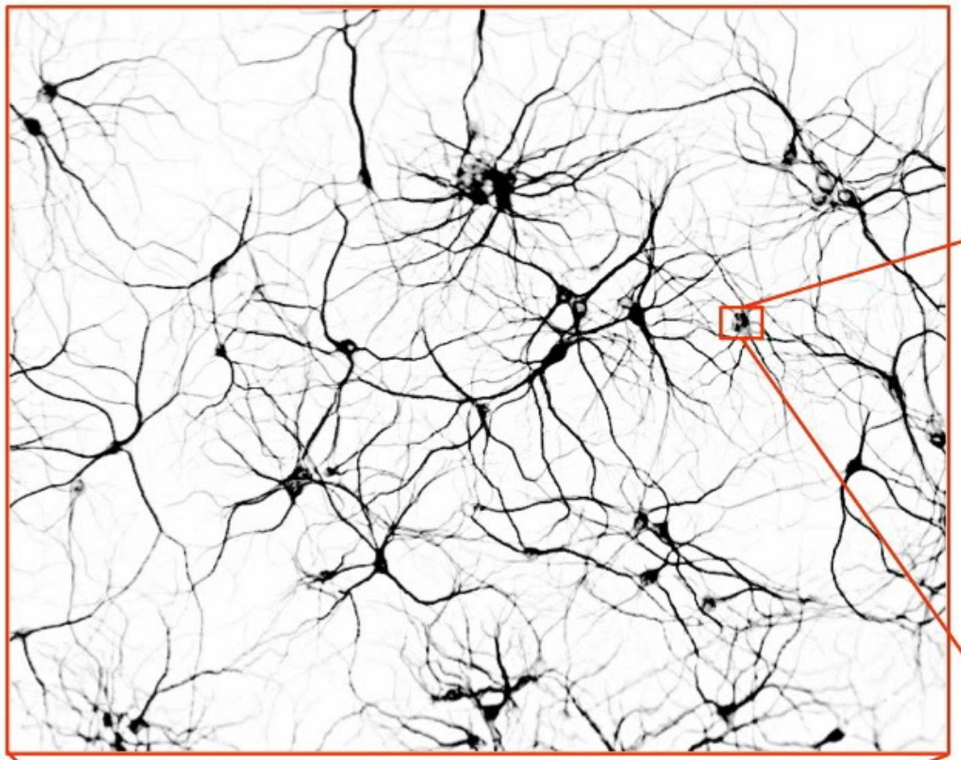
Neural Networks & Deep Learning

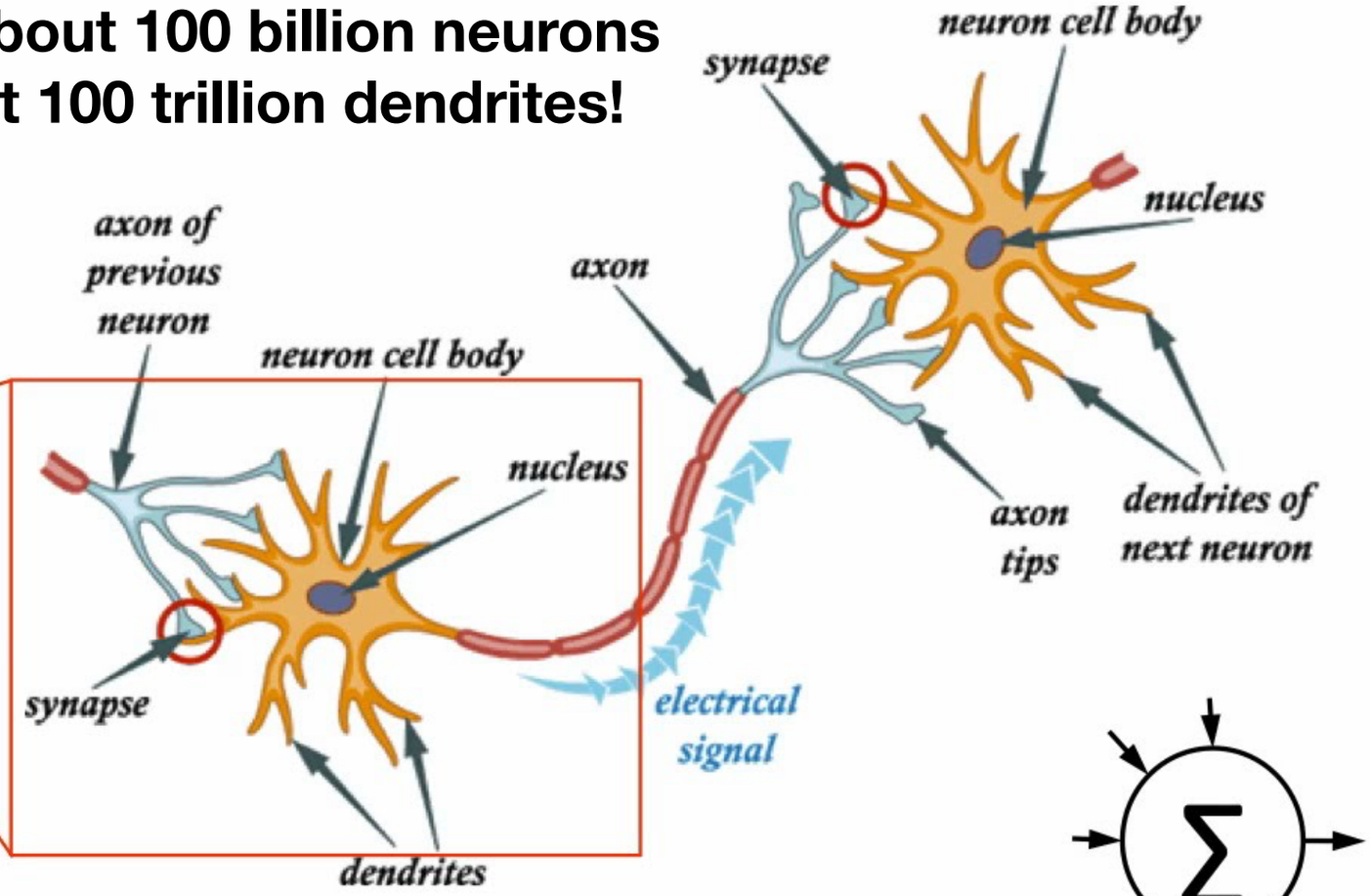
Why they are hot:

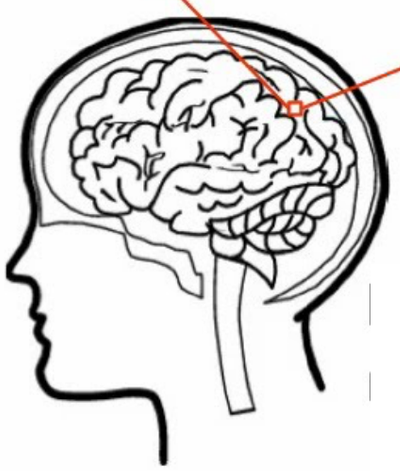
Catalyst (Oct 17): Autonomous cars, robots, etc.
iPhone X: ‘Bionic’ chip with Neural engine.
MAC OS High Sierra: “Your GPU … adds capabilities like machine learning, etc.”
… and doubtless many others.

Neurons and the brain

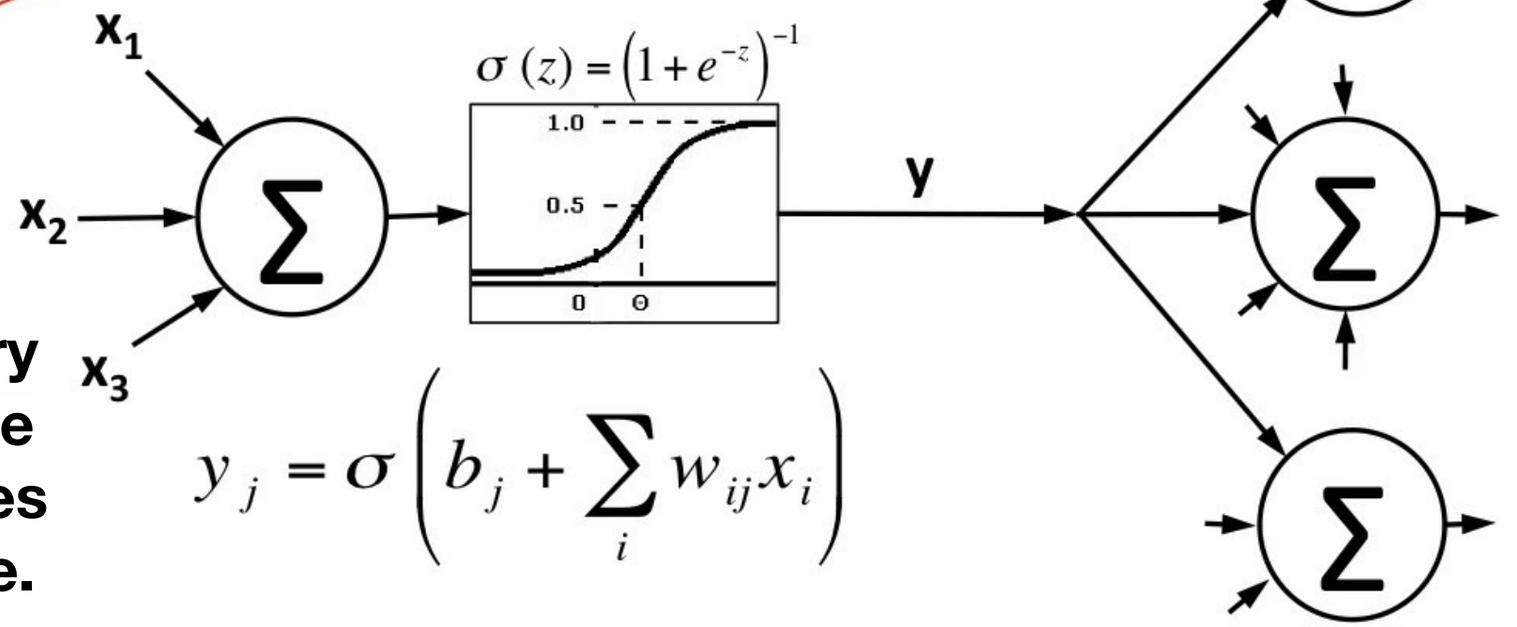
We have about 100 billion neurons and about 100 trillion dendrites!







