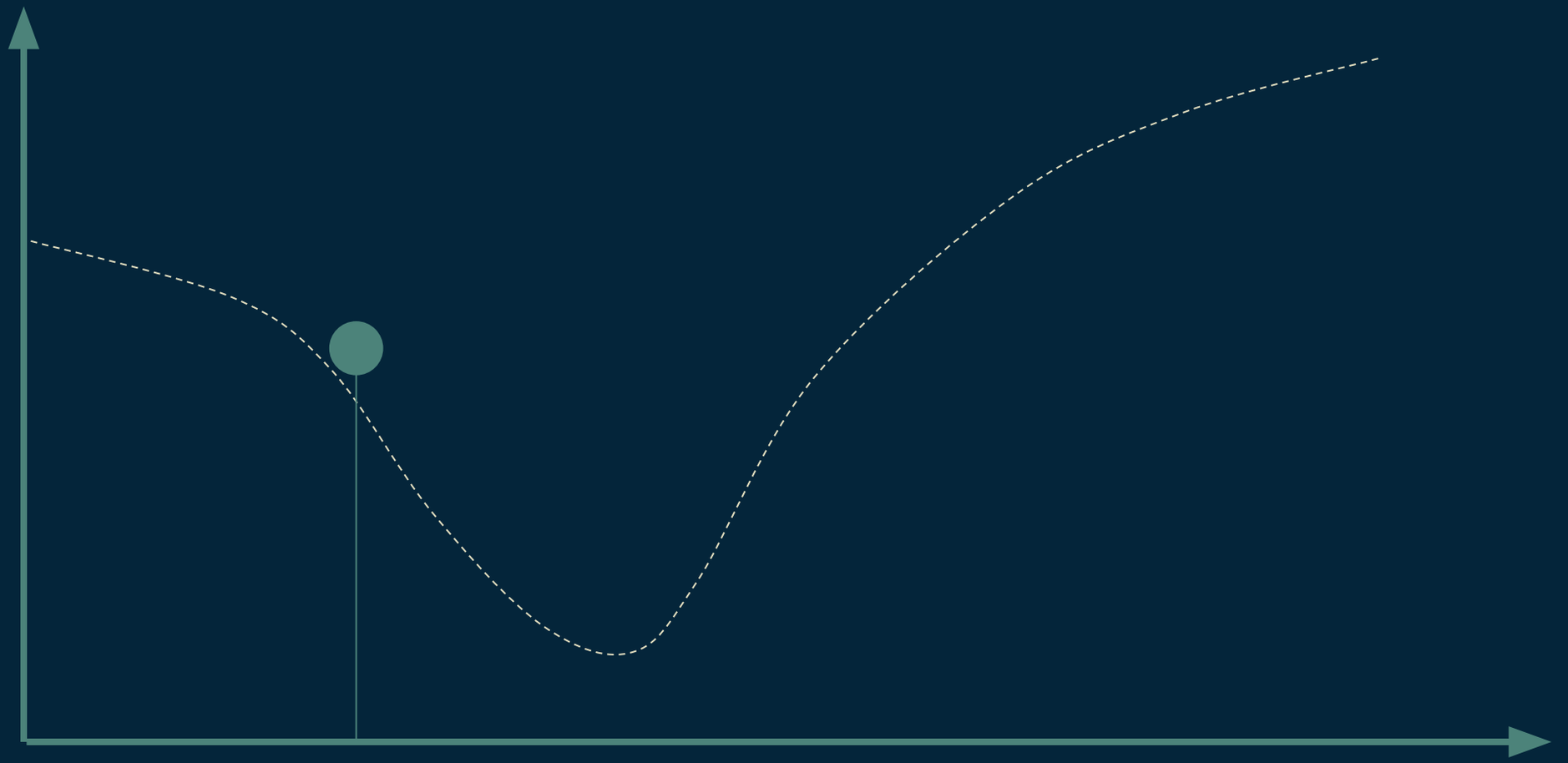
Let the marble roll downhill (Gradient descent)



