an introduction to:
Deep Learning

aka or related to

Deep Neural Networks

Deep Structural Learning

Deep Belief Networks

etc,

DL is providing breakthrough results in speech recognition and image classification …

From this Hinton et al 2012 paper:

http://static.googleusercontent.com/media/research.google.com/en//pubs/archive/38131.pdf

modeling technique	#params [10^6]	WER Hub5'00-SWB	WER RT03S-FSH
GMM, 40 mix DT 309h SI	29.4	23.6	27.4
NN 1 hidden-layer×4634 units	43.6	26.0	29.4
+ 2×5 neighboring frames	45.1	22.4	25.7
DBN-DNN 7 hidden layers×2048 units	45.1	17.1	19.6
+ updated state alignment	45.1	16.4	18.6
+ sparsification	15.2 nz	16.1	18.5
GMM 72 mix DT 2000h SA	102.4	17.1	18.6

task	hours of training data	DNN-HMM	GMM-HMM with same data	GMM-HMM with more data
Switchboard (test set 1)	309	18.5	27.4	18.6 (2000 hrs)
Switchboard (test set 2)	309	16.1	23.6	17.1 (2000 hrs)
English Broadcast News	50	17.5	18.8	
Bing Voice Search (Sentence error rates)	24	30.4	36.2	
Google Voice Input	5,870	12.3		16.0 (>>5,870hrs)
Youtube	1,400	47.6	52.3	

go here: http://yann.lecun.com/exdb/mnist/

From here:
http://people.idsia.ch/~juergen/cvpr2012.pdf

Dataset	Best result of others [%]	MCDNN [%]	Relative improv. [%]
MNIST	0.39	0.23	41
NIST SD 19	see Table 4	see Table 4	30-80
HWDB1.0 on.	7.61	5.61	26
HWDB1.0 off.	10.01	6.5	35
CIFAR10	18.50	11.21	39
traffic signs	1.69	0.54	72
NORB	5.00	2.70	46

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?

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.**

hmmm… OK, but:

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

hmmm… OK, 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
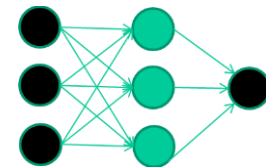
but these algorithms are not good at learning the weights for networks with more hidden layers
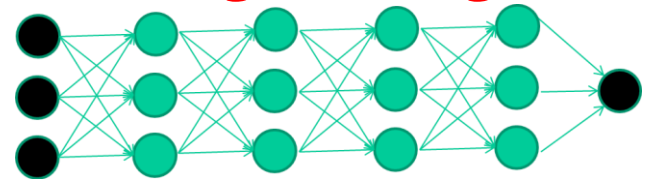
what's new is: algorithms for training many-later networks

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

1.4

W3

$f(x)$

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

-0.06

2.7

-2.5

-8.6

$f(x)$

0.002

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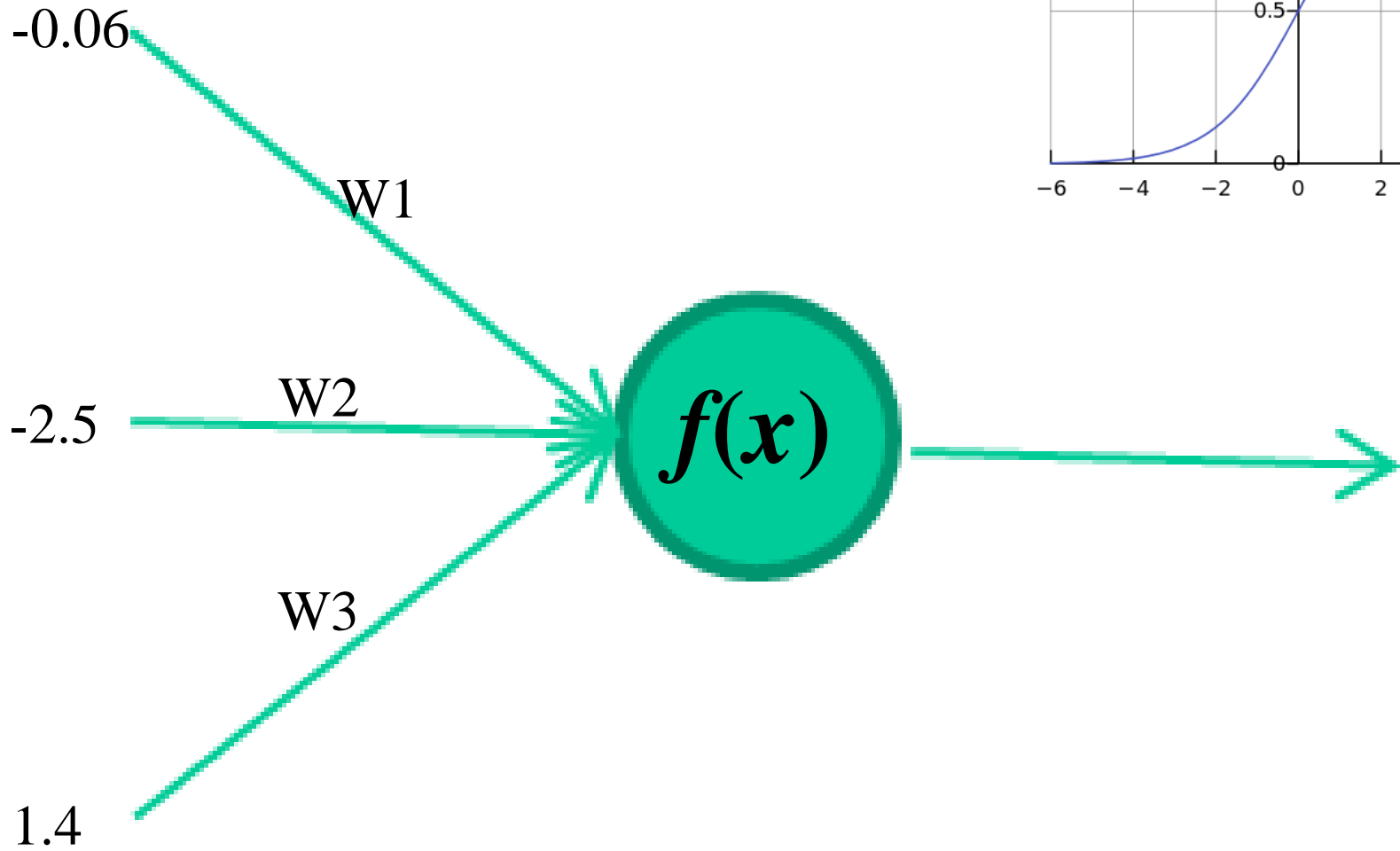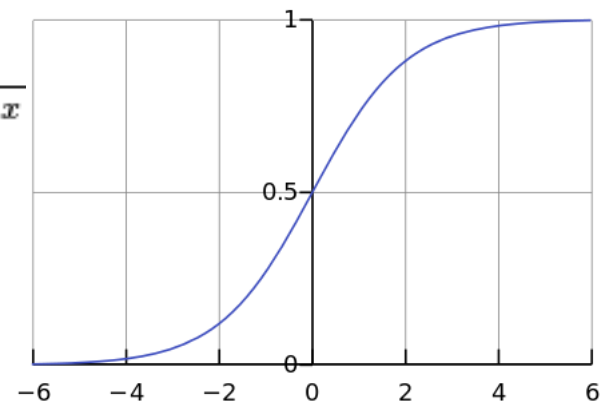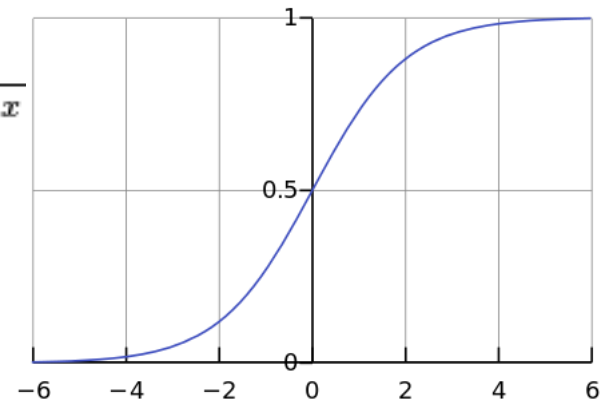
$x =$ -0.06×2.7 + 2.5×8.6 + 1.4×0.002 = 21.34

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 …
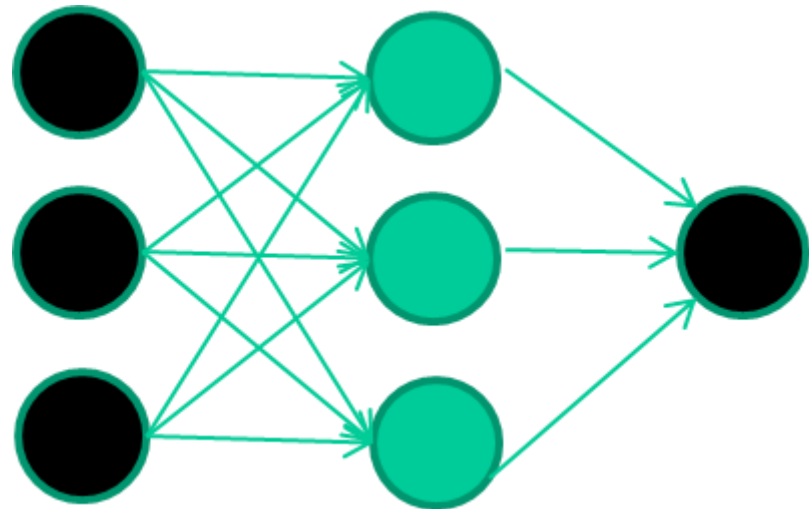
Training the neural network

Fields   **class**

1.4 2.7 1.9   0
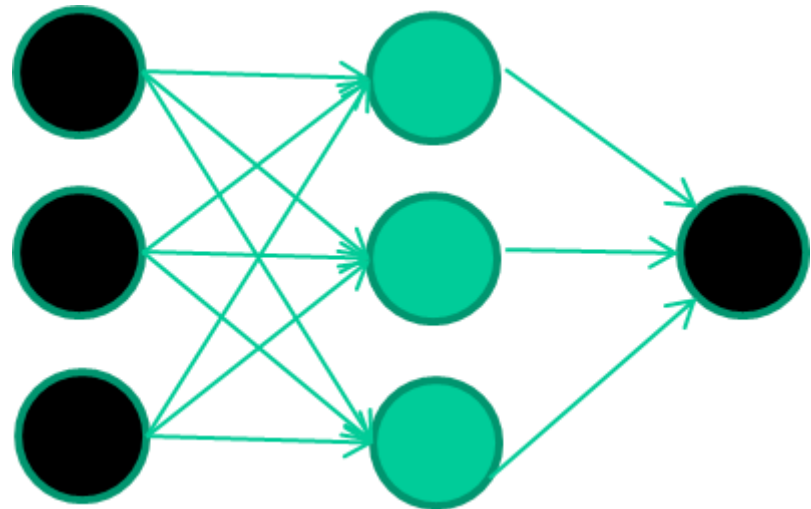
3.8 3.4 3.2   0

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 …

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 …

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 …

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 …

1.4

2.7   **0.8**

    0

1.9   *error* 0.8

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 …

1.4

2.7

1.9

0.8

0

error 0.8
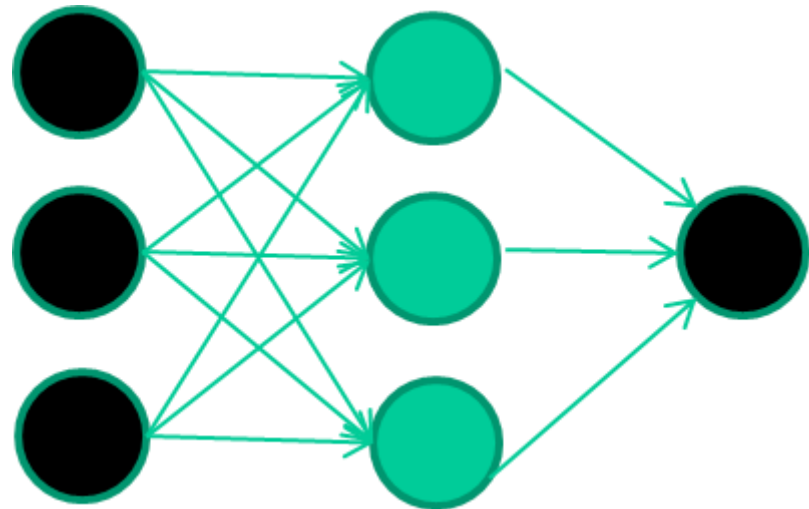
Training data
Fields   **class**

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 …
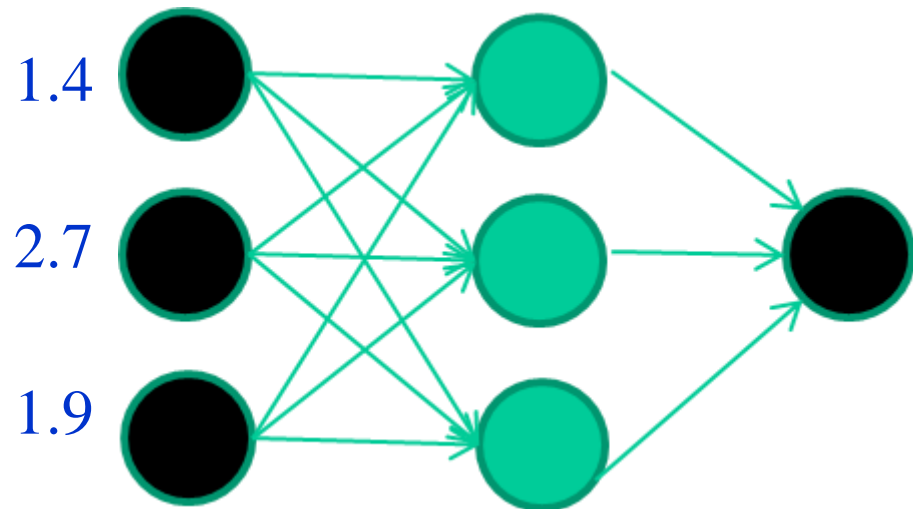
Training data
Fields   **class**

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
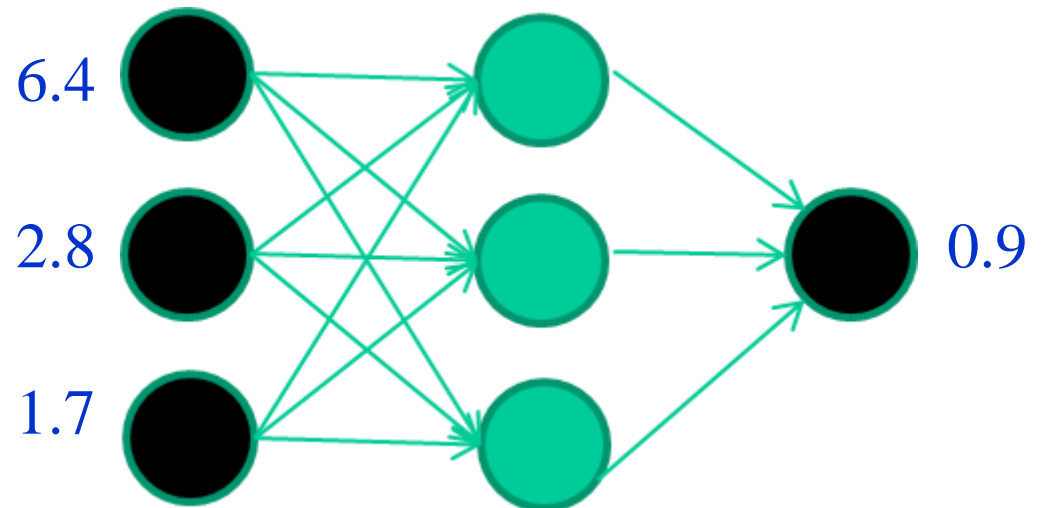
6.4   2.8   1.7   0.9