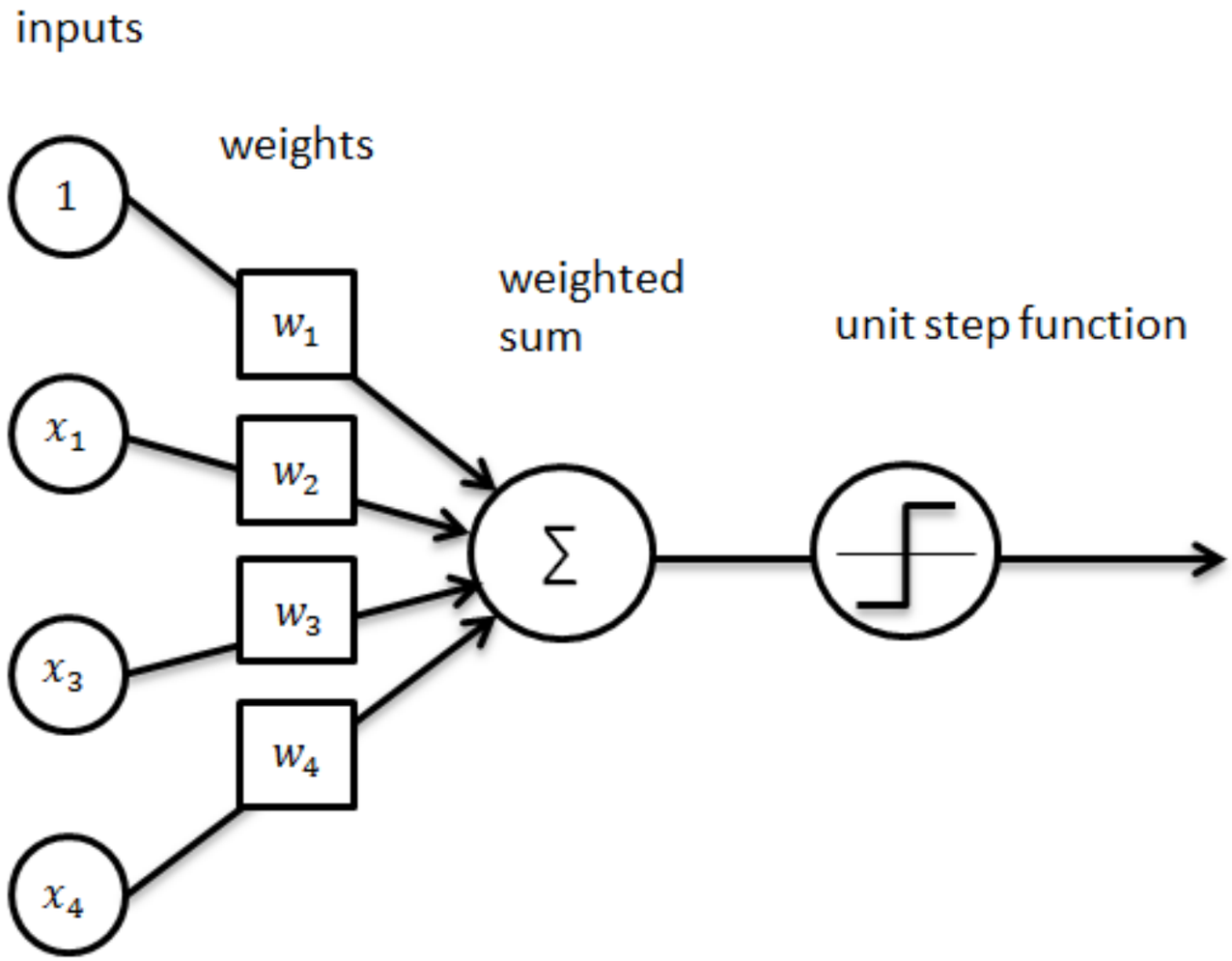
Real neurons vary their output pulse rate; artificial ones vary output value.



$$y_j = \sigma\left(b_j + \sum_i w_{ij} x_i\right)$$

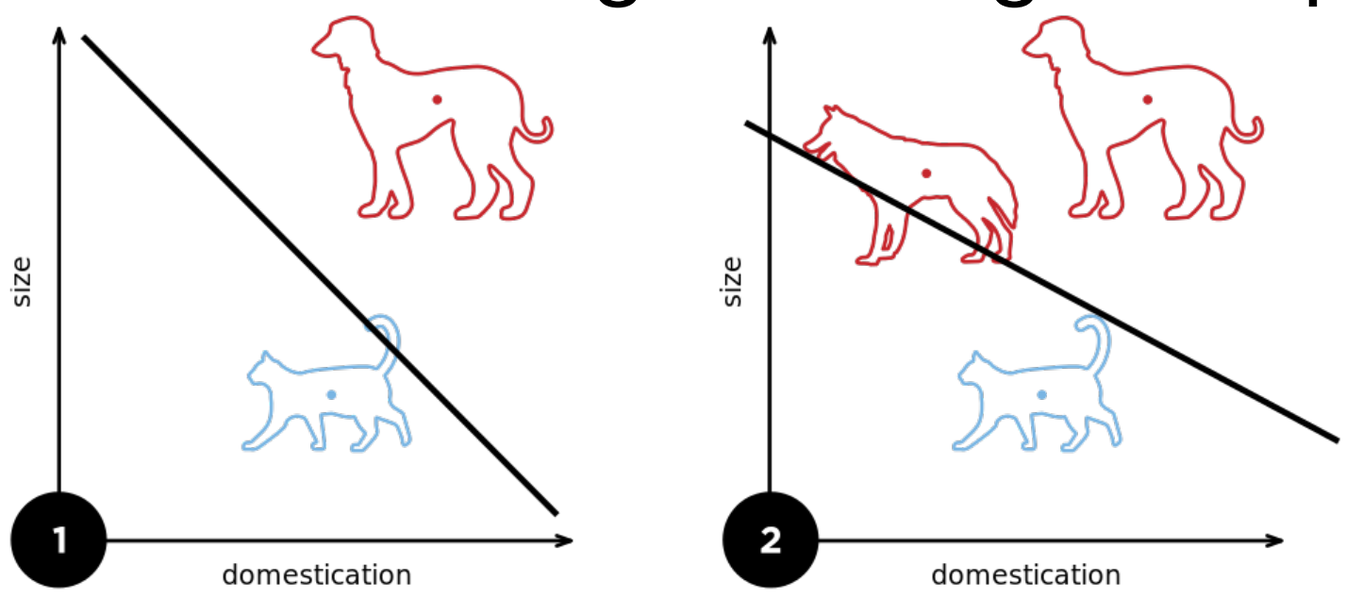
The Perceptron

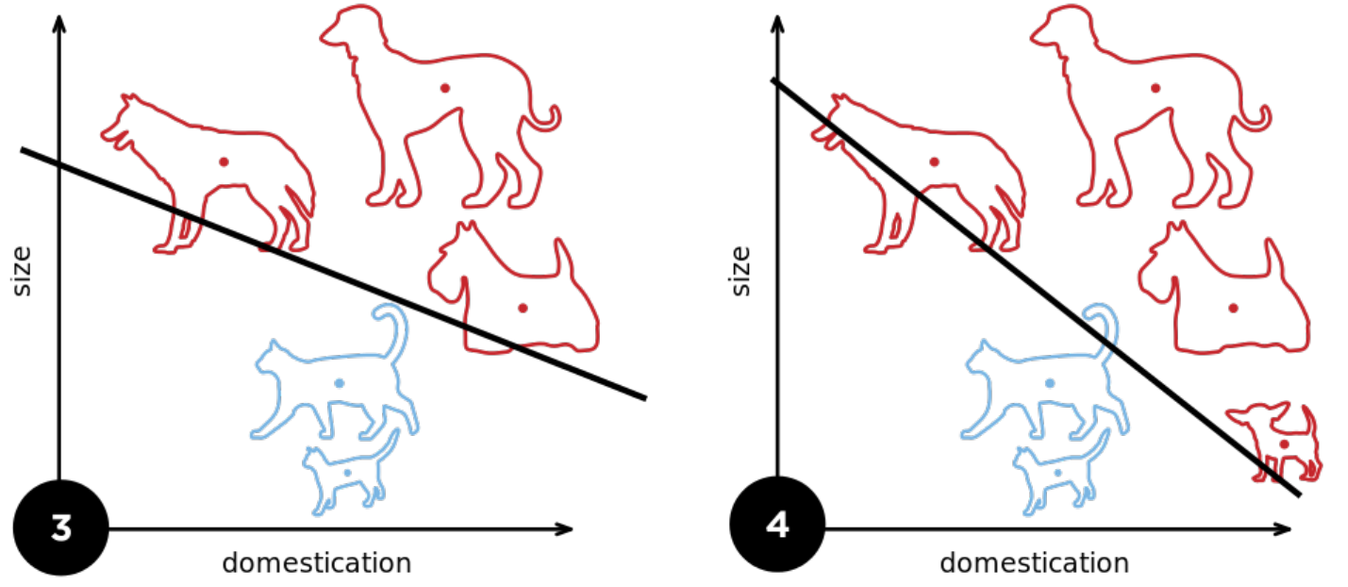
A perceptron can combine many inputs to make a binary decision.



We begin with a random set of weights.
If the output is incorrect, we adjust each weight *w* in proportion to the error times the value of its input *x*.
The proportion is called the *learning rate*.
(Here, the output error can only be +1 or -1.)

