CAP 5610: Machine Learning

Lecture 9:
Deep Learning: Convolutional NNs

Instructor: Dr. Gita Sukthankar
Email: gitars@eecs.ucf.edu

Reference Materials From: GuoJun Qi, Yann LeCun, Rahul Sukthankar, M. Ranzato, and CNN tutorial

Reading

http://cs231n.github.io/convolutional-networks/

Chap 9 in Intro to Deep Learning

Homework #2: SVM

- Use SVM to classify UCI Glass identification set using libsvm or sci-kit learn
- Apply different kernels and find the optimal hyperparameters
- Implement one vs. all multiclass algorithm using SVM
- Reweight the training examples to fix the class imbalance problem

So, 1. **what exactly is deep learning** ?

And, 2. **why is it generally better** than other methods on image, speech and certain other types of data?

The short answers

1. **'Deep Learning' means** using a **neural network** with **several layers of nodes** between input and output

2. **the series of layers between input & output do feature identification and processing in a series of stages, just as our brains seem to.**

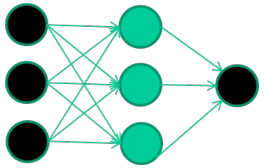
But:

3. **multilayer neural networks have been around for 25 years. What's actually new?**

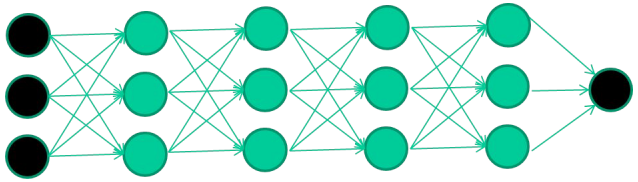
But:

3. **multilayer neural networks have been around for 25 years. What's actually new?**

we have always had good algorithms for learning the weights in networks with 1 hidden layer



but these algorithms are not good at learning the weights for networks with more hidden layers

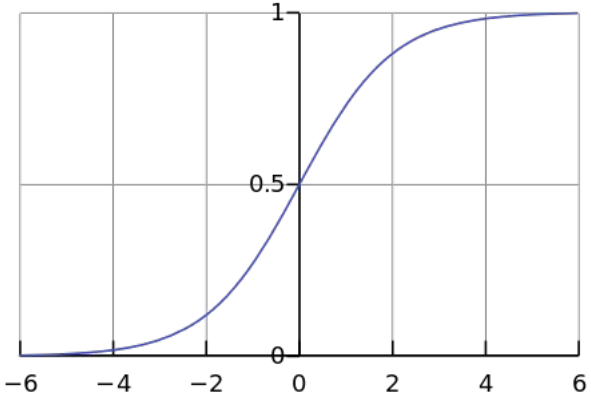


what's new is: **<u>algorithms for training many-later networks</u>**

longer answers

1. reminder/quick-explanation of how neural network weights are learned;
2. the idea of **unsupervised feature learning** (why ‘intermediate features’ are important for difficult classification tasks, and how NNs seem to naturally learn them)
3. The ‘breakthrough’ – the simple trick for training Deep neural networks

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



-0.06

W1

-2.5

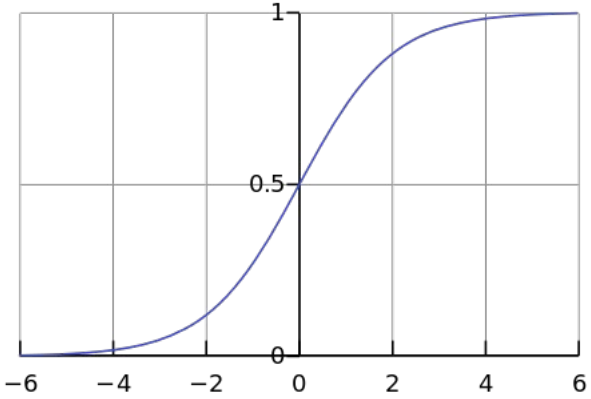
W2

W3

1.4

f(x)

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



-0.06

2.7

-2.5

-8.6

***f*(*x*)**

0.002

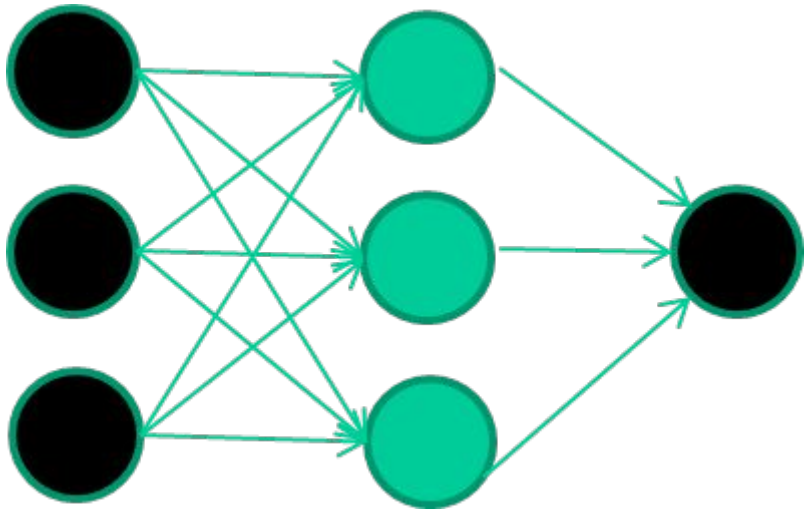
1.4

$x = -0.06 \times 2.7 + 2.5 \times 8.6 + 1.4 \times 0.002 = 21.34$

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

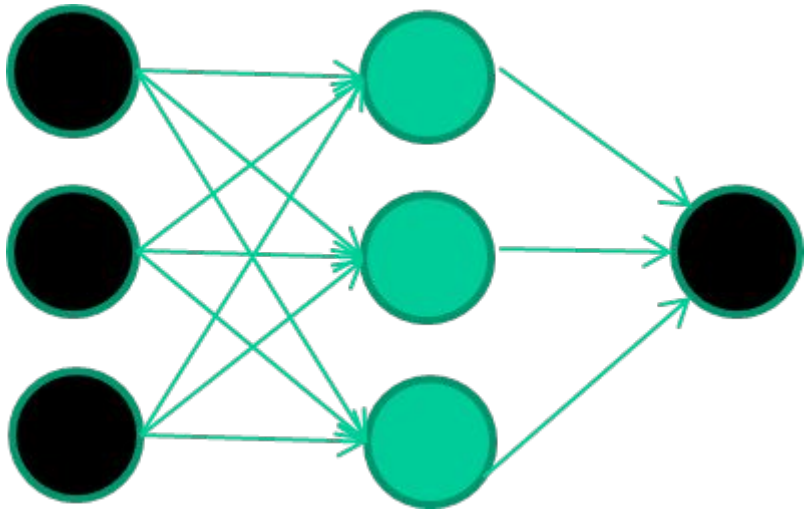
etc …



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 …

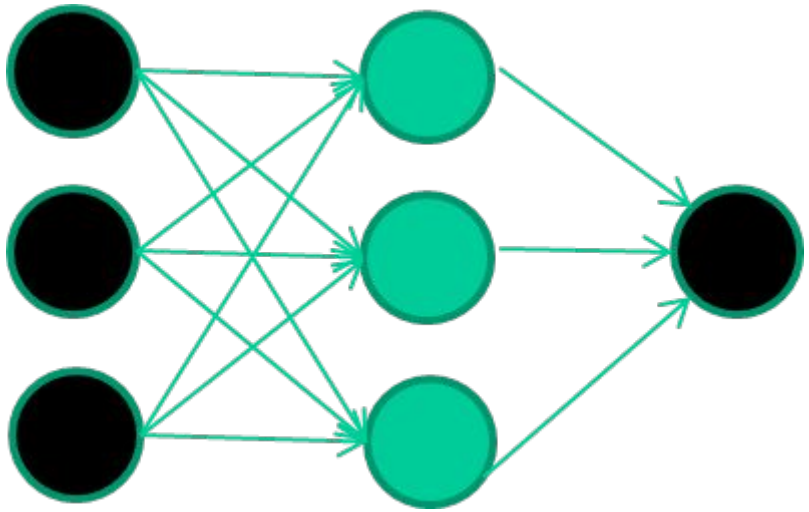


Training data

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 …

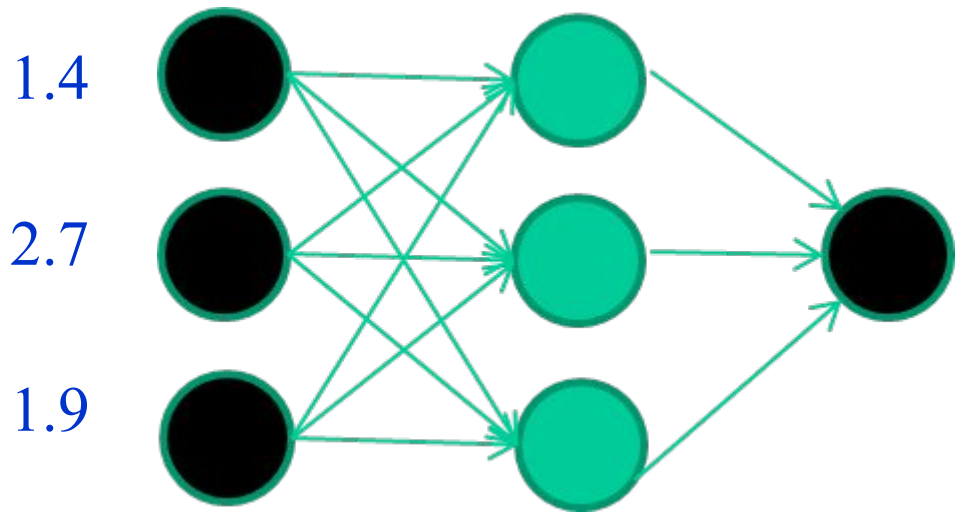
Initialise with random weights



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 …			

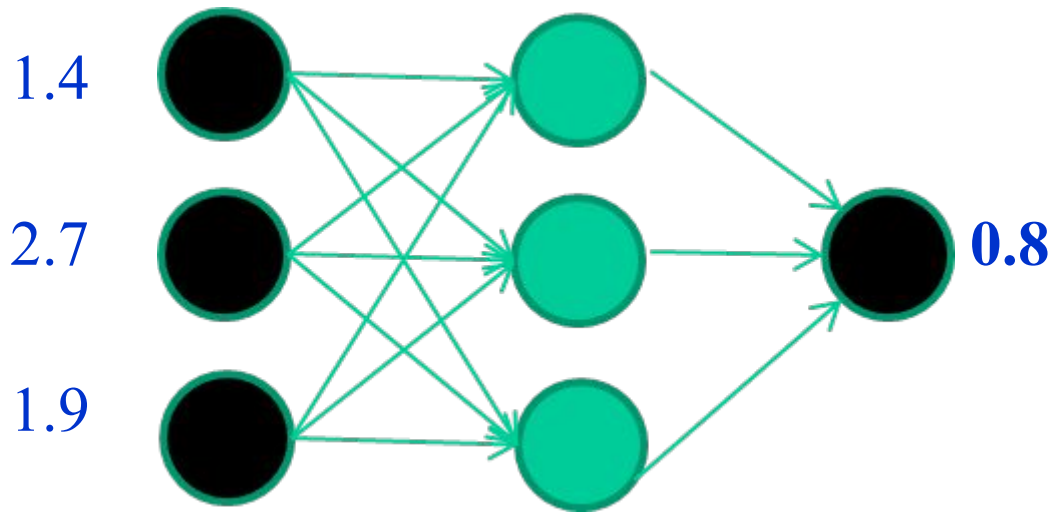
Present a training pattern



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 …			

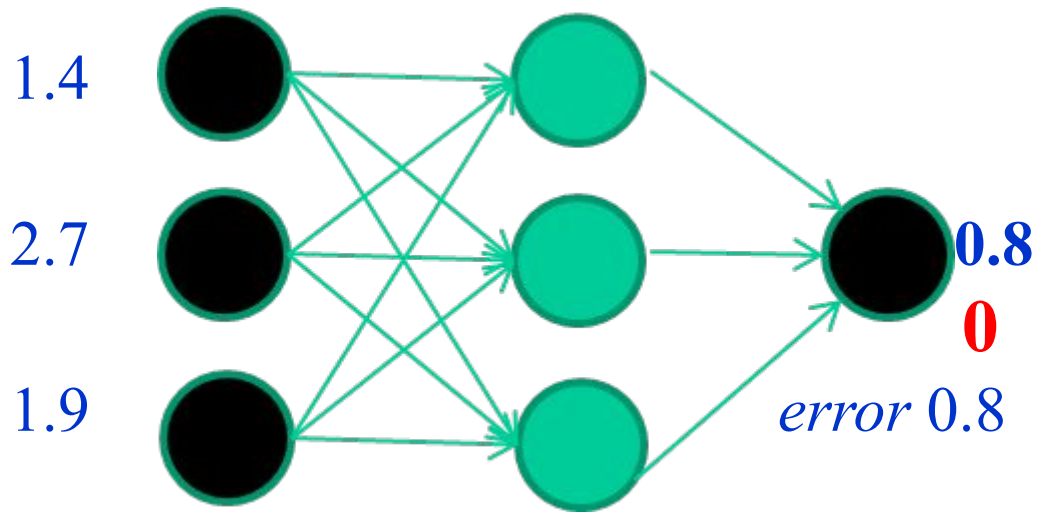
Feed it through to get output



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 …			

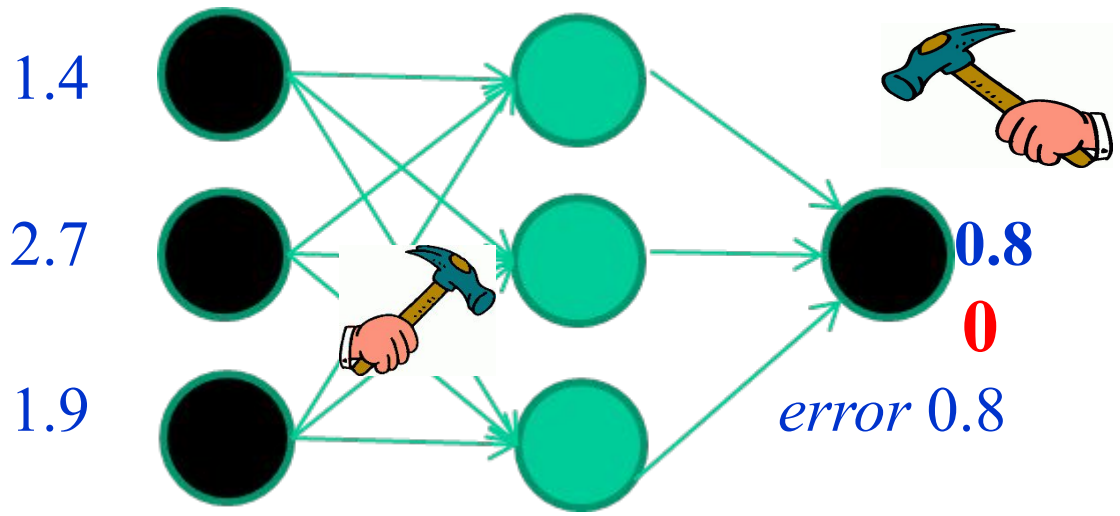
Compare with target output



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

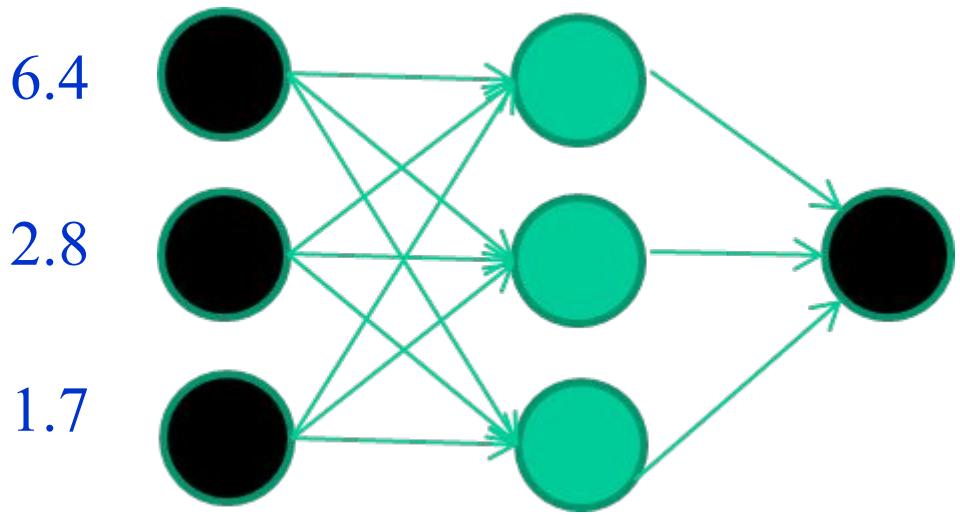


Training data

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

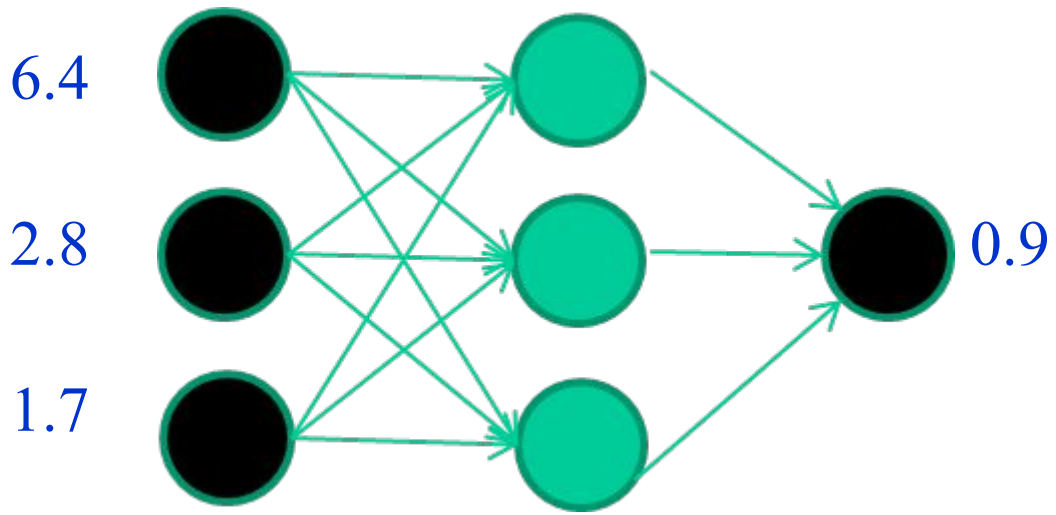


Training data

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

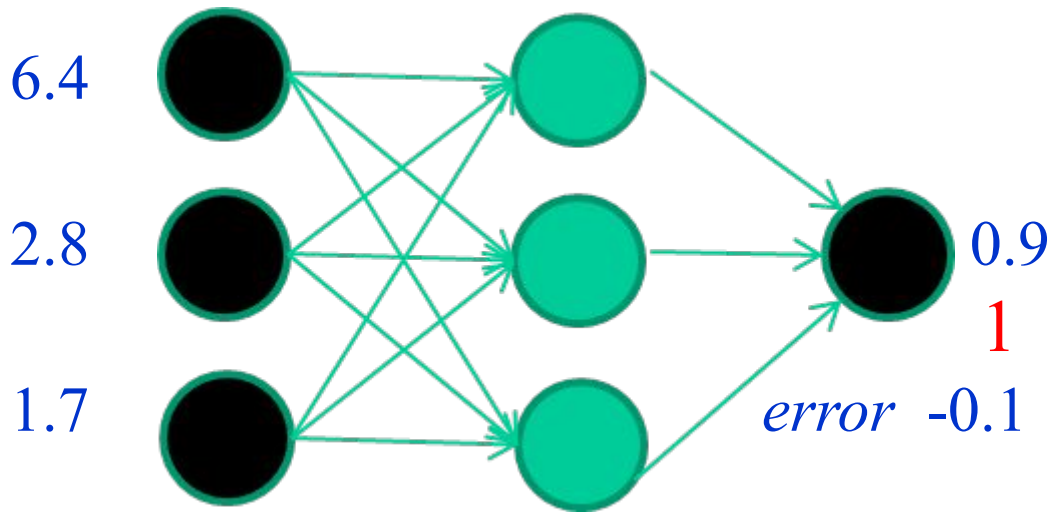


Training data

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

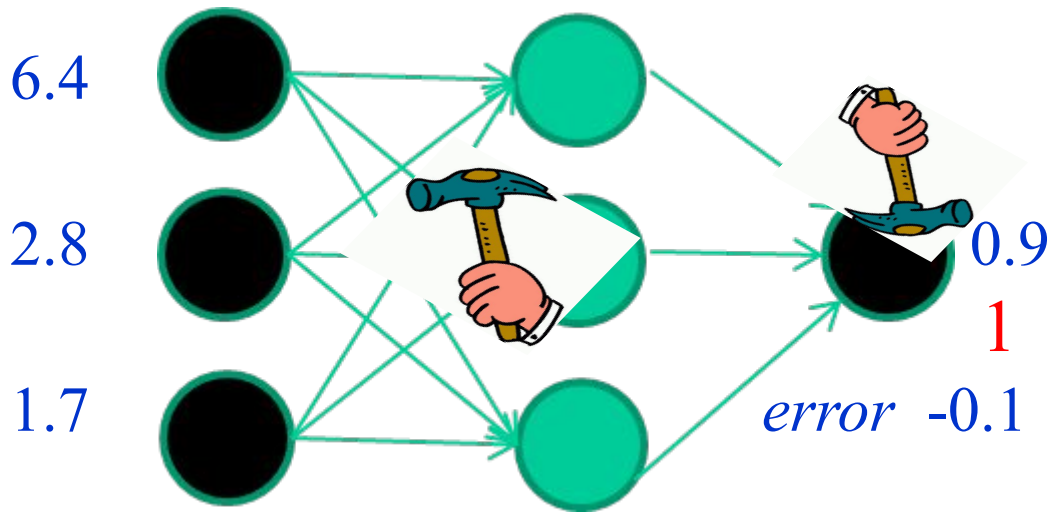


Training data

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

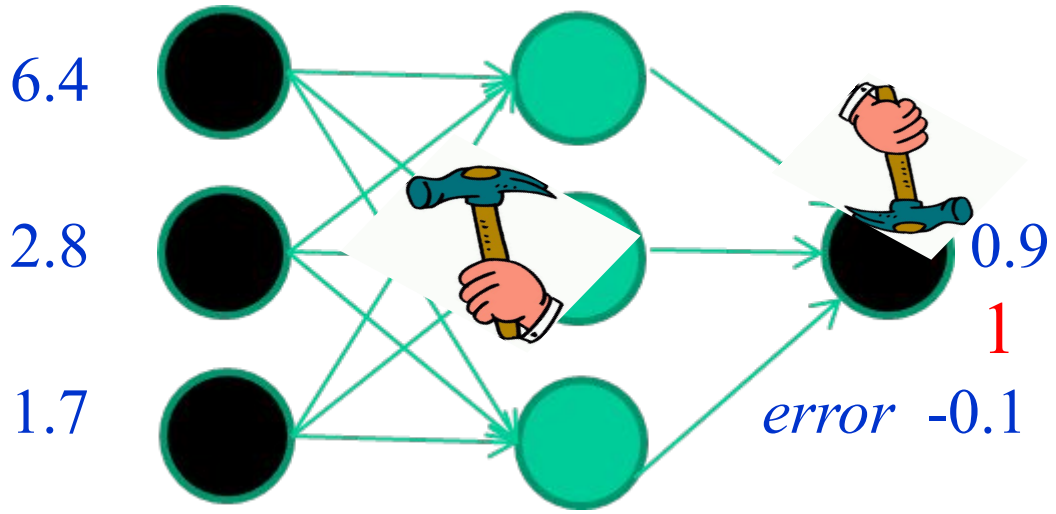


Training data

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 ….

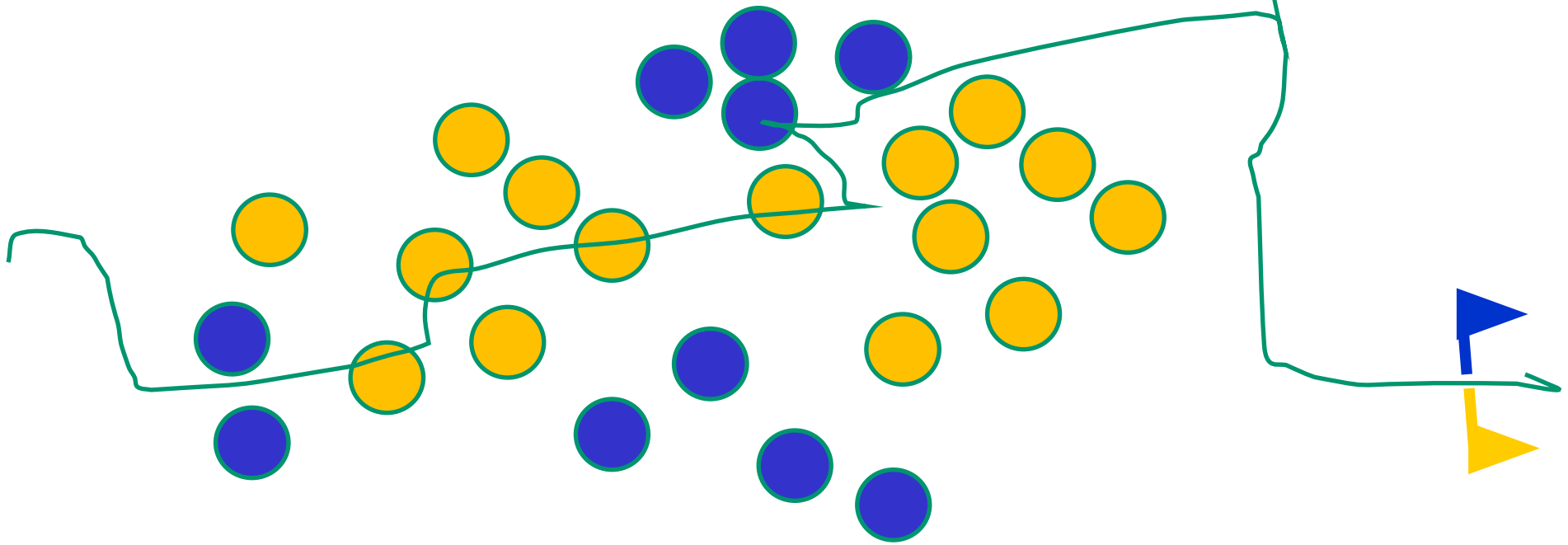


Repeat this thousands, maybe millions of times – each time taking a random training instance, and making slight weight adjustments