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 …

6.4

2.8   0.9
                                       1
1.7   *error* -0.1

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 …

6.4

2.8

1.7

0.9

1

error   -0.1
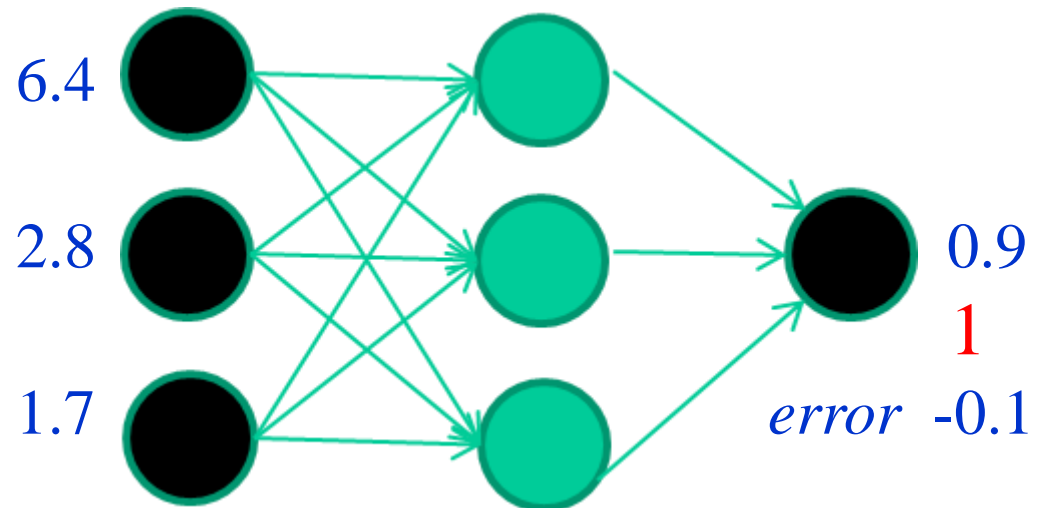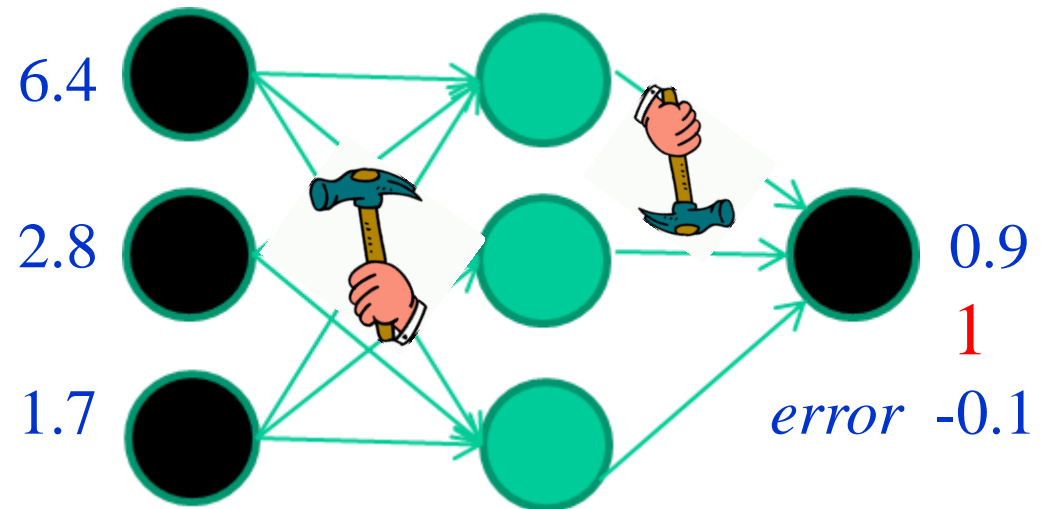
Training data
Fields   **class**

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 ….

6.4

2.8

1.7

0.9

1

error -0.1
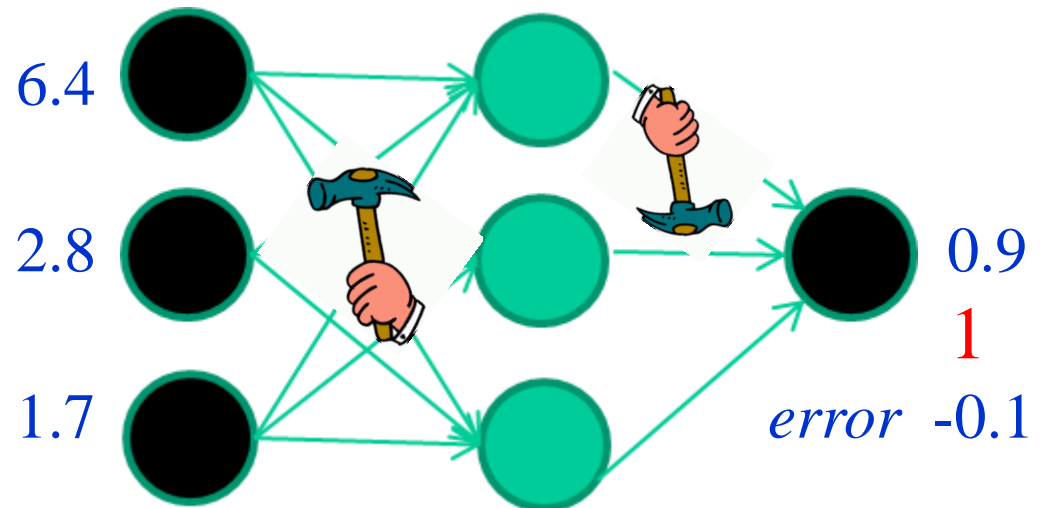
Repeat this thousands, maybe millions of times – each time taking a random training instance, and making slight weight adjustments

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

The decision boundary perspective…
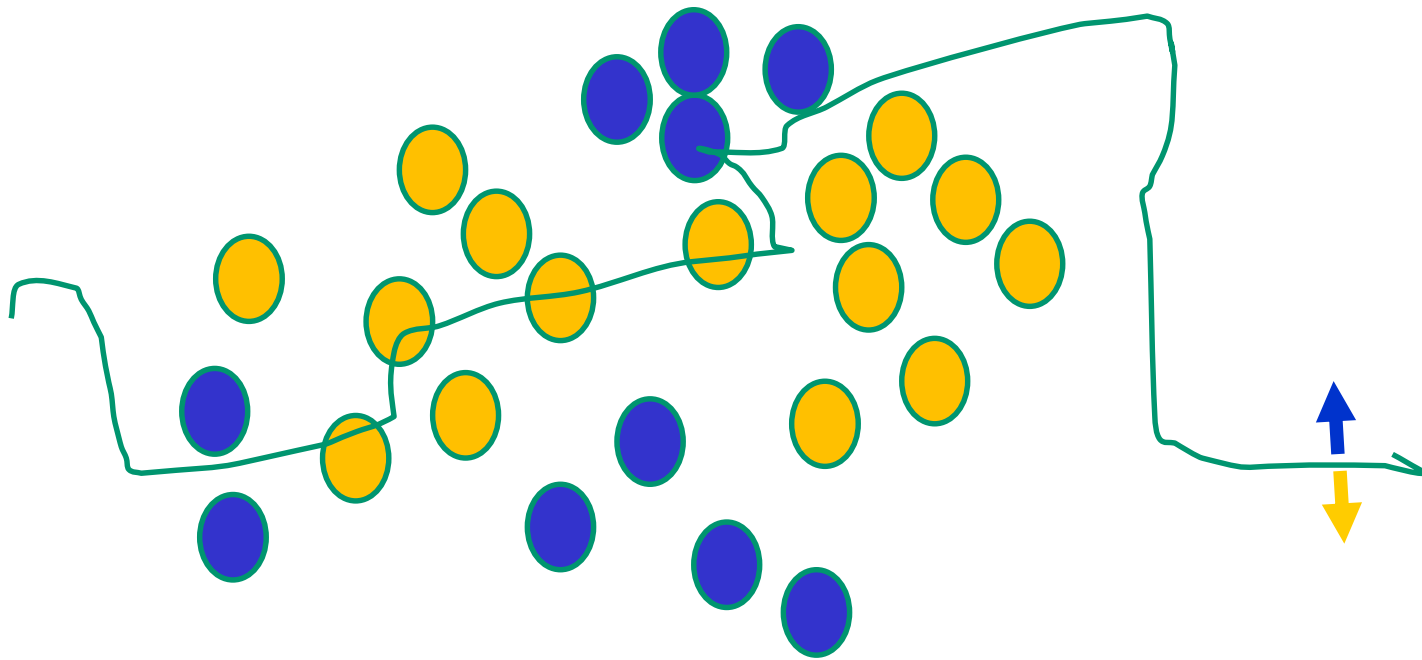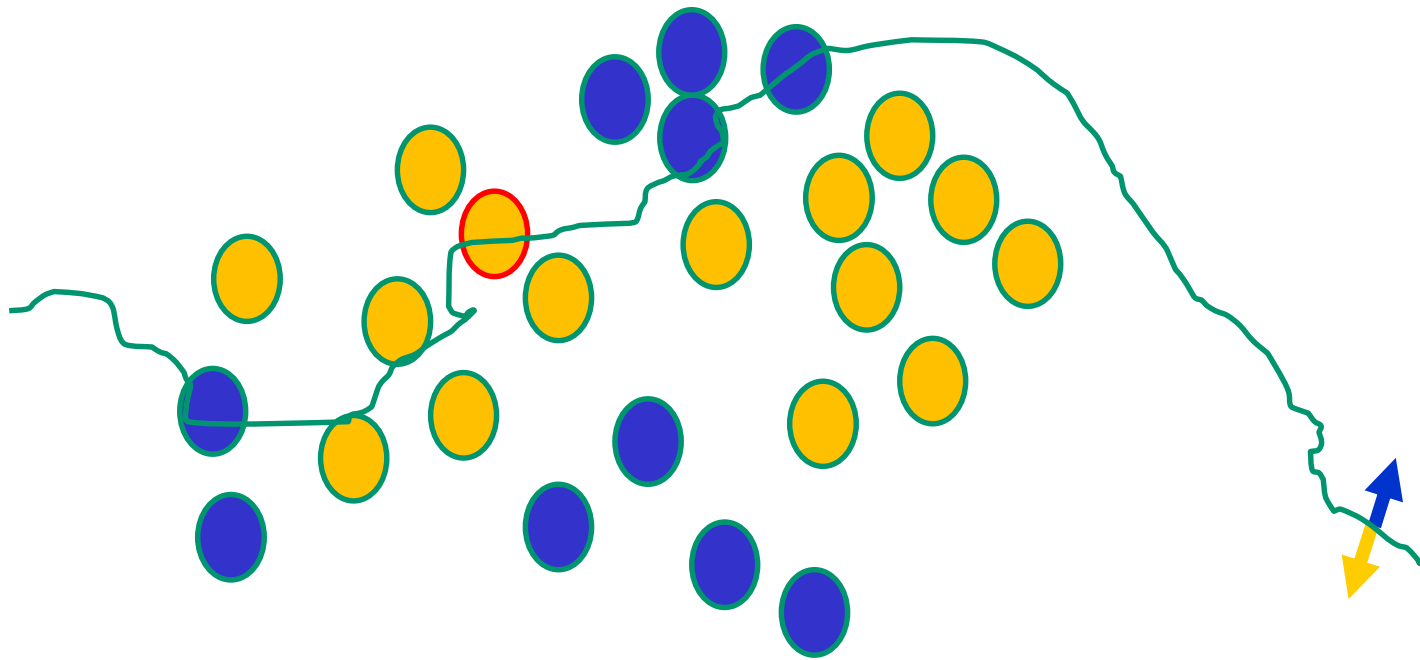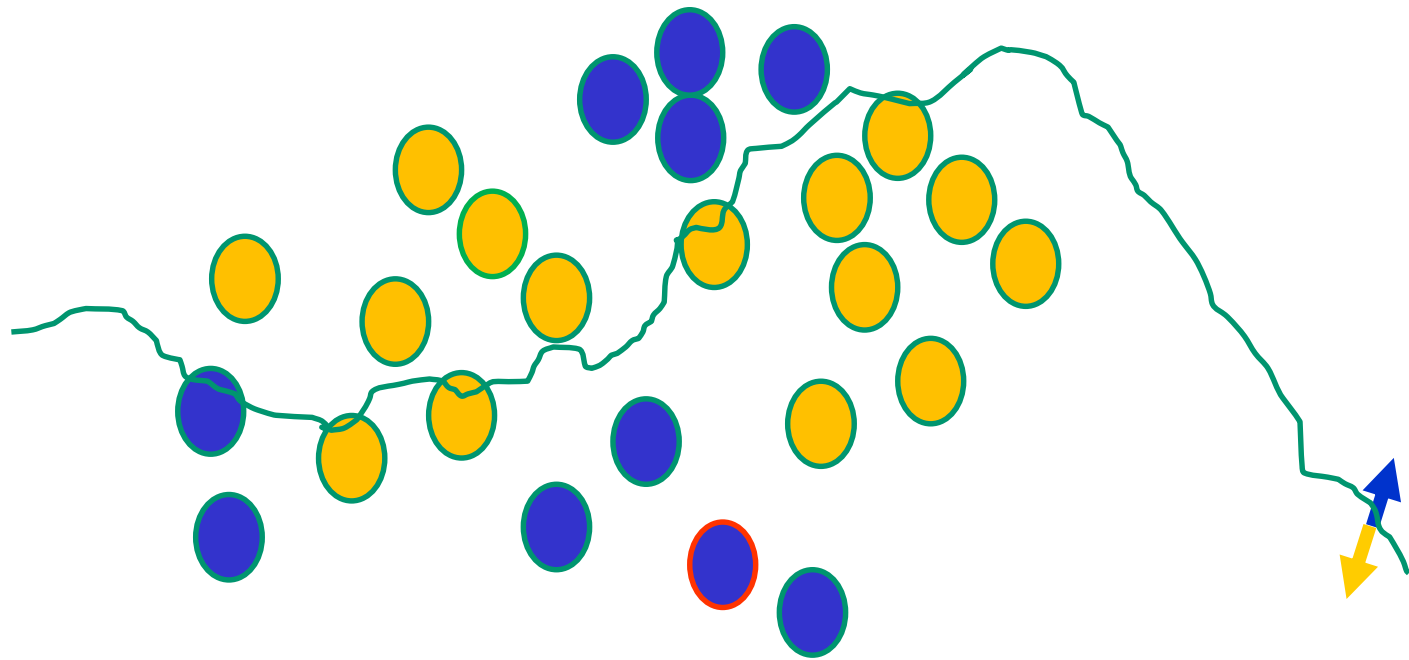
The decision boundary perspective…
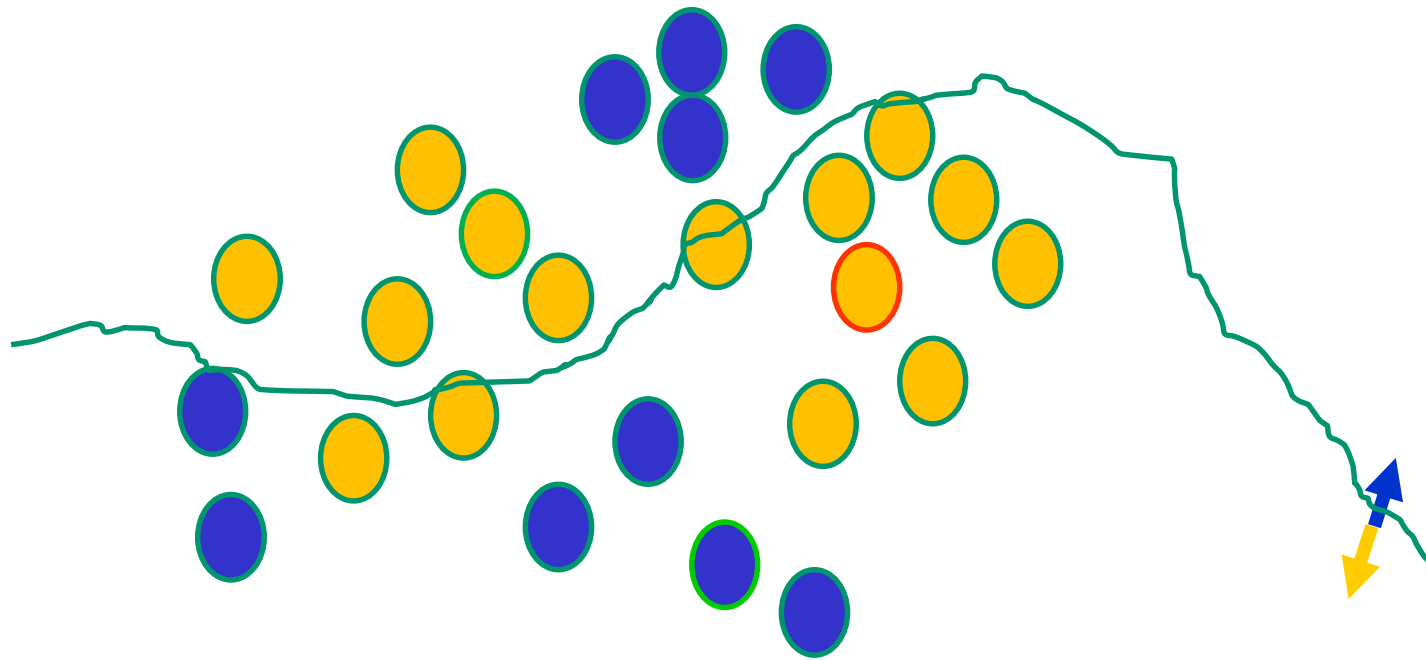
The decision boundary perspective…