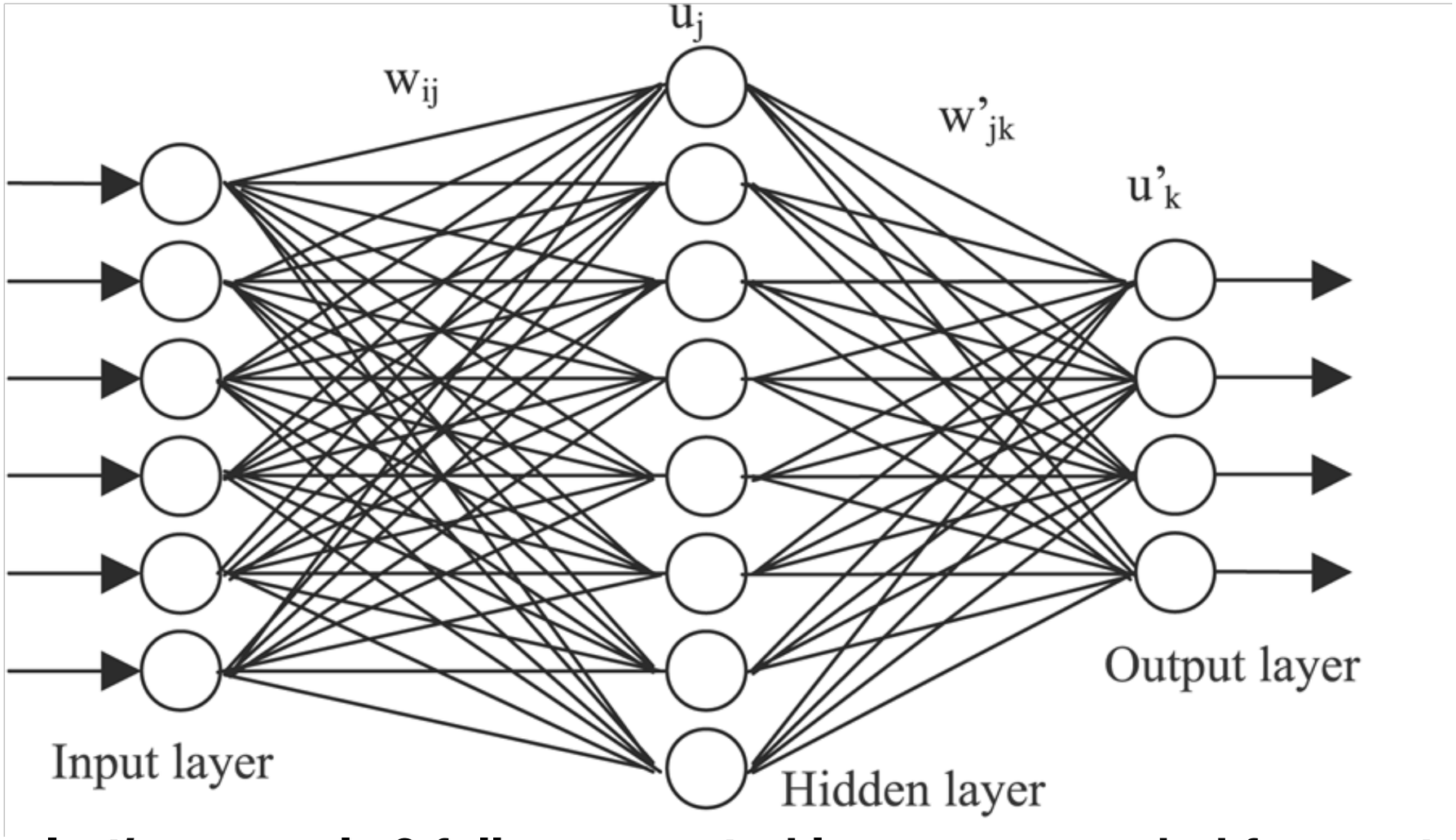
Effect of Presenting Teaching Examples





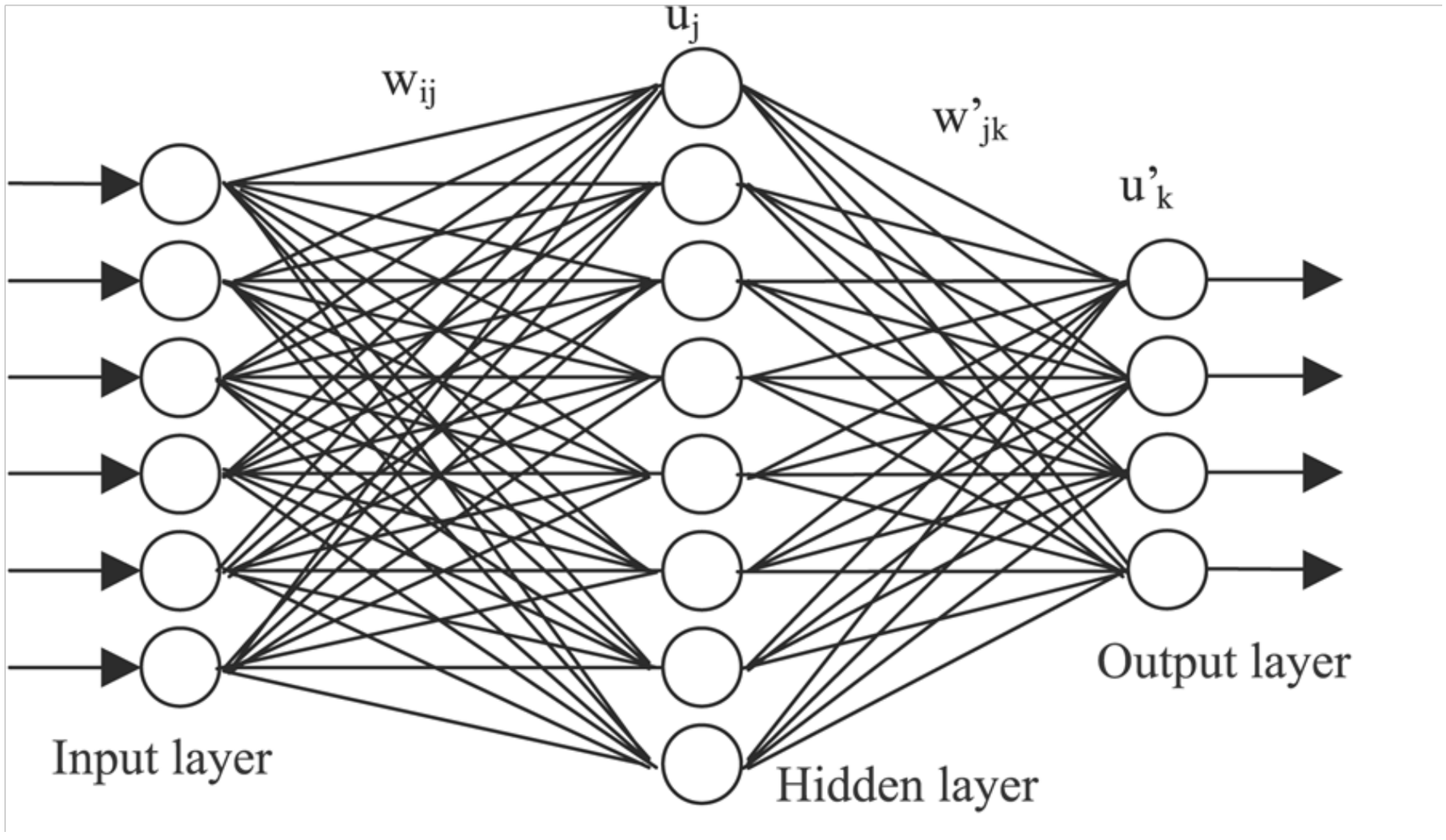
However many examples are presented, a perceptron can only choose *one* best straight boundary to separate them.
Many simple problems need multiple boundaries.
(For example, X≠Y occupies *two* regions either side of the line X=Y.)

Shallow Neural Networks



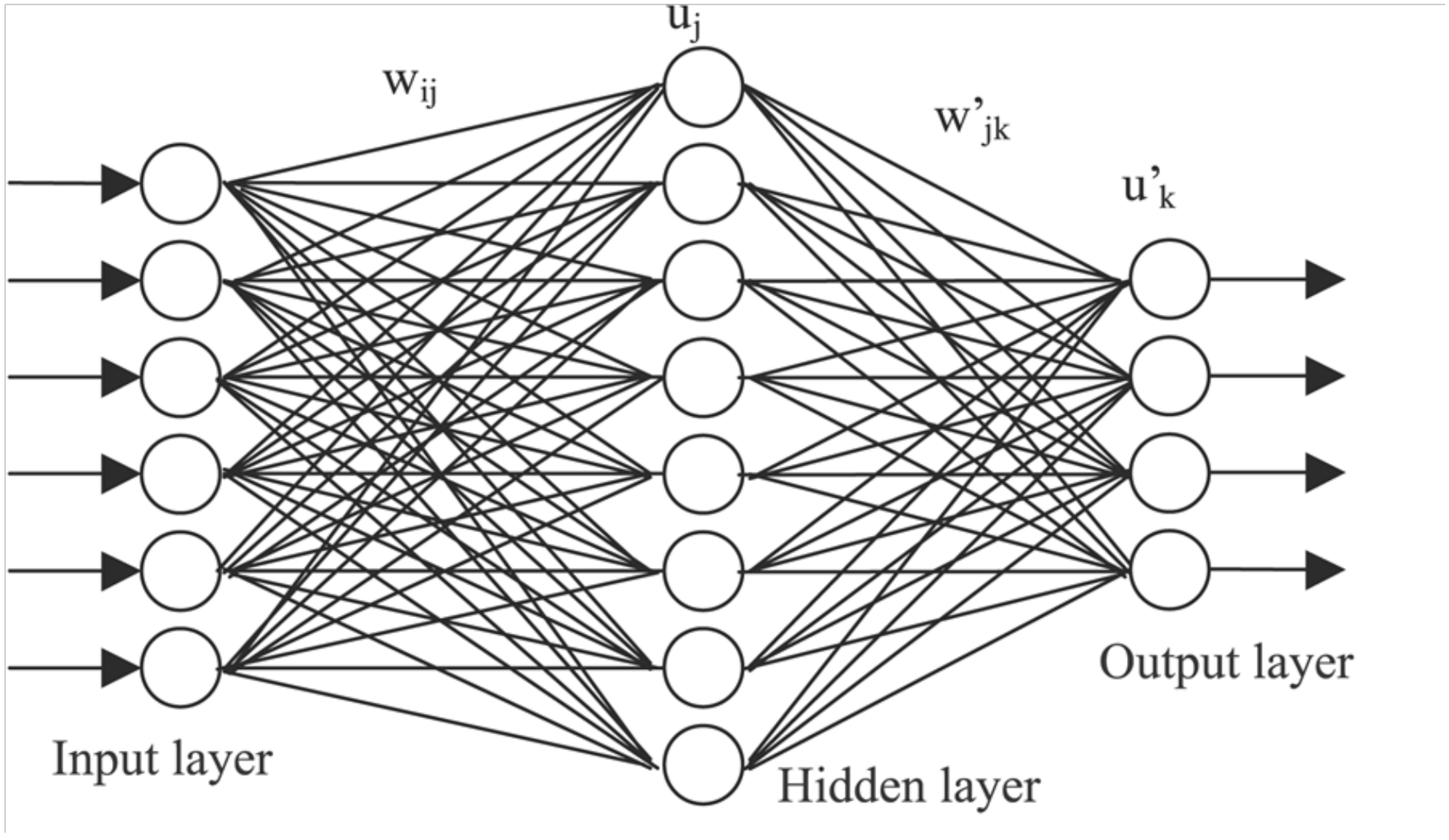
In *theory*, only 3 fully-connected layers are needed for *any* task.
Easy to prove—but how many hidden-layer neurons are needed?
Too few: poor performance; too many: 'over-training'.
How long will it take to train them? What is the best learning rate?

Training a Neural Network



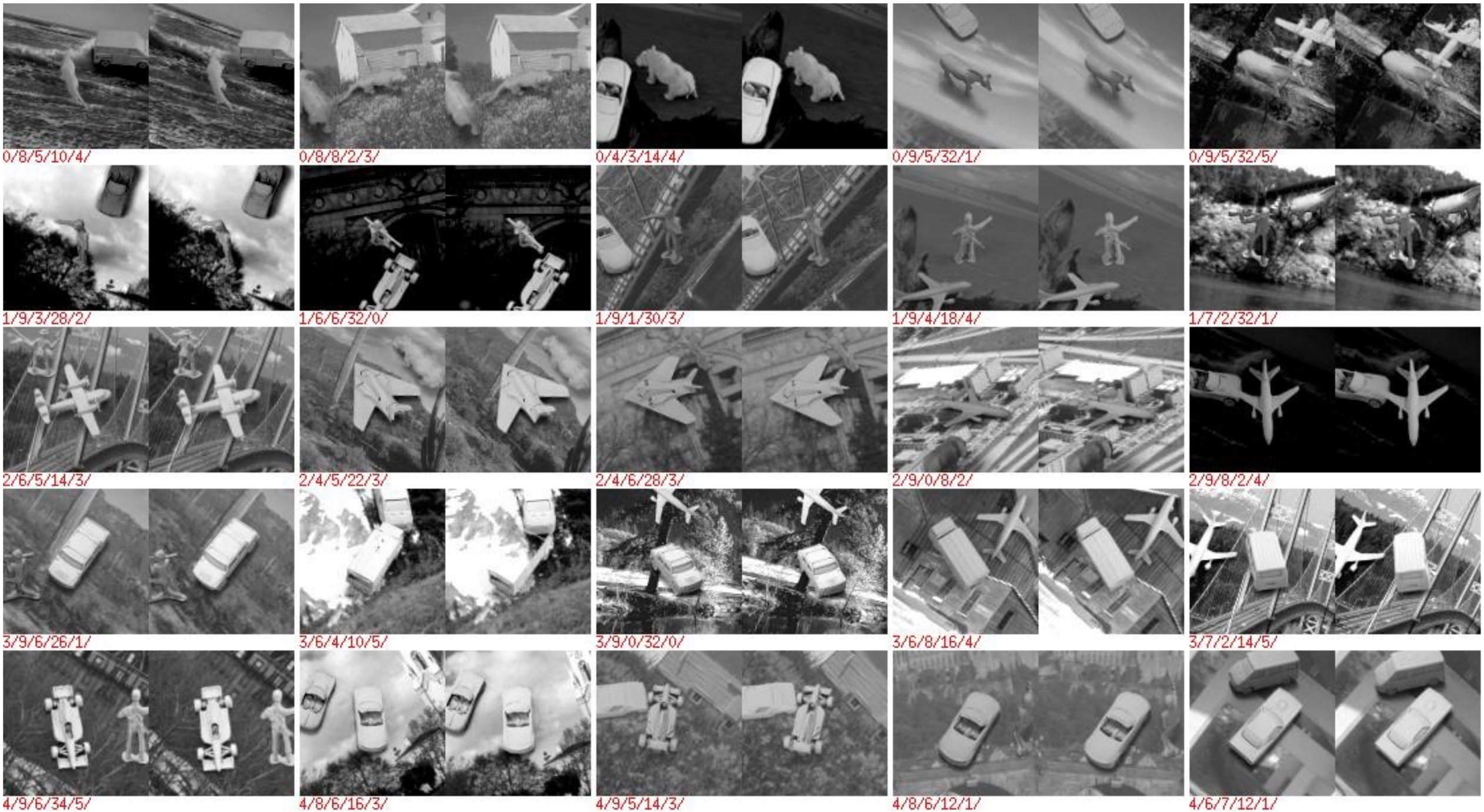
We measure the error in each output (*Actual - Desired*) and adjust the weights in the same way as a perceptron.