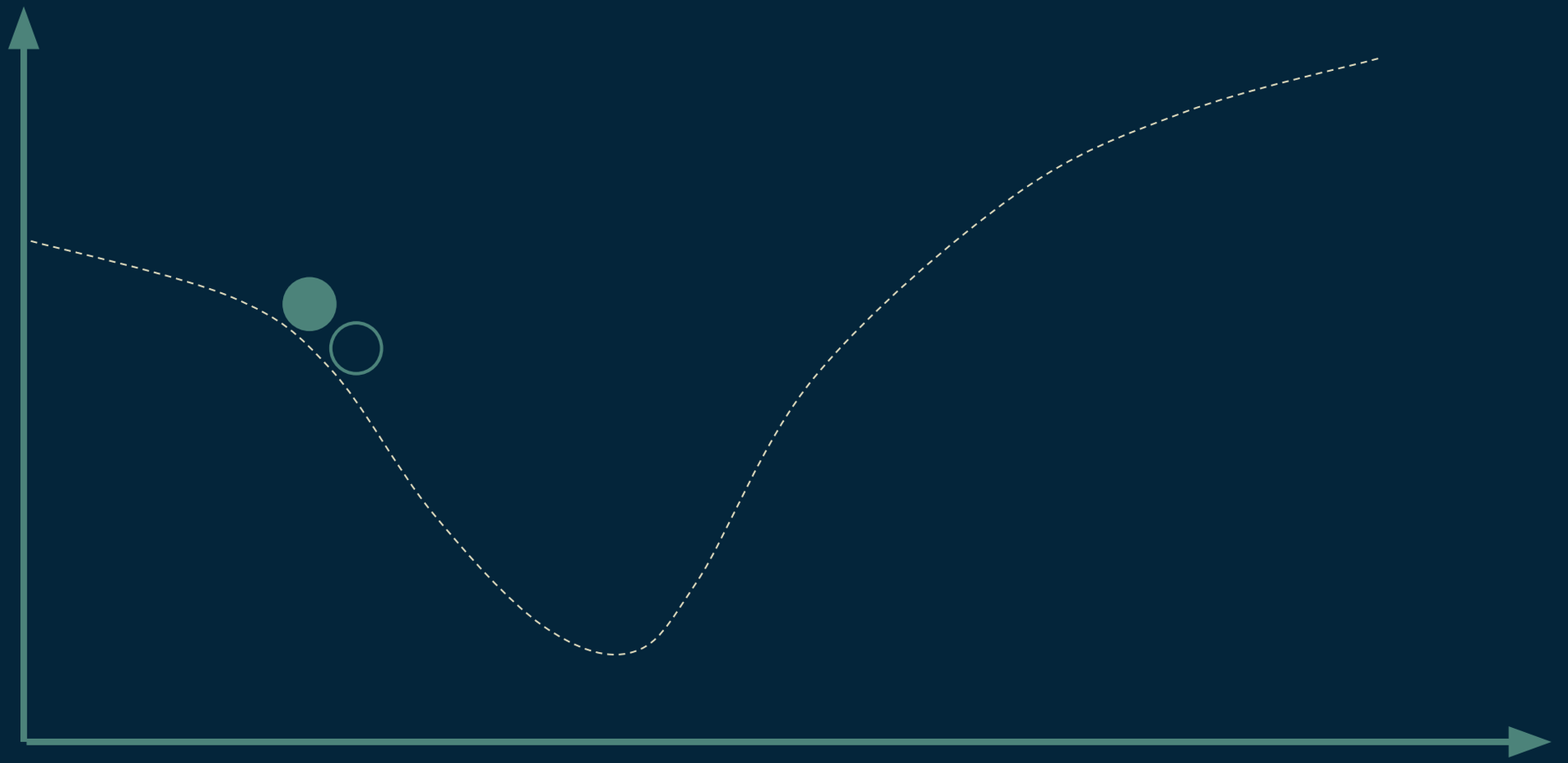
Gradient descent



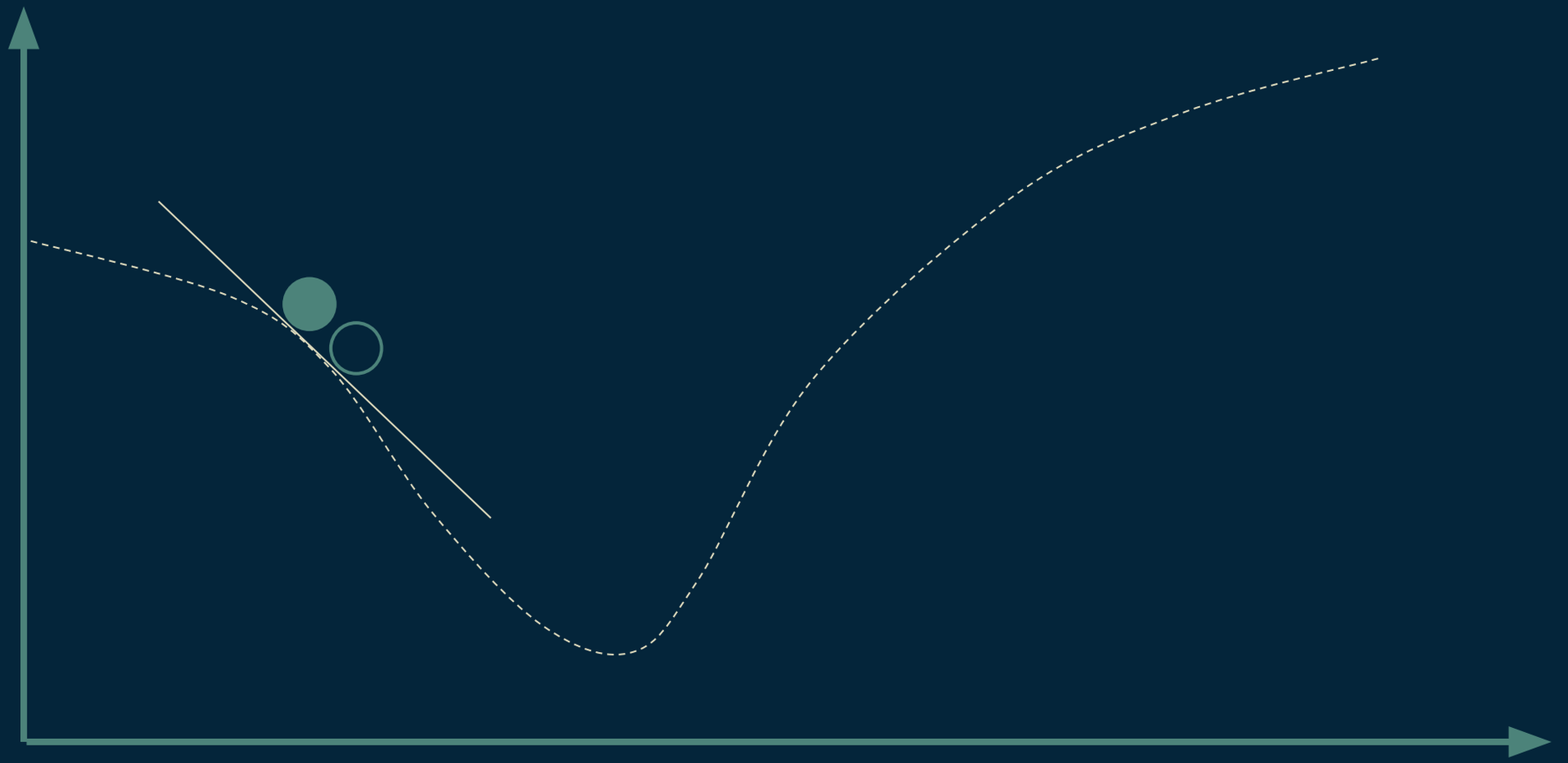
Gradient descent



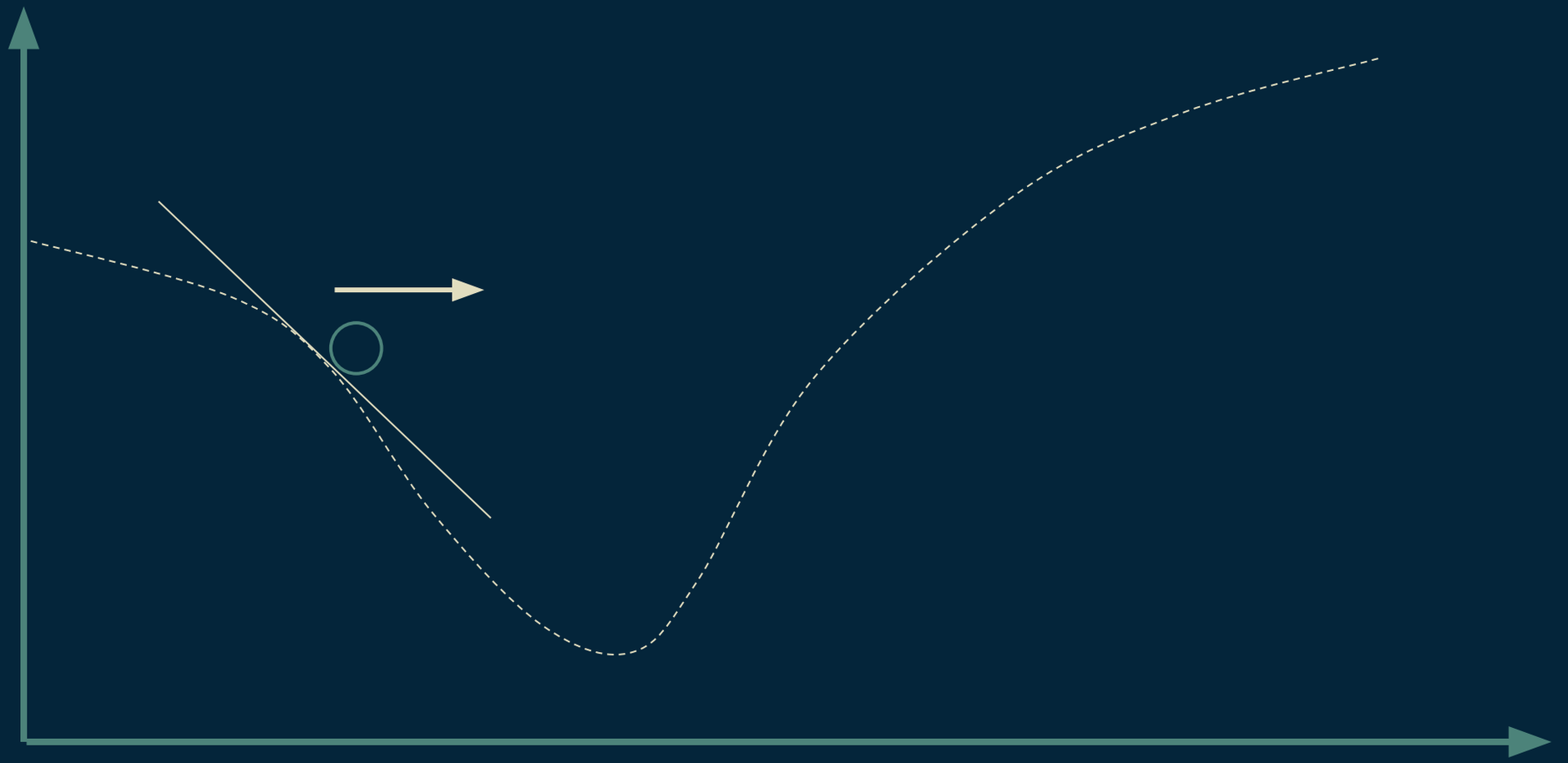
Gradient descent



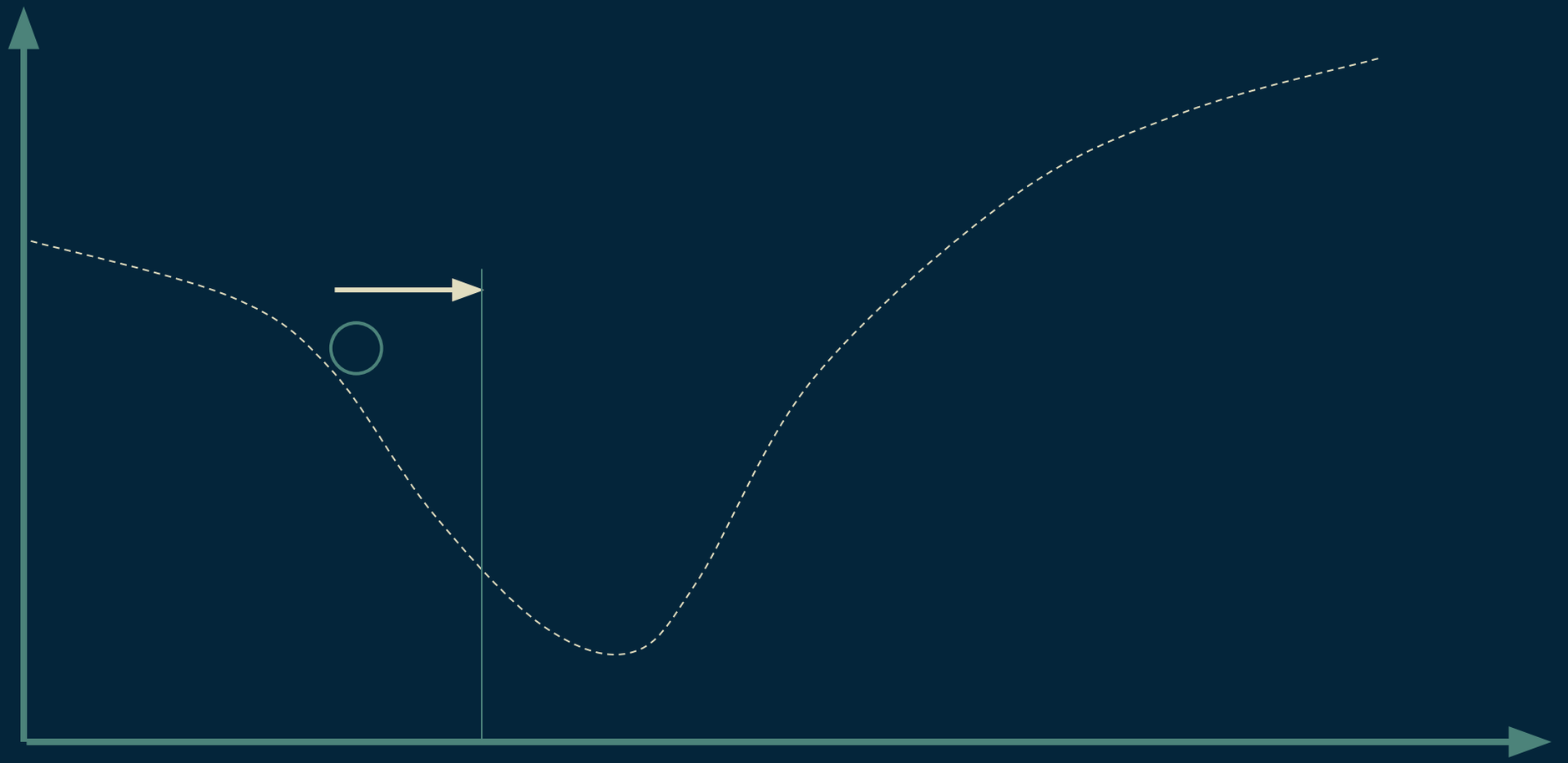
Gradient descent



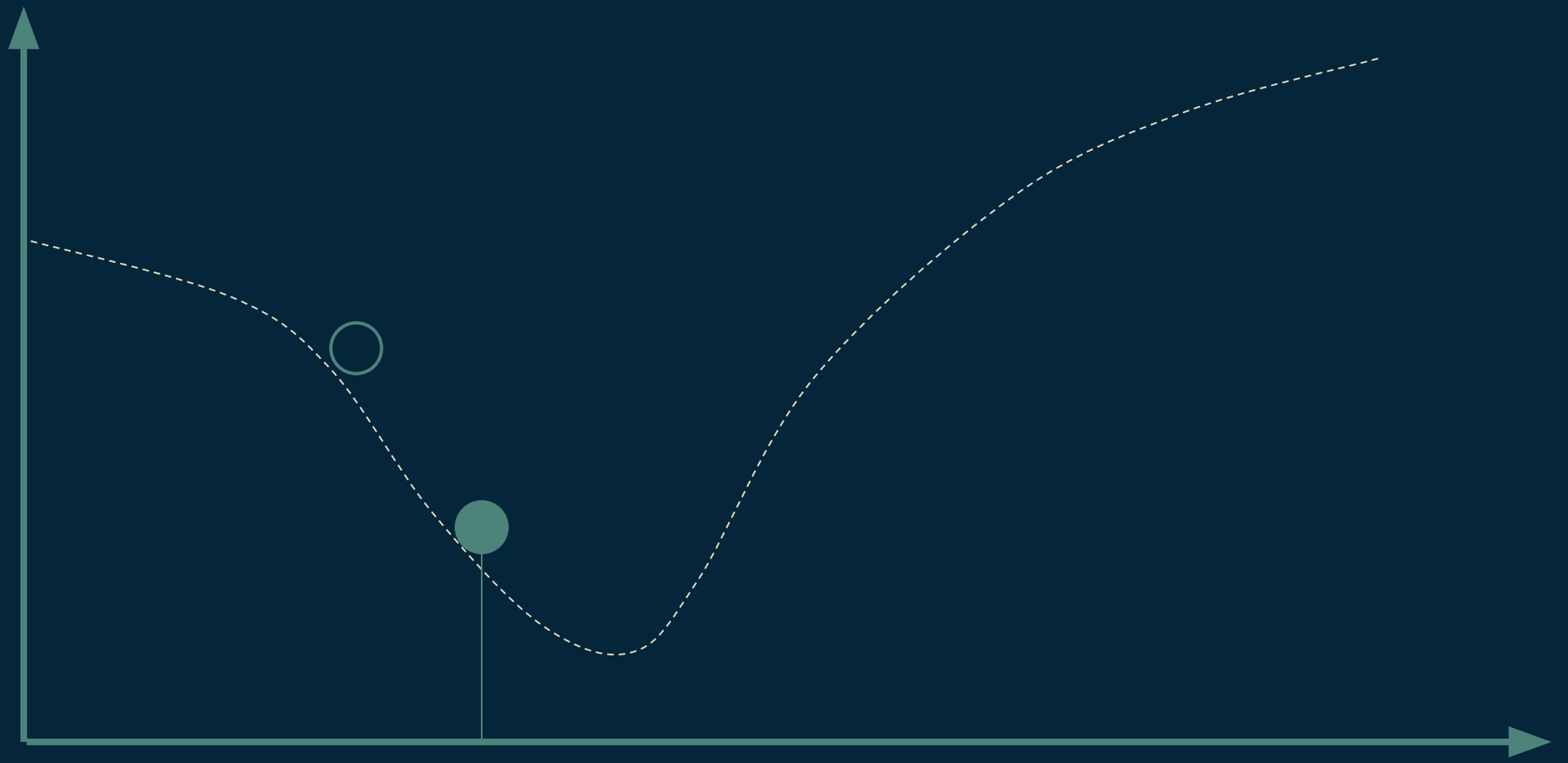
Gradient descent



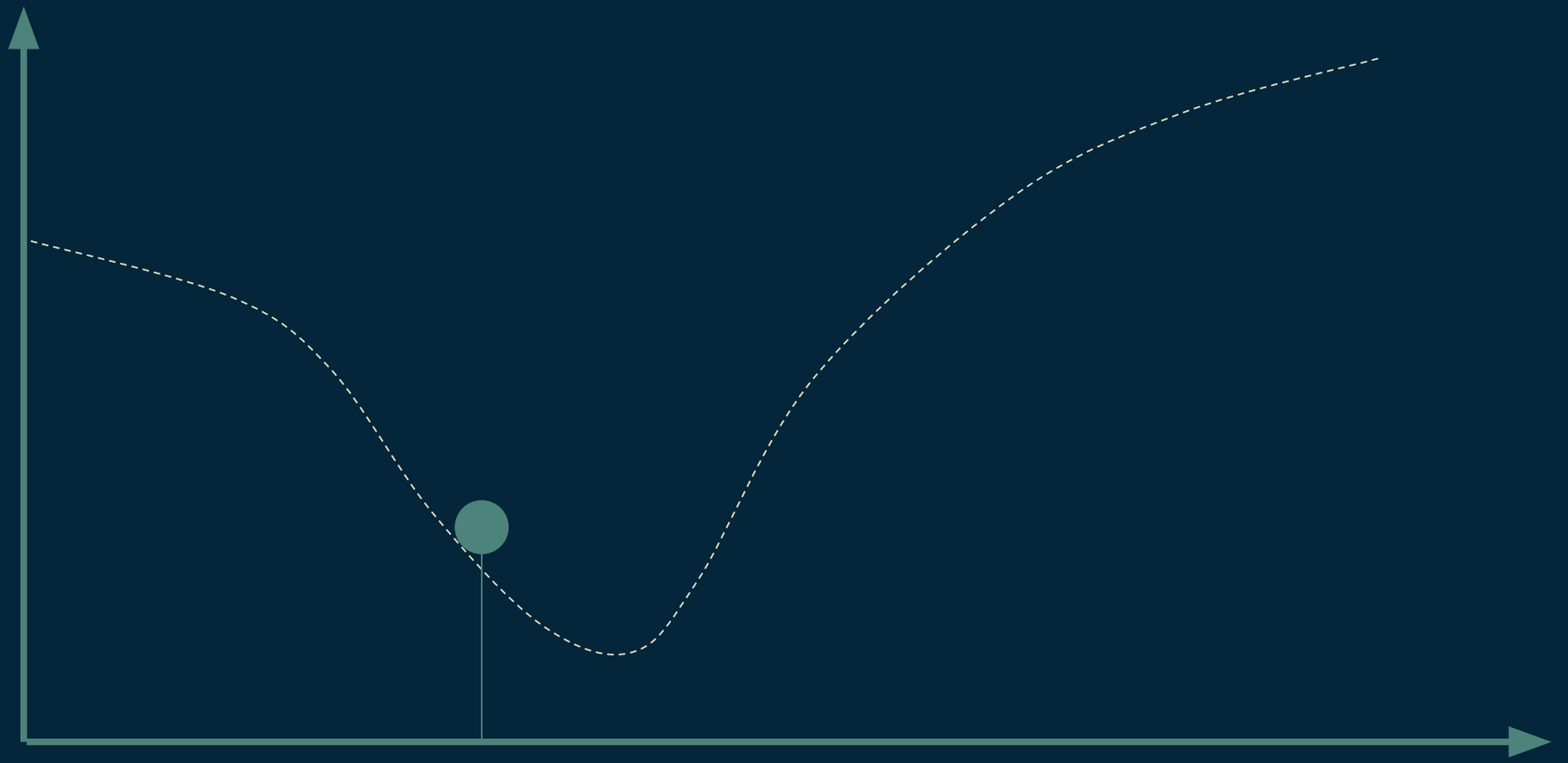
Gradient descent



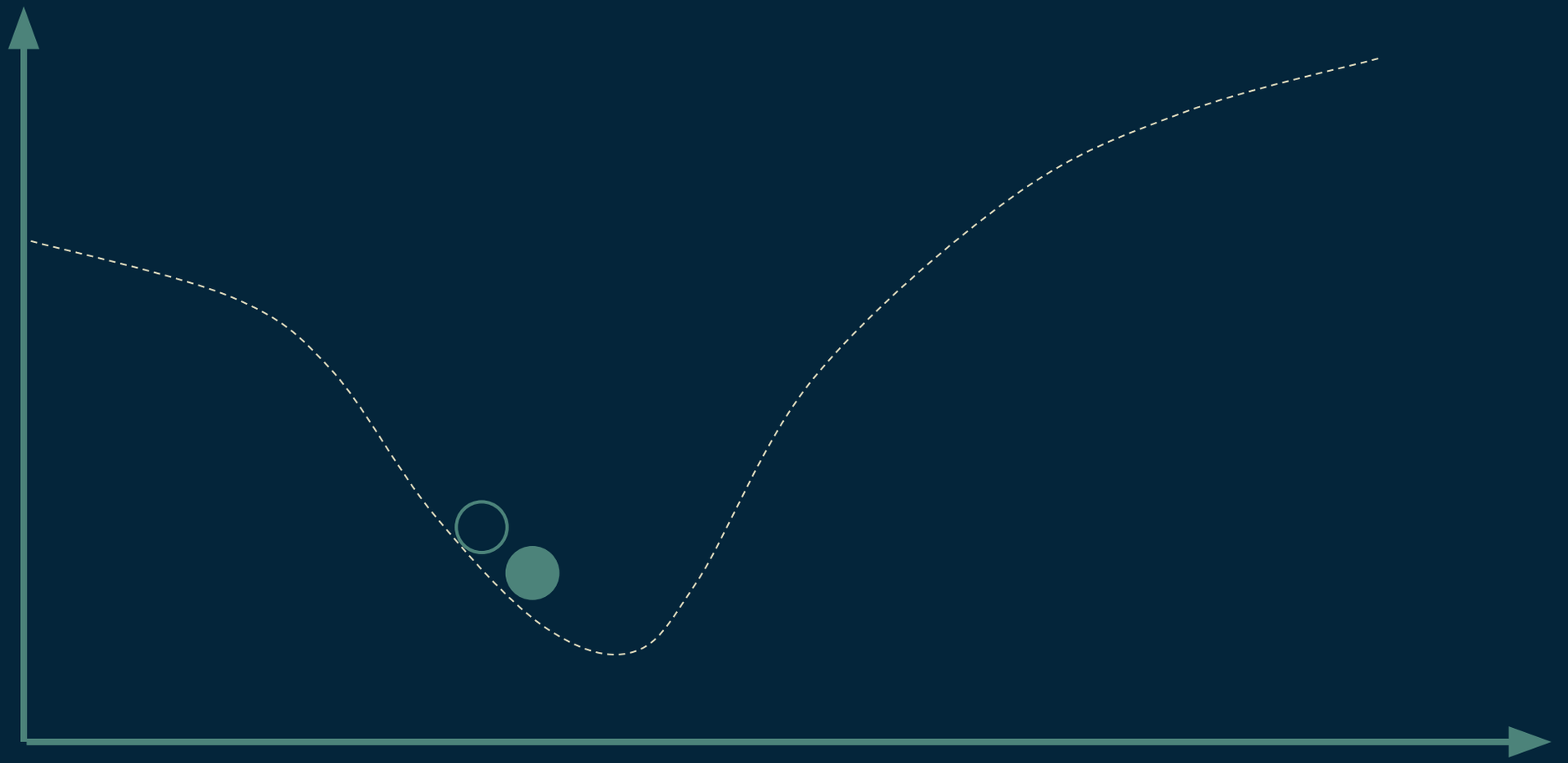
Gradient descent



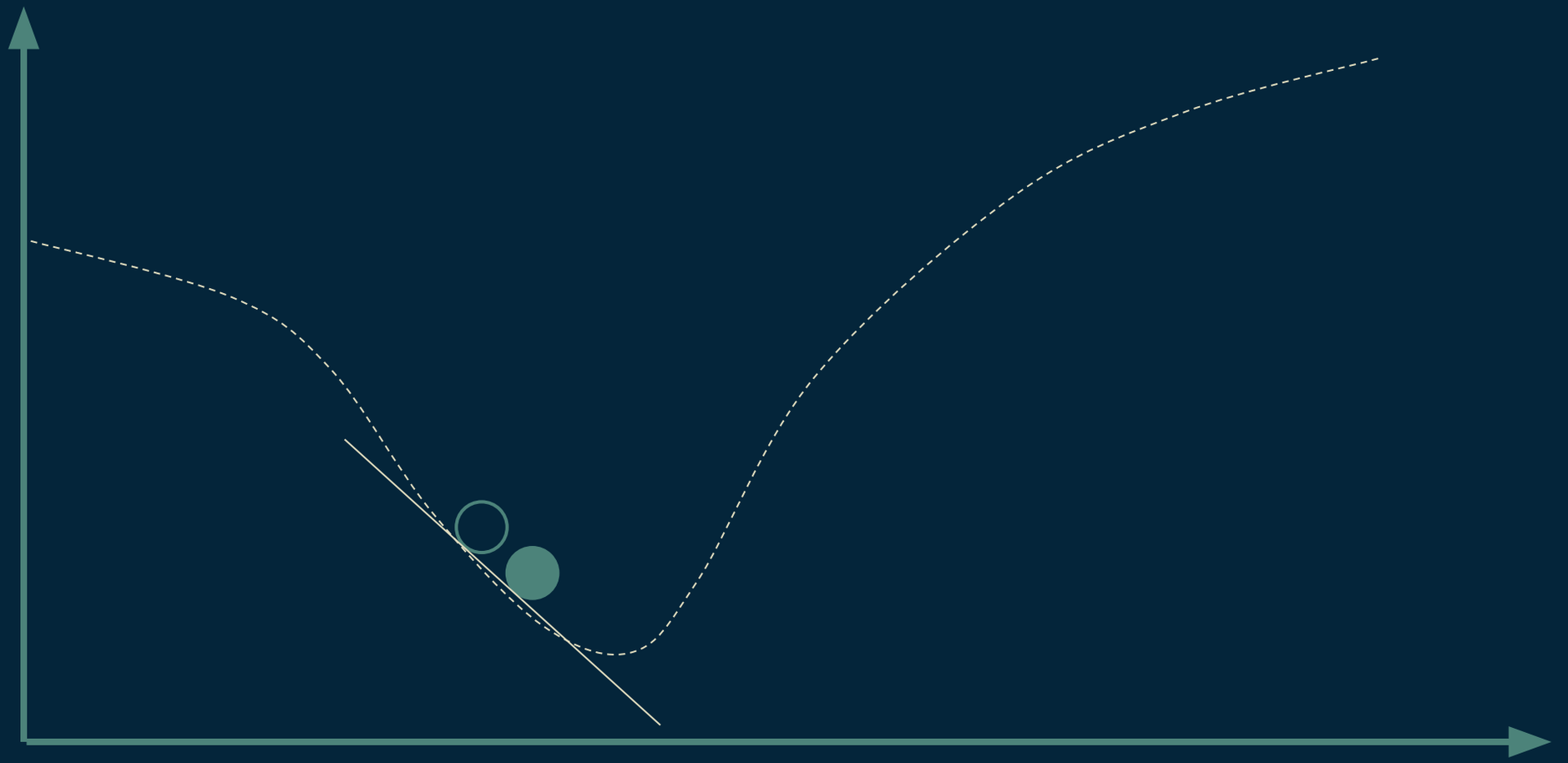
Gradient descent



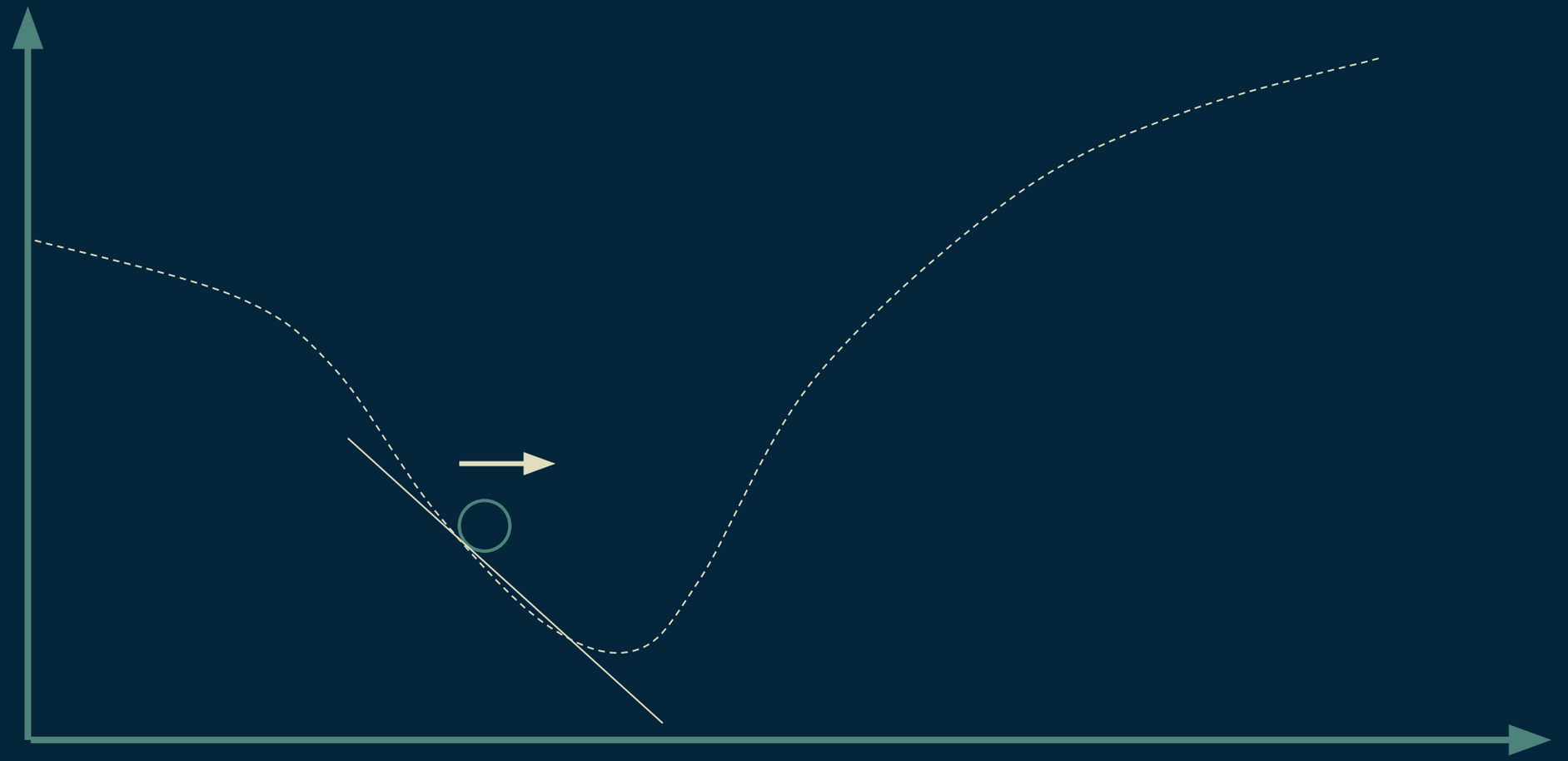
Gradient descent



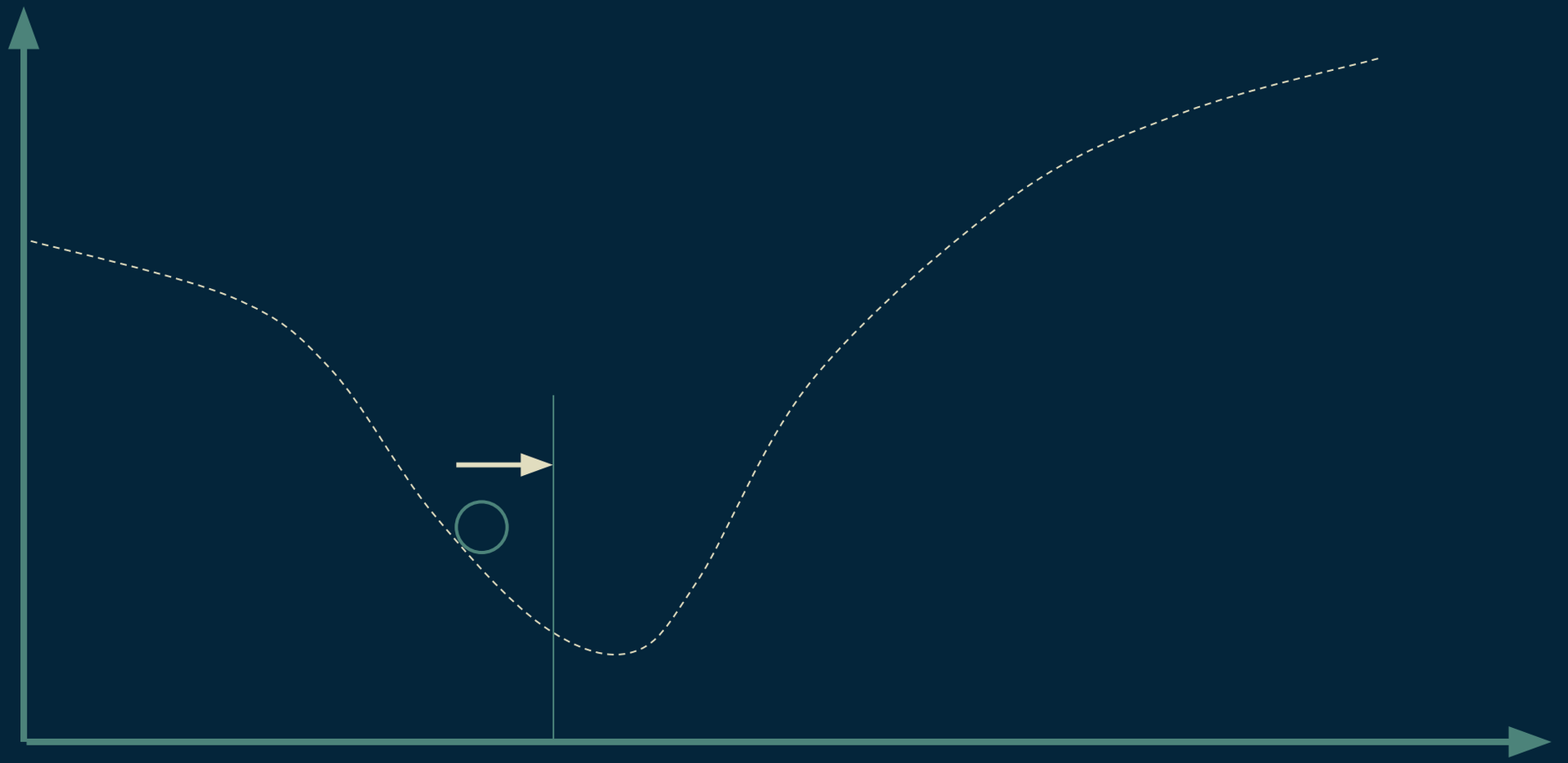
Gradient descent



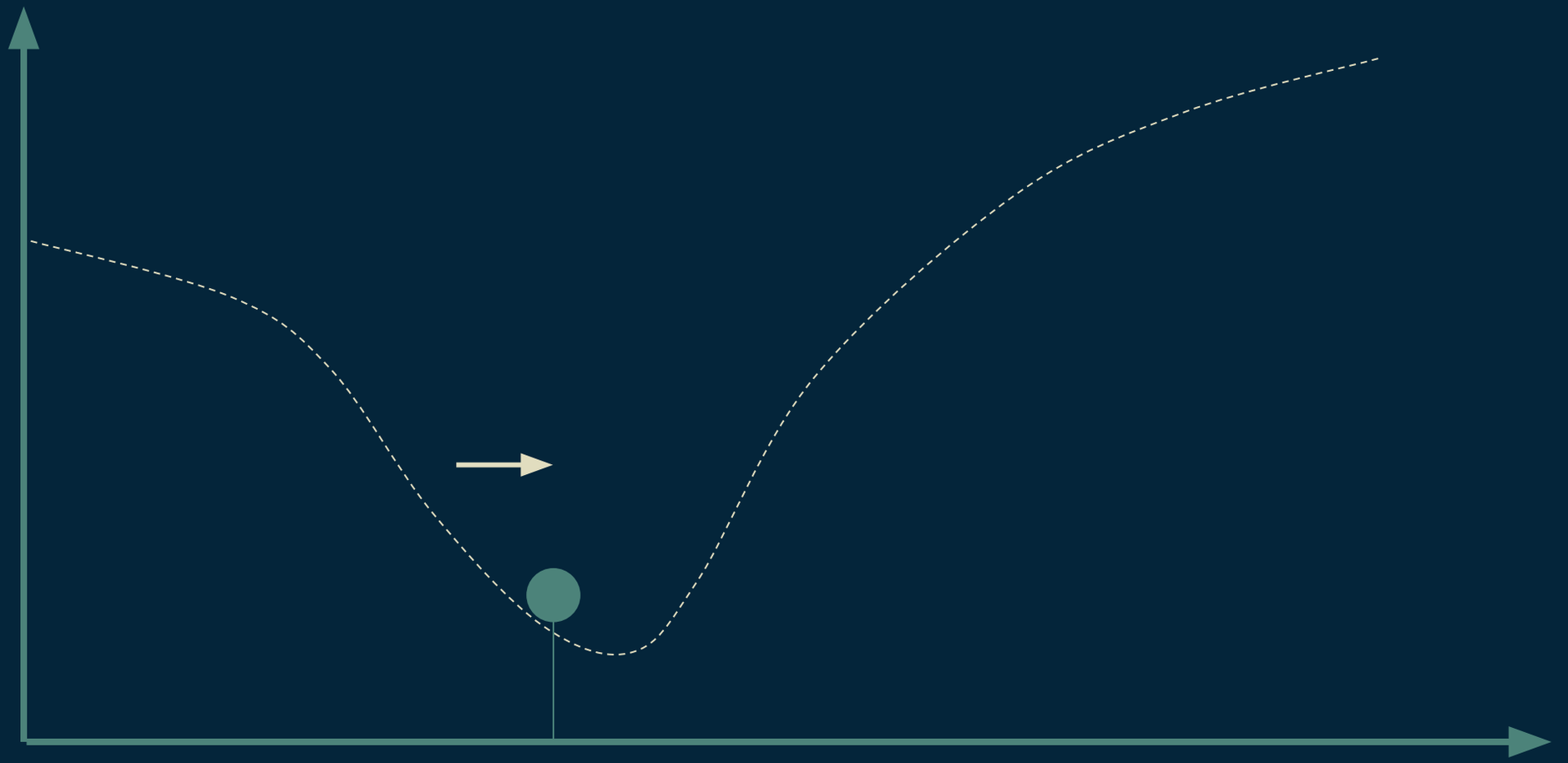
Gradient descent



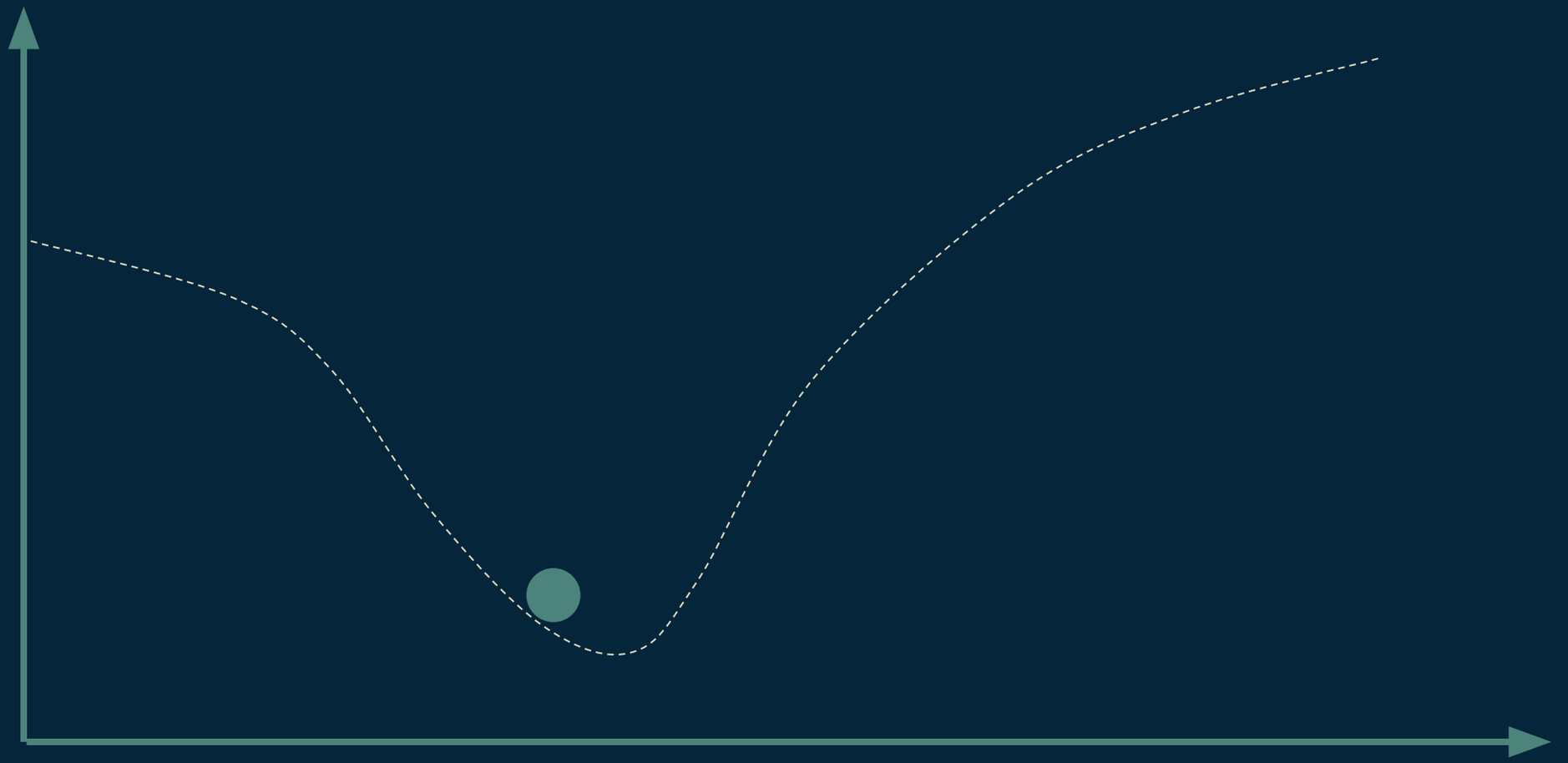
Gradient descent



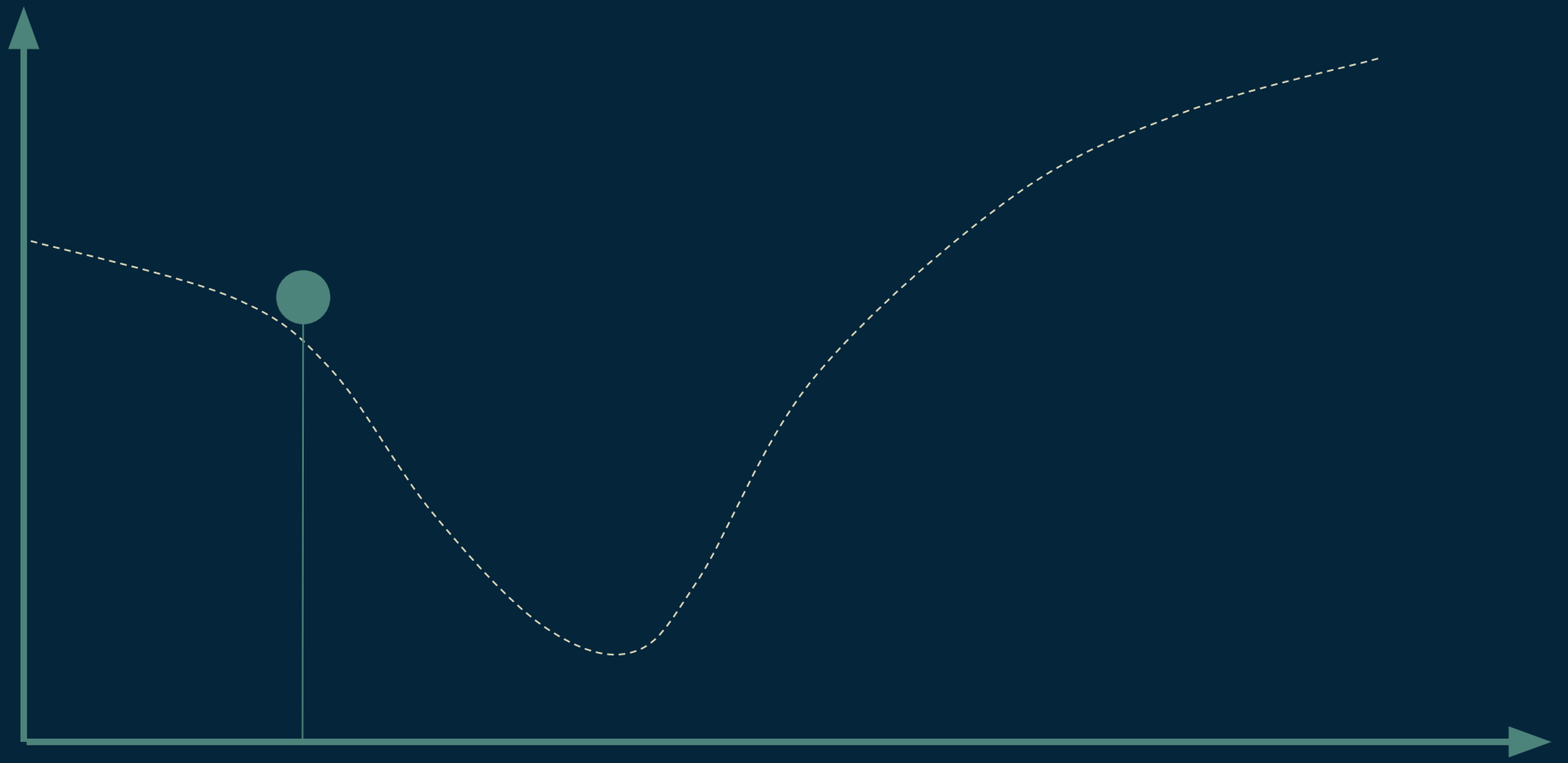
Gradient descent



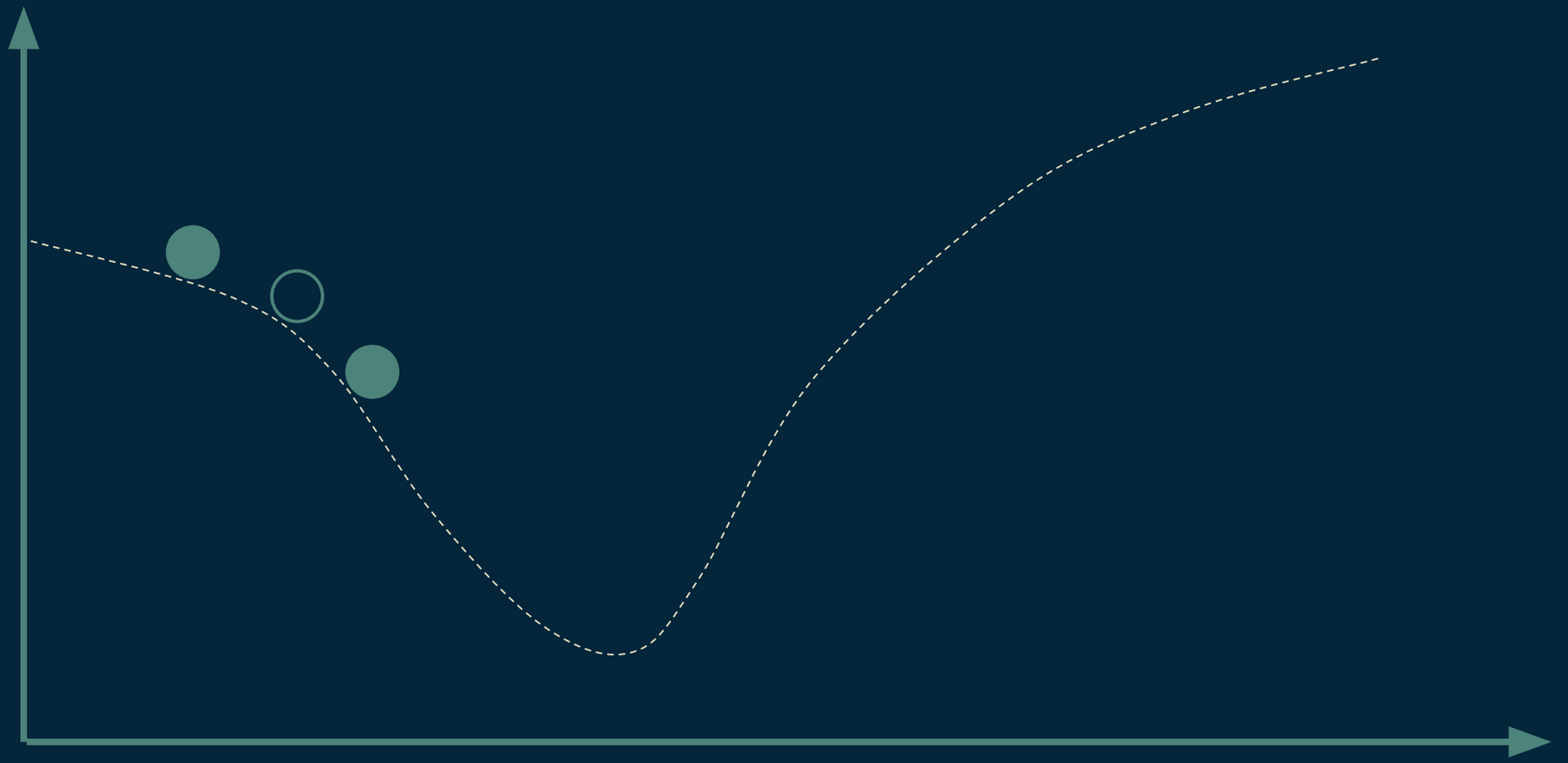
Gradient descent



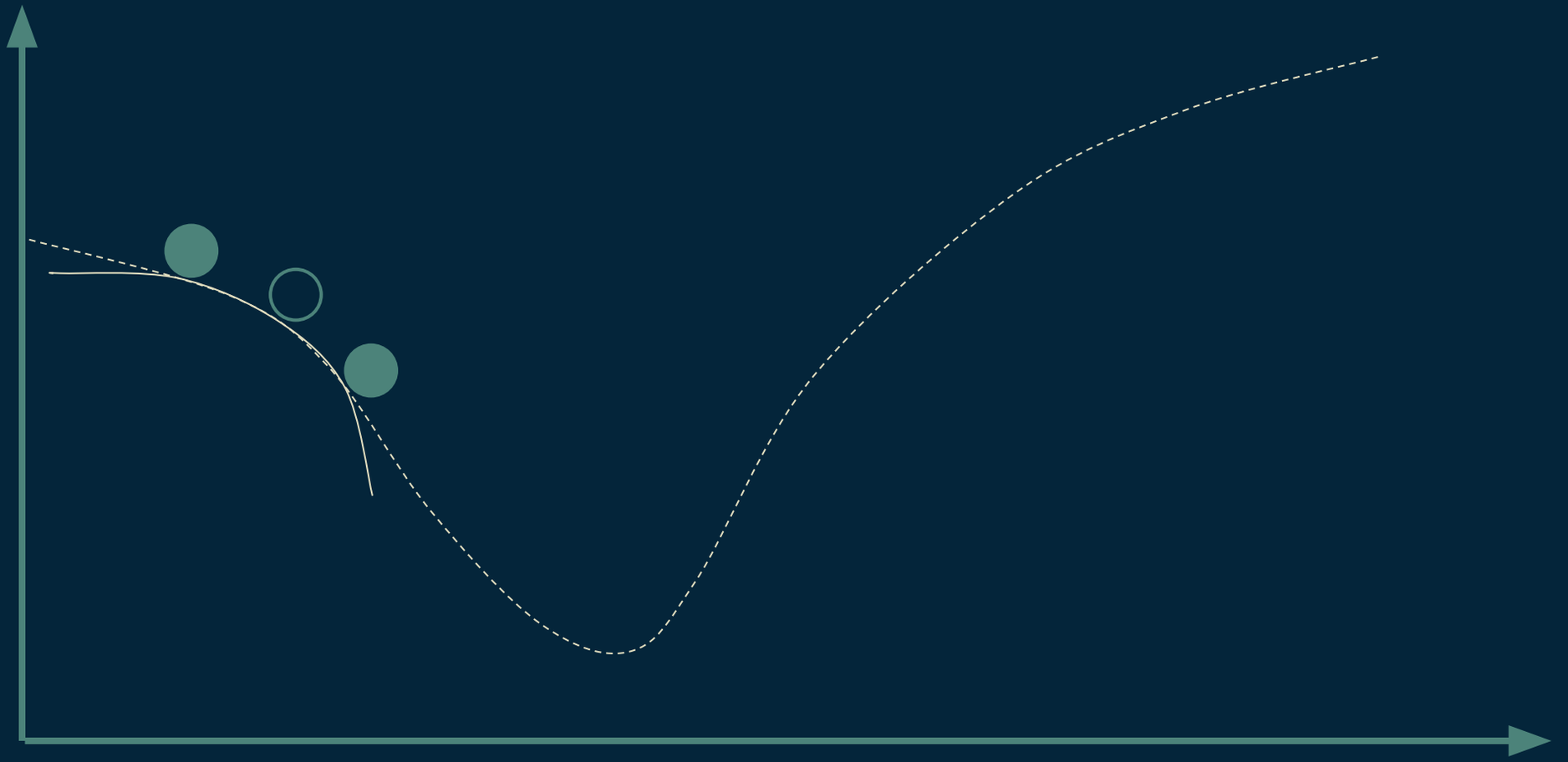
Gradient descent with curvature



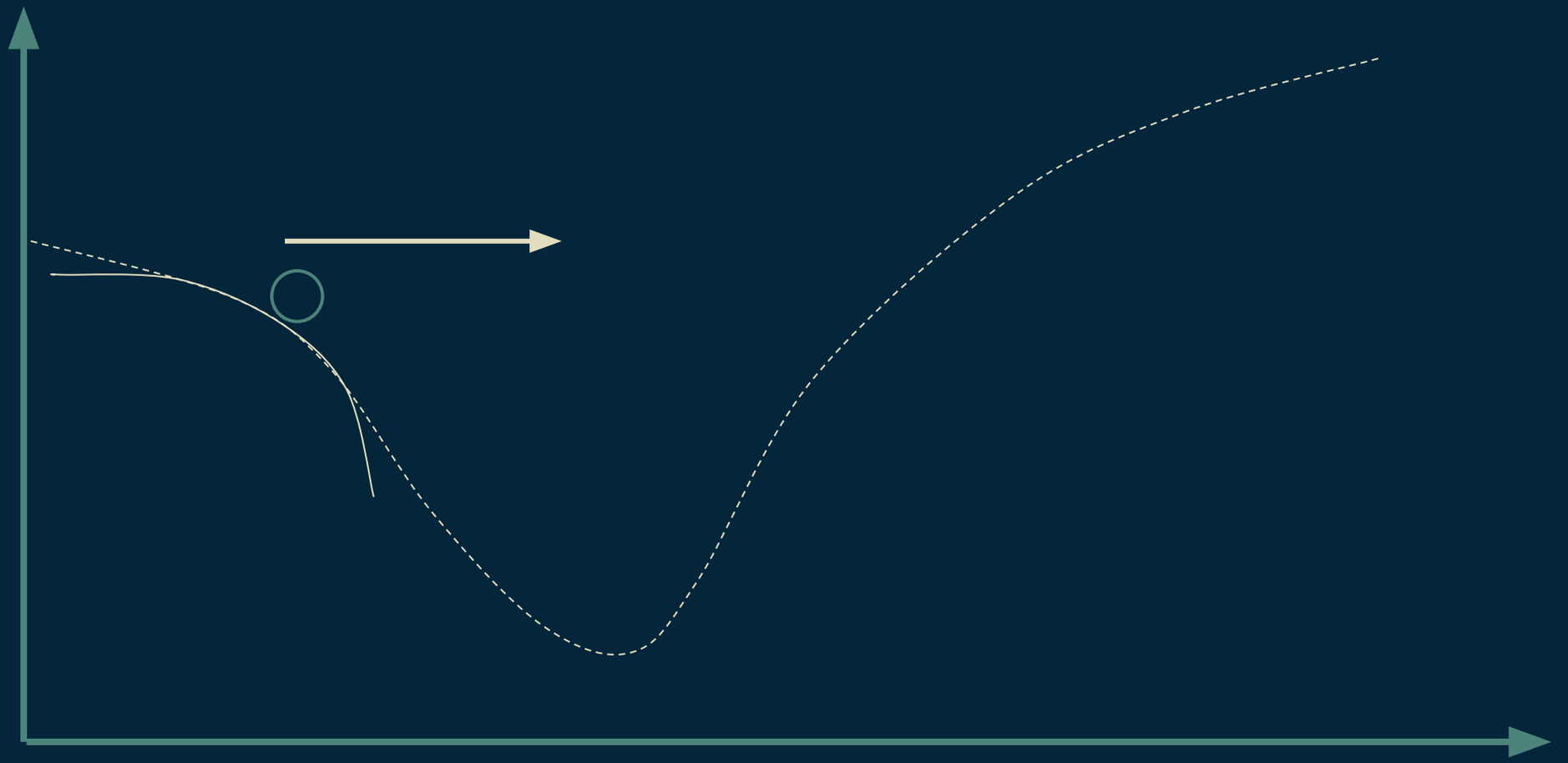
