Deep Learning Introduction

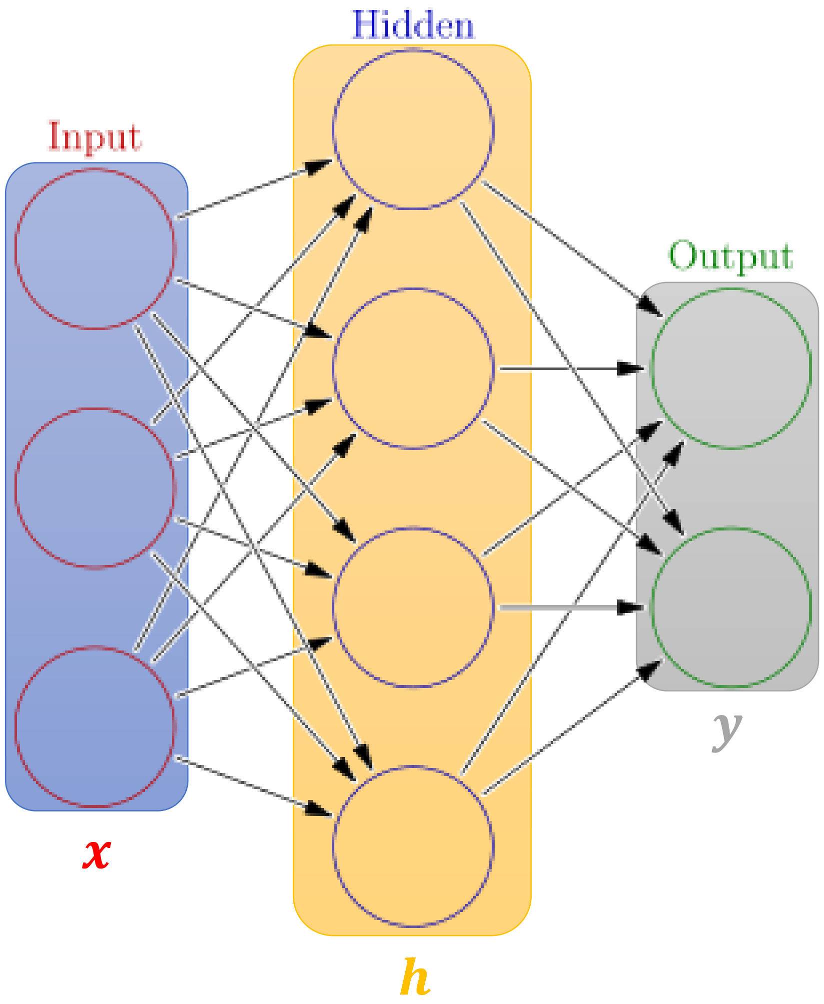
Md Modasshir

Topics

1. Introduction to Neural Network
2. Backpropagation and Gradient Descent
3. Edge Detection
4. Convolution as a layer
5. Convolutional Network

Topics

1. Introduction to Neural Network
2. Backpropagation and Gradient Descent
3. Edge Detection
4. Convolution as a layer
5. Convolutional Network



Weights

$$h = \sigma(\mathbf{W}_1 x + b_1)$$

$$y = \sigma(W_2 h + b_2)$$

Activation functions

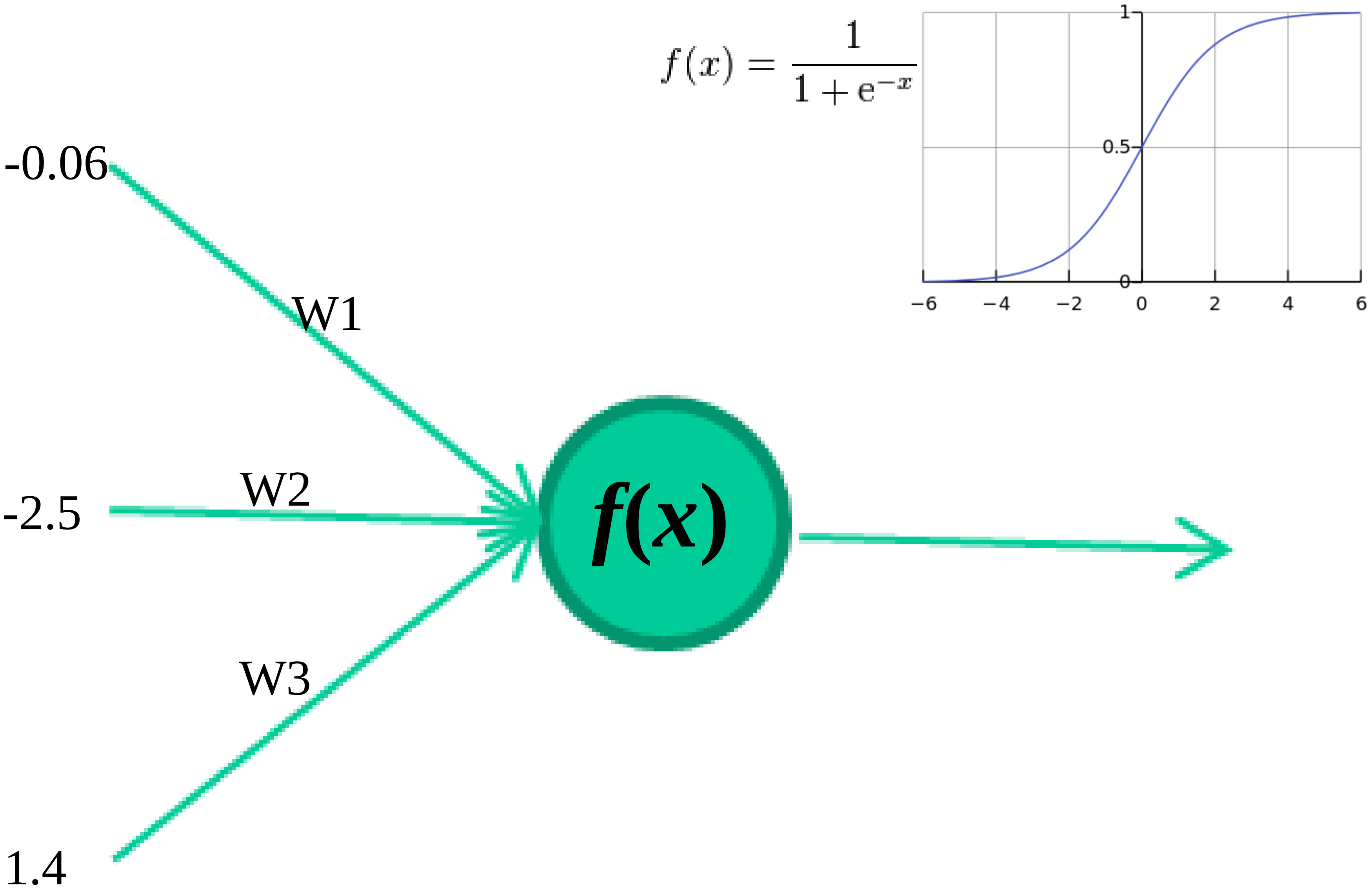
4 + 2 = 6 neurons (not counting inputs)

[3 x 4] + [4 x 2] = 20 weights

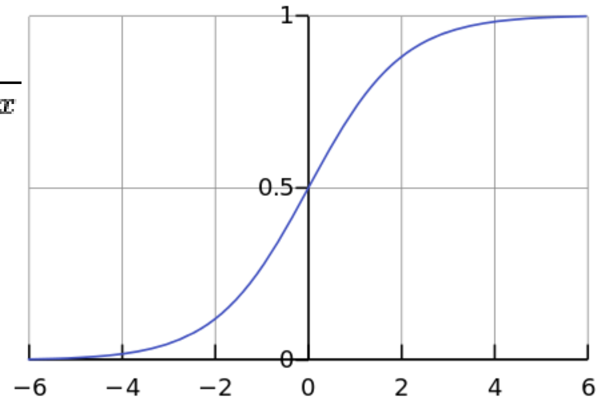
4 + 2 = 6 biases

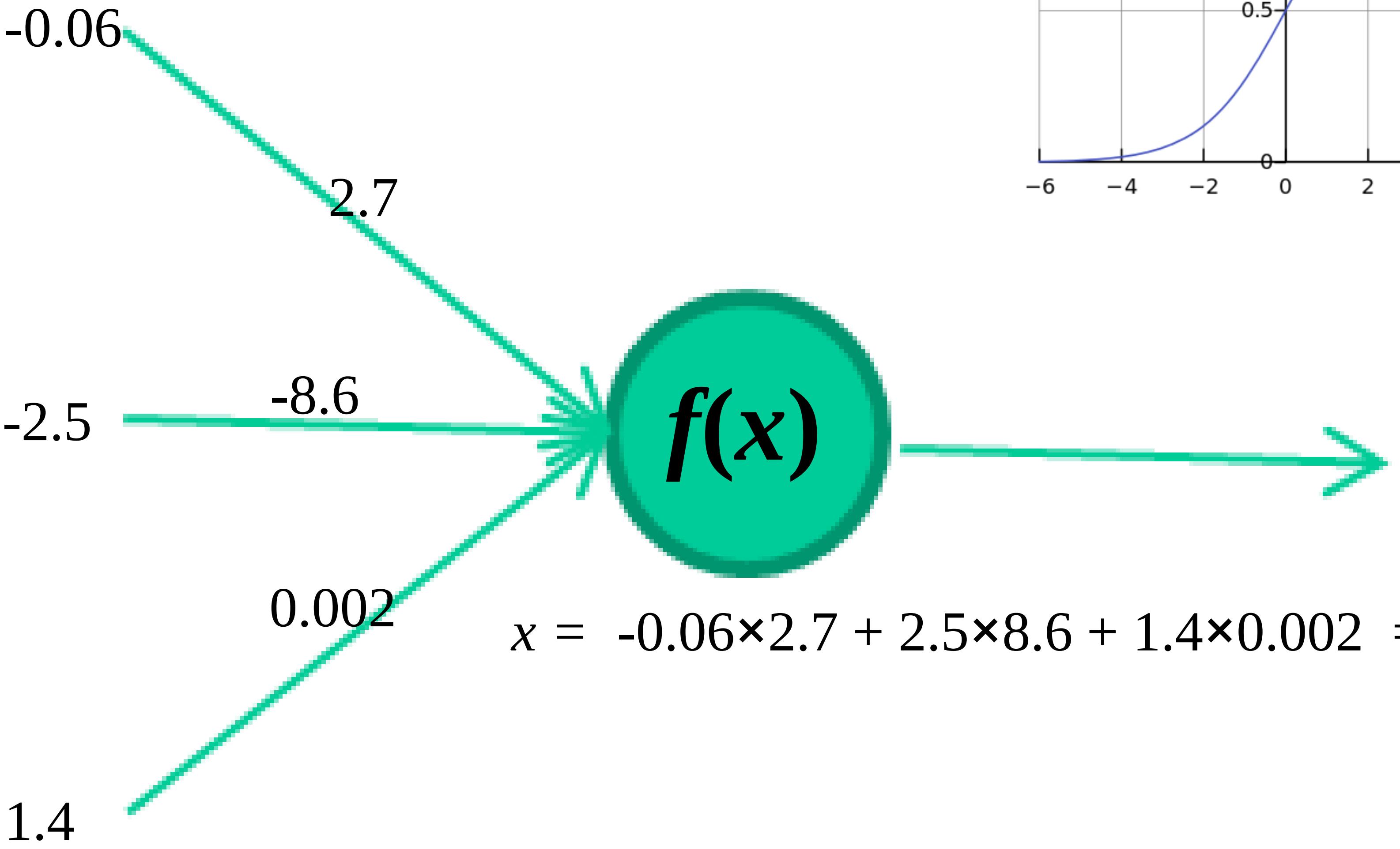
---

26 learnable **parameters**


$f(x) = \frac{1}{1+e^{-x}}$
-0.06
-2.5
1.4
W1
W2
W3
f(x)
1
0.5
0
−6
−4
−2
0
2
4
6