Back Propagation



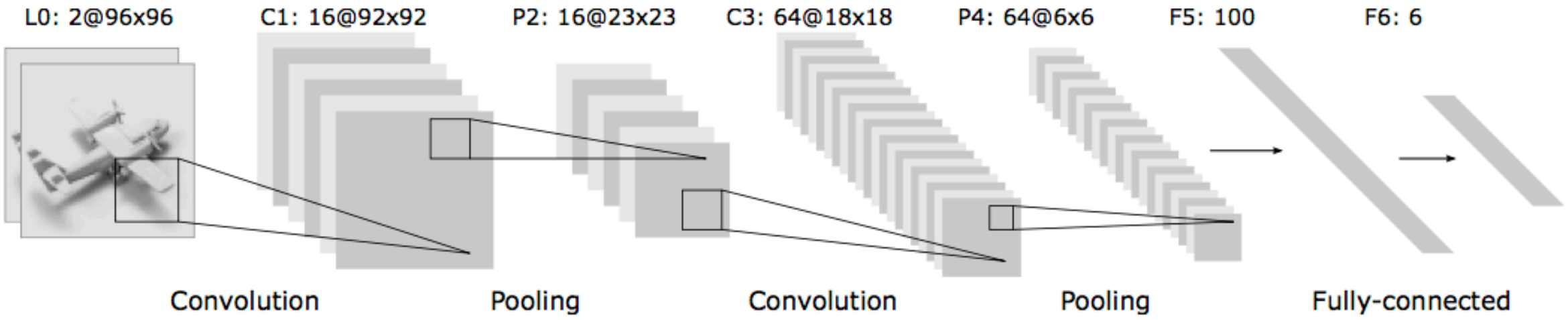
How do we estimate *Desired* for a *hidden* layer neuron?
We *sum* its error contributions to each output neuron: i.e., what output would drive the next layer of neurons in the right direction *on average*? We work *backwards* from the desired output.

The NORB dataset contains normalised 8-bit grey-scale 96x96 pixel stereo pair photos of toys: animals, humans, planes, trucks and cars.



This *Deep Convolutional Network*
was set up specifically for the NORB dataset.

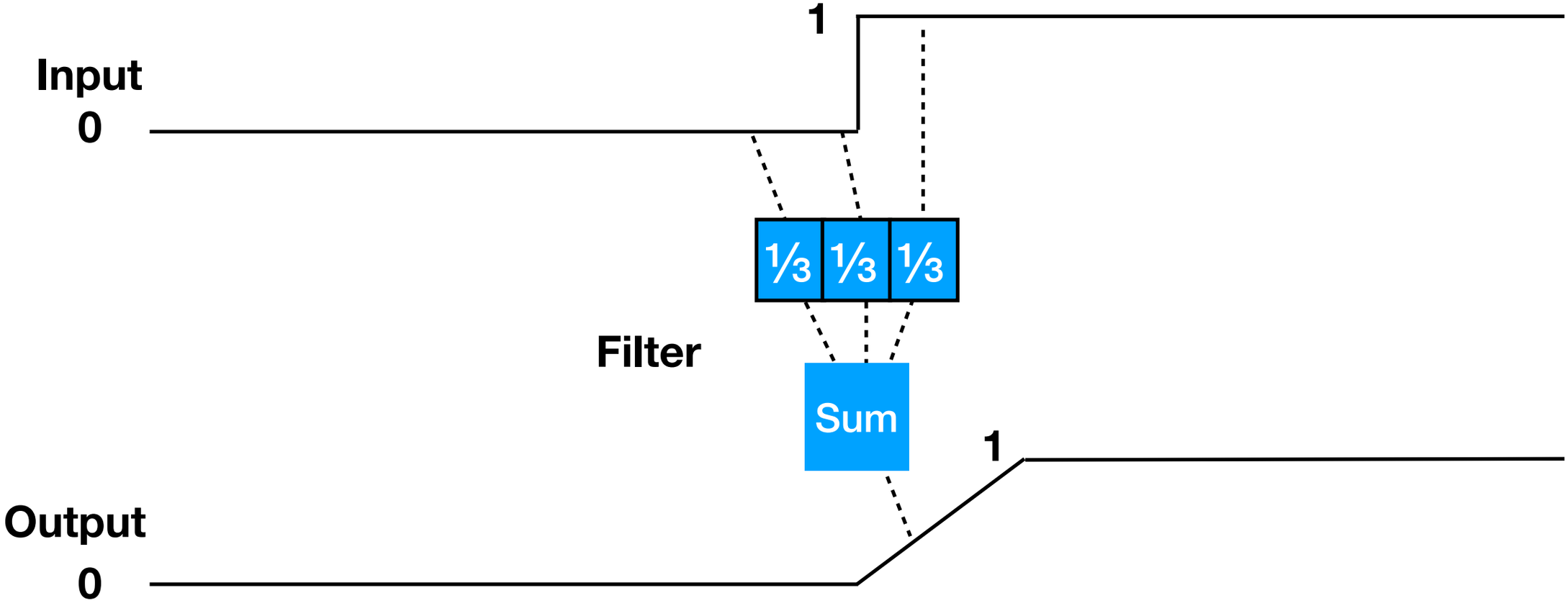
L0 contains the original binocular 96x96 pixel 8-bit grey-scale images.
C1 passes 5x5 patches of the image through 16 different *convolutional* filters.
P2 *pools* 4x4 patches of micro-features, reducing by a factor of 16.
C3 passes 6x6 patches of *features* through 64 different 18x18 *convolutional* filters.
P4 *pools* 3x3 patches of macro-features.
F5 is a fully connected layer of 100 neurons with over 200,000 weights.
F6 is the fully connected output layer: 5 toy categories, or 'No'.



What is 'convolution'?

Digital filtering

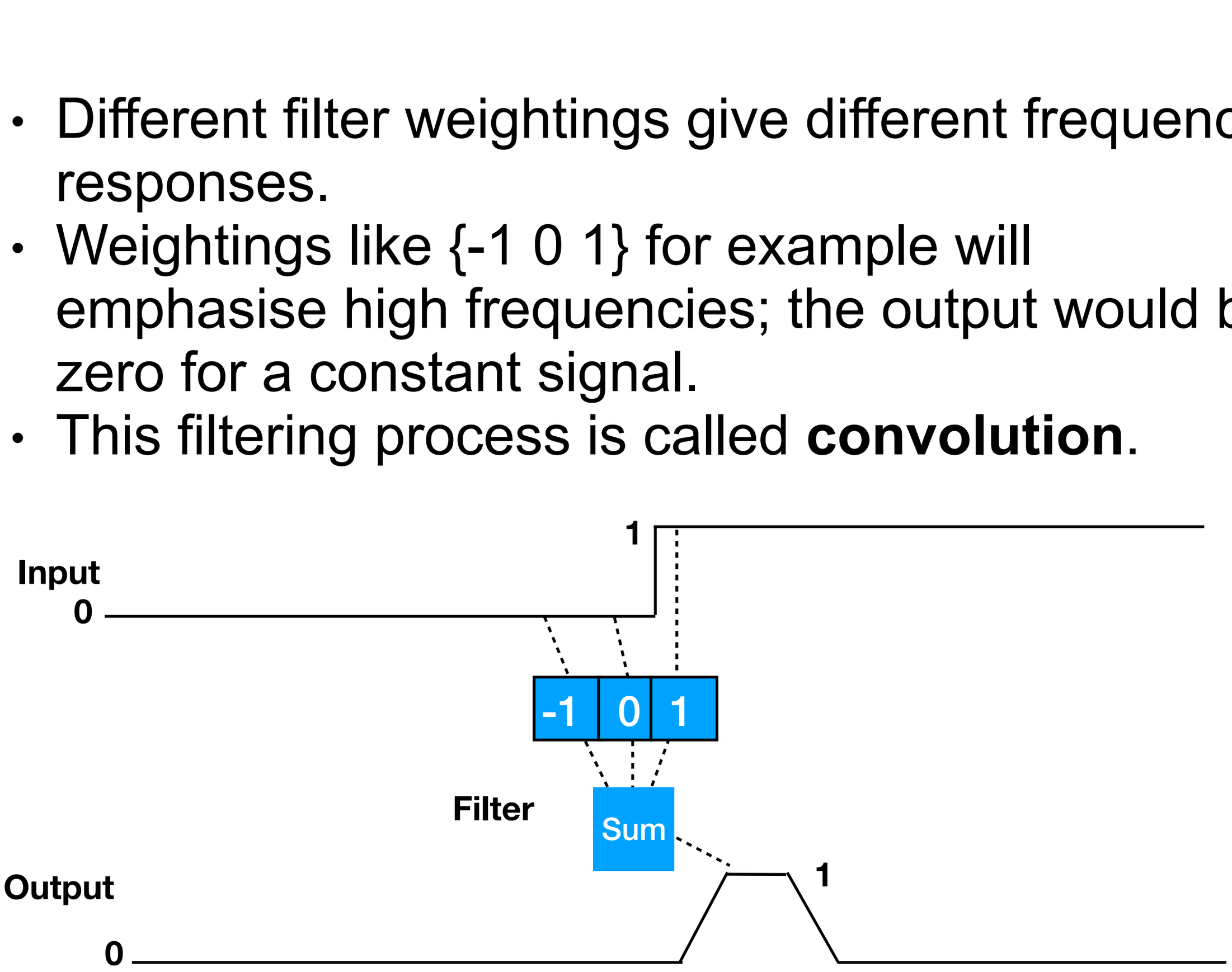
- Think about a digitised audio stream
- A very simple filter is a “running average”
- This tends to remove high frequencies in the input; if the weightings are {1/3 1/3 1/3} it has unity gain for a constant signal (DC).



- Now consider what might happen in 2 dimensions – an image
- The equivalent filter would be
- { 1/9 1/9 1/9
  1/9 1/9 1/9
  1/9 1/9 1/9}

I would “contaminate” each pixel with some data from the adjacent ones. We call this *blur*.

- Different filter weightings give different frequency responses.
- Weightings like {-1 0 1} for example will emphasise high frequencies; the output would be zero for a constant signal.
- This filtering process is called **convolution**.



- A filter like:

- { -1/6 0 1/6
  -1/6 0 1/6
  -1/6 0 1/6}

will give zero output if all pixels are the same, and maximum output as it passes over a dark-to-light step; it is a detector for vertical edges;
Change the signs and it will give maximum output for light-to-dark edges.

- Want horizontal edges?

- {-1/6 -1/6 -1/6
  0 0 0
  1/6 1/6 1/6}

will do it.