Gradient descent with curvature



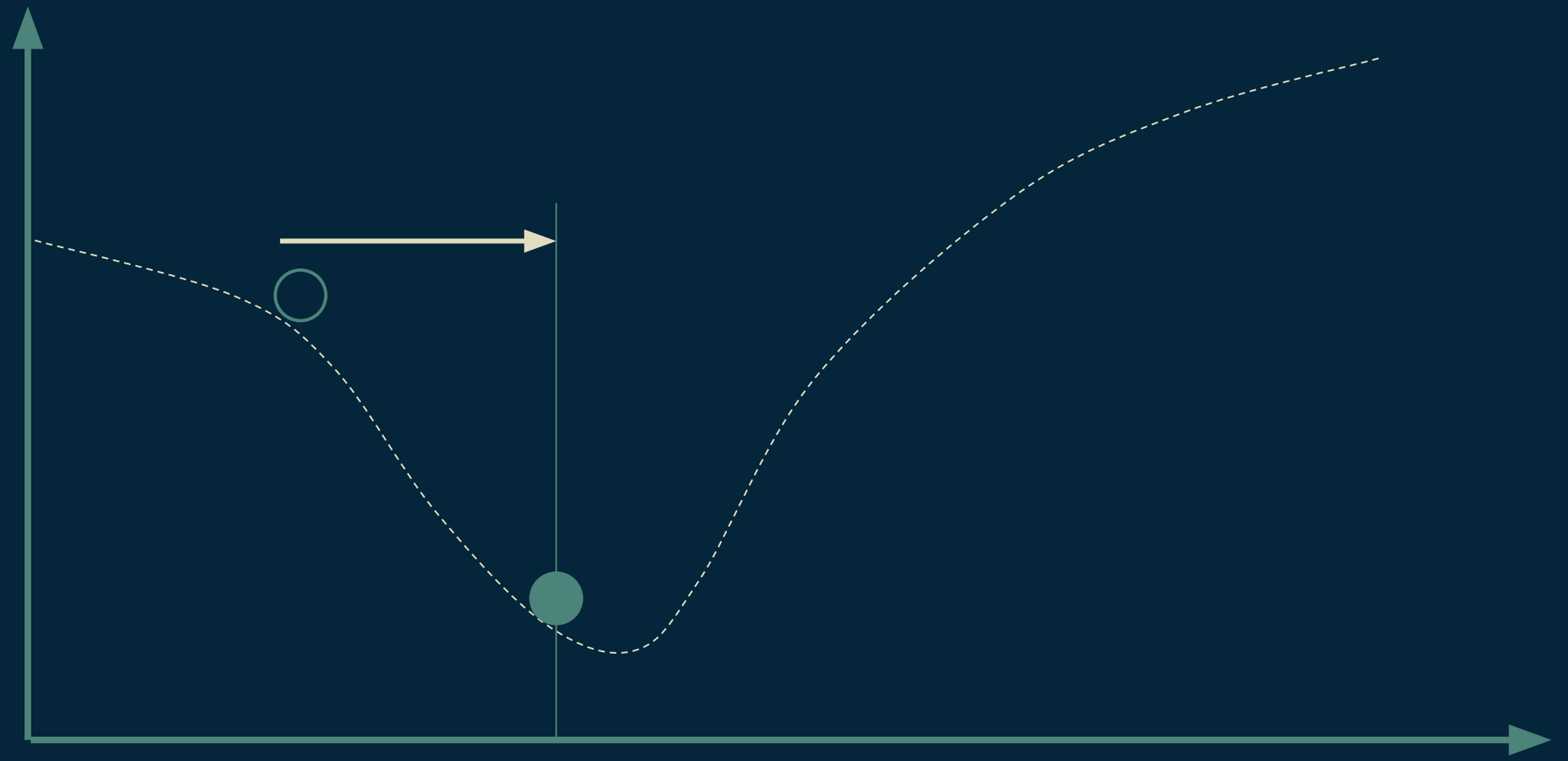
Gradient descent with curvature



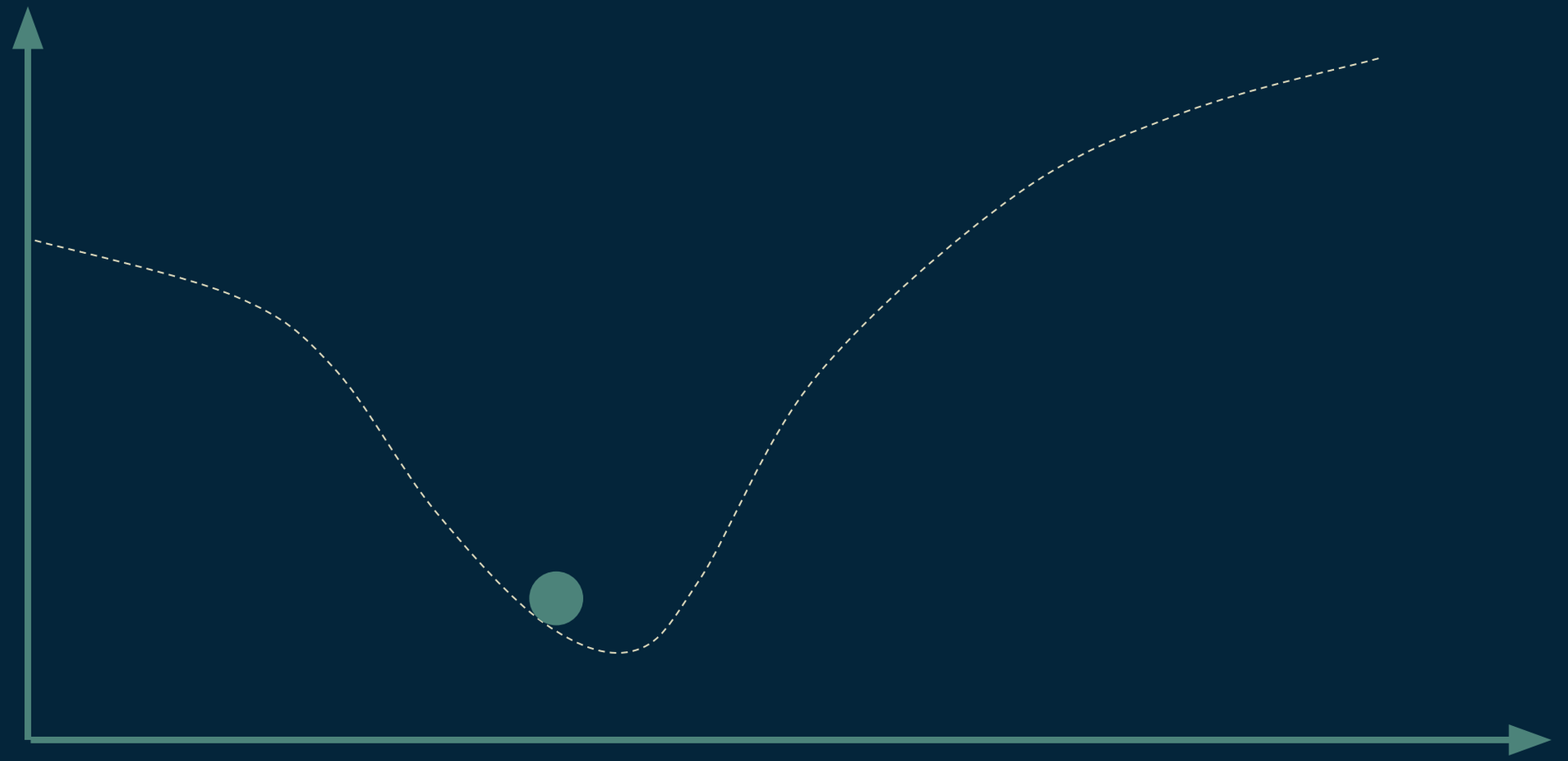
Gradient descent with curvature



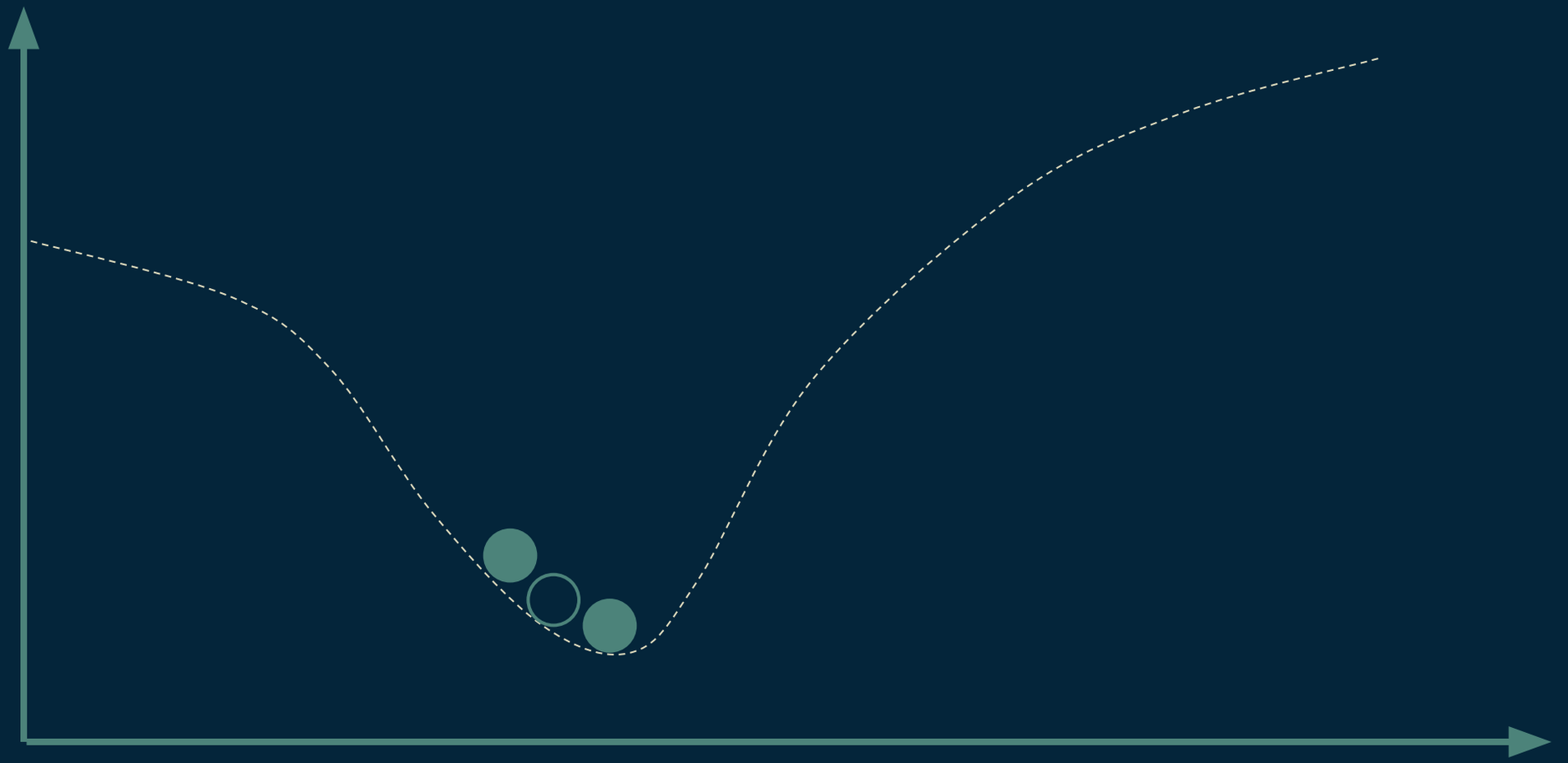
Gradient descent with curvature



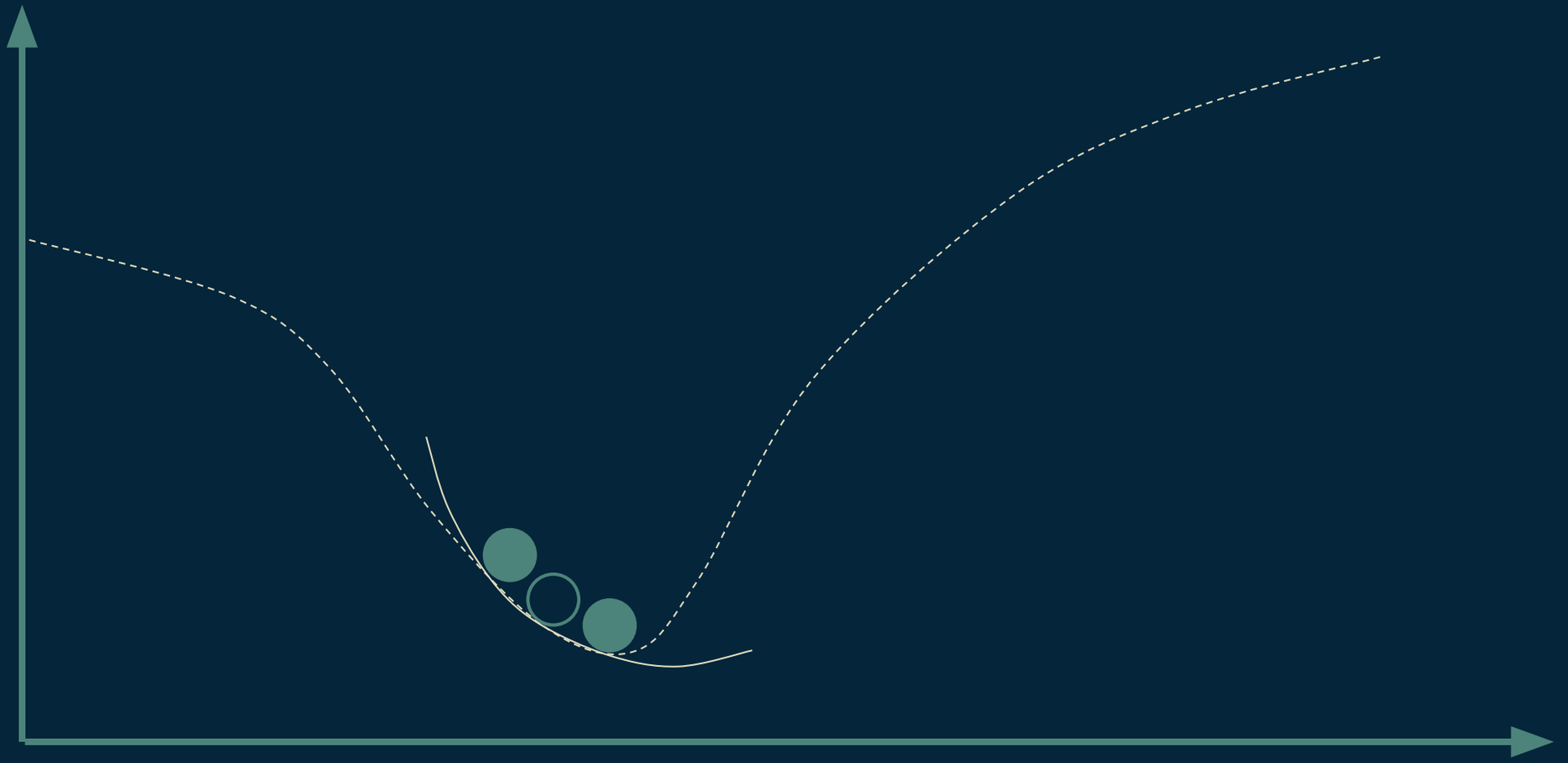
Gradient descent with curvature



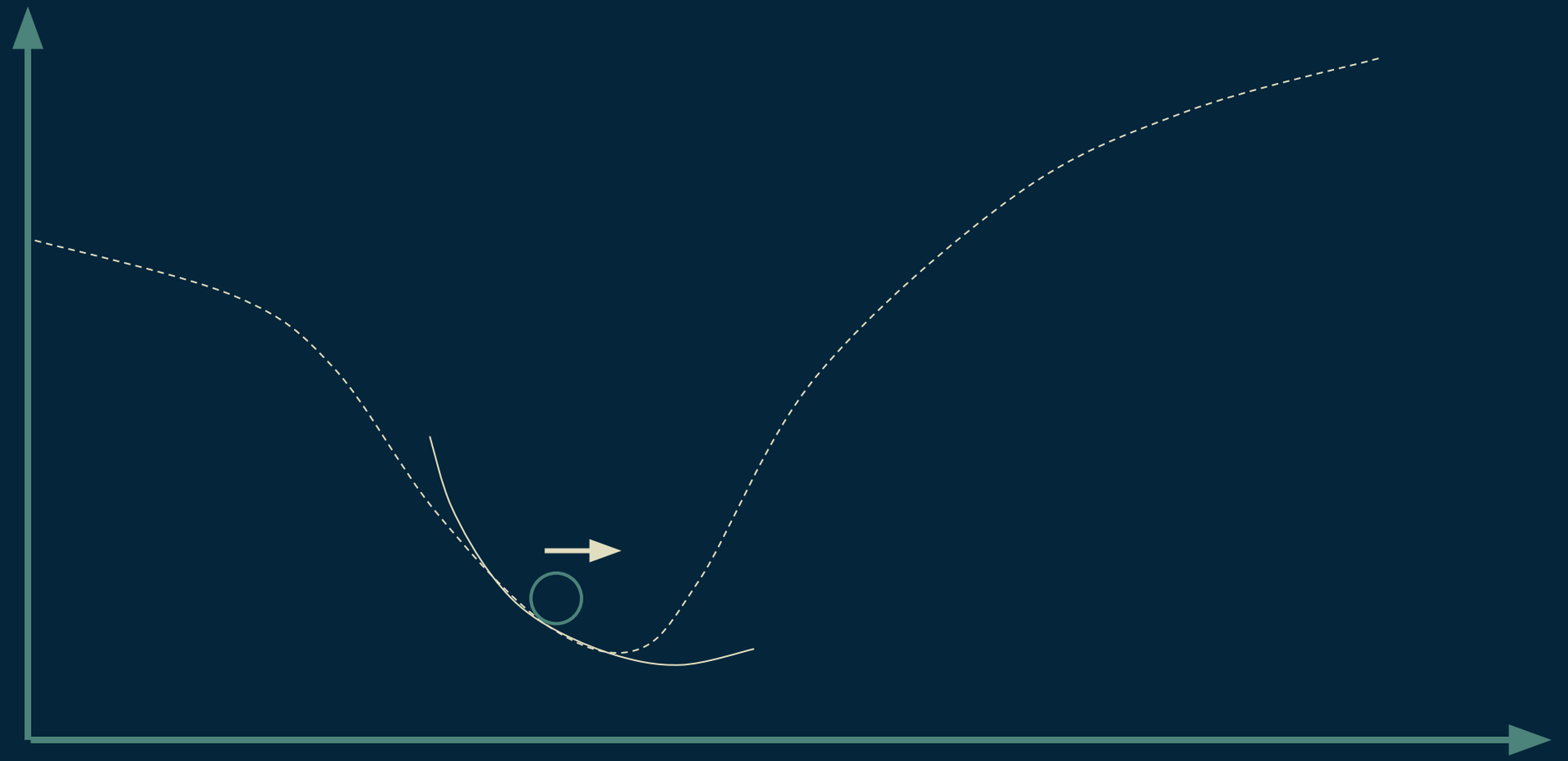
Gradient descent with curvature



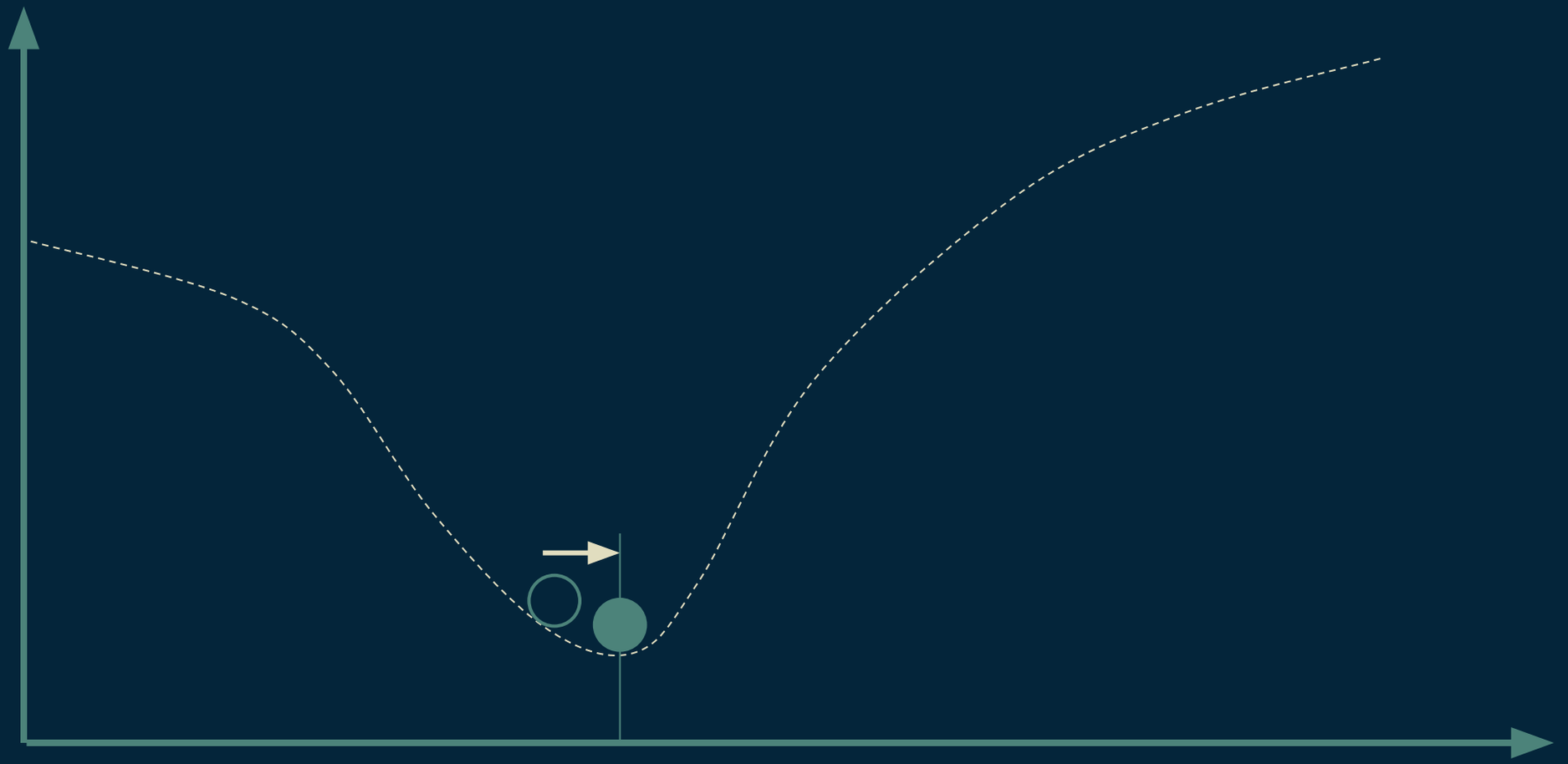
Gradient descent with curvature



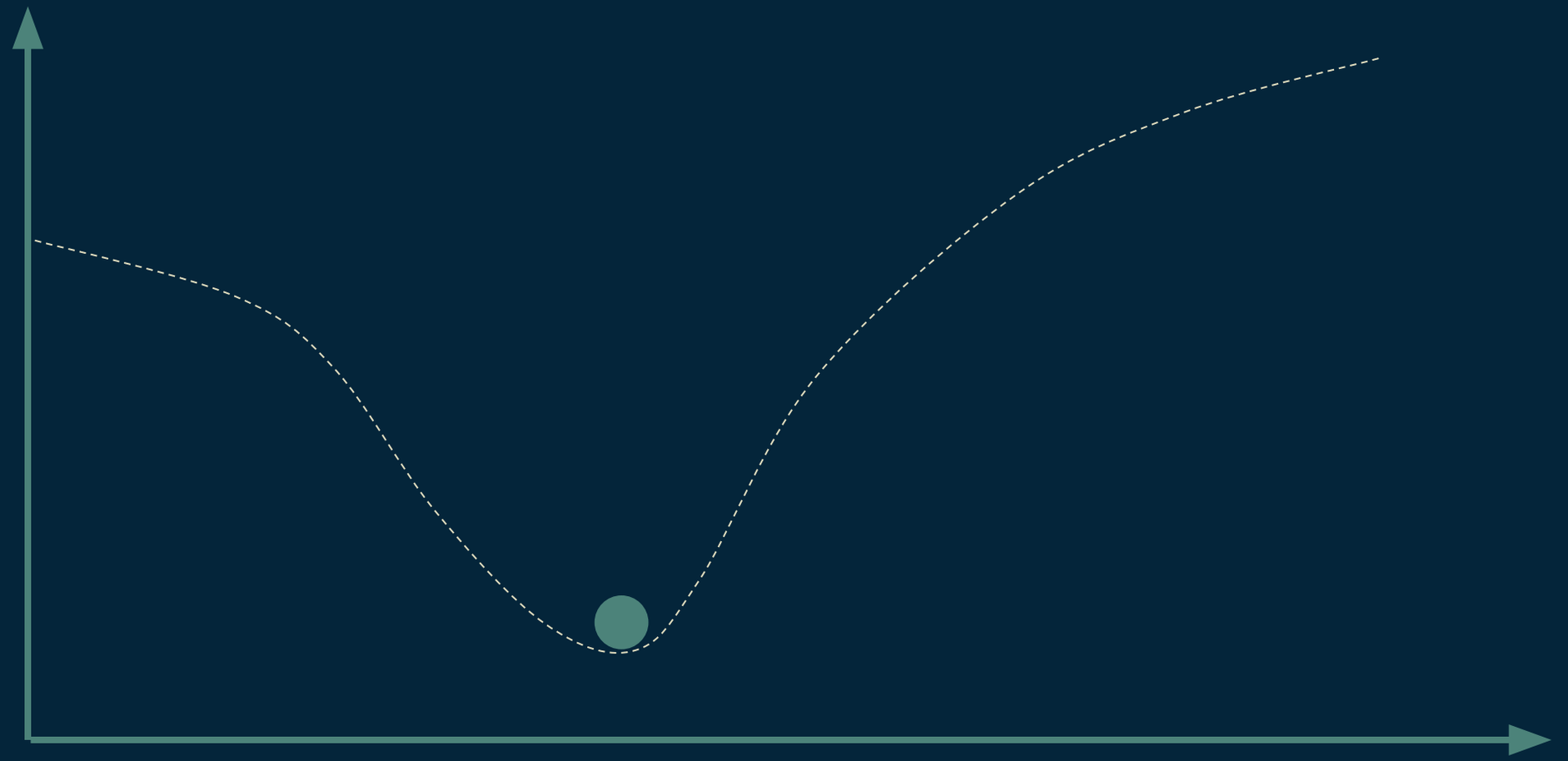
Gradient descent with curvature



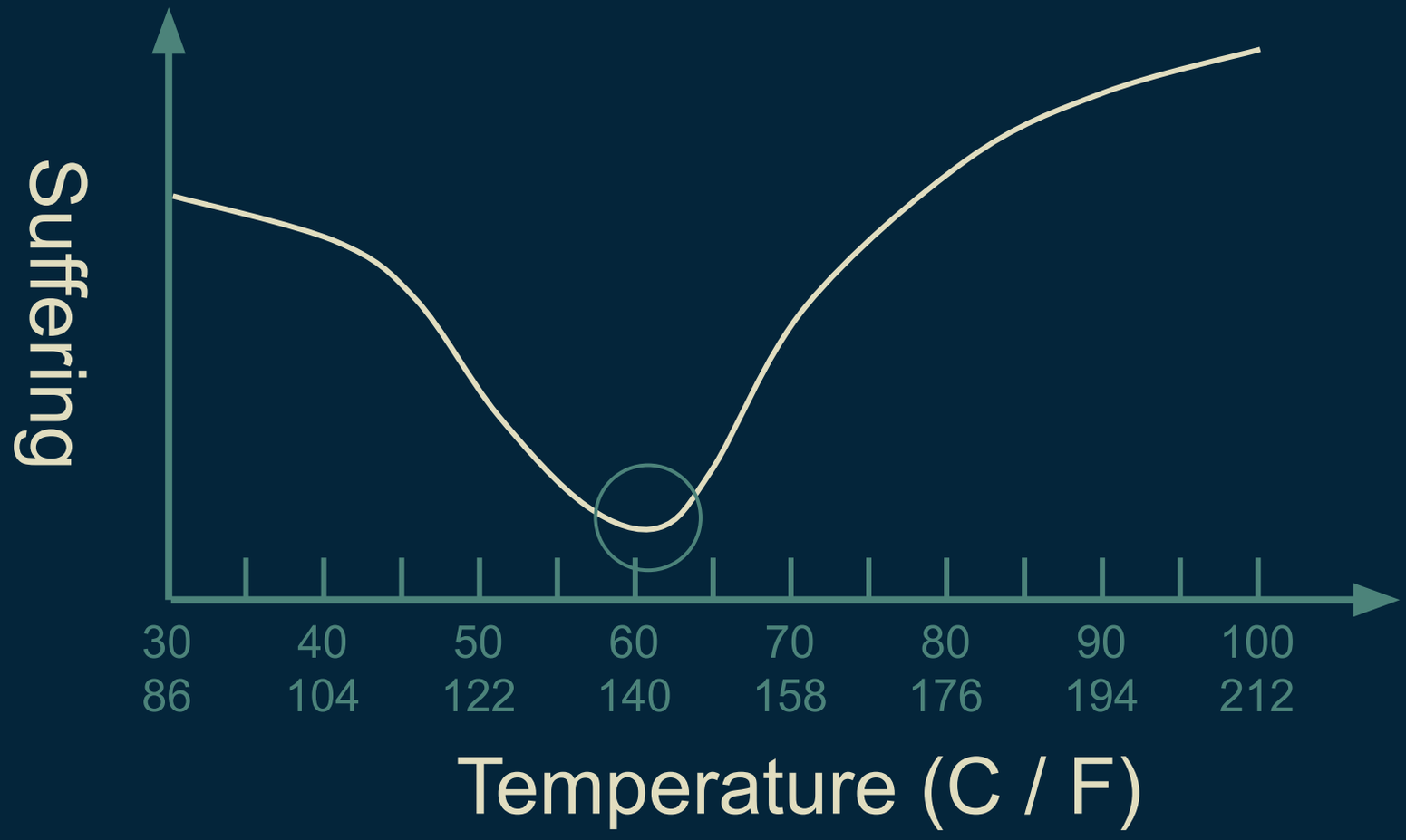
Gradient descent with curvature



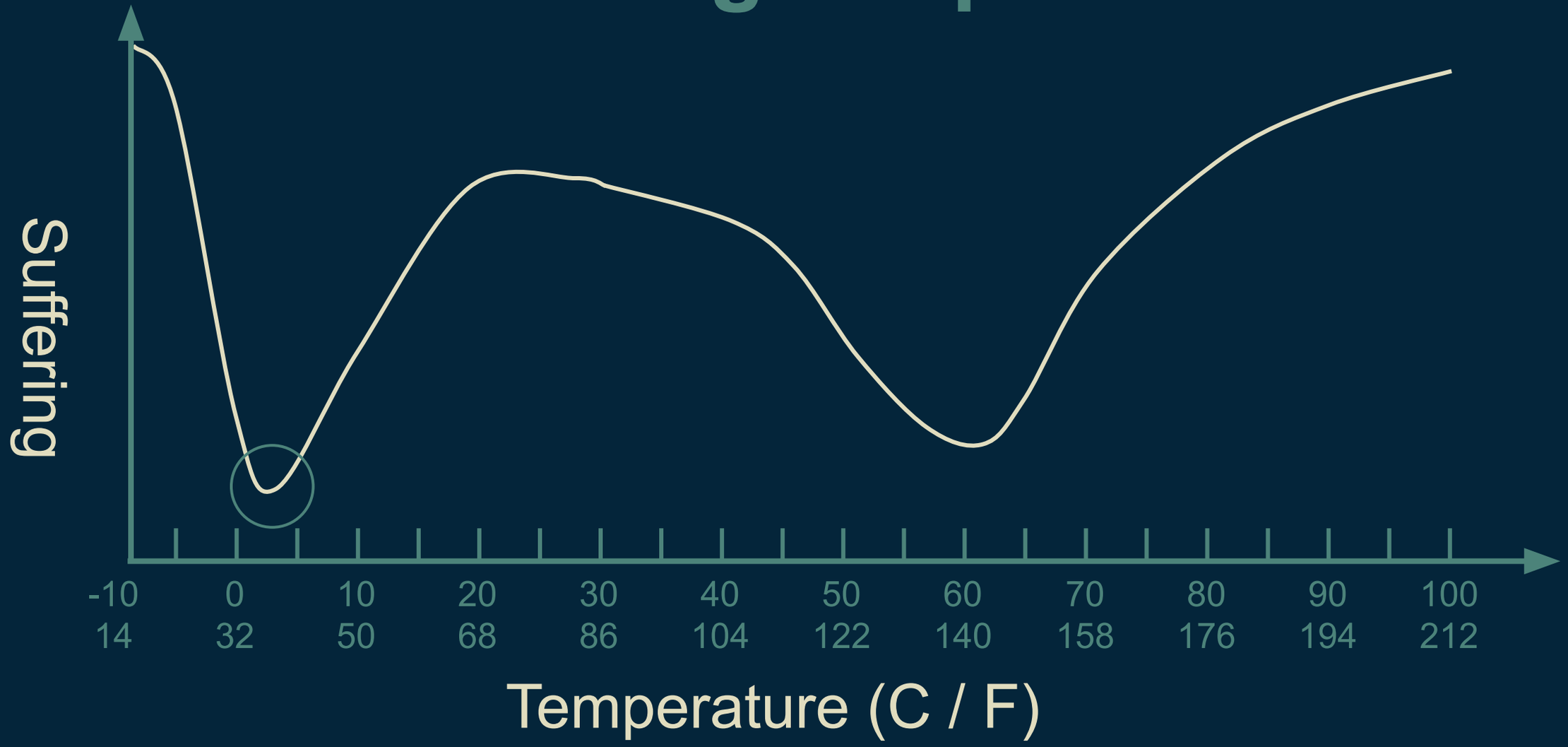
Gradient descent with curvature



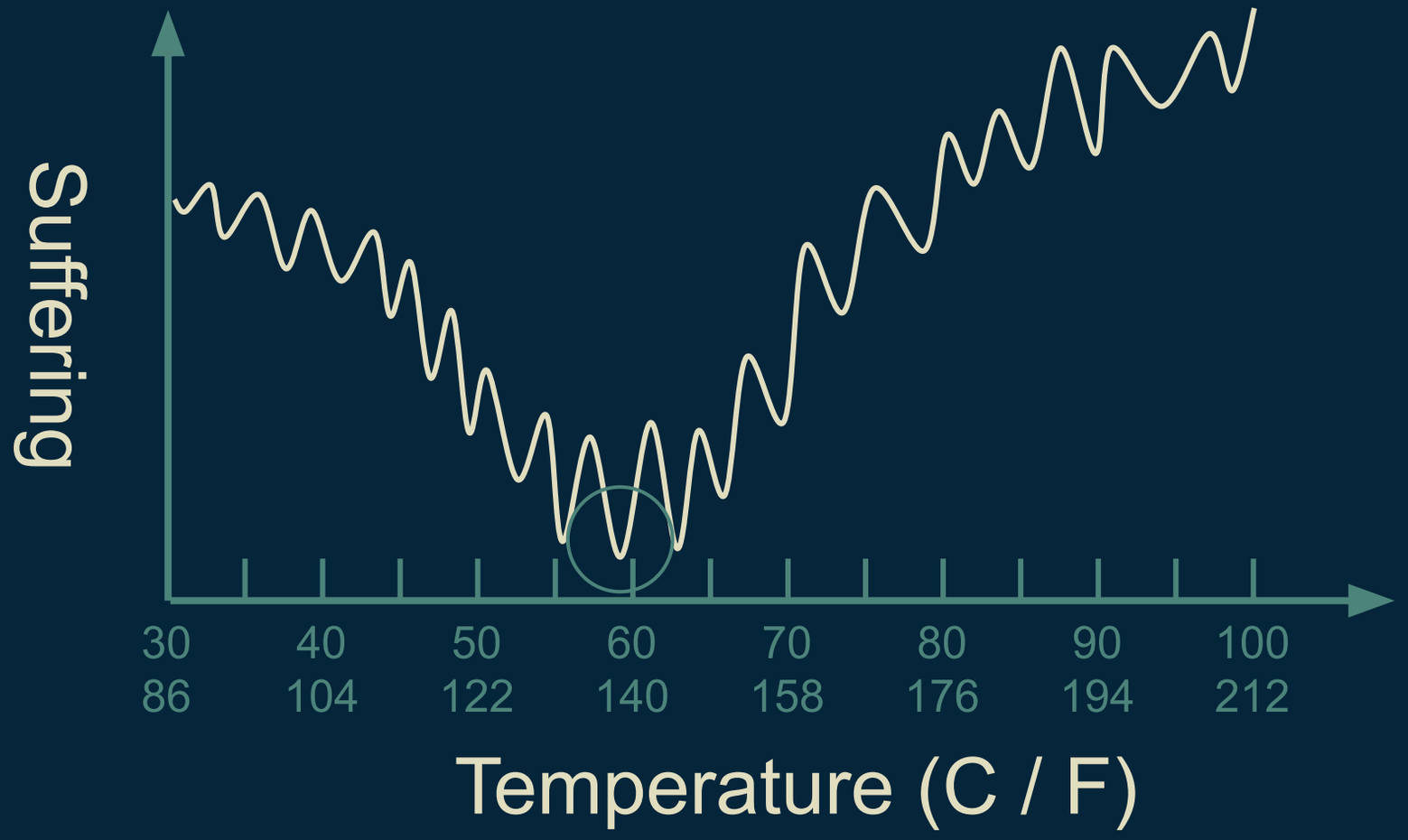
Tea drinking temperature



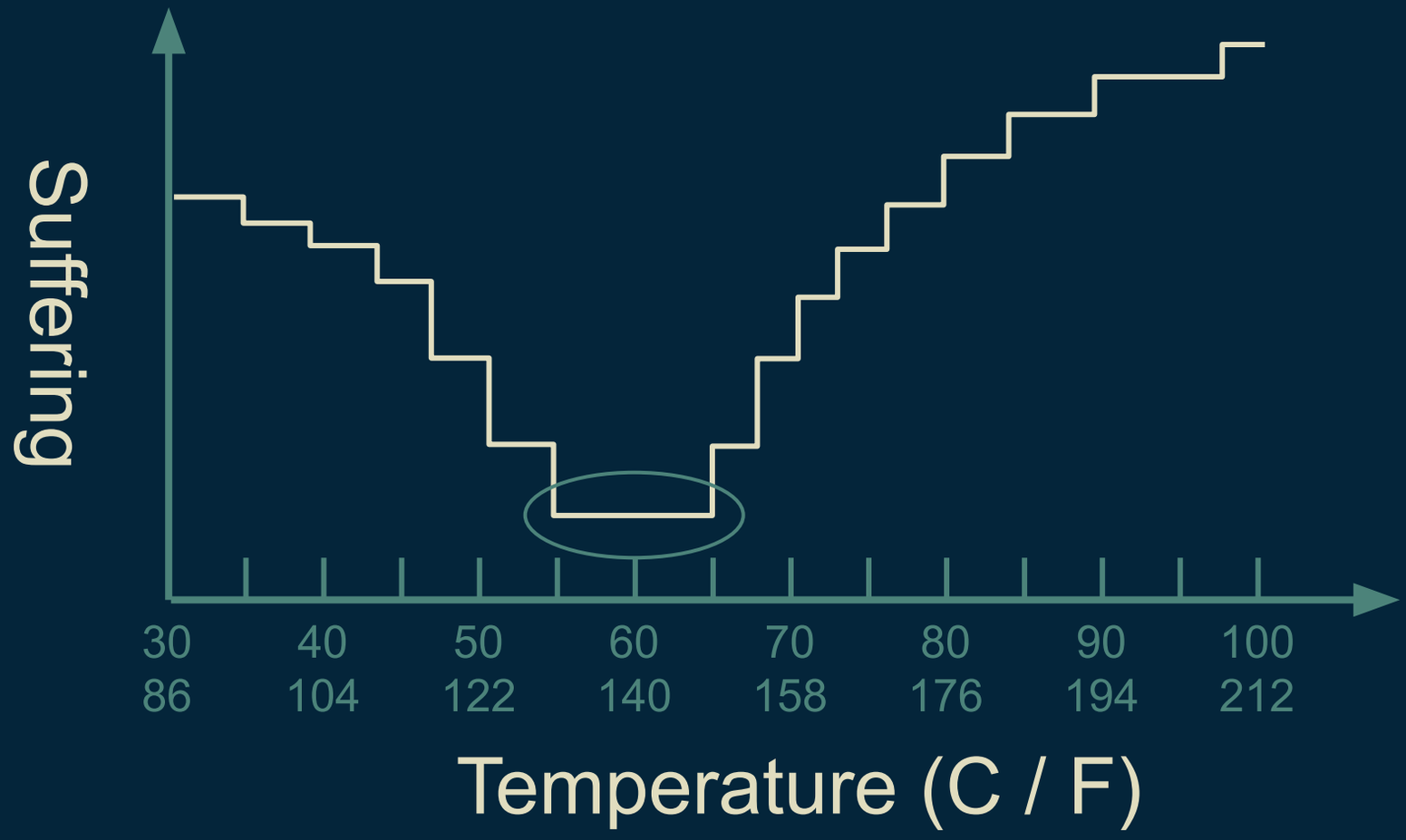
Tea drinking temperature



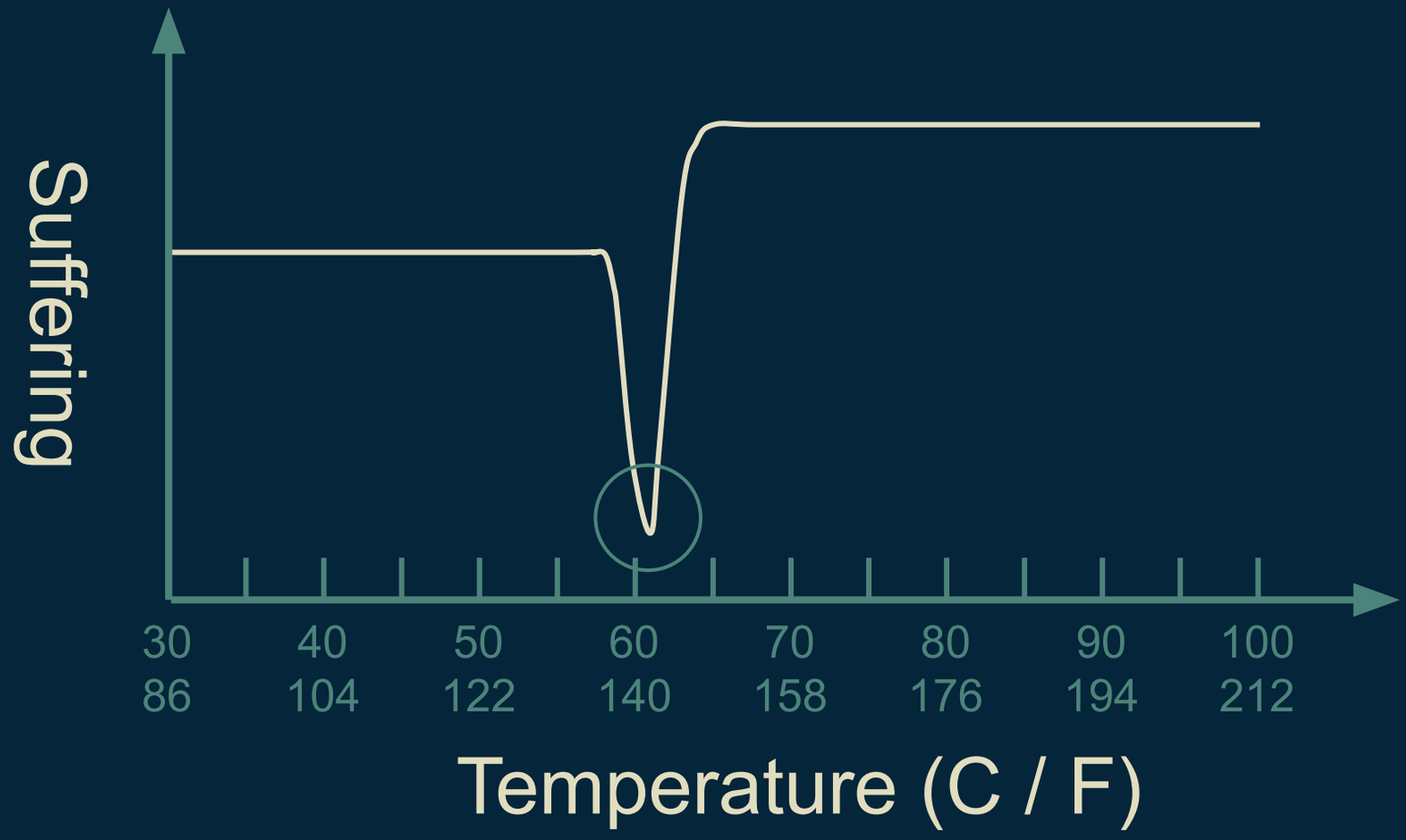
Tea drinking temperature



Tea drinking temperature



Tea drinking temperature