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

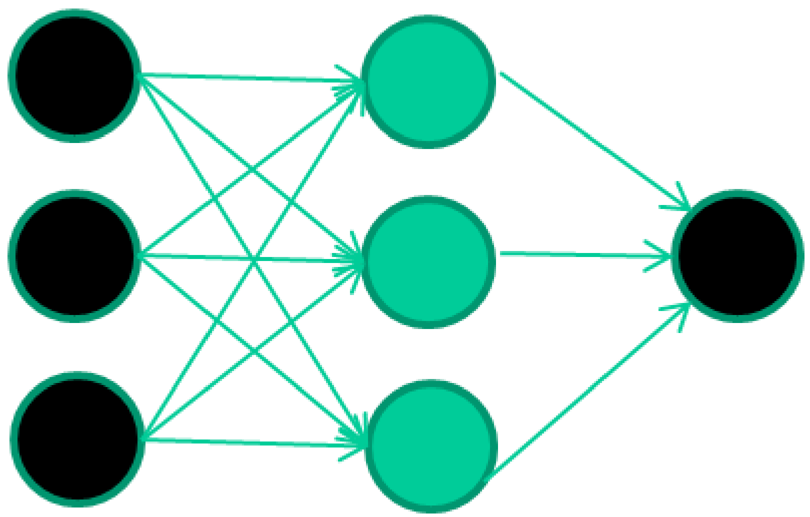




$$x = -0.06\times2.7 + 2.5\times8.6 + 1.4\times0.002 = 21.34$$

A dataset

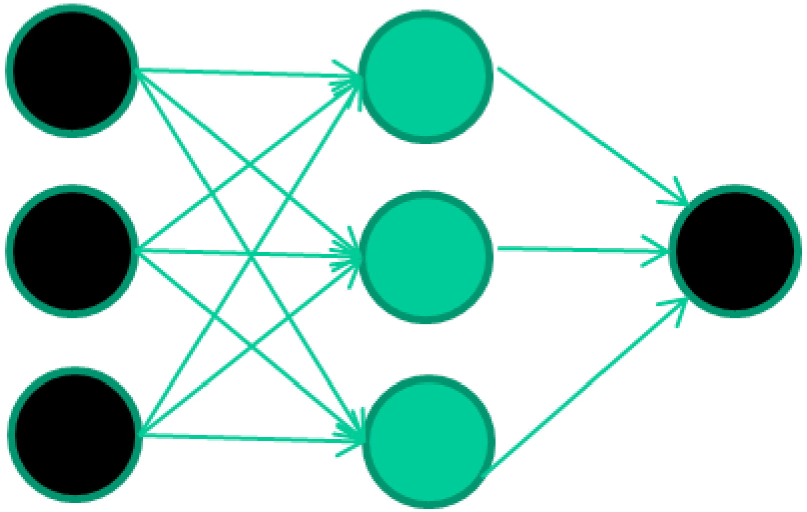
Fields			***class***
1.4	2.7	1.9	0
3.8	3.4	3.2	0
6.4	2.8	1.7	1
4.1	0.1	0.2	0
etc …			



Training the neural network

Fields			***class***
1.4	2.7	1.9	0
3.8	3.4	3.2	0
6.4	2.8	1.7	1
4.1	0.1	0.2	0

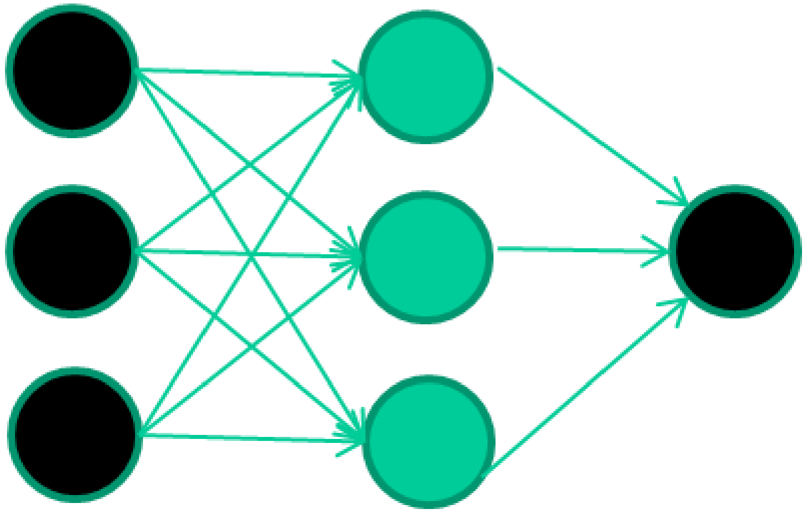
etc …



Training data

Fields			***class***
1.4	2.7	1.9	0
3.8	3.4	3.2	0
6.4	2.8	1.7	1
4.1	0.1	0.2	0
etc …			

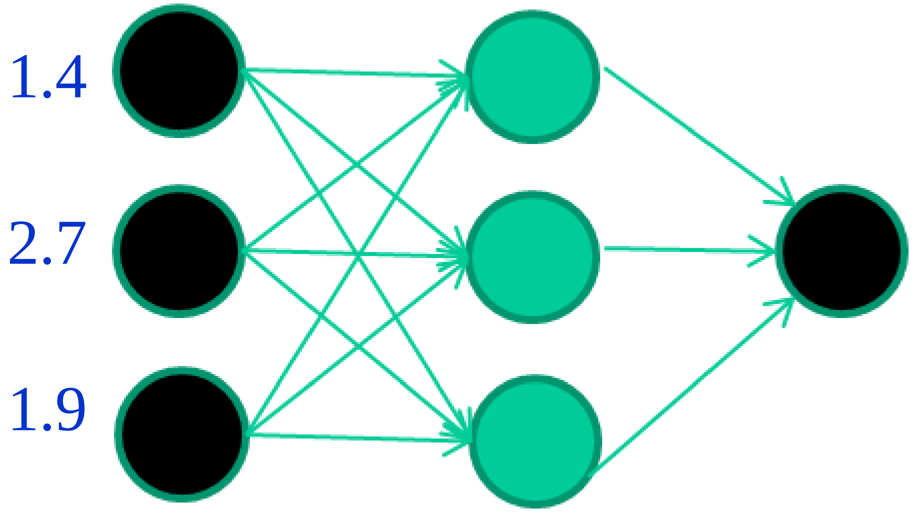
Initialise with random weights



Training data

Fields			***class***
1.4	2.7	1.9	0
3.8	3.4	3.2	0
6.4	2.8	1.7	1
4.1	0.1	0.2	0
etc …			

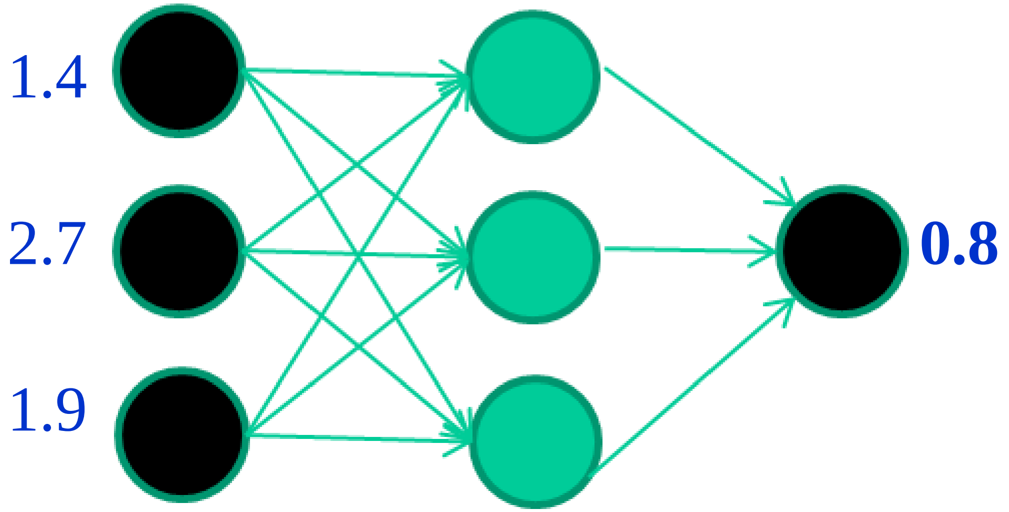
Present a training pattern



Training data

Fields			***class***
1.4	2.7	1.9	0
3.8	3.4	3.2	0
6.4	2.8	1.7	1
4.1	0.1	0.2	0
etc …			

Feed it through to get output

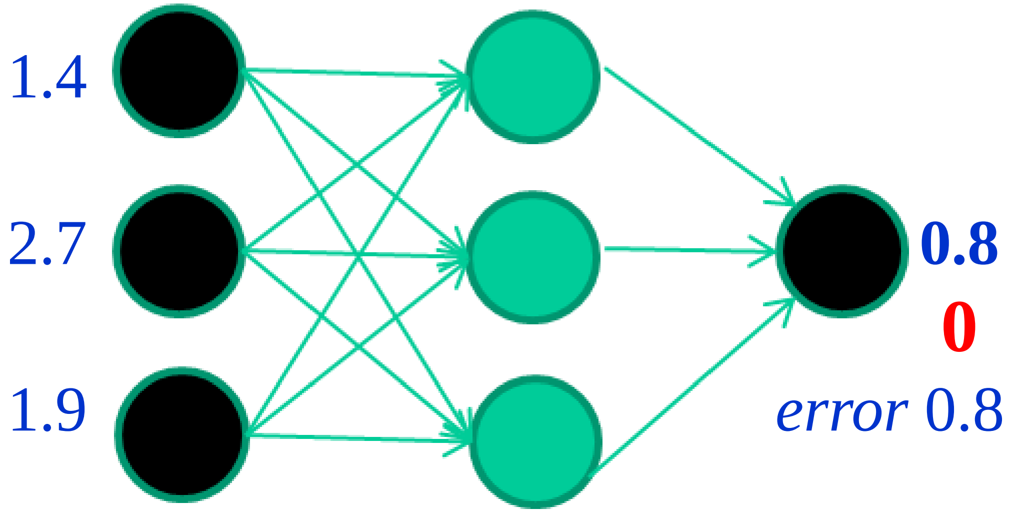


Training data

Fields			***class***
1.4	2.7	1.9	0
3.8	3.4	3.2	0
6.4	2.8	1.7	1
4.1	0.1	0.2	0

etc …

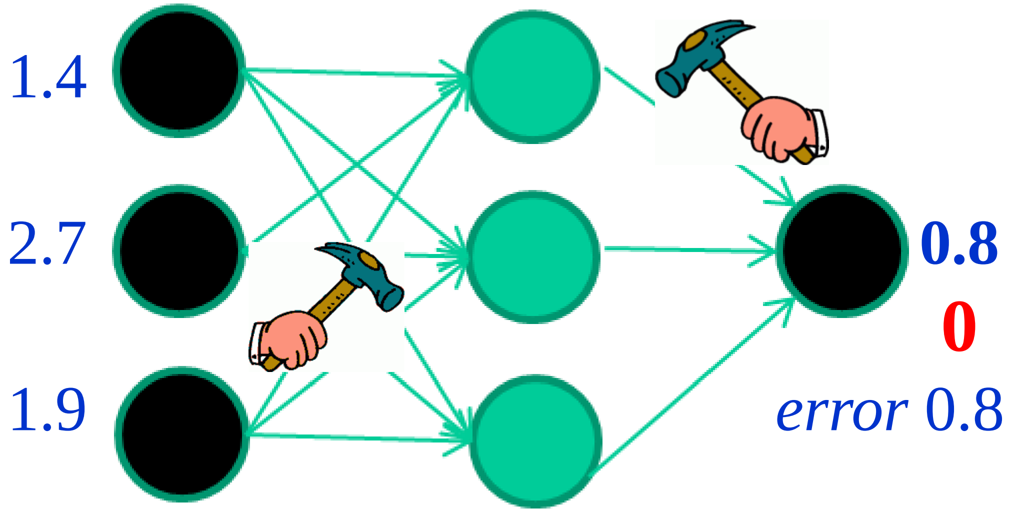
Compare with target output



Training data

Fields			***class***
1.4	2.7	1.9	0
3.8	3.4	3.2	0
6.4	2.8	1.7	1
4.1	0.1	0.2	0
etc …			

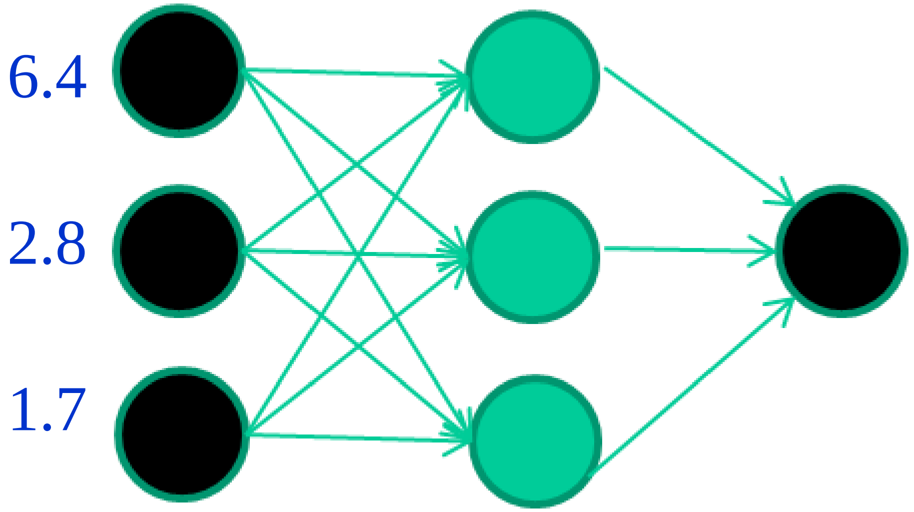
Adjust weights based on error



Training data

Fields			***class***
1.4	2.7	1.9	0
3.8	3.4	3.2	0
6.4	2.8	1.7	1
4.1	0.1	0.2	0
etc …			