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

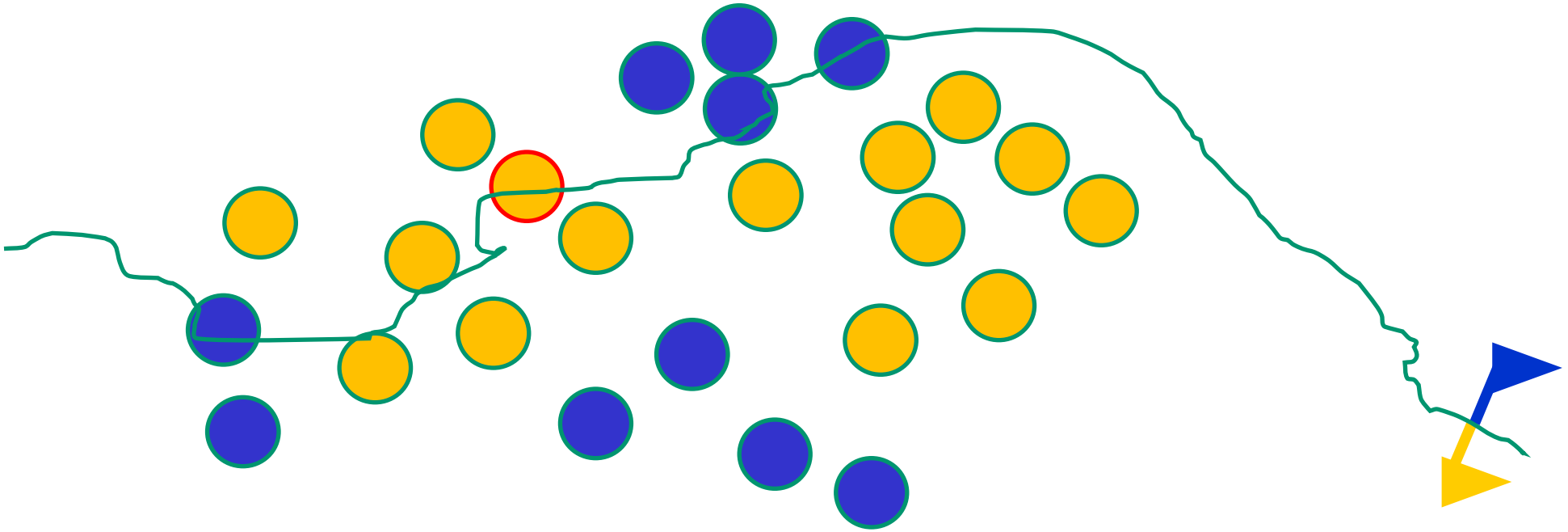
The decision boundary perspective…

Initial random weights



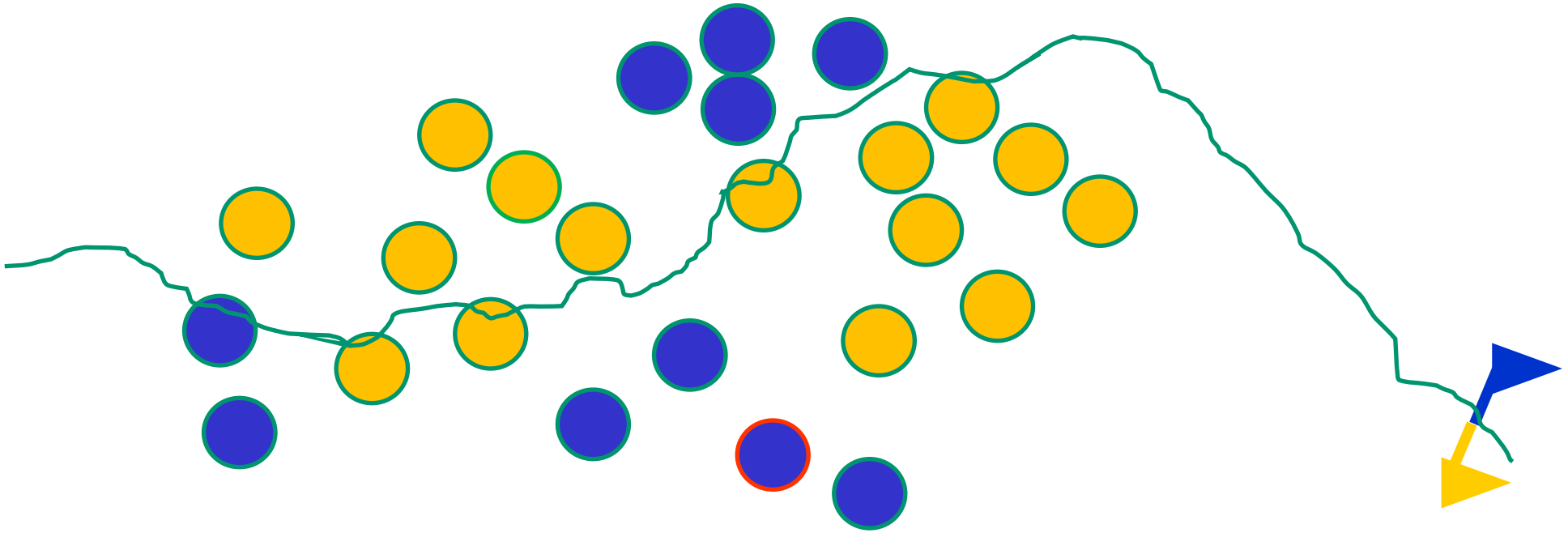
The decision boundary perspective…

Present a training instance / adjust the weights



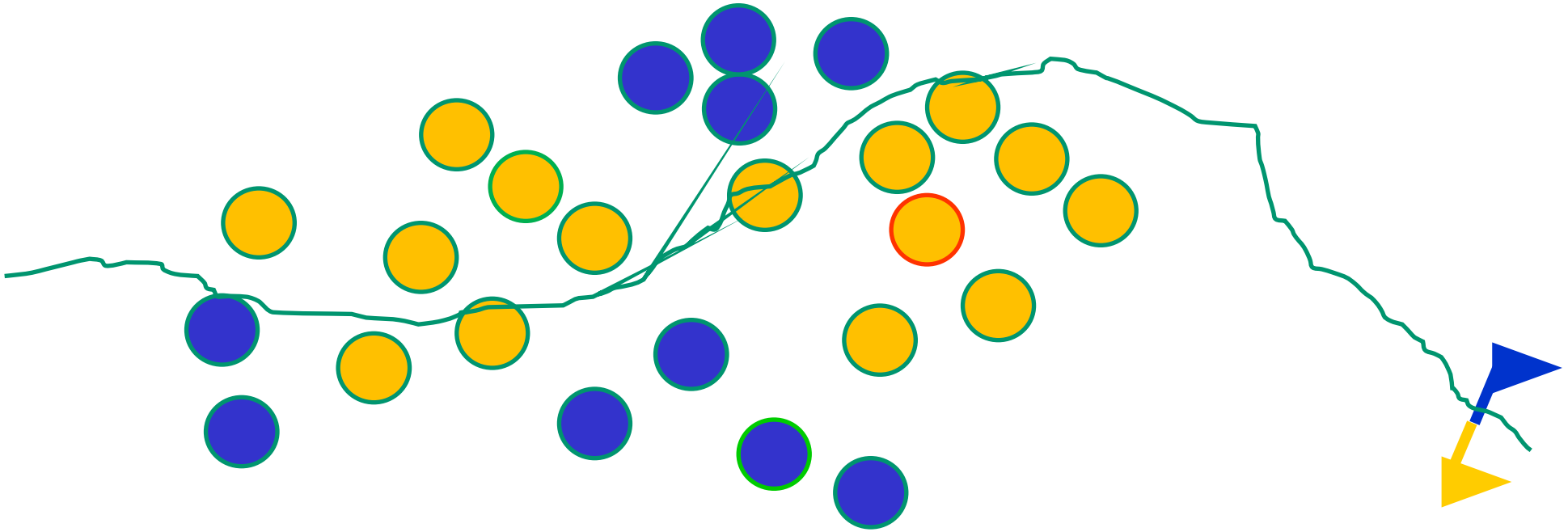
The decision boundary perspective…

Present a training instance / adjust the weights



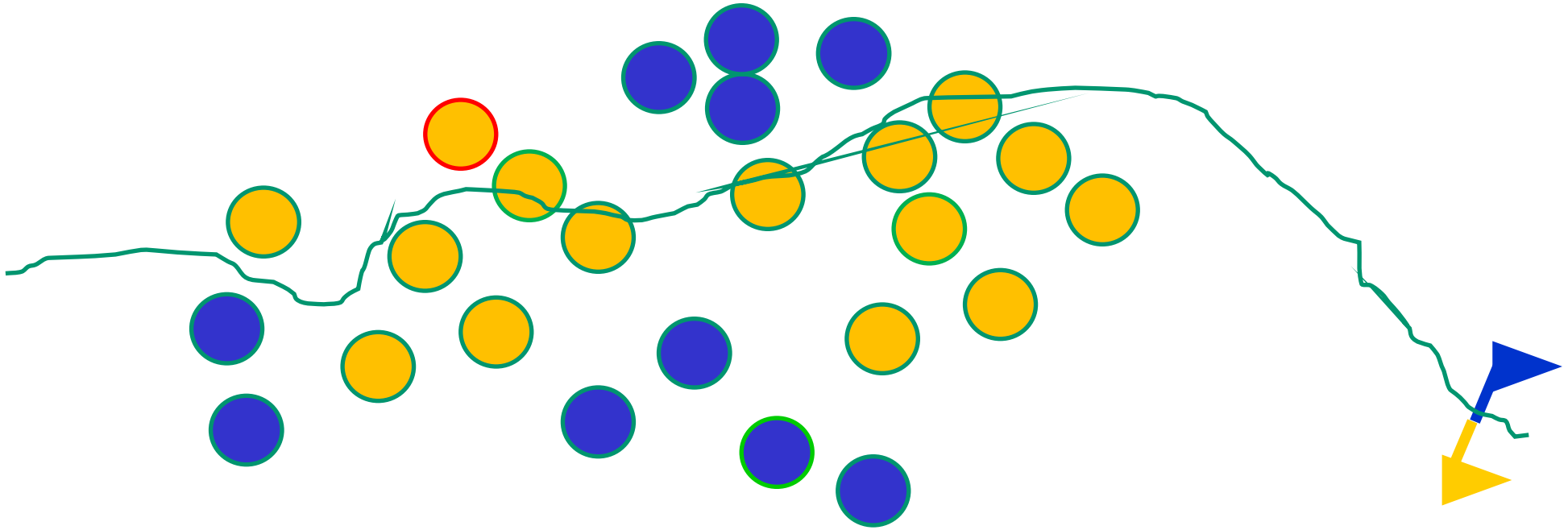
The decision boundary perspective…

Present a training instance / adjust the weights



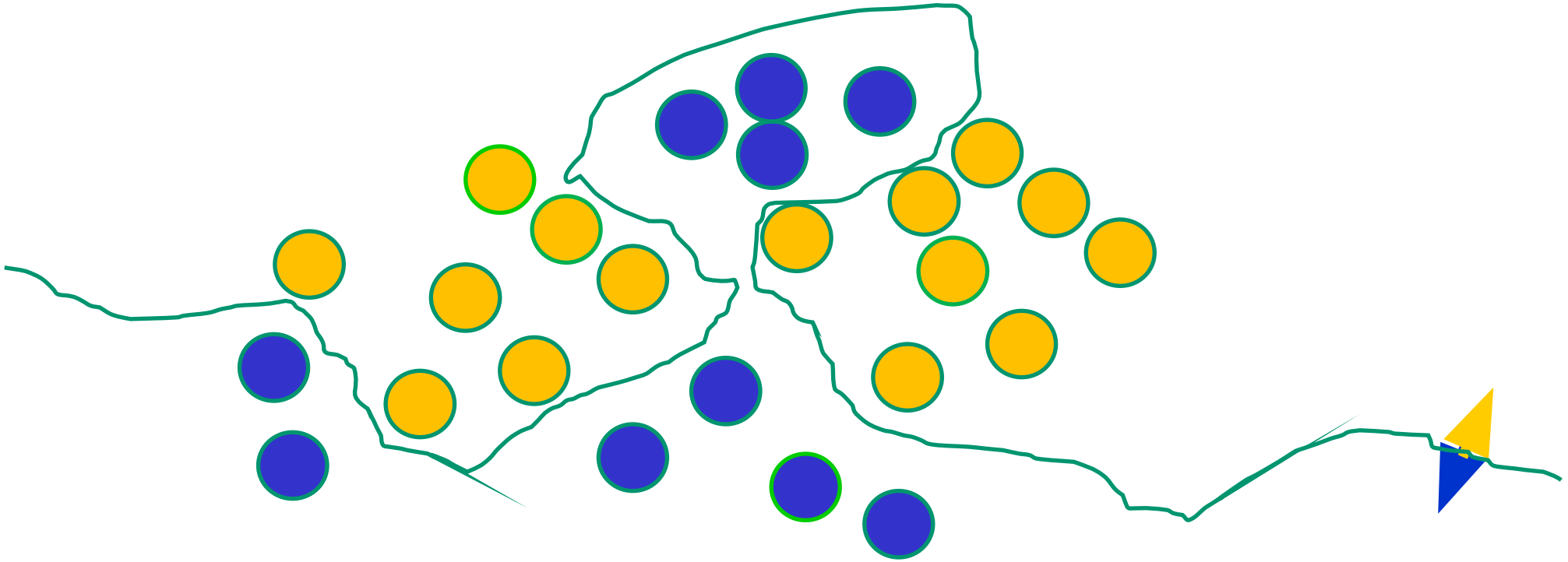
The decision boundary perspective…

Present a training instance / adjust the weights



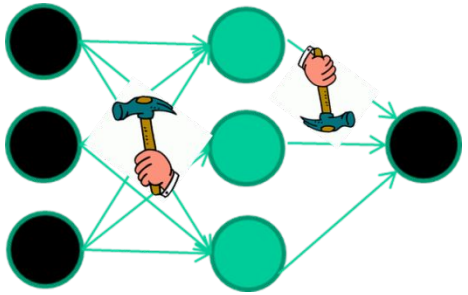
The decision boundary perspective…

Eventually ….



Summary

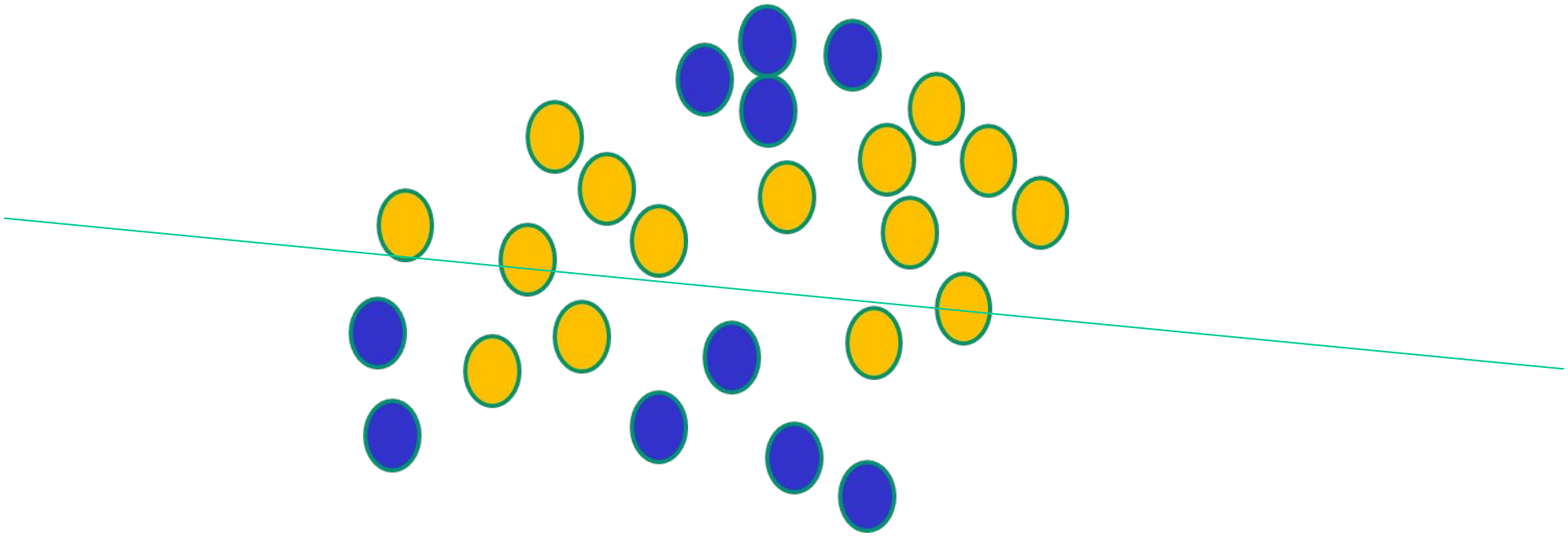
- weight-learning algorithms for NNs are dumb
- they work by making thousands and thousands of tiny adjustments, each making the network do better at the most recent pattern, but perhaps a little worse on many others
- but, by dumb luck, eventually this tends to be good enough to learn effective classifiers for many real applications



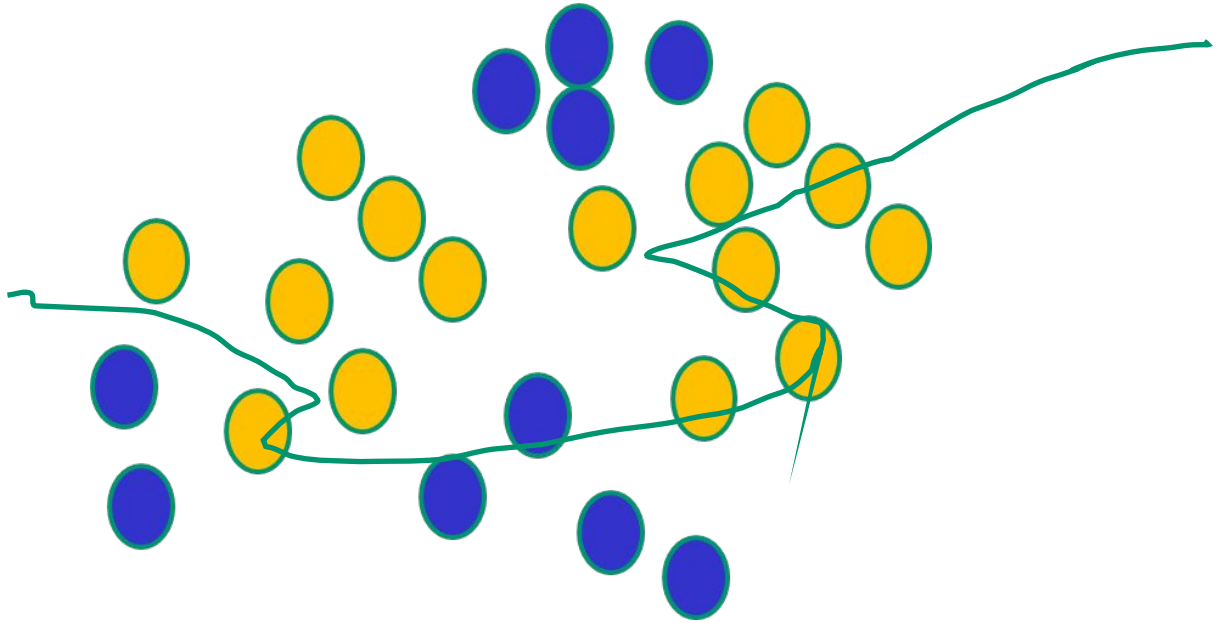
Universal Approximation Theorem

If $f(x)$ is non-linear, a network with 1 hidden layer can, in theory, learn perfectly any classification problem. A set of weights exists that can produce the targets from the inputs. The problem is finding them.

If $f(x)$ is linear, the NN can **only** draw straight decision boundaries (even if there are many layers of units)

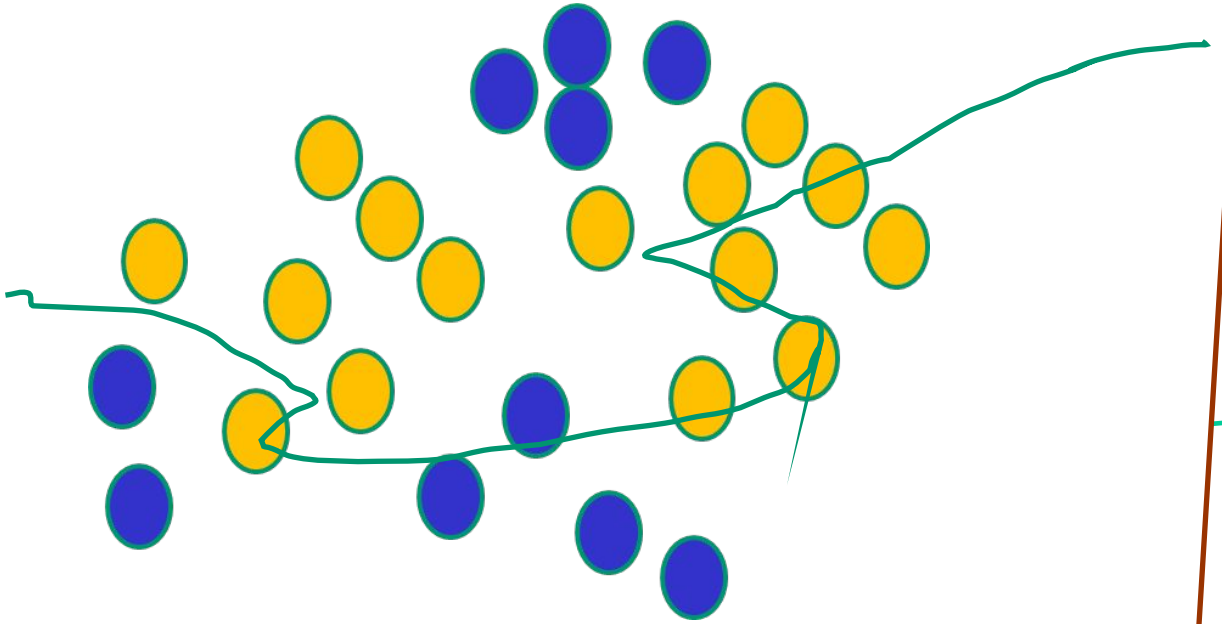


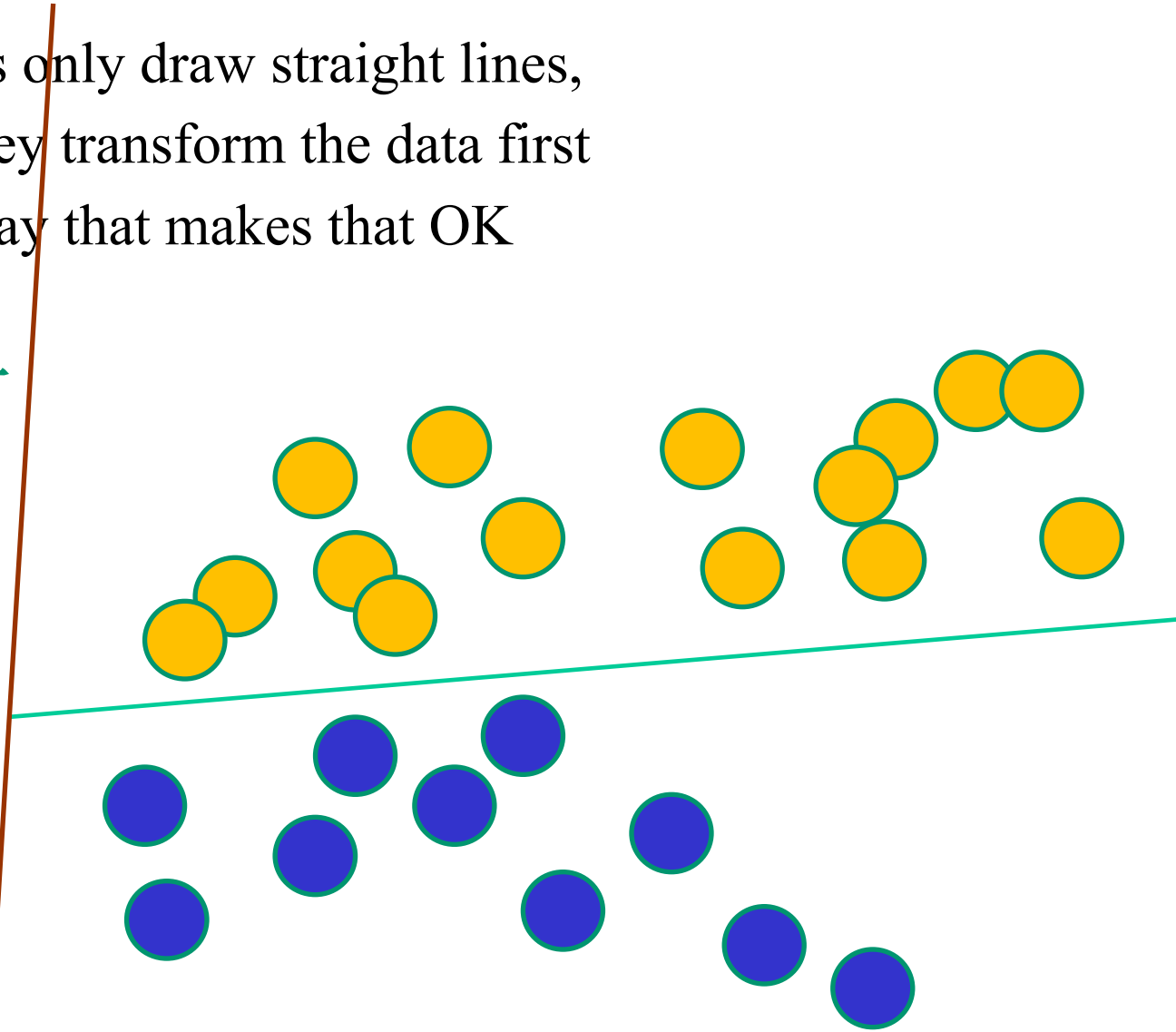
NNs use nonlinear $f(x)$ so they can draw complex boundaries, but keep the data unchanged



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





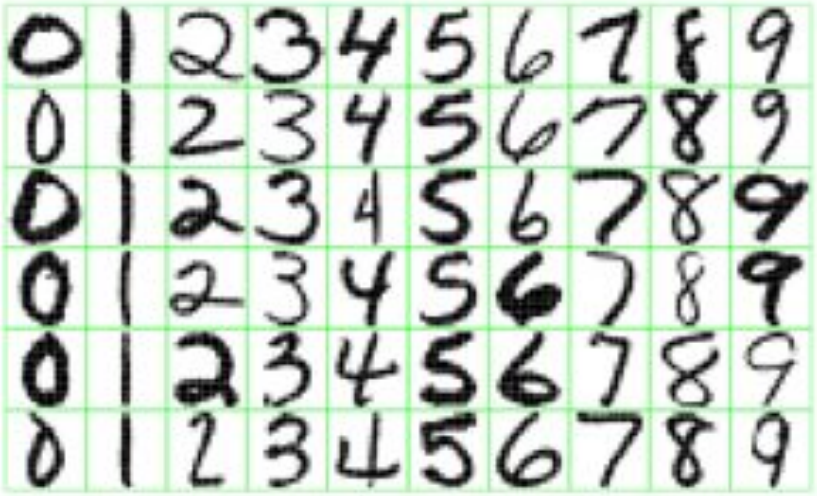
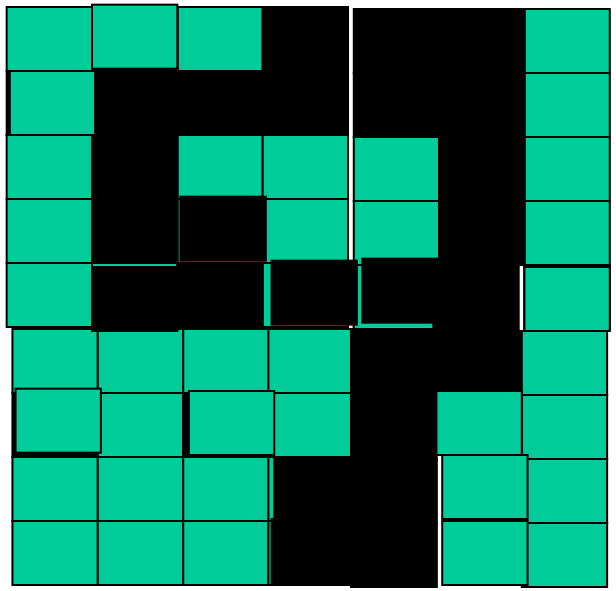
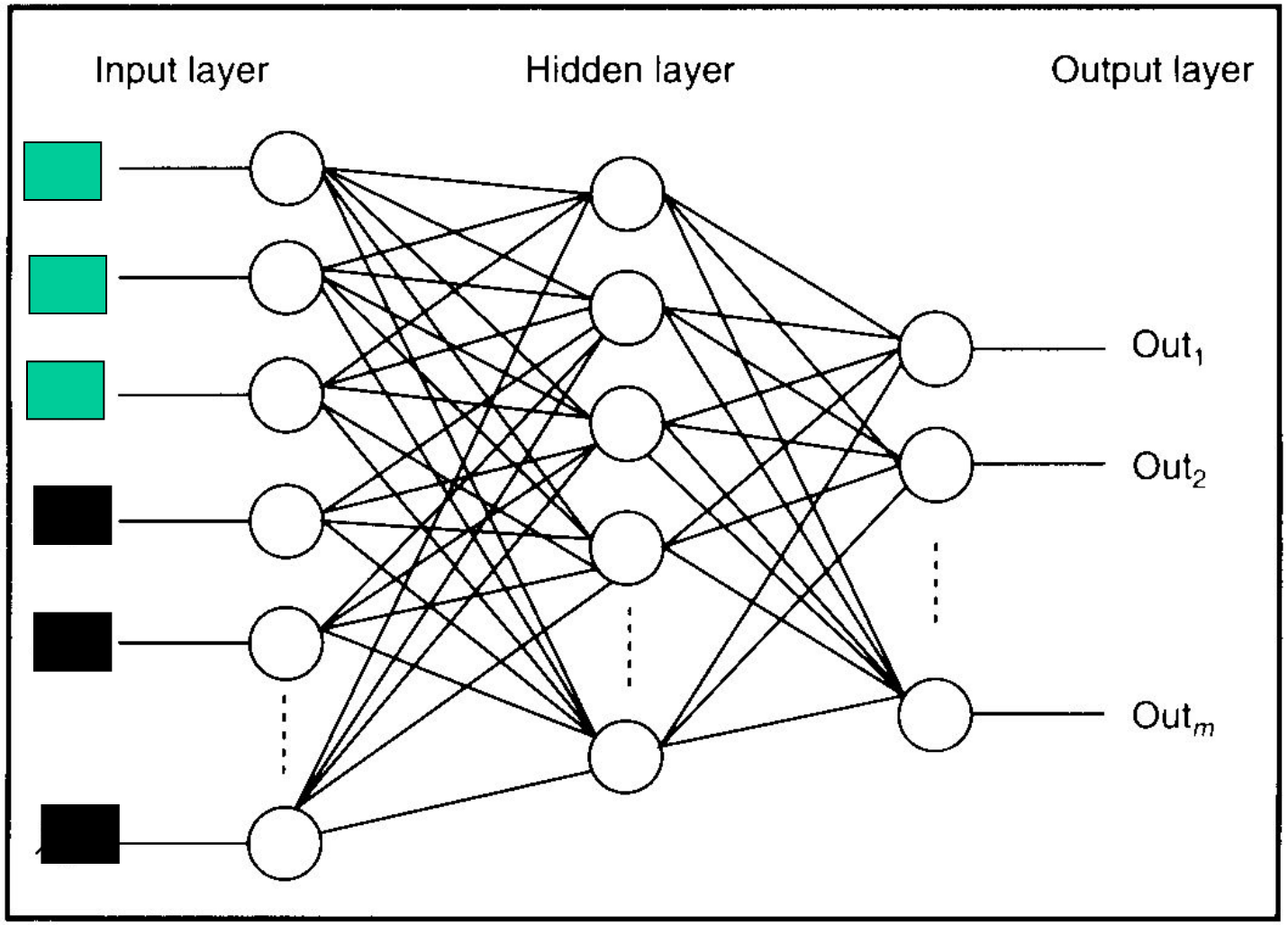