And diagonals? How about:

{ 0 1/6 1/6
-1/6 0 1/6
-1/6 -1/6 0 }

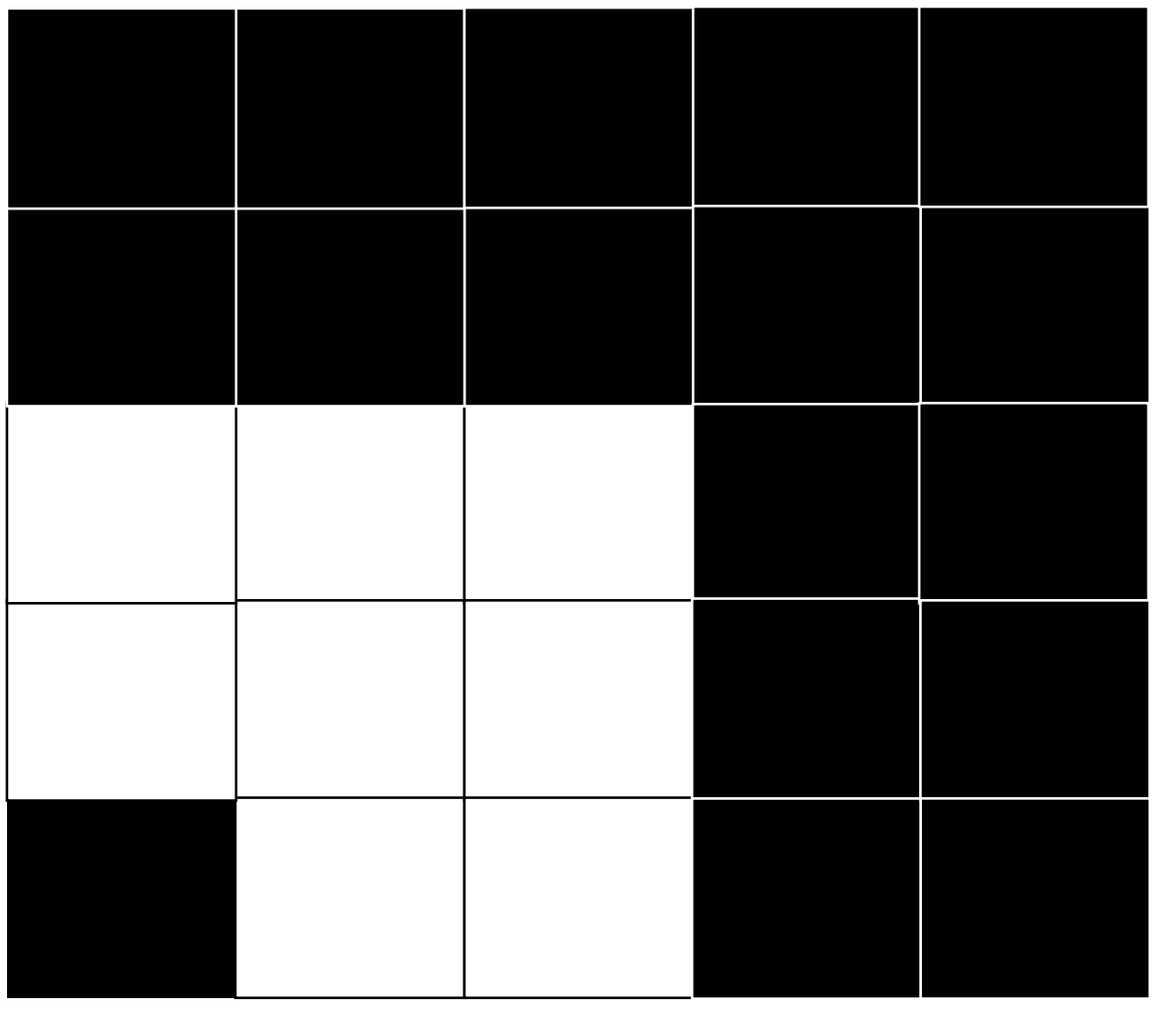
- This shows how simple arithmetic can be used to detect “features” in an image.

- These features can help a neural network decide what the image represents.

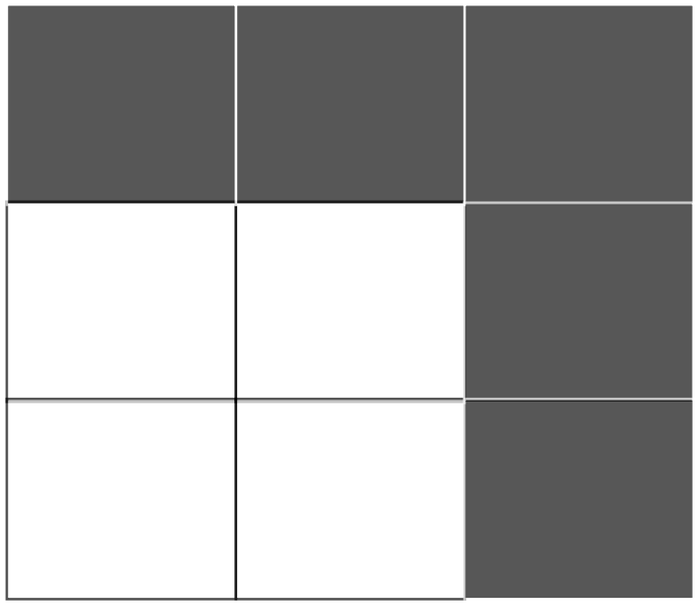
- Of course, more complex features can be extracted, using larger filters …

Convolution

5x5 1-bit *Image*



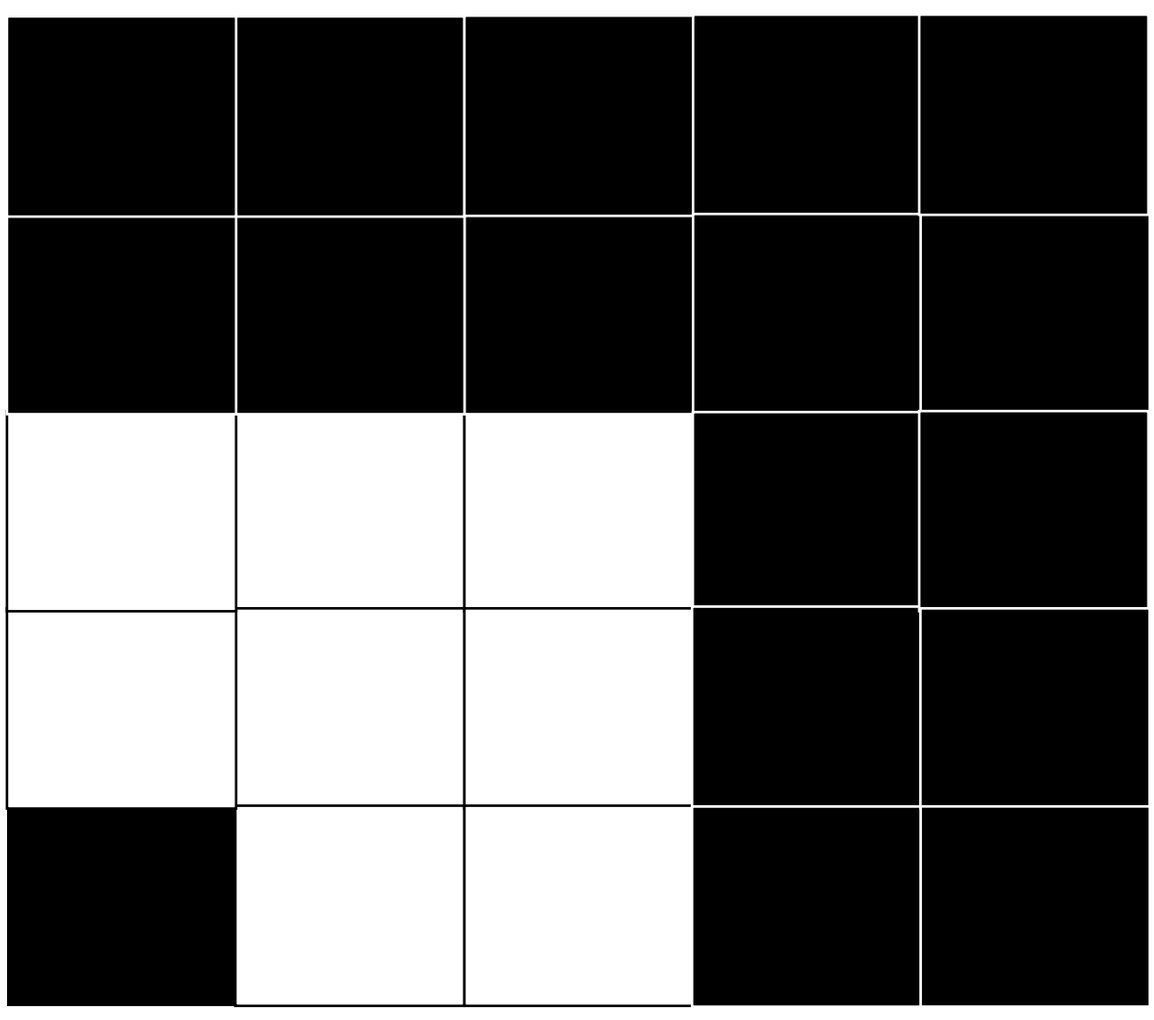
3x3 1-bit *Filter*

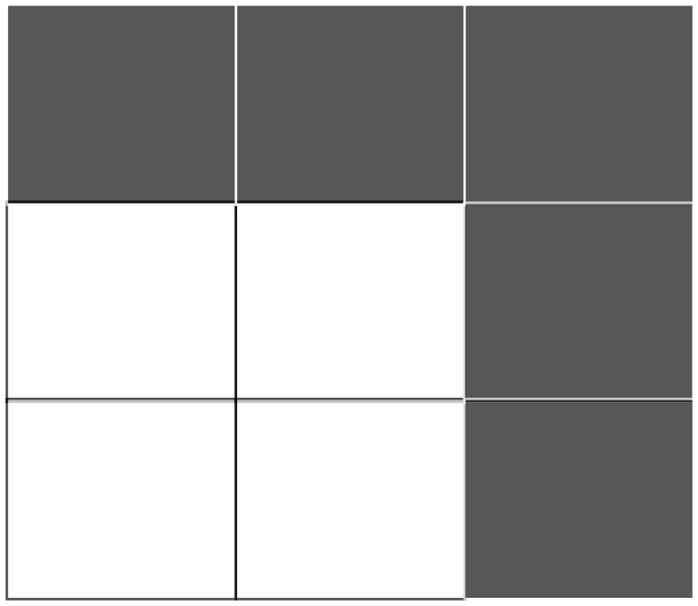


We slide the filter all over the image. At each filter position, we sum *Image* x *Filter* for each pixel. For simplicity, White scores +1, black scores -1.

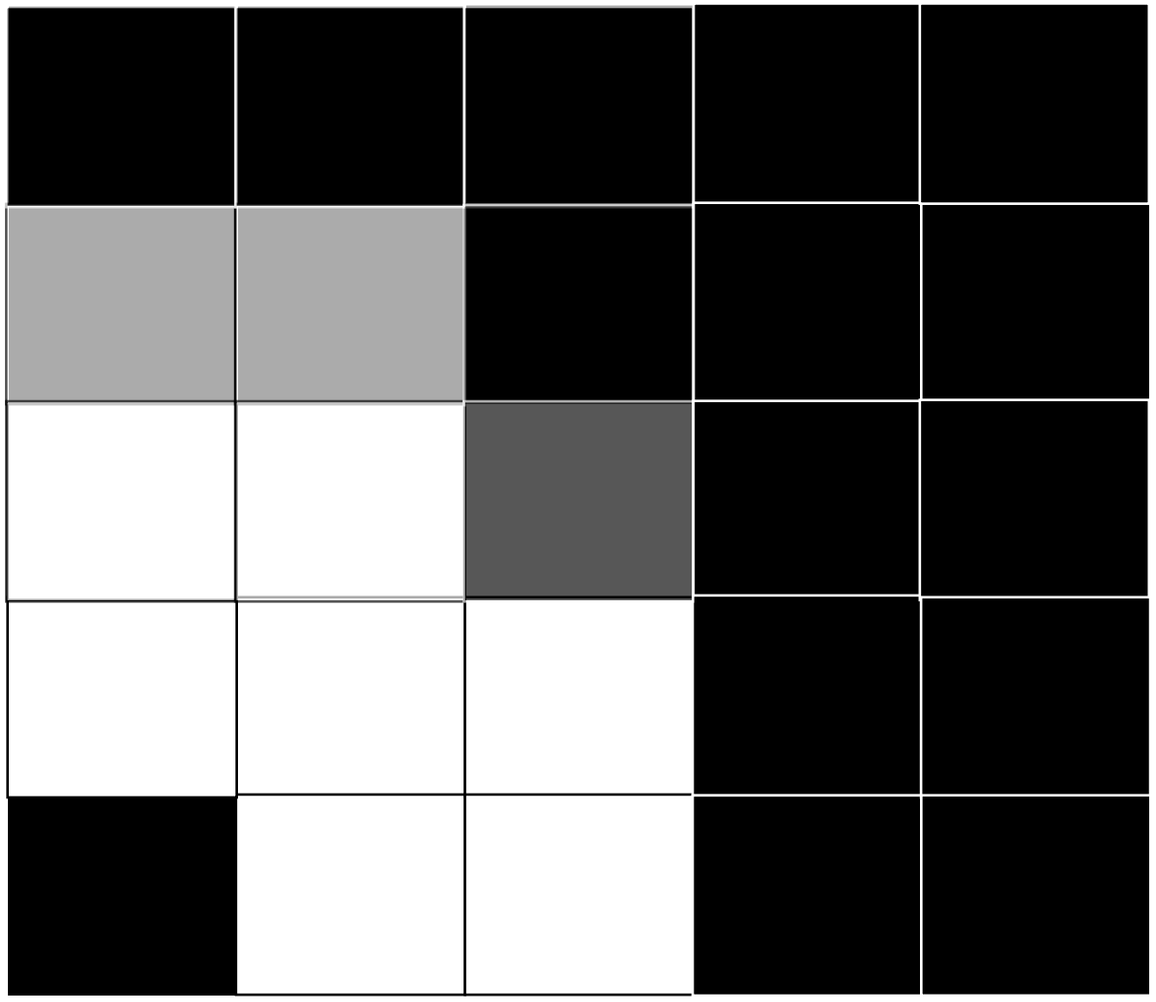
In practice, *several* random filters are used.