Assumptions vs. performance

Assumptions vs. performance

Exhaustive
exploration

Assumptions vs. performance

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

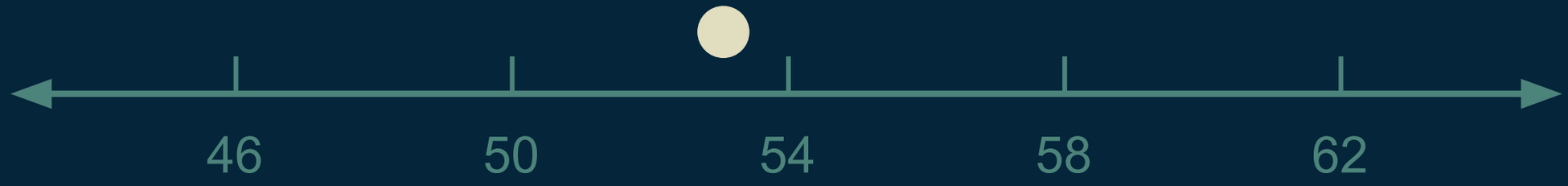
Few assumptions
Robust
Expensive to compute

More assumptions
Sensitive
Efficient to compute

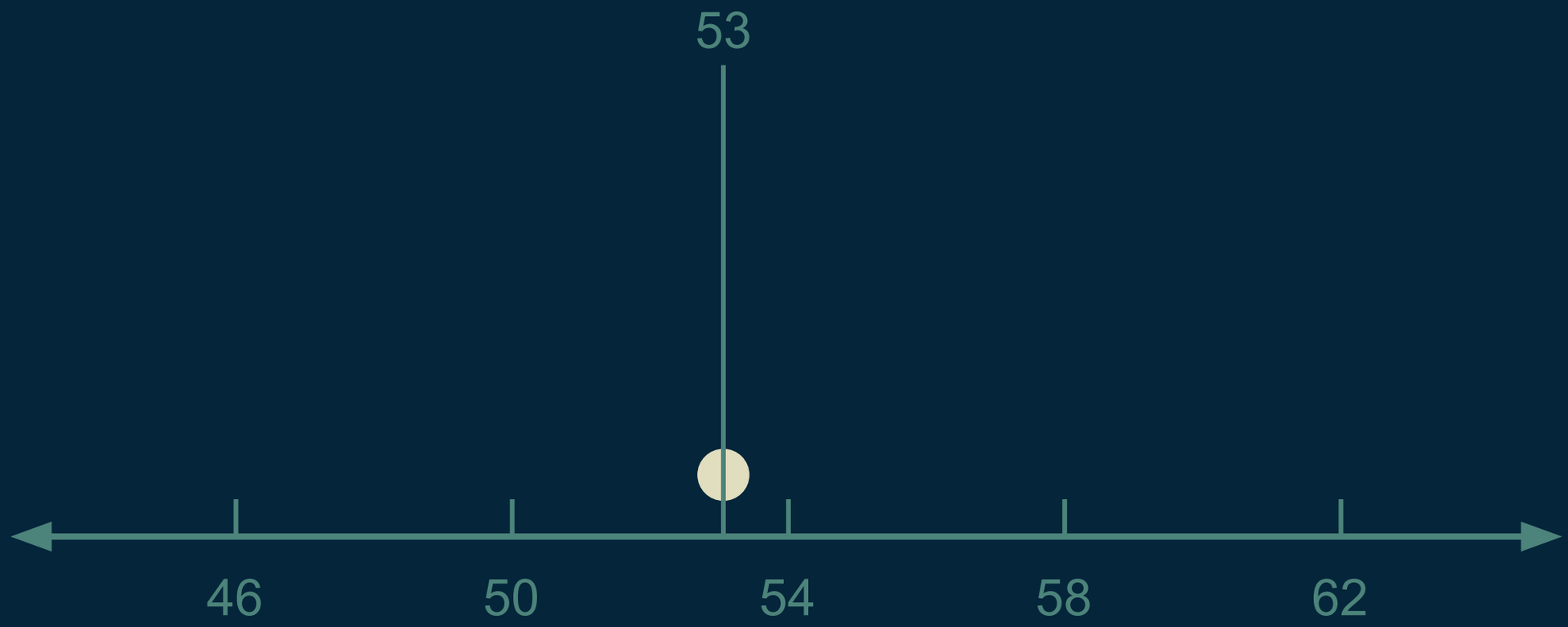
From data to model:
How many M&Ms in a bag?

Number of M&Ms in a bag

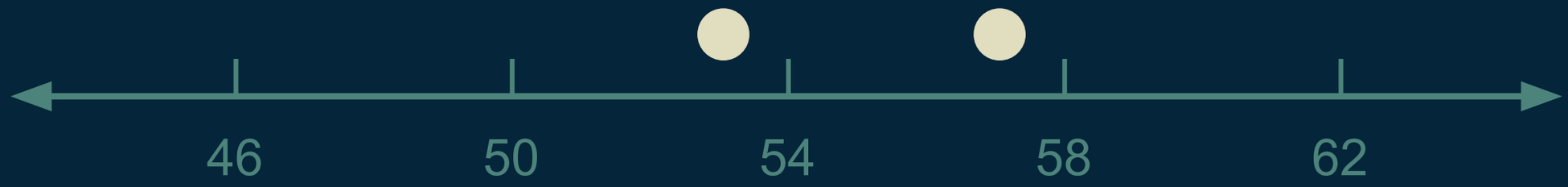
How many M&Ms in a bag?



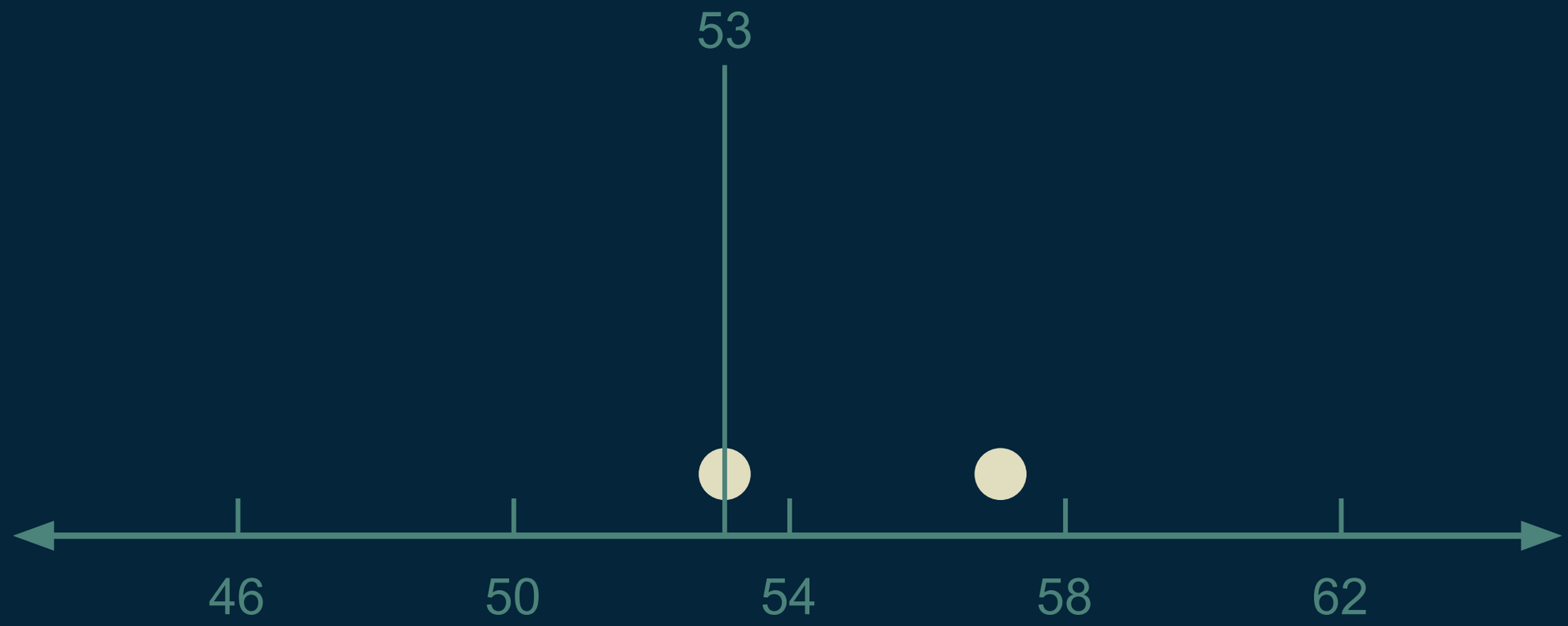
How many M&Ms in a bag?



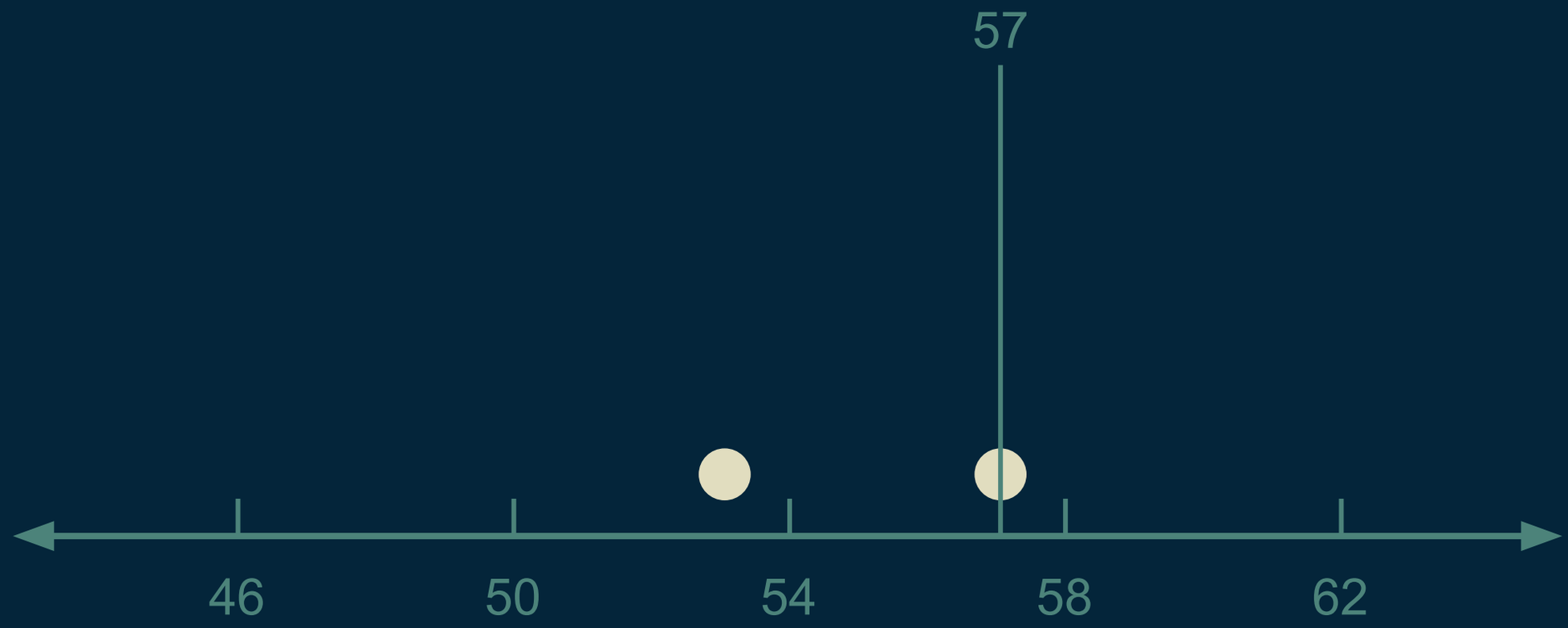
How many M&Ms in a bag?



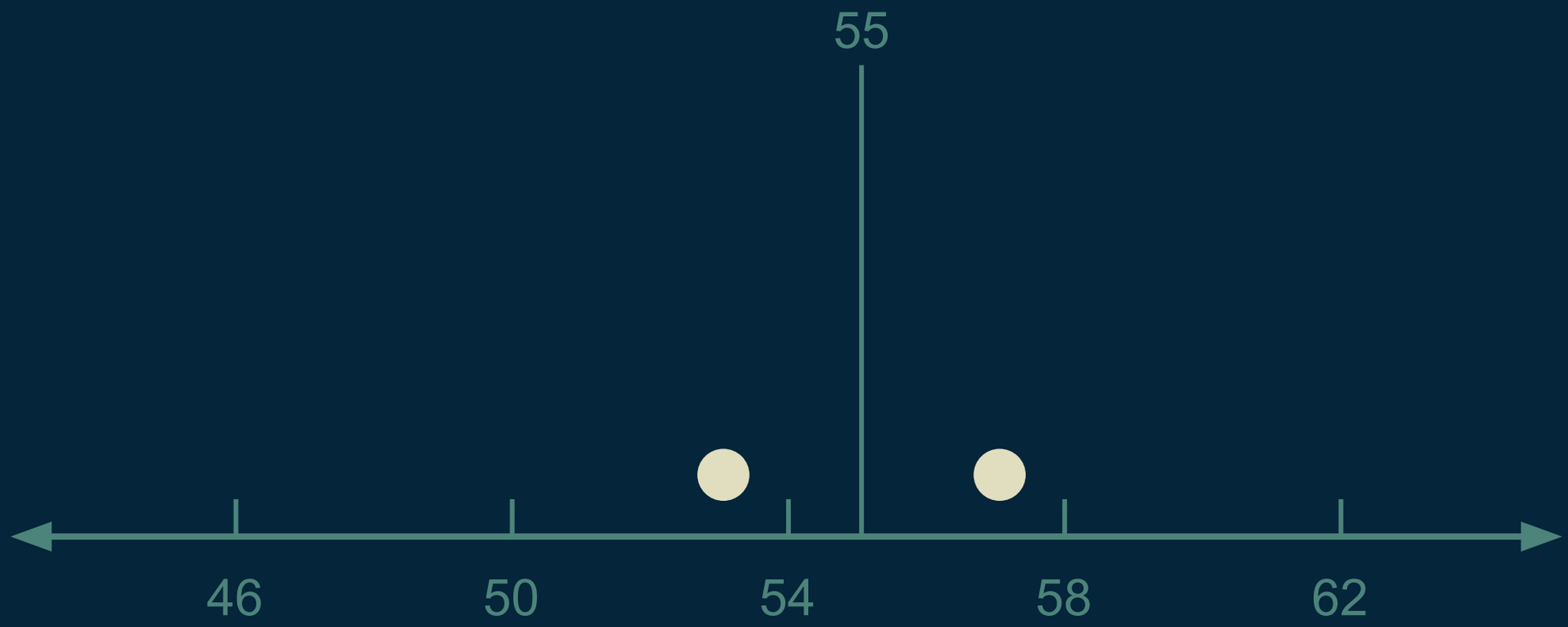
How many M&Ms in a bag?



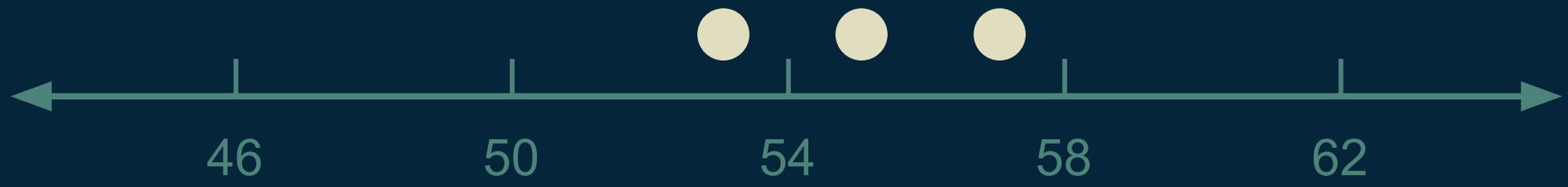
How many M&Ms in a bag?



How many M&Ms in a bag?

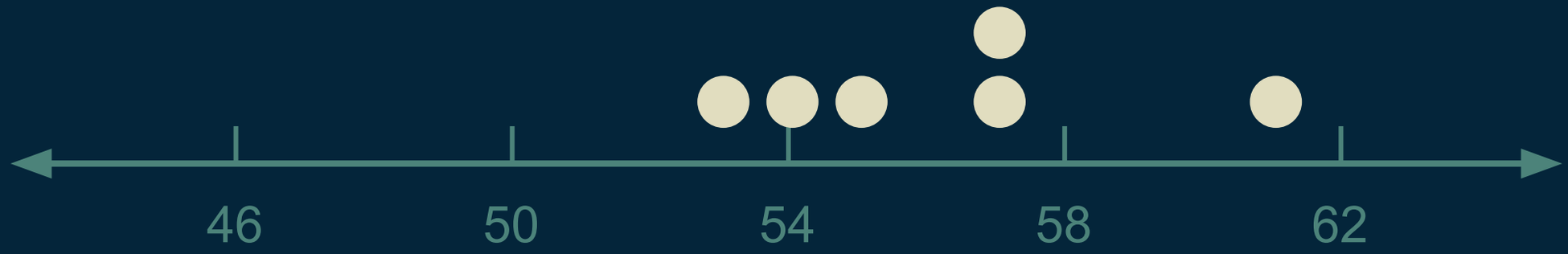


How many M&Ms in a bag?

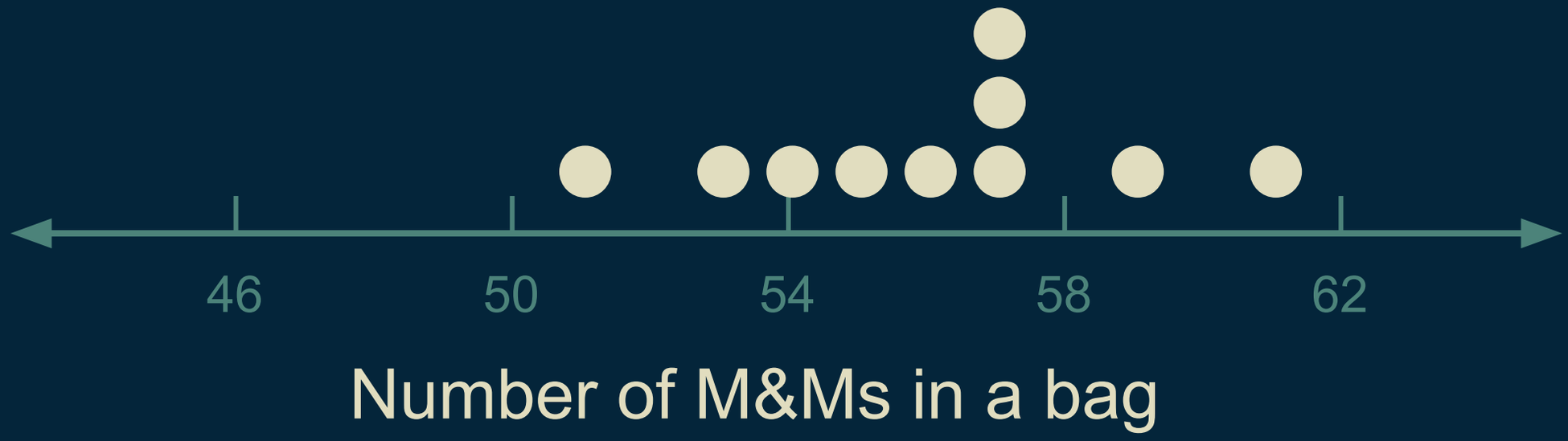


Number of M&Ms in a bag

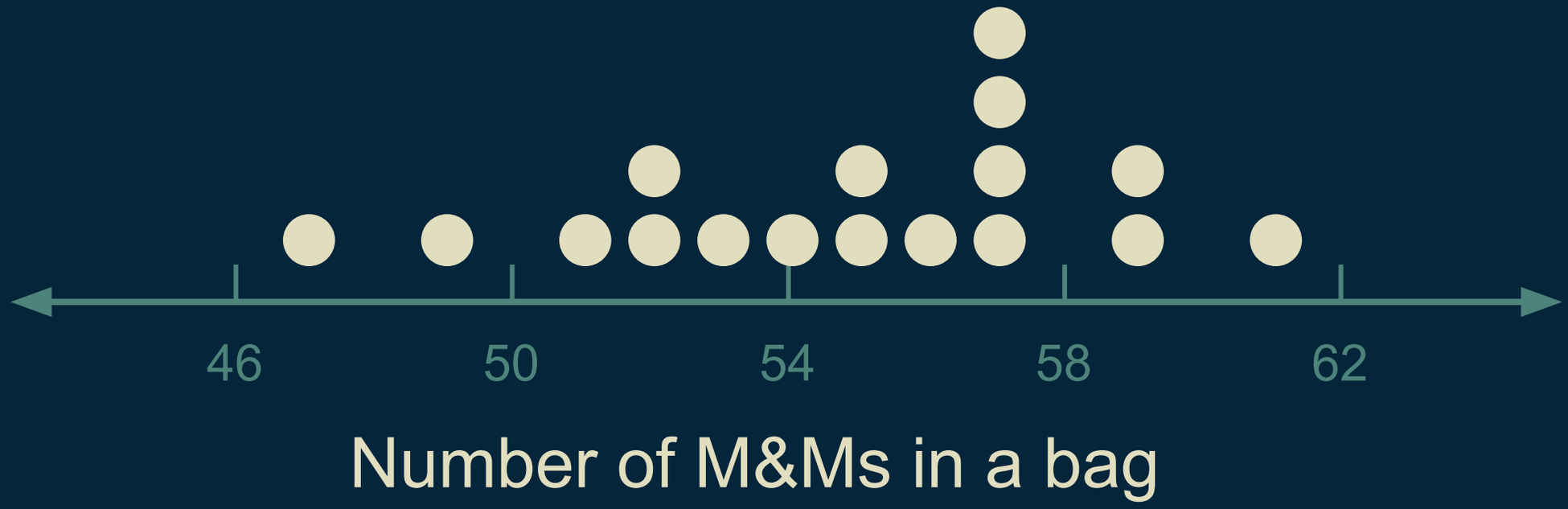
How many M&Ms in a bag?