Figure 1.2: *Examples of handwritten digits from U.S. postal envelopes.*



Feature detectors



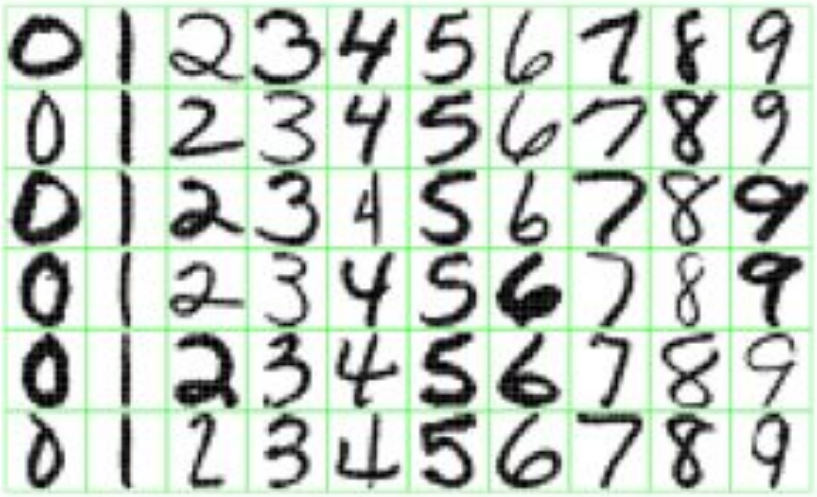
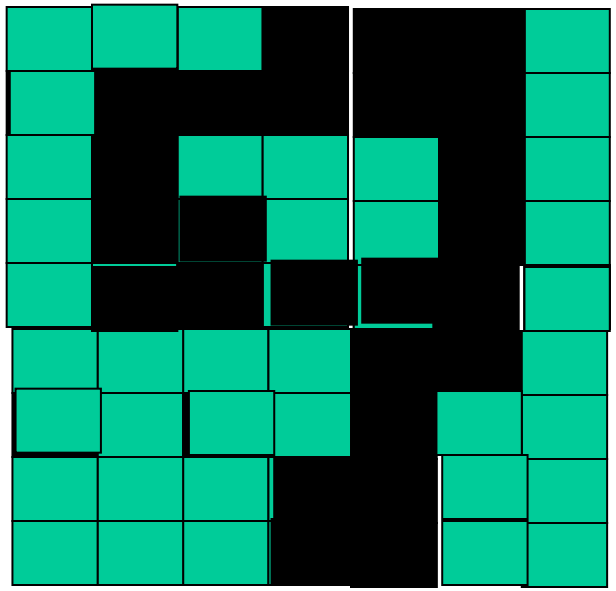
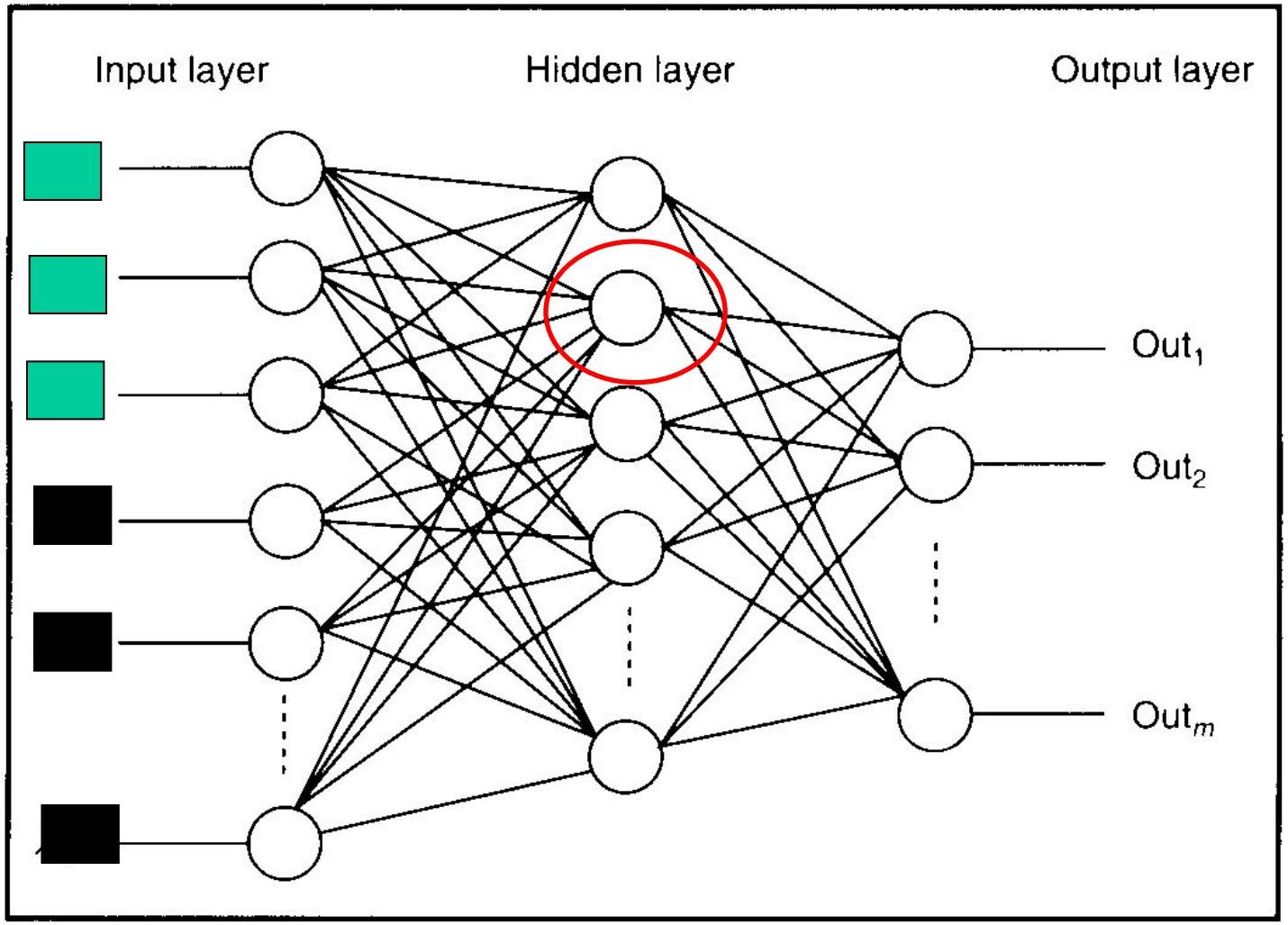


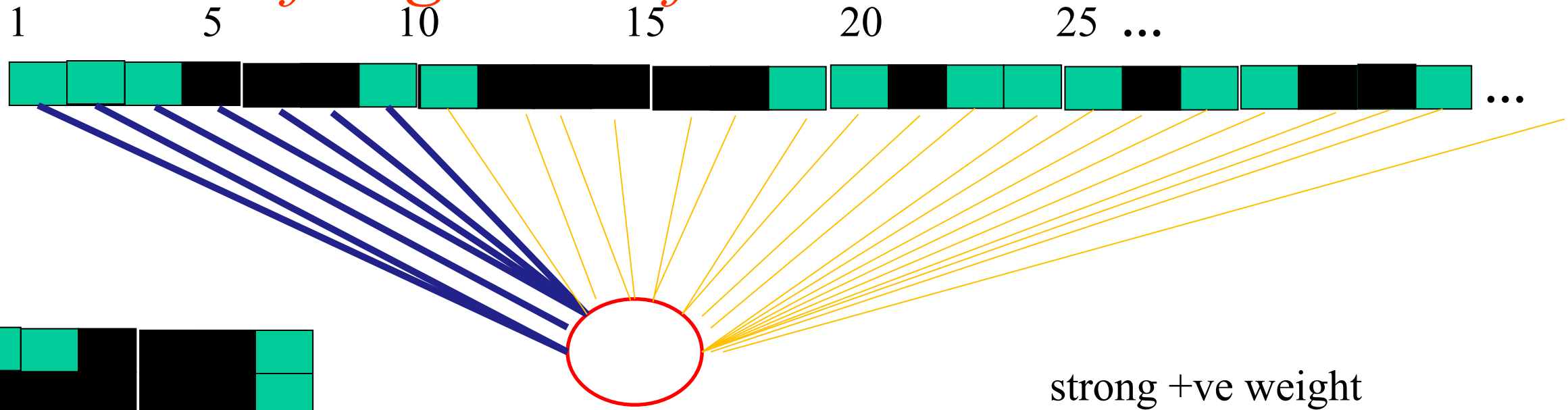
Figure 1.2: *Examples of handwritten digits from U.S. postal envelopes.*

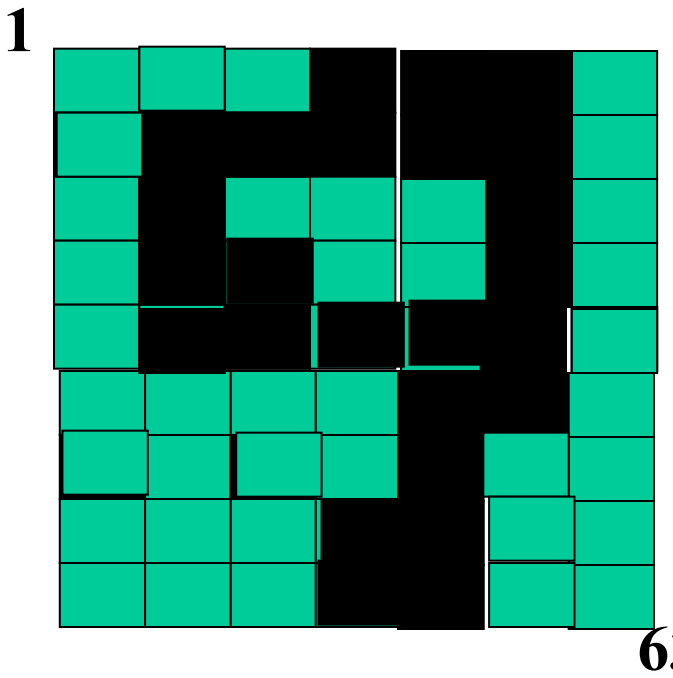


what is this unit doing?



Hidden layer units become *self-organised feature detectors*

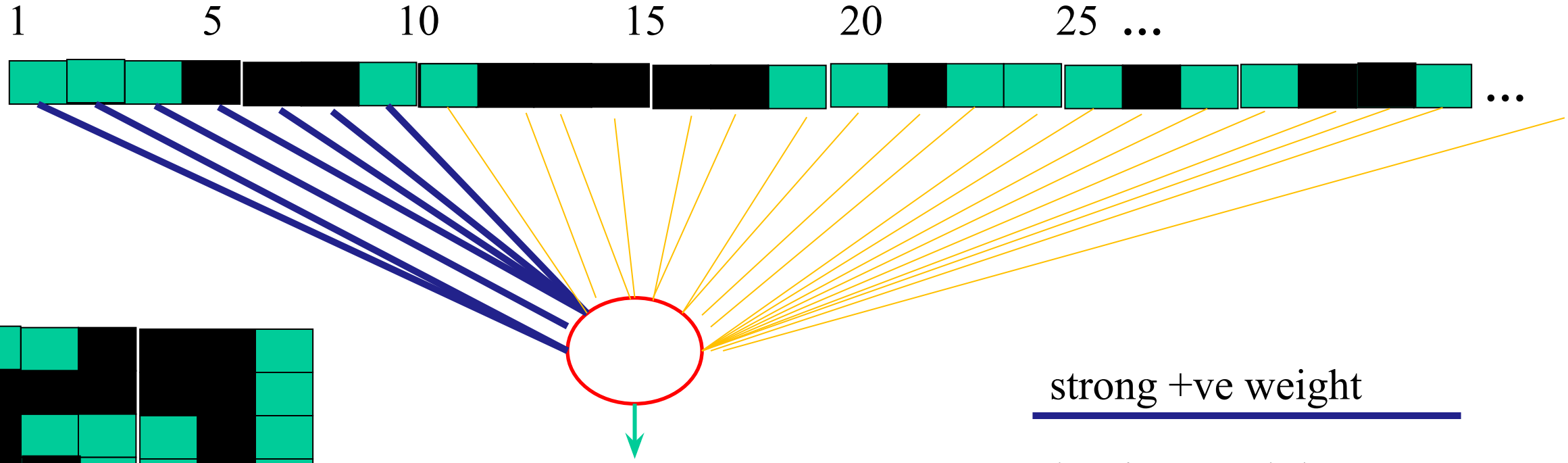


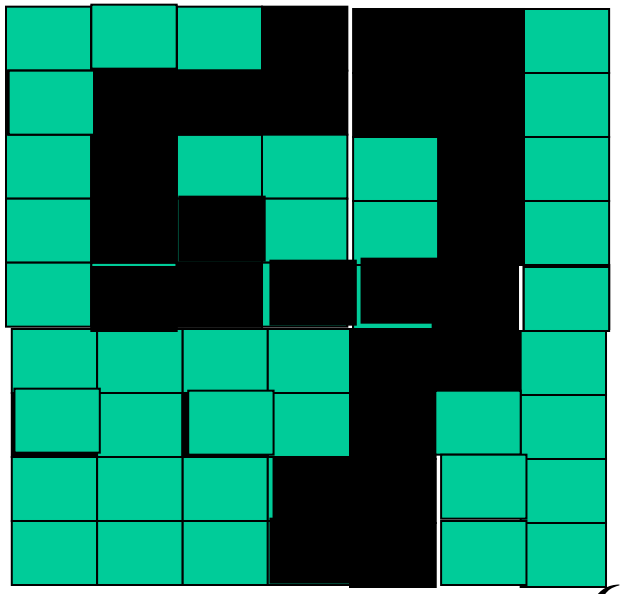


strong +ve weight

low/zero weight

What does this unit detect?

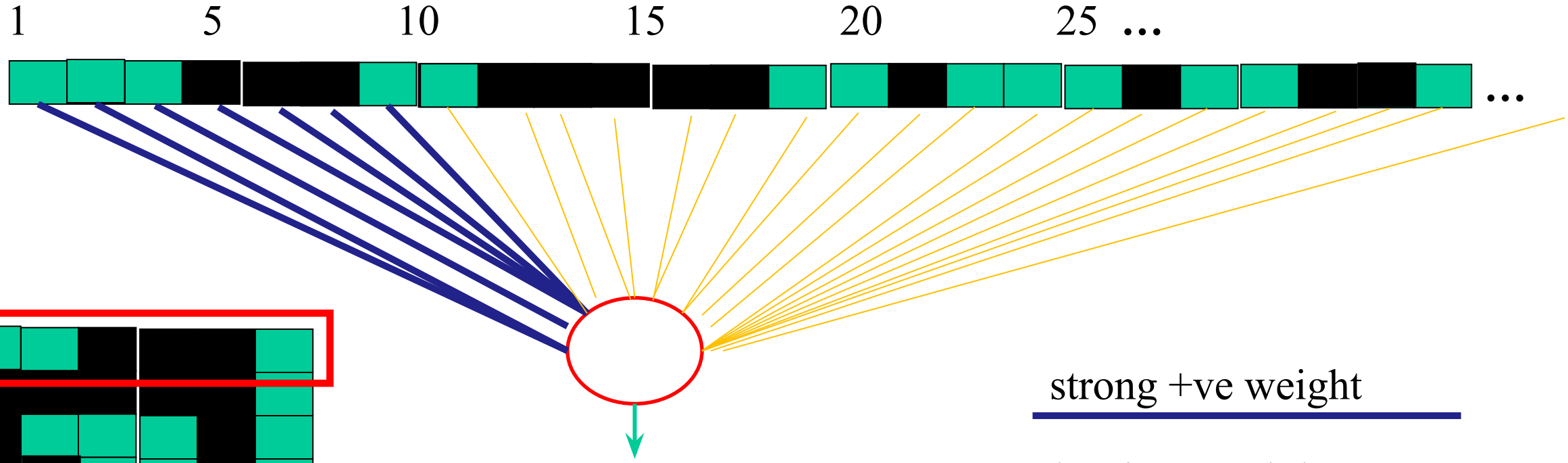


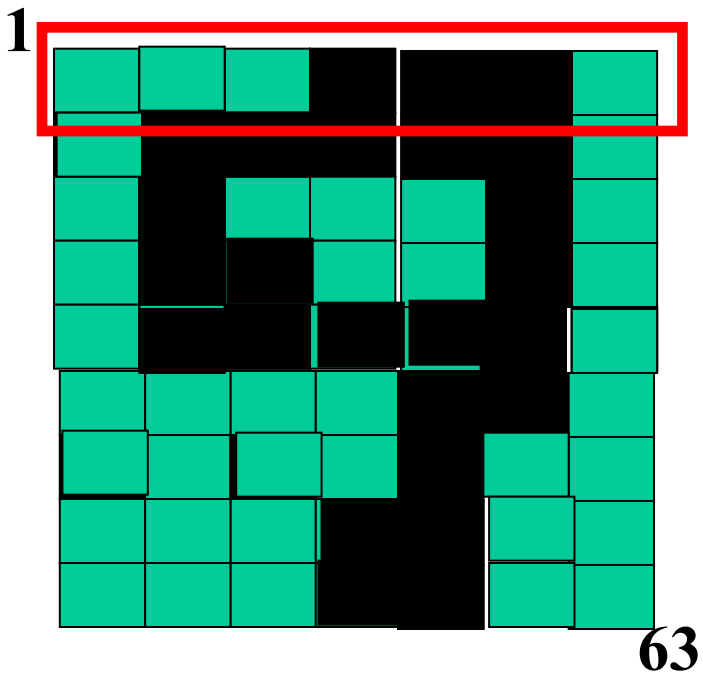


strong +ve weight

low/zero weight

What does this unit detect?



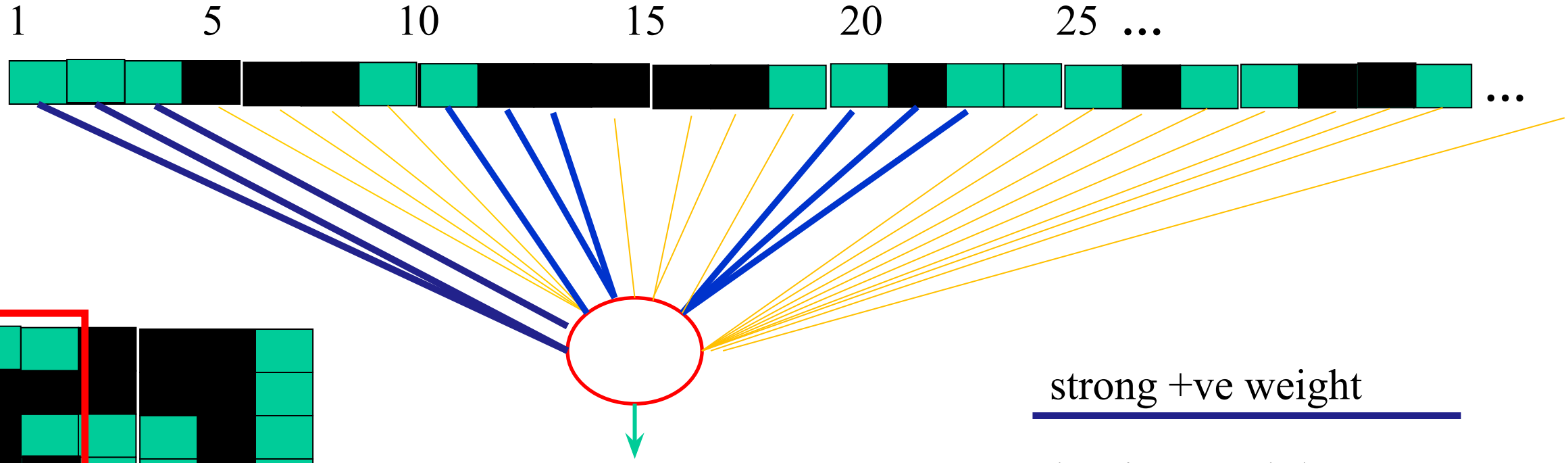


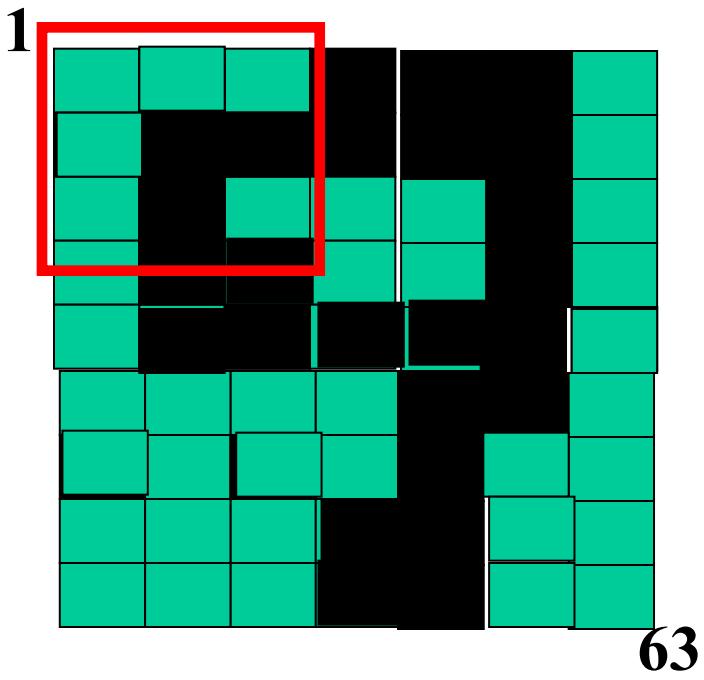
strong +ve weight

low/zero weight

it will send strong signal for a horizontal line in the top row, ignoring everywhere else

What does this unit detect?

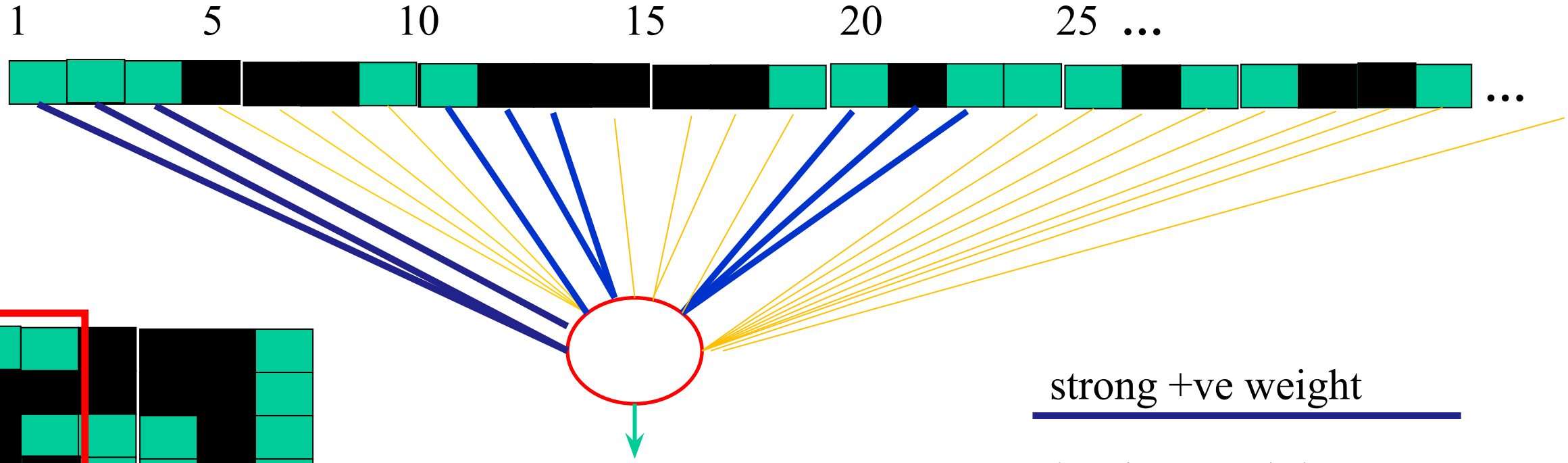


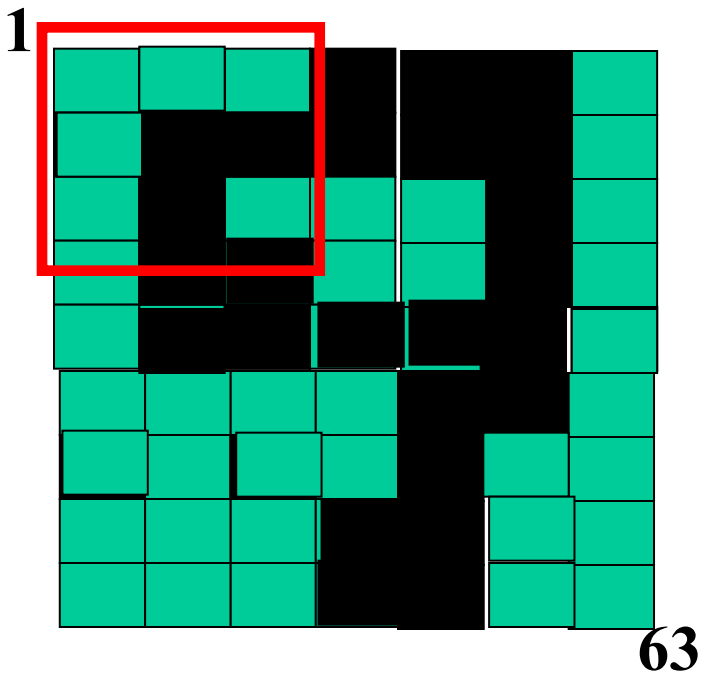


strong +ve weight

low/zero weight

What does this unit detect?