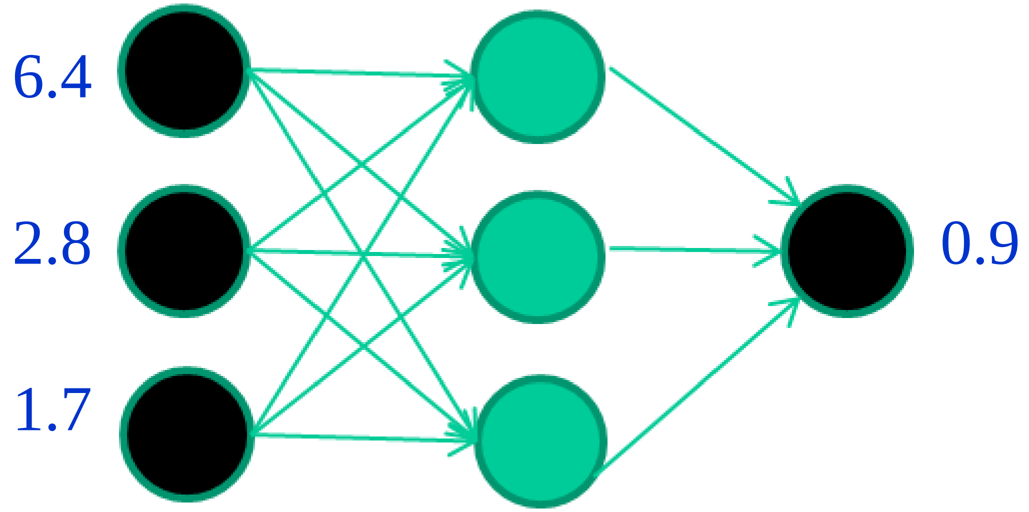
Present a training pattern



Training data

Fields			***class***
1.4	2.7	1.9	0
3.8	3.4	3.2	0
6.4	2.8	1.7	1
4.1	0.1	0.2	0
etc …			

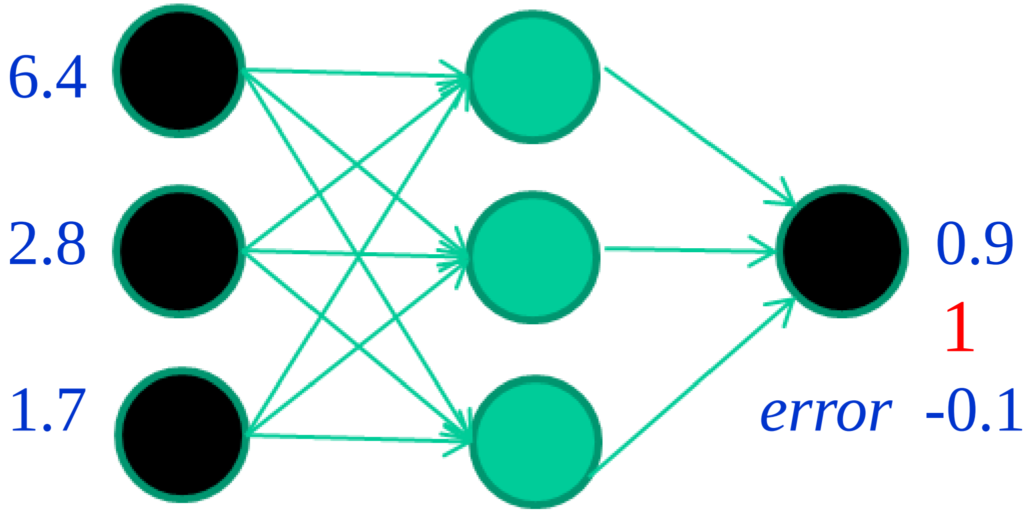
Feed it through to get output



Training data

Fields			***class***
1.4	2.7	1.9	0
3.8	3.4	3.2	0
6.4	2.8	1.7	1
4.1	0.1	0.2	0
etc …			

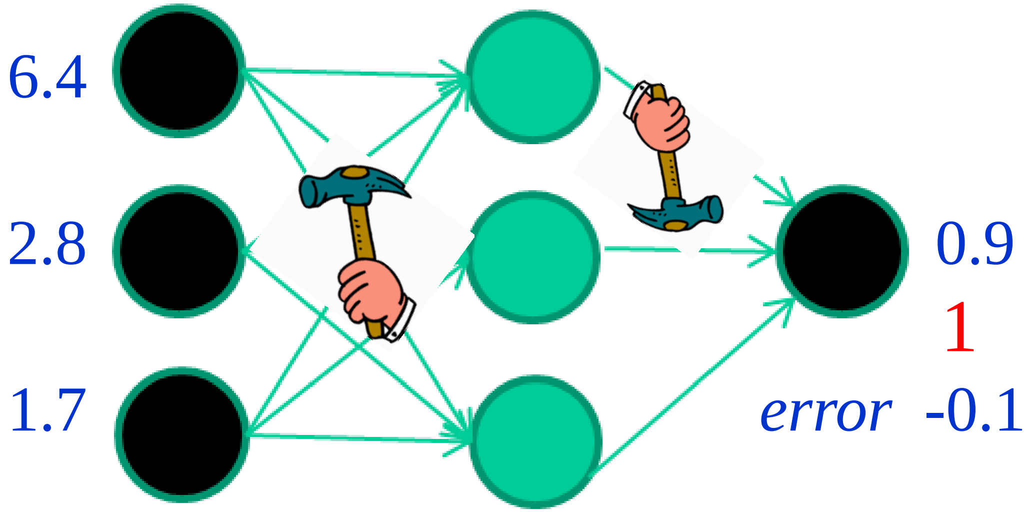
Compare with target output



Training data

Fields			***class***
1.4	2.7	1.9	0
3.8	3.4	3.2	0
6.4	2.8	1.7	1
4.1	0.1	0.2	0
etc …			

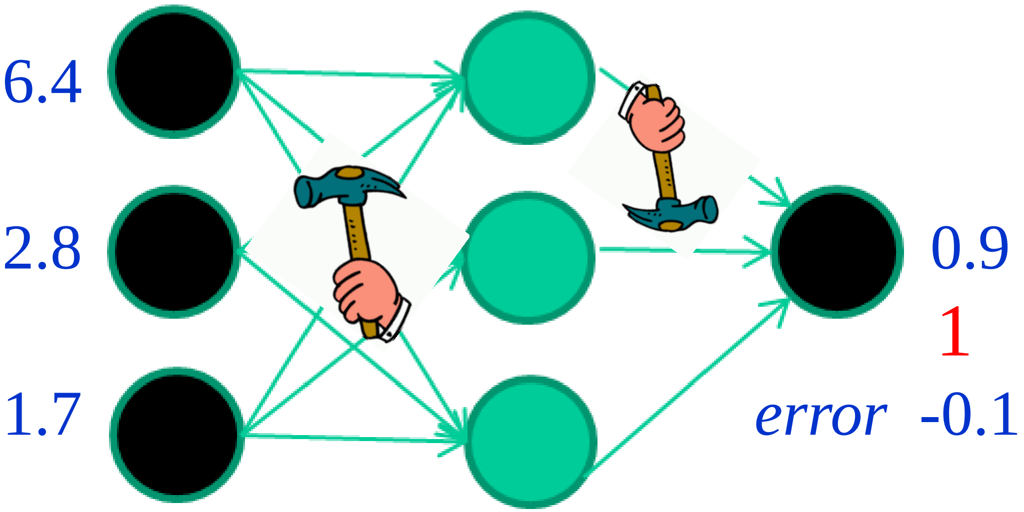
Adjust weights based on error



Training data

Fields			***class***
1.4	2.7	1.9	0
3.8	3.4	3.2	0
6.4	2.8	1.7	1
4.1	0.1	0.2	0
etc …			

And so on ….

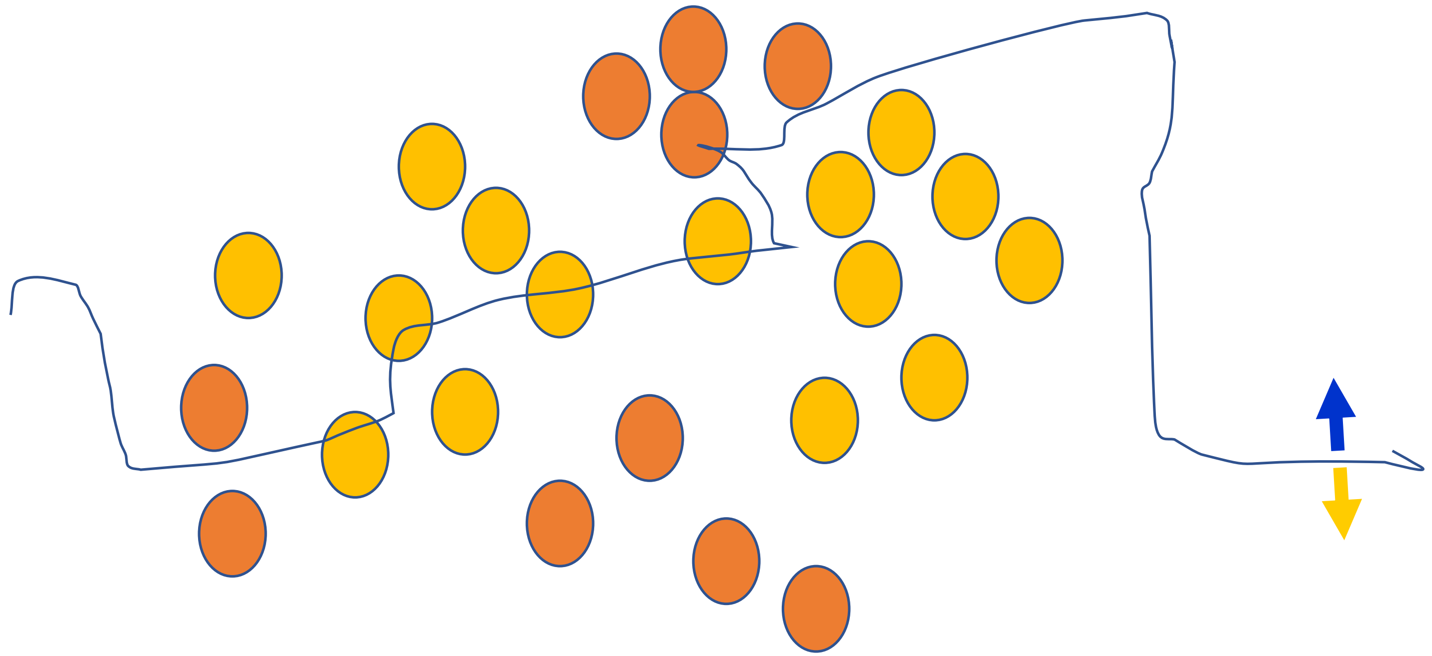


Repeat this thousands, maybe millions of times – each time taking a random training instance, and making slight weight adjustments

Algorithms for weight adjustment are designed to make changes that will reduce the error