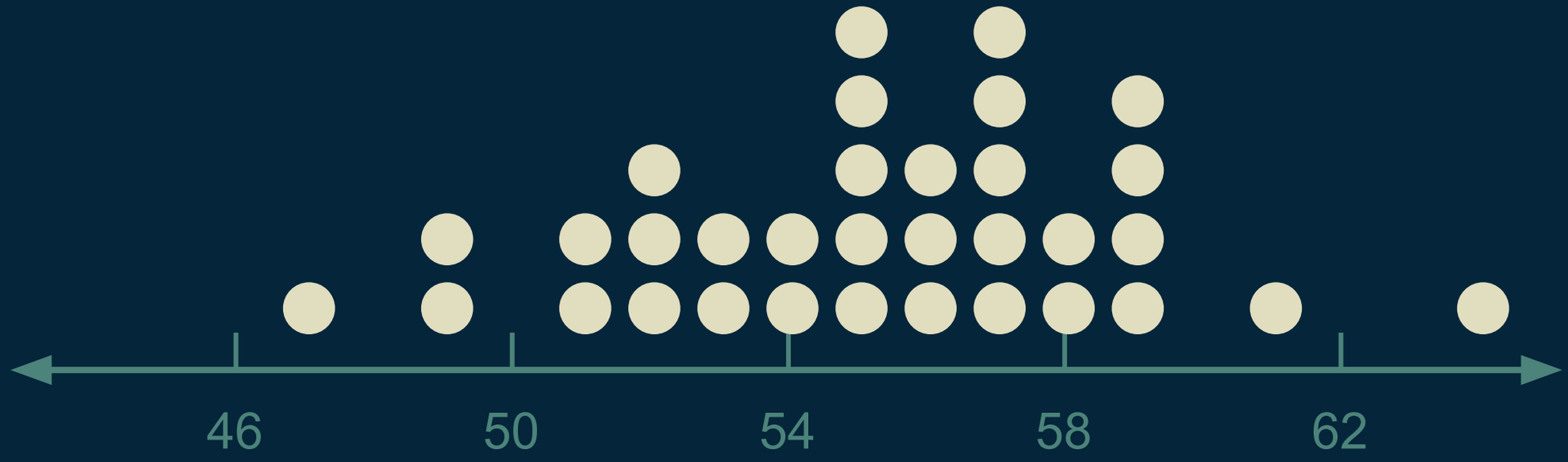
How many M&Ms in a bag?



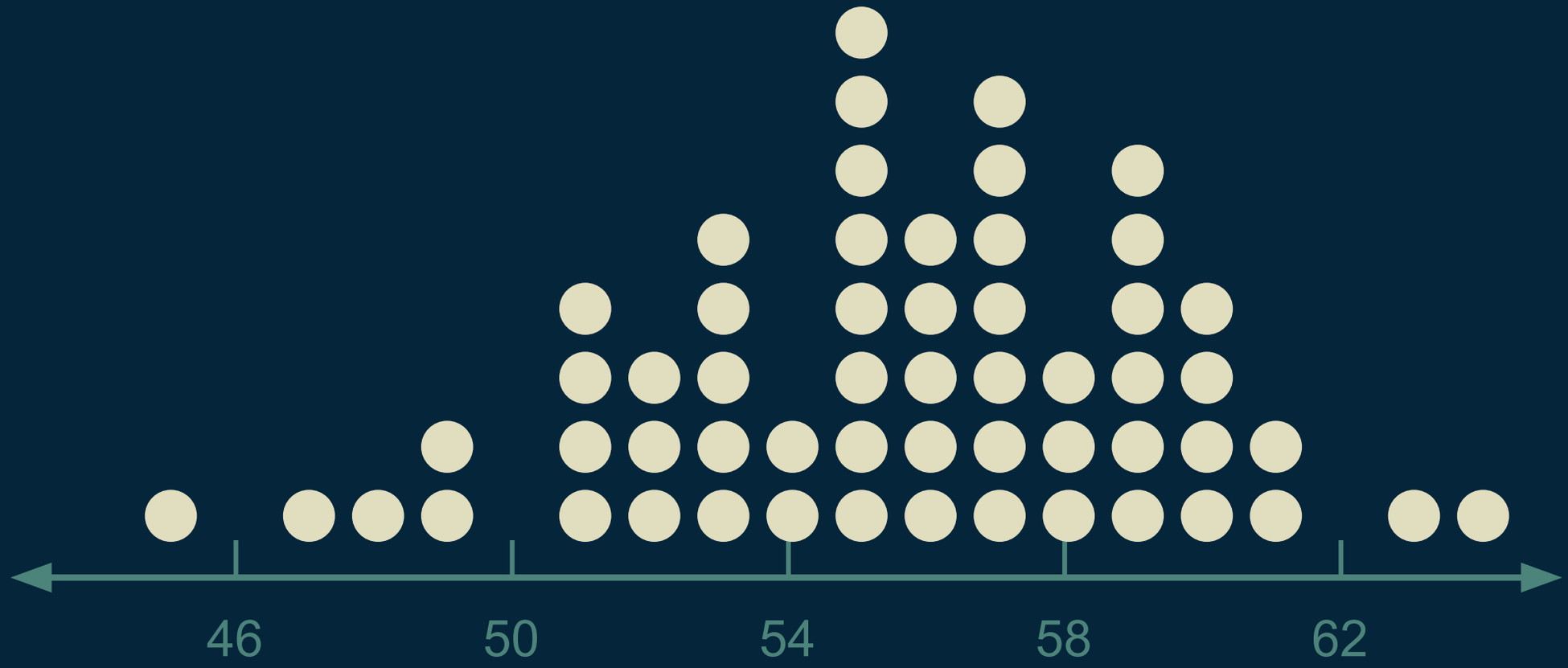
How many M&Ms in a bag?



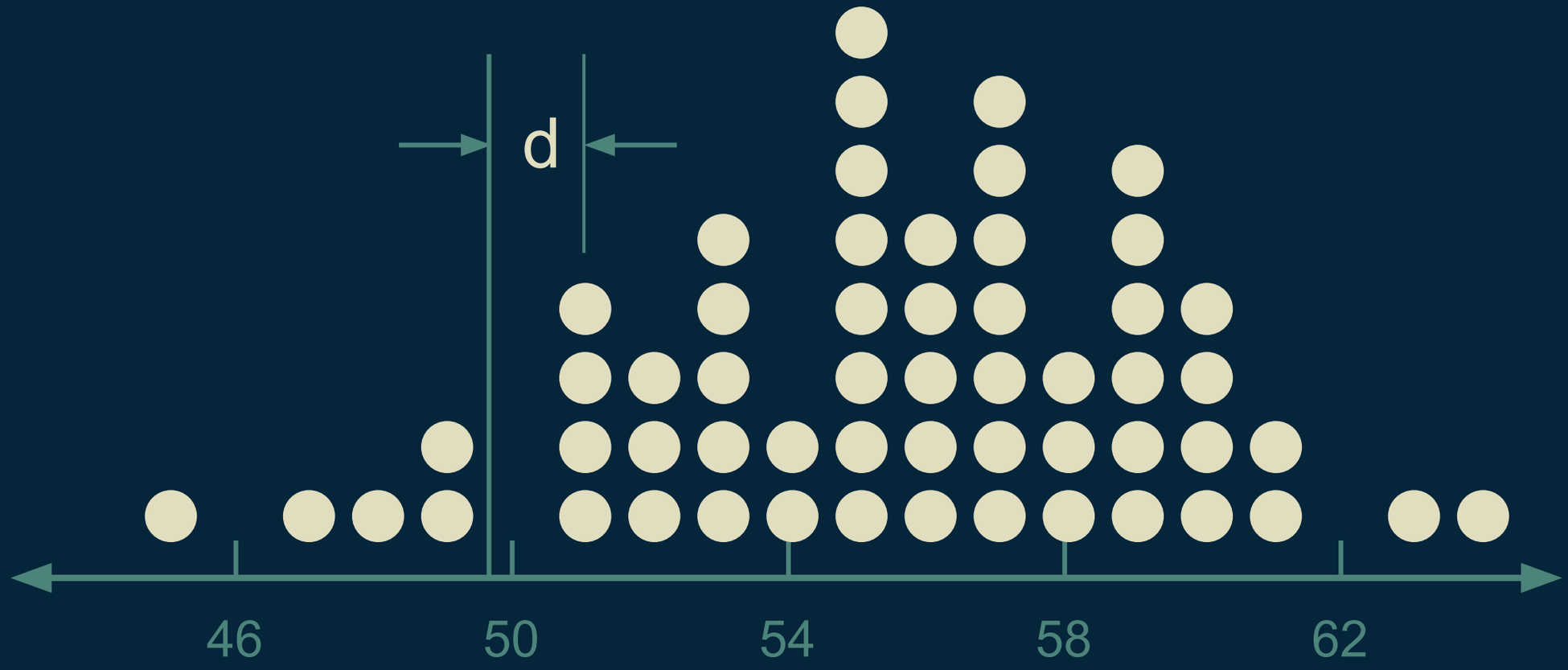
How many M&Ms in a bag?



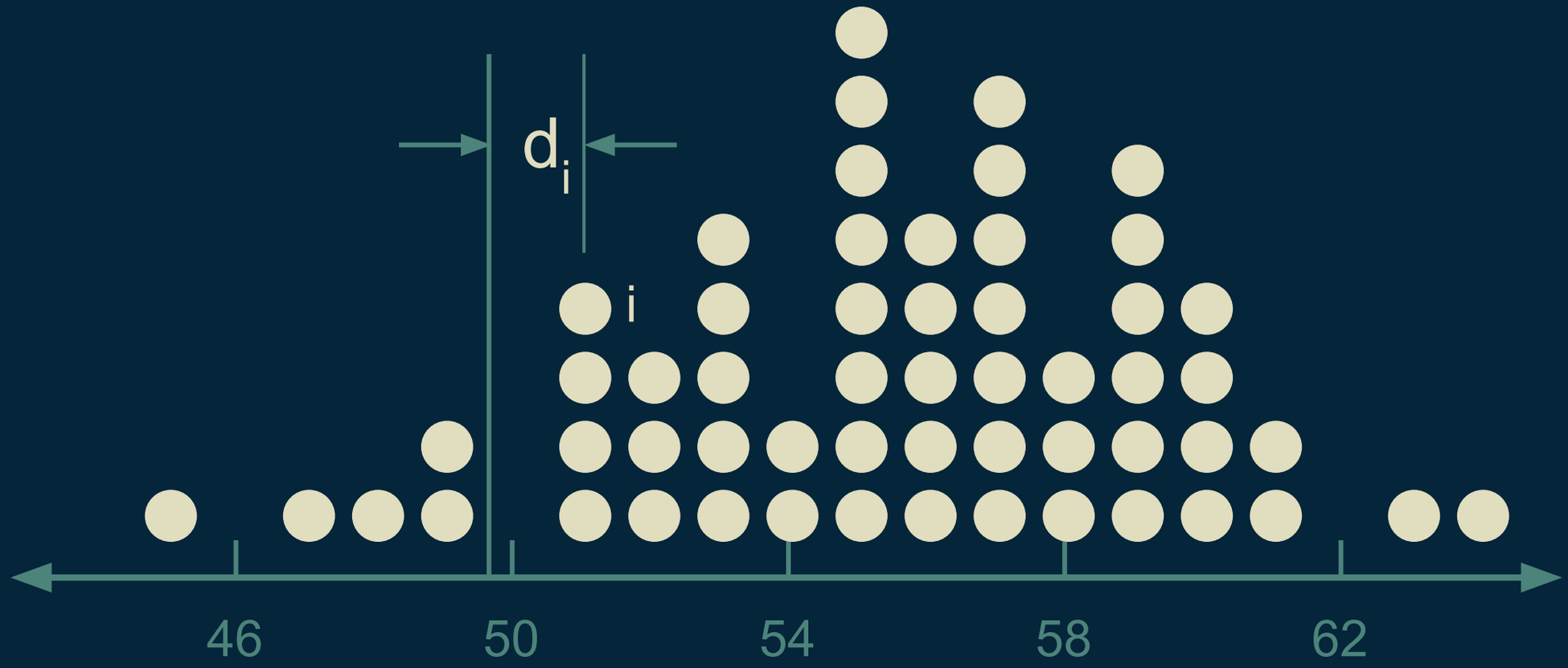
How many M&Ms in a bag?



How wrong is any answer?



How wrong is an answer?



What is the cost of being off by d?

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

What is the cost of being off by d?

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

cost
candidate

---

d^2

What is the cost of being off by d?

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

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

What is the cost of being off by d?

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

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?

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

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

What is the cost of being off by d?

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

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?

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

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?

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

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},

$\mathscr{L}(n_{est})$

What is the total cost of any guess?

For guess n_{est},

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

What is the total cost of any guess?

For guess n_{est},

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

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

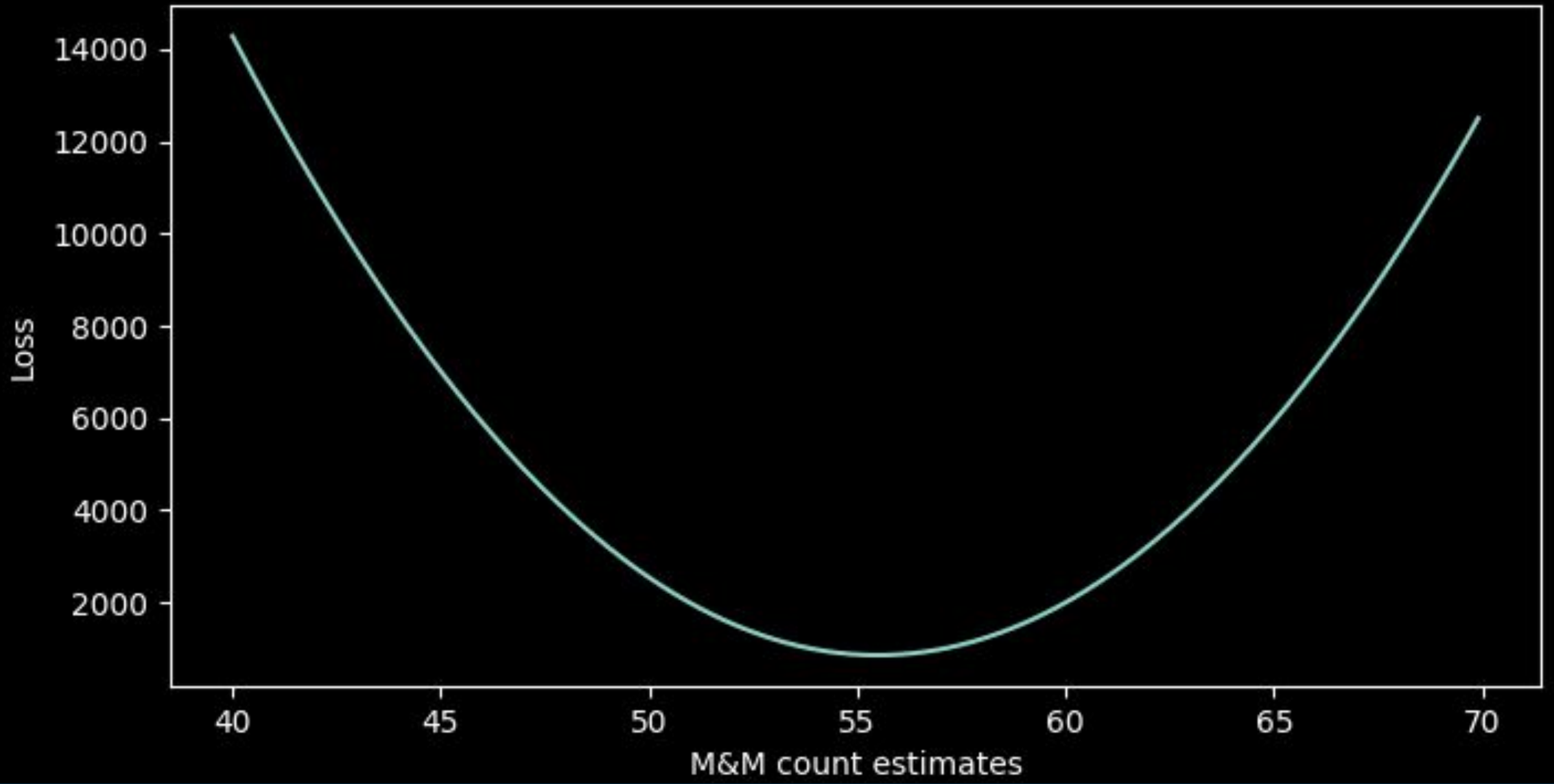
What is the total cost of any guess?

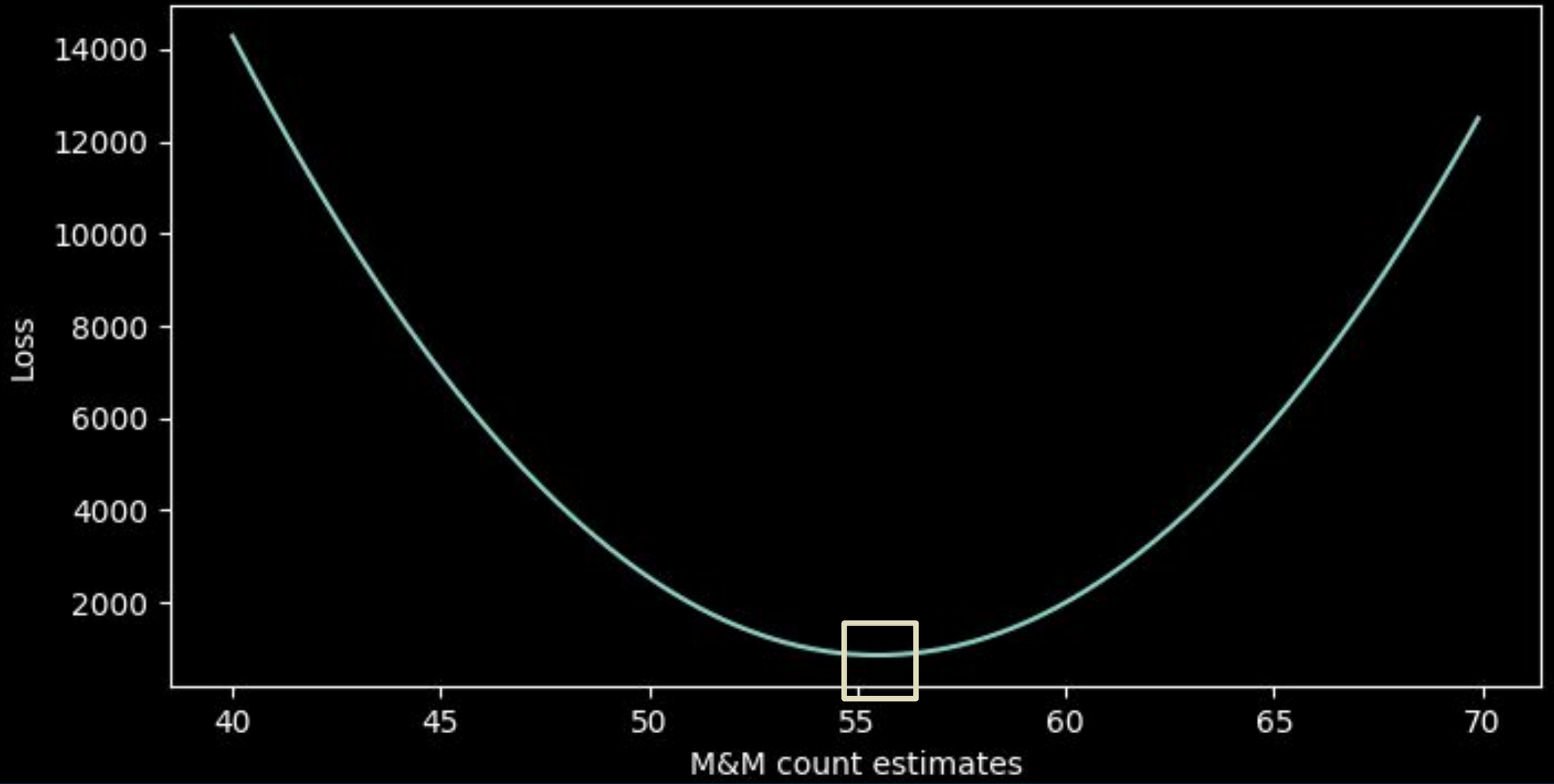
For guess n_{est},

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

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

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


14000
12000
10000
8000
6000
4000
2000
Loss
40
45
50
55
60
65
70
M&M count estimates


14000
12000
10000
8000
6000
4000
2000
Loss
40
45
50
55
60
65
70
M&M count estimates

What is the total cost of any guess?

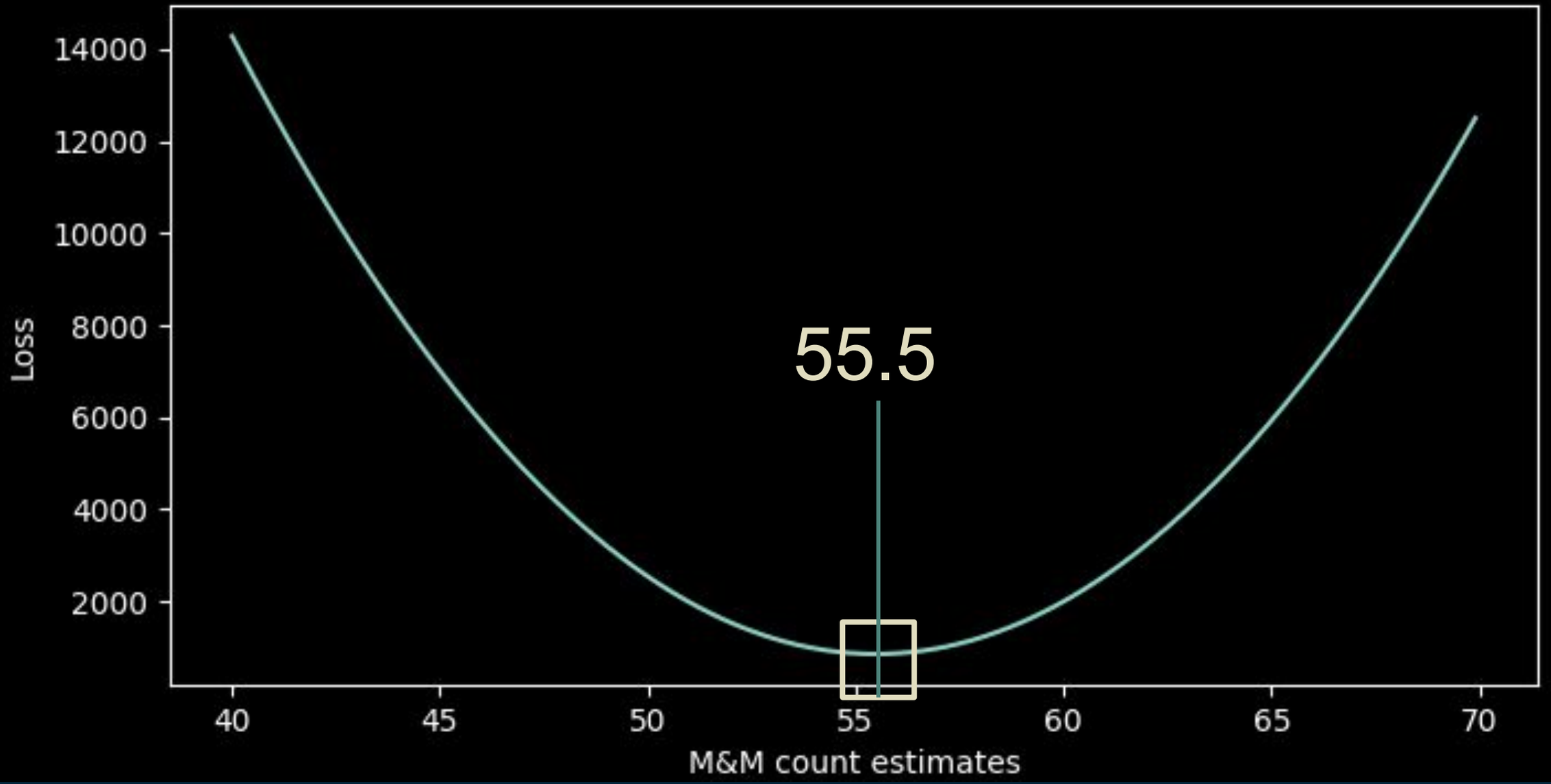
For guess n_{est},

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

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

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

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


14000
12000
10000
8000
6000
4000
2000
Loss
55.5
40
45
50
55
60
65
70
M&M count estimates

Another way: Find where slope = 0

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

Another way: Find where slope = 0

$$d/dn\left(\mathcal{L}(n_{est})\right) = 0$$

Another way: Find where slope = 0

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

$$= d/dn(\Sigma_i (n_i - n_{est})^2)$$

Another way: Find where slope = 0

$$d/dn\left(\mathcal{L}(n_{est})\right) = 0$$

$$= d/dn\left(\Sigma_i (n_i - n_{est})^2\right)$$

$$= \Sigma_i \, d/dn\left((n_i - n_{est})^2\right)$$

Another way: Find where slope = 0

$$d/dn\left(\mathcal{L}(n_{est})\right) = 0$$

$$= d/dn\left(\Sigma_i (n_i - n_{est})^2\right)$$

$$= \Sigma_i \, d/dn\left((n_i - n_{est})^2\right)$$

$$= \Sigma_i \, 2(n_i - n_{est})$$

Another way: Find where slope = 0

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

$$= d/dn(\Sigma_i (n_i - n_{est})^2)$$

$$= \Sigma_i \, d/dn((n_i - n_{est})^2)$$

$$= \Sigma_i \, 2(n_i - n_{est})$$

$$= \Sigma_i \, (n_i - n_{est})$$

Another way: Find where slope = 0

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

Another way: Find where slope = 0

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

$$\Sigma_i n_i - \Sigma_i n_{est} = 0$$

Another way: Find where slope = 0

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

$$\Sigma_i n_i - \Sigma_i n_{est} = 0$$

$$\Sigma_i n_i - m\, n_{est} = 0$$

Another way: Find where slope = 0

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

$$\Sigma_i n_i - \Sigma_i n_{est} = 0$$

$$\Sigma_i n_i - m\, n_{est} = 0$$

$$m\, n_{est} = \Sigma_i n_i$$

Another way: Find where slope = 0

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

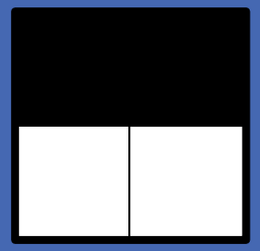
$$\Sigma_i n_i - \Sigma_i n_{est} = 0$$

$$\Sigma_i n_i - m\, n_{est} = 0$$

$$m\, n_{est} = \Sigma_i n_i$$

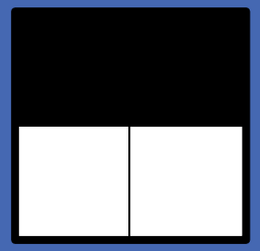
$$n_{est} = \Sigma_i n_i \;/\; m = \bar{n}$$

Errors



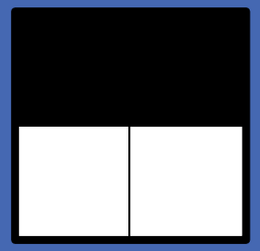
truth	
0.	solid
0.	vertical
0.	diagonal
1.	horizontal

Errors



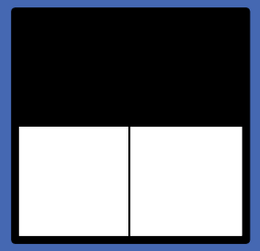
truth	answer	
0.	.5	solid
0.	.75	vertical
0.	-.25	diagonal
1.	-.75	horizontal

Errors



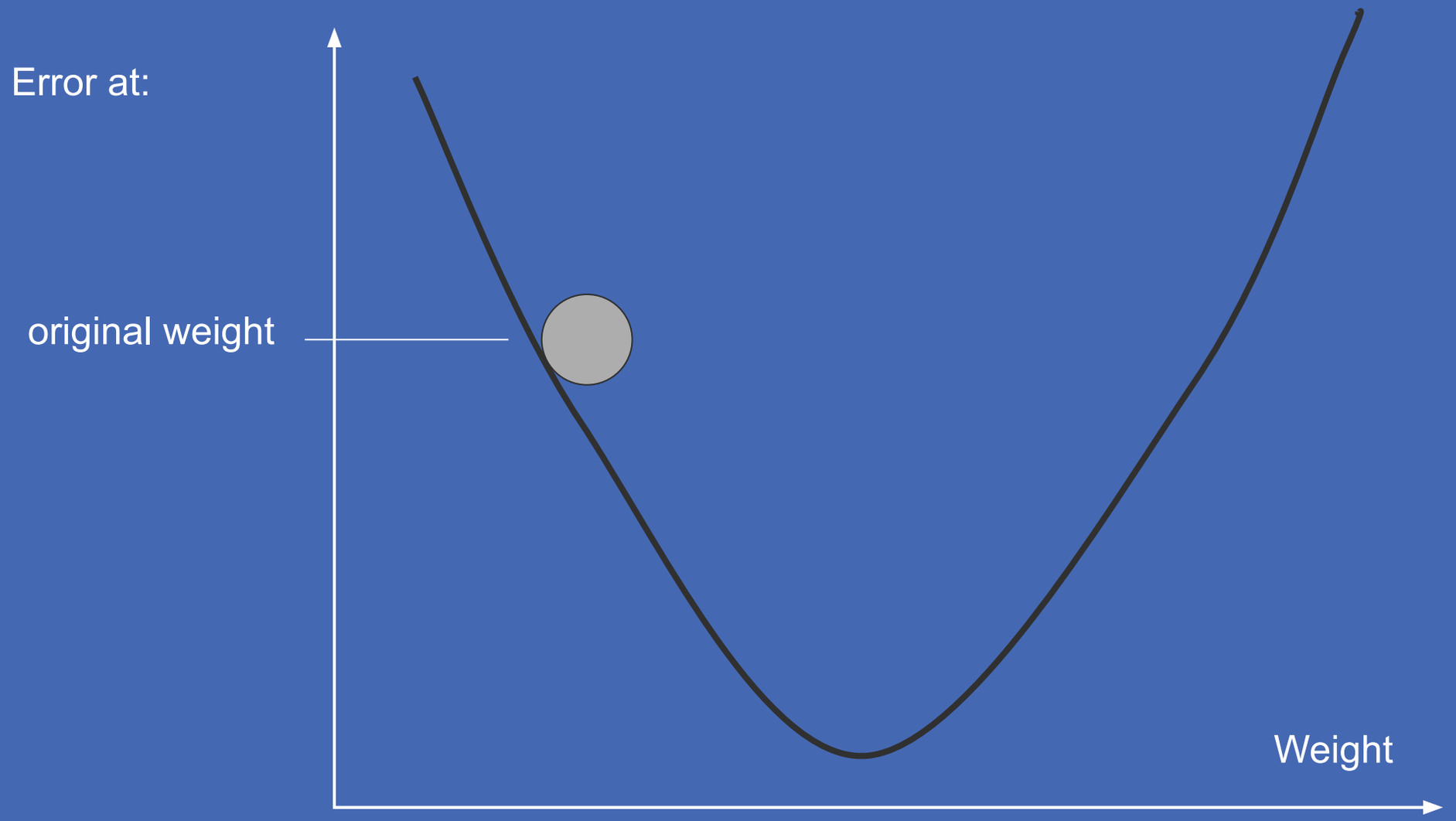
error	truth	answer	solid
.5	0.	.5	
.75	0.	.75	vertical
.25	0.	-.25	diagonal
1.75	1.	-.75	horizontal

Errors

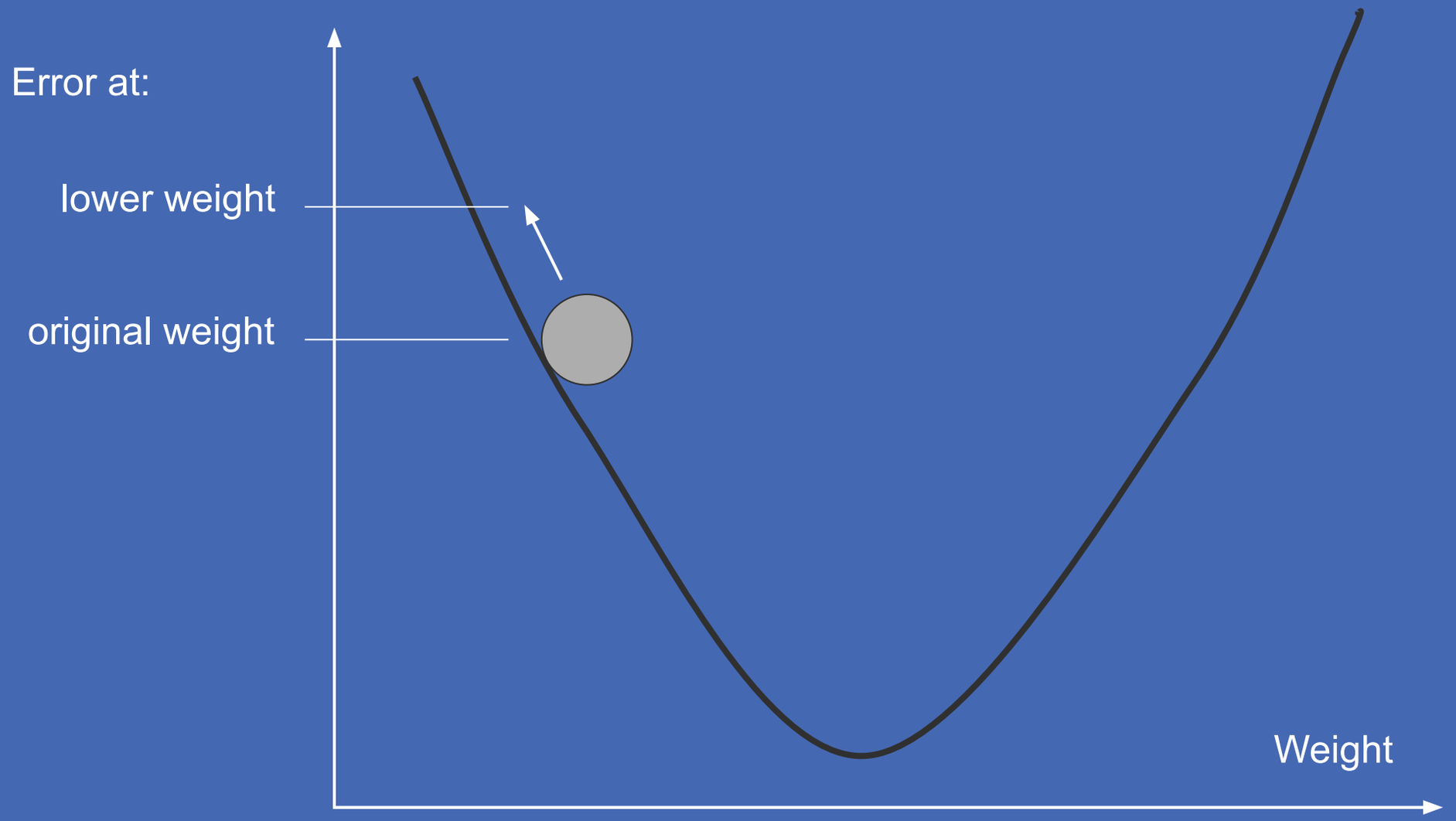


	error	truth	answer	
solid	.5	0.	.5	
vertical	.75	0.	.75	
diagonal	.25	0.	-.25	
horizontal	1.75	1.	-.75	
total	3.25			