strong +ve weight

low/zero weight

Strong signal for a dark area in the top left corner

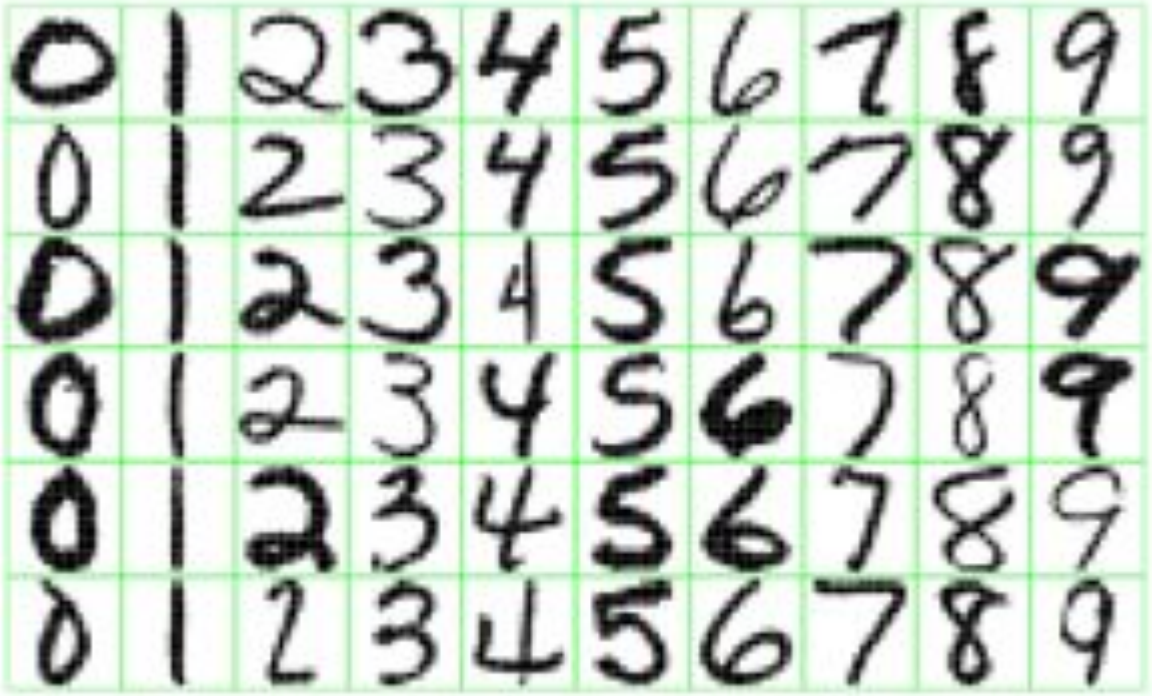


Figure 1.2: *Examples of handwritten digits from U.S. postal envelopes.*

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

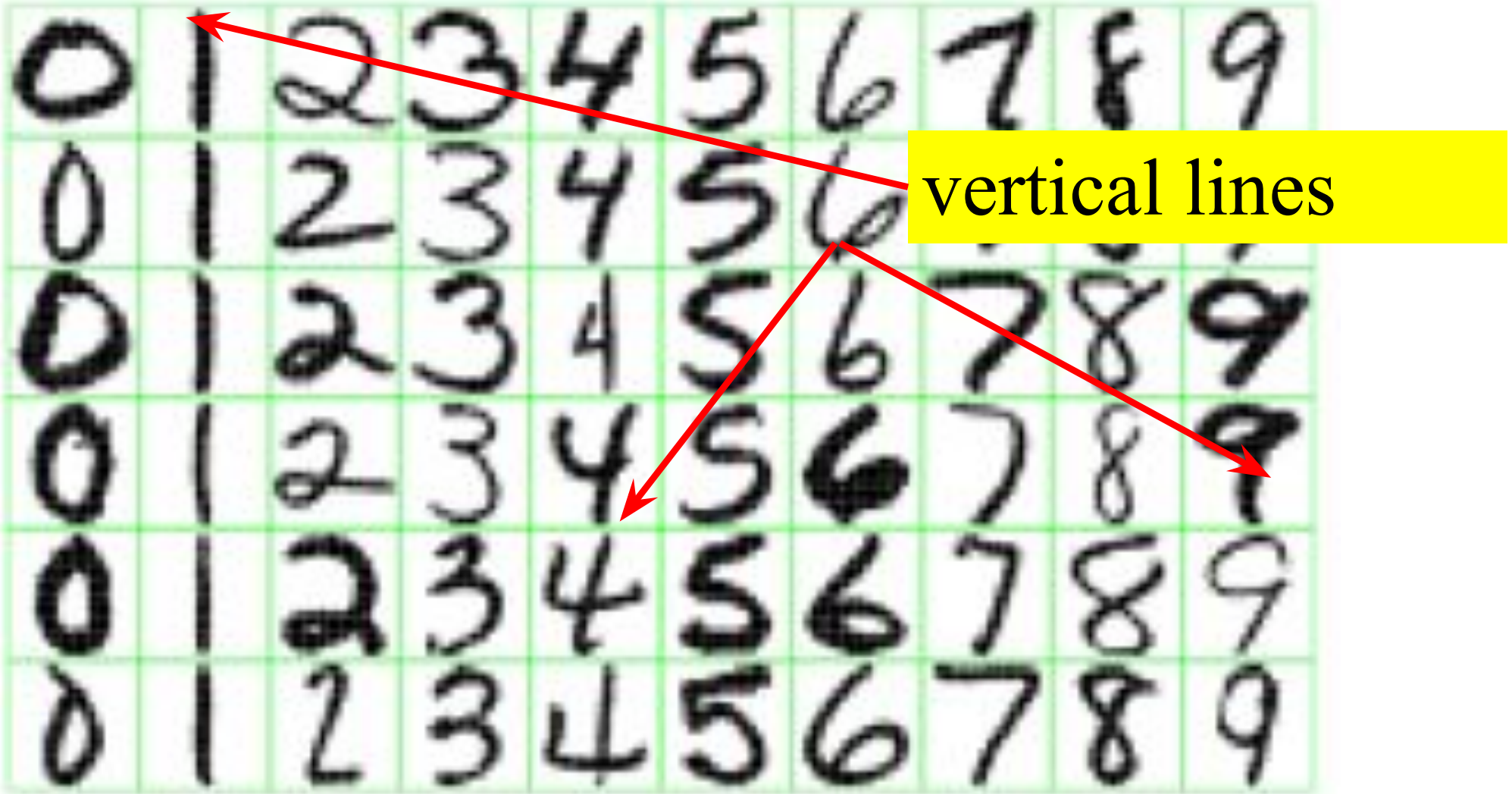


Figure 1.2: *Examples of handwritten digits from U.S. postal envelopes.*

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

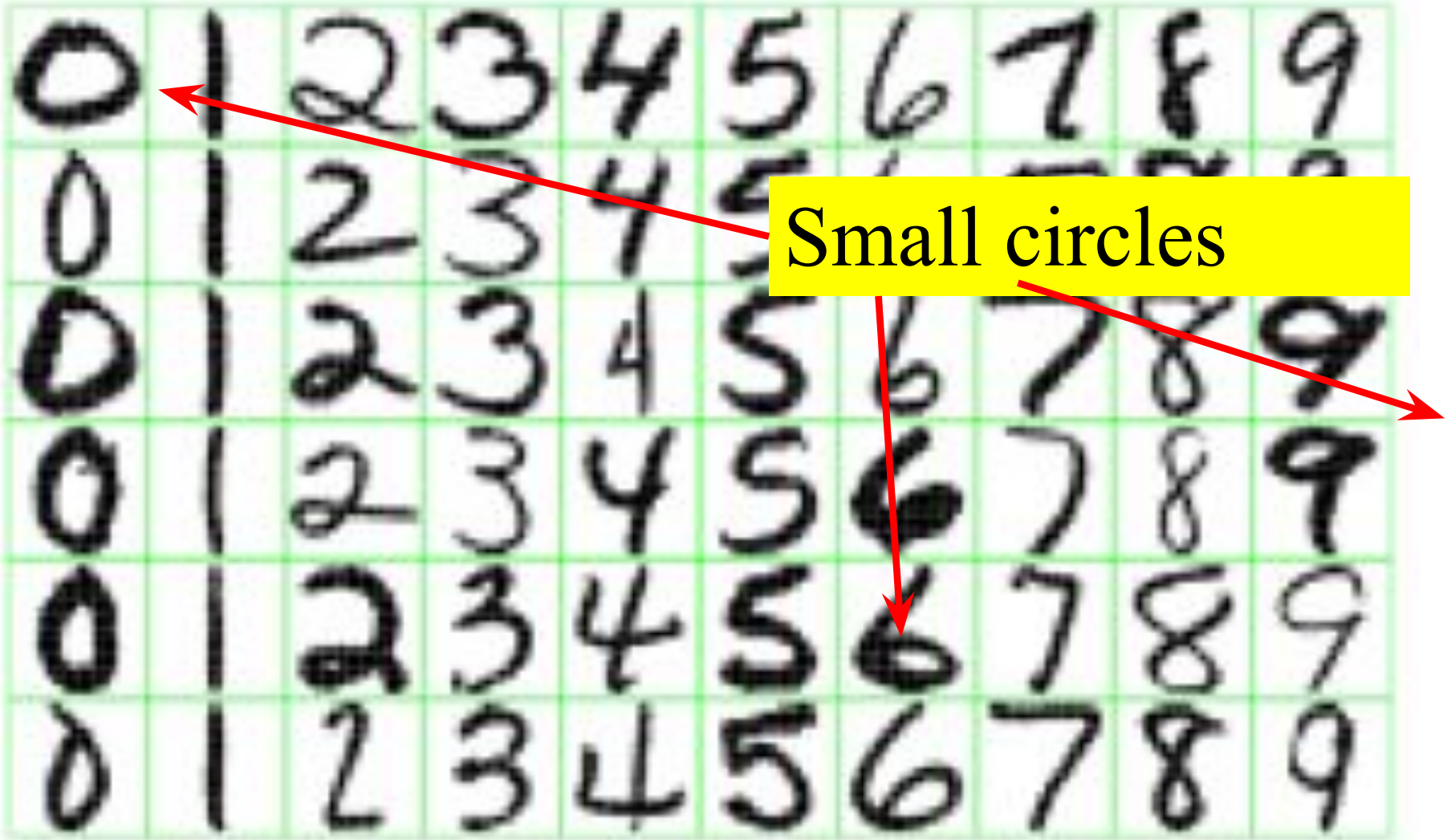
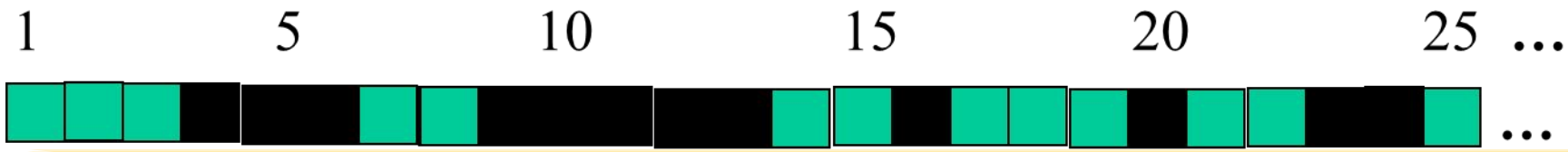


Figure 1.2: *Examples of handwritten digits from U.S. postal envelopes.*

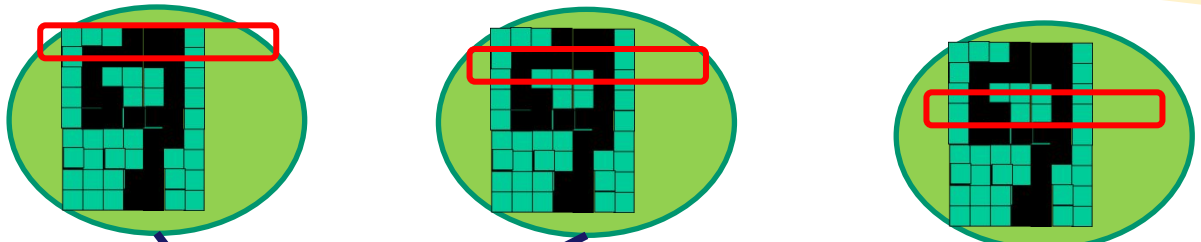
1

But what about position invariance ???
our example unit detectors were tied to specific parts of the image

successive layers can learn higher-level features …

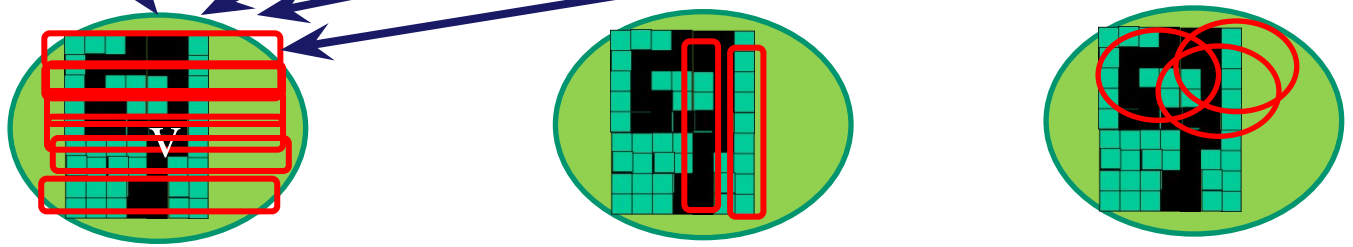


detect lines in
Specific positions



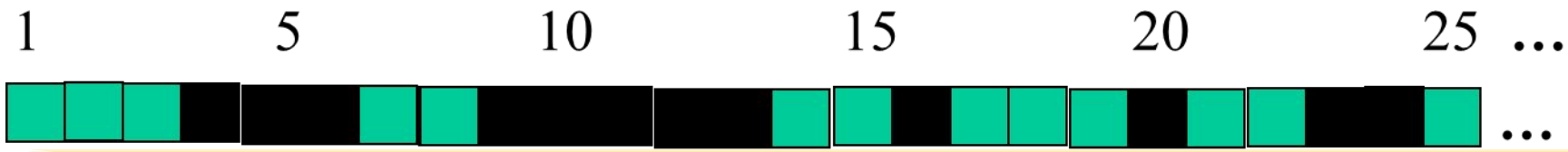
etc …

Higher level detetors
(horizontal line,
“RHS vertical lune”
“upper loop”, etc…

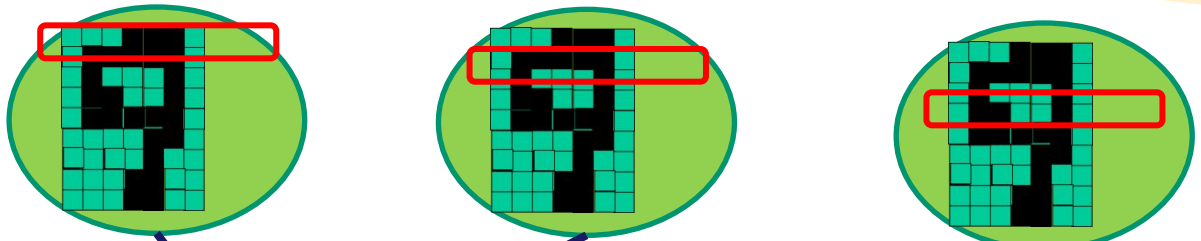


etc …

successive layers can learn higher-level features …

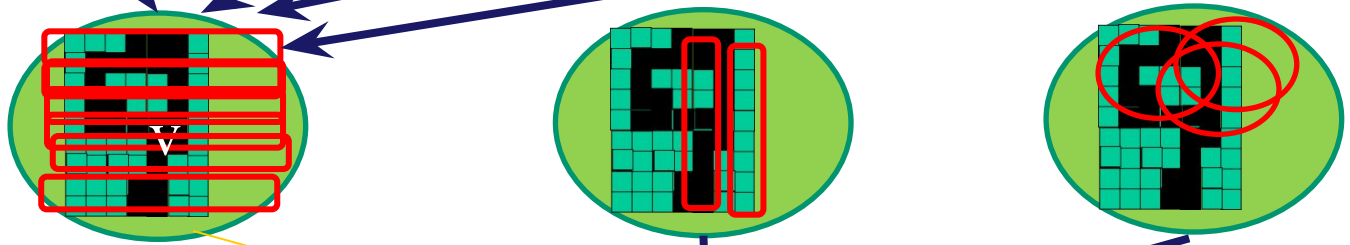


detect lines in
Specific positions



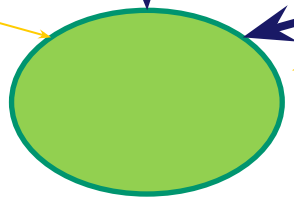
etc …

Higher level detetors
(horizontal line,
“RHS vertical lune”
“upper loop”, etc…



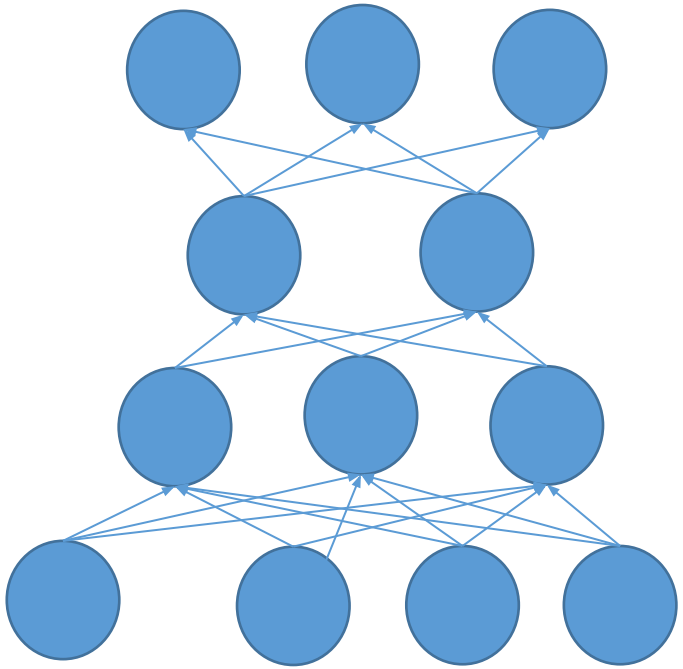
etc …

Digit



Multiple Layers of Feature Representation

- The neurons at each layer provides distinct levels of abstraction
  - The higher layer neurons model more abstract features than those at lower layers

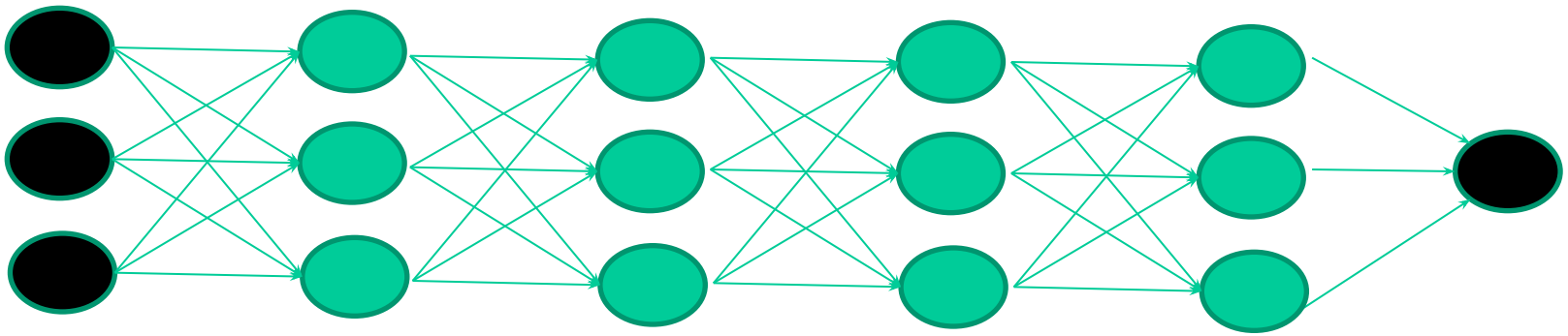


Recognize the whole object

Recognize different parts of object

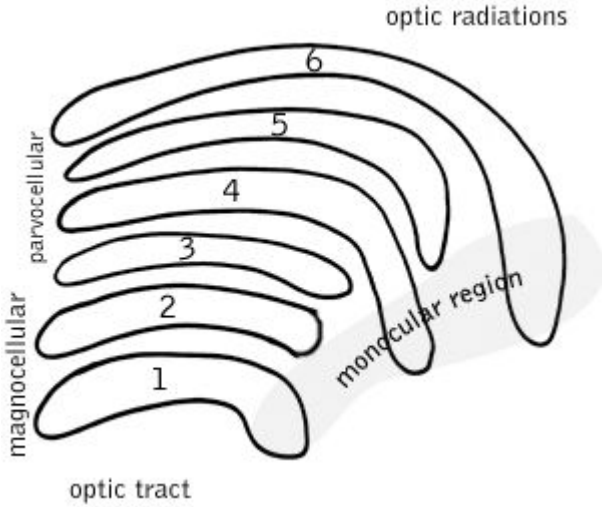
Recognize the edge points

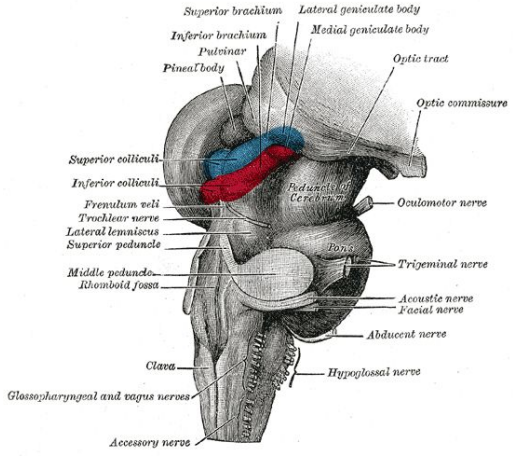
So: *multiple layers make sense*



So: *multiple layers make sense*

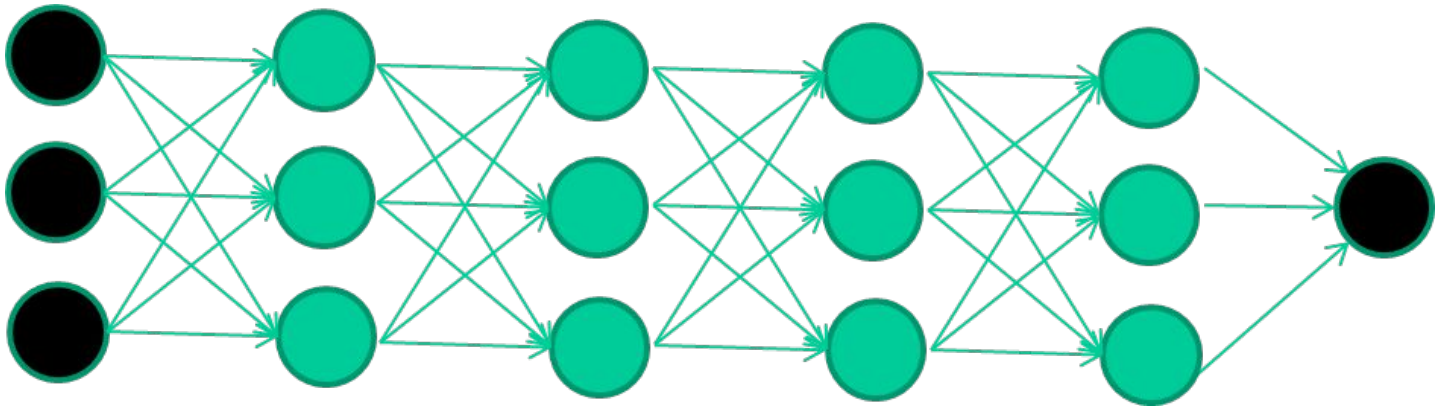
Your brain works that way



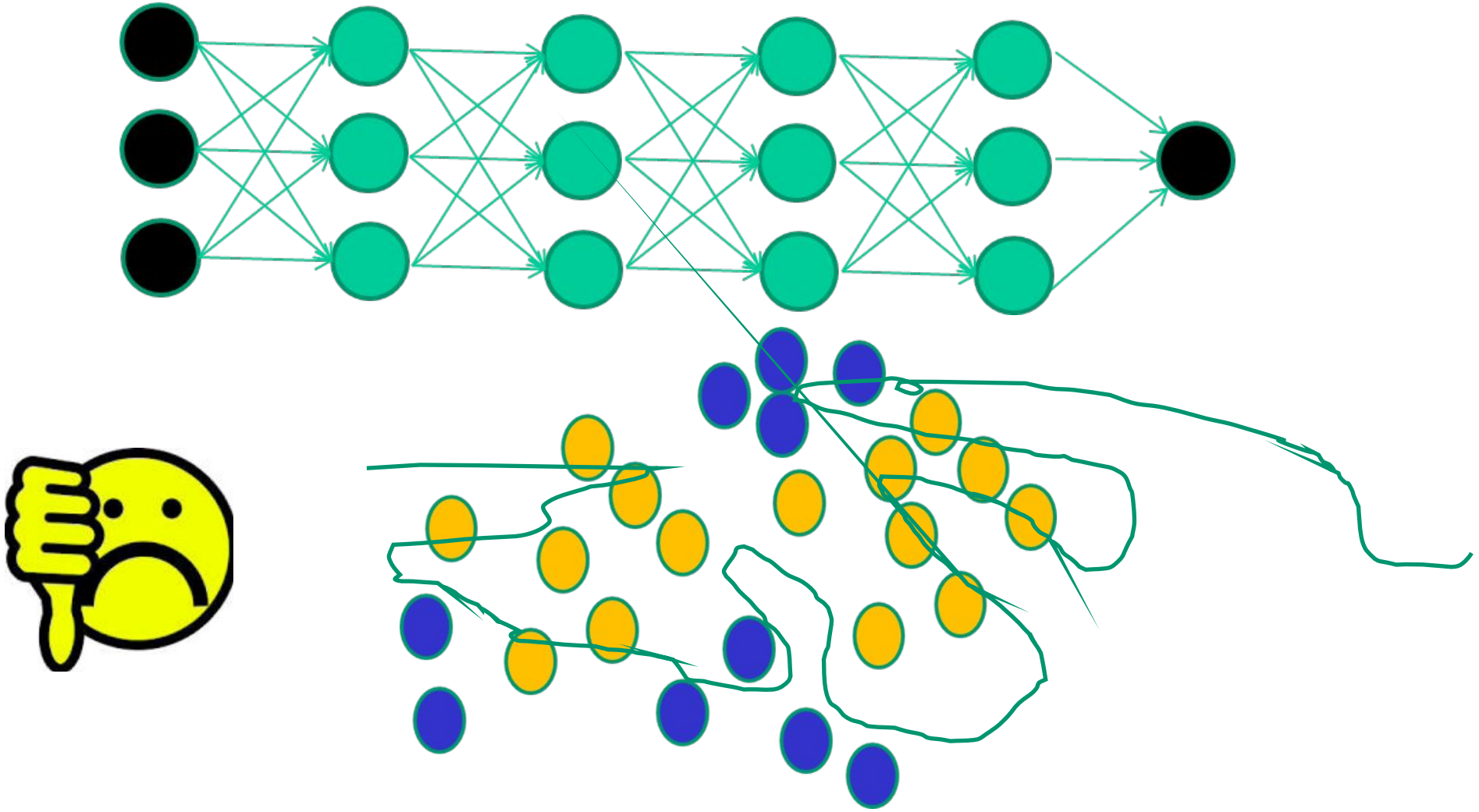


So: *multiple layers make sense*

Many-layer neural network architectures should be capable of learning the true underlying features and 'feature logic', and therefore generalise very well …

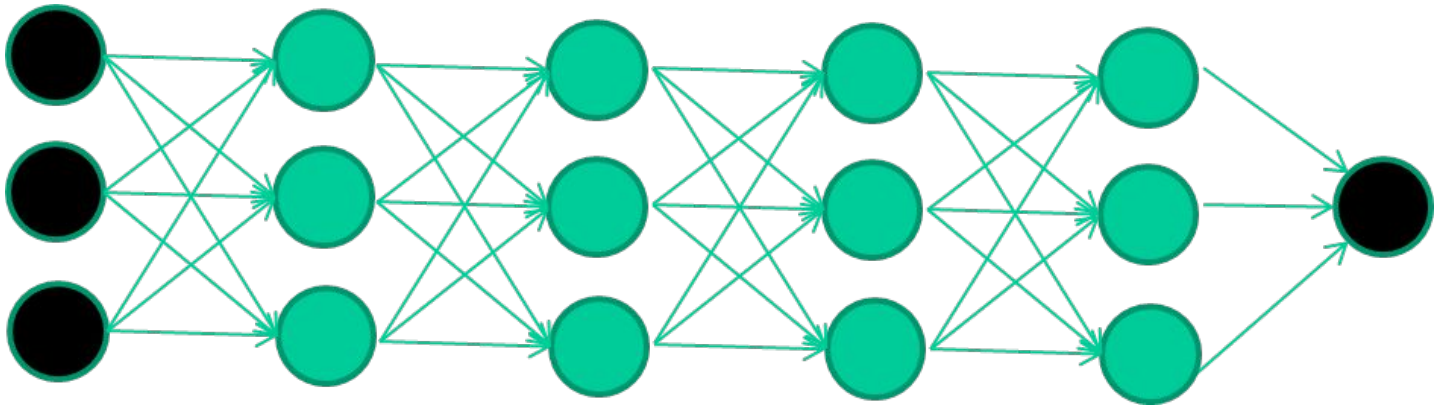


But, until very recently, our weight-learning algorithms simply did not work on multi-layer architectures