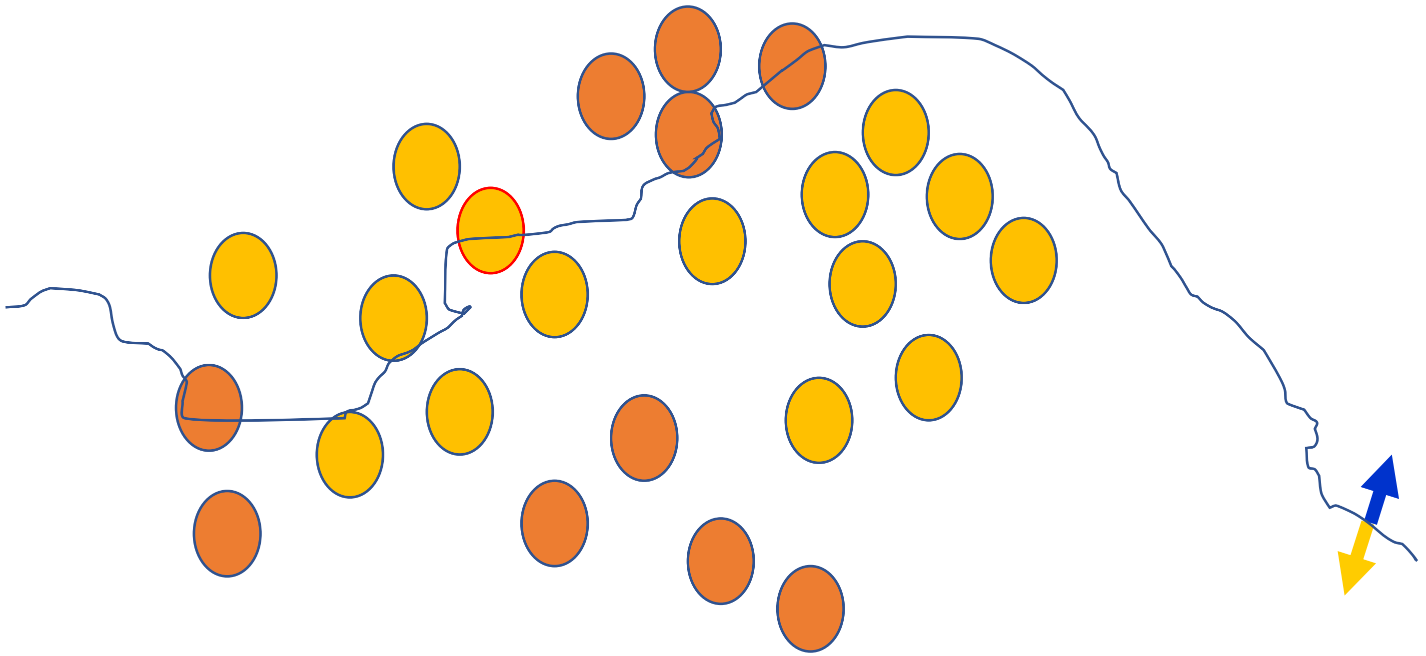
The decision boundary perspective…

Initial random weights



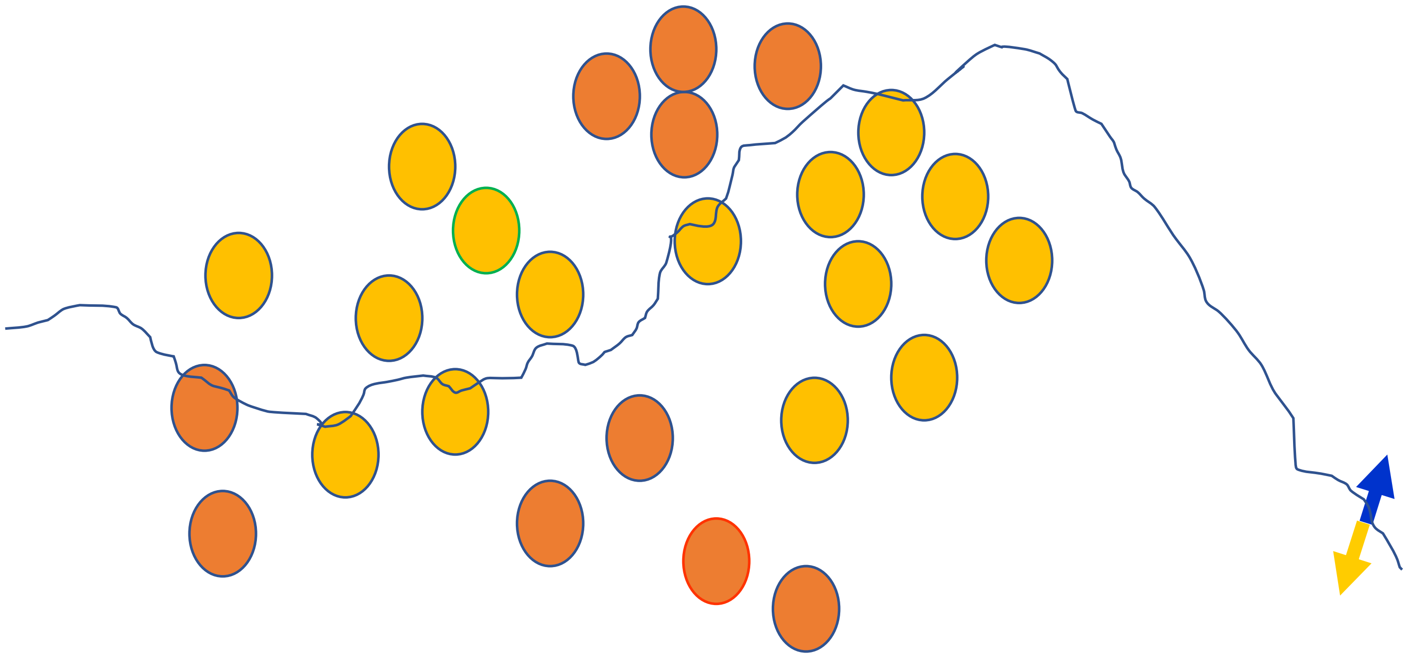
The decision boundary perspective…

Present a training instance / adjust the weights



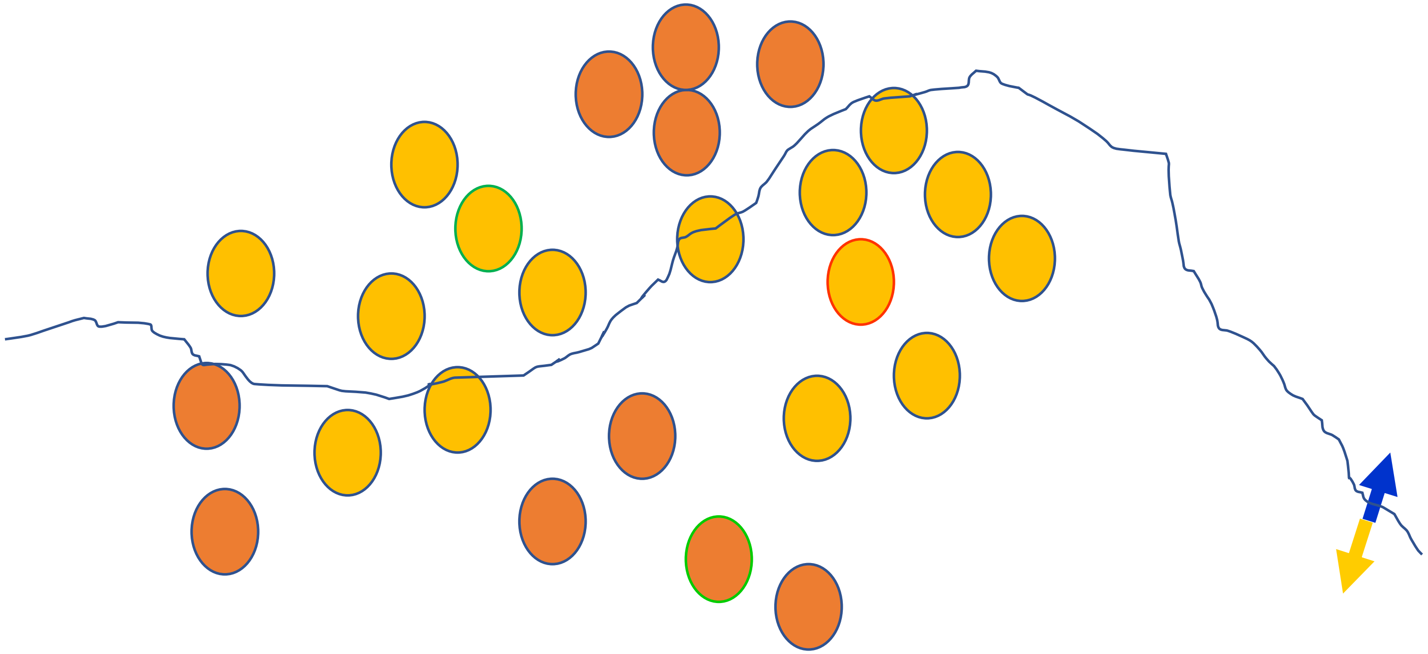
The decision boundary perspective…

Present a training instance / adjust the weights



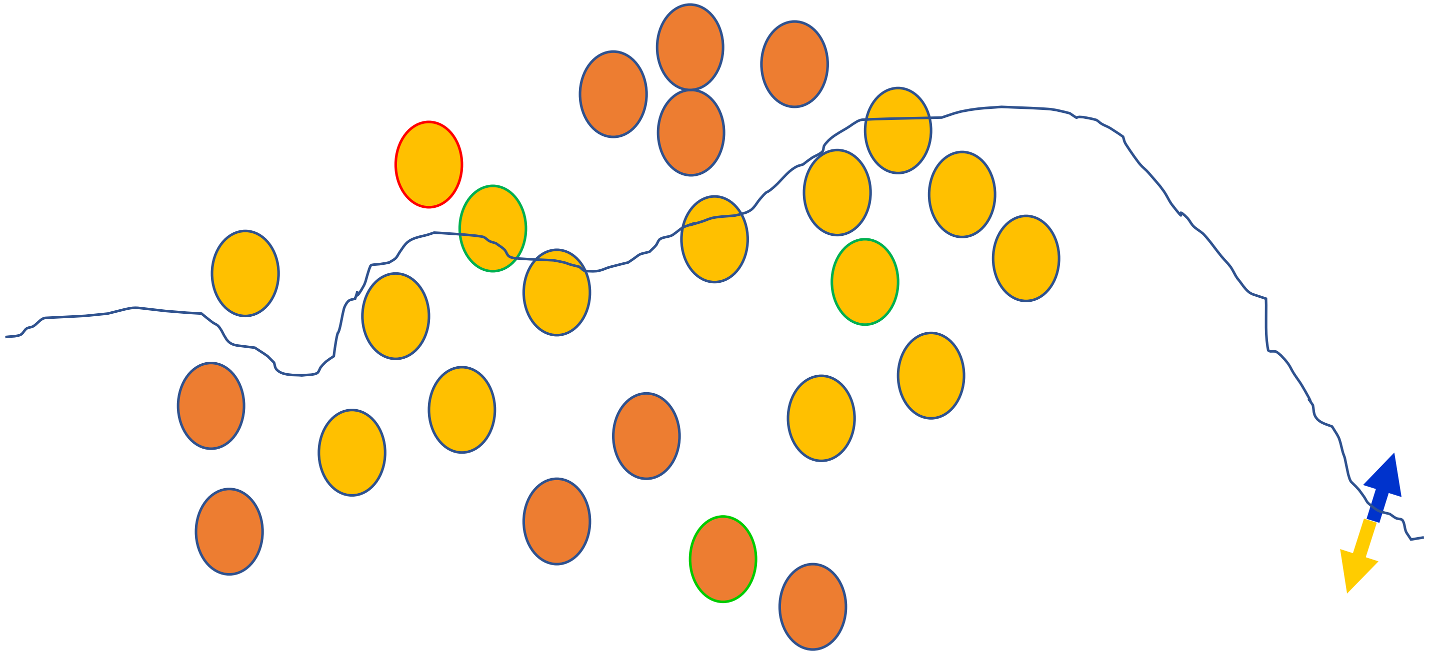
The decision boundary perspective…

Present a training instance / adjust the weights



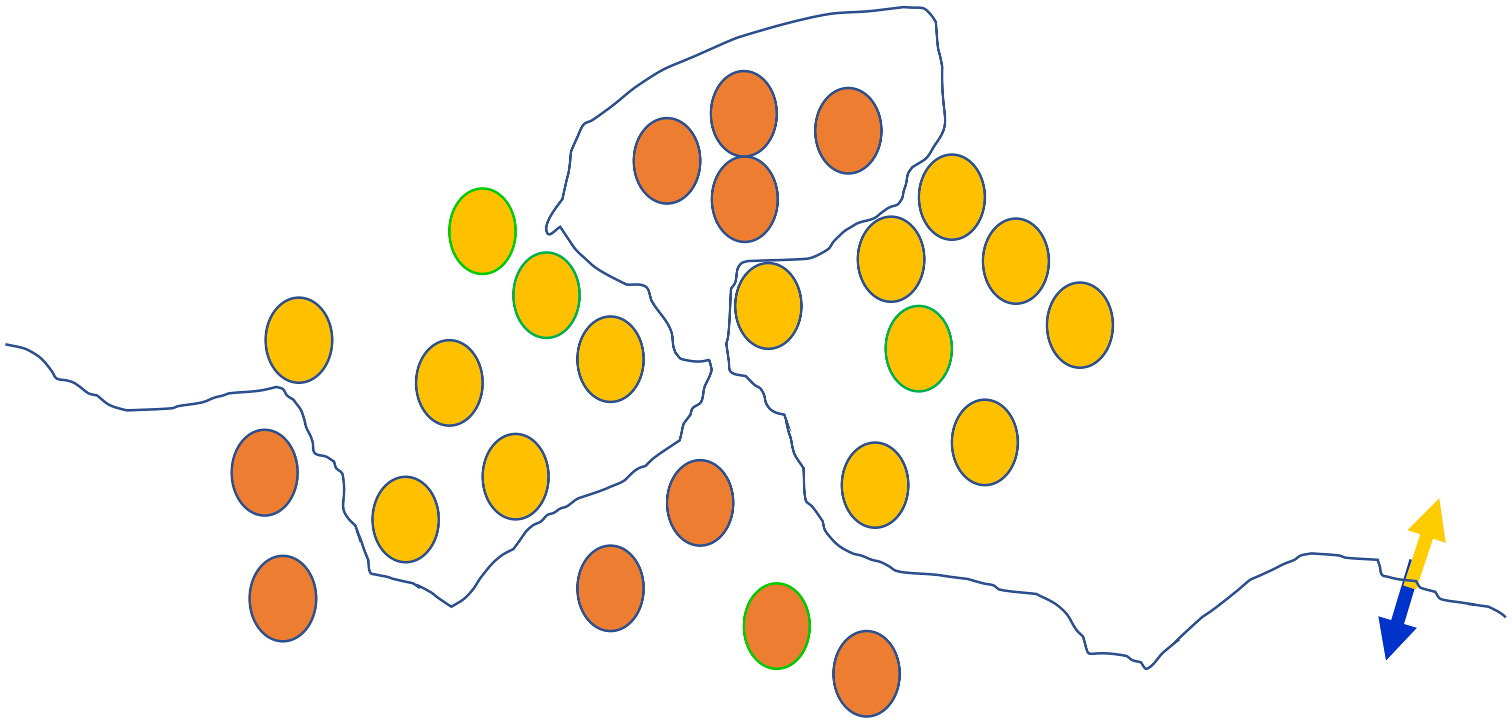
The decision boundary perspective…