The decision boundary perspective…

The decision boundary perspective…
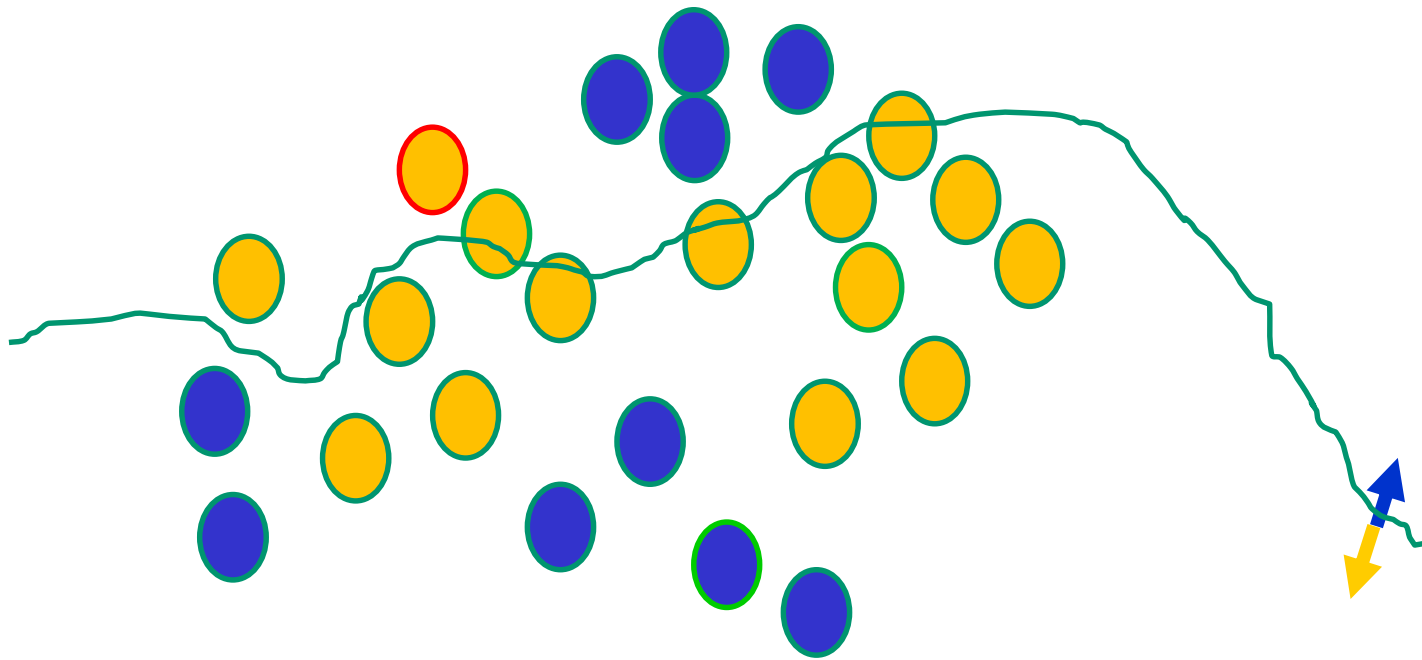
The decision boundary perspective…
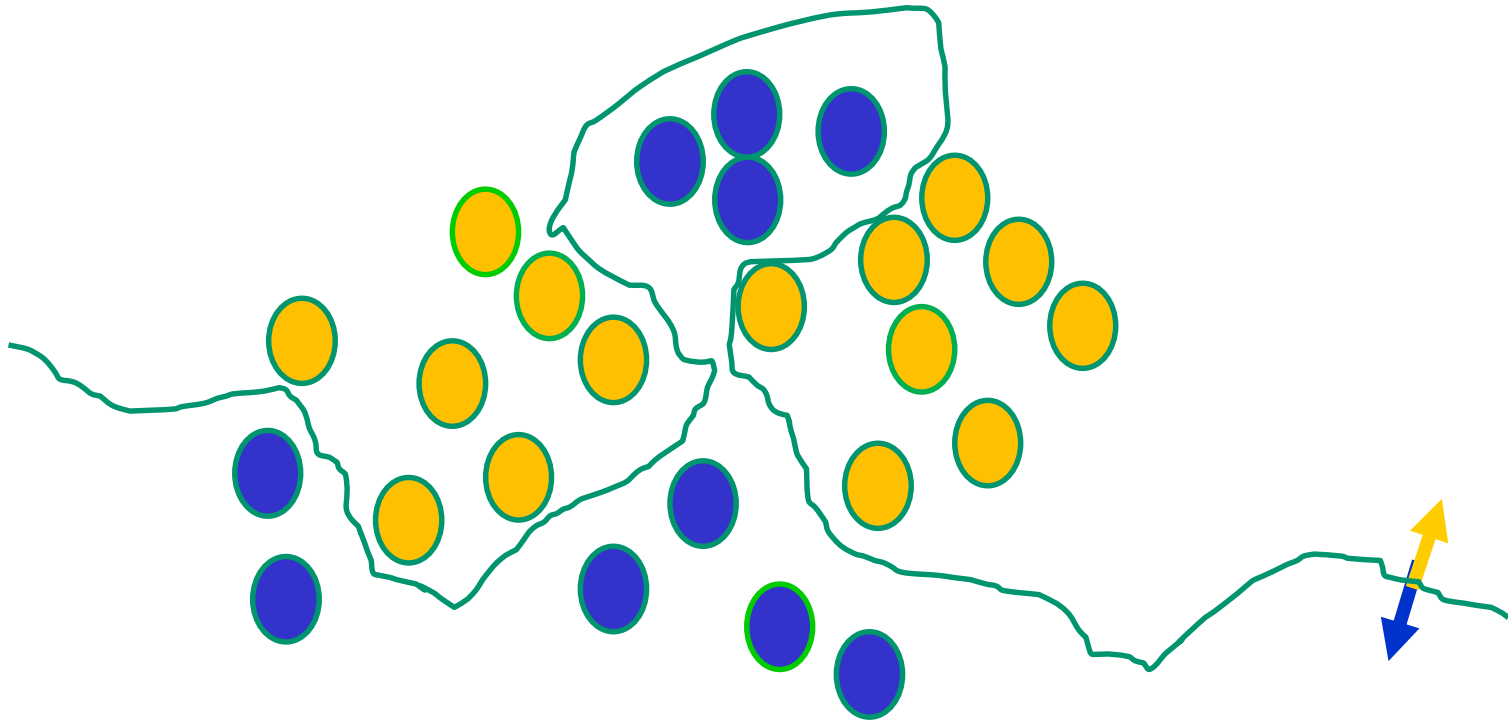
The point I am trying to make

- 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
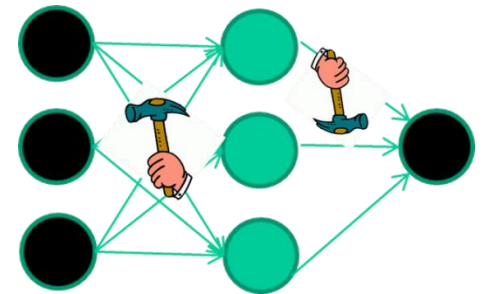
Some other points

Detail of a standard NN weight learning algorithm – **later**

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.

Some other 'by the way' points

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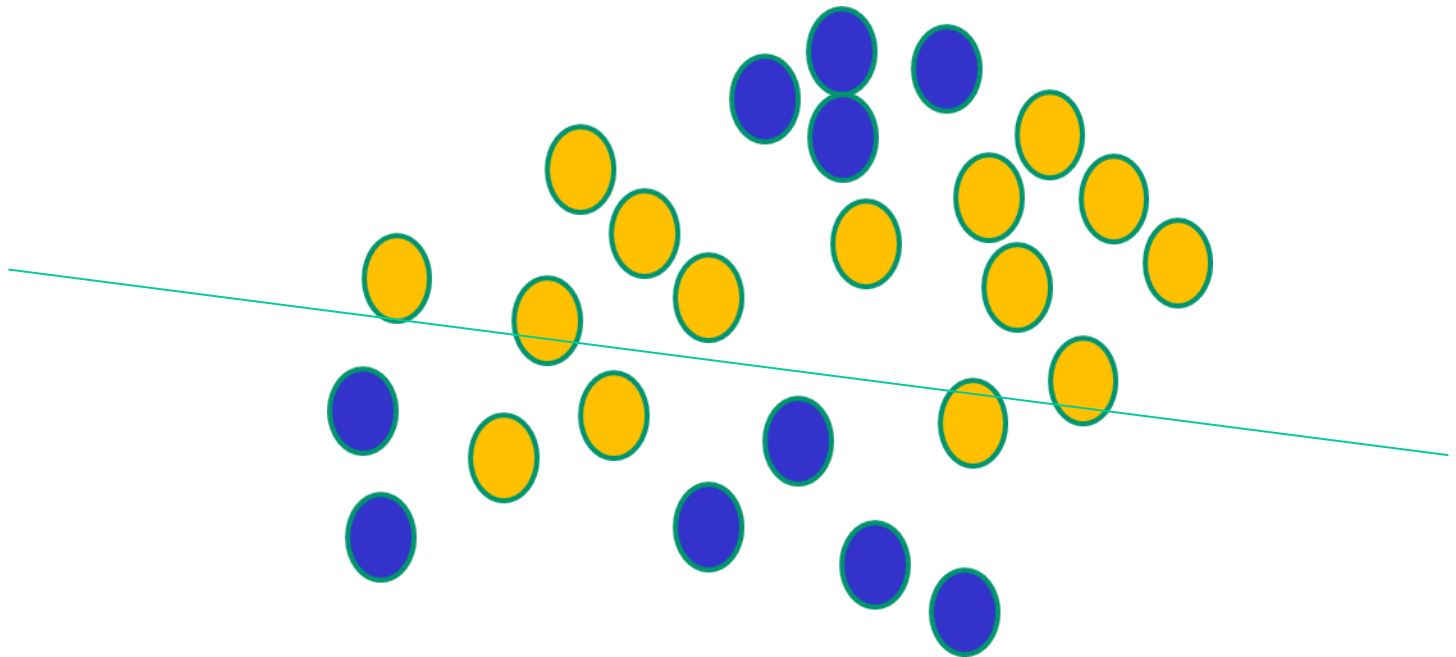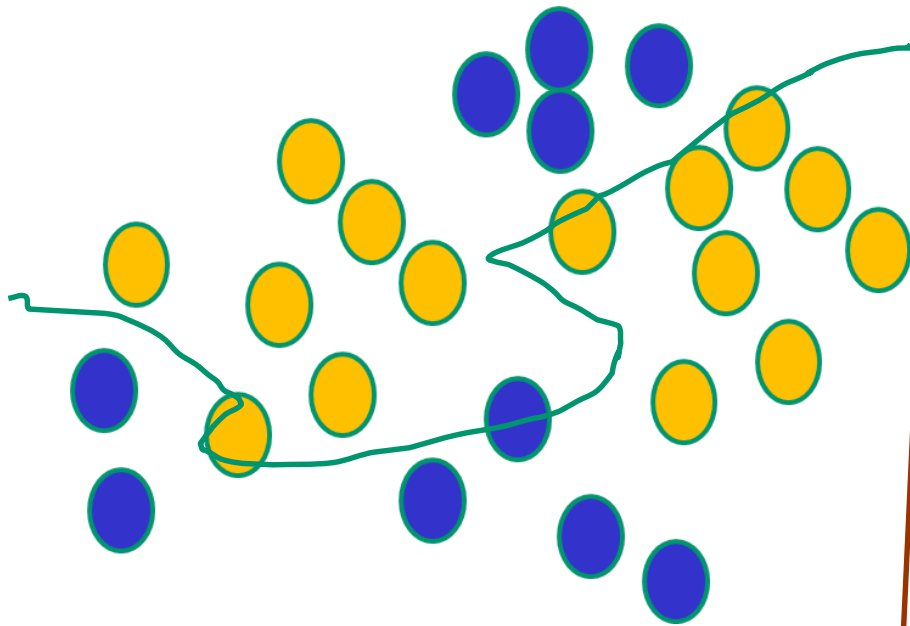
Some other 'by the way' points

NNs use nonlinear $f(\mathrm{x})$ so they

can draw complex boundaries,

but keep the data unchanged
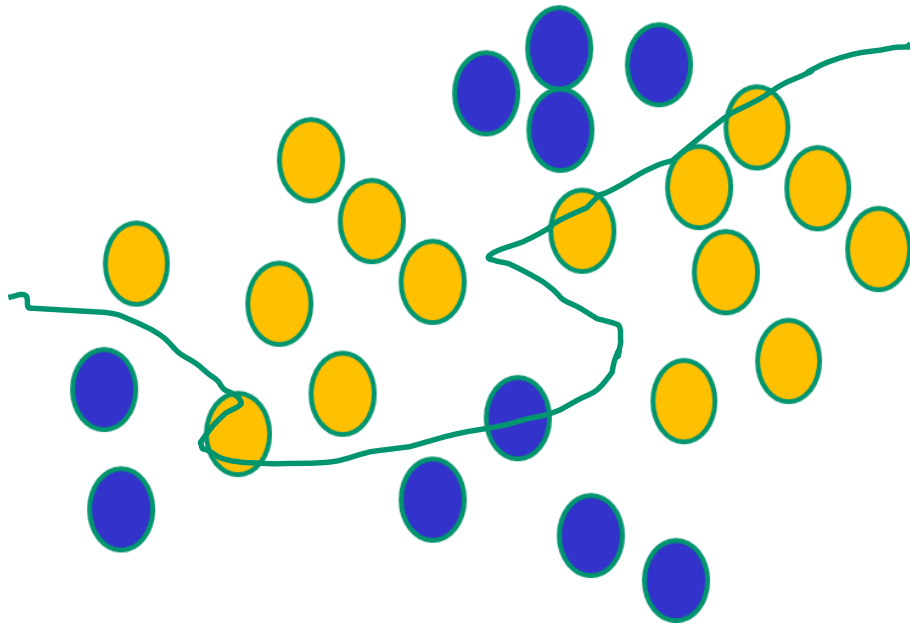
Some other 'by the way' points