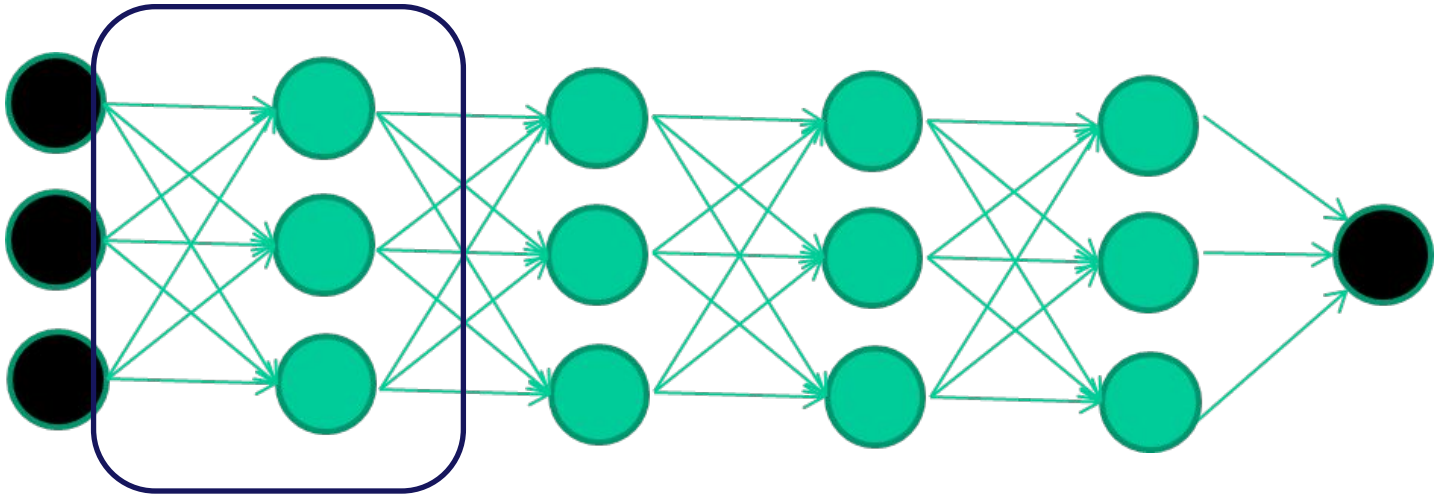


Along came deep learning …

The new way to train multi-layer NNs…

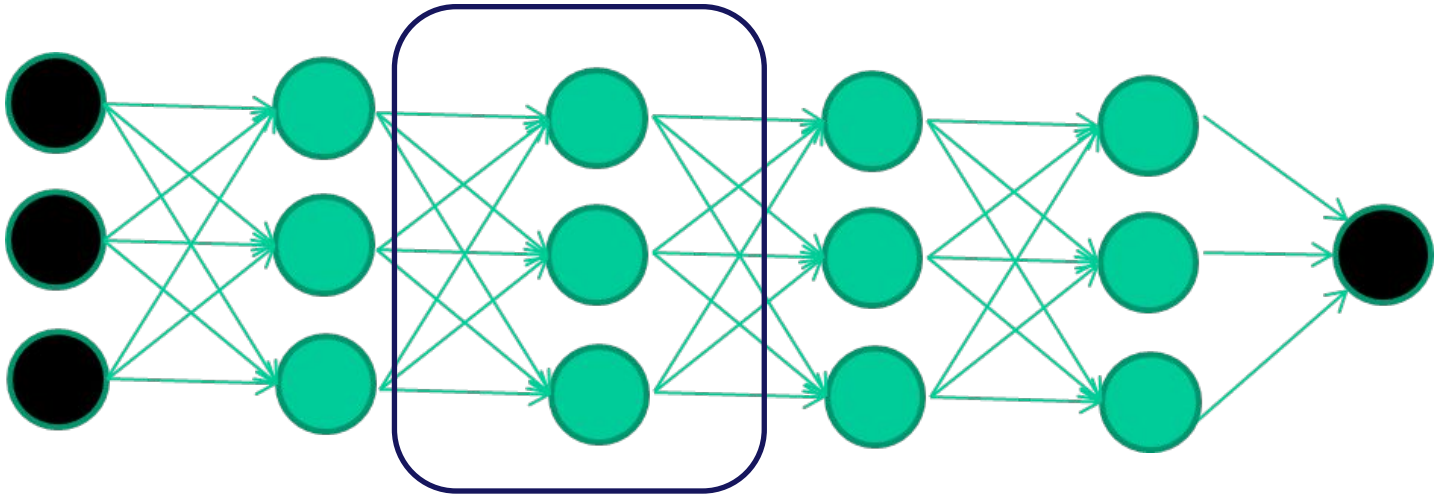


The new way to train multi-layer NNs…



Train **this** layer first

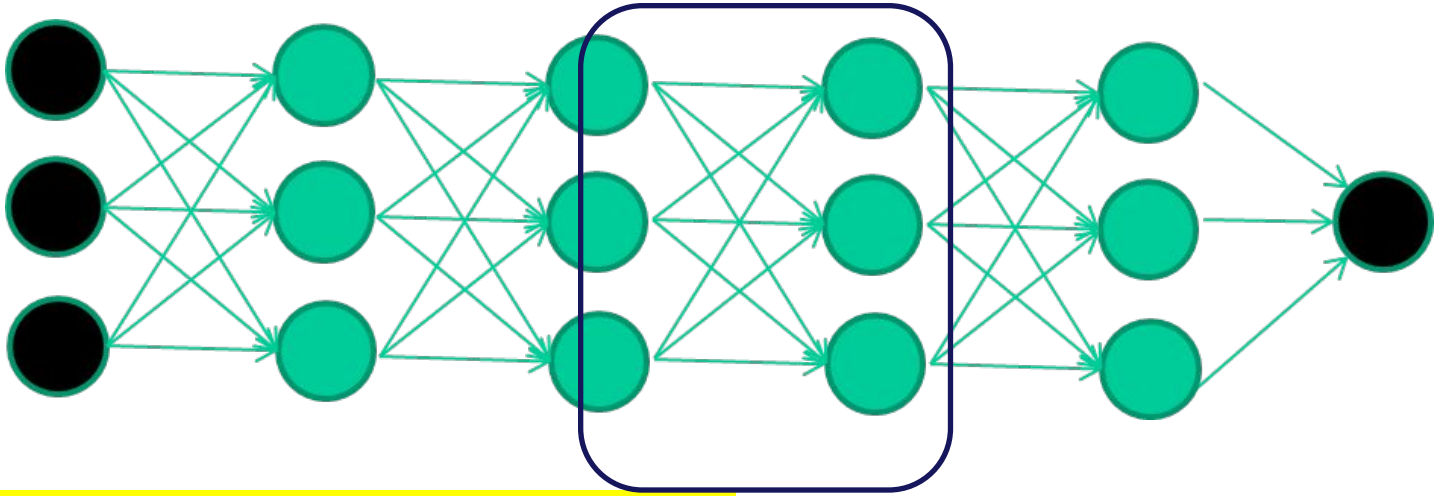
The new way to train multi-layer NNs…



Train **this** layer first

then **this** layer

The new way to train multi-layer NNs…

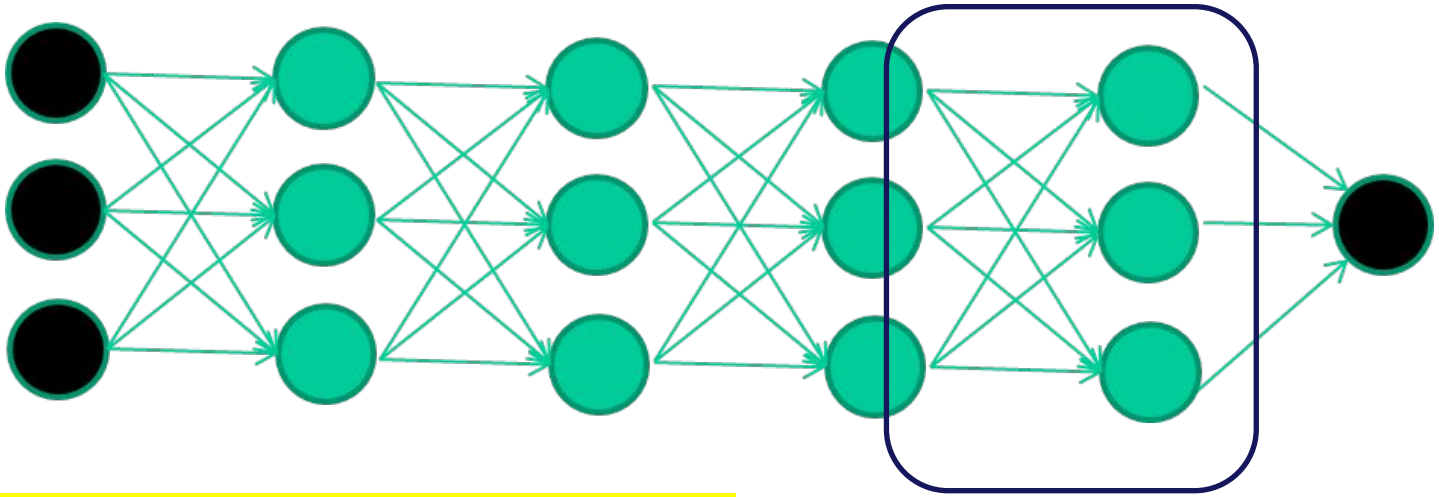


Train **this** layer first

then **this** layer

then **this** layer

The new way to train multi-layer NNs…



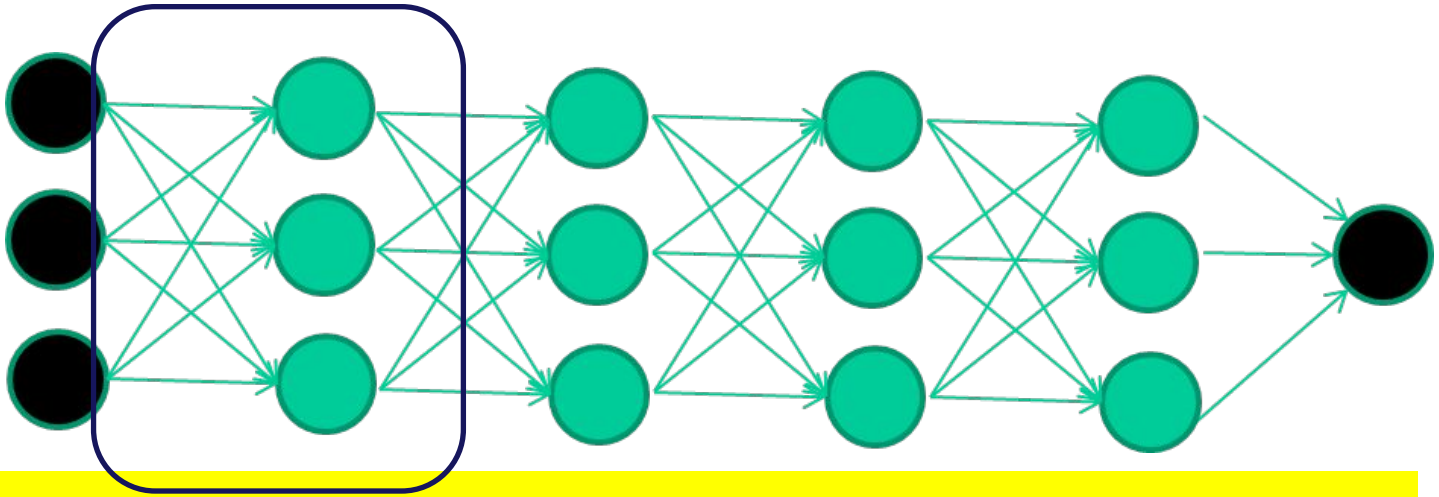
Train **this** layer first

then **this** layer

then **this** layer

then **this** layer

The new way to train multi-layer NNs…

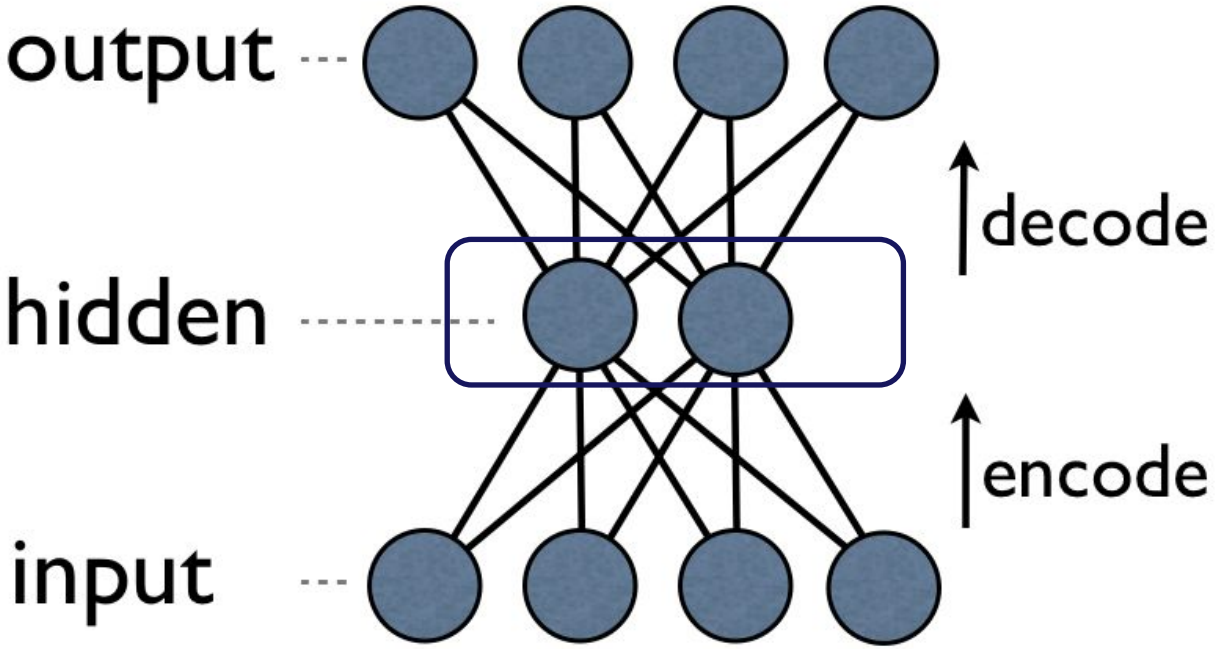


EACH of the (non-output) layers is trained to be an

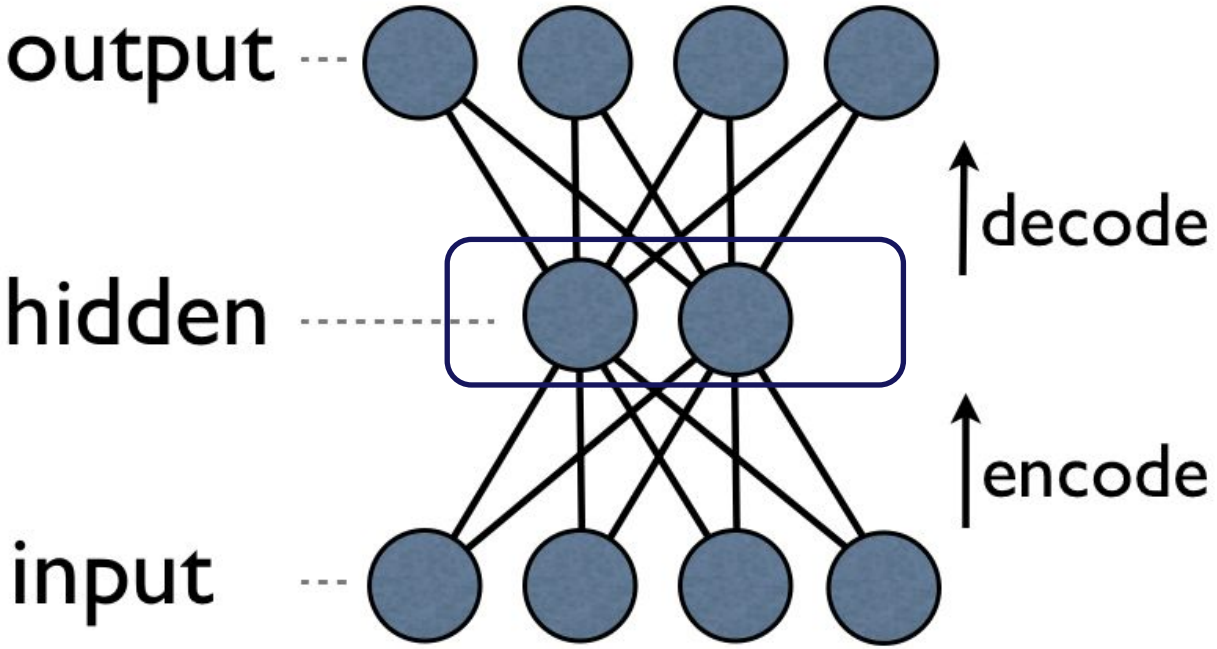
auto-encoder

Basically, it is forced to learn good features that describe what comes from the previous layer

an auto-encoder is trained, with an absolutely standard weight-adjustment algorithm to reproduce the input

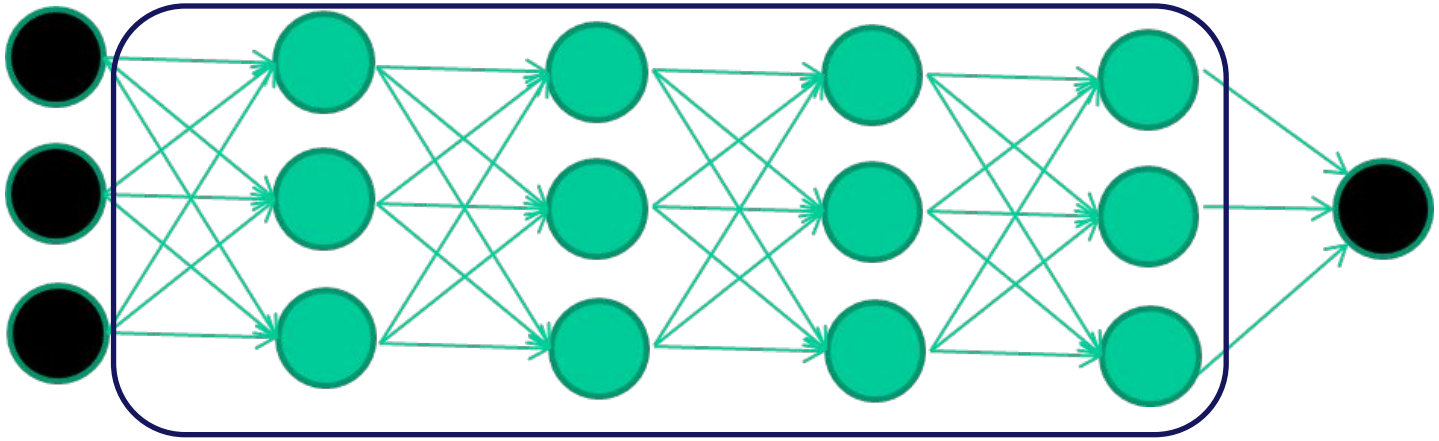


an auto-encoder is trained, with an absolutely standard weight-adjustment algorithm to reproduce the input

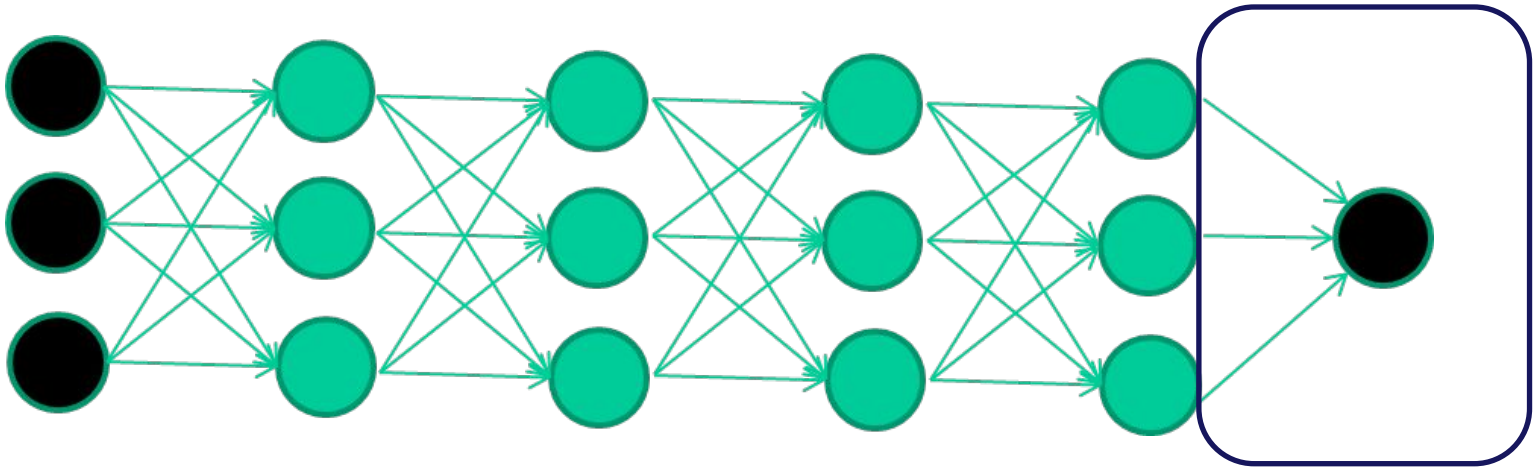


By making this happen with (many) fewer units than the inputs, this forces the 'hidden layer' units to become good feature detectors

intermediate layers are each trained to be auto encoders (or similar)

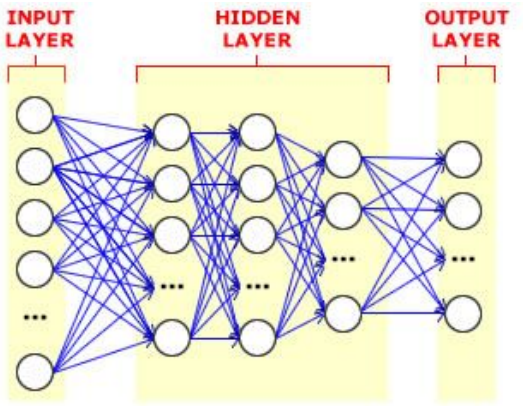


Final layer trained to predict class based on outputs from previous layers



And that's that

- That's the basic idea
- There are many many types of deep learning,
- different kinds of autoencoder, variations on architectures and training algorithms, etc…
- Very fast growing area …



History of the Field

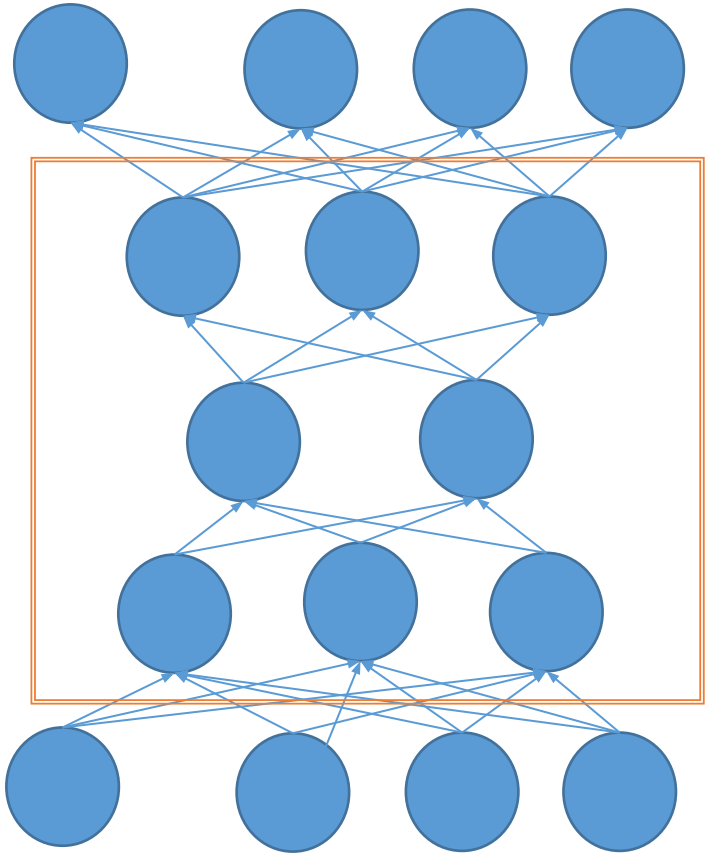
- Before 2006, no successful results were reported on training deep architectures
  - Typically with three/four layers of neural network (one or two hidden layers), the performance saturates.
  - More layers yield poorer performance.
  - Exception: Convolutional Neural Networks (CNNs), LeCun, 1998.
    https://www.youtube.com/watch?v=lxYj1Mpoiy4&feature=youtu.be
- Breakthrough: Deep Belief Networks
  - Stacked set of Restricted Boltzmann Machines

Vanishing Gradient

- Problem for gradient-based learning methods that use a sigmoid activation function.
- Gradient of the output with respect to parameters in the early layers becomes very small.
- Even a large change in the early parameters doesn’t have much effect on the output.
- Changing to a Rectified Linear Unit (max(0,x)) can fix this problem
- Cross entropy (log) loss functions can improve on L2 loss

Common Deep Network Architectures

- Deep Belief Network: Multiple Layers of Restricted Boltzman Machines
  - Effective training algorithms
- Autoencoders
  - Recap: one hidden layer autoencoder
    - May only capture simple feature structures
    - Corner points, interest points, edge points
  - Multiple layers: stacked autoencoders
    - Capture more complex feature structures



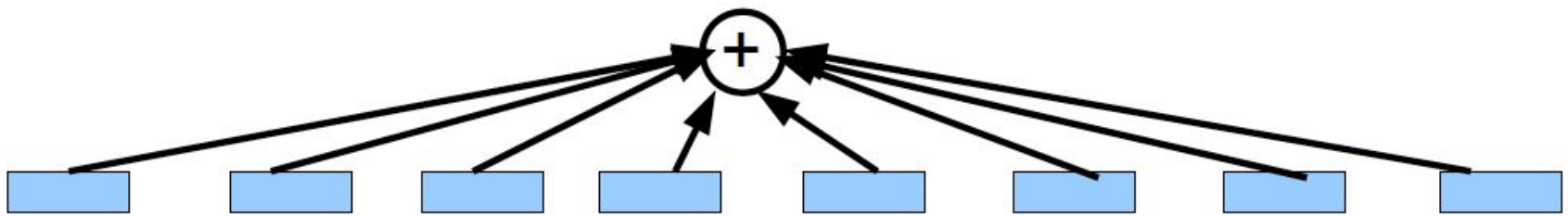
Advantages of Deep Networks

- Main argument:
  - Shallow structures may oversimplify the complexity underlying a function mapping input feature vectors to target output variables
  - Deep structures are more adequate to simulate these complex functions
    - E.g., our vision/acoustic systems
- Reason: a shallow structure may require exponentially more neurons to model a function that can be represented more compactly by a deeper structure.

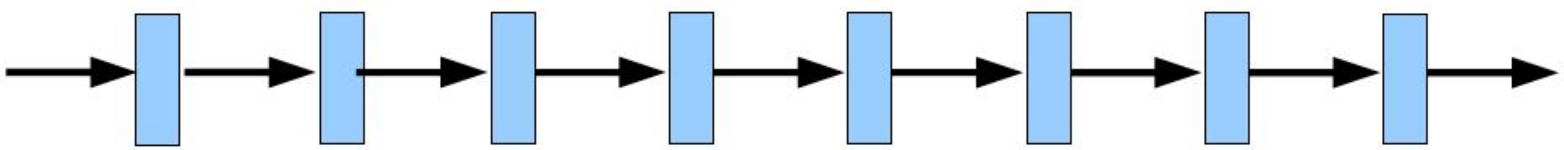
Breadth vs. Depth

Given a dictionary of simple non-linear functions: $g_1, \ldots, g_n$

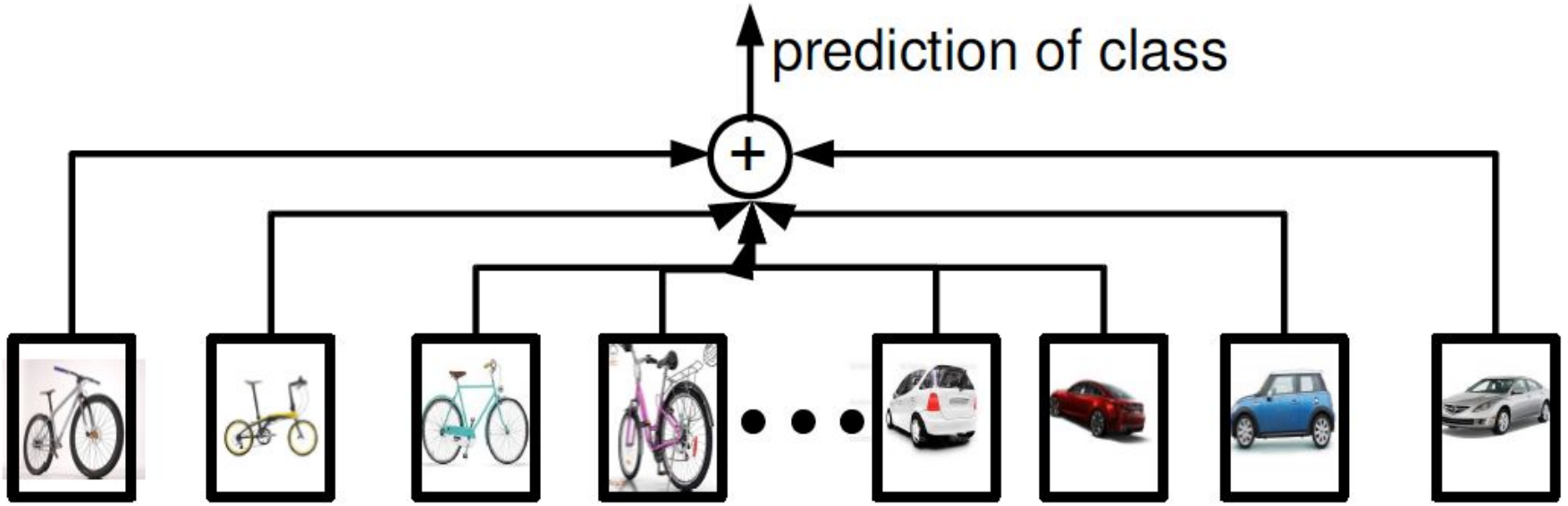
Proposal #1: linear combination $f(x) \approx \sum_j g_j$



Proposal #2: composition $f(x) \approx g_1(g_2(\ldots g_n(x)\ldots))$



Combination: Broad & Shallow

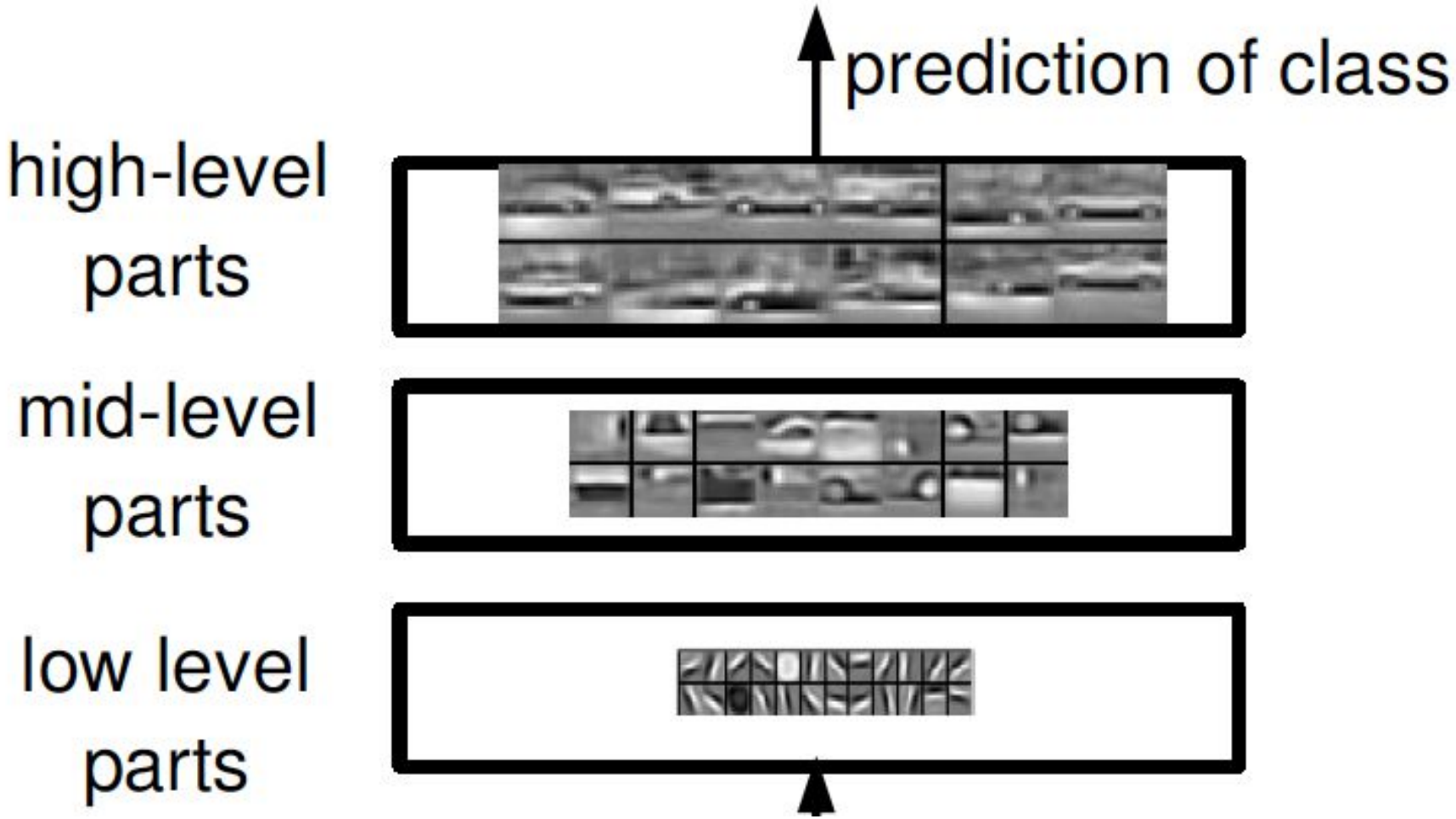


Linear combination of classifiers (e.g., templates)
Architecture is like boosting or kernel learning

What's bad about this?

[Diagram: Ranzato]

Composition: Narrow & Deep



Parts are shared/reused

Can be *exponentially* more efficient!