Convolution

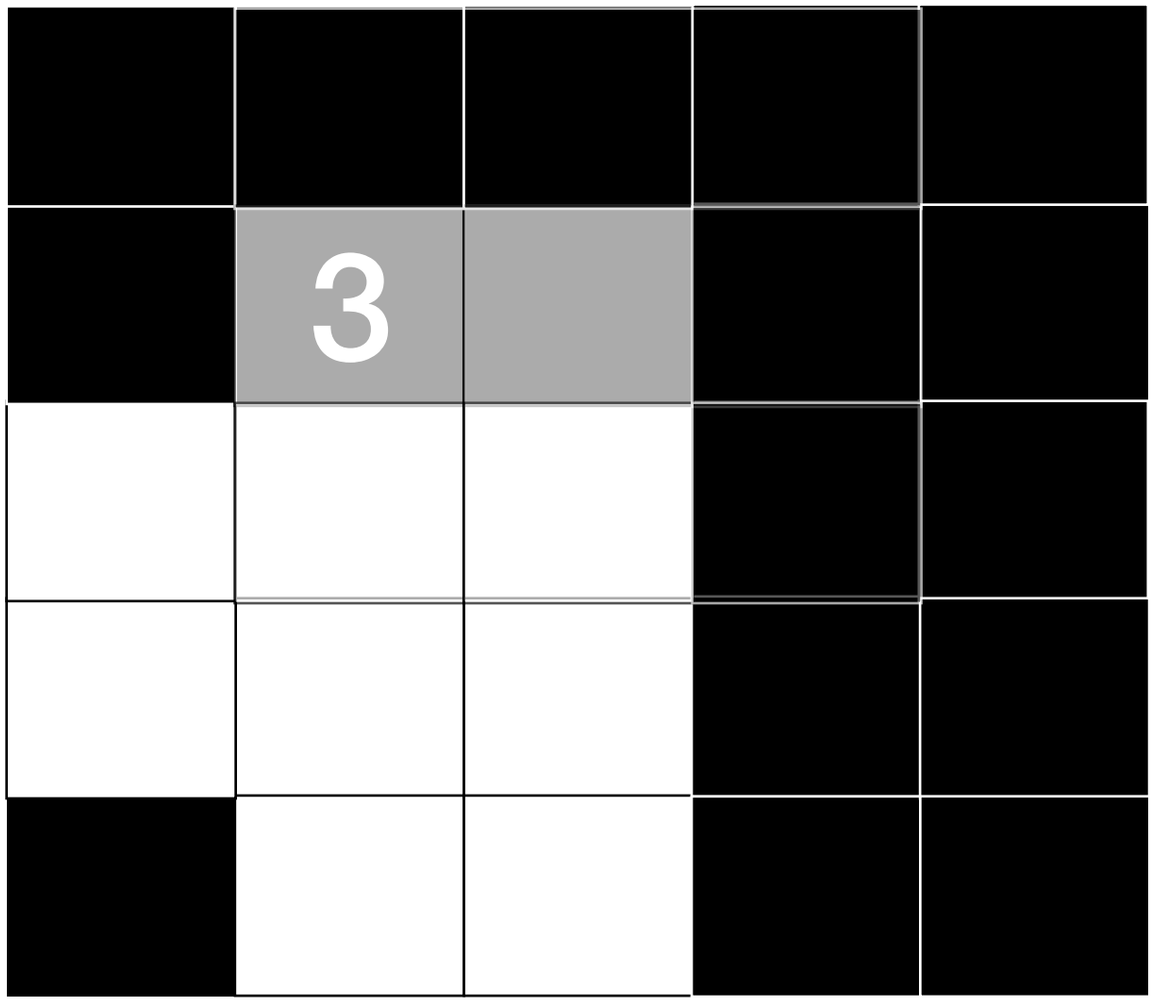




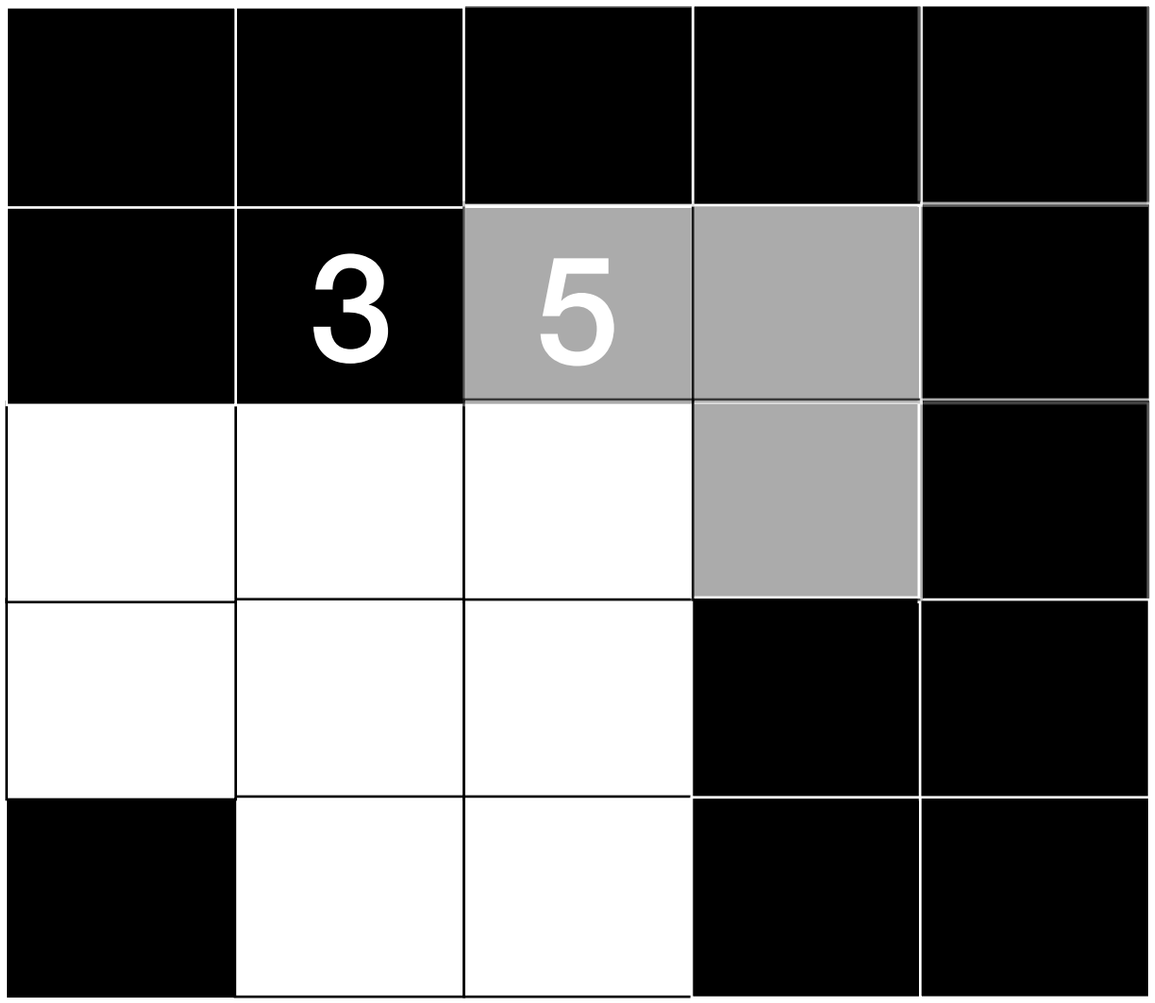
Convolution



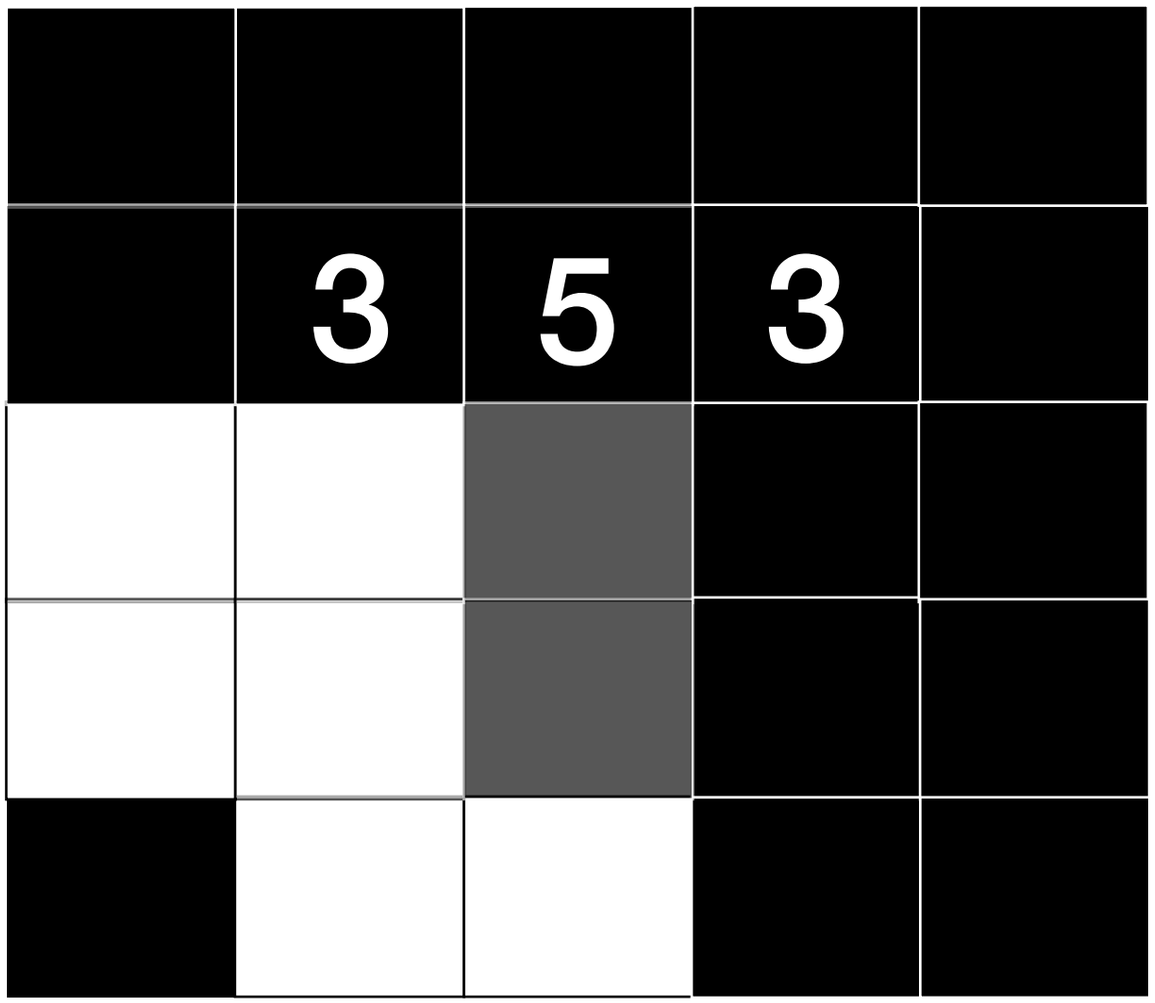
Convolution



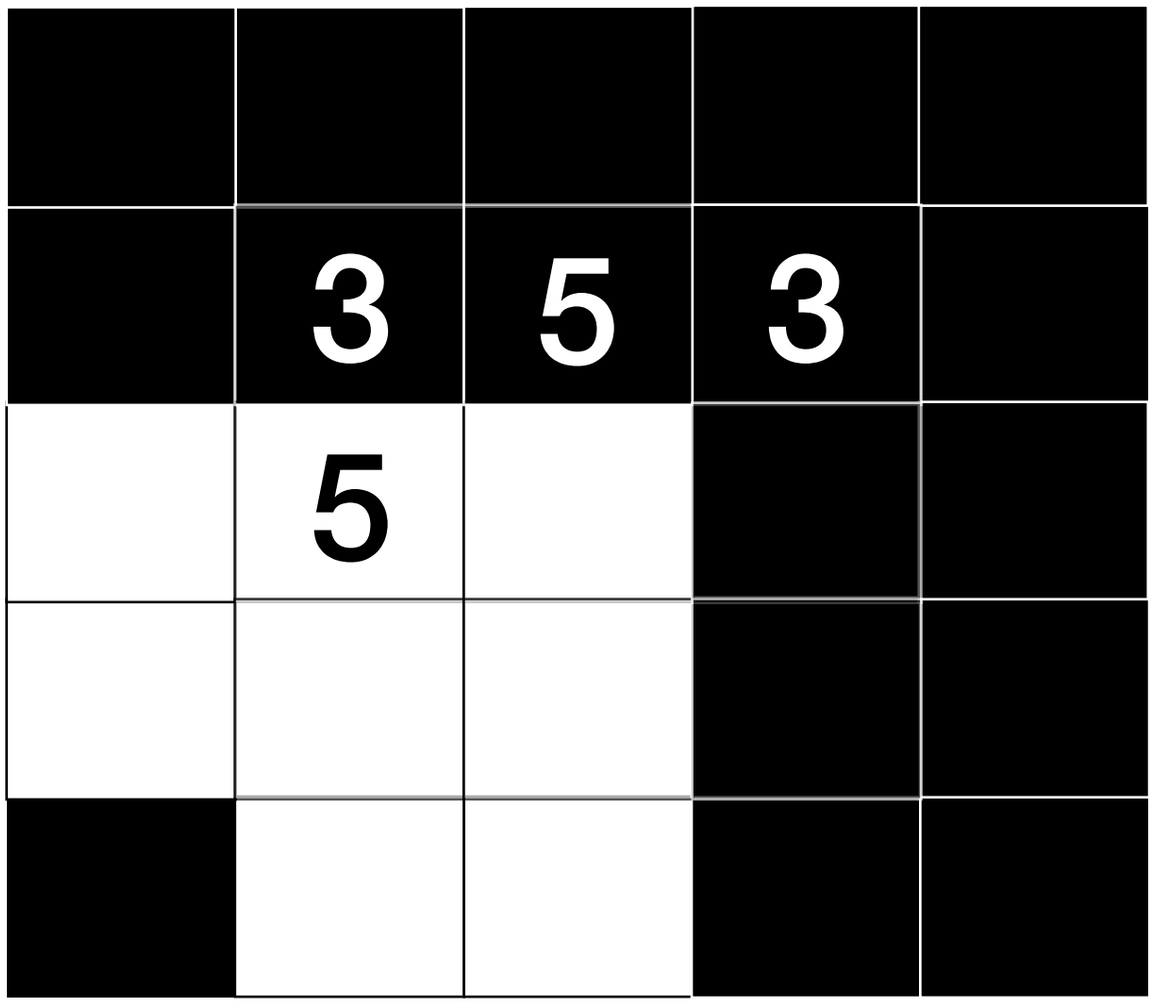
Convolution



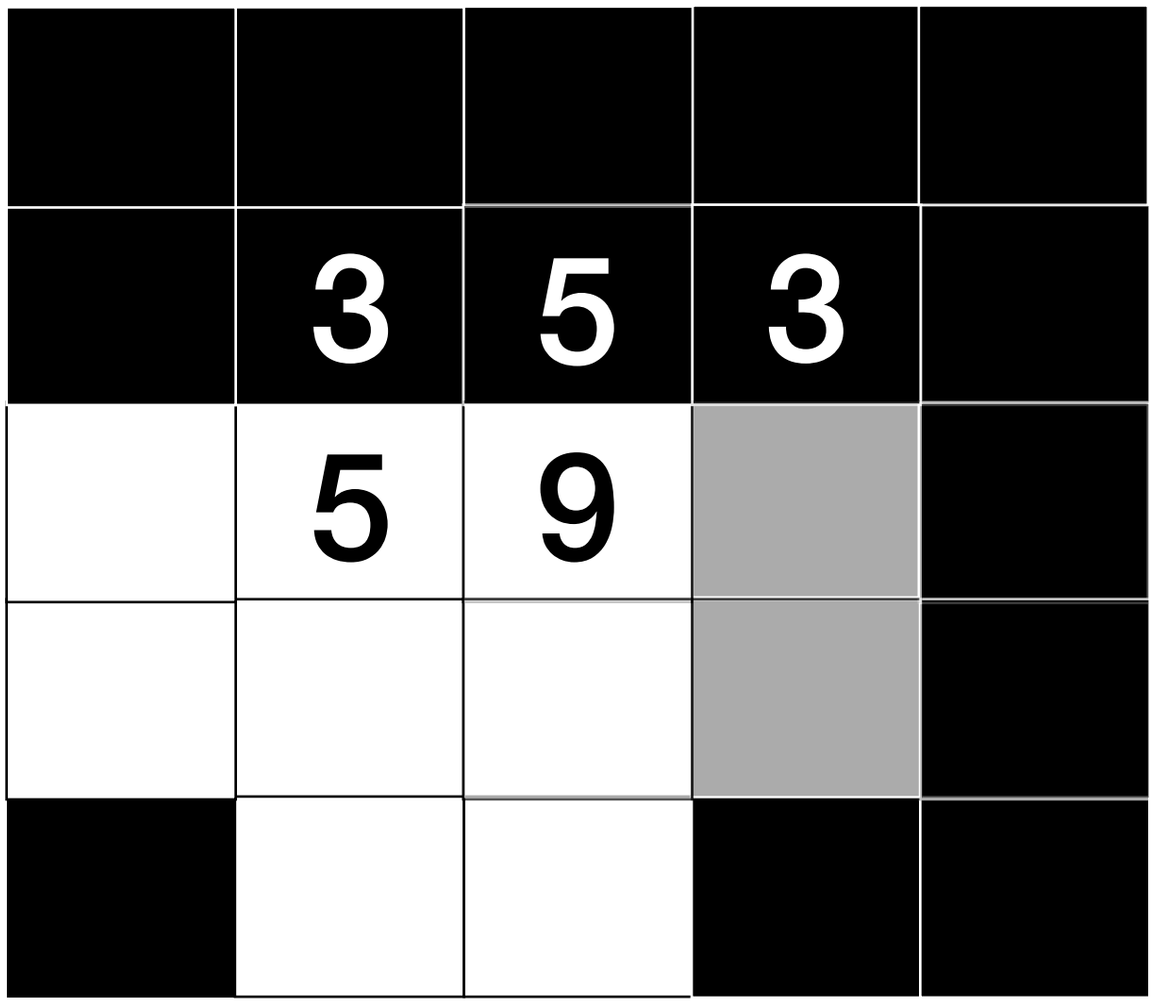
Convolution



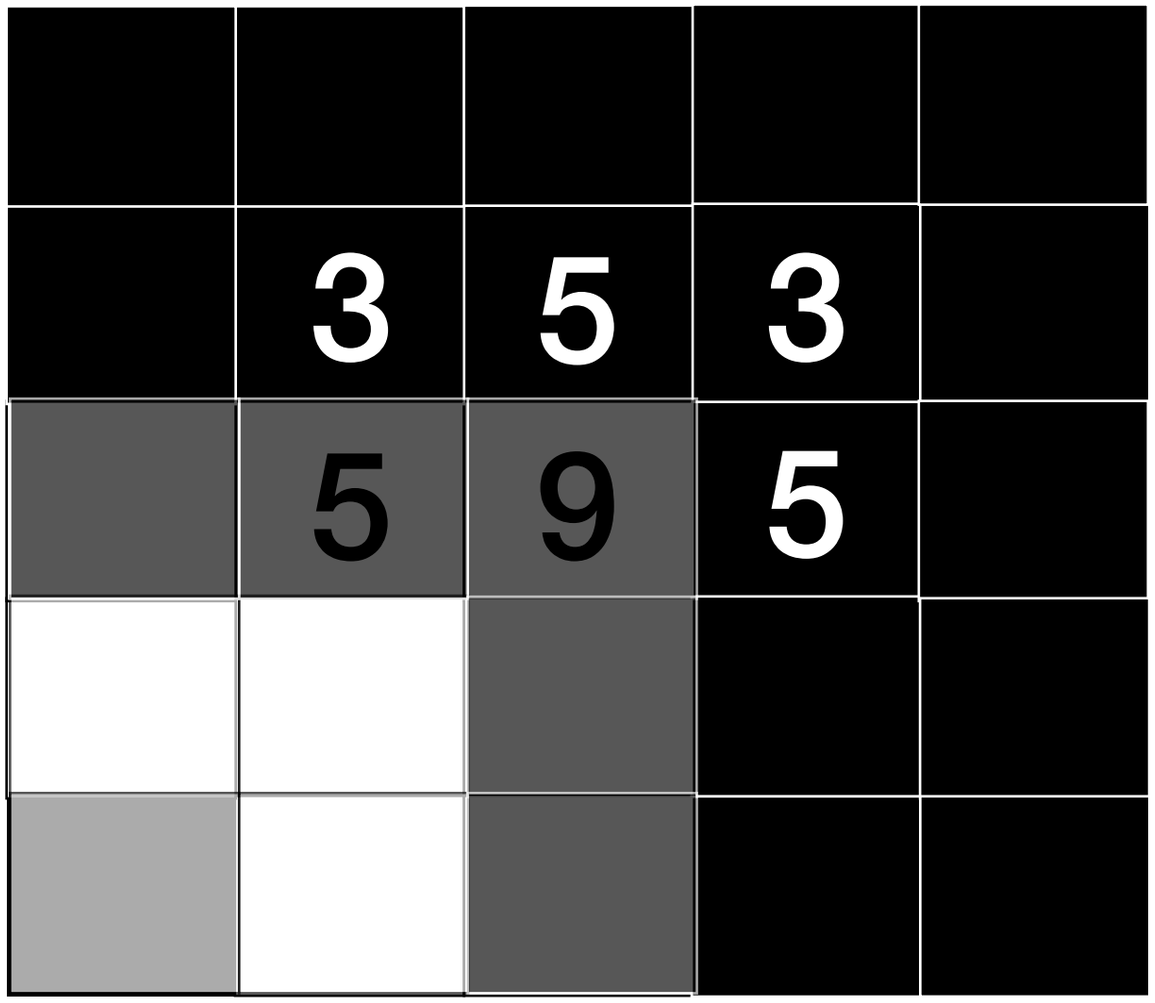
Convolution



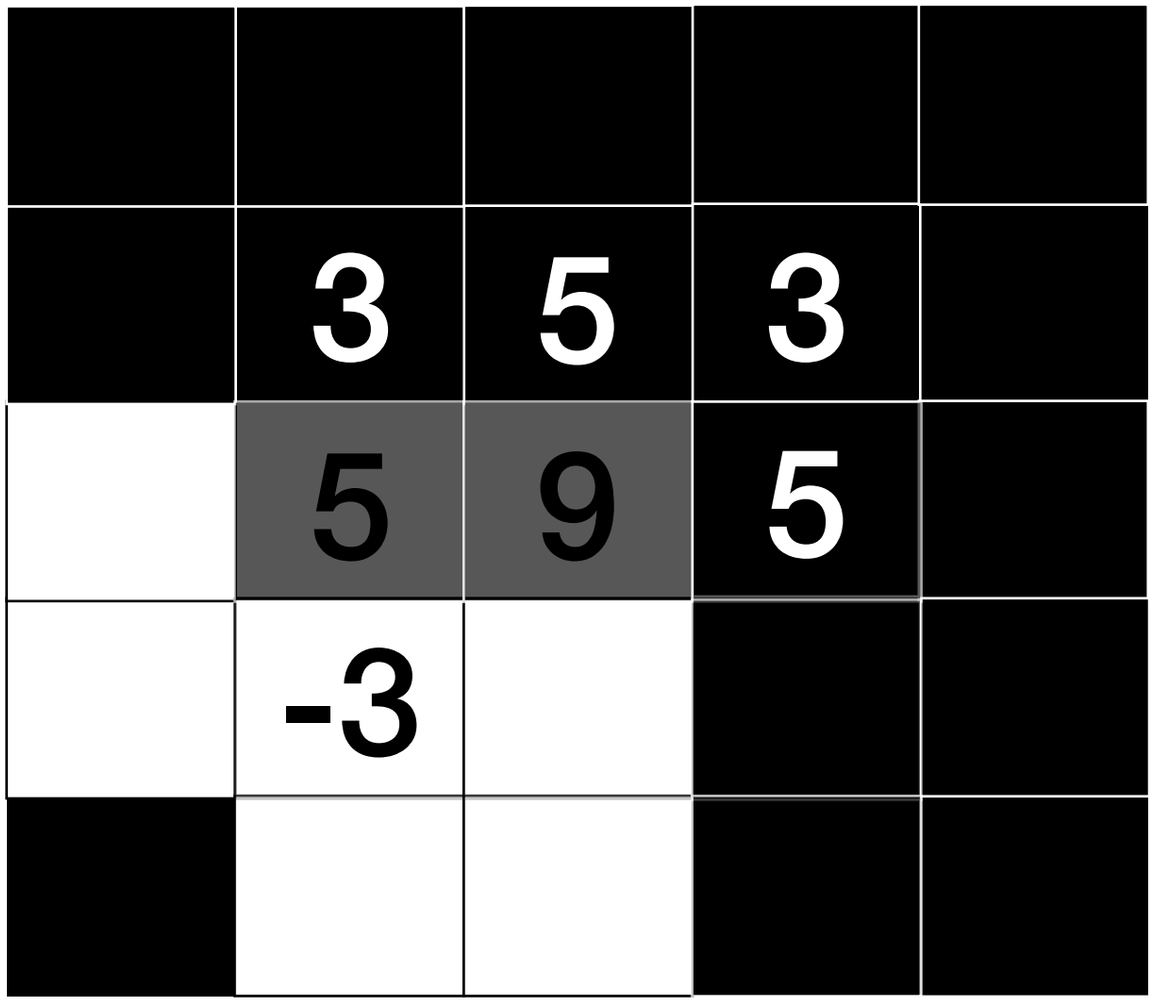
Convolution



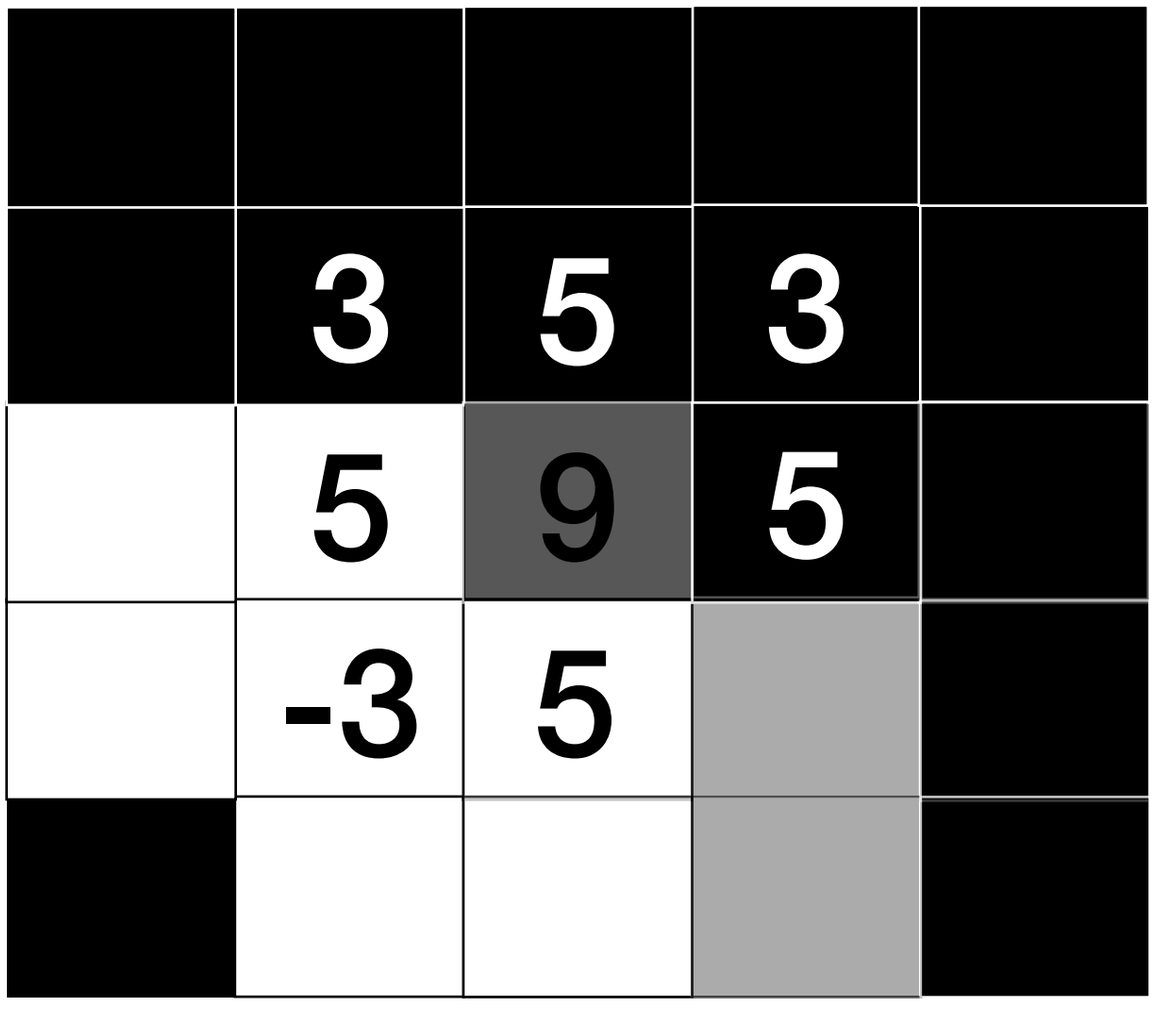
Convolution



Convolution



Convolution



Convolution

	3	5	3	
	5	9	5	
	-3	5	3	

Convolution

0	-2	-2	-2	-4
2	3	5	3	-4
2	5	9	5	-2
-4	-3	5	3	-2
-4	-4	2	2	0

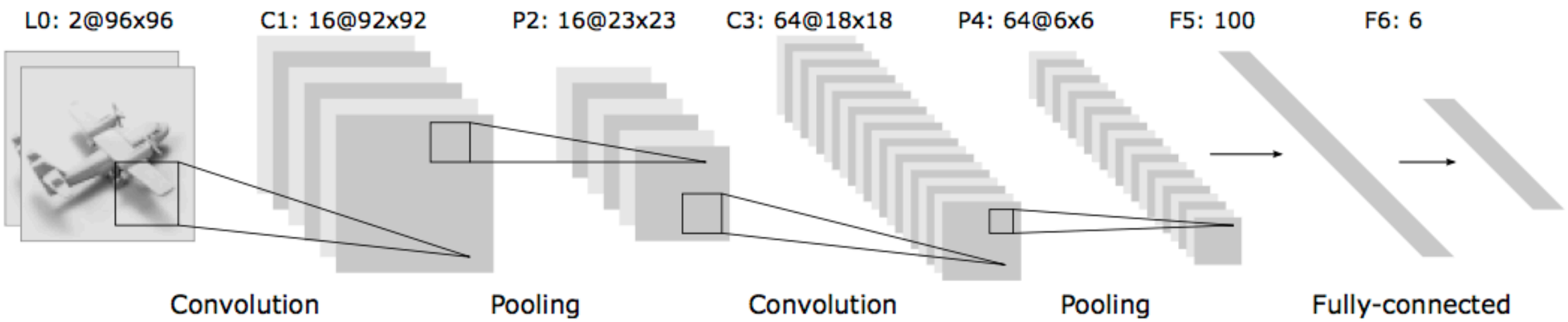
The edge pixels of the image see only part of the filter and are often ignored.

In practice, many filter positions are evaluated in parallel using a graphics processor or a special 'neural processor unit'.
There is only *one set of weights per filter*, independent of position.
Using the same weights in all positions assumes *translation invariance,* either in space (images) or time (sounds).
The filter weights are adjusted using back propagation.

What is ‘pooling’?