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
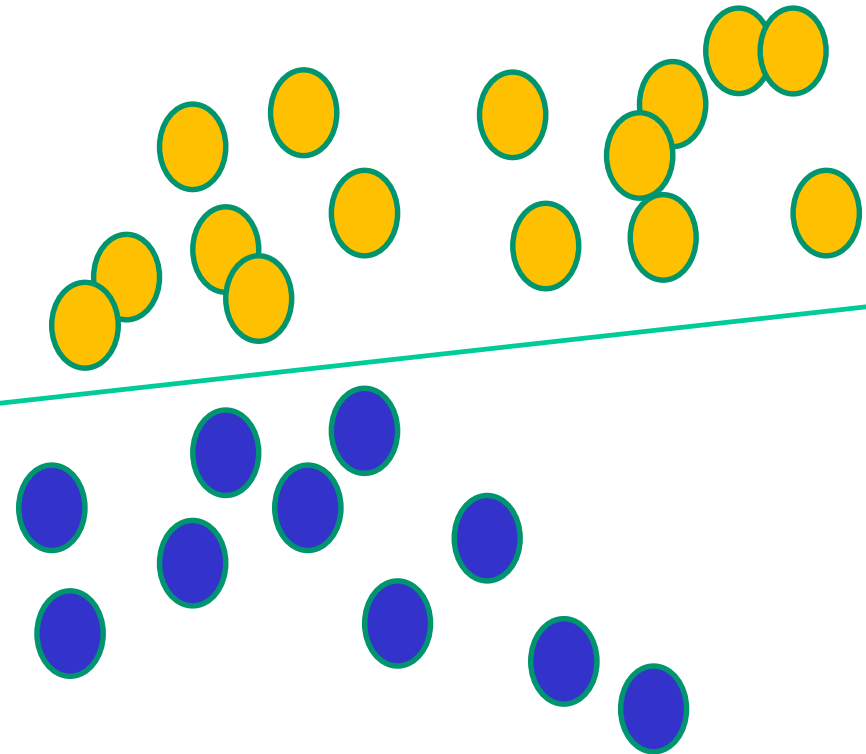
Feature detectors
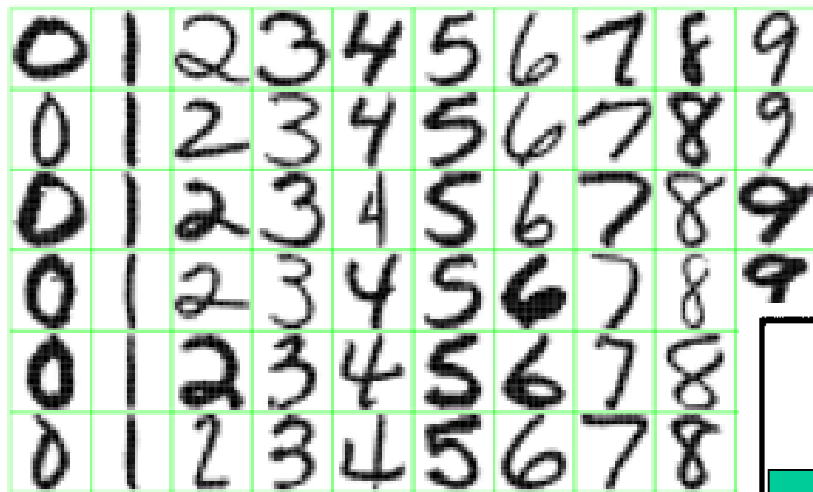
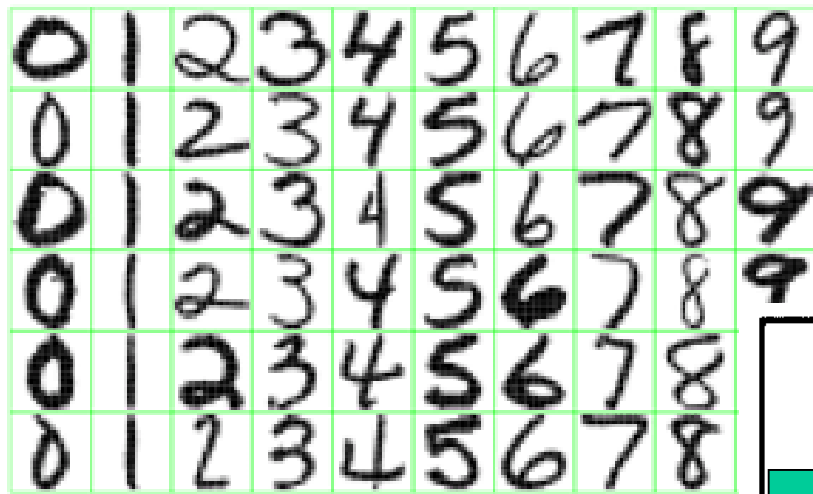Figure 1.2: *Examples of handwritten digits from postal envelopes.*
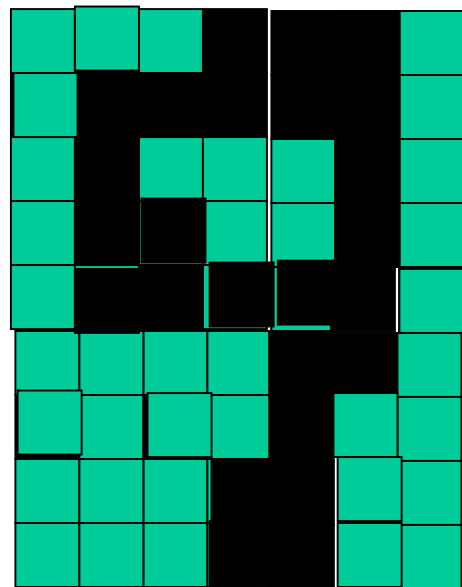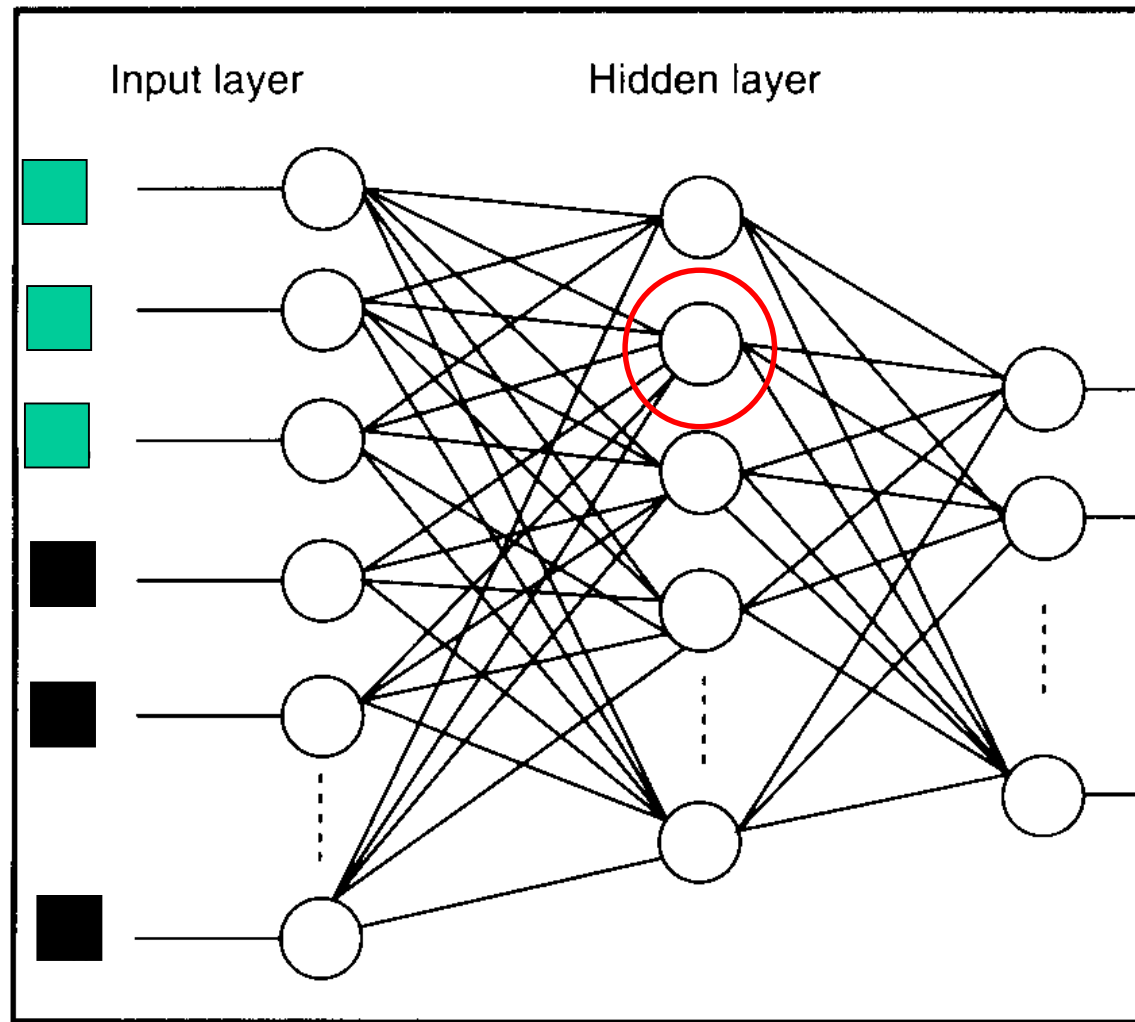
Input layer
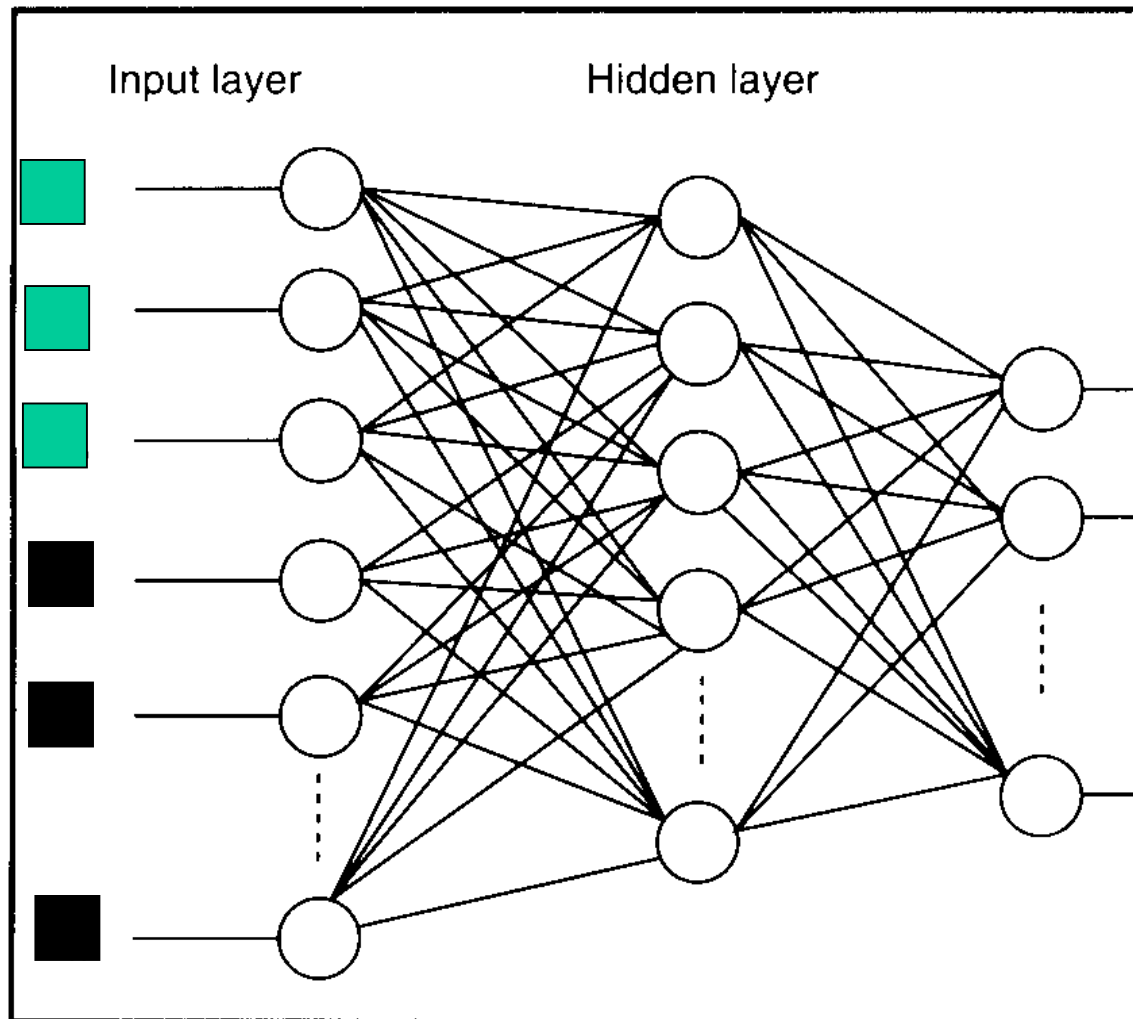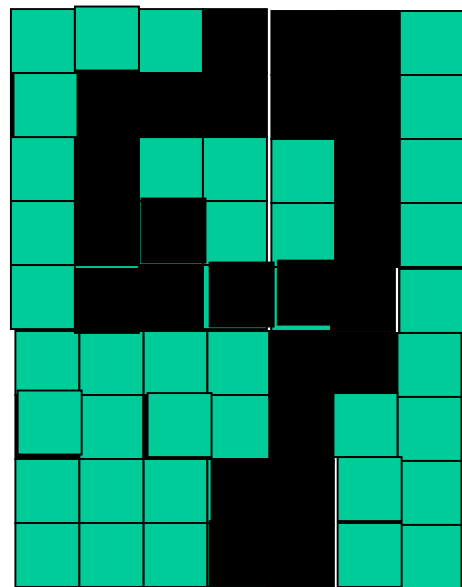
Hidden layer

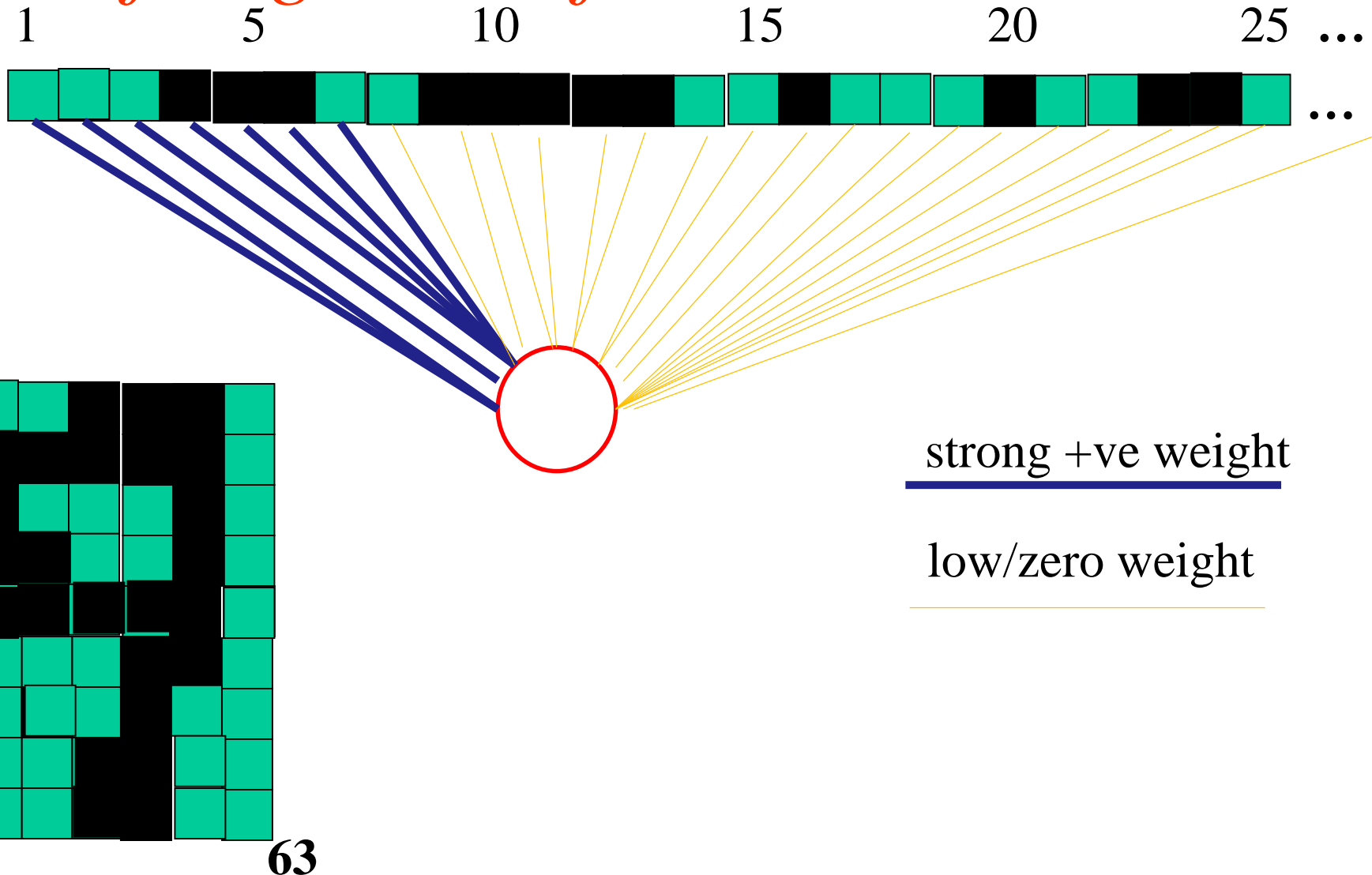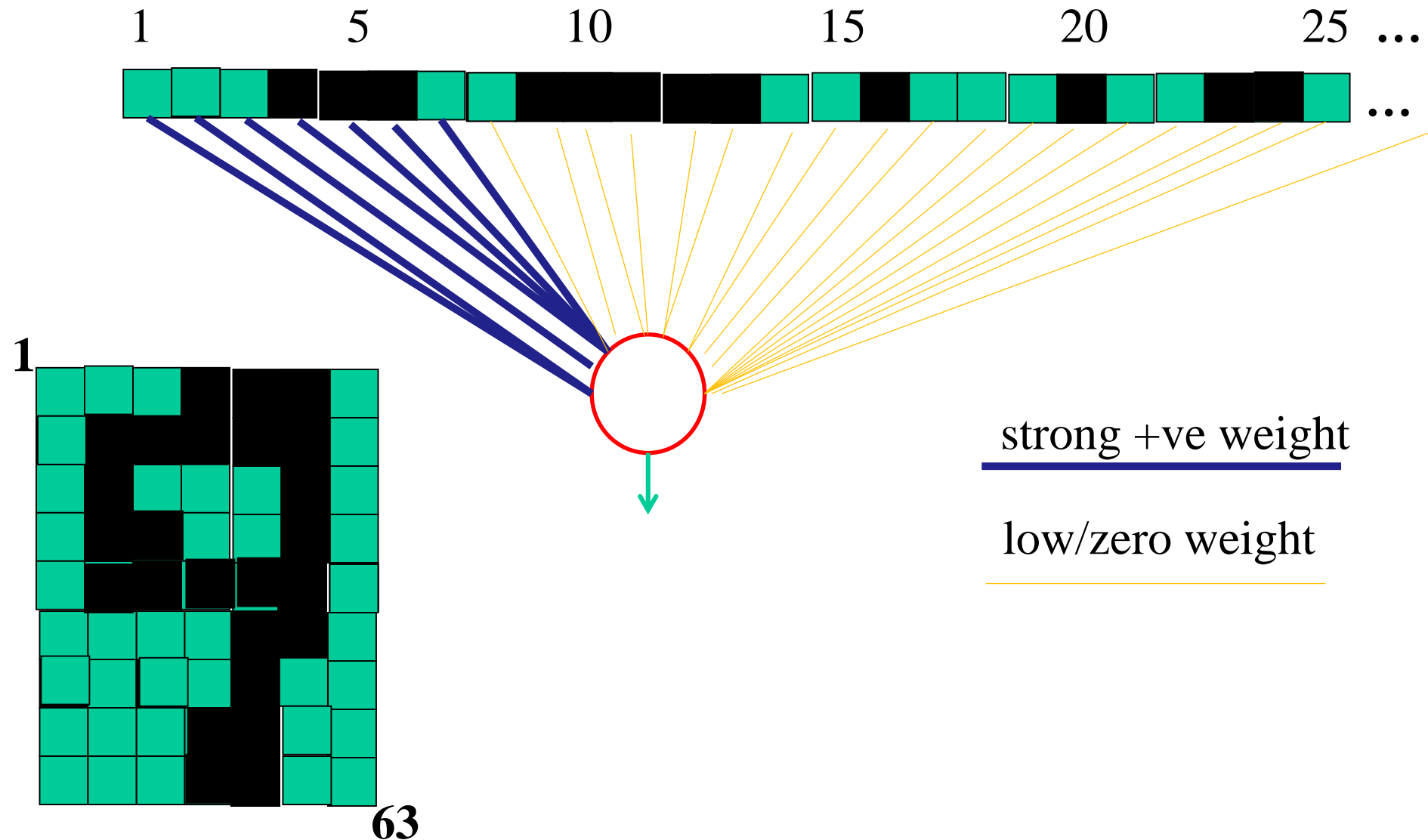Figure 1.2: *Examples of handwritten digits from postal envelopes.*

what is this unit doing?