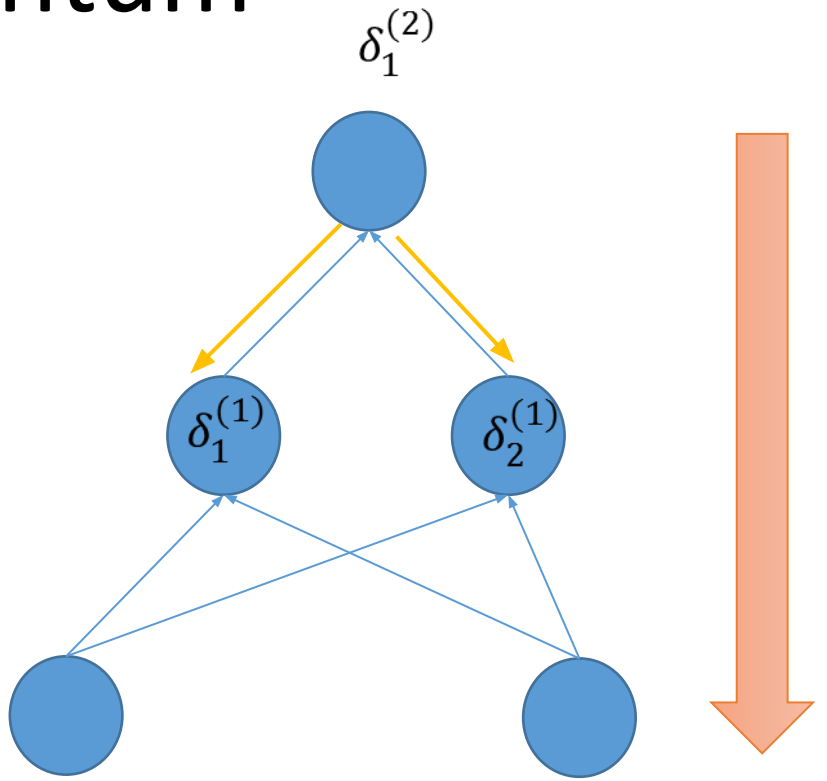
[Diagram: Ranzato]

Challenges

- More attentions must be paid to tune a deep structure
  - Shallow structure often has a complex objective function
    - Linear model/kernelized nonlinear model
    - It has globally optimal solution in training phase, which can be found by many well-developed optimization algorithms
  - The objective function associated with deep structure is often nonconvex
    - Many local optimum, and special attentions must be paid to avoid being trapped in bad local optimum
    - Finding global optimum is impossible

Train Deep Networks: Momentum

- For each iteration t
  - For each weight $w_{ki}^{(2)}$: $\Delta w_{ki}^{(2)}(t) \equiv \frac{\partial L}{\partial w_{ki}^{(2)}}(t)$
    - Update $w_{ki}^{(2)}(t) \leftarrow w_{ki}^{(2)}(t) - \alpha \Delta w_{ki}^{(2)}(t)$
  - For each weight $w_{jn}^{(1)}$: $\Delta w_{jn}^{(1)} \equiv \frac{\partial L}{\partial w_{jn}^{(1)}}(\mathrm{t})$
    - Update $w_{jn}^{(1)}(t) \leftarrow w_{jn}^{(1)}(t) - \alpha \Delta w_{jn}^{(1)}(t)$

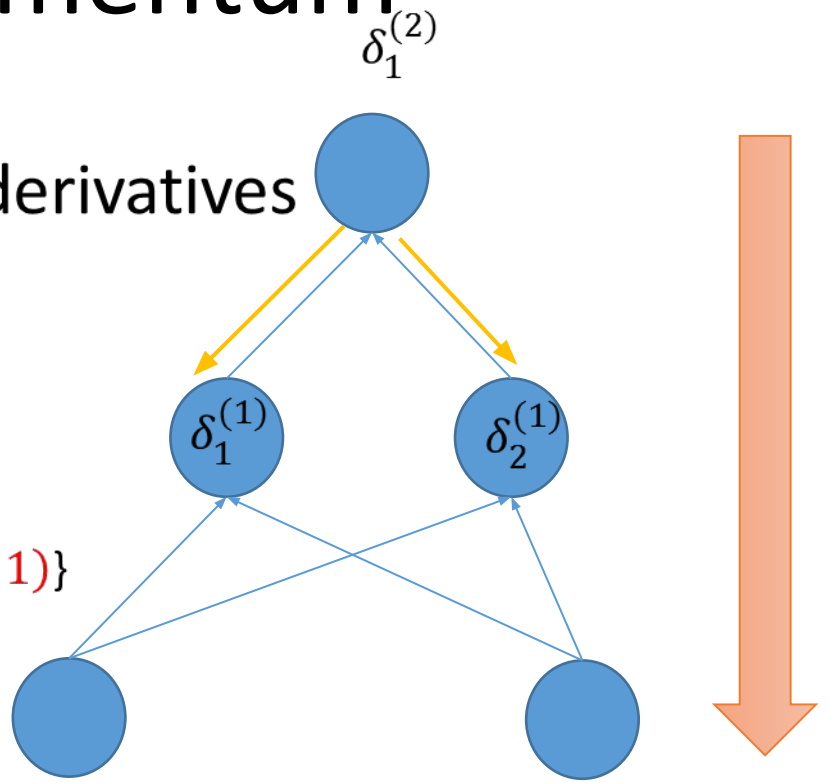


Note: $\alpha > 0$ is a learning rate which controls the step size for each weight update

Training Deep Networks: Momentum

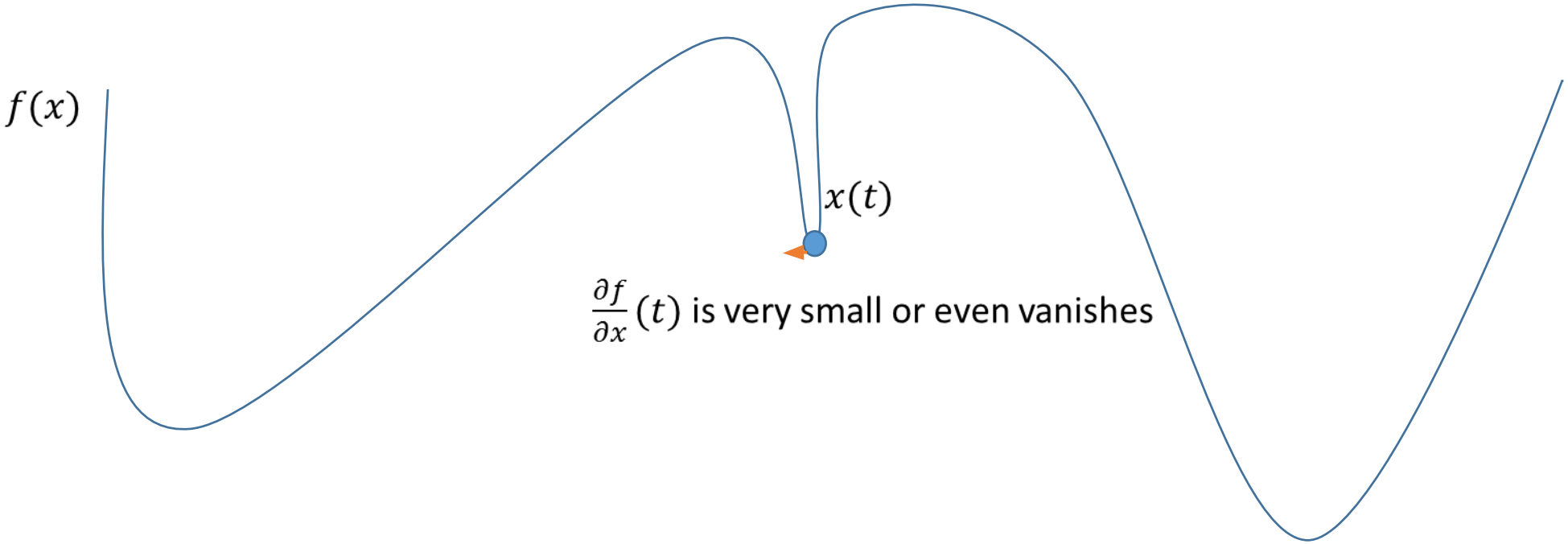
- Idea: exponentially weighed sum of recent derivatives
- For each iteration t
  - For each weight $w_{ki}^{(2)}$: $\Delta w_{ki}^{(2)}(t) \equiv \frac{\partial L}{\partial w_{ki}^{(2)}}(t)$
    - Update $w_{ki}^{(2)}(t) \leftarrow w_{ki}^{(2)}(t) - \alpha\{\Delta w_{ki}^{(2)}(t) + \epsilon \Delta w_{ki}^{(2)}(t-1)\}$
  - For each weight $w_{jn}^{(1)}$: $\Delta w_{jn}^{(1)} \equiv \frac{\partial L}{\partial w_{jn}^{(1)}}(\mathrm{t})$
    - Update $w_{jn}^{(1)}(t) \leftarrow w_{jn}^{(1)}(t) - \alpha\{\Delta w_{jn}^{(1)}(t) + \epsilon \Delta w_{jn}^{(1)}(t-1)\}$

ϵ is typically set to [0.7 0.95].



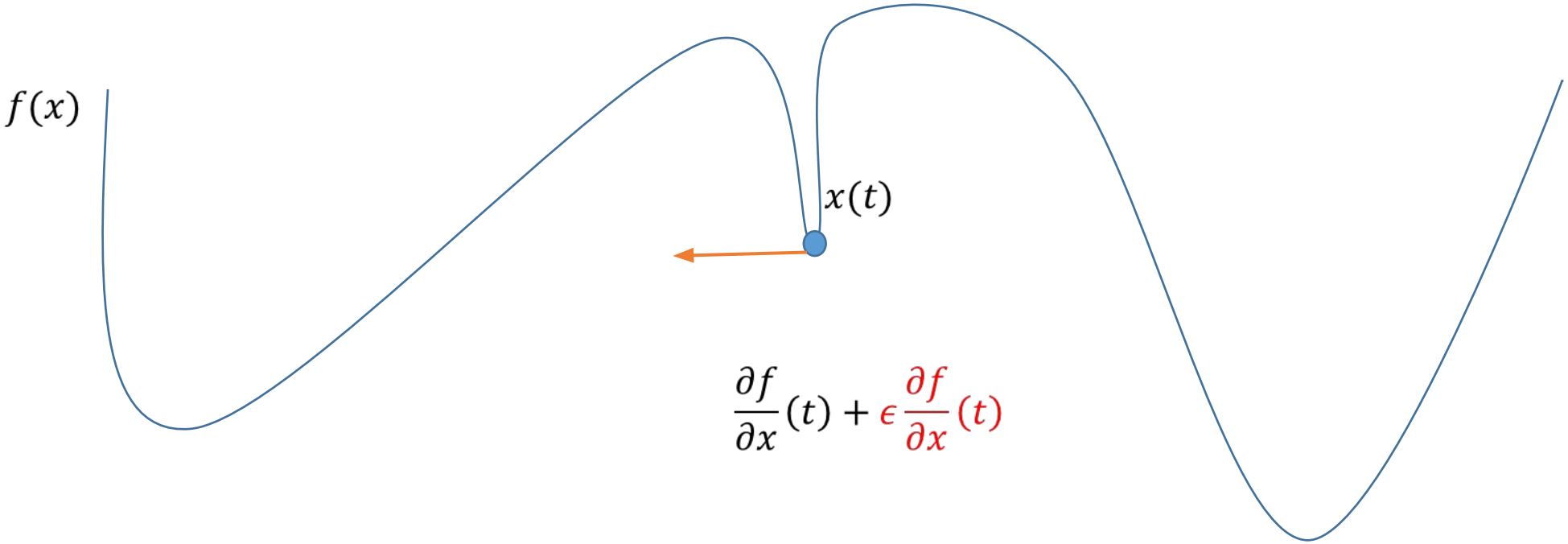
Momentum

- The search for the minimum point can be trapped in a steep local basin.



Momentum

Can give a extra force to push the current search for the optimal point from the trap of local minimum



Adaptive Learning Rate

- Too large learning rate
  - Cause oscillation in searching for the minimal point
- Too slow learning rate
  - Too slow convergence to the minimal point
- Adaptive learning rate
  - At the beginning, the learning rate can be large when the current point is far from the optimal point
  - Gradually, the learning rate will decay as time goes by.

Adaptive Learning Rate

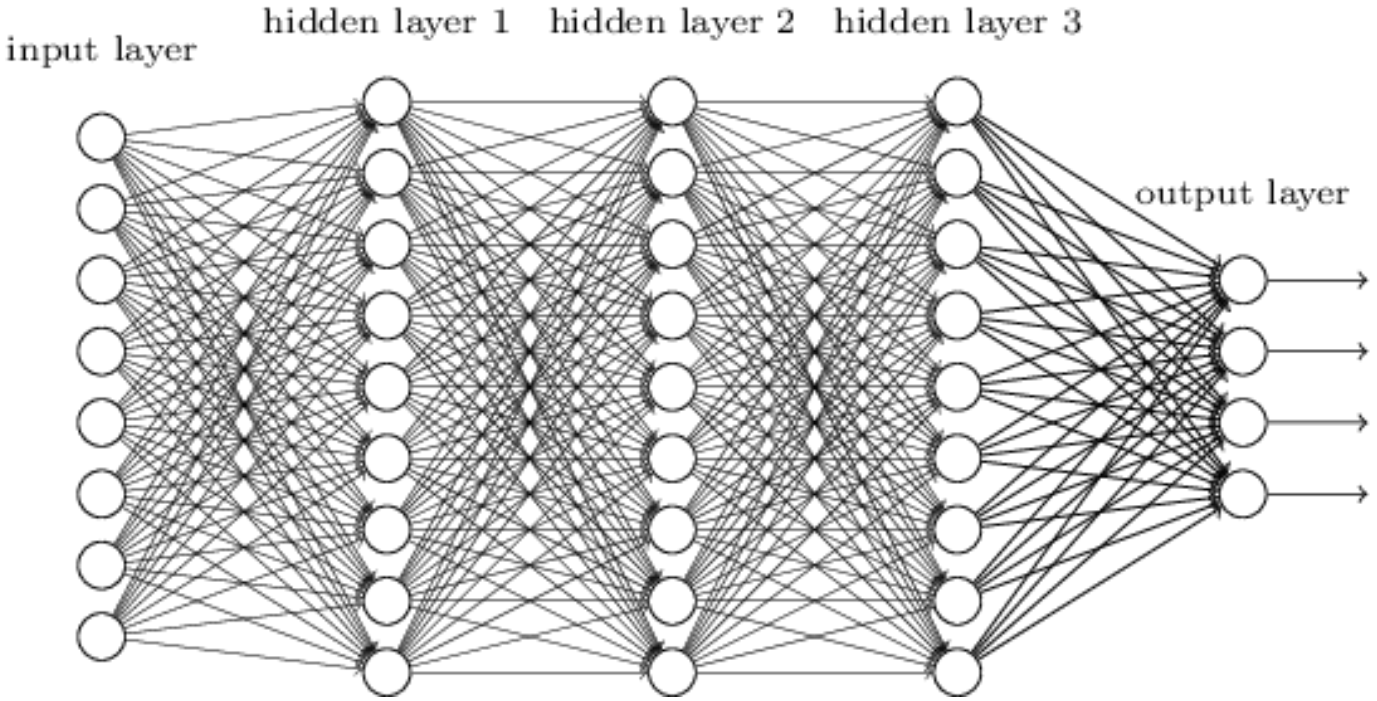
- Should not be too large or too small

$$\text{Annealing rate: } \alpha(t) = \frac{\alpha(0)}{1+\frac{t}{T}}$$

  - $\alpha(t)$ will eventually go to zero, but at the beginning it is almost a constant.

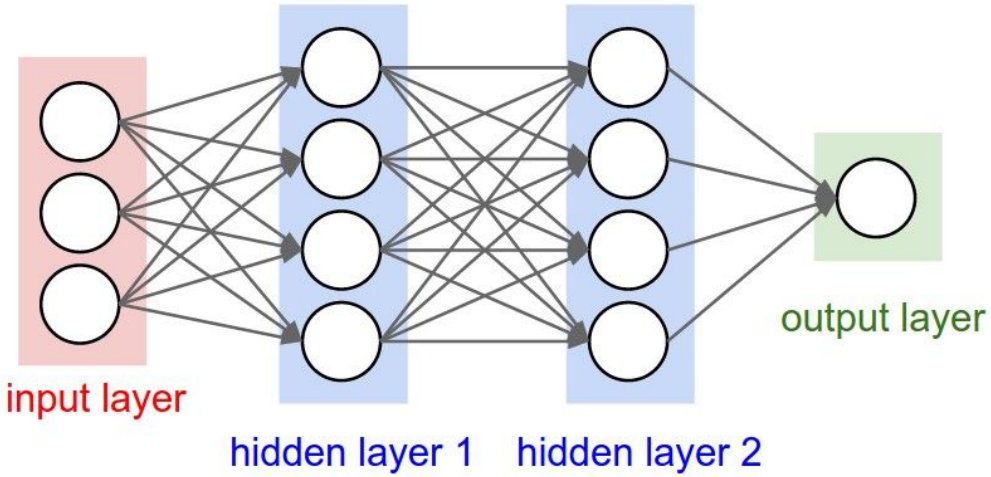
Convolutional Neural Networks

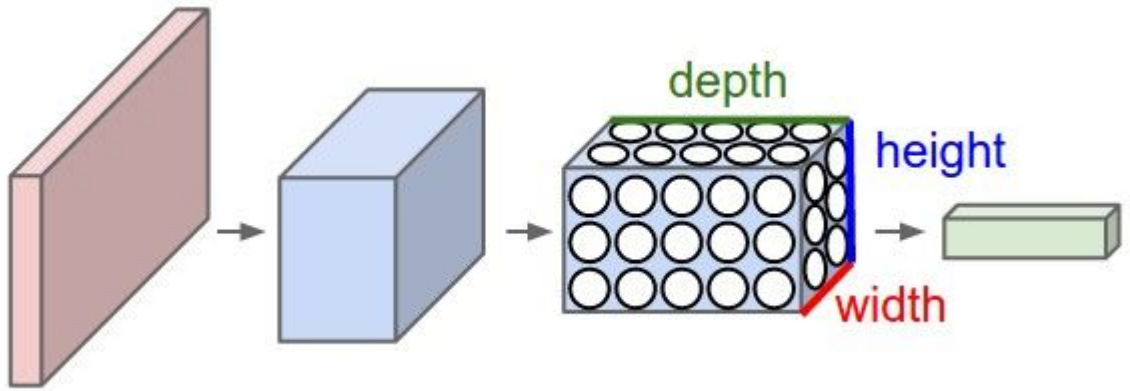
- This is a NN architecture that has performed well on images.
- From this fully connected model, do we really need all the edges?
- Can some of these be shared?



Two layers with N, M neurons will have NM parameters for the connection weights between them

Standard vs. ConvNet



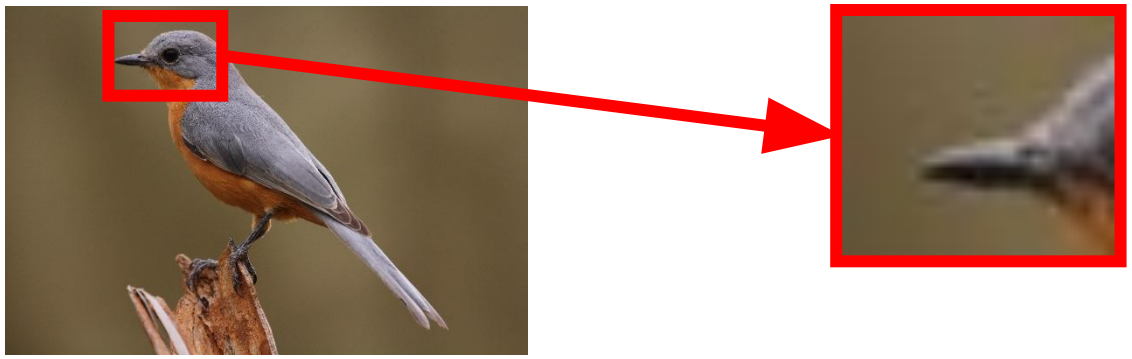


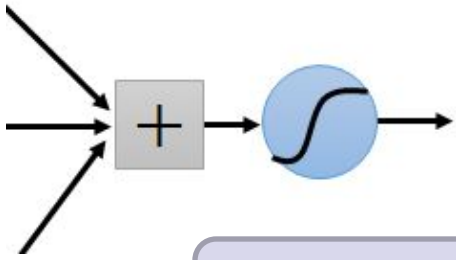
A ConvNet is made up of Layers. Every Layer has a simple API: It transforms an input 3D volume to an output 3D volume with some differentiable function that may or may not have parameters.

Consider learning an image:

- Some patterns are much smaller than the whole image

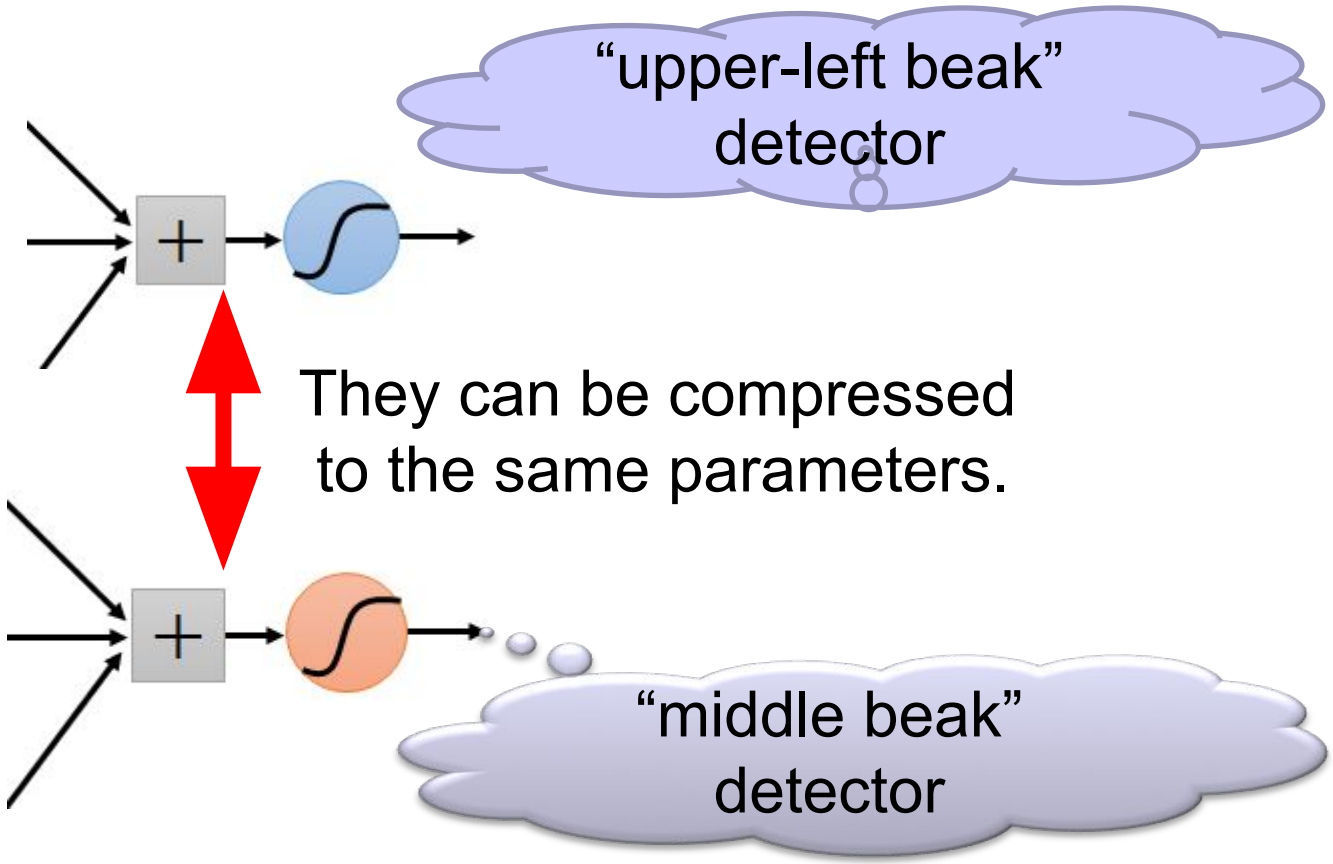
Can represent a small region with fewer parameters





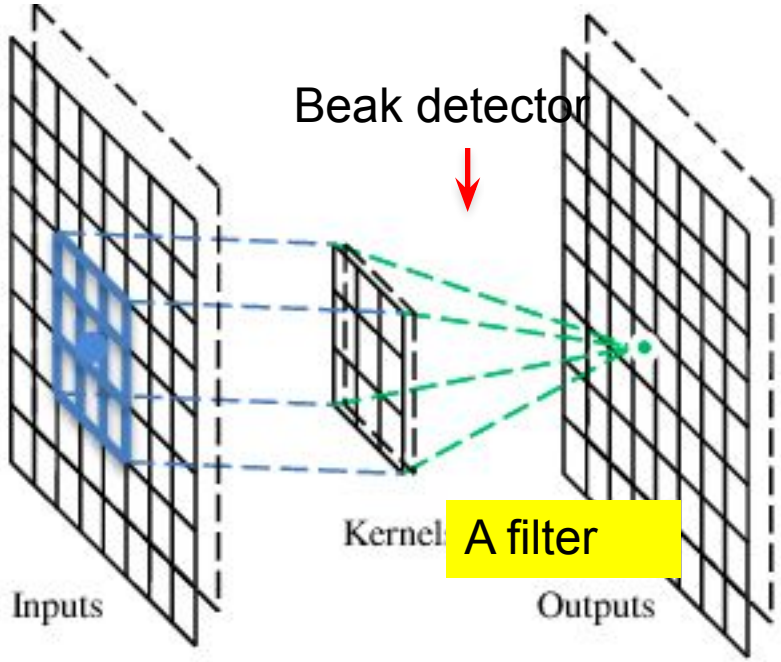
"beak" detector

Same pattern appears in different places:
They can be compressed!
What about training a lot of such “small” detectors and each detector must “move around”.



A convolutional layer

A CNN is a neural network with some convolutional layers (and some other layers). A convolutional layer has a number of filters that does convolutional operation.



Convolution

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

6 x 6 image

These are the network parameters to be learned.

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

Filter 1

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

Filter 2

⋮ ⋮

Each filter detects a small pattern (3 x 3).