The *NORB Network* again.

Outputs	Pooled	Unpooled	Weights	Unpooled
L0	18,432	18,432		
C1	135,424	135,424	400	400
P2	8,464	135,424		
C3	20,736	311,040	2,304	2,304
P4	2,304	311,040		
F5	100	2,799,360	230,400	31,104,000
F6	6	6	600	600



Pooling

Finding patterns of *features*

0	-2	-2	-2	-4
2	3	5	3	-4
2	5	9	5	-2
-4	-3	5	3	-2
-4	-4	2	2	0

Notice that ‘features’ cannot be smaller than the size of the filter.

Possible pooling methods are to take (weighted) averages or find the maximum value. Maximum seems to work well.

Libraries

- Caffe: A popular library for convolutional neural networks. It supports both CPU and GPU. Developed in C++, and has Python and MATLAB wrappers.

- MXNet: An open-source deep learning framework which is scalable, including support for multiple GPUs and CPUs in distribution. It supports interfaces in multiple languages (C++, Python, Julia, Matlab, JavaScript, Go, R, Scala, Perl, Wolfram Language).

- neon: The fastest framework for convolutional neural networks and Deep Learning with support for GPU and CPU backends. The front-end is in Python. Acquired by *Intel*.

- TensorFlow: Apache 2.0-licensed library with support for CPU, GPU and Google's proprietary TPU.

- Theano: The reference deep-learning library for Python with an API largely compatible with the popular NumPy library. Allows user to write symbolic mathematical expressions, then automatically generates their derivatives, saving the user from having to code gradients or back-propagation.

- Torch (www.torch.ch): A scientific computing framework with wide support for machine learning algorithms. It is used at *Facebook* AI Research and *Twitter*.

Summary

The limitations of the simple *Perceptron* caused a long delay in development.

The proof that 3 layers are enough led to further delays.

Current progress in face and speech recognition is based on *deep* networks.

Convolutional filters provide *translational invariance*. They have few weights and are efficiently implemented using GPUs.

Currently rotational and scale invariance are provided by fully-connected networks—with luck! (Scope for research!)

Back propagation works, but it's slow & not how nature does it.