Present a training instance / adjust the weights



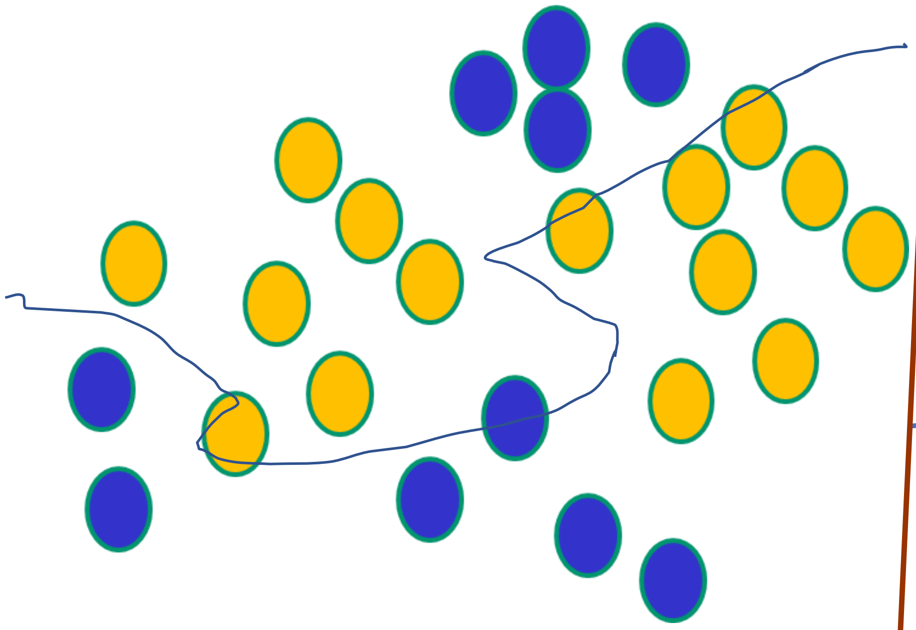
The decision boundary perspective…

Eventually ….

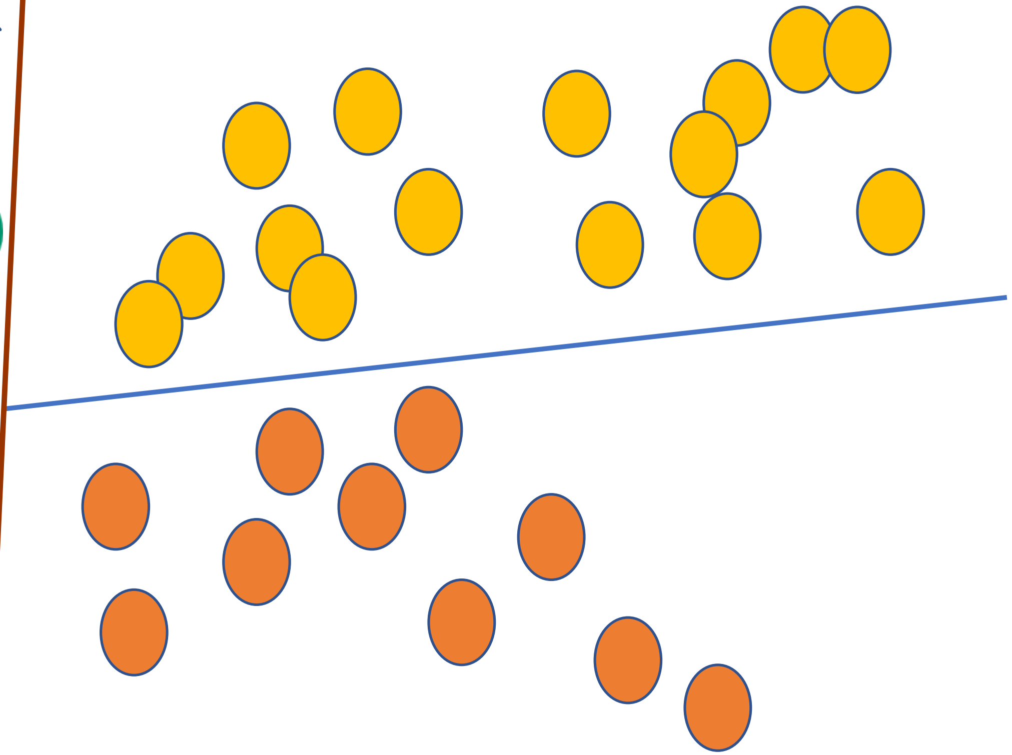


Some other ‘by the way’ points

NNs use nonlinear *f*(x) so they can draw complex boundaries, but keep the data unchanged



SVMs only draw straight lines, but they transform the data first in a way that makes that OK



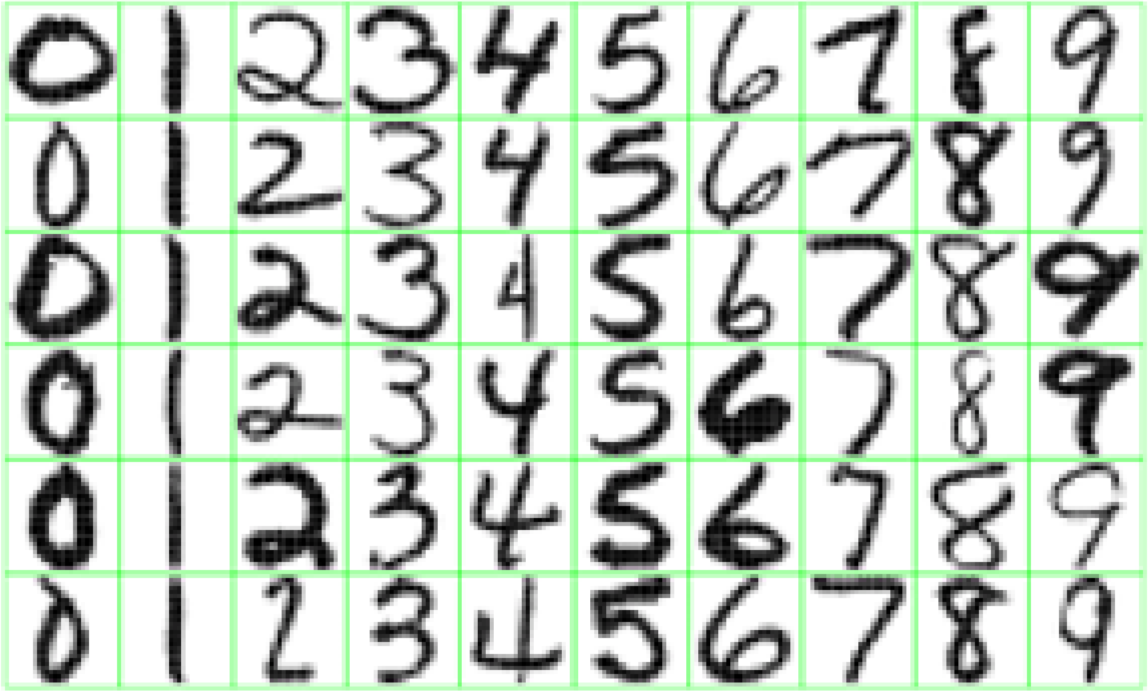


Figure 1.2: *Examples of handwritten digits from U.S. postal envelopes.*

What features might you expect a good NN to learn, when trained with data like this?