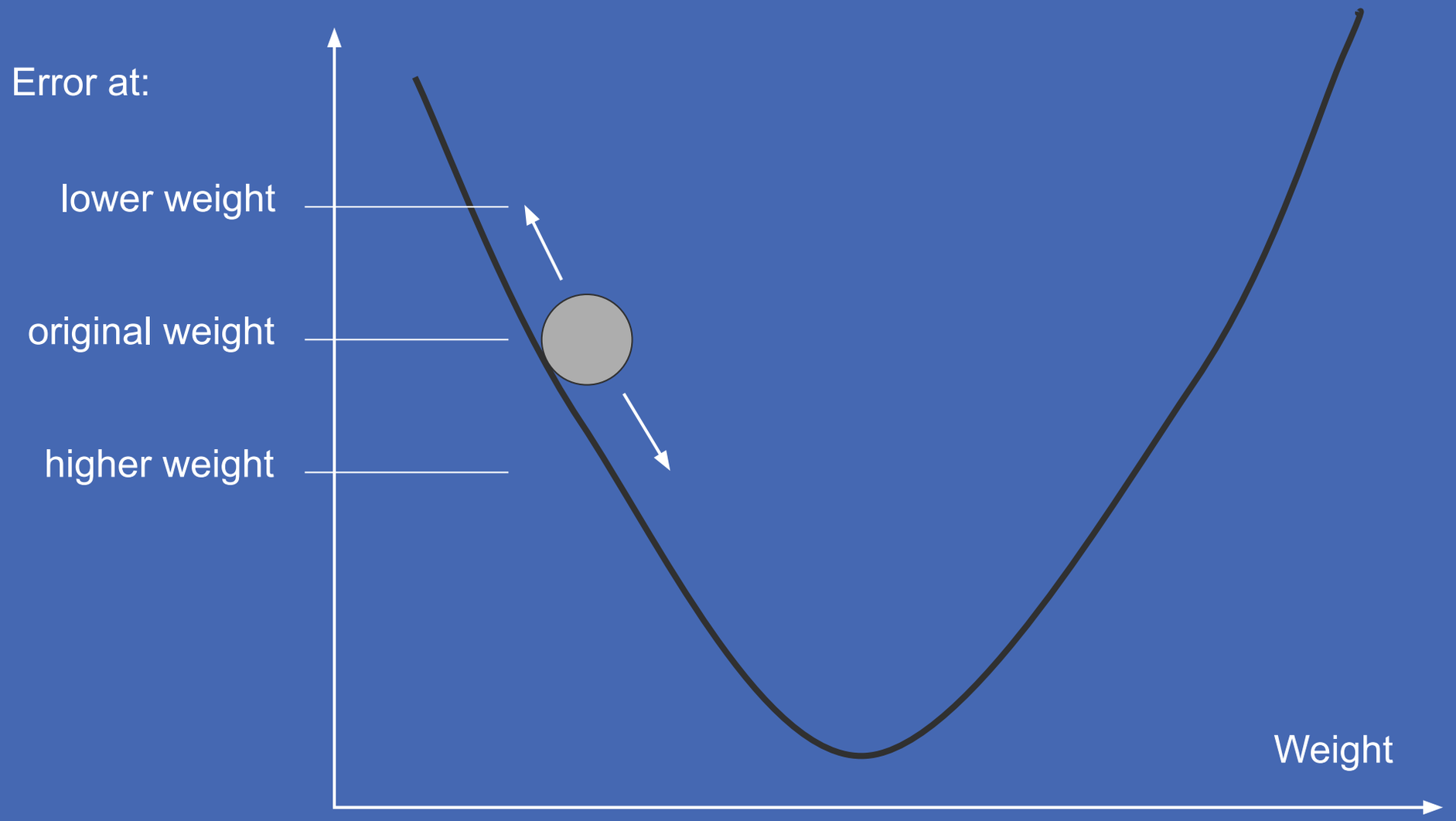
Learn all the weights: Gradient descent



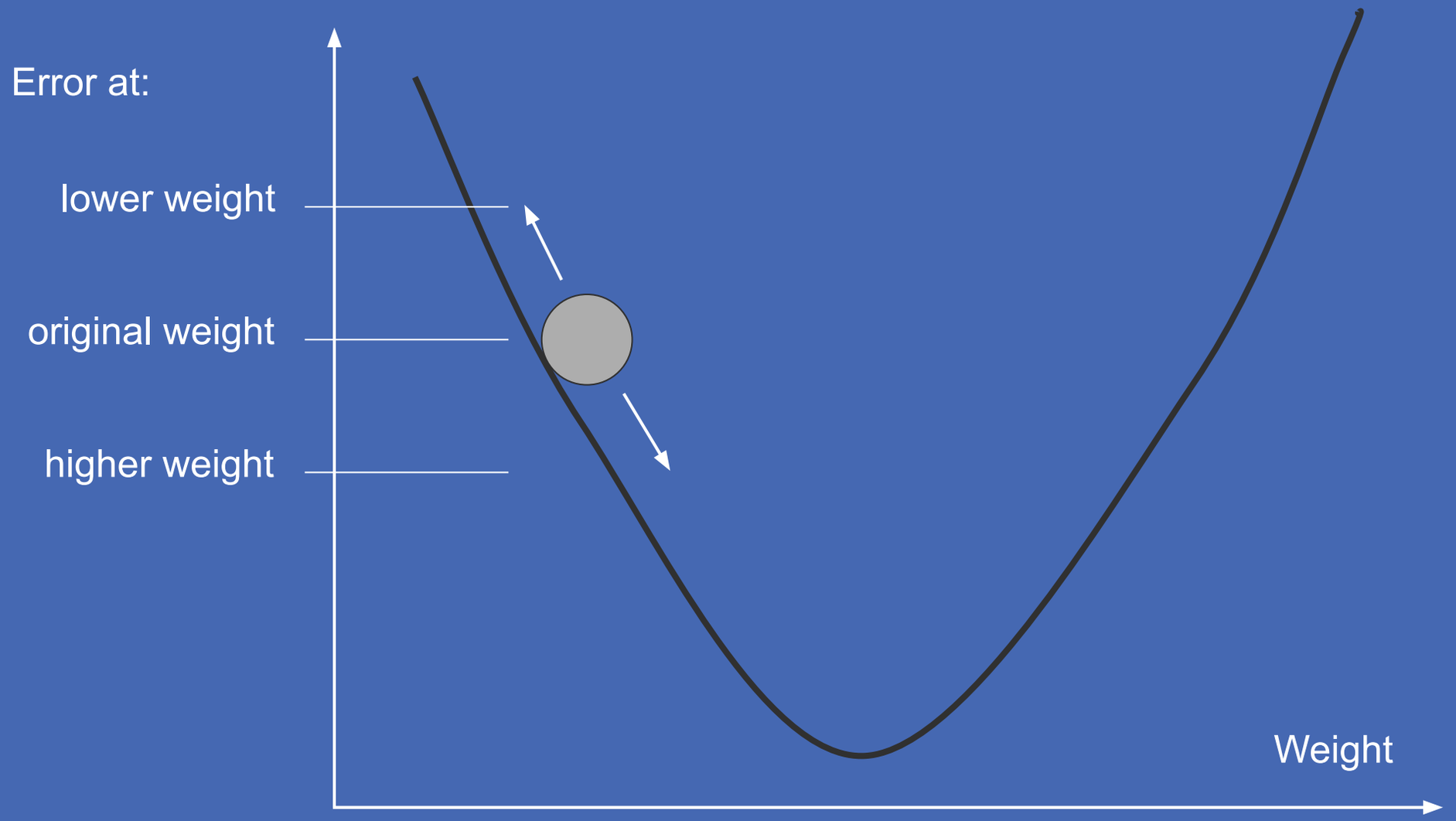
Learn all the weights: Gradient descent



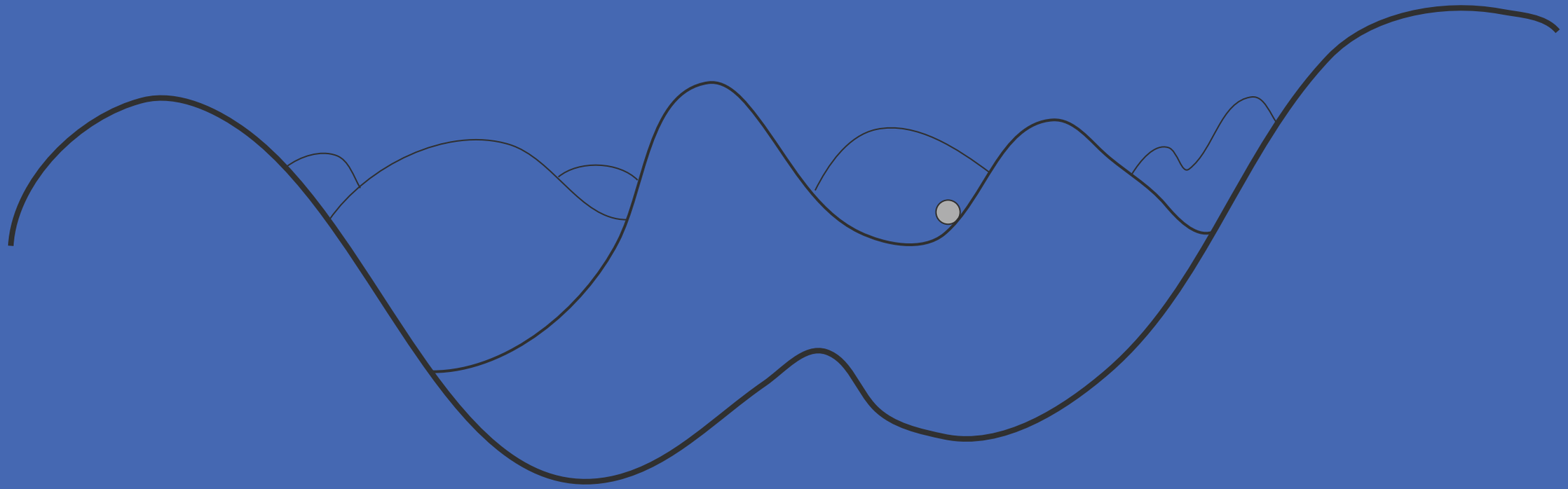
Learn all the weights: Gradient descent



Numerically calculating the gradient is expensive

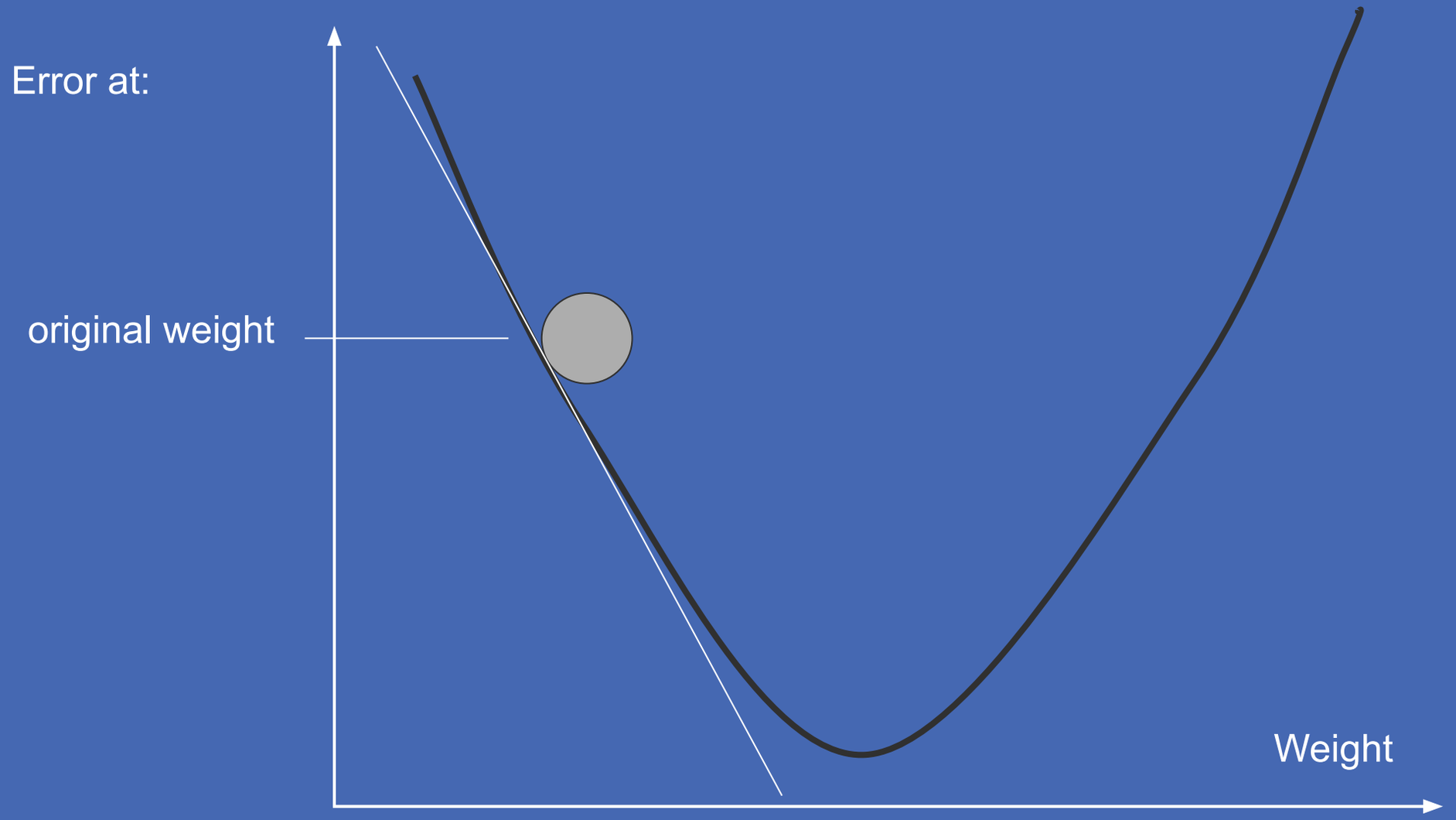


Numerically calculating the gradient is very expensive

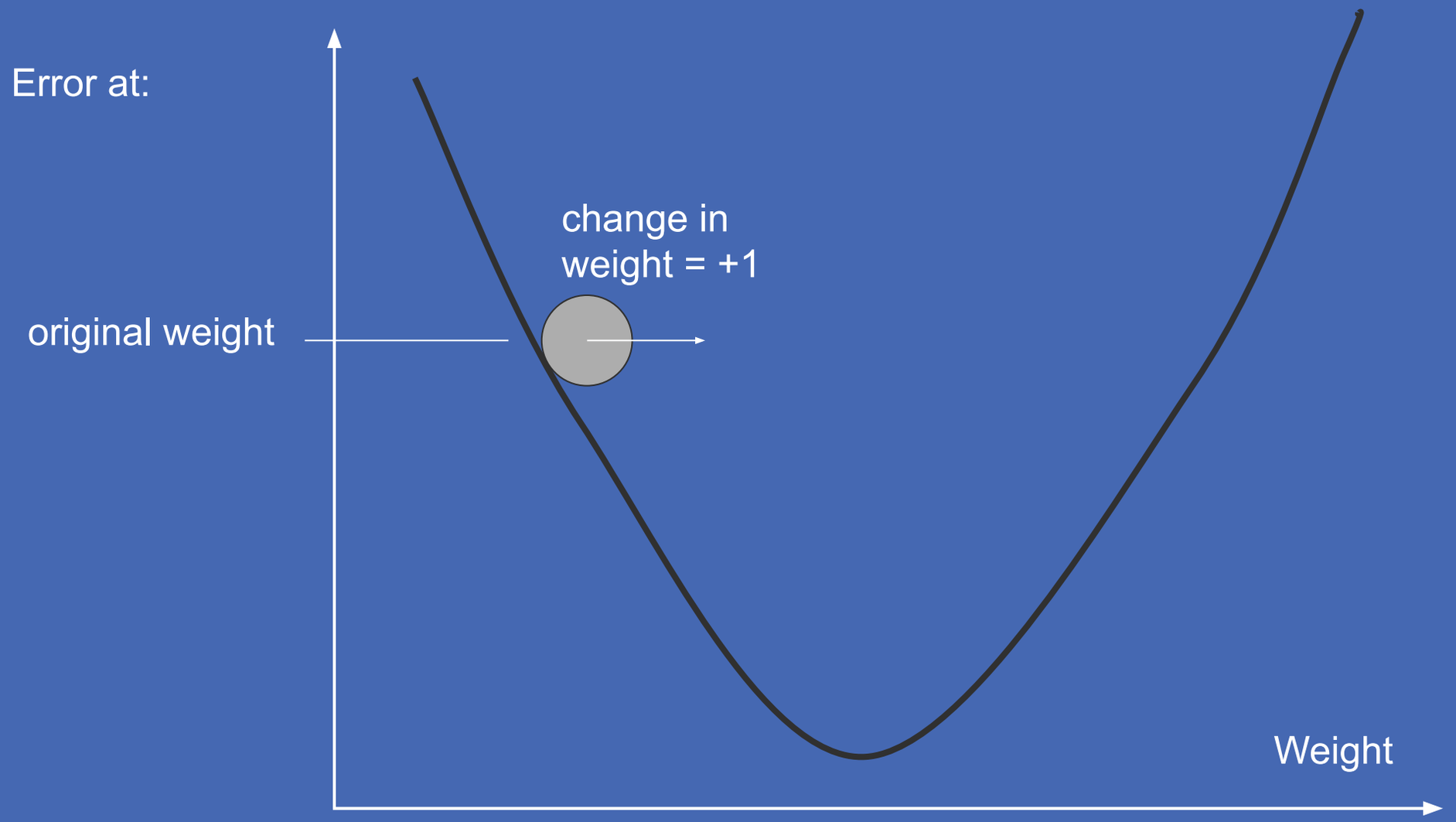


How backpropagation works

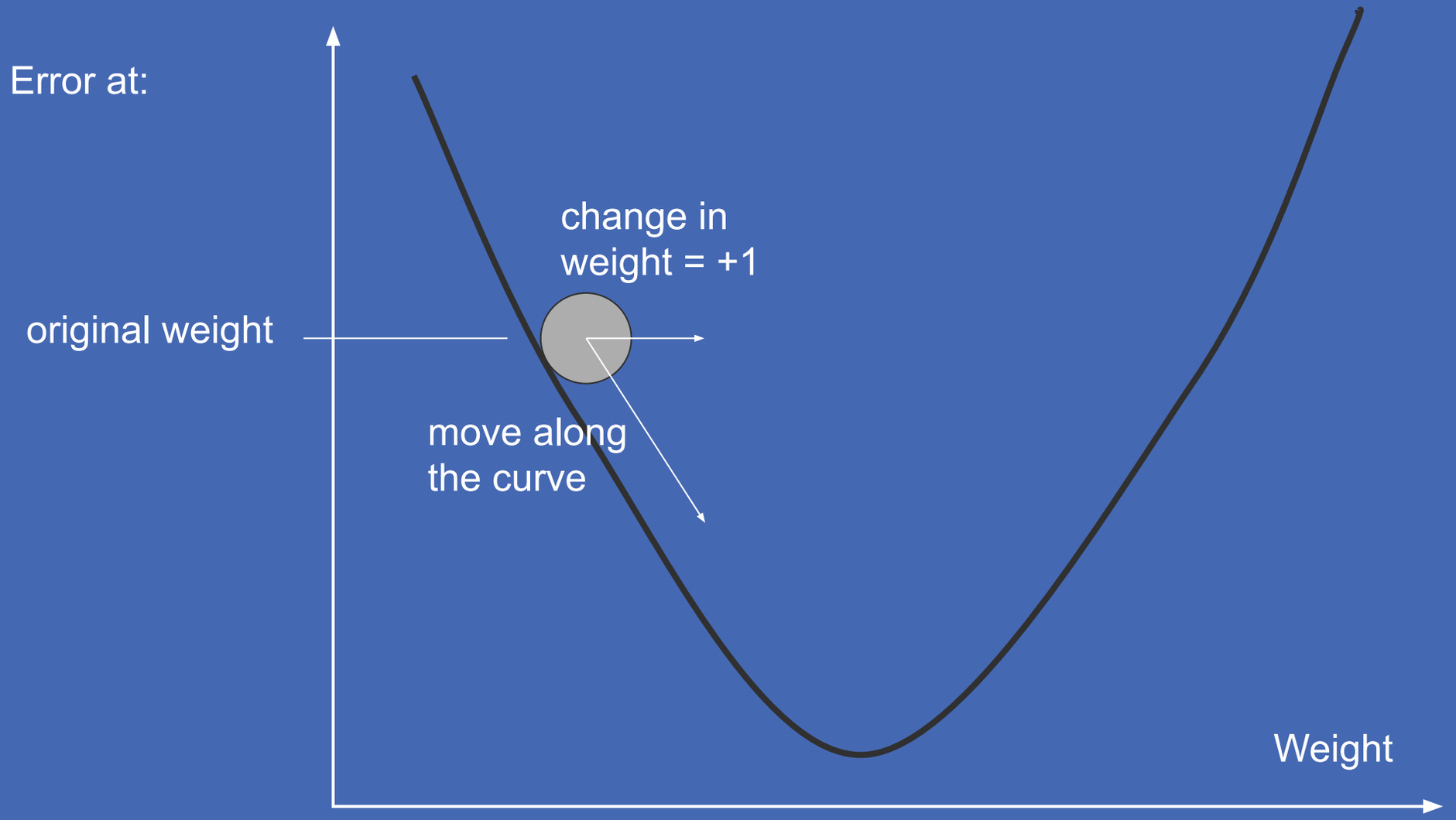
Calculate the gradient (slope) directly



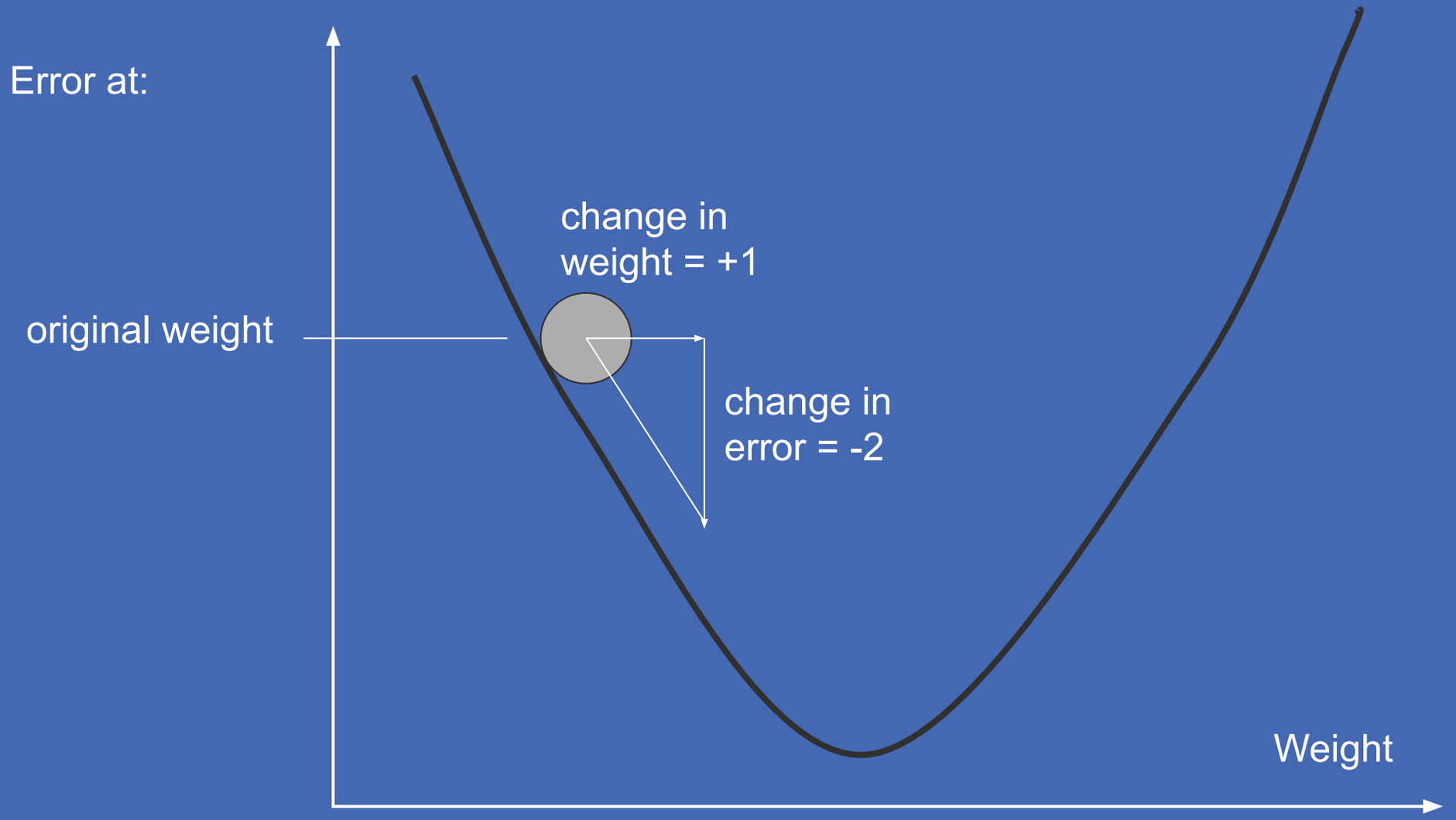
Slope



Slope

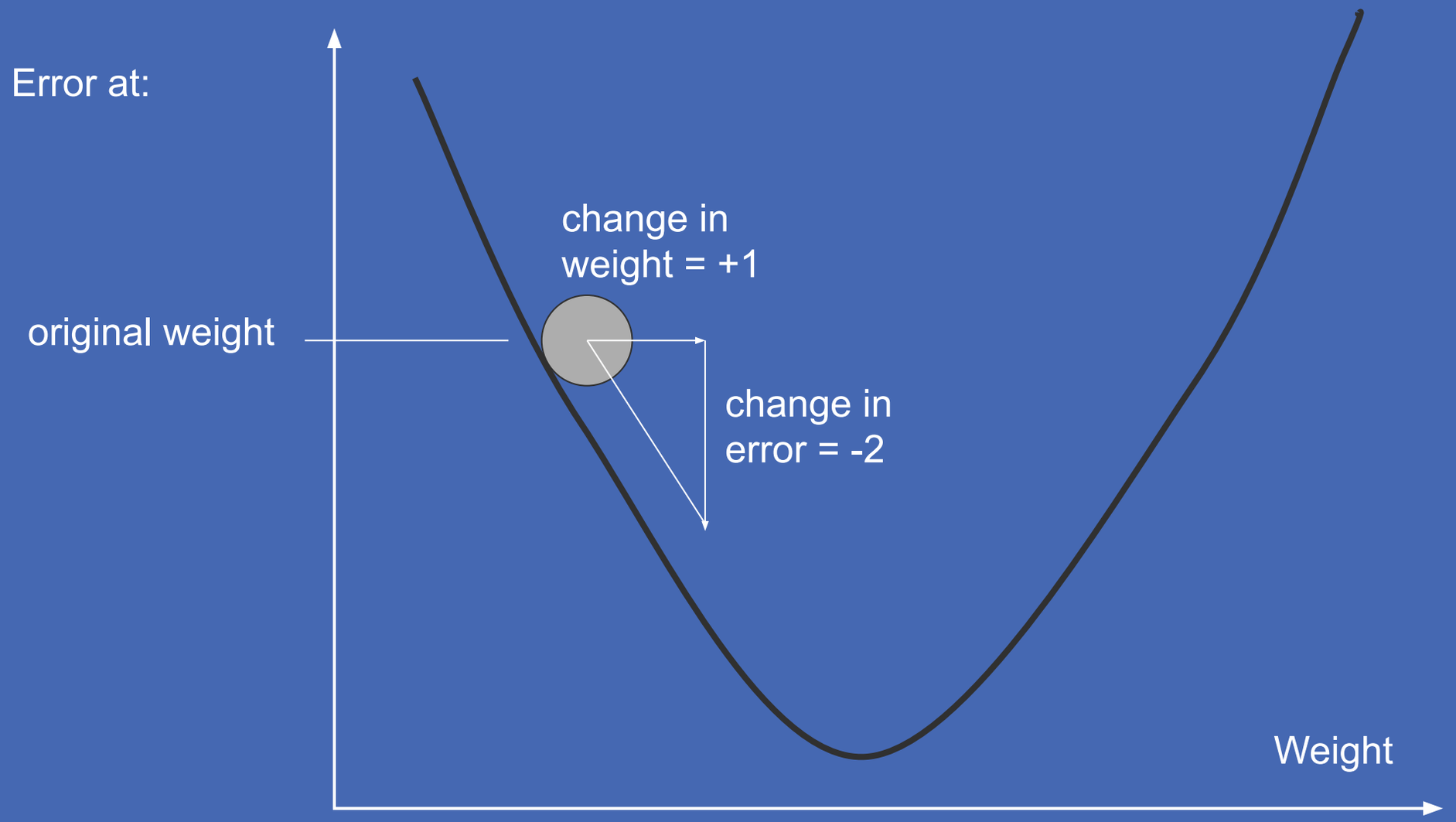

Error at:
change in
weight = +1
original weight
move along
the curve
Weight

Slope


Error at:
change in
weight = +1
original weight
change in
error = -2
Weight

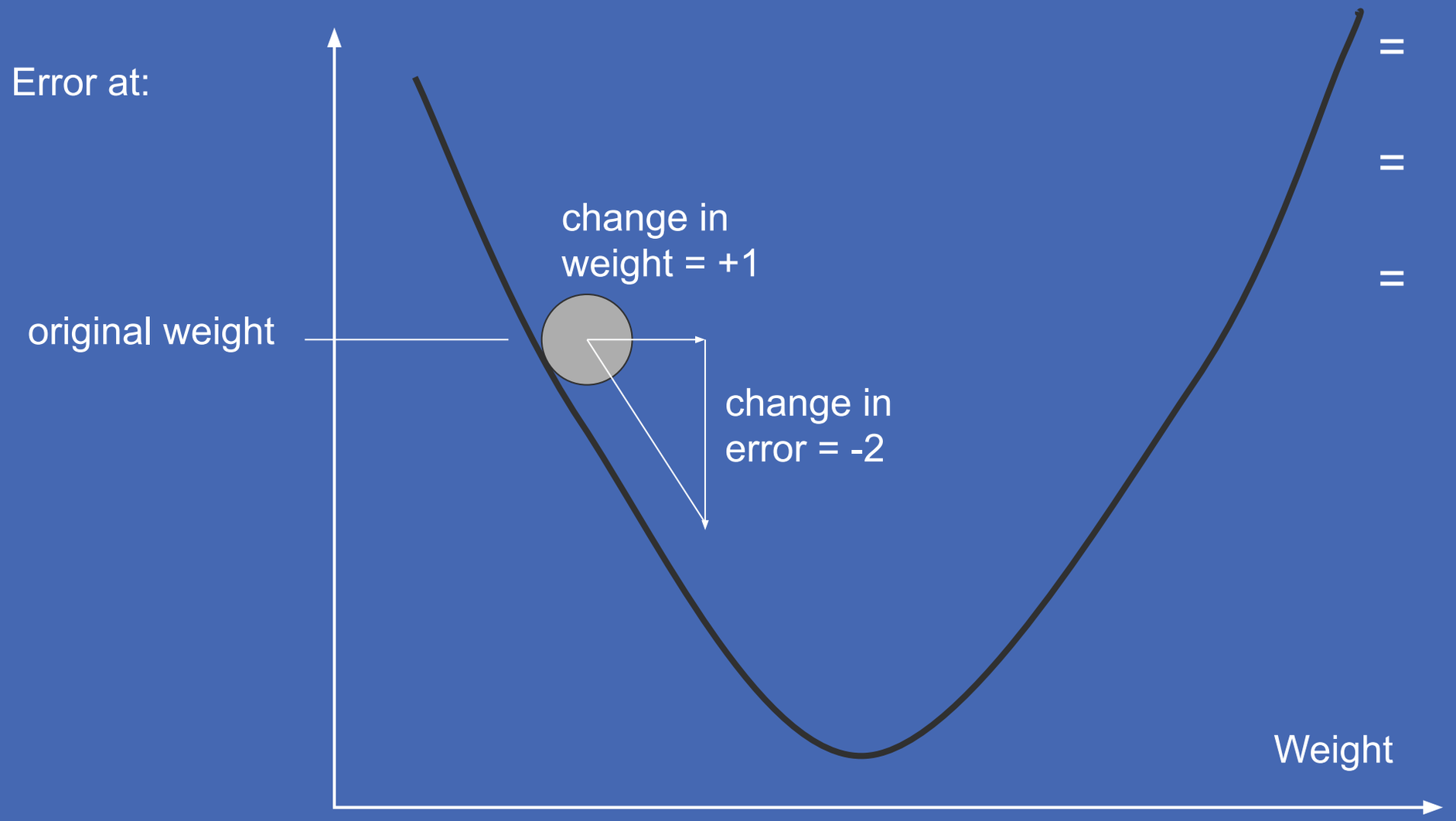
Slope

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



Slope

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



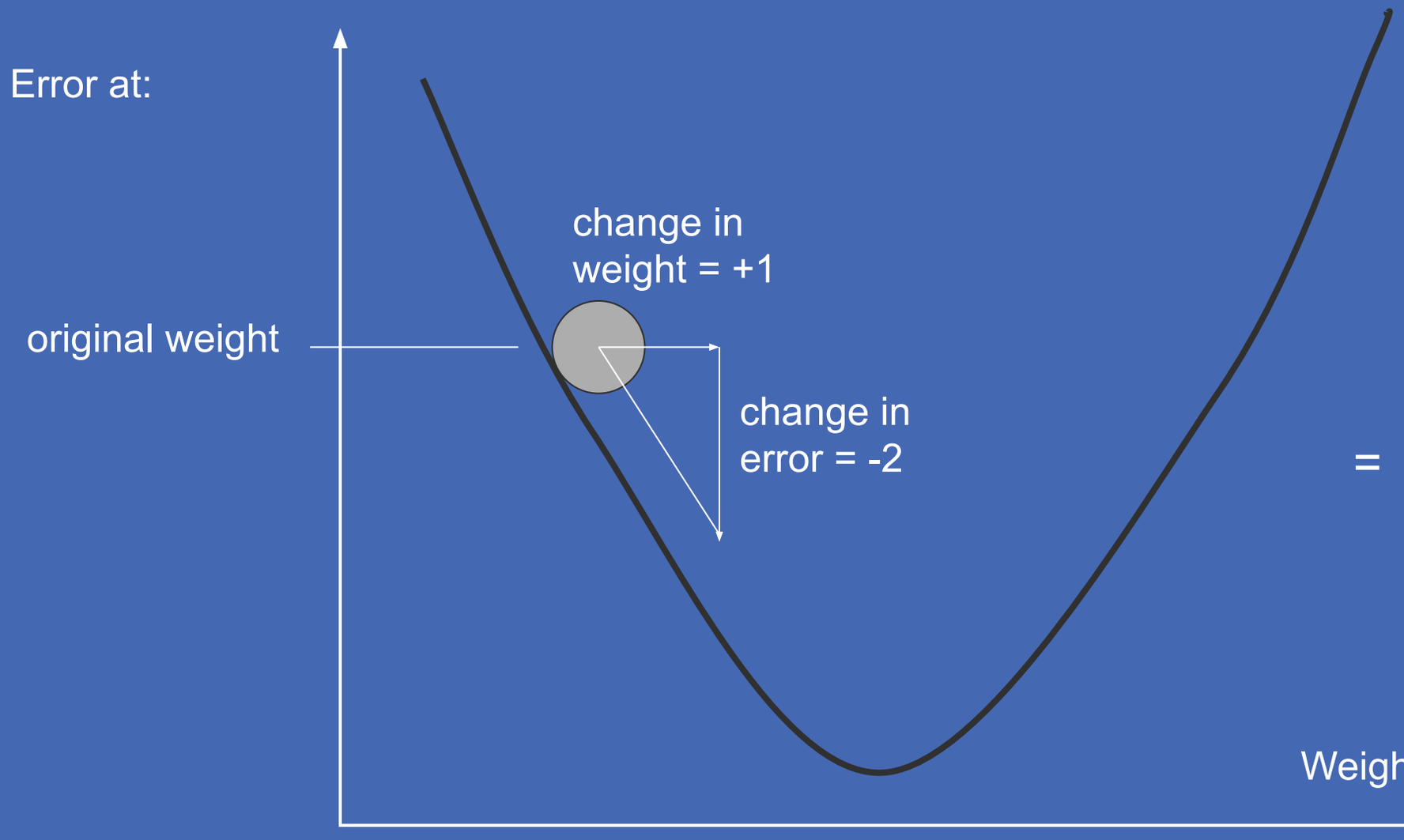
$$= \frac{\Delta \text{ error}}{\Delta \text{ weight}}$$

$$= \frac{d(\text{error})}{d(\text{weight})}$$

$$= \frac{\partial e}{\partial w}$$

Slope

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



$$= \frac{\Delta \text{ error}}{\Delta \text{ weight}}$$

$$= \frac{d(\text{error})}{d(\text{weight})}$$

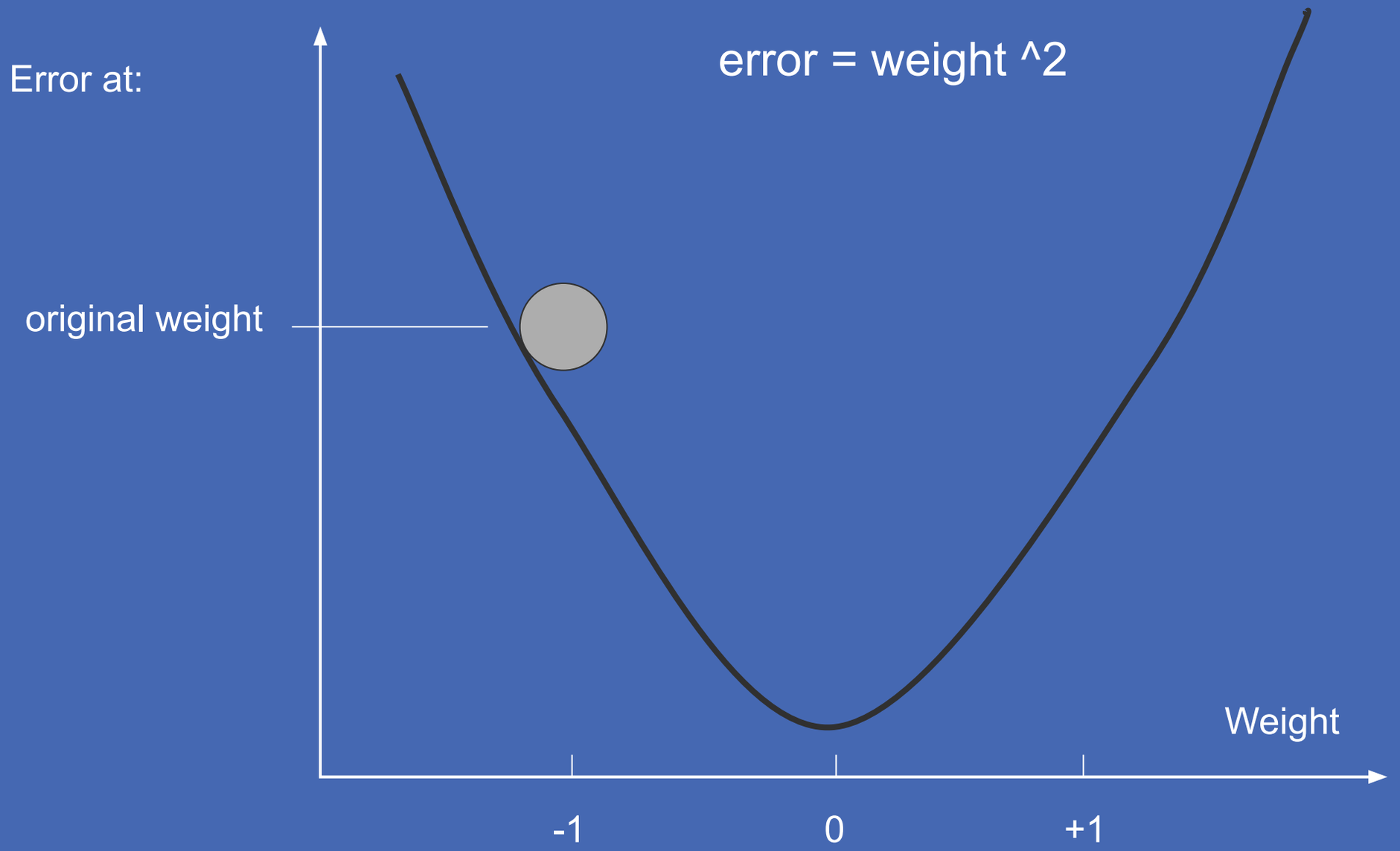
$$= \frac{\partial e}{\partial w}$$

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

Slope

You have to know your error function.
For example:

error = weight ^2

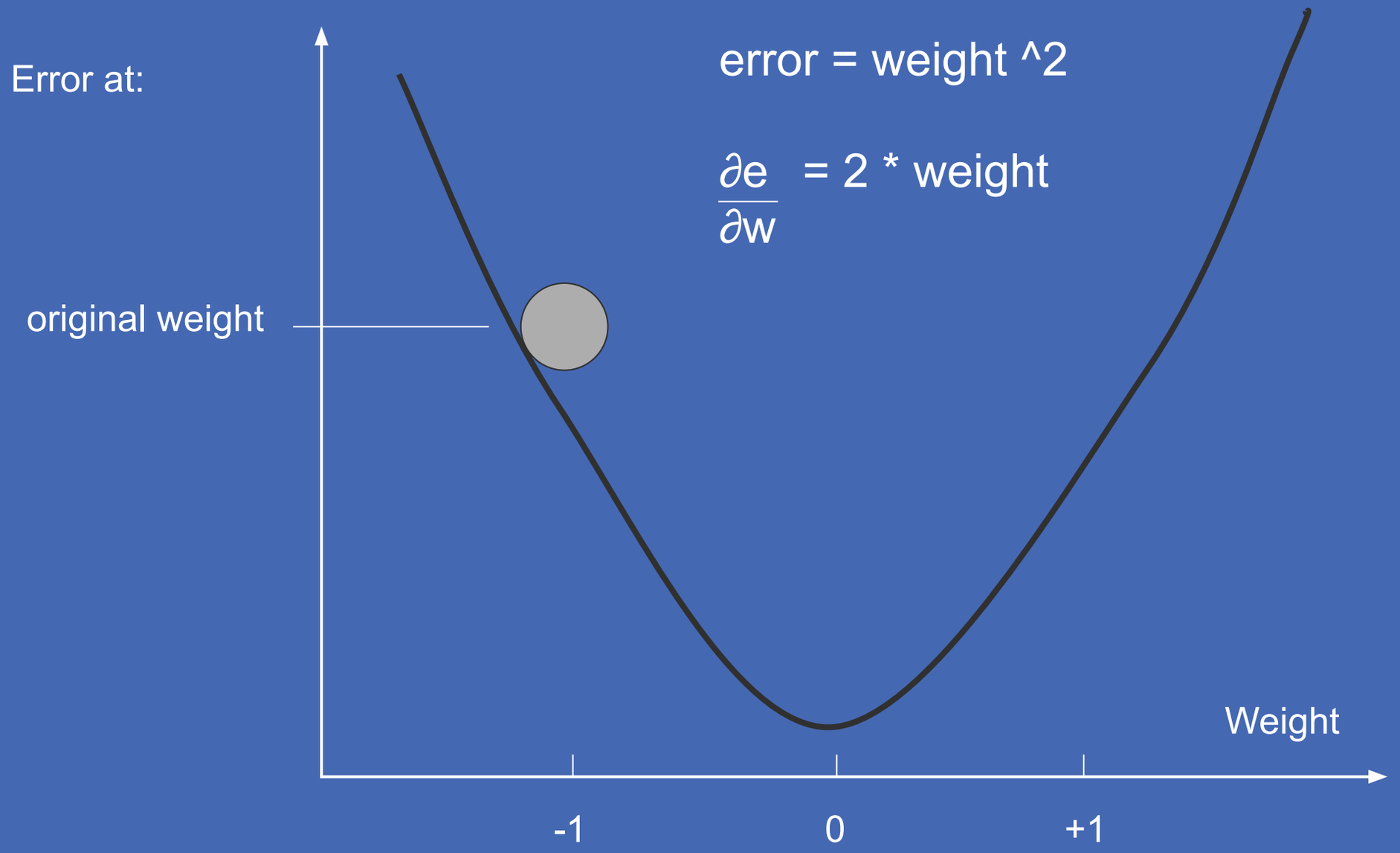


Slope

You have to know your error function.
For example:

error = weight ^2

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



Slope

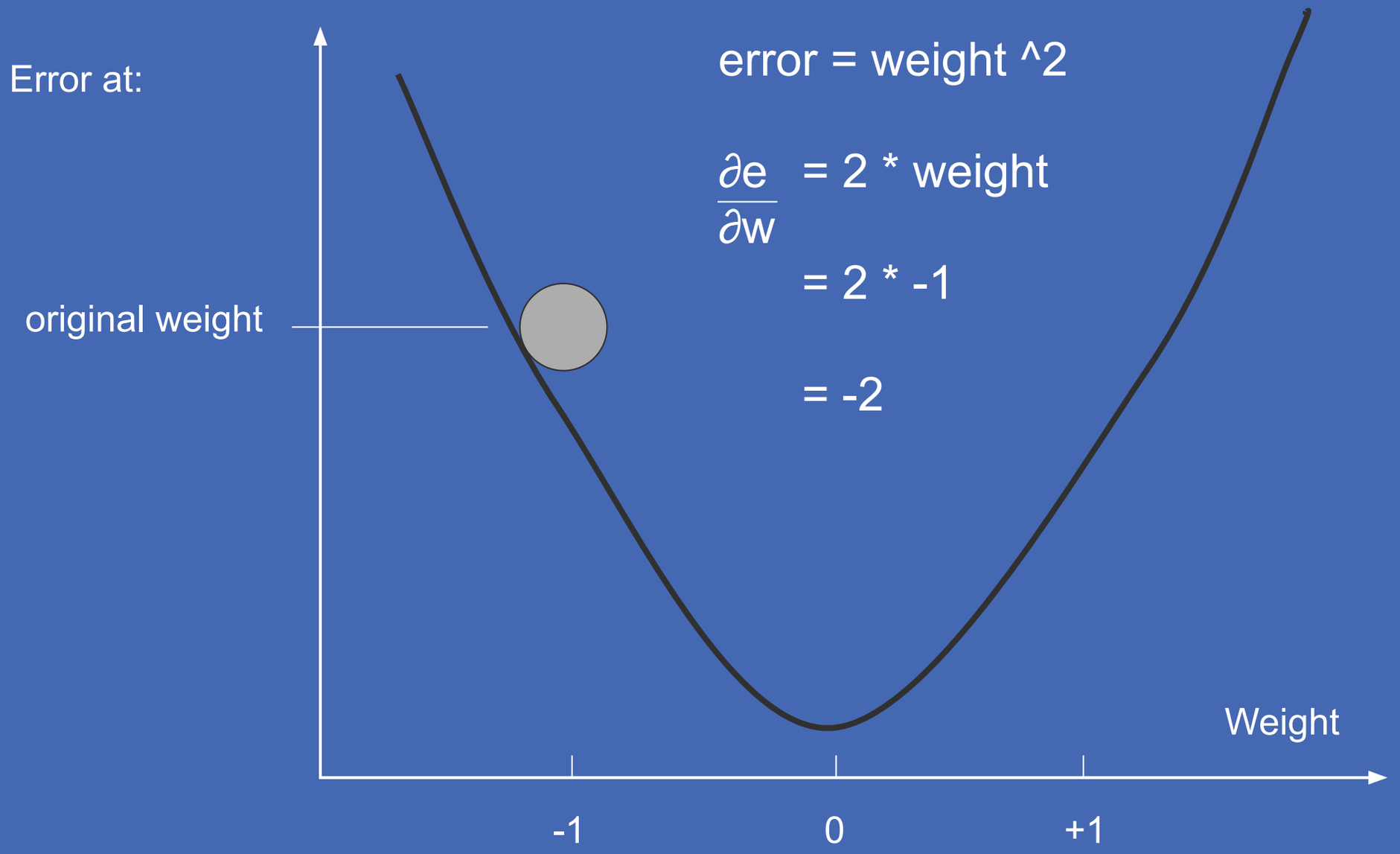
You have to know your error function.
For example:

error = weight ^2

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

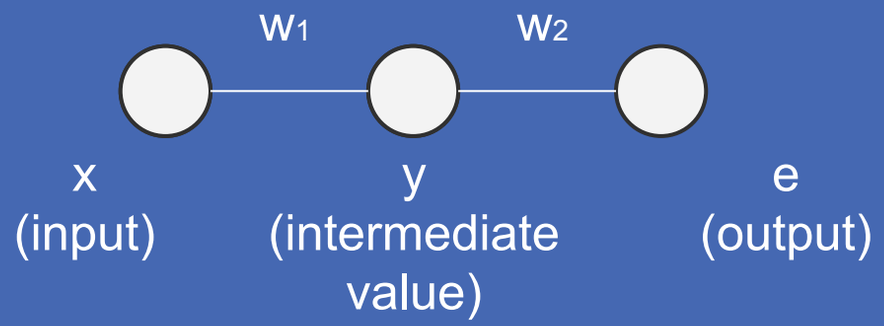
$$= 2 * -1$$

$$= -2$$



Chaining

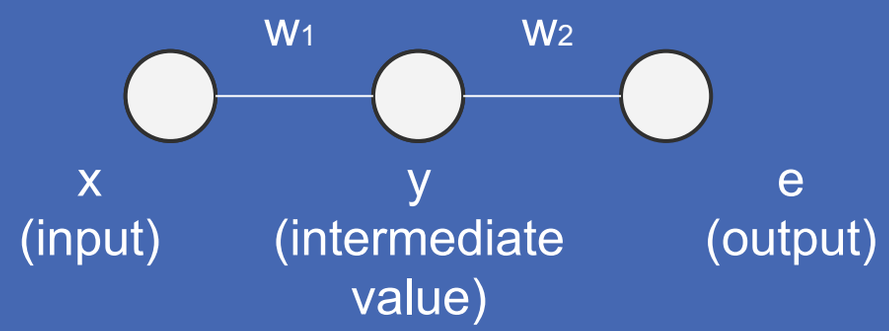
$y = x * w_1$



Chaining

$$y = x * w_1$$

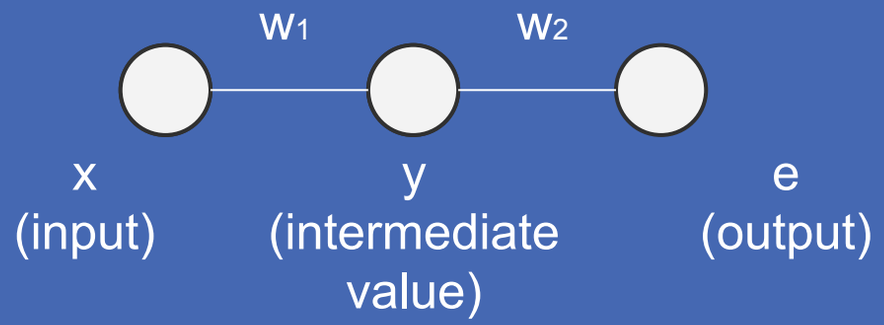
$$\frac{\partial y}{\partial w_1} = x$$