Input layer

Hidden layer

Hidden layer units become
self-organised feature detectors

1   5   10   15   20   25 ...

strong +ve weight

low/zero weight

What does this unit detect?

What does this unit detect?
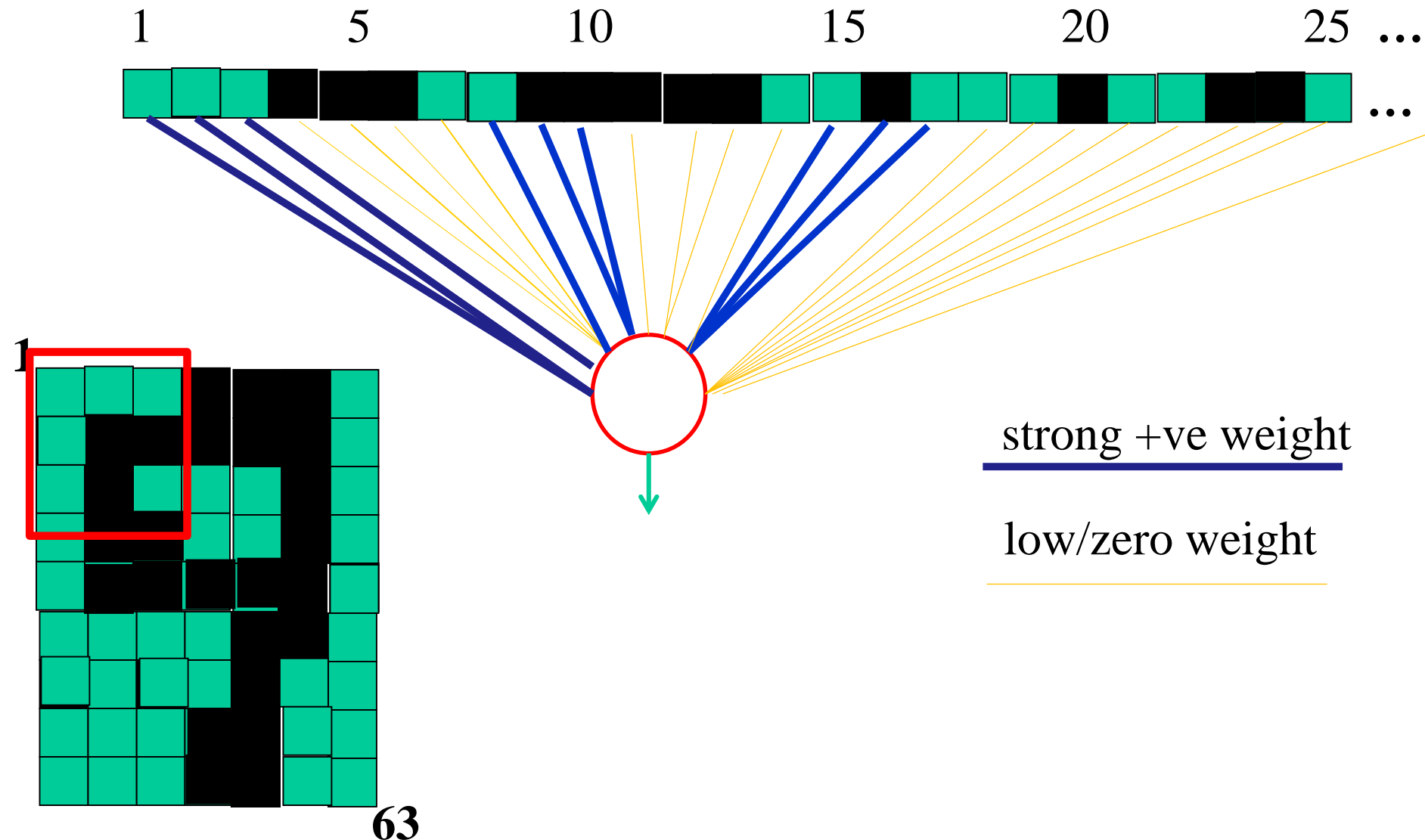
1   5   10   15   20   25 ...

strong +ve weight

low/zero weight

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

1

63

What does this unit detect?

1   5   10   15   20   25 ...

strong +ve weight

low/zero weight

What does this unit detect?

1   5   10   15   20   25 ...

strong +ve weight

low/zero weight

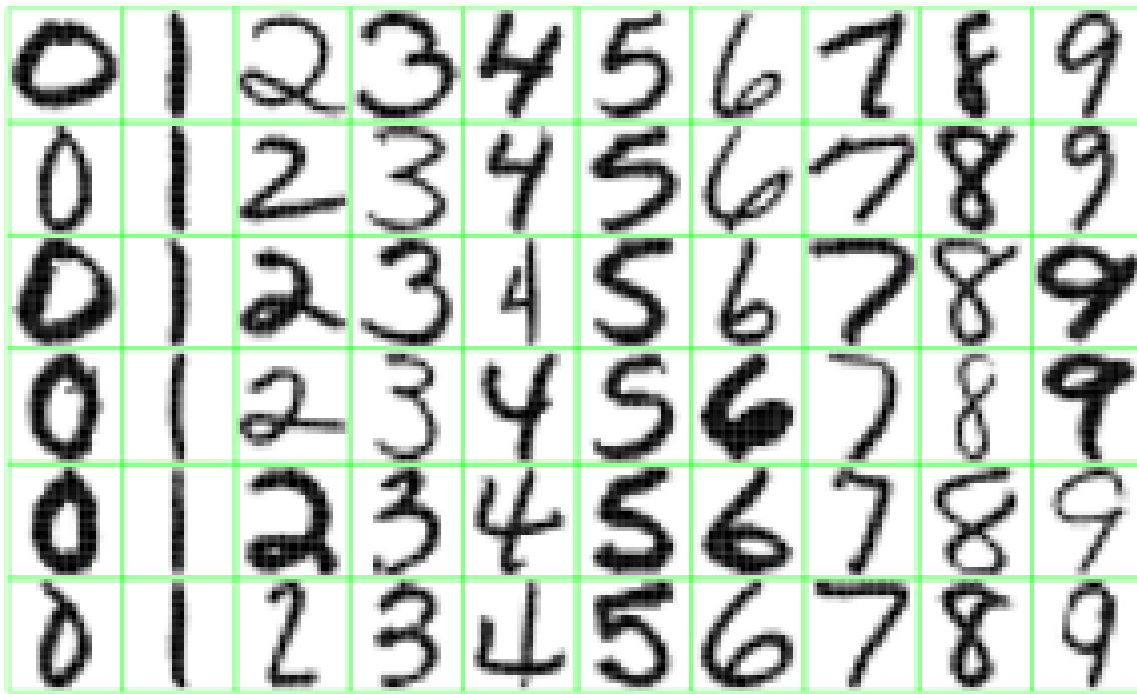Strong signal for a dark area in the top left corner

1

63
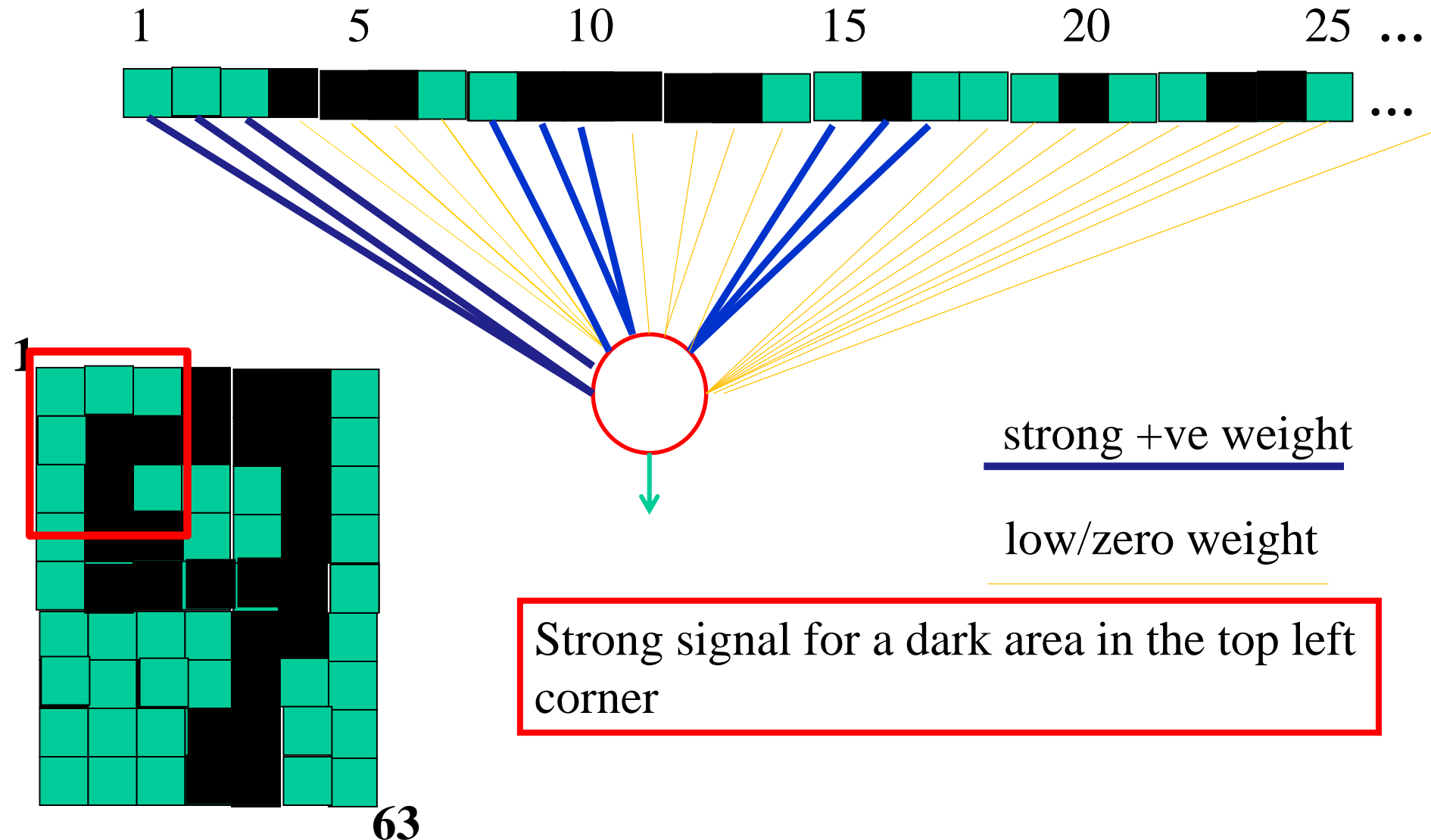
Figure 1.2: *Examples of handwritten digits from U.S. postal envelopes.*

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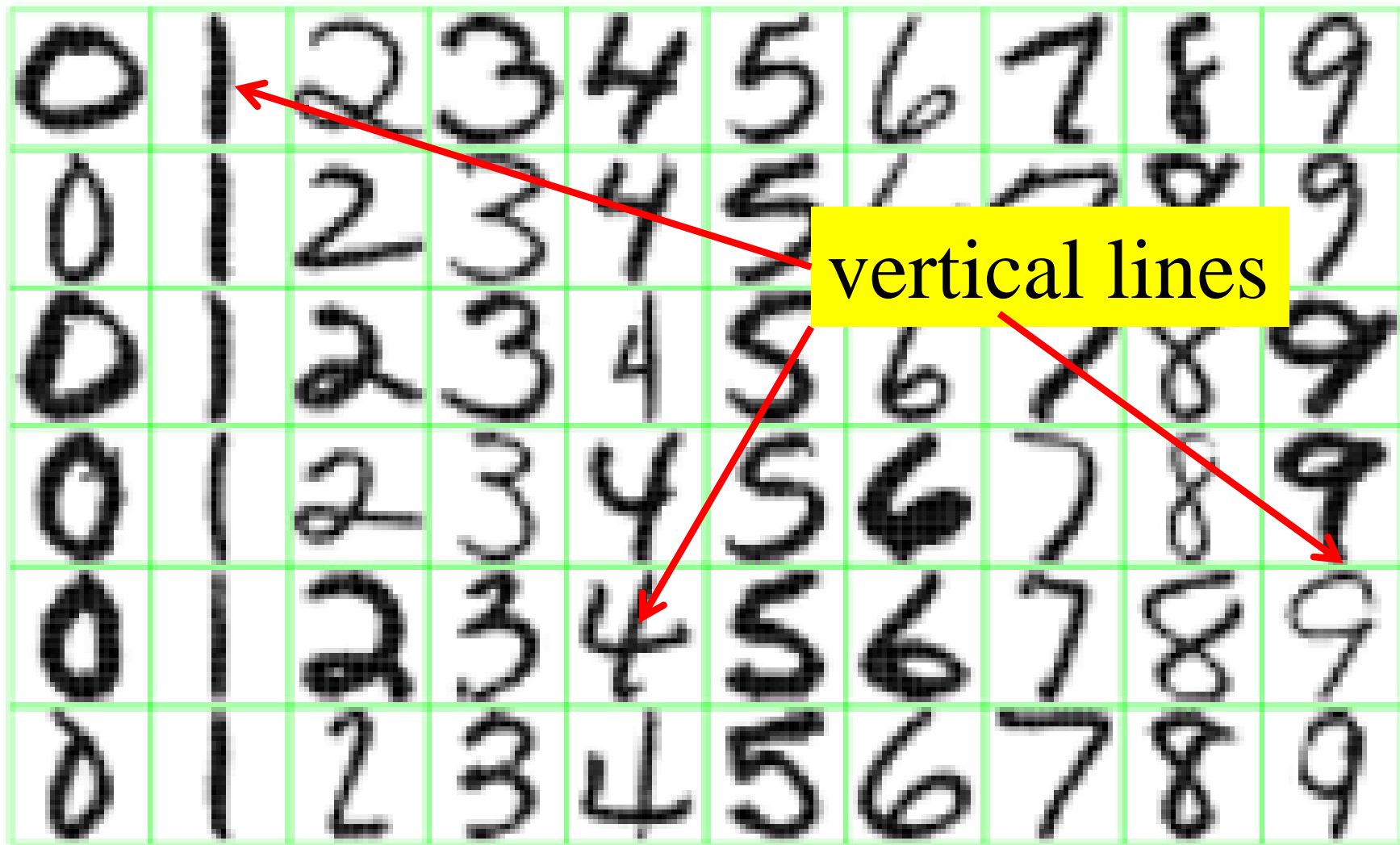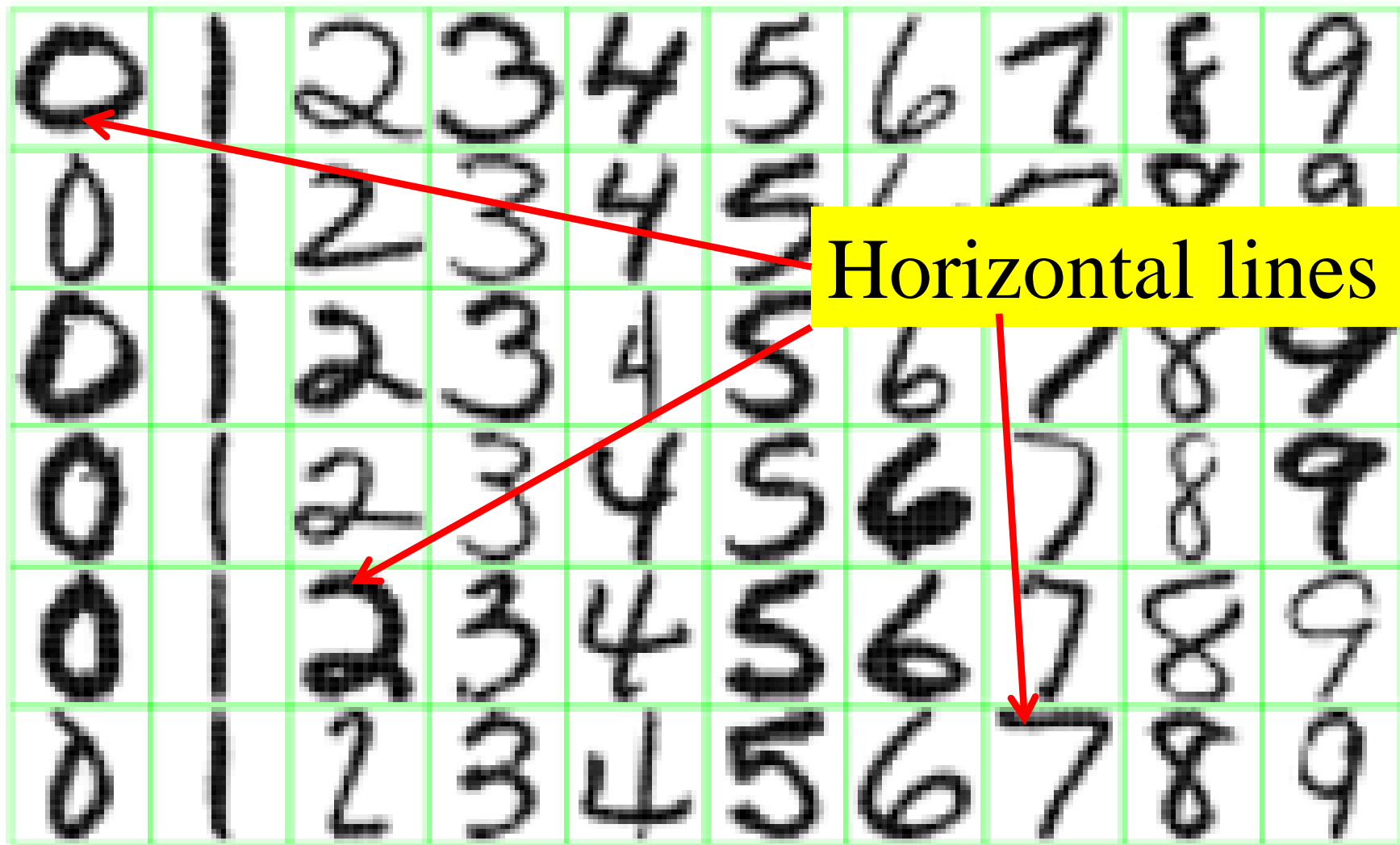
vertical lines