1

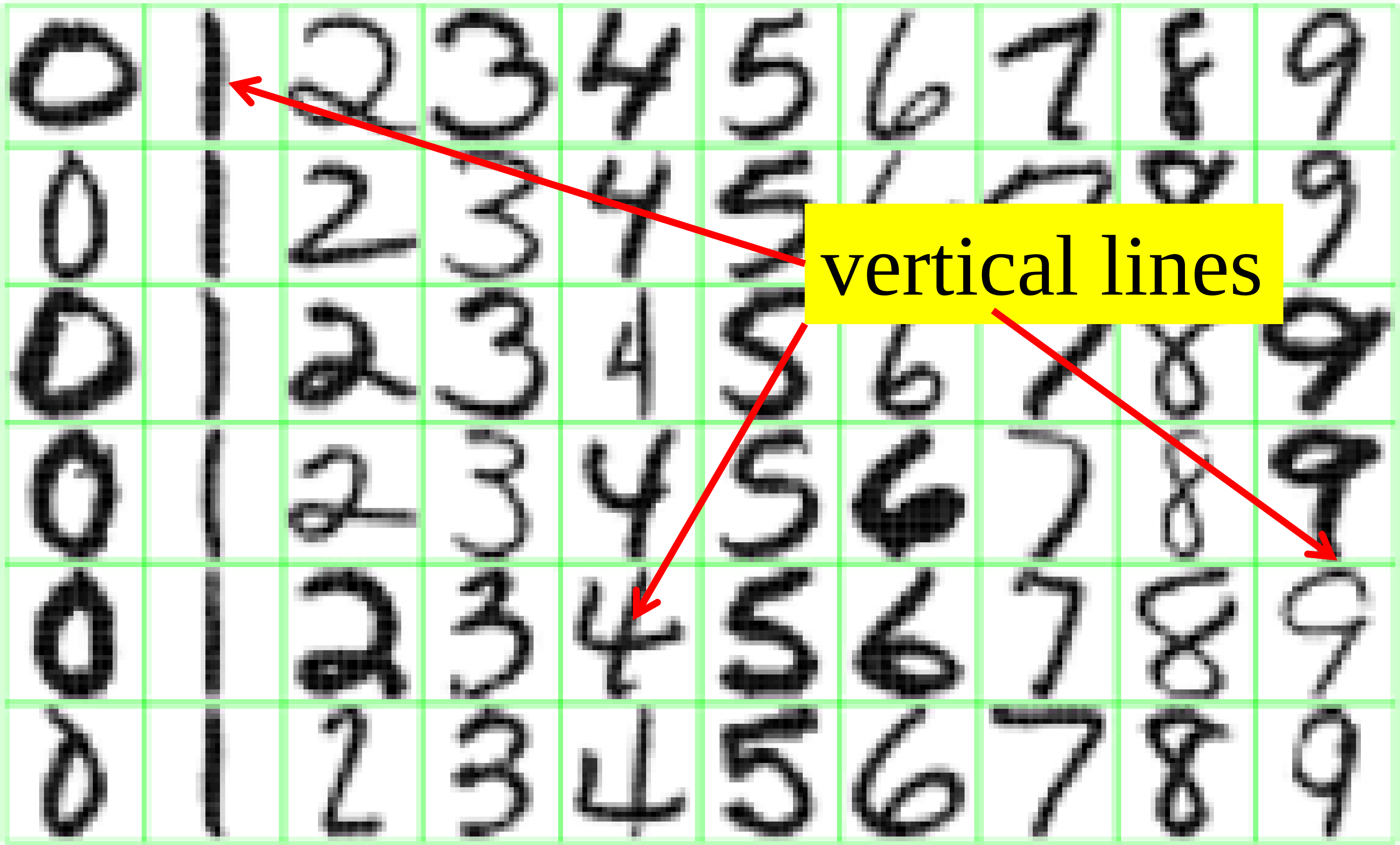


Figure 1.2: *Examples of handwritten digits from U.S. postal envelopes.*

1

Figure 1.2: *Examples of handwritten digits from U.S. postal envelopes.*

1

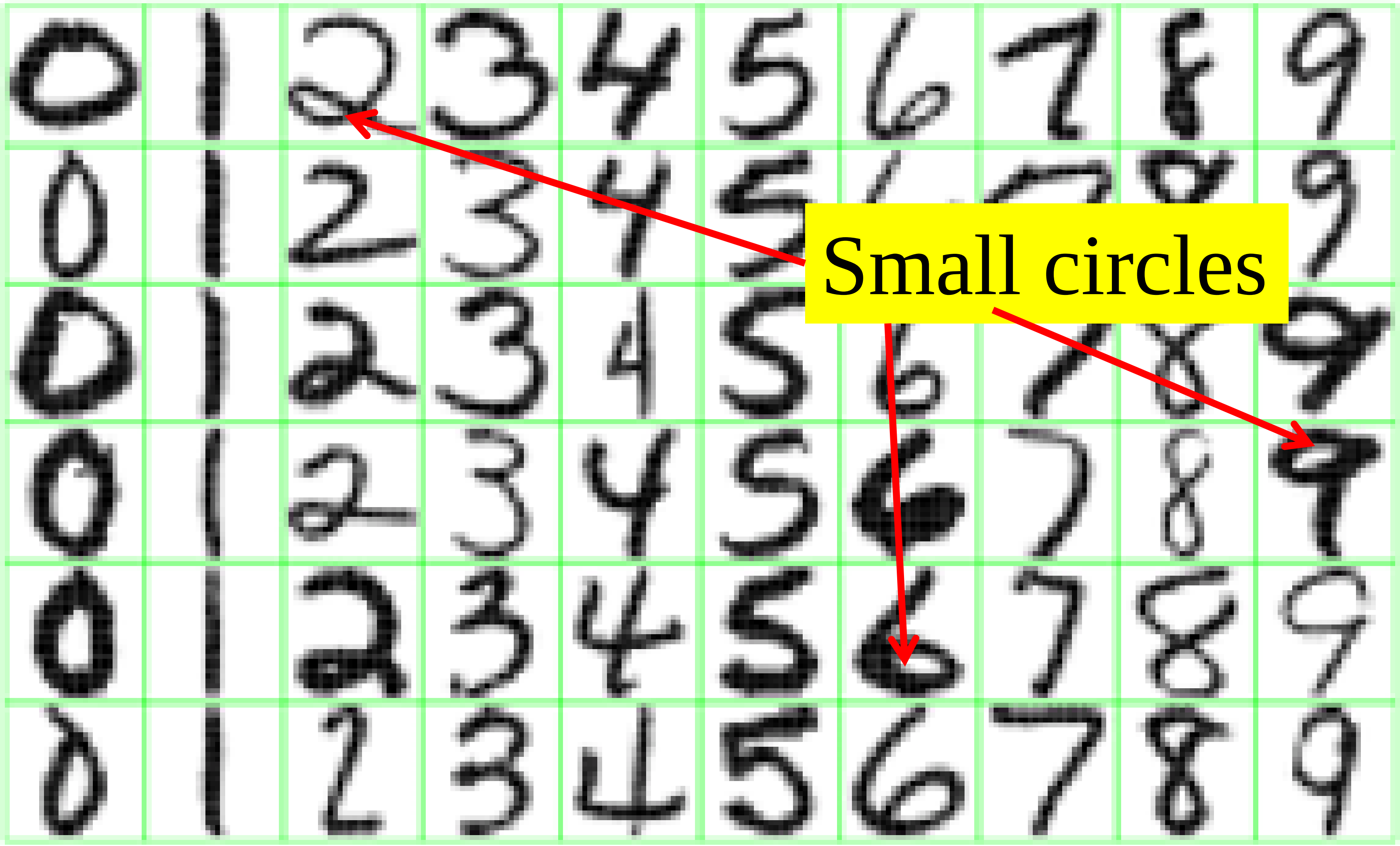


Figure 1.2: *Examples of handwritten digits from U.S. postal envelopes.*

1

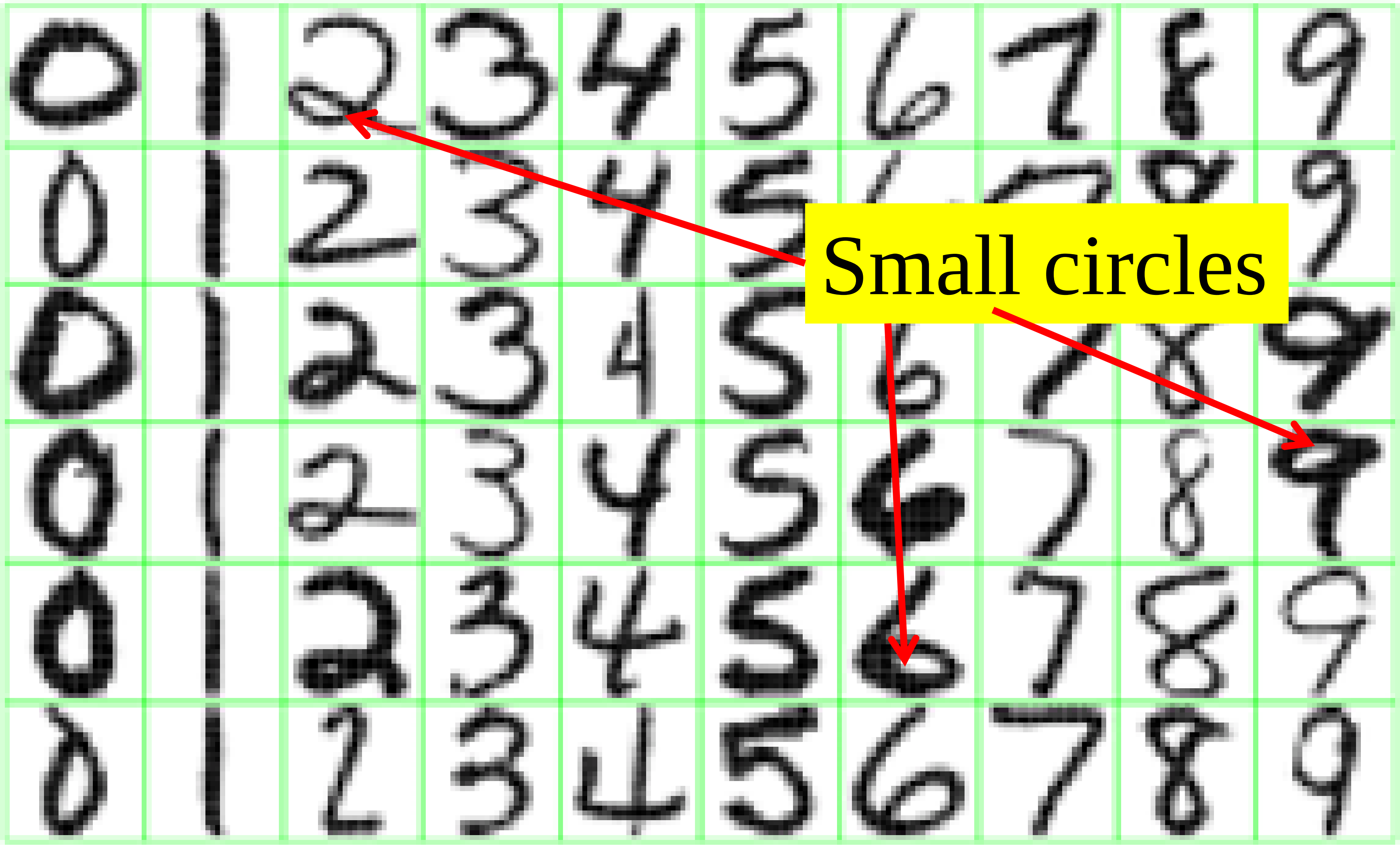
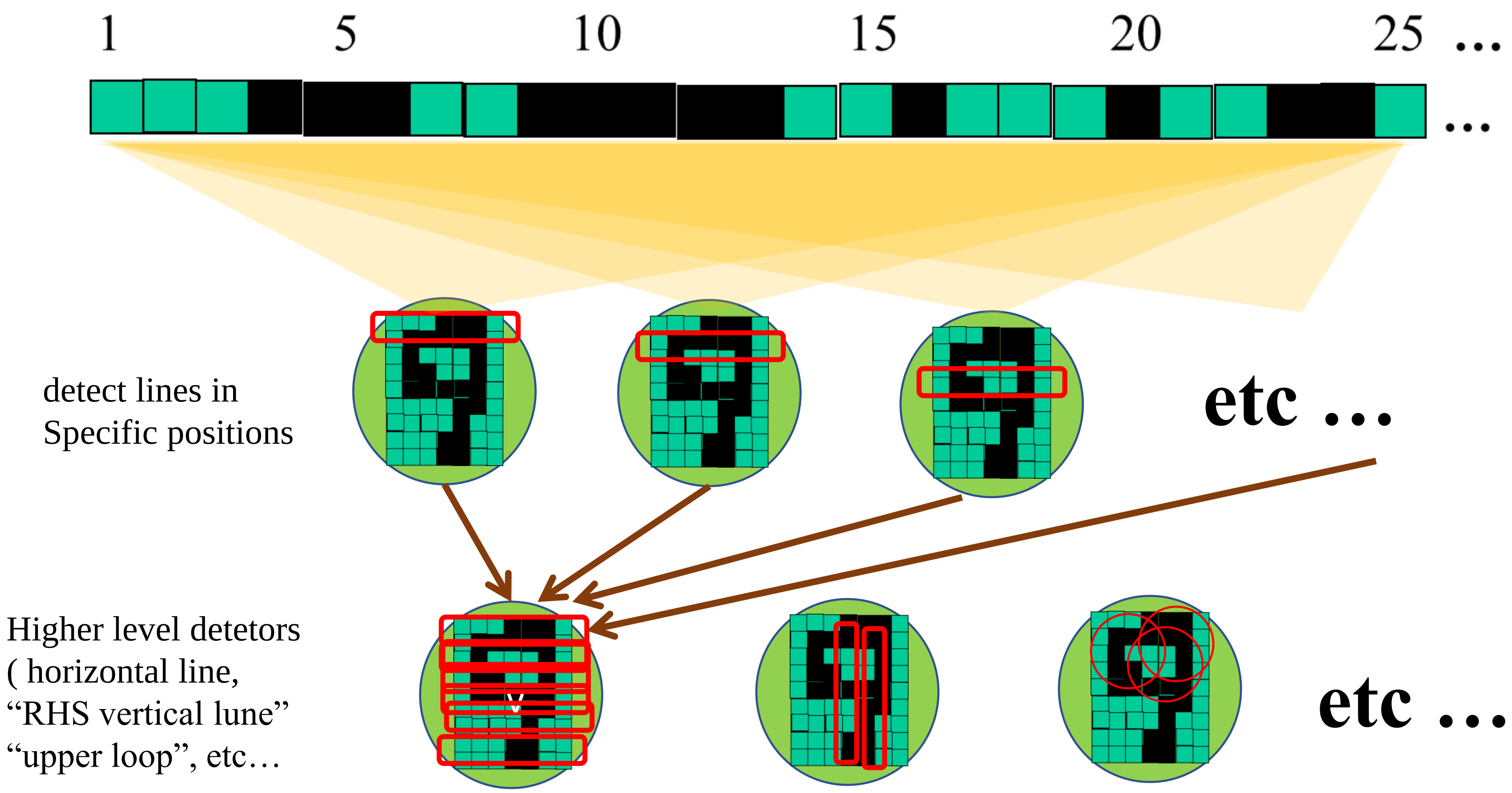