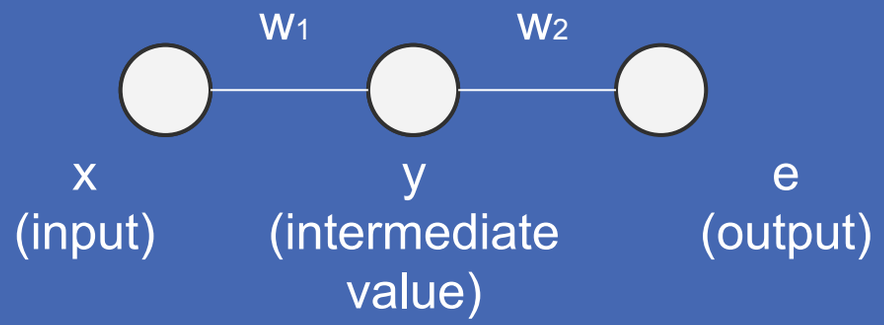
Chaining

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

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



Chaining

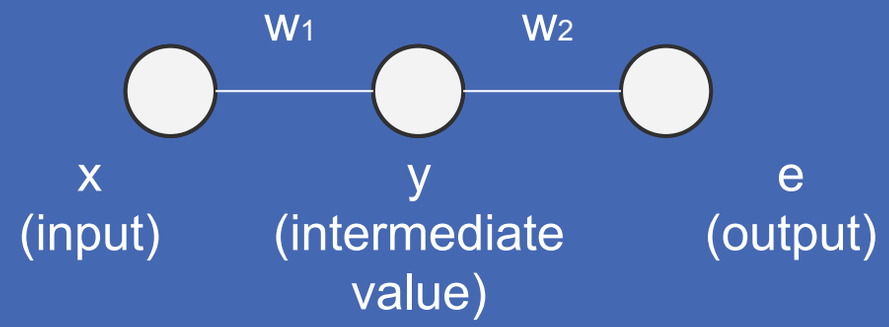


$$y = x * w_1$$
$$\frac{\partial y}{\partial w_1} = x$$

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

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

Chaining



$$y = x * w_1$$

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

$$e = y * w_2$$

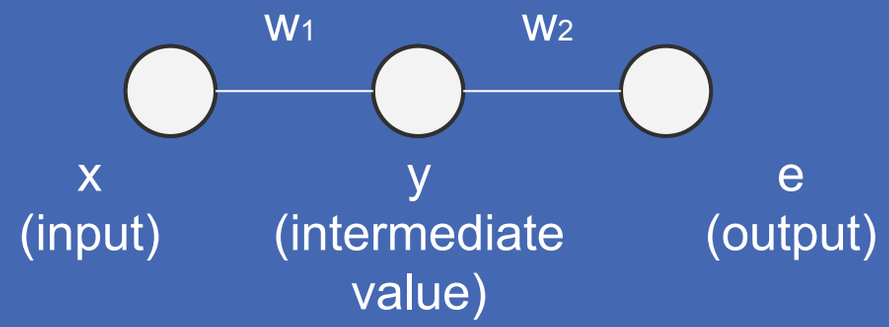
$$\frac{\partial e}{\partial y} = w_2$$

$$e = x * w_1 * w_2$$

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

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

Chaining



$$y = x * w_1$$

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

$$e = y * w_2$$

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

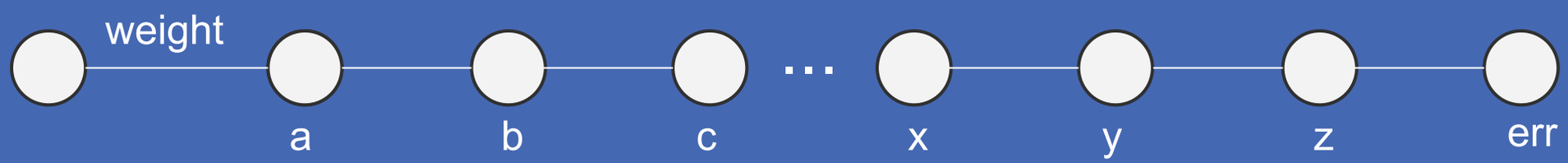
$$e = x * w_1 * w_2$$

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

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

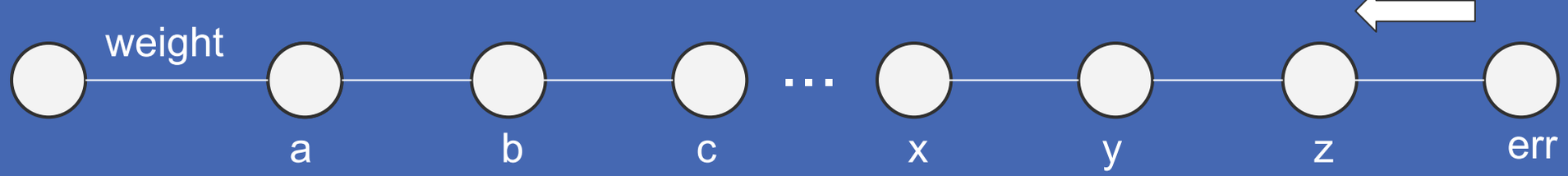
Chaining

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



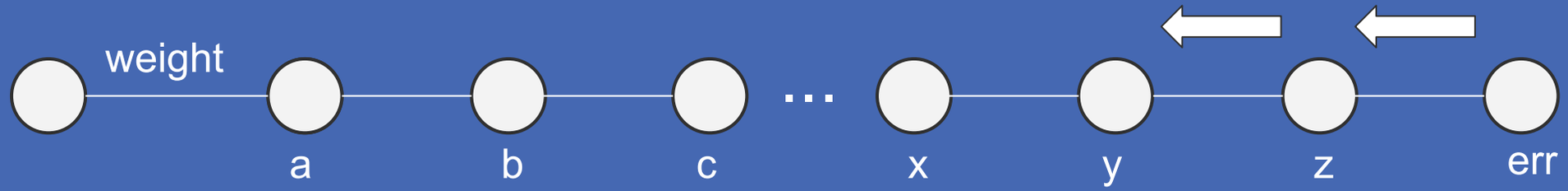
Chaining

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



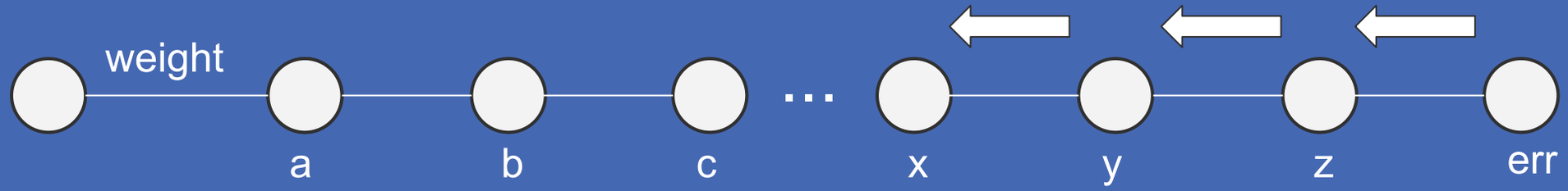
Chaining

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



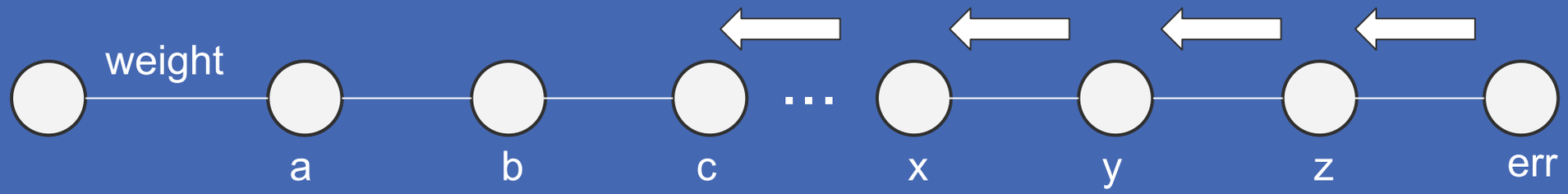
Chaining

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



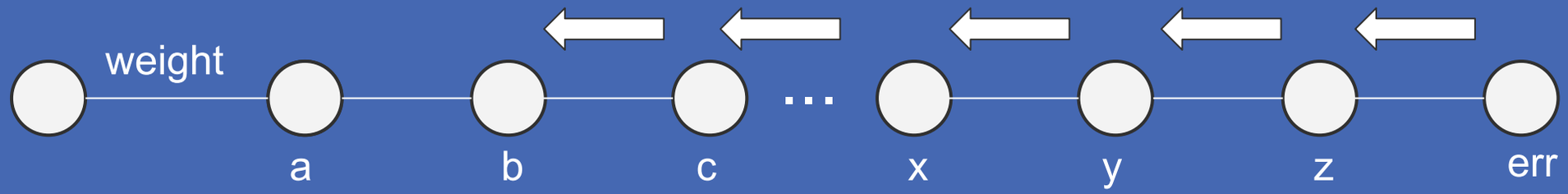
Chaining

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



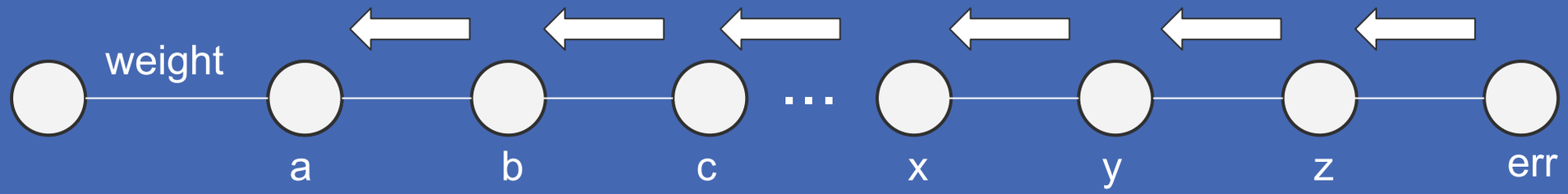
Chaining

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



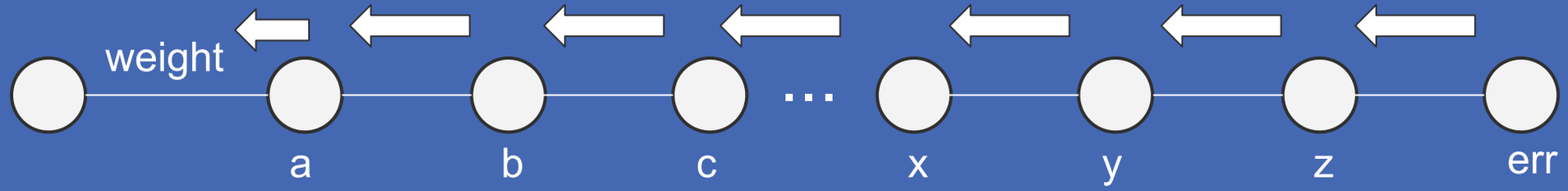
Chaining

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

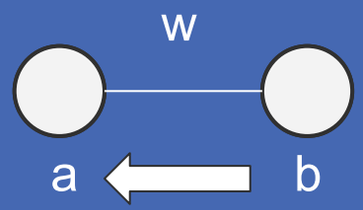


Chaining

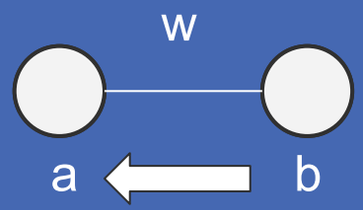
$$\frac{\partial err}{\partial weight} = \frac{\partial a}{\partial weight} * \frac{\partial b}{\partial a} * \frac{\partial c}{\partial b} * \frac{\partial d}{\partial c} * \ldots * \frac{\partial y}{\partial x} * \frac{\partial z}{\partial y} * \frac{\partial err}{\partial z}$$