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

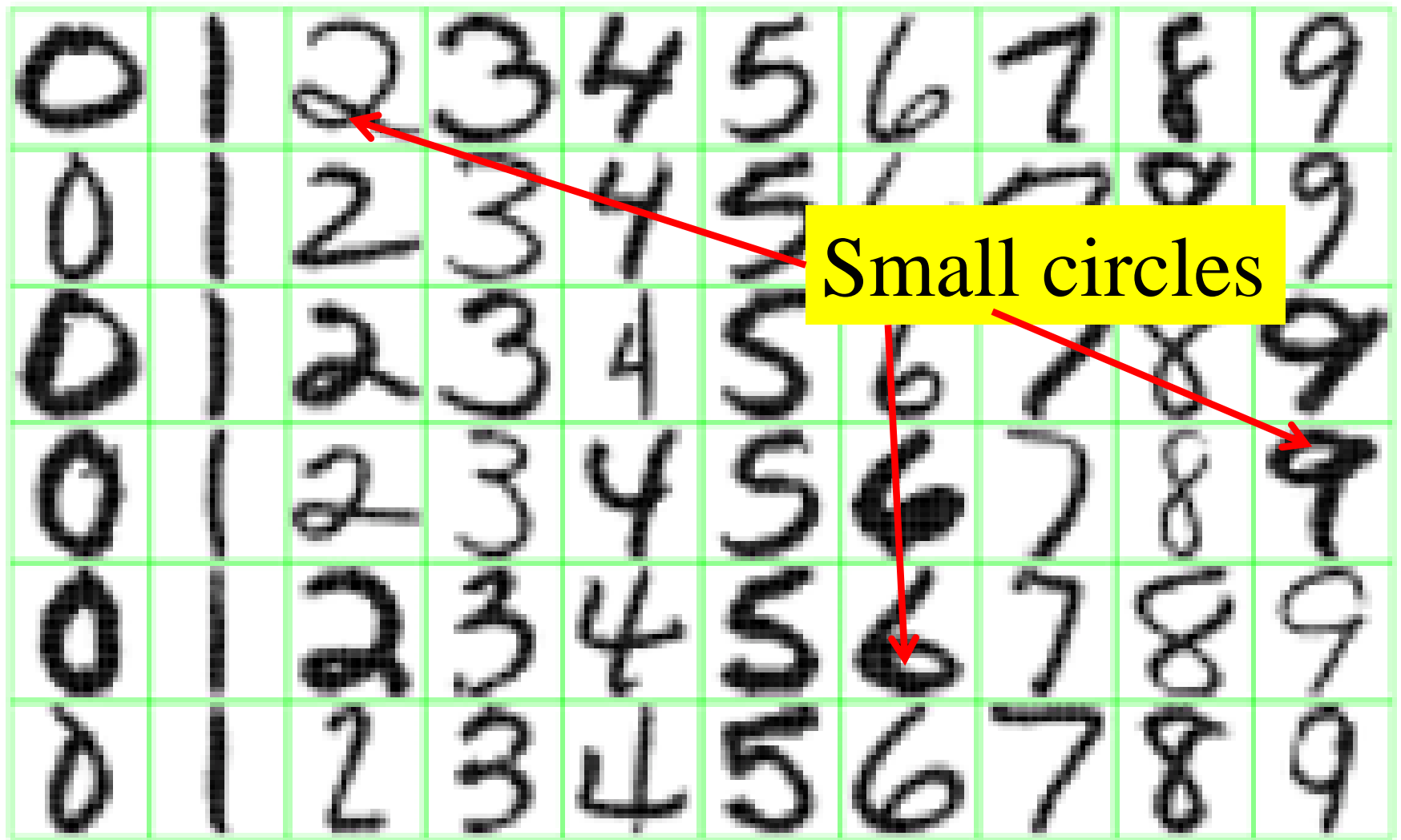Figure 1.2: *Examples of handwritten digits from U.S. postal envelopes.*

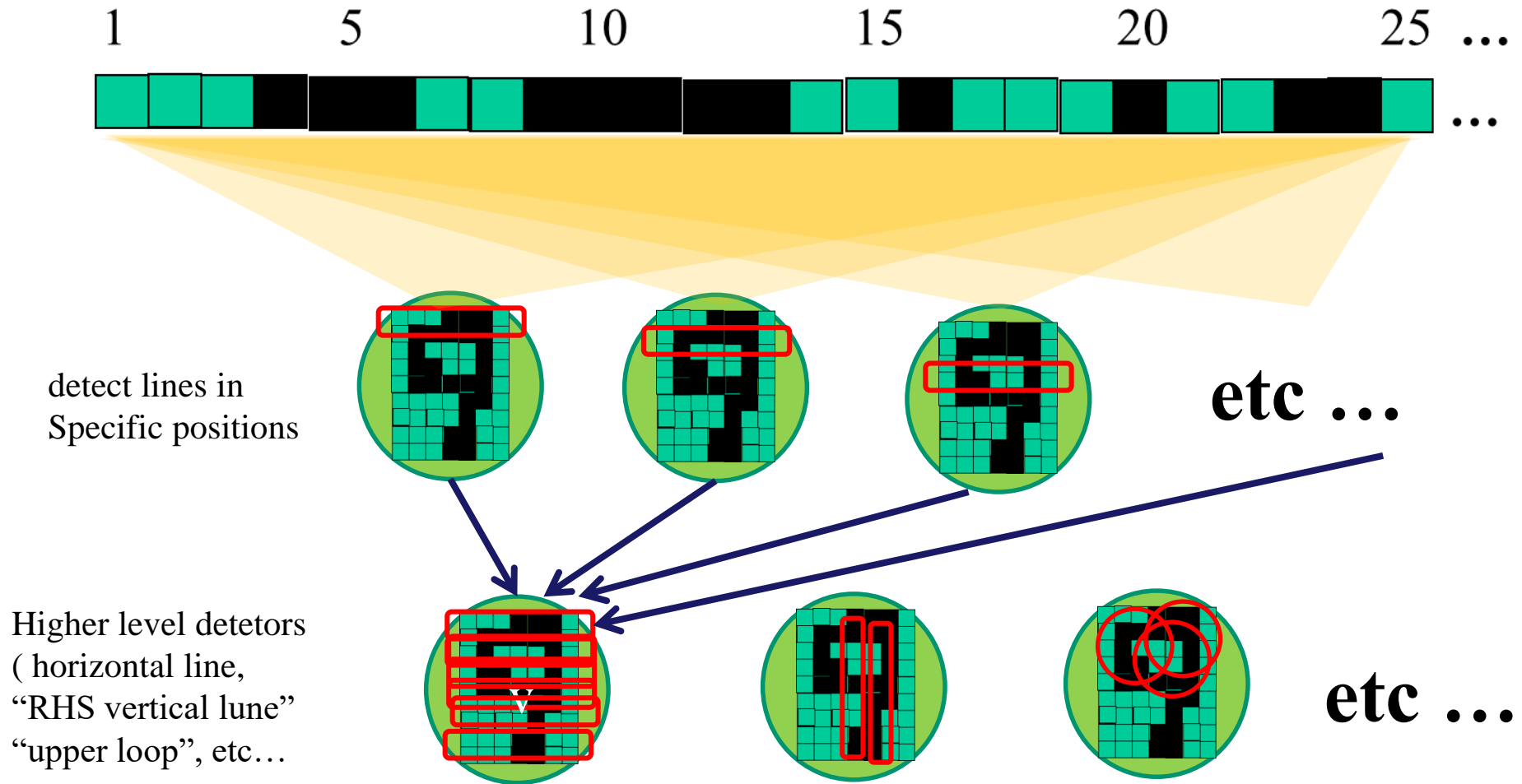Figure 1.2: *Examples of handwritten digits from U.S. postal envelopes.*
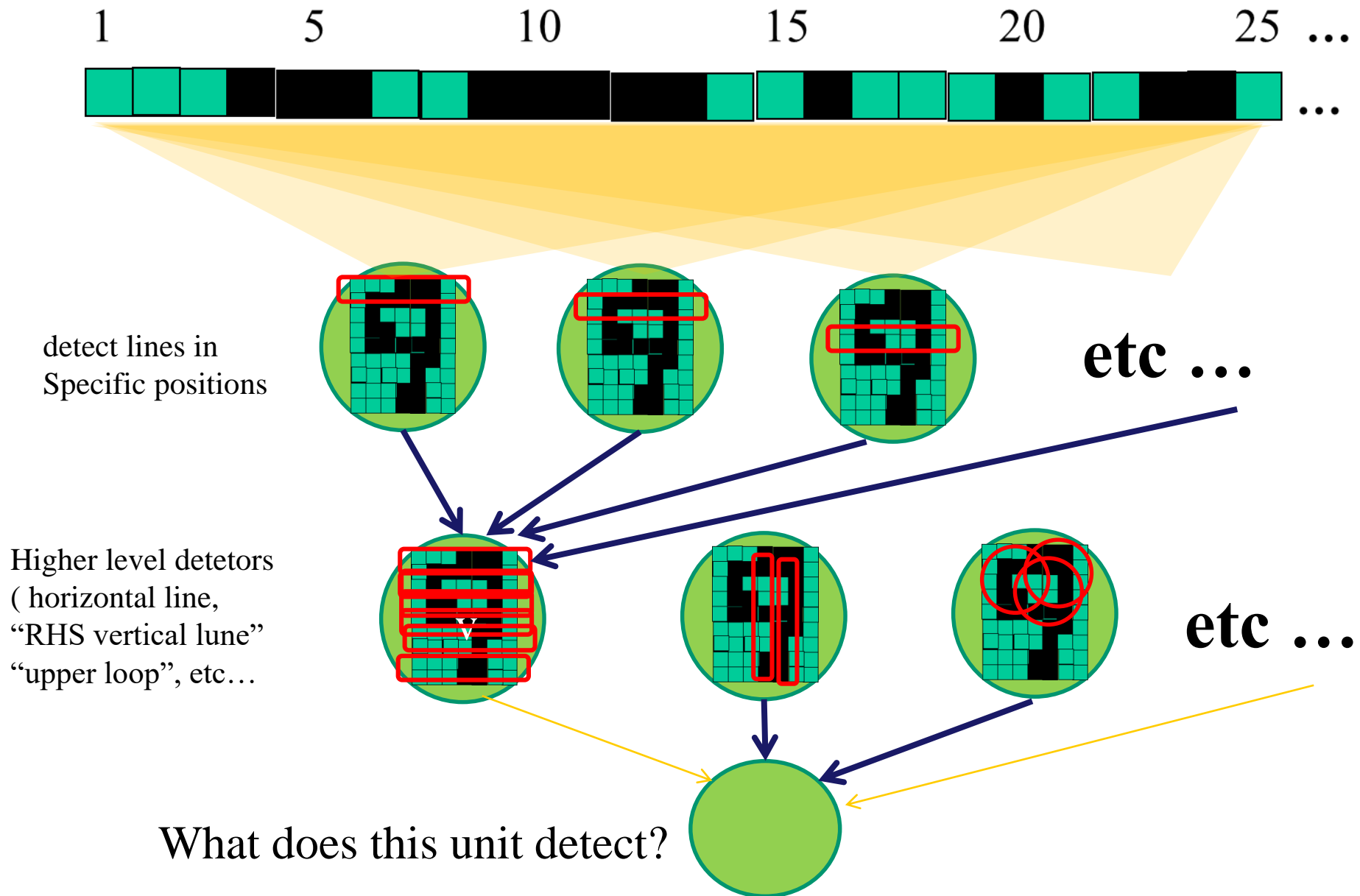
Small circles

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 …

1   5   10   15   20   25 …

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 ...

1   5   10   15   20   25 ...

detect lines in
Specific positions

etc ...

Higher level detetors
(horizontal line,
"RHS vertical lune"
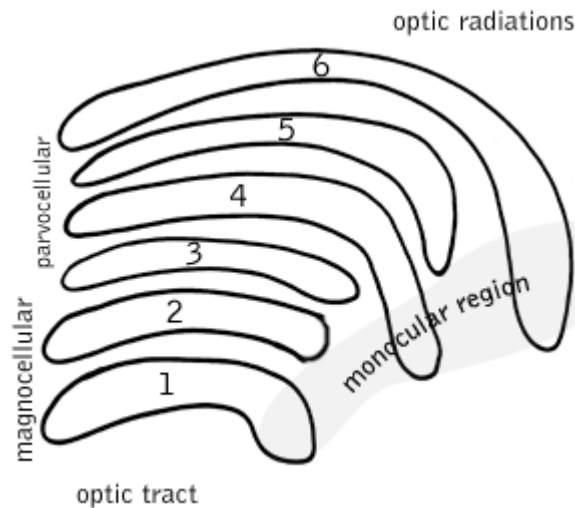"upper loop", etc...

etc ...

