

# an introduction to: Deep Learning

Based on Lecture Notes by David Corne

<http://www.macs.hw.ac.uk/~dwcorne/>

And others

<http://www.cs.umd.edu/~djacobs/CMSC733/CNN.pdf>

<http://cs231n.stanford.edu>

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.

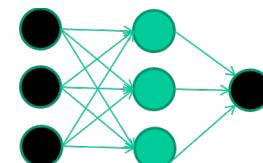
hmmm... OK, but:

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

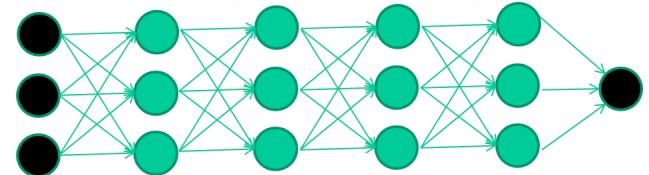
hmmm... OK, but:

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

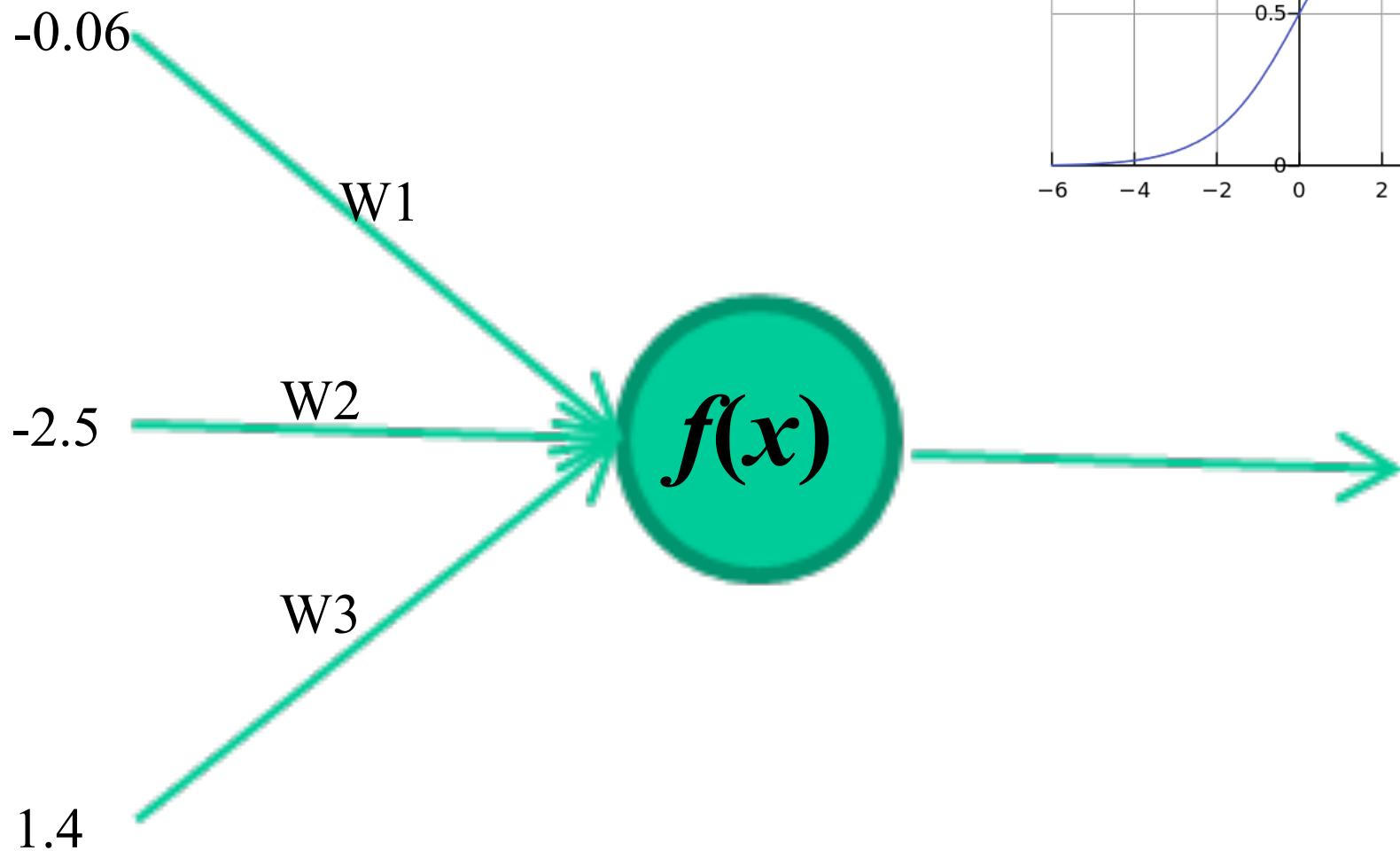
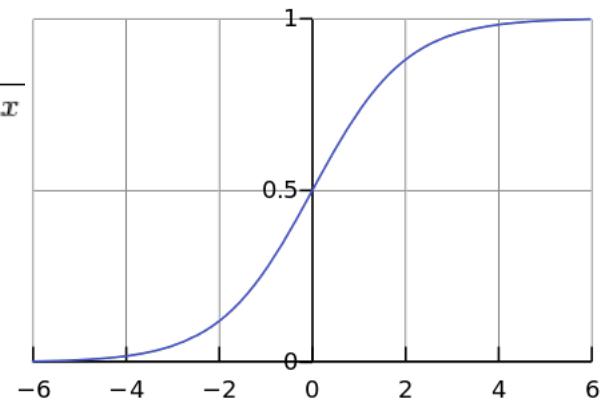


what's new is: algorithms for training many-layer networks

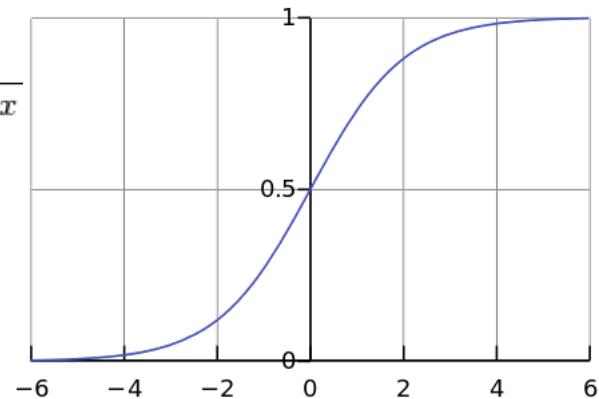
# longer answers

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

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



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



-0.06

2.7

-2.5

-8.6

1.4

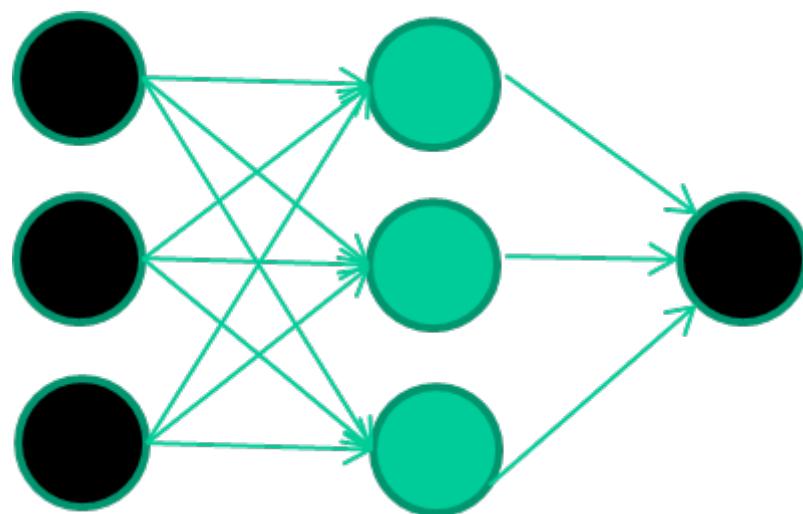
$f(x)$

0.002

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

## *A dataset*

<i>Fields</i>	<i>class</i>
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 ...	

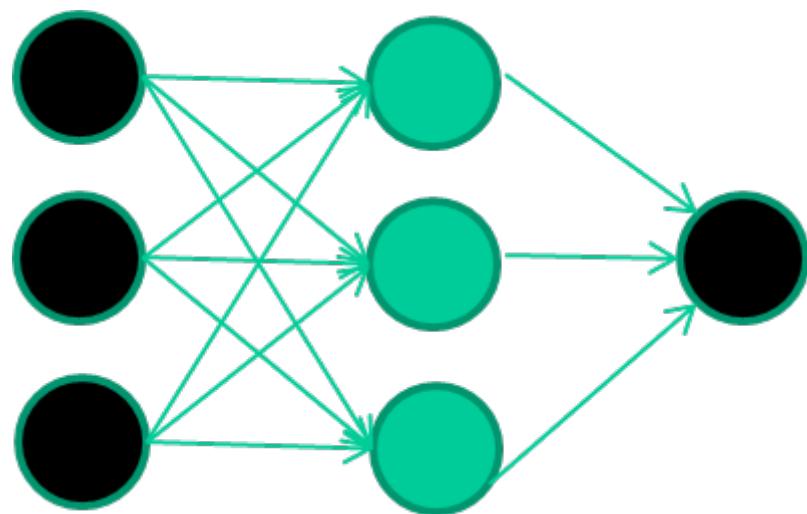


## *Training the neural network*

**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

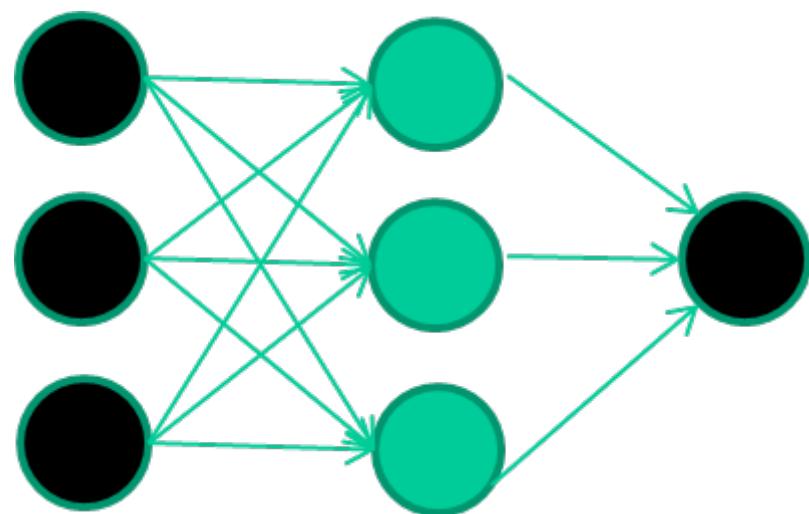
etc ...



*Training data*

<i>Fields</i>	<i>class</i>
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 ...	

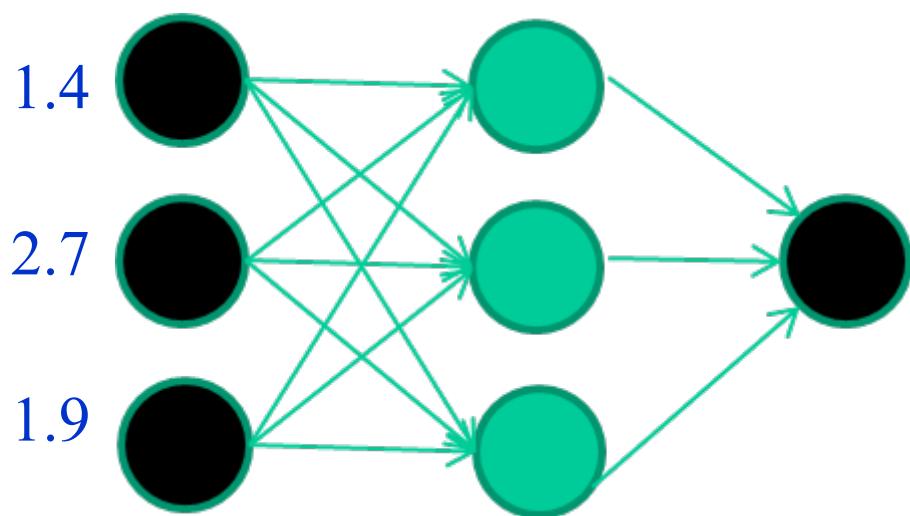
Initialise with random weights



*Training data*

<i>Fields</i>	<i>class</i>		
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
-----	-----	-----	---

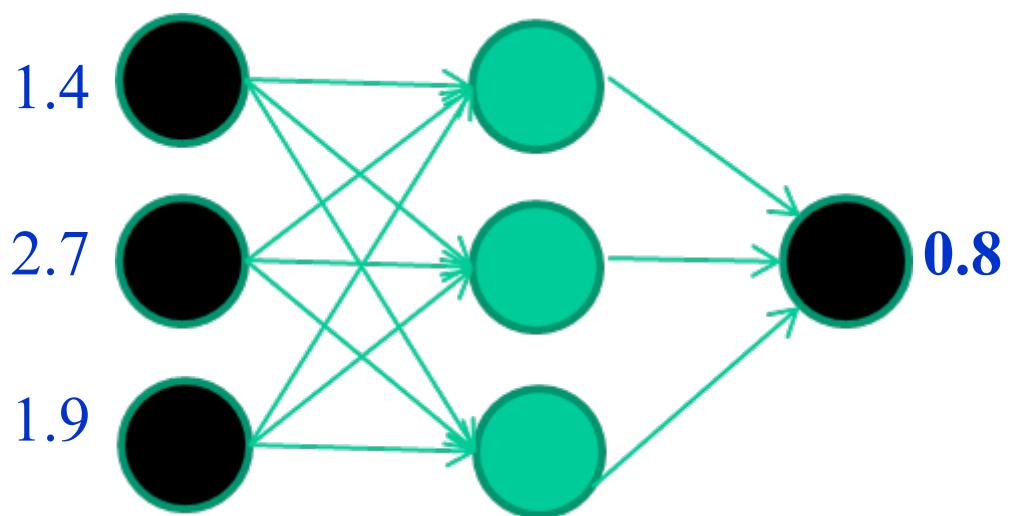
3.8	3.4	3.2	0
-----	-----	-----	---

6.4	2.8	1.7	1
-----	-----	-----	---

4.1	0.1	0.2	0
-----	-----	-----	---

etc ...

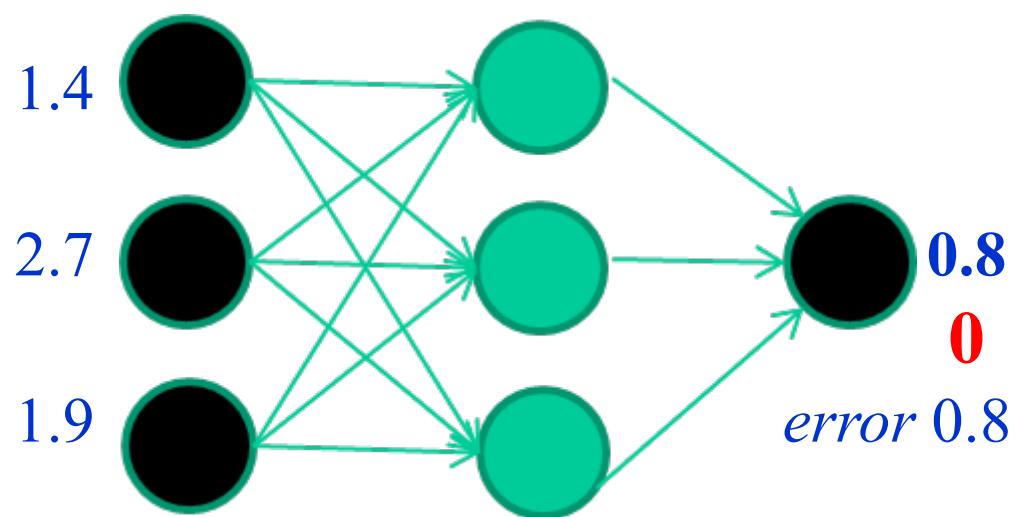
Feed it through to get output



## 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



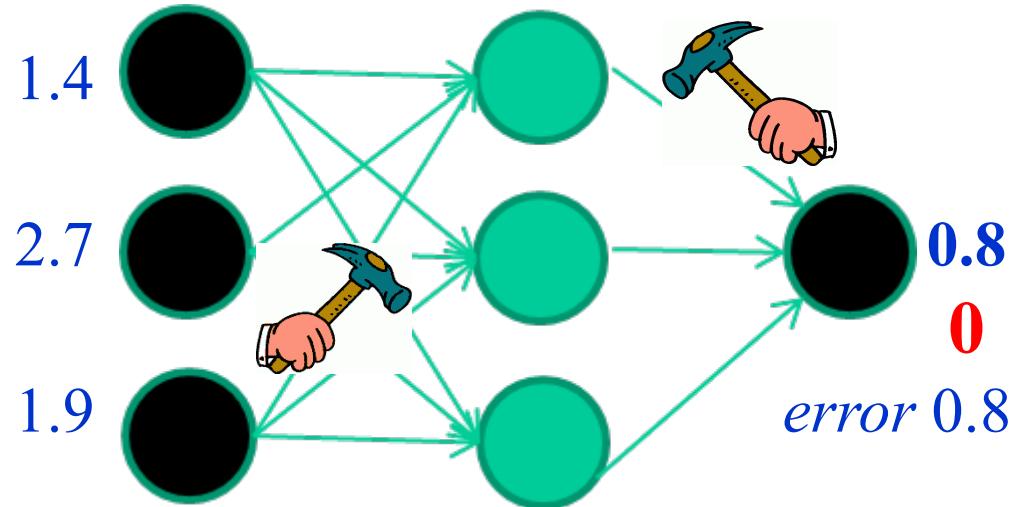
## Training data

### Fields

### class

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 ...	

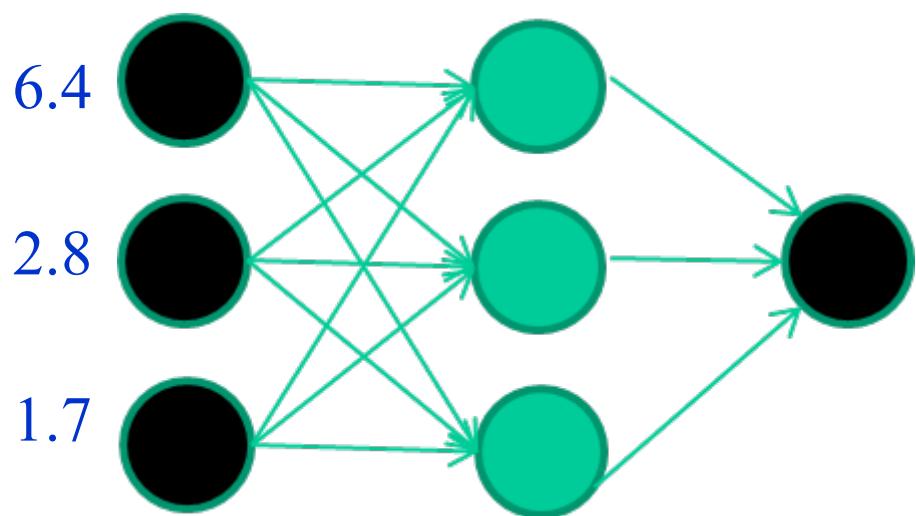
## Adjust weights based on error



*Training data*

<i>Fields</i>	<i>class</i>		
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 ...			

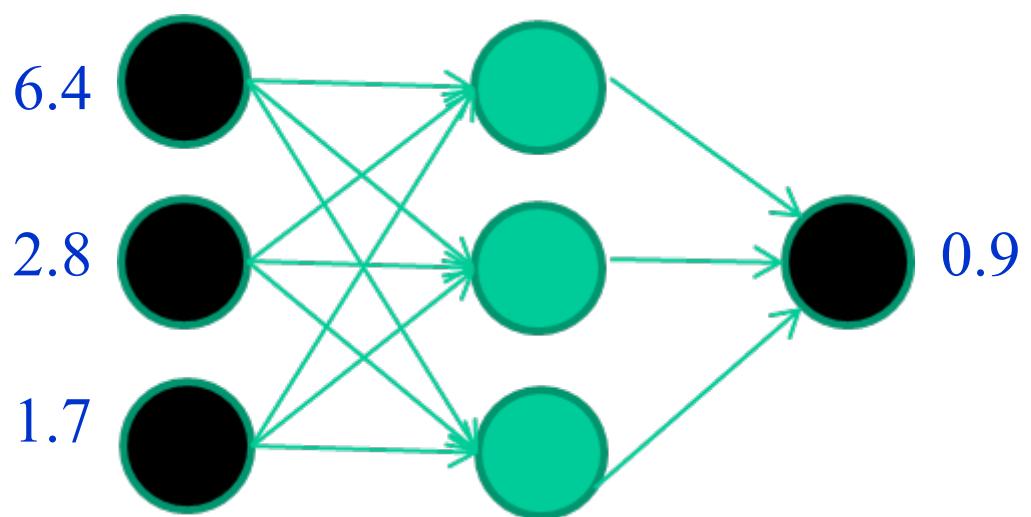
Present a training pattern



*Training data*

<i>Fields</i>	<i>class</i>		
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 ...			

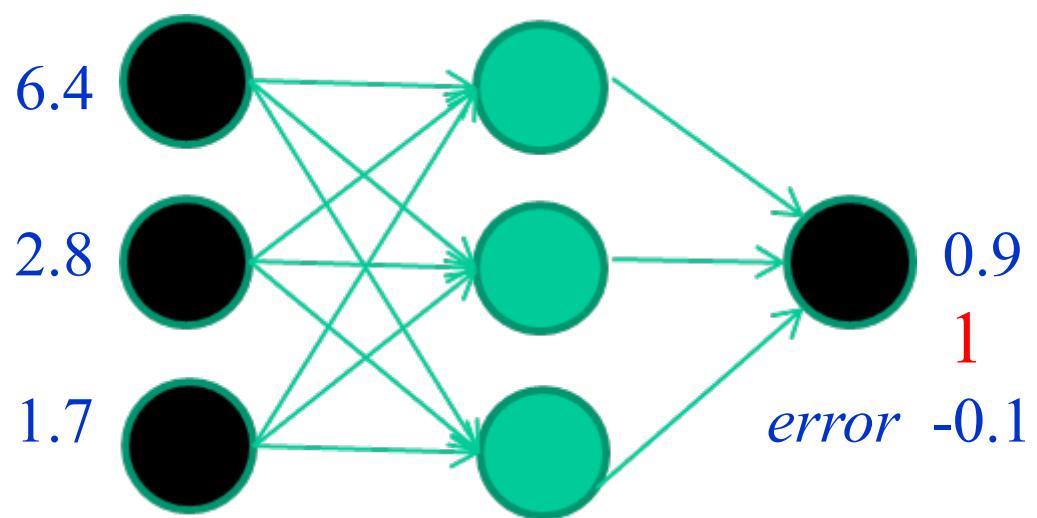
Feed it through to get output



## Training data

<i>Fields</i>	<i>class</i>
1.4 2.7 1.9	0
3.8 3.4 3.2	0
<b>6.4 2.8 1.7</b>	<b>1</b>
4.1 0.1 0.2	0
etc ...	

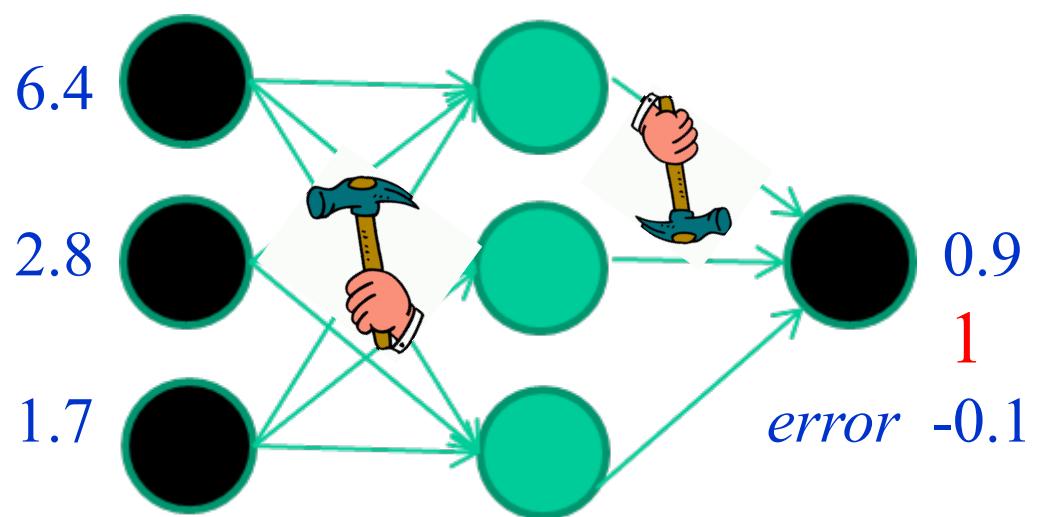
Compare with target output



*Training data*

<i>Fields</i>	<i>class</i>		
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

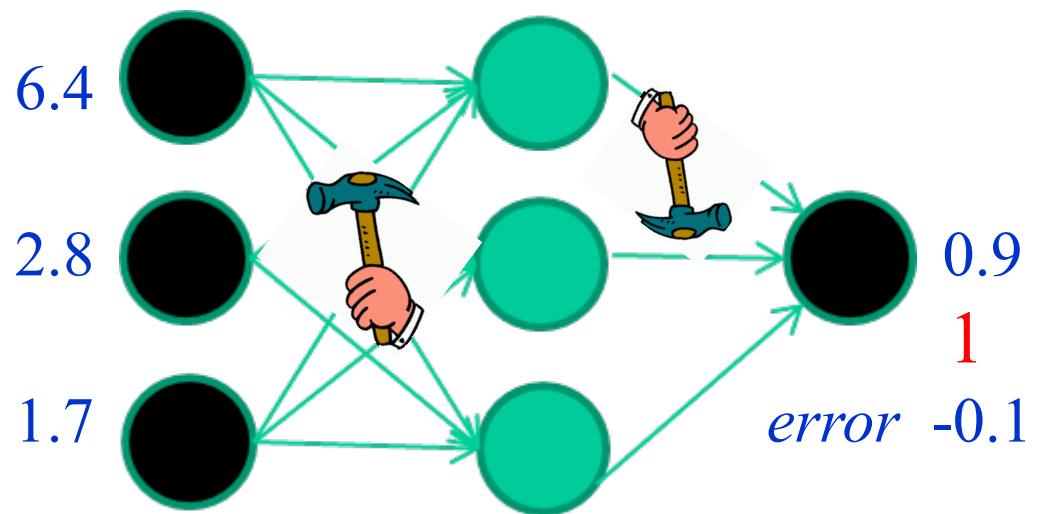


*Training data*

<i>Fields</i>	<i>class</i>		
1.4	2.7	1.9	0
3.8	3.4	3.2	0
<b>6.4</b>	<b>2.8</b>	<b>1.7</b>	<b>1</b>
4.1	0.1	0.2	0

etc ...

**And so on ....**

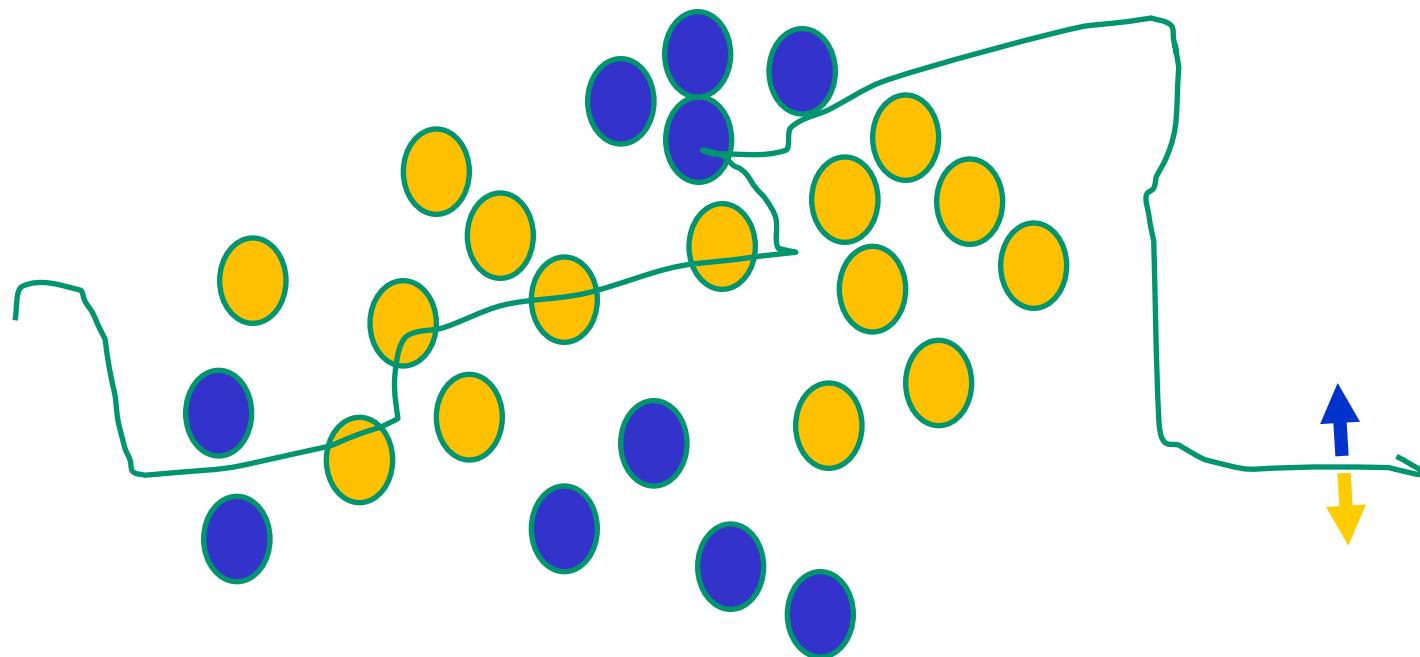


Repeat this thousands, maybe millions of times – each time taking a random training instance, and making slight weight adjustments

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

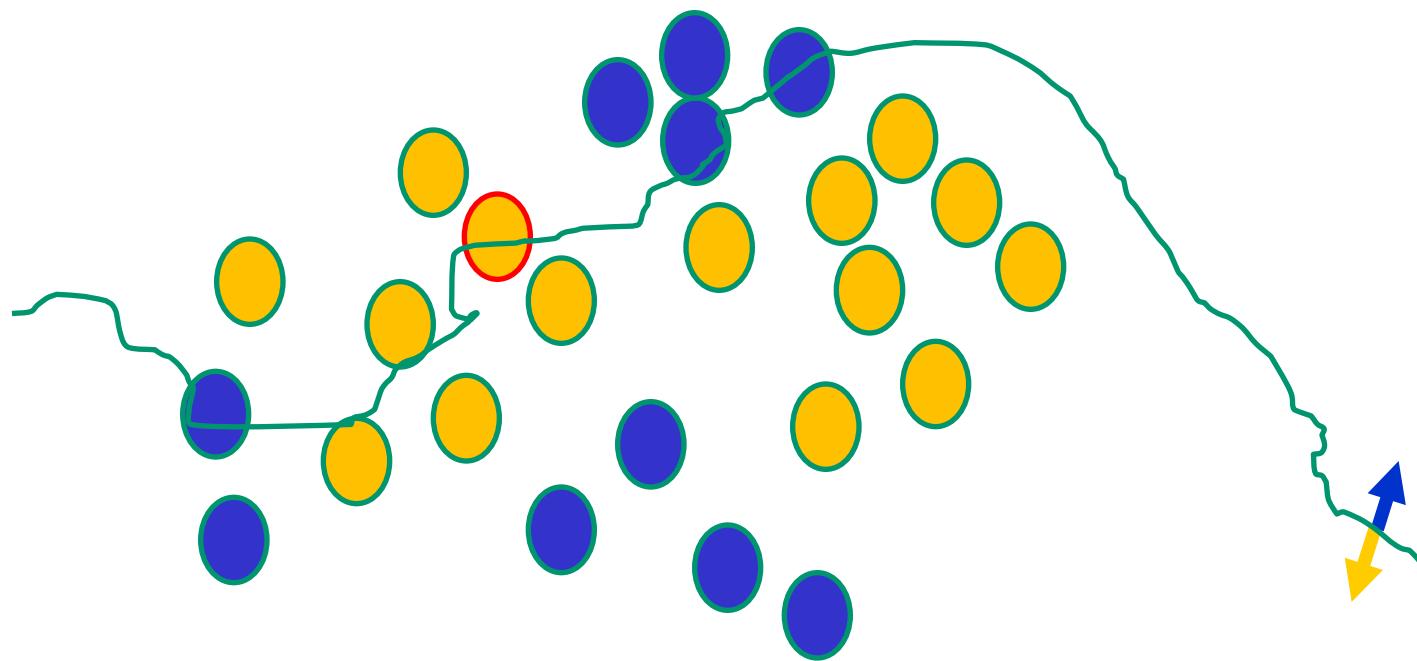
# The decision boundary perspective...

Initial random weights



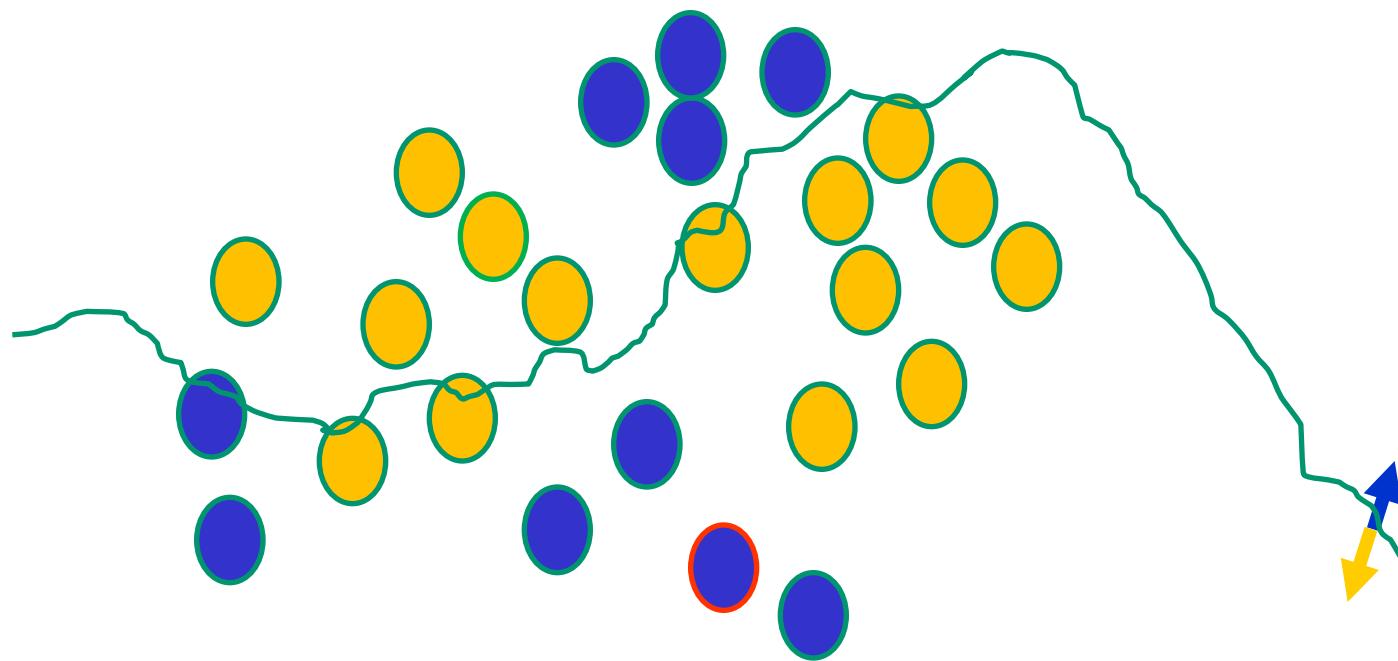
# The decision boundary perspective...

Present a training instance / adjust the weights



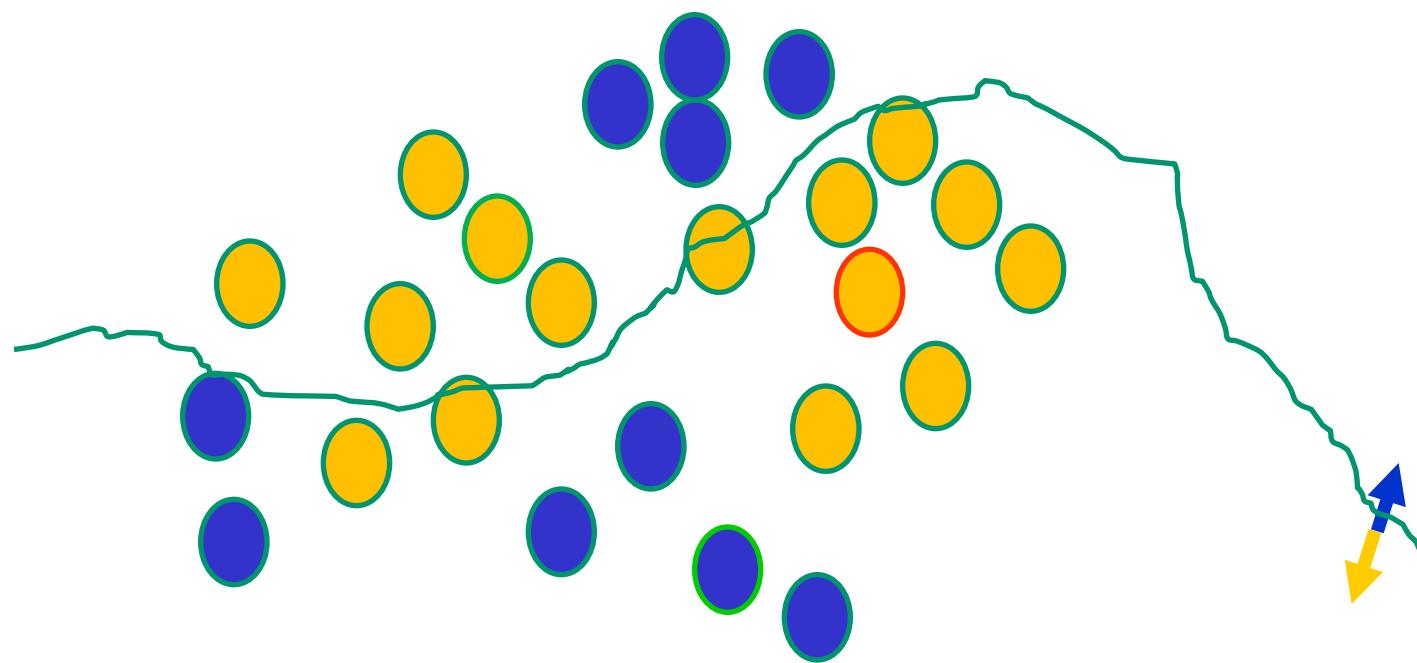
# The decision boundary perspective...

Present a training instance / adjust the weights