Convolution

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

Filter 1

stride=1

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

6 x 6 image

Dot product →

3 -1

Convolution

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

Filter 1

If stride=2

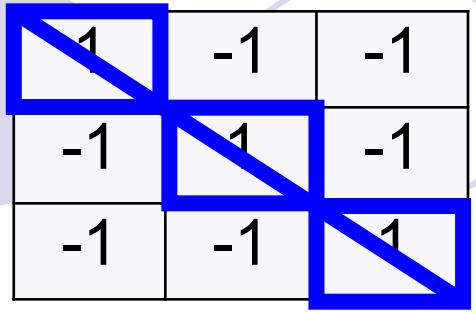
1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

6 x 6 image

3

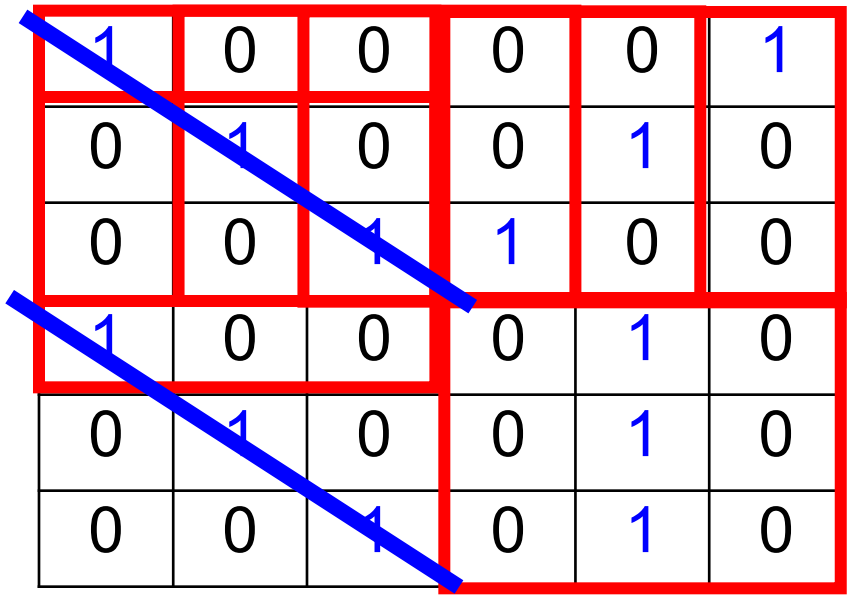
-3

Convolution

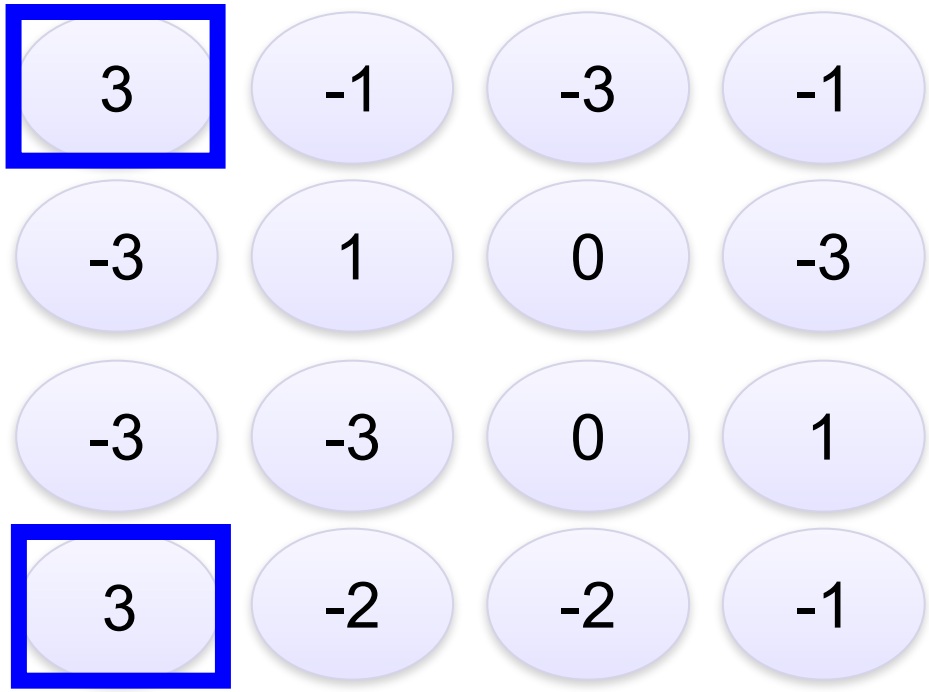


Filter 1

stride=1



6 x 6 image



Convolution

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

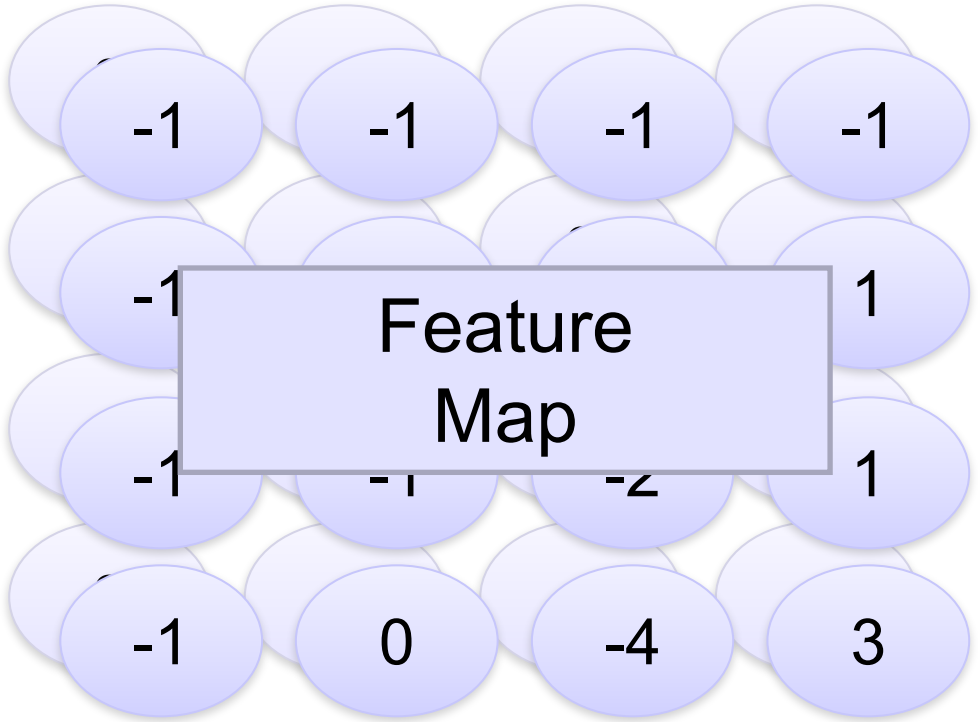
Filter 2

stride=1

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

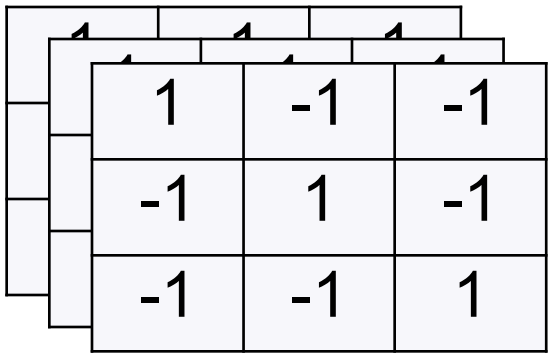
6 x 6 image

Repeat this for each filter

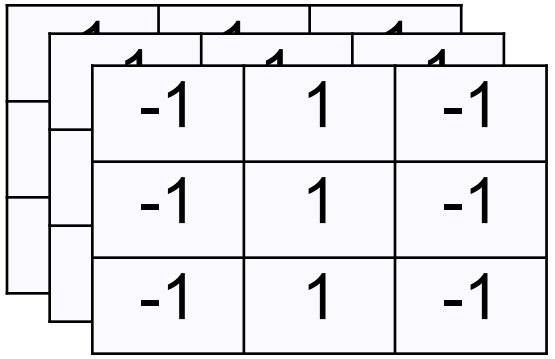


Two 4 x 4 images
Forming 2 x 4 x 4 matrix

Color image: RGB 3 channels

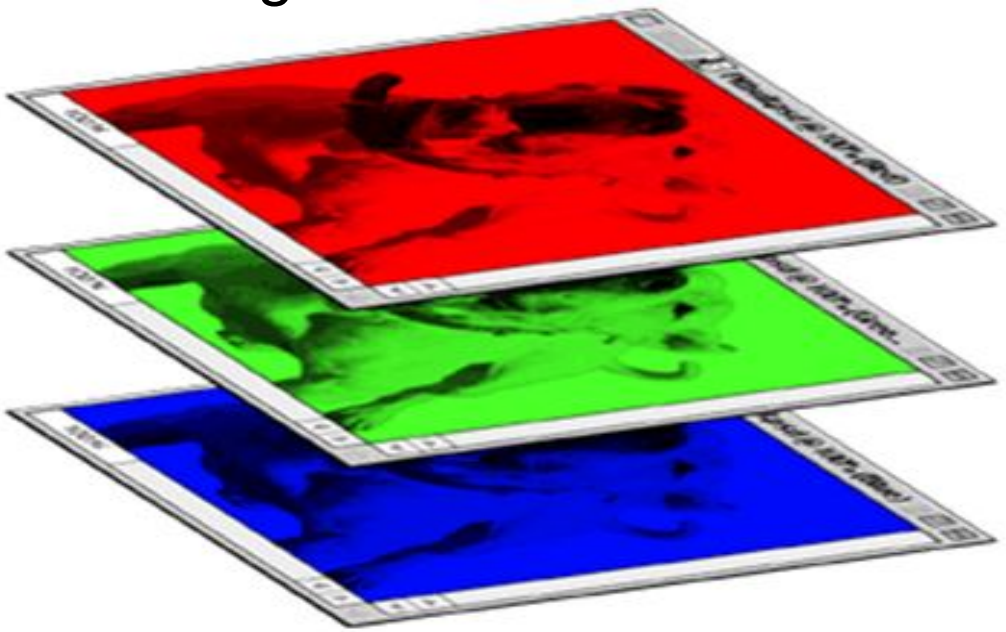


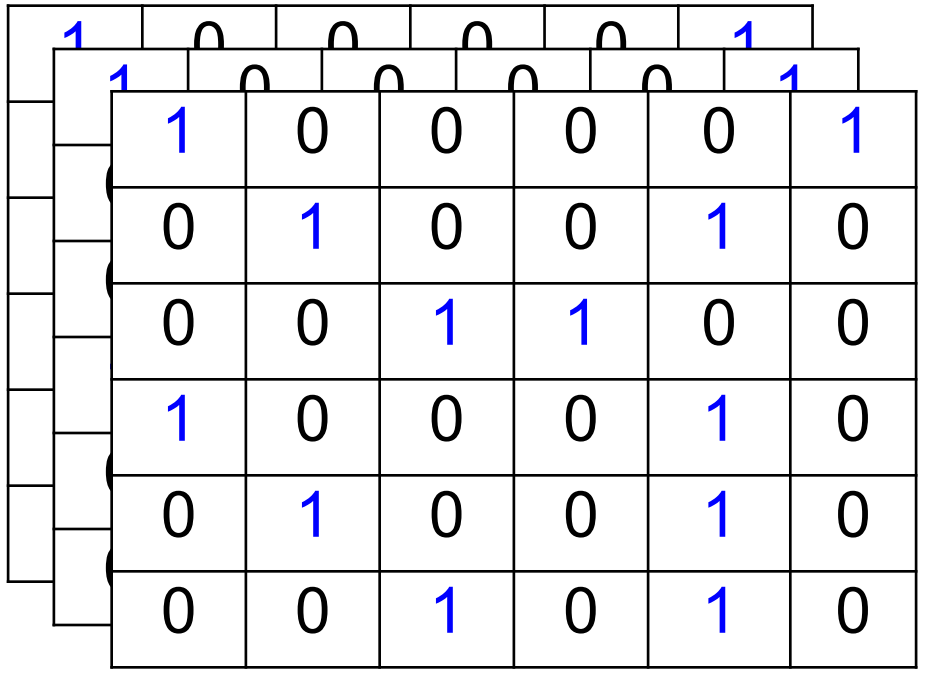
Filter 1



Filter 2

Color image





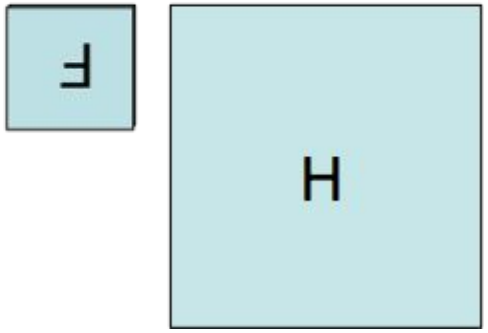
Convolution in Computer Vision

Convolution was commonly used in computer vision to detect features in images.

$$G[i,j] = \sum_{u=-k}^{k} \sum_{v=-k}^{k} H[u,v]F[i-u,j-v]$$

$$G = H \star F$$

Notation for convolution operator



In CNNs, the filter is learned rather than applying a standard one (Gaussian, edge detection)

Convolution vs. Fully Connected

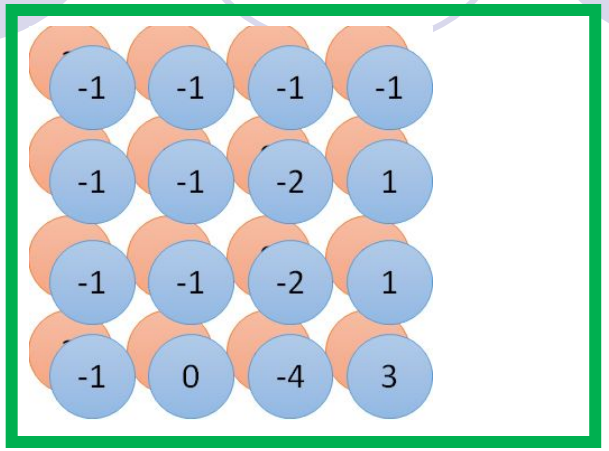
1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

image

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

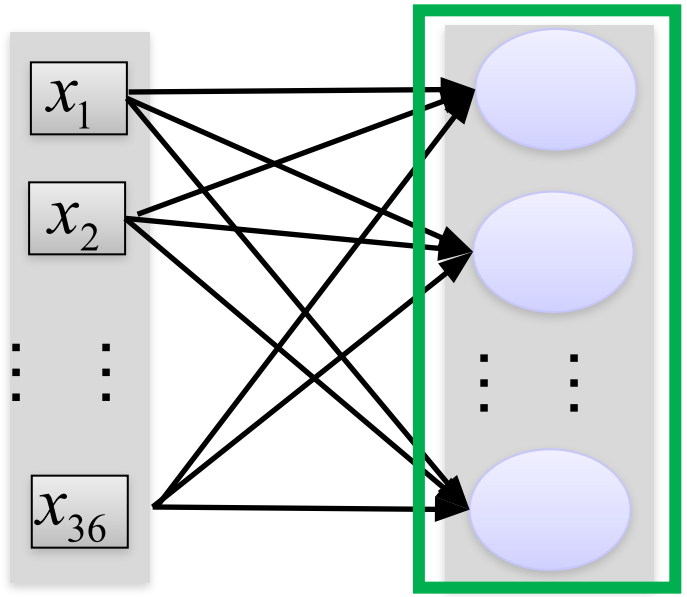
-1	1	-1
-1	1	-1
-1	1	-1

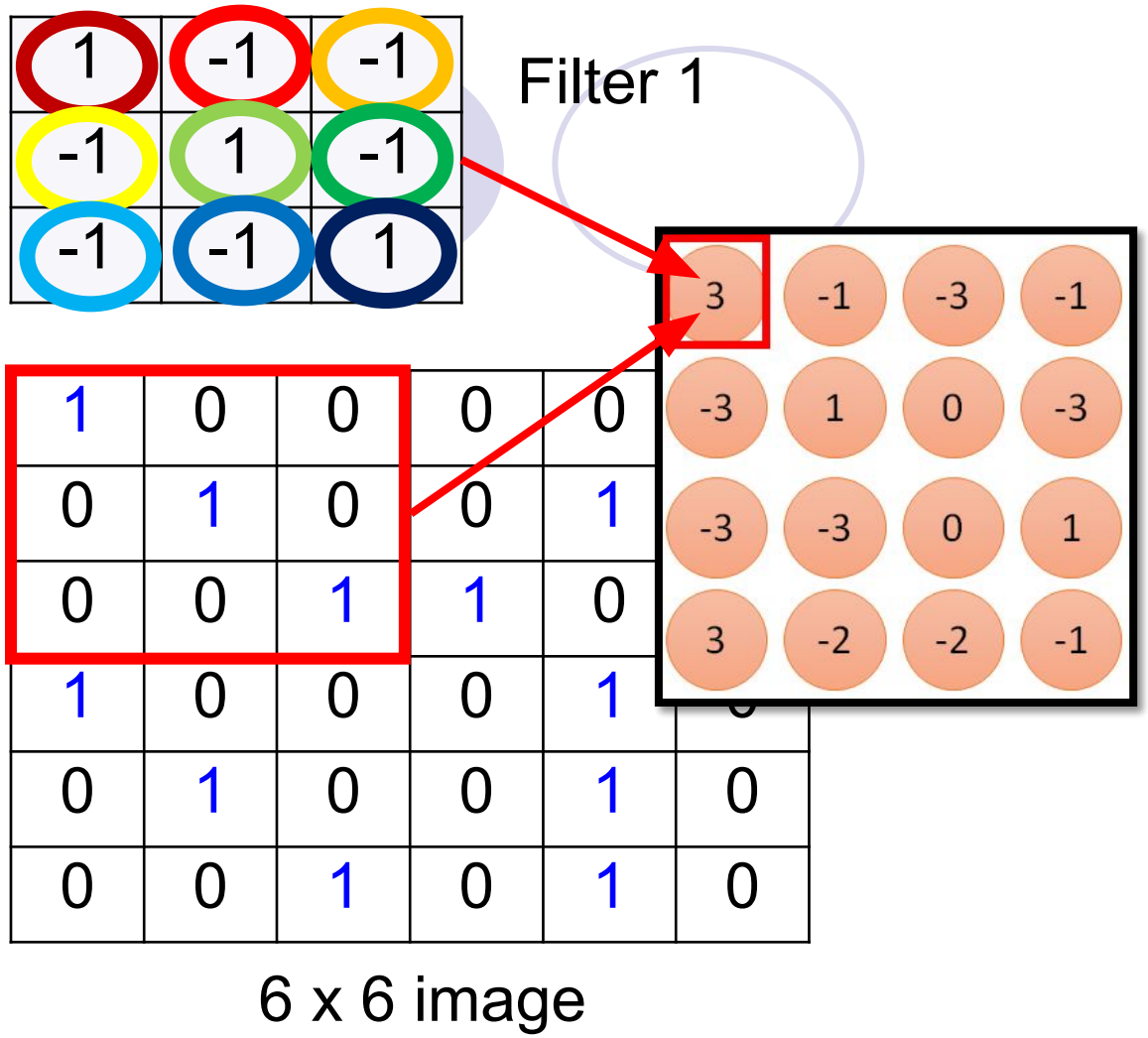
convolution



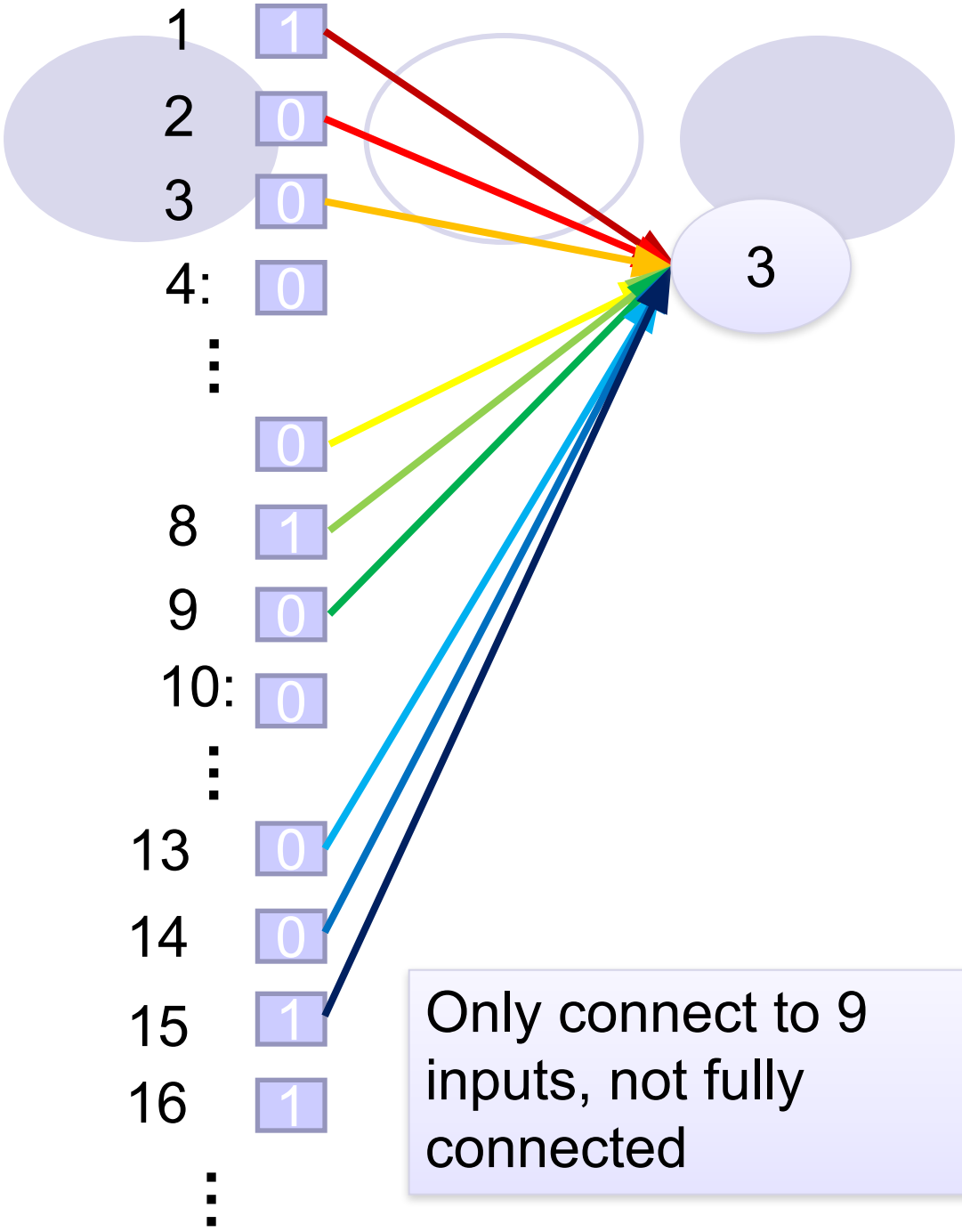
Fully-connected

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

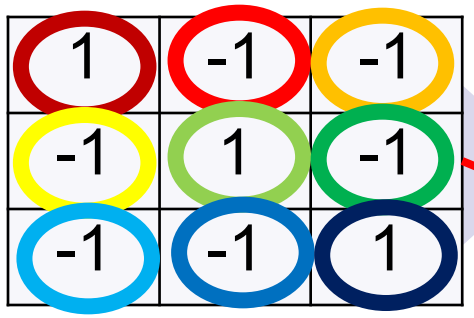




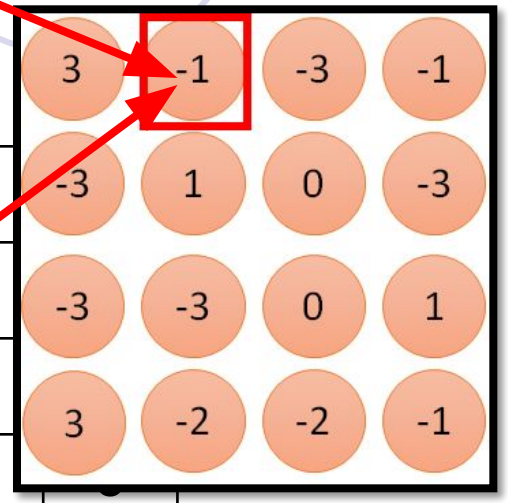
fewer parameters!



Only connect to 9 inputs, not fully connected



Filter 1

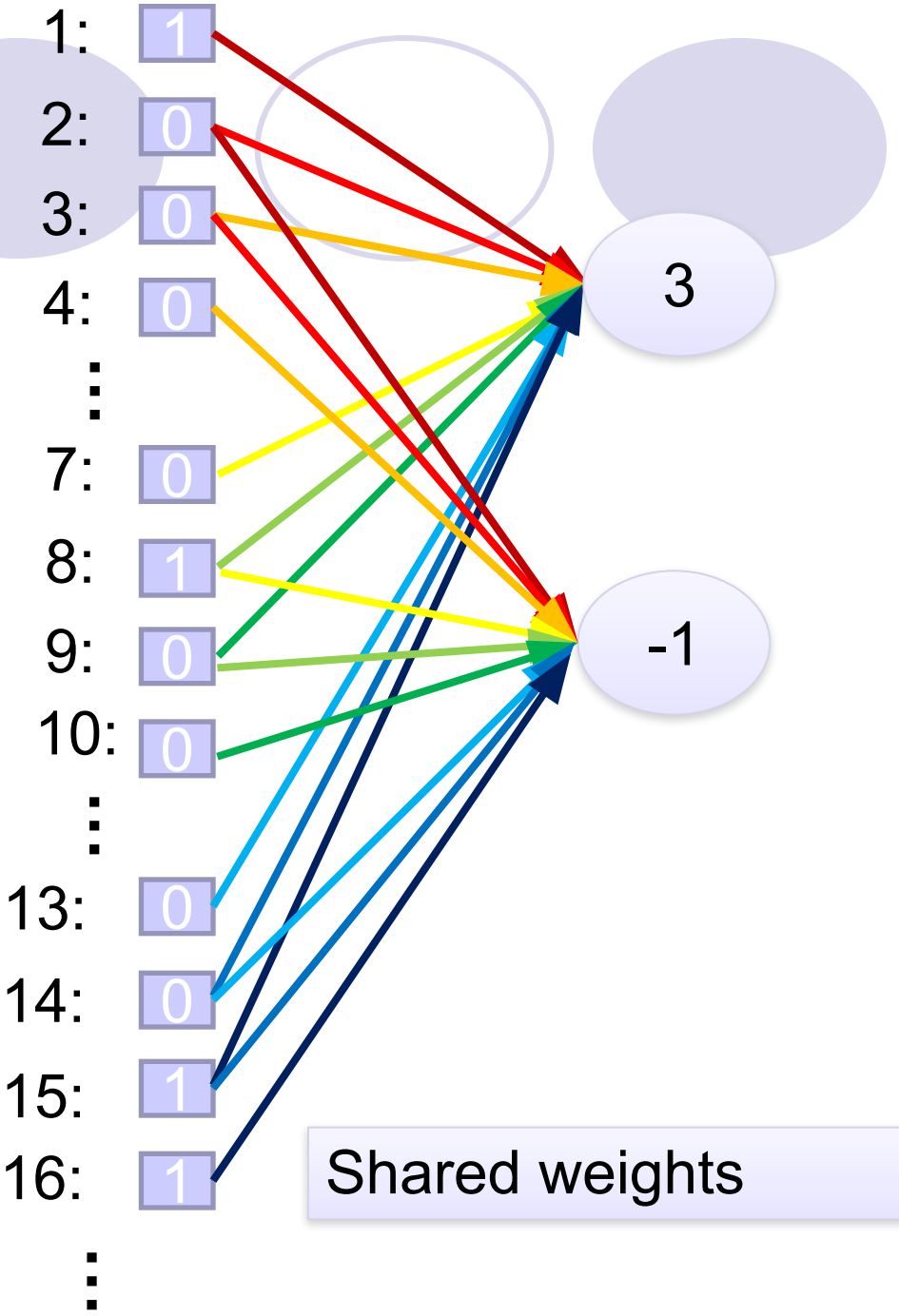


1	0	0	0	0	
0	1	0	0	1	
0	0	1	1	0	
1	0	0	0	1	
0	1	0	0	1	0
0	0	1	0	1	0

6 x 6 image

Fewer parameters

Even fewer parameters



Shared weights

The whole CNN

Convolution

Max Pooling

Convolution

Max Pooling

Can repeat many times

Flattened

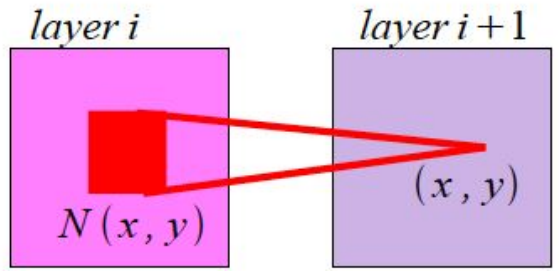
Fully Connected Feedforward network

cat dog ……

SPECIAL LAYERS

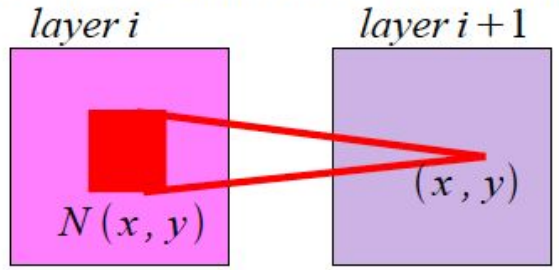
Over the years, some new modules have proven to be very effective when plugged into conv-nets:

- **Pooling** (average, L2, max)



$$\boldsymbol{h}_{i+1,x,y} = max_{(j,k)\in N(x,y)} \boldsymbol{h}_{i,j,k}$$

- **Local Contrast Normalization** (over space / features)



$$\boldsymbol{h}_{i+1,x,y} = \frac{\boldsymbol{h}_{i,x,y} - m_{i,x,y}}{\sigma_{i,x,y}}$$

Jarrett et al. "What is the best multi-stage architecture...?" ICCV 2009

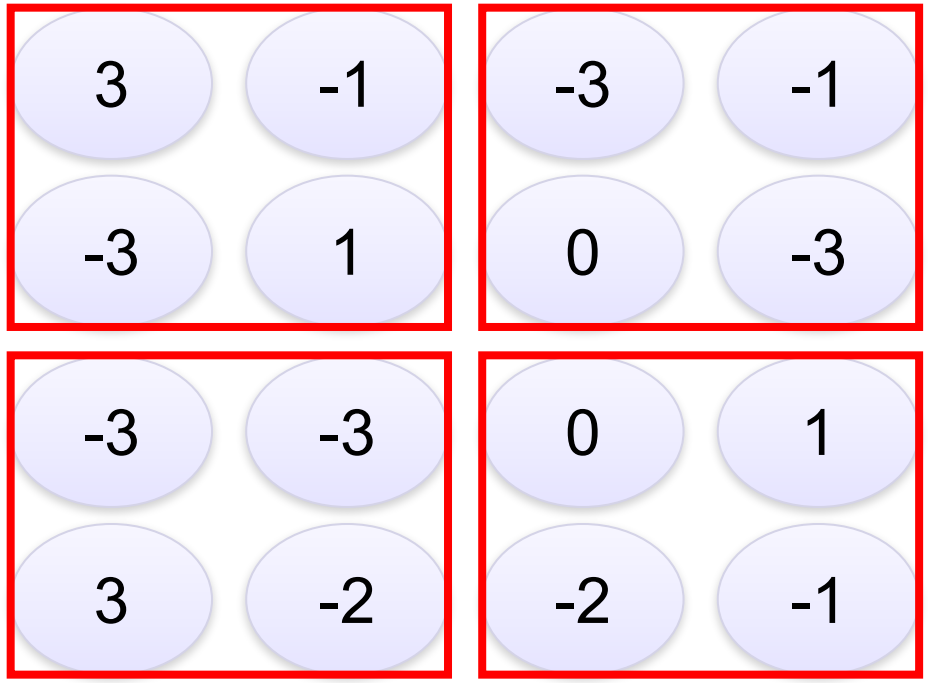
Max Pooling

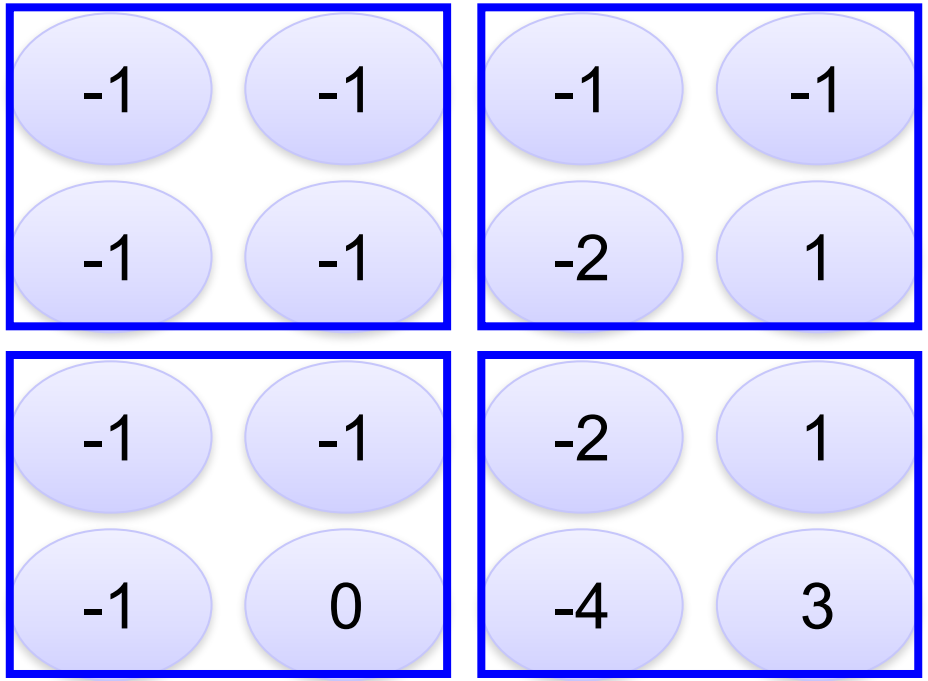
1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