What does this unit detect?
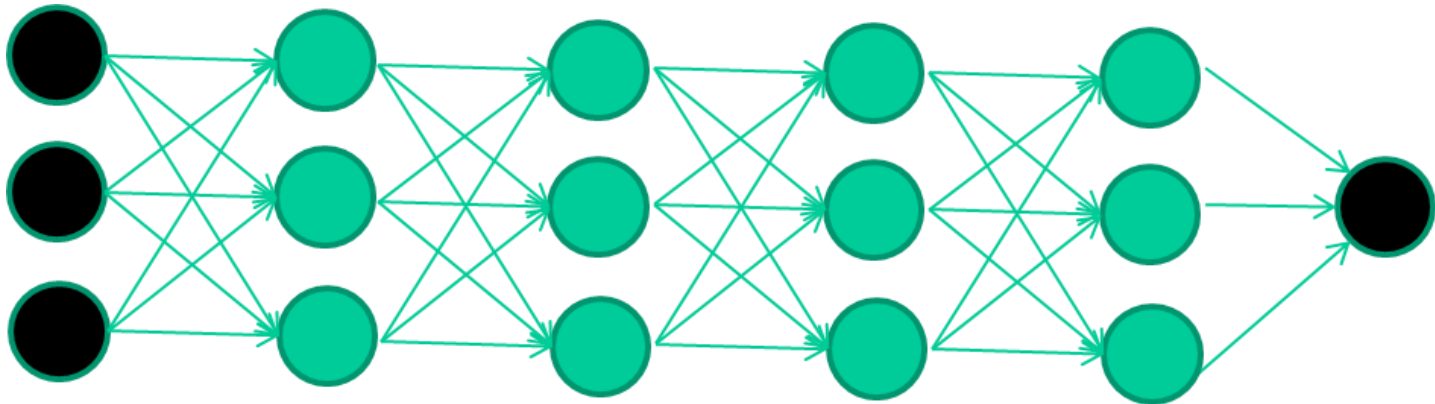
So: *multiple layers make sense*
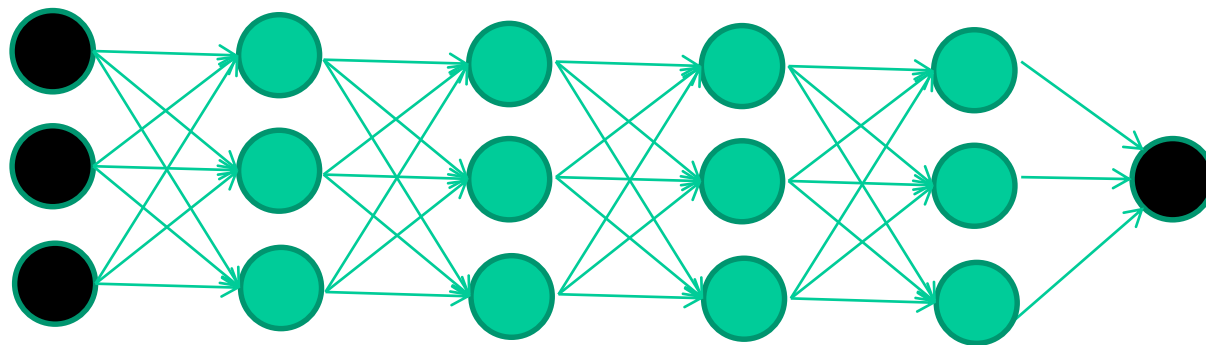
So: *multiple layers make sense*
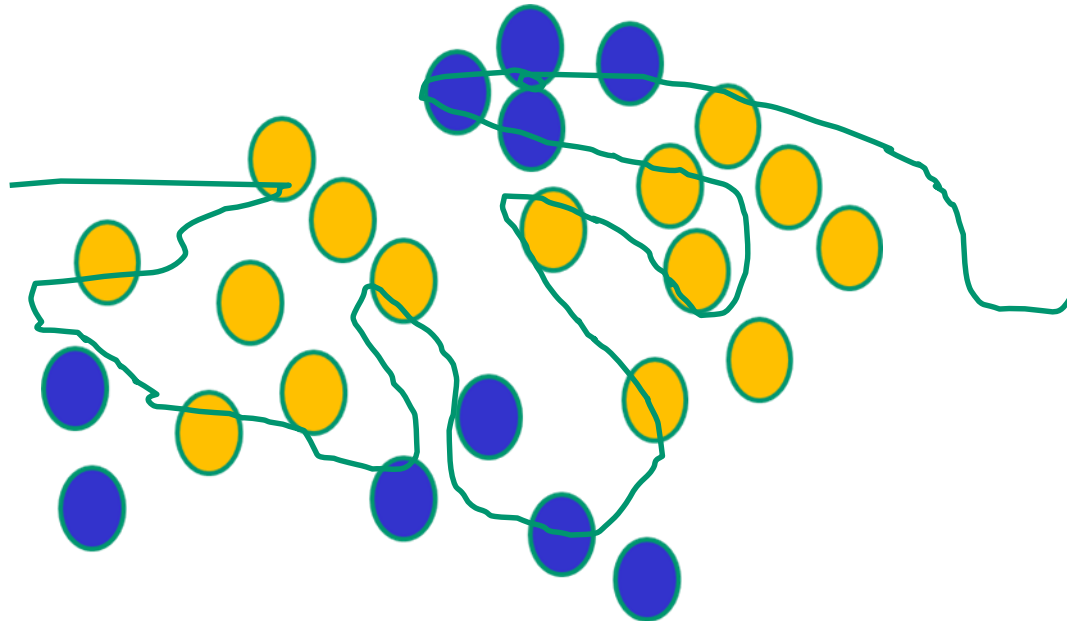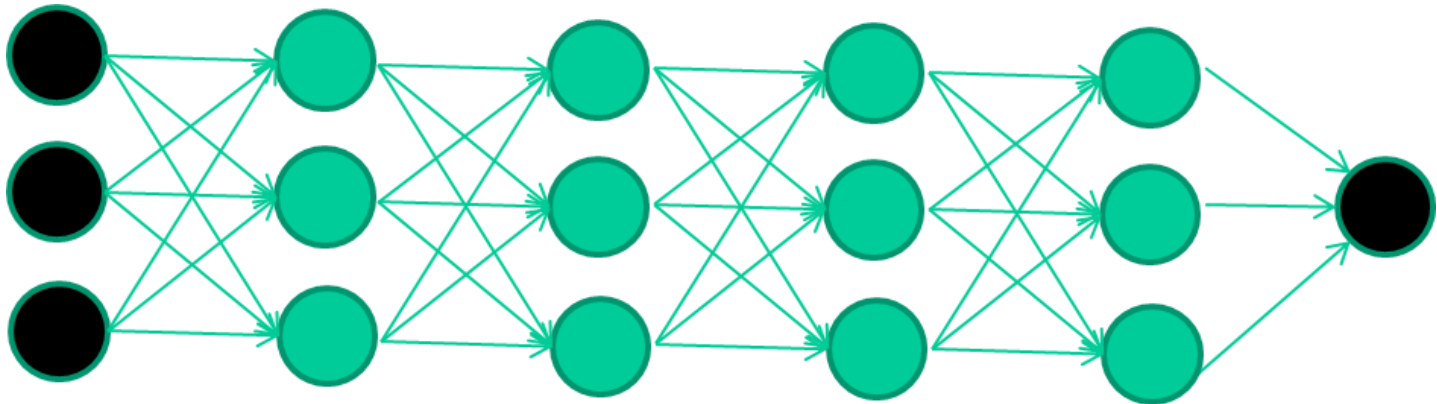
Your brain works that way
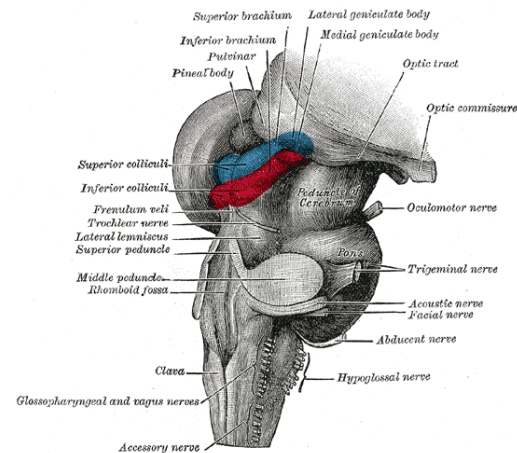
So: *multiple layers make sense*

Many-layer neural network architectures should be capable of learning the true underlying features and 'feature logic', and therefore generalise very well …
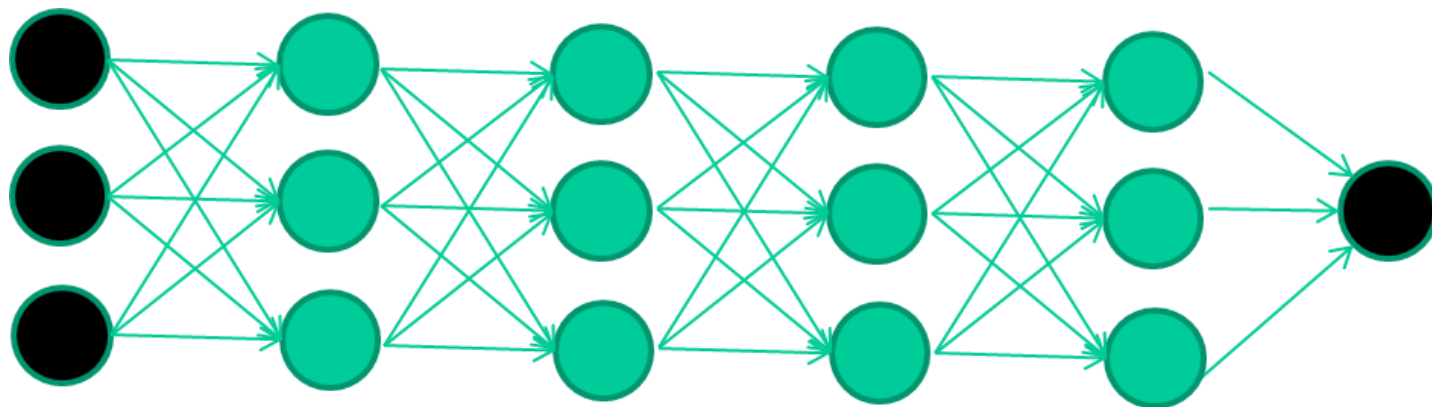
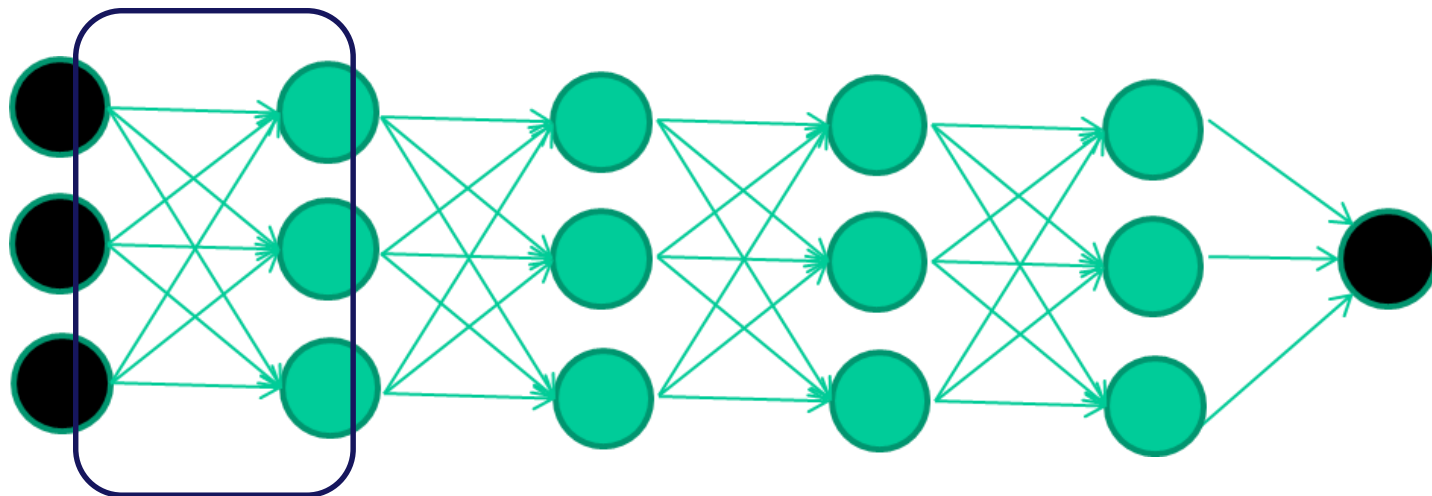But, until very recently, our weight-learning algorithms simply did not work on multi-layer architectures

Along came deep learning …
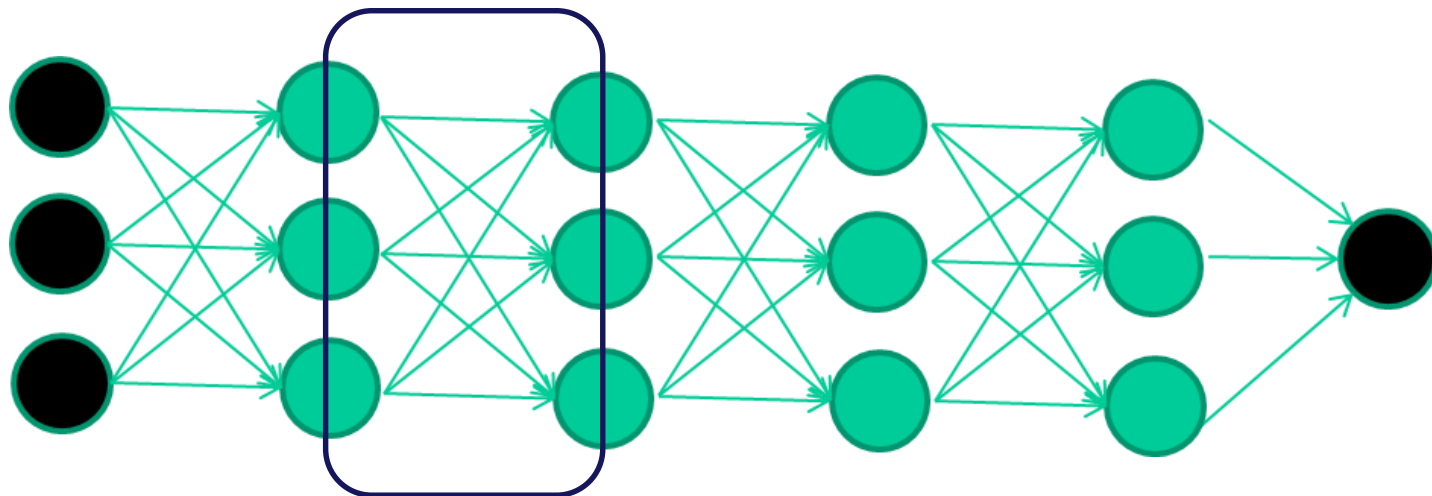
The new way to train multi-layer NNs…