Backpropagation challenge: weights

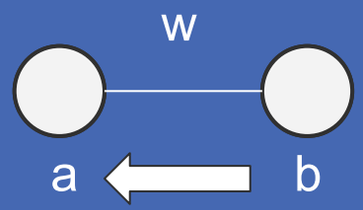


Backpropagation challenge: weights



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

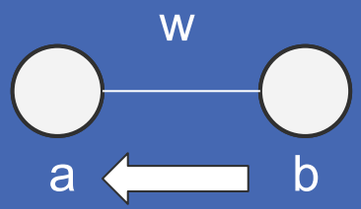
Backpropagation challenge: weights



$$b = wa$$

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

Backpropagation challenge: weights

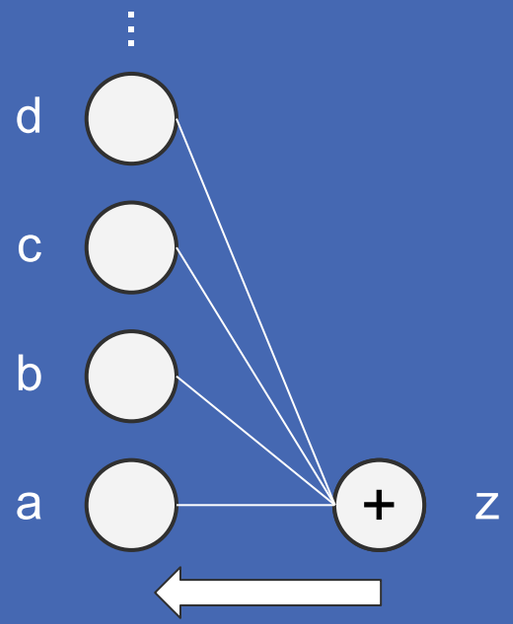


$$b = wa$$

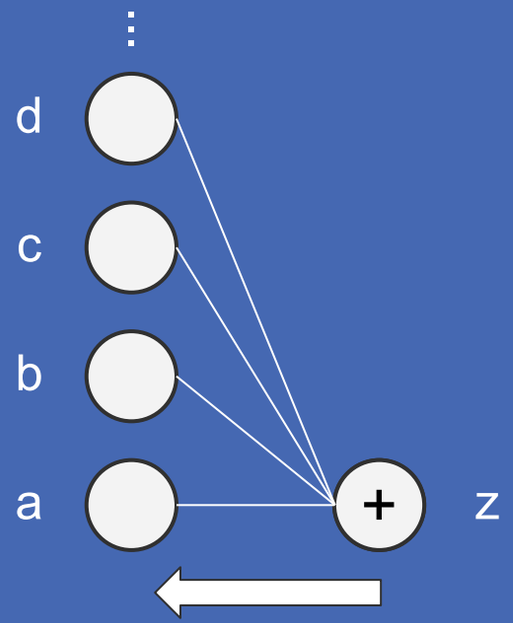
$$\frac{\partial err}{\partial a} = \frac{\partial b}{\partial a} * \frac{\partial err}{\partial b}$$

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

Backpropagation challenge: sums

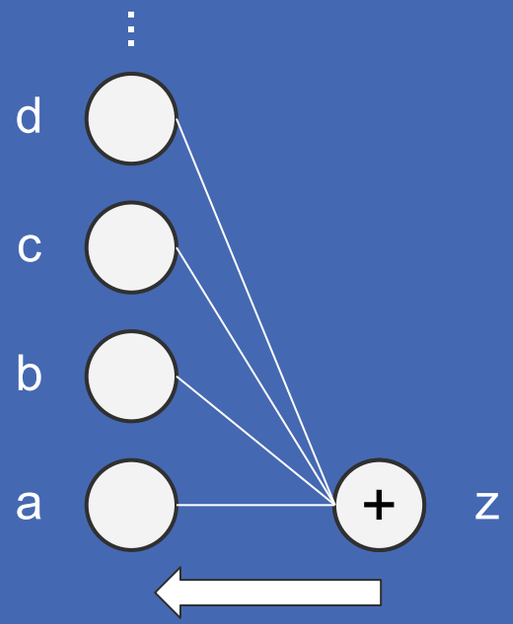


Backpropagation challenge: sums



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

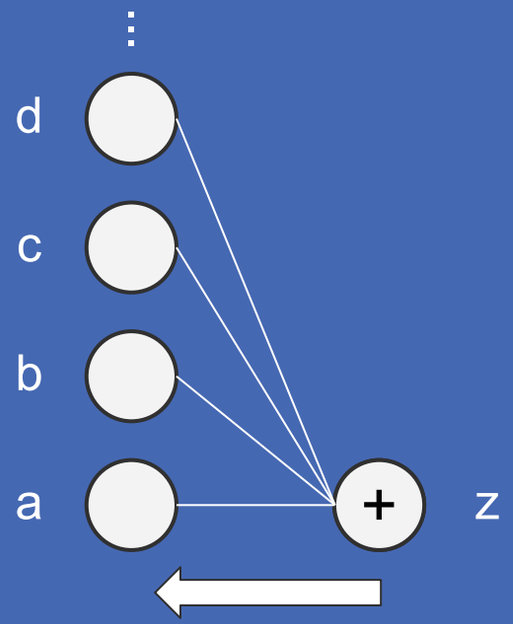
Backpropagation challenge: sums



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

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

Backpropagation challenge: sums

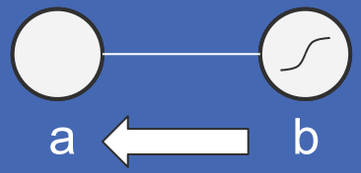


$$\frac{\partial err}{\partial a} = \frac{\partial z}{\partial a} * \frac{\partial err}{\partial z}$$

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

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

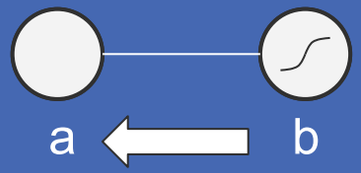
Backpropagation challenge: sigmoid



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

Backpropagation challenge: sigmoid

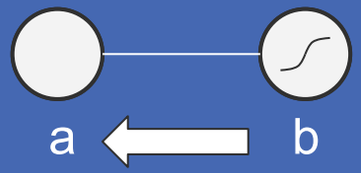
$$b = \frac{1}{1 + e^{-a}}$$



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

Backpropagation challenge: sigmoid

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

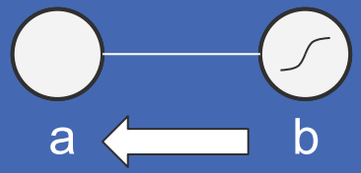


$$\frac{\partial err}{\partial a} = \frac{\partial b}{\partial a} * \frac{\partial err}{\partial b}$$

Backpropagation challenge: sigmoid

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

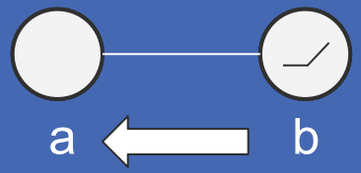
Because math is beautiful / dumb luck:



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

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

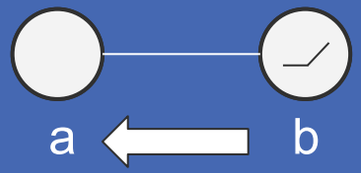
Backpropagation challenge: ReLU



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

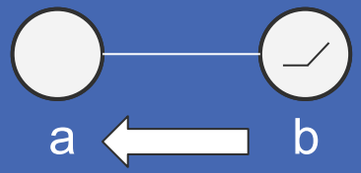
Backpropagation challenge: ReLU

$$b = a, \quad a > 0$$
$$= 0, \quad \text{otherwise}$$



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

Backpropagation challenge: ReLU

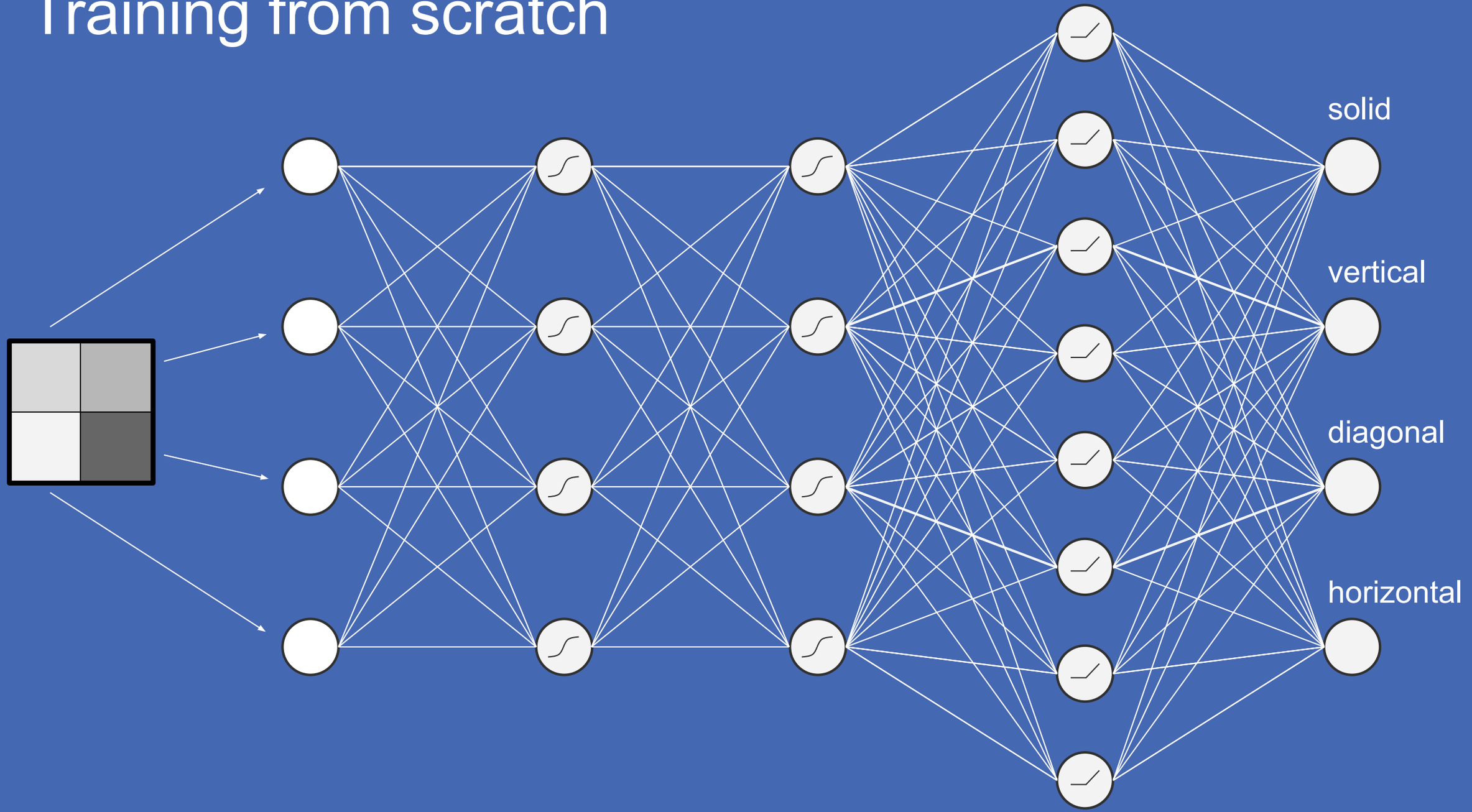


$$\frac{\partial err}{\partial a} = \frac{\partial b}{\partial a} * \frac{\partial err}{\partial b}$$

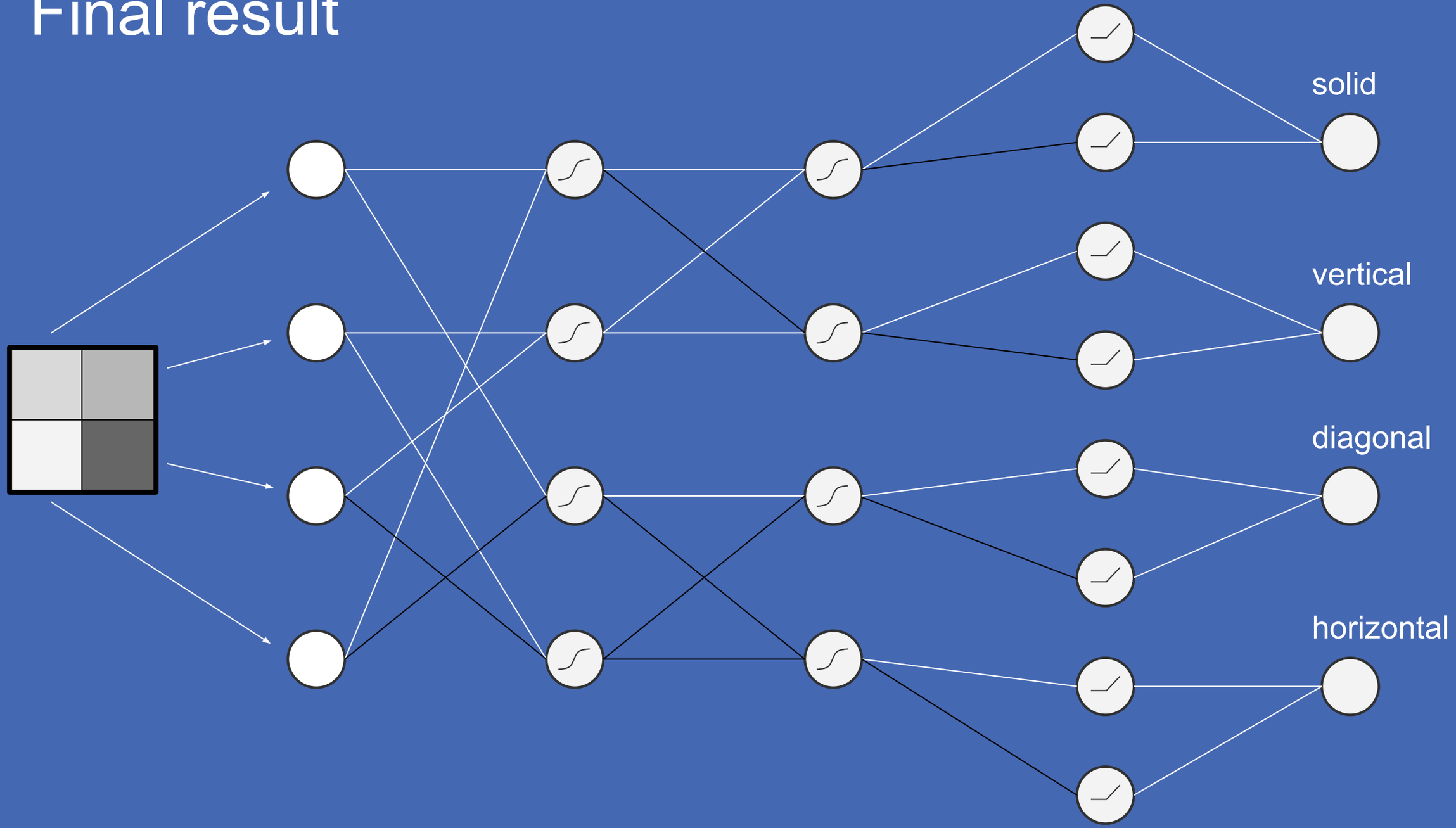
$$b = a, \quad a > 0$$
$$= 0, \quad \text{otherwise}$$

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

Training from scratch

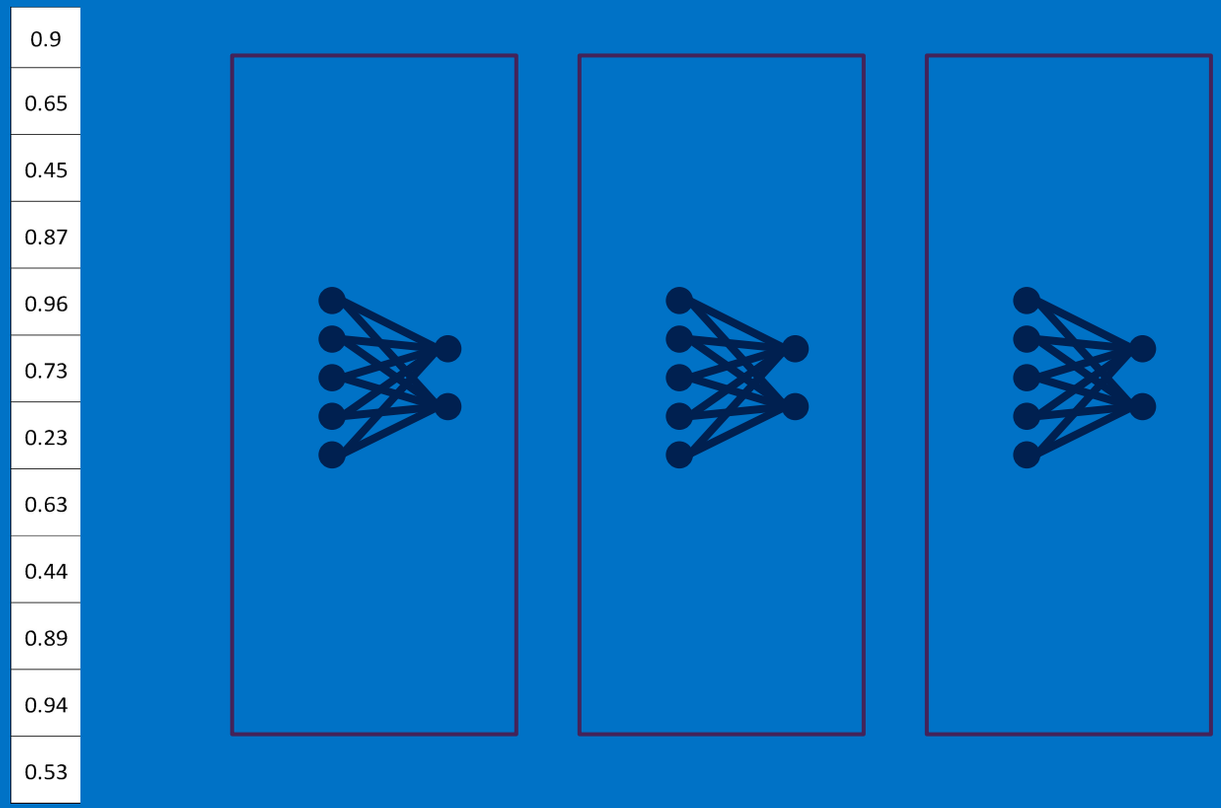


Final result



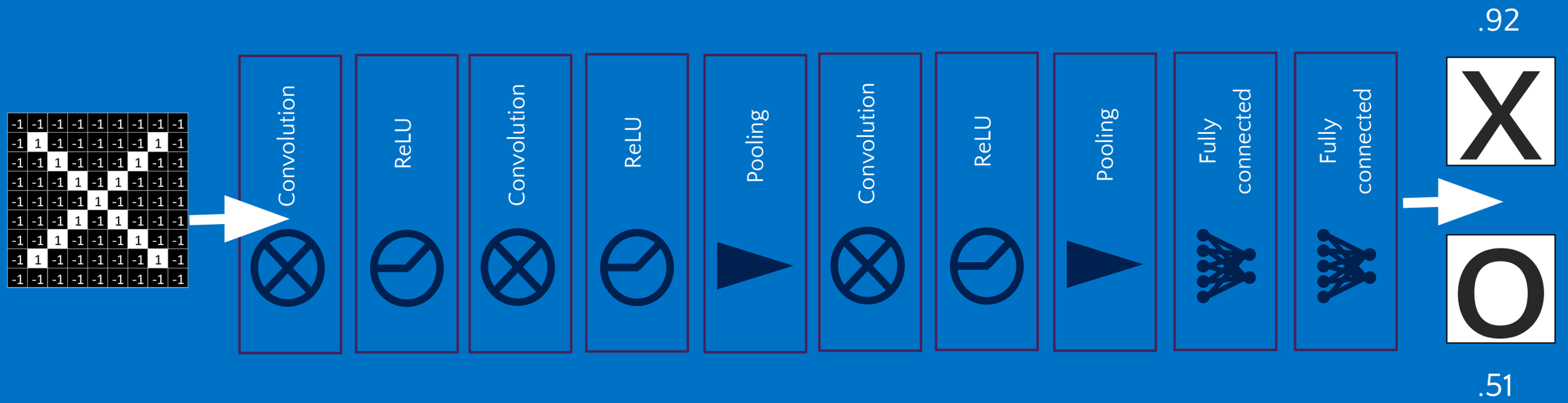
Fully connected layer

These can also be stacked.



Putting it all together

A set of pixels becomes a set of votes.



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?

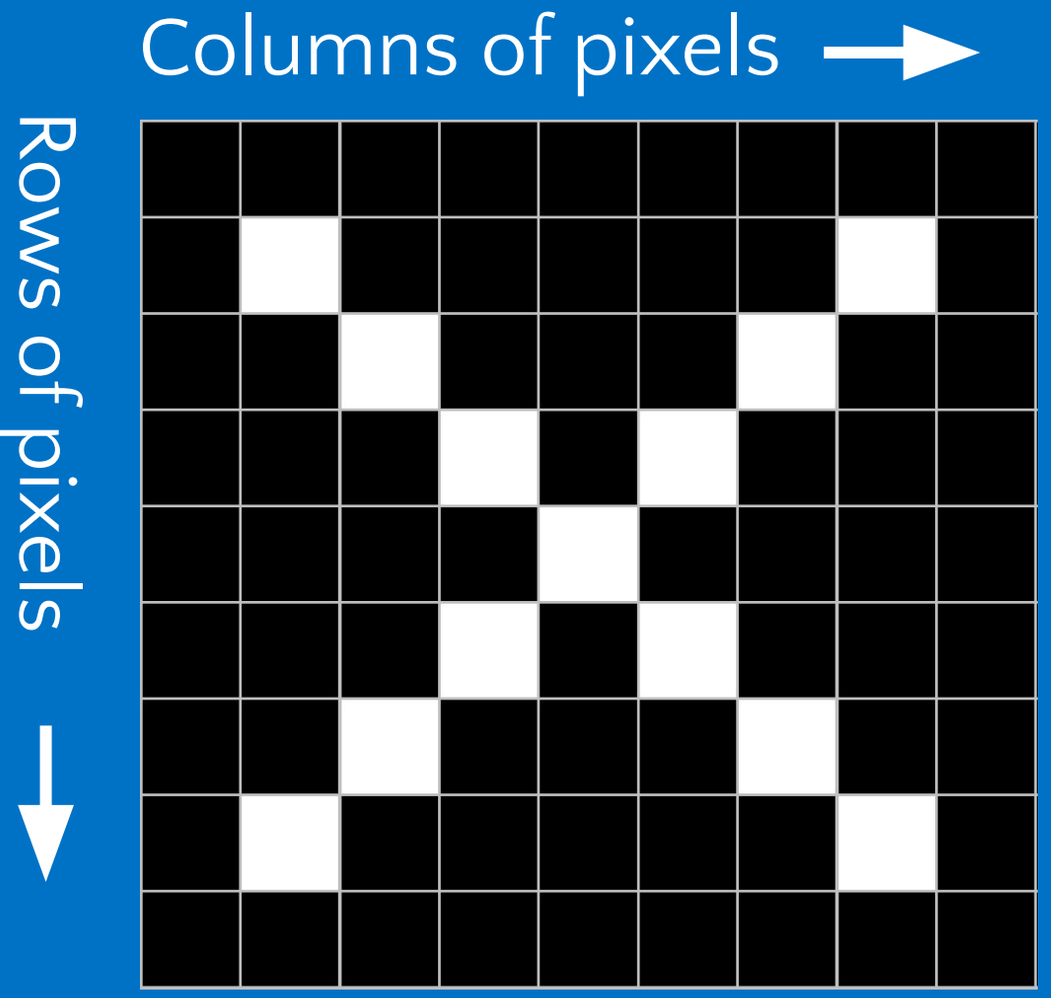
In what order?

Not just images

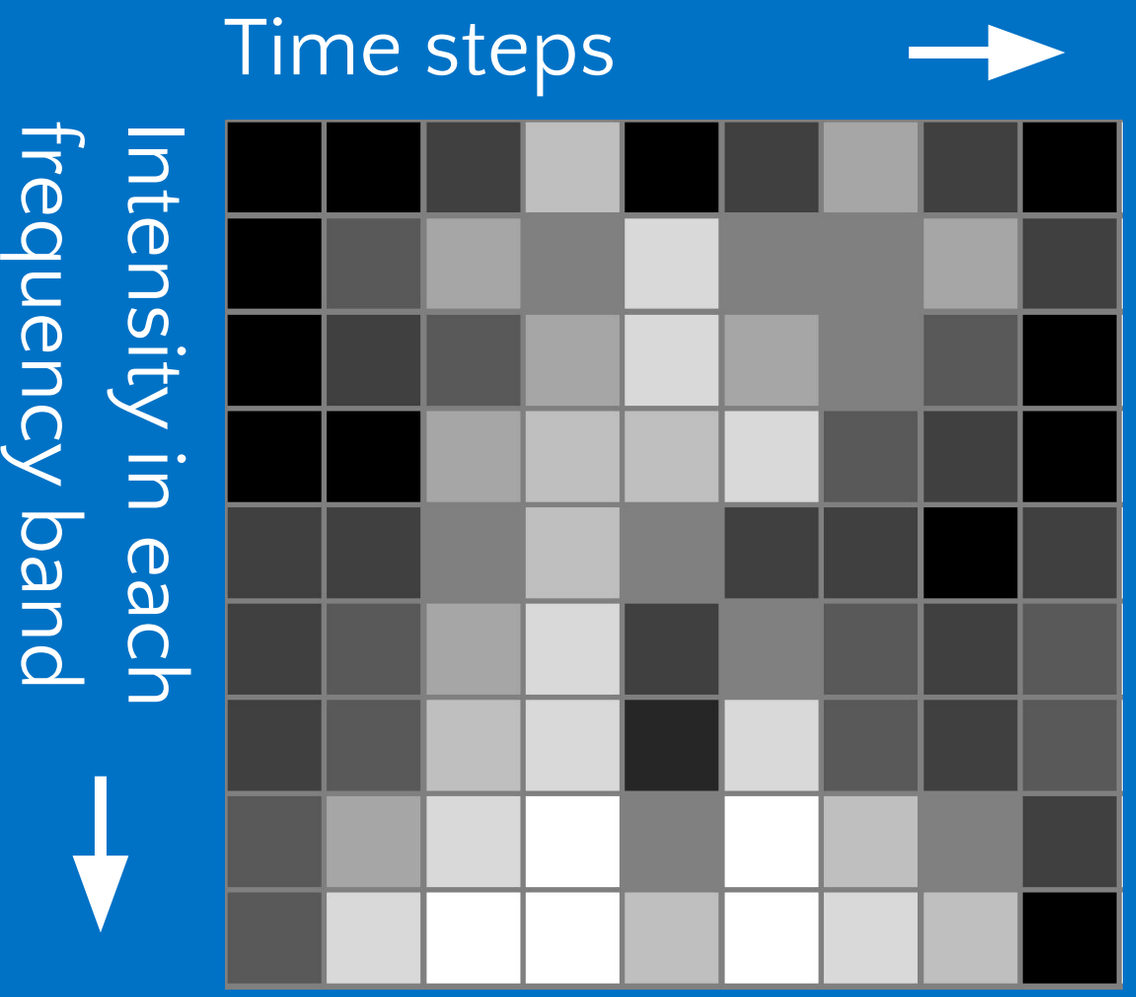
Any 2D (or 3D) data.

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

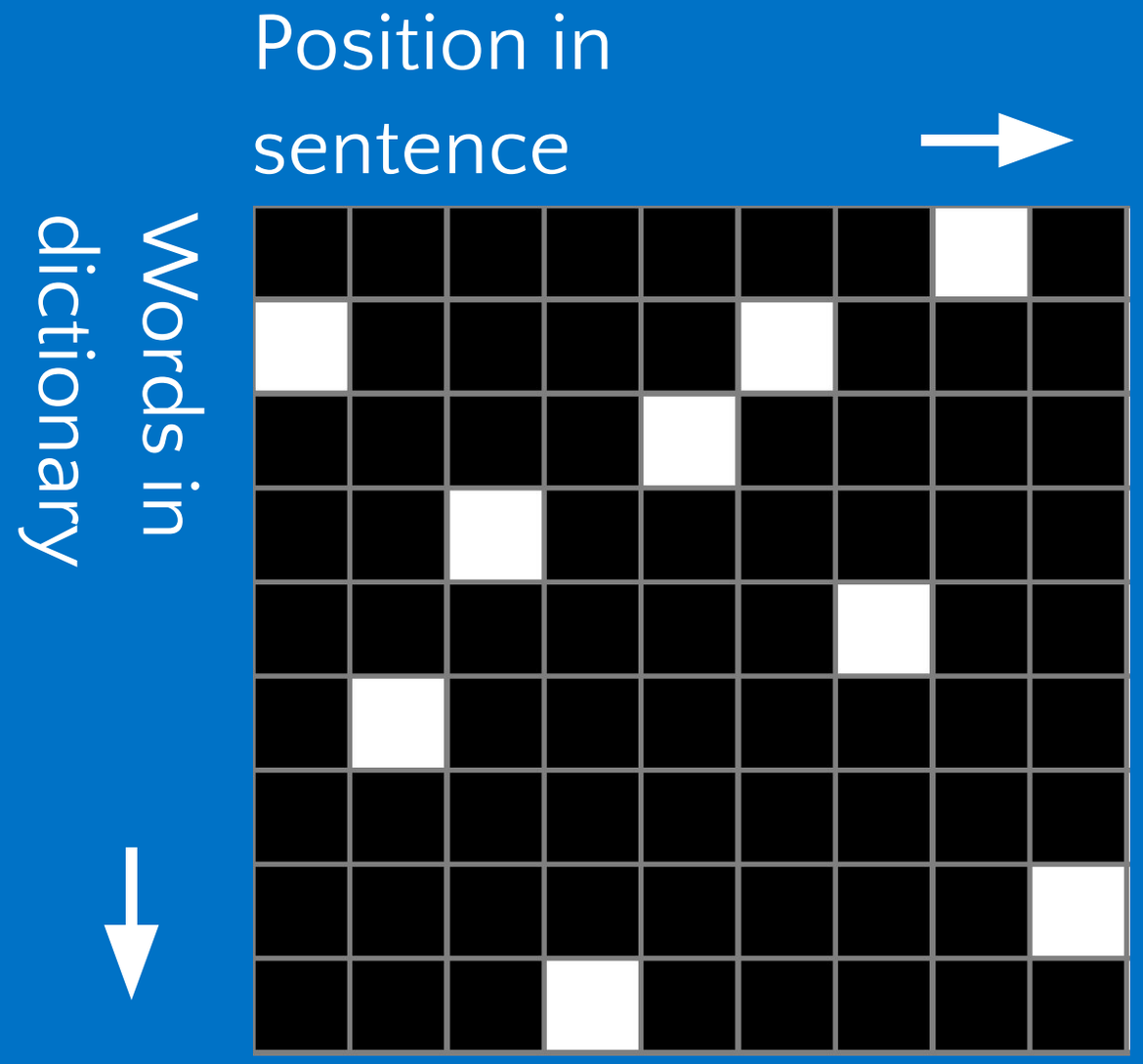
Images



Sound



Text



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

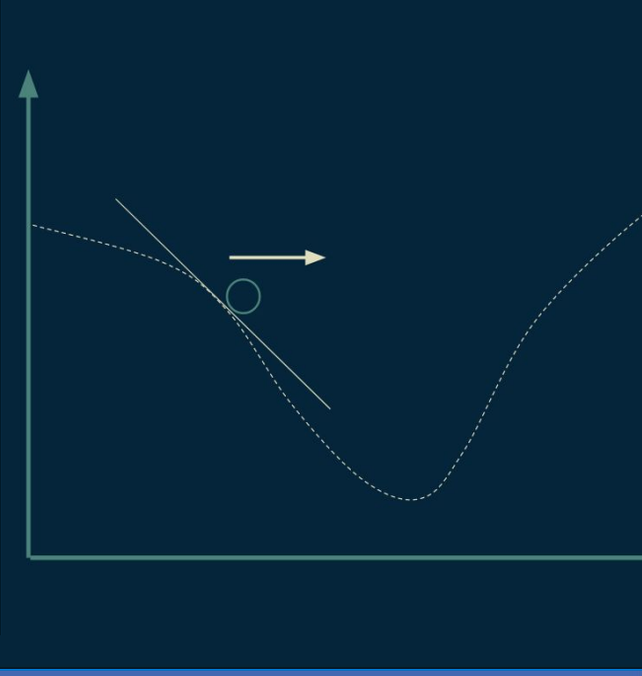
If your data is just as useful after swapping any of your columns with each other, then you can't use Convolutional Neural Networks.

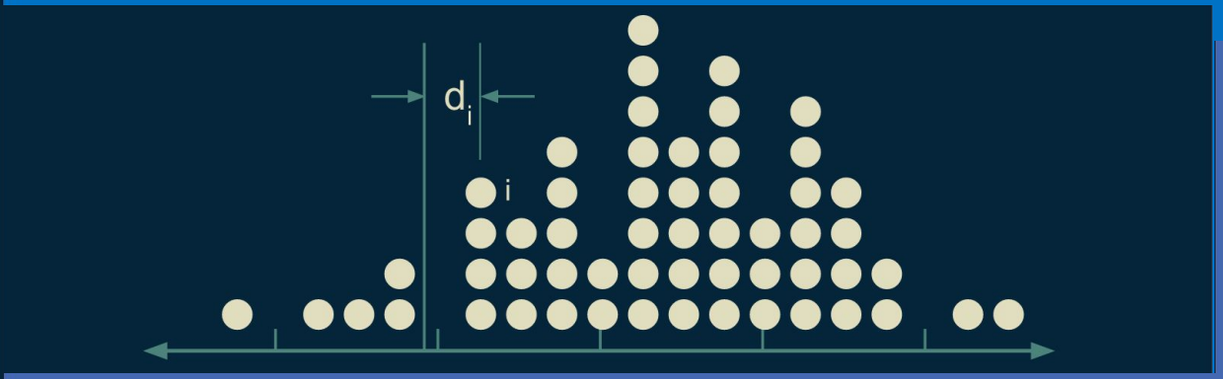
In a nutshell

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





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