Figure 1.2: *Examples of handwritten digits from U.S. postal envelopes.*

successive layers can learn higher-level features …



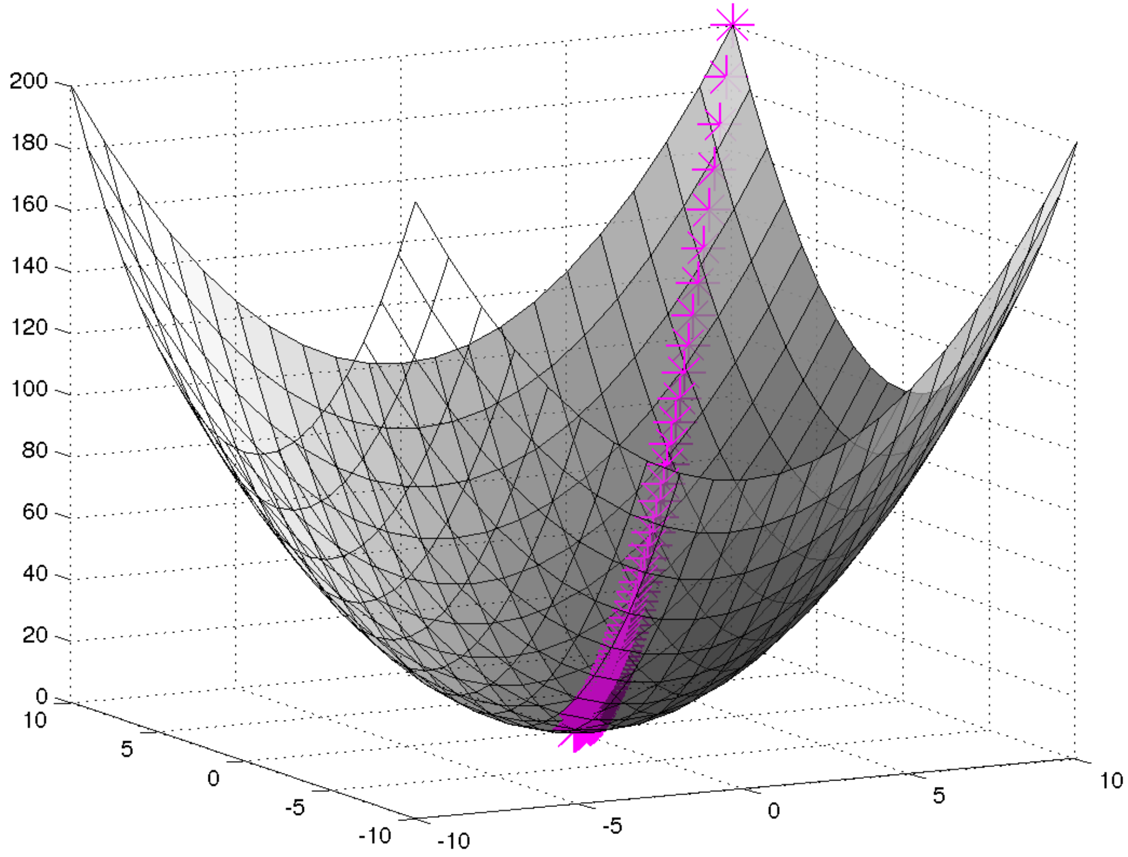
Topics

1. Introduction to Neural Network
2. Backpropagation and Gradient Descent
3. Edge Detection
4. Convolution
5. Convolutional Network

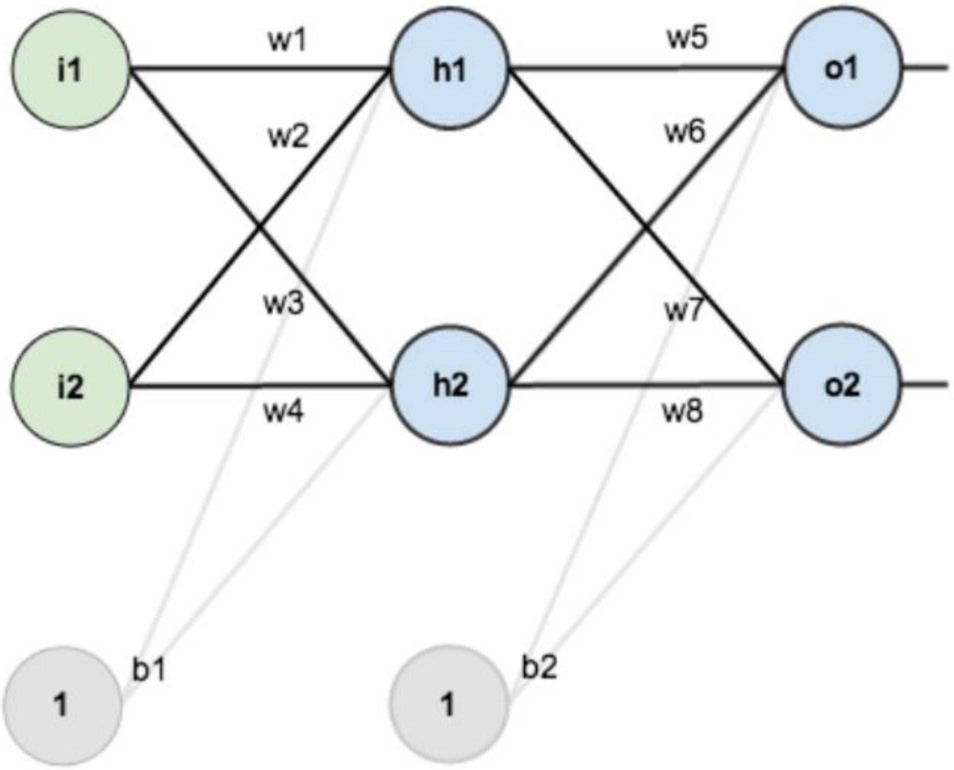
Gradient descent

Likelihood: ascent

Loss: descent

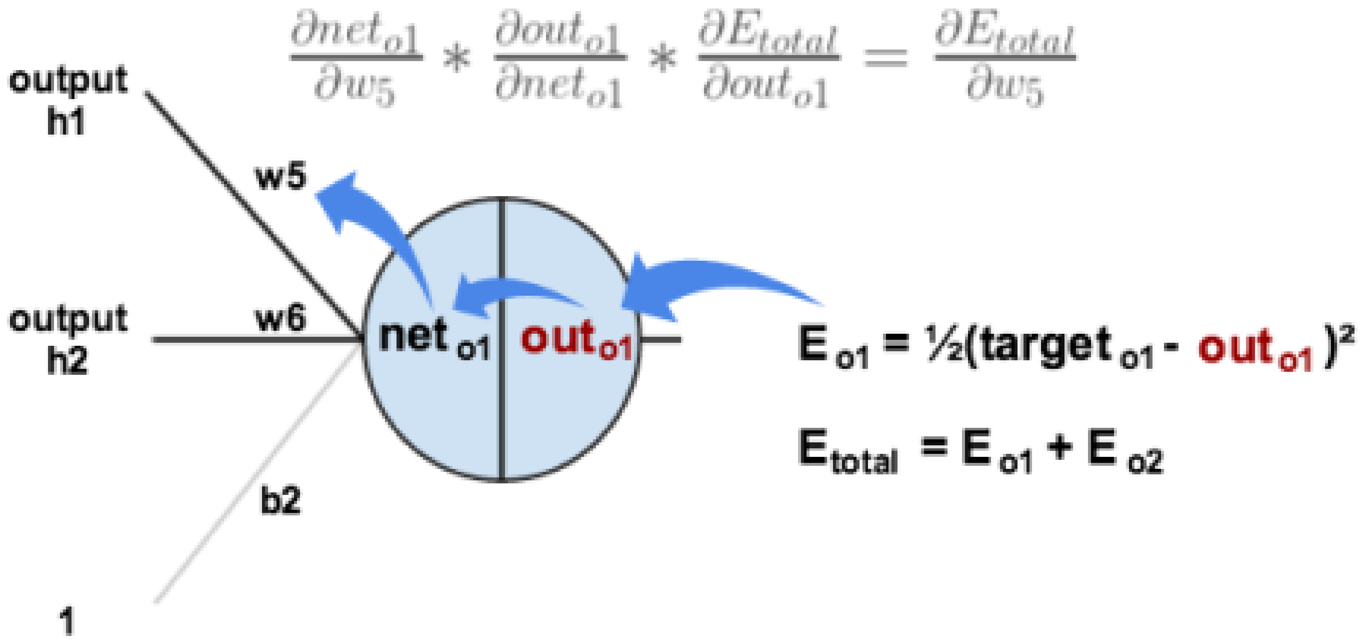


Backpropagation



https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/

Backpropagation and Gradient Descent



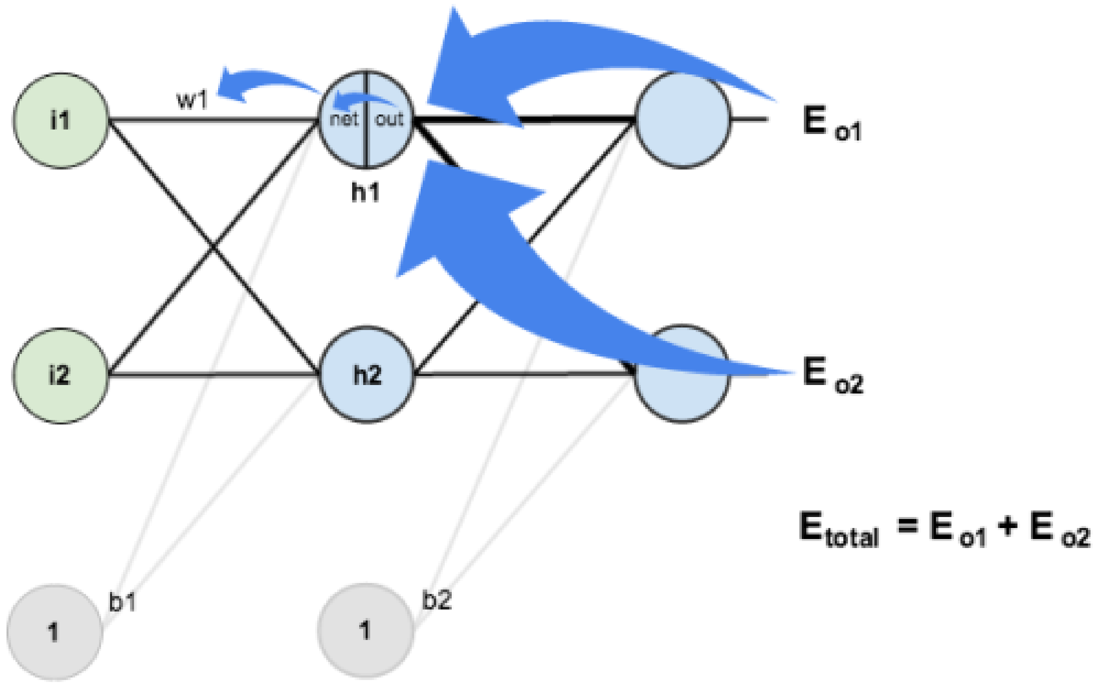
https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/

Backpropagation and Gradient Descent

$$\frac{\partial E_{total}}{\partial w_1} = \frac{\partial E_{total}}{\partial out_{h1}} * \frac{\partial out_{h1}}{\partial net_{h1}} * \frac{\partial net_{h1}}{\partial w_1}$$

$$\downarrow$$

$$\frac{\partial E_{total}}{\partial out_{h1}} = \frac{\partial E_{o1}}{\partial out_{h1}} + \frac{\partial E_{o2}}{\partial out_{h1}}$$



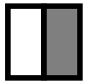
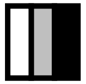
https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/

Topics

1. Introduction to Neural Network
2. Backpropagation and Gradient Descent
3. Edge Detection
4. Convolution as a layer
5. Convolutional Network

Vertical edge detection example

$$
\begin{bmatrix} 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \end{bmatrix} * \begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix} = \begin{bmatrix} 0 & 30 & 30 & 0 \\ 0 & 30 & 30 & 0 \\ 0 & 30 & 30 & 0 \\ 0 & 30 & 30 & 0 \end{bmatrix}
$$


$$
\begin{bmatrix} 0 & 0 & 0 & 10 & 10 & 10 \\ 0 & 0 & 0 & 10 & 10 & 10 \\ 0 & 0 & 0 & 10 & 10 & 10 \\ 0 & 0 & 0 & 10 & 10 & 10 \\ 0 & 0 & 0 & 10 & 10 & 10 \\ 0 & 0 & 0 & 10 & 10 & 10 \end{bmatrix} * \begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix} = \begin{bmatrix} 0 & -30 & -30 & 0 \\ 0 & -30 & -30 & 0 \\ 0 & -30 & -30 & 0 \\ 0 & -30 & -30 & 0 \end{bmatrix}
$$



Vertical and Horizontal Edge Detection

1	0	-1
1	0	-1
1	0	-1

Vertical

1	1	1
0	0	0
-1	-1	-1

Horizontal

$$\begin{bmatrix} 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \\ 10 & 10 & 10 & 0 & 0 & 0 \\ 0 & 0 & 0 & 10 & 10 & 10 \\ 0 & 0 & 0 & 10 & 10 & 10 \\ 0 & 0 & 0 & 10 & 10 & 10 \end{bmatrix} * \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 30 & 10 & -10 & -30 \\ 30 & 10 & -10 & -30 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