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

Filter 2





Why Pooling

- Subsampling pixels will not change the object

bird

Subsampling

bird

We can subsample the pixels to make image smaller

fewer parameters to characterize the image

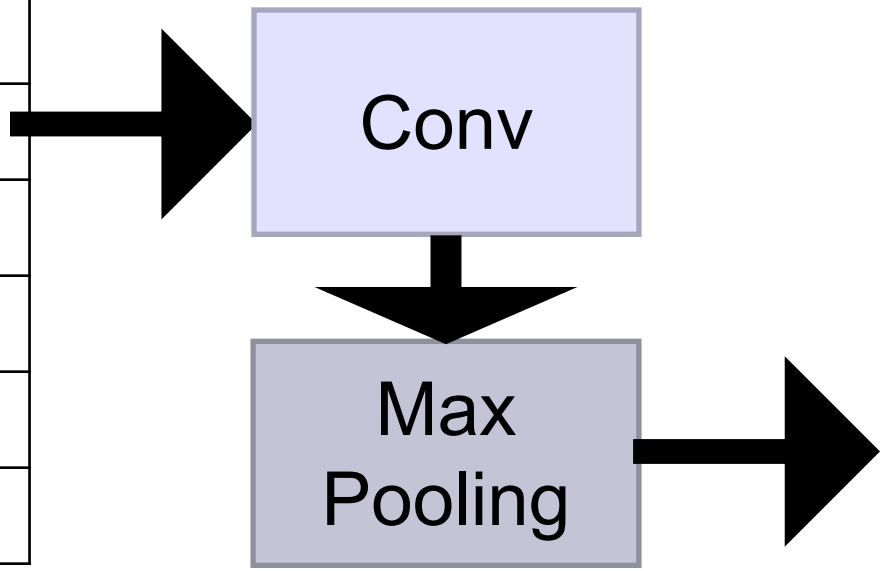
A CNN compresses a fully connected network in two ways:

- Reducing number of connections
- Shared weights on the edges
- Max pooling further reduces the complexity

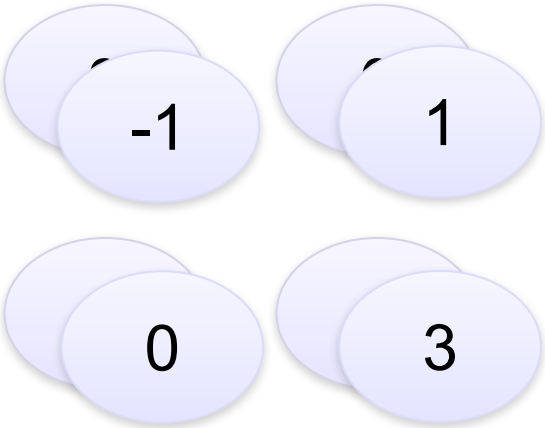
Max Pooling

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

6 x 6 image



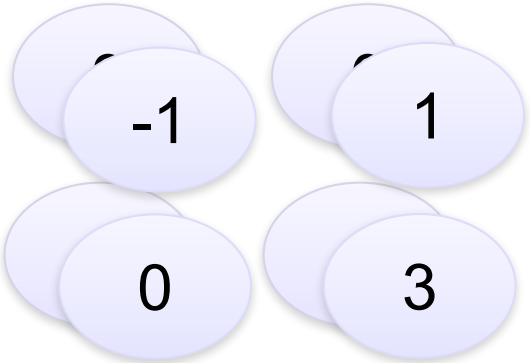
New image but smaller



2 x 2 image

Each filter is a channel

The whole CNN



Convolution

Max Pooling

Convolution

Max Pooling

Can repeat many times

A new image

Smaller than the original image

The number of channels is the number of filters

The whole CNN

Convolution

Max Pooling

A new image

Convolution

Max Pooling

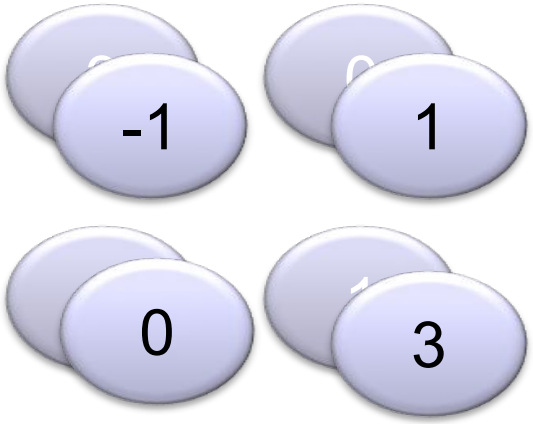
A new image

Flattened

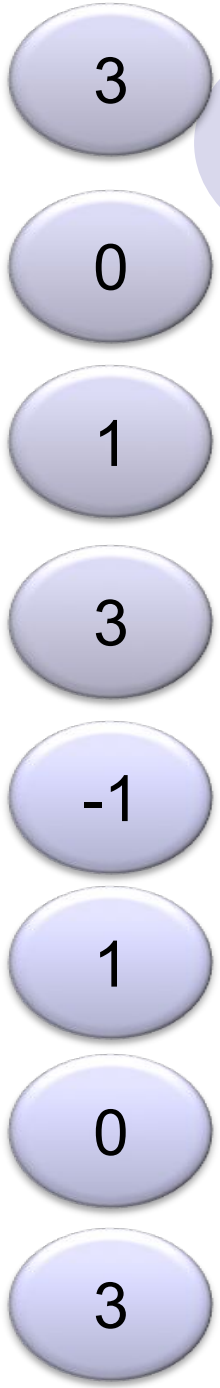
Fully Connected Feedforward network

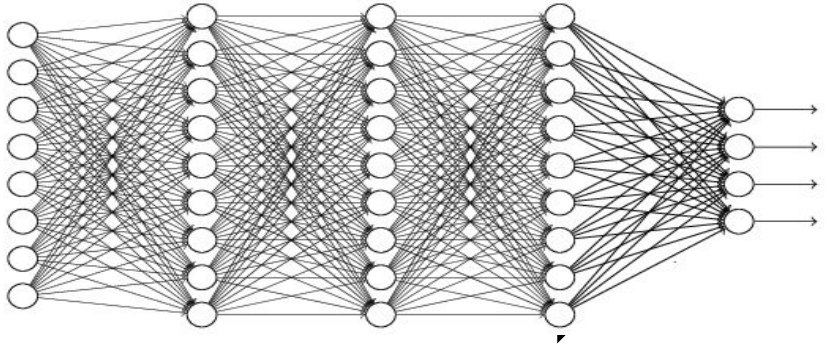
cat dog ……

Flattening



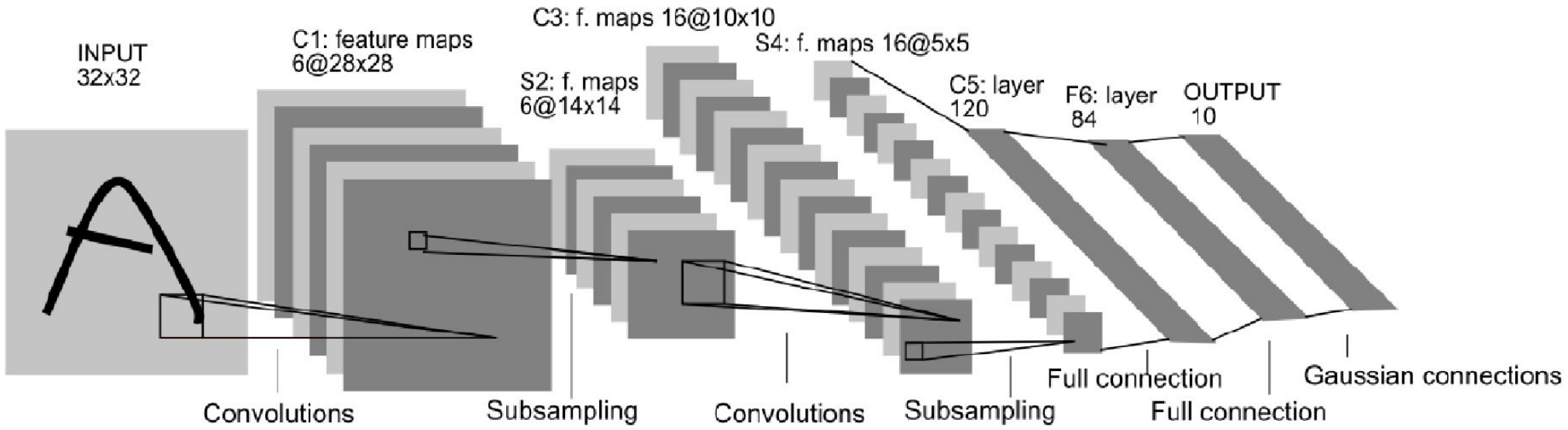
Flattened





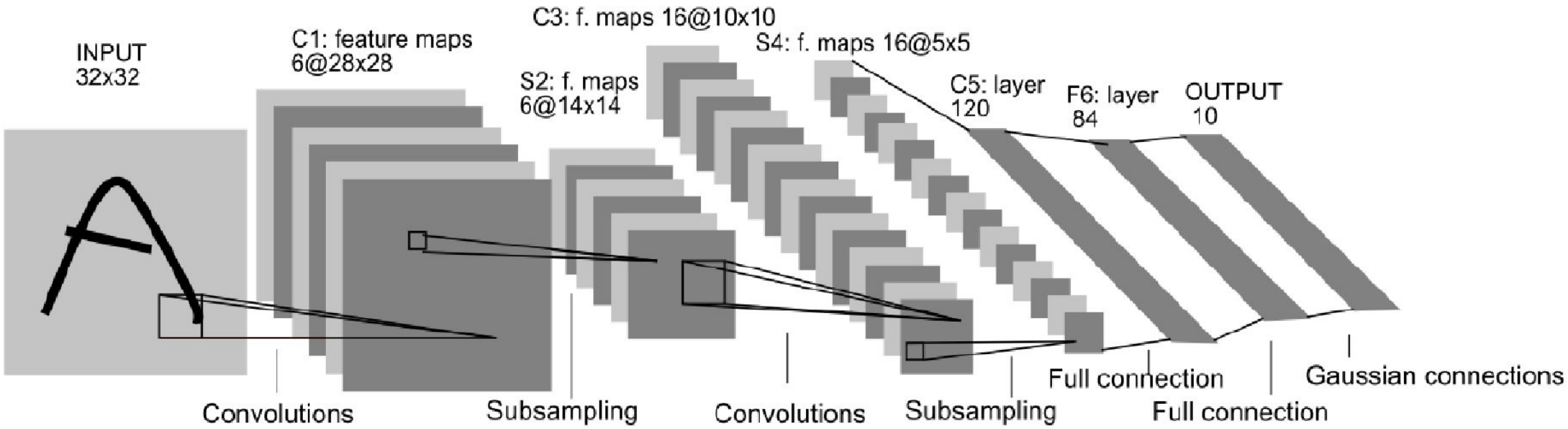
Fully Connected Feedforward network

A real CNN: LeNet5



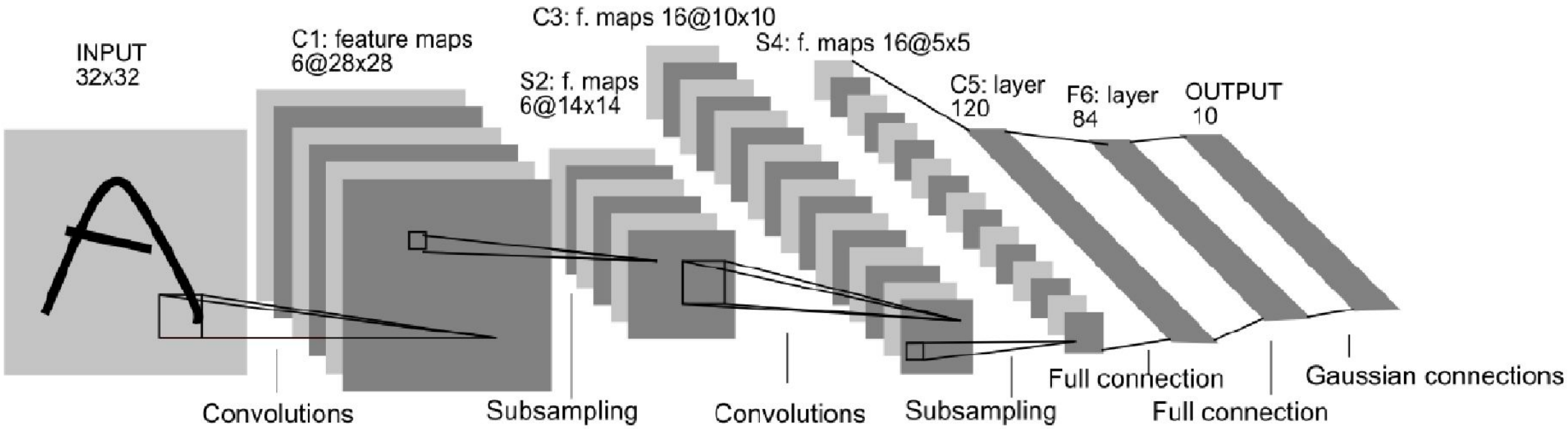
- C1: Convolutional layer
- S2: Subsampling layer, reducing the number of neurons
- C3: Convolutional layer
- S4: Subsampling layer
- C5,F6, output: full connection

A real CNN: Convolutional Layer



- C1: Convolutional layer
  - 6 feature maps of size 28X28
  - Each C1 neuron has a kernel of size 5X5
  - In total we have (5X5+1)X6=156 weights
  - If fully connected between Input and C1, there will be (32X32+1)X28X28X6= 4,821,600

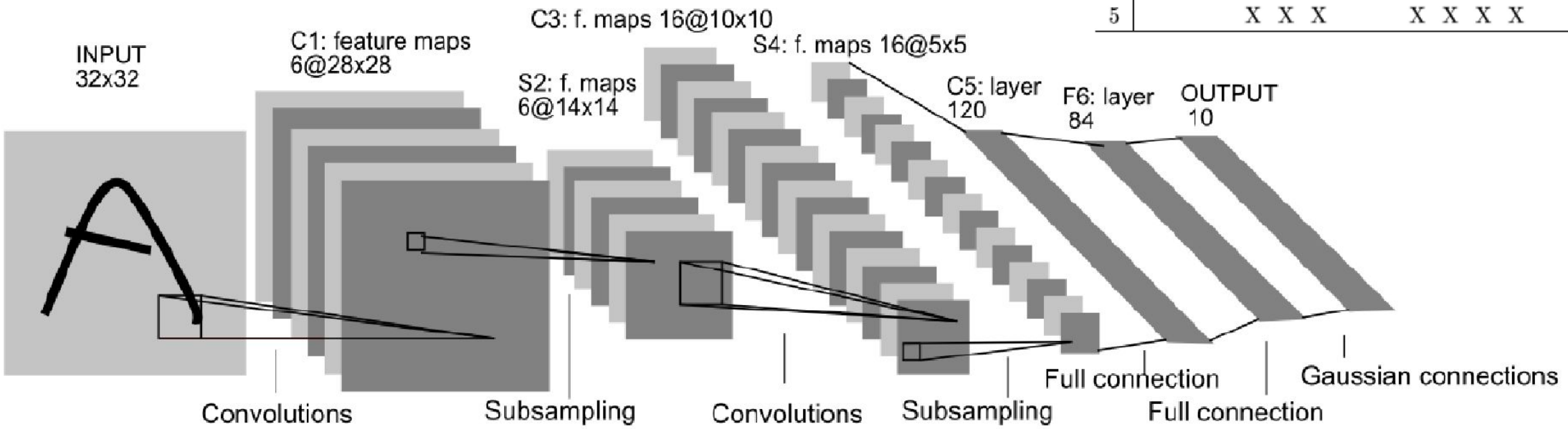
A real CNN: Subsampling



- S2: Subsampling layer
  - Each neuron in S2 takes the maximum output from 2X2 neurons underneath it in C1
  - 2x2 nonoverlapping receptive fields

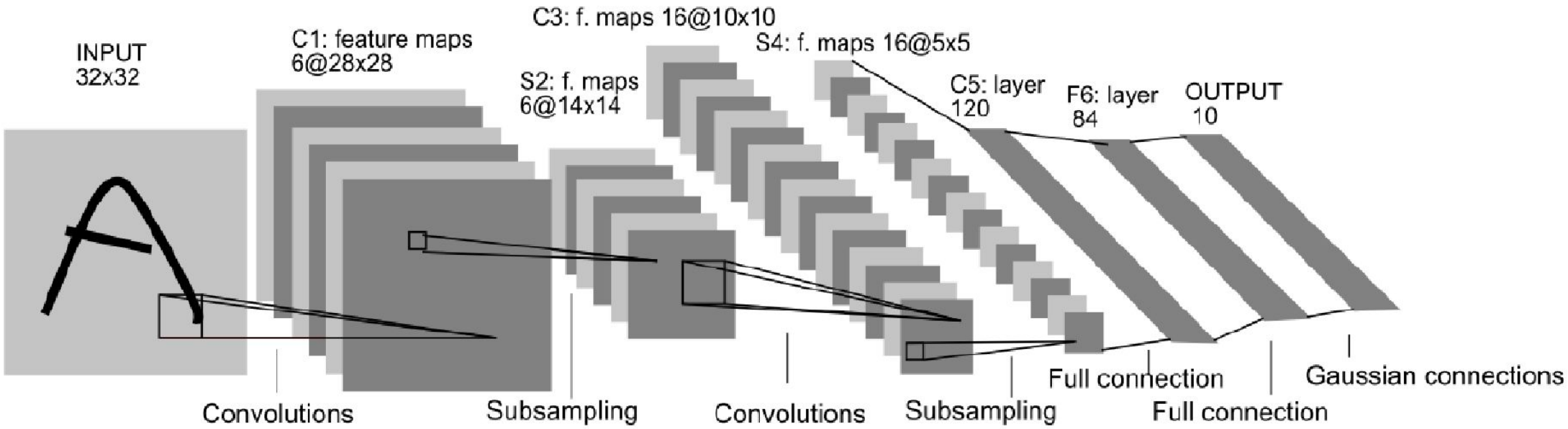
A real CNN: Convolutional layer

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	X				X	X	X			X	X	X	X		X	X
1	X	X				X	X	X			X	X	X	X		X
2	X	X	X				X	X	X			X		X	X	X
3		X	X	X			X	X	X	X			X		X	X
4			X	X	X			X	X	X	X		X	X		X
5				X	X	X			X	X	X	X		X	X	X



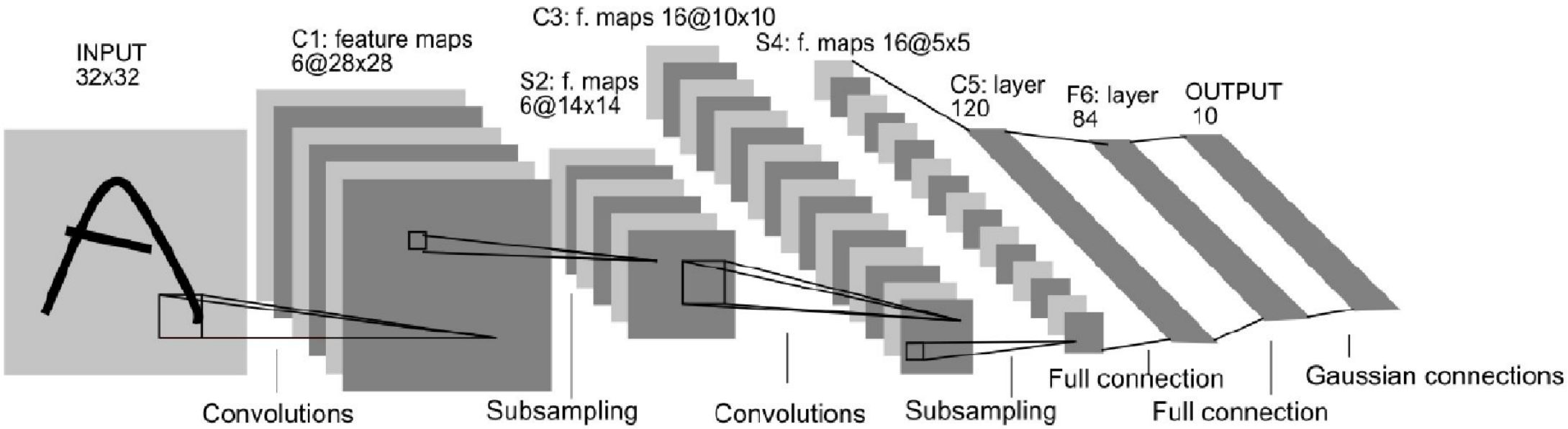
- C3: Convolutional layer
  - Each neuron in C3 is connected to some of S2 layers with a kernel of size 5X5 at identical locations in S2
  - Its activation is the sum of convolutions on these layers
  - 1516 trainable parameters

A real CNN: Subsampling



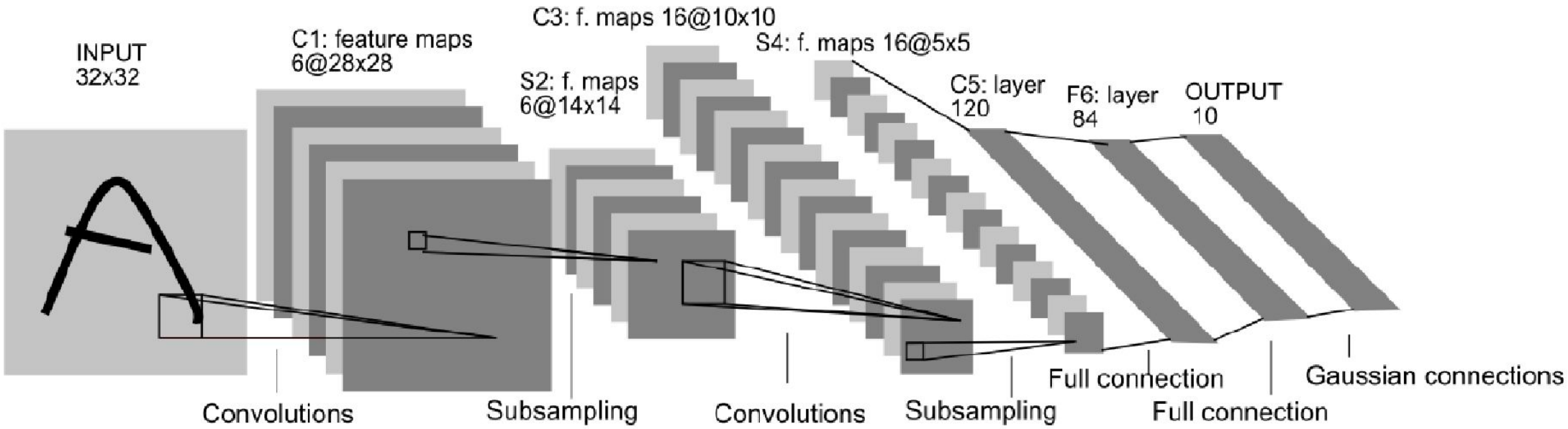
- S4: Subsampling layer
  - Each neuron in S4 takes the maximum output from 2X2 neurons underneath it in C3

A real CNN: Convolutional layer



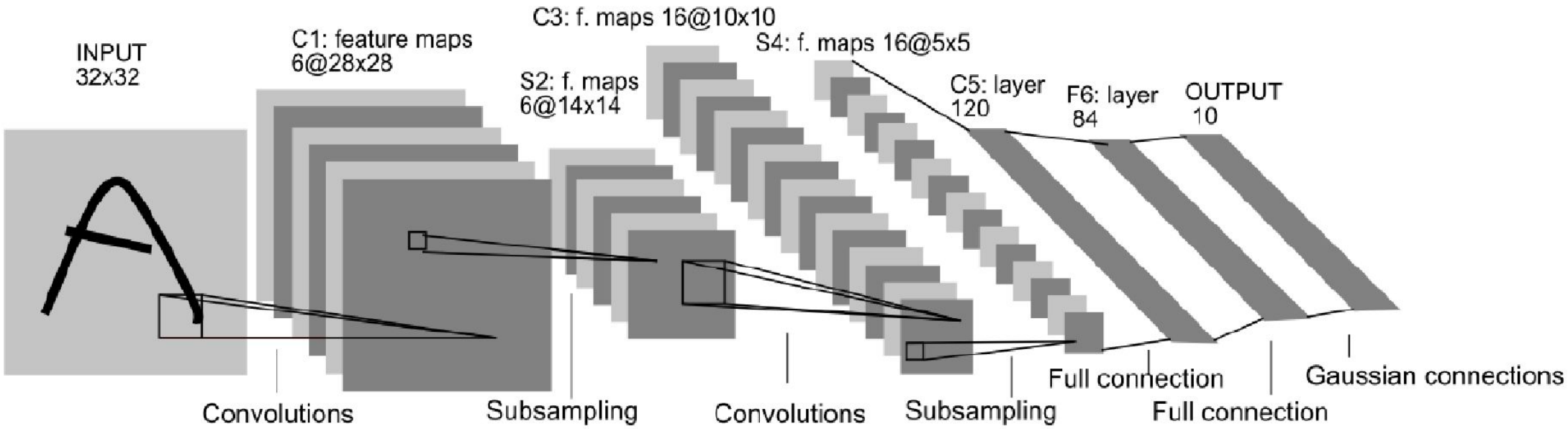
- C5: Convolutional layer
  - 120 feature maps of size 1x1
  - Each neuron in C5 takes a kernel convolution with all 16 feature maps in C5, and sums the convolution results together.

A real CNN: Fully connected layer



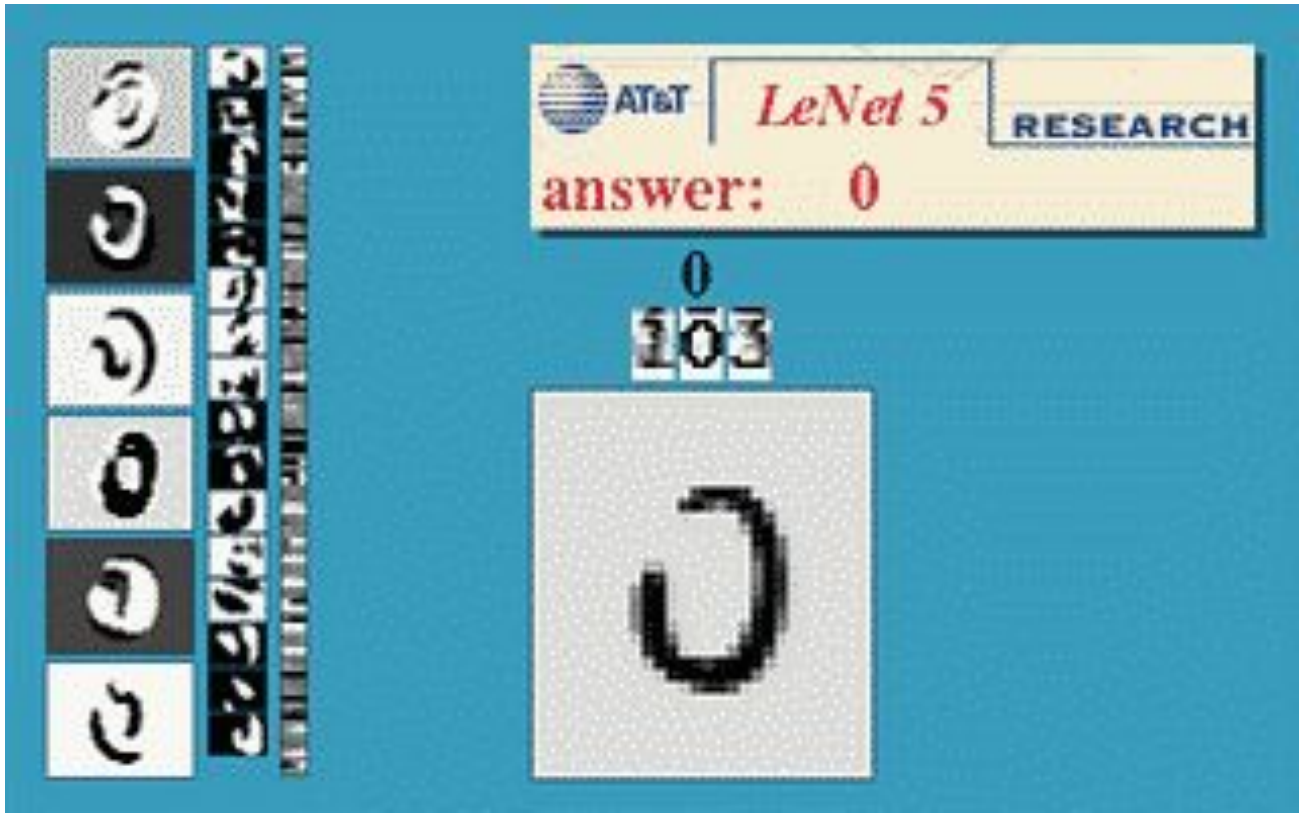
- F5: Fully connected layer
  - Each neuron in F6 is fully connected with all neurons in C5.
  - 84*(120+1)=10164 trainable connections

A real CNN: Output



- Output: RBF/Gaussian connections
  - Each neuron output from F6 is transformed by a Gaussian kernel before it is injected into output layer
  - This layer is fully connected with all neurons in F6

CNN Results



Challenges

- Neural network is still slightly fully connected.
  - The model may be too complex for many applications.
  - More elegant regularization method is demanded to avoid the overfitting risk.
    - Reducing the magnitudes of connection weights
    - Making the connection more sparse, while not losing too much modeling power.
- Dropout:
    - Some (about half of) neurons in a fully connected layer become inactive whose outputs will not participate in the forward pass and backpropagation.
    - Every time a neural network with reduced complexity is generated to process the input signals forwards, or updated by backpropagation.

Summary

- Overview of Deep learning
- Deep Network
- Practical issues
  - Momentum
  - Adaptive learning rate
- Convolutional neural networks
- LeNet5
- Challenges