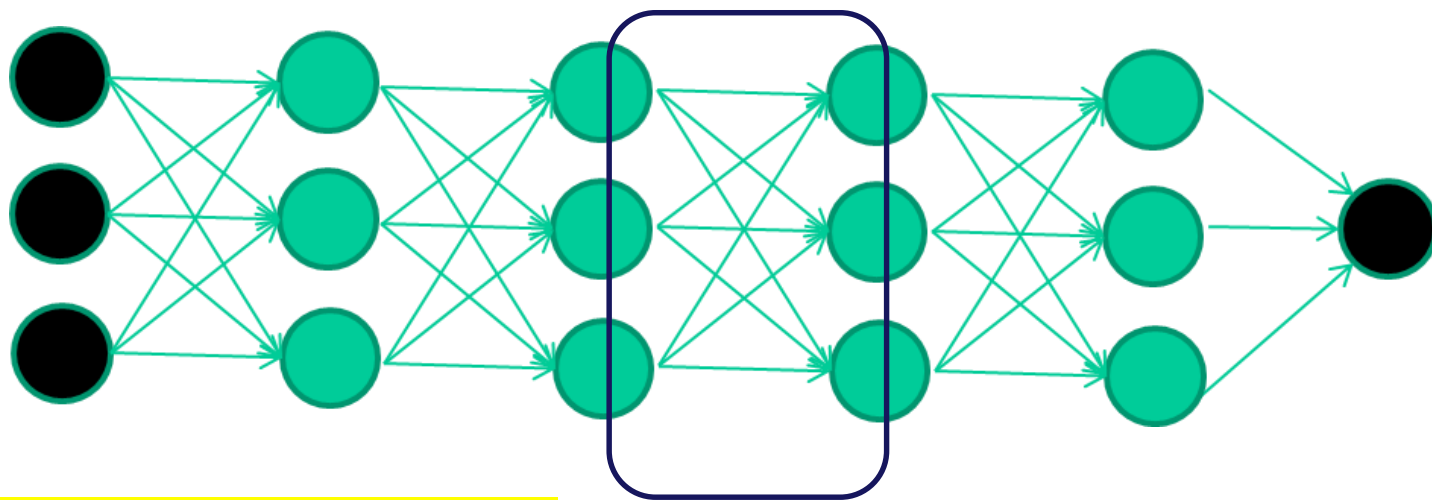# The new way to train multi-layer NNs…

Train **this** layer first

The new way to train multi-layer NNs…

Train **this** layer first

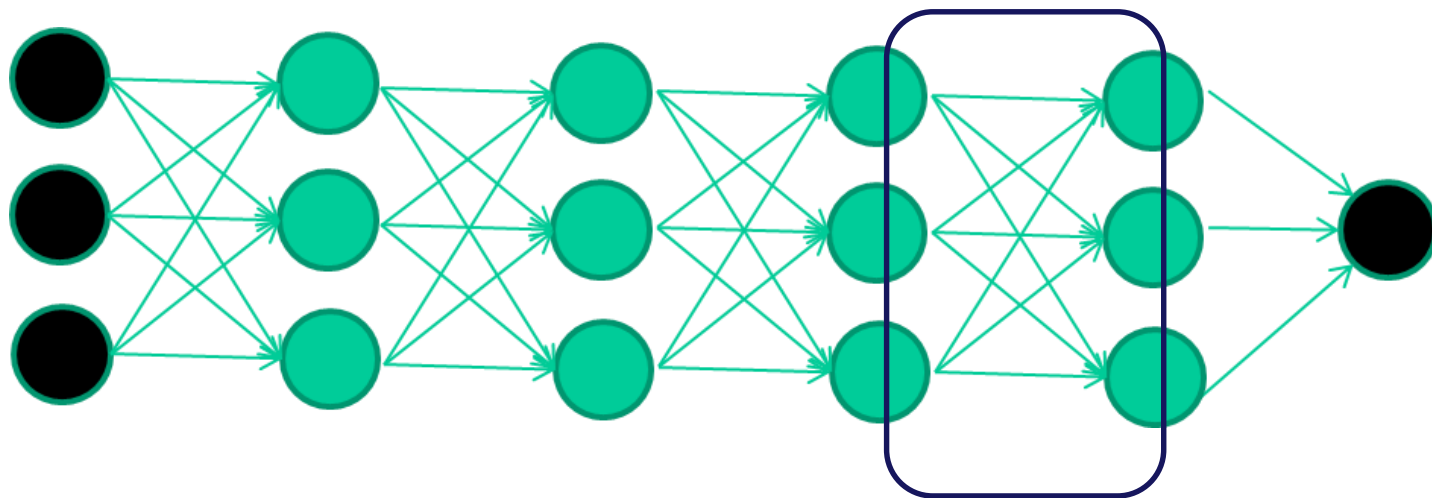then **this** layer

The new way to train multi-layer NNs…

Train **this** layer first

then **this** layer

then **this** layer

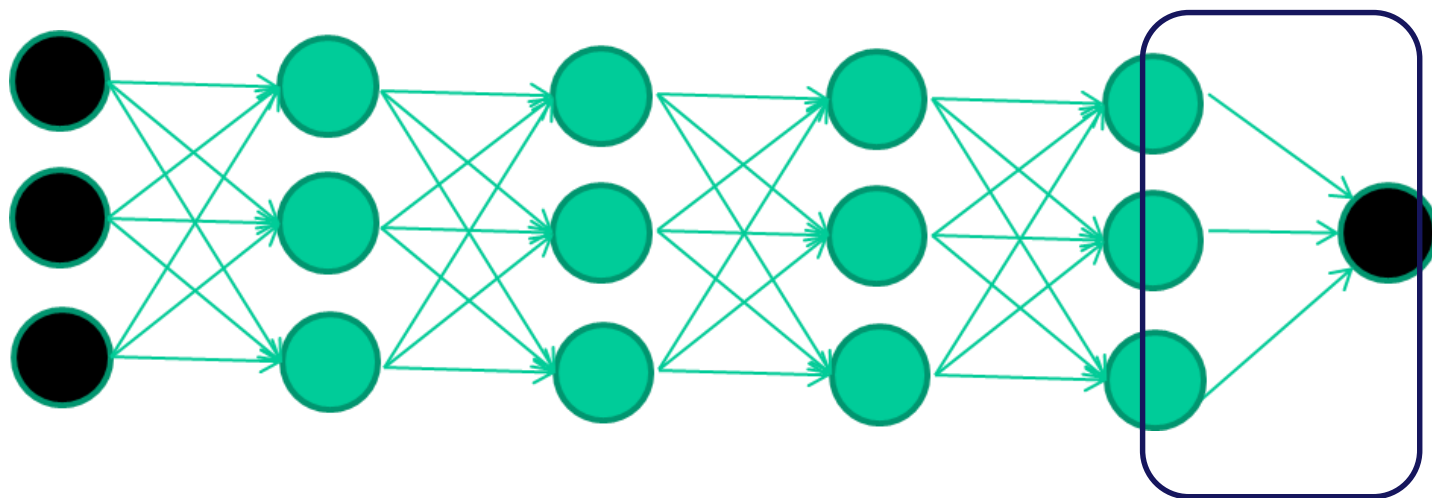# The new way to train multi-layer NNs…

Train **this** layer first

then **this** layer

then **this** layer

then **this** layer

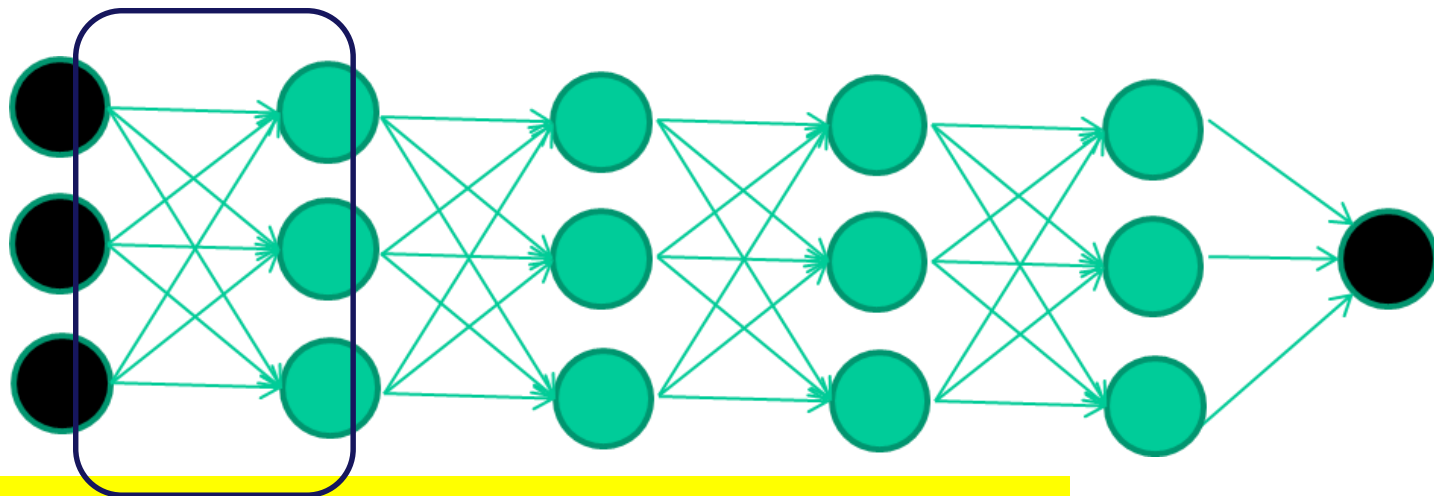# The new way to train multi-layer NNs…

Train **this** layer first

then **this** layer

then **this** laver

then **this** laver

finally **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 <u>reproduce the input</u>
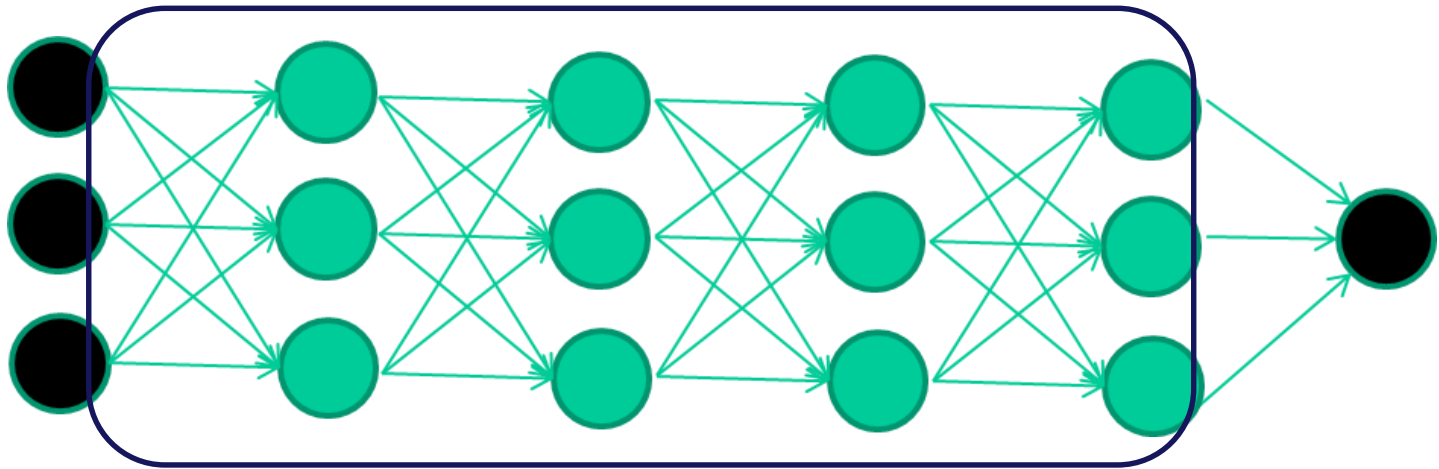

output ---

decode

hidden ---

encode

input ---

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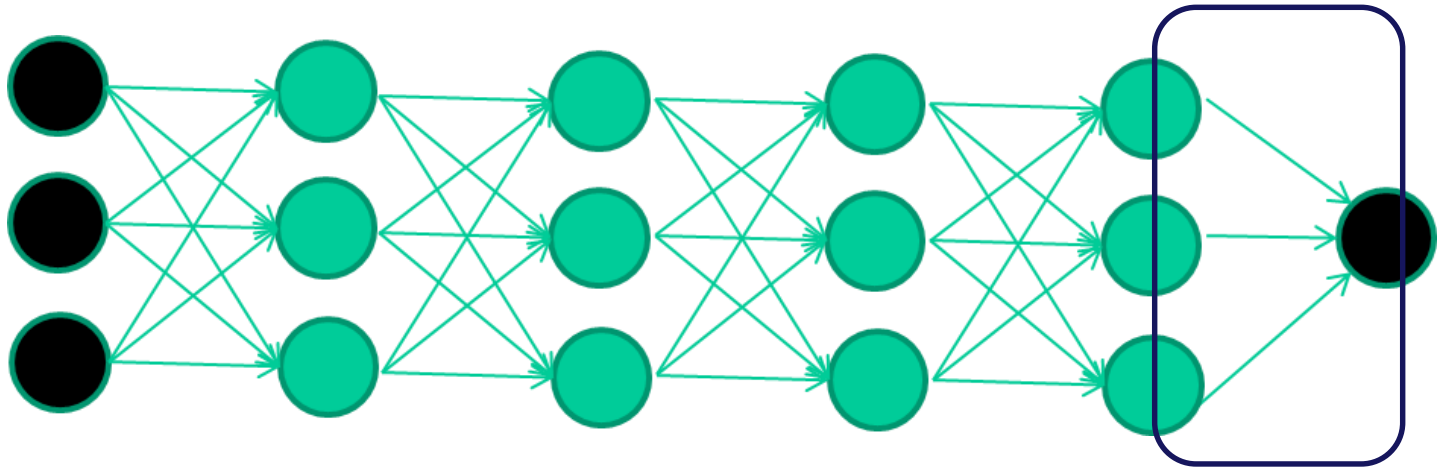

output

hidden

input

↑ decode
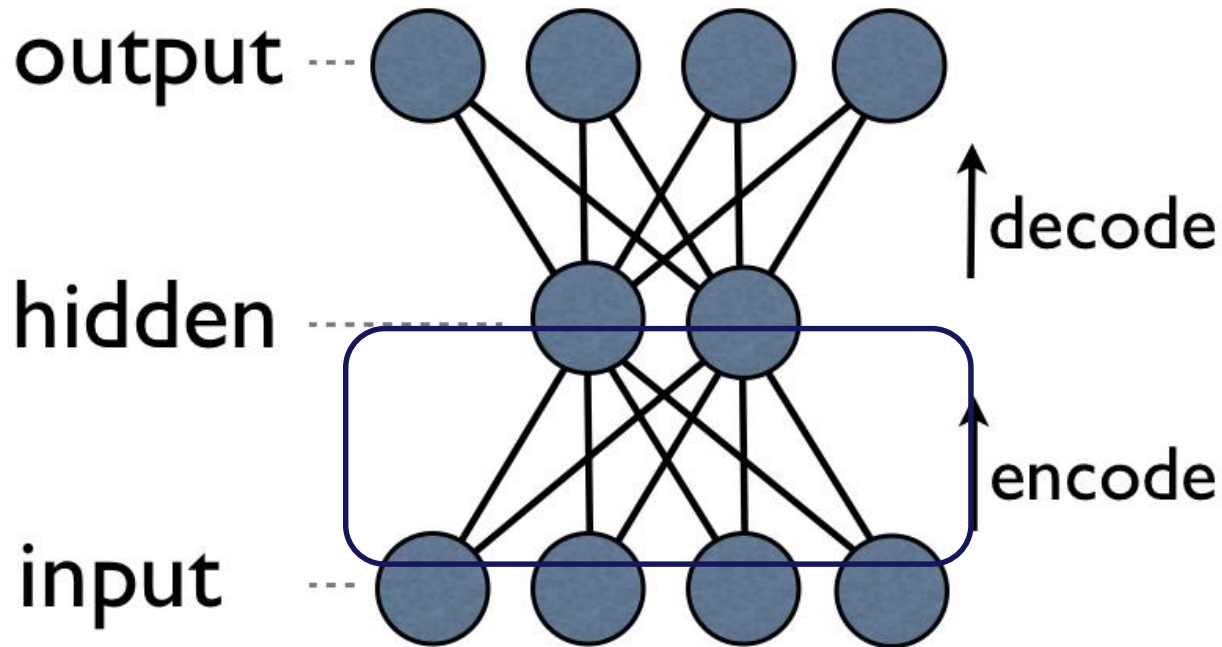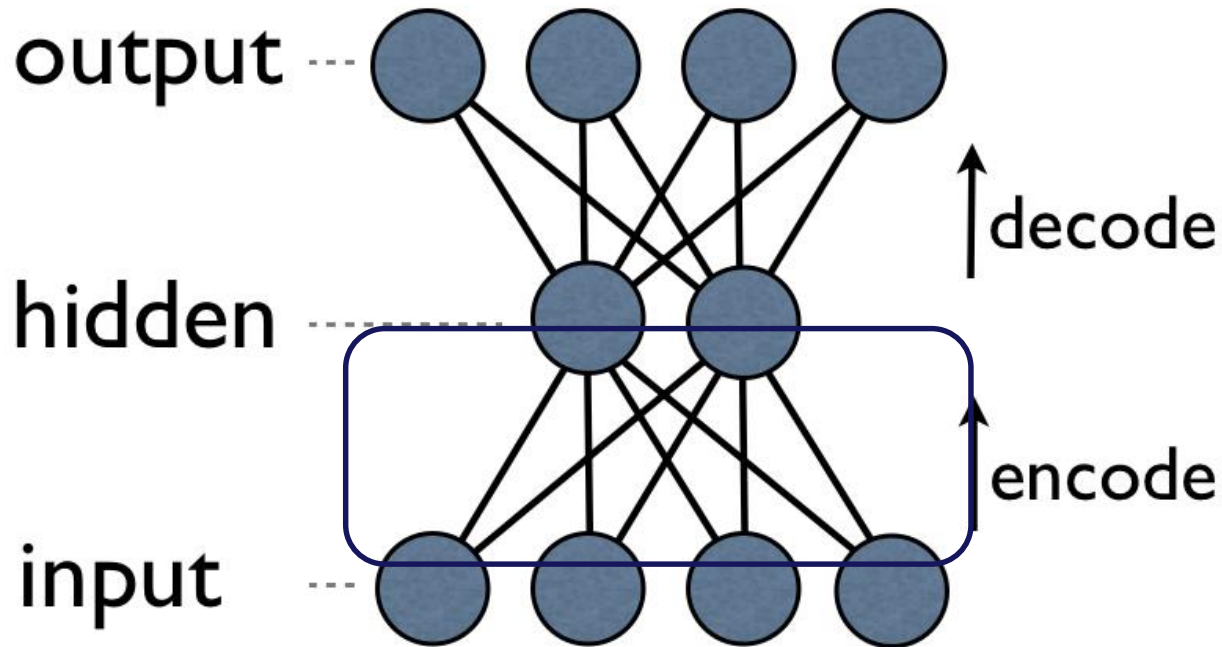
↑ encode

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 …