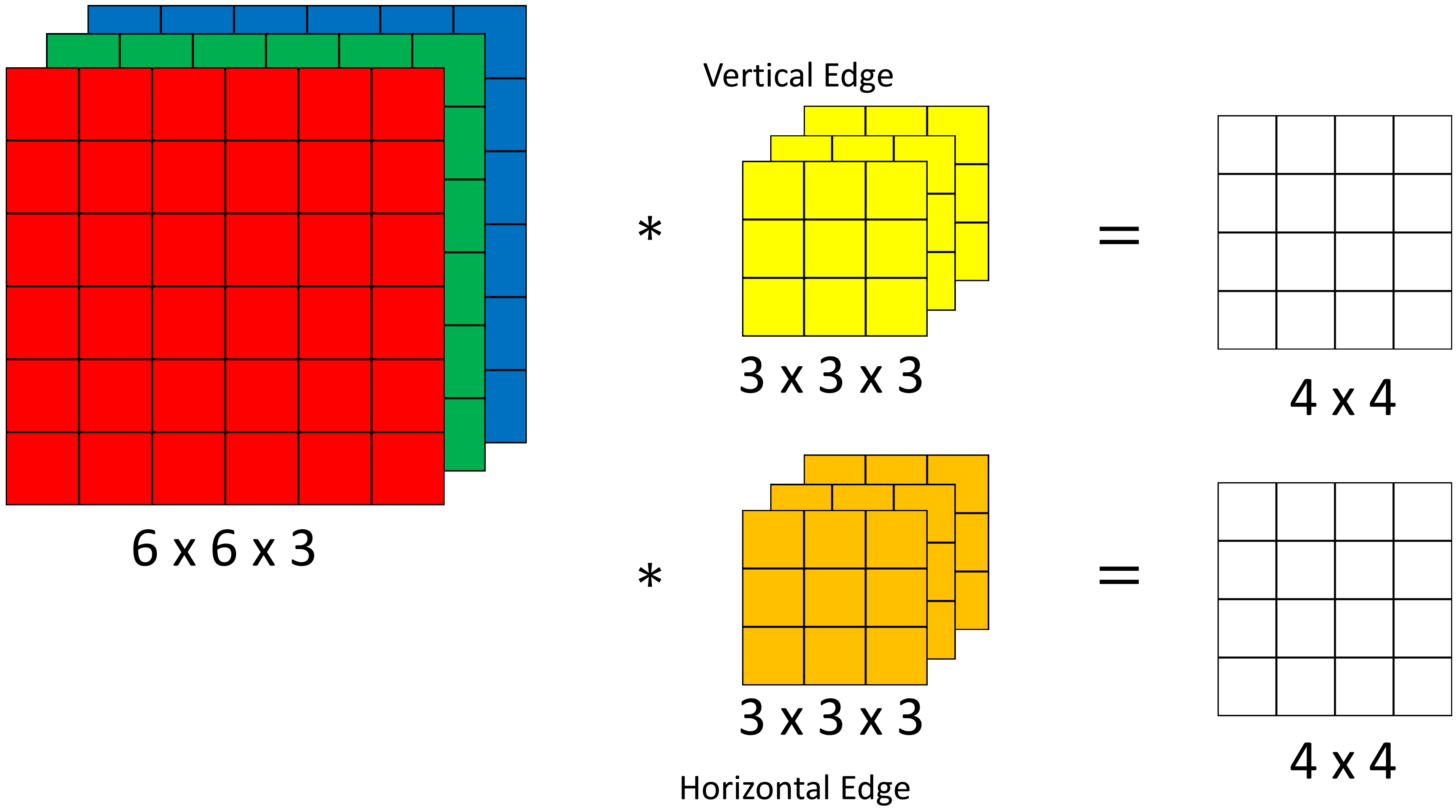
Edge Detection

[Interactive Edge Detection]

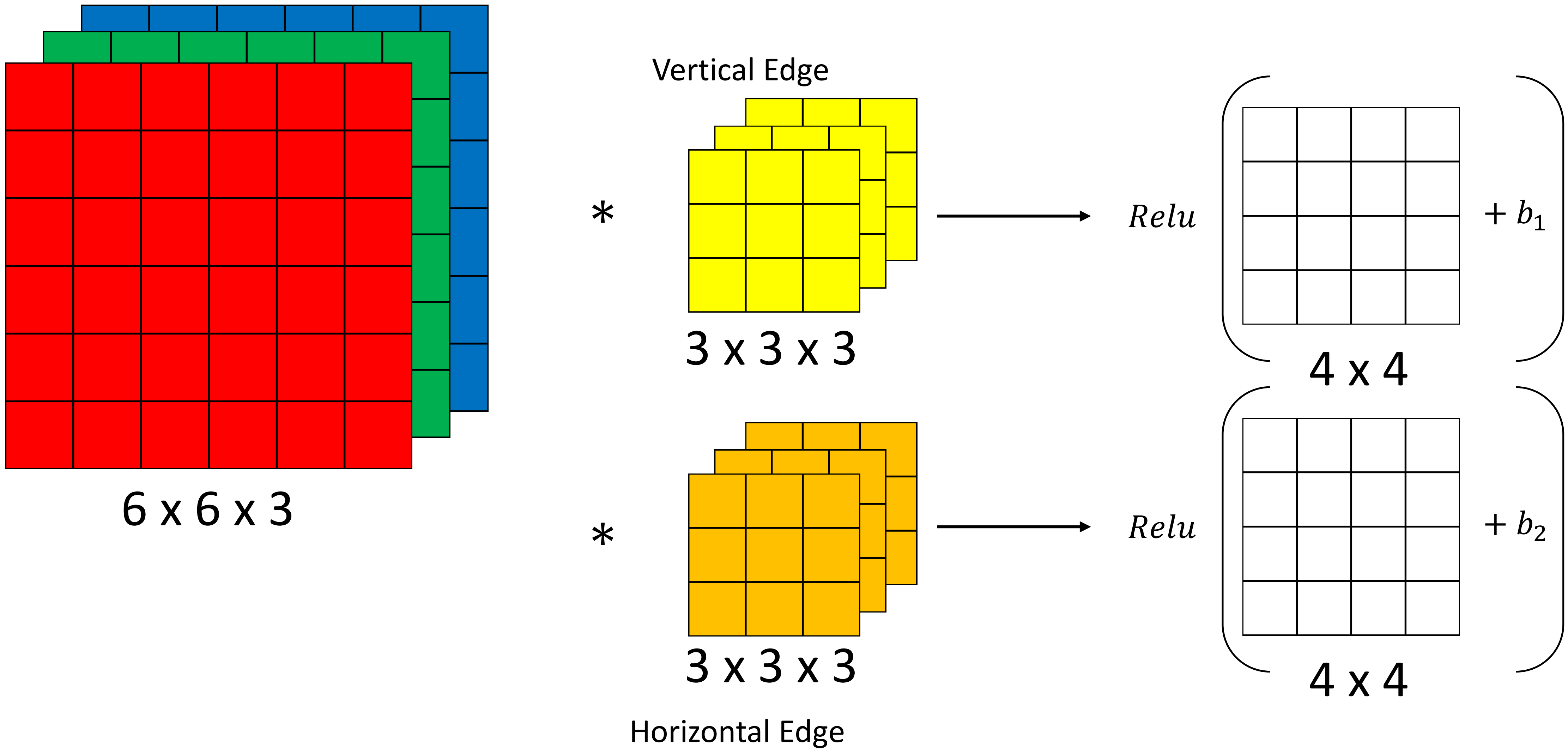
Topics

1. Intro to Neural Network
2. Backpropagation and Gradient Descent
3. Edge Detection
4. Convolution as a layer
5. Convolutional Network

Convolution as a layer



Convolution as a layer



Topics

1. Intro to Neural Network
2. Backpropagation and Gradient Descent
3. Edge Detection
4. Convolution as a layer
5. Convolutional Network

Convolutional Layer: Padding

- **Same:** Pad so that output size is the same as the input size.

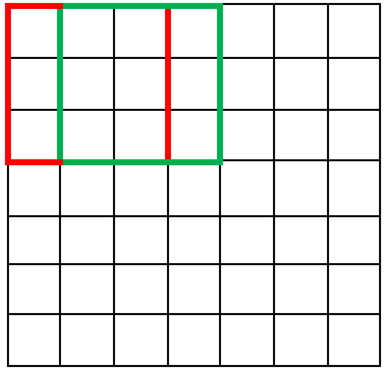
- **Valid:** for $n * n$ image and $k * k$ kernel,

  the output is $(n - k + 1) \; * \; (n - k + 1)$

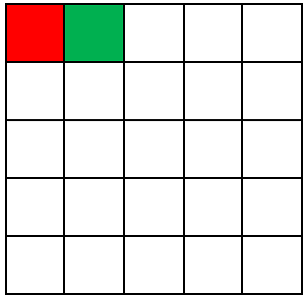
Convolutional Layer: Stride

Stride 1

7 x 7 Input Volume

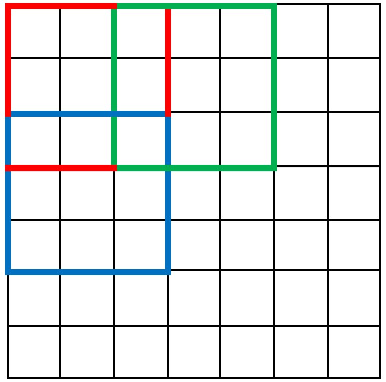


5 x 5 Output Volume

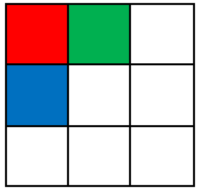


Stride 2

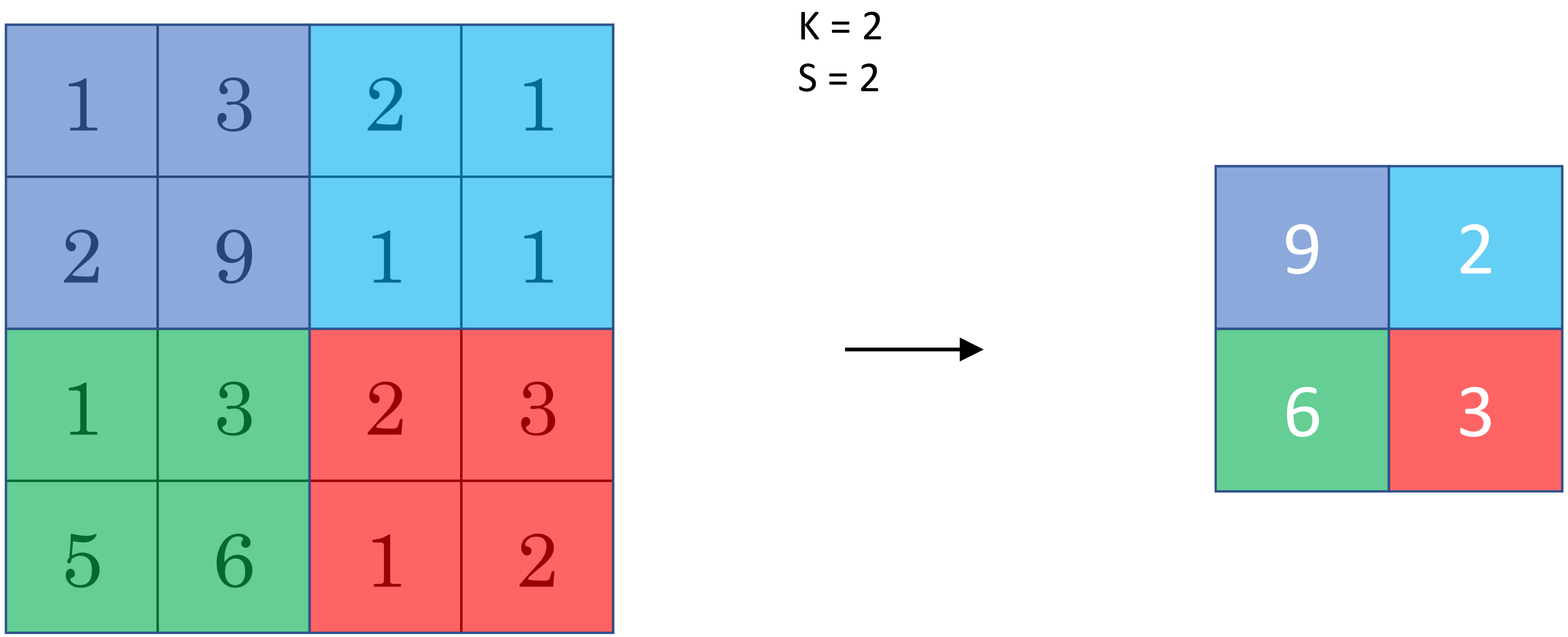
7 x 7 Input Volume



3 x 3 Output Volume



Pooling layer: Max pooling



Pooling layer: Average pooling

1	3	2	1
2	9	1	1
1	4	2	3
5	6	1	2

K = 2
S = 2

→

3.75	1.25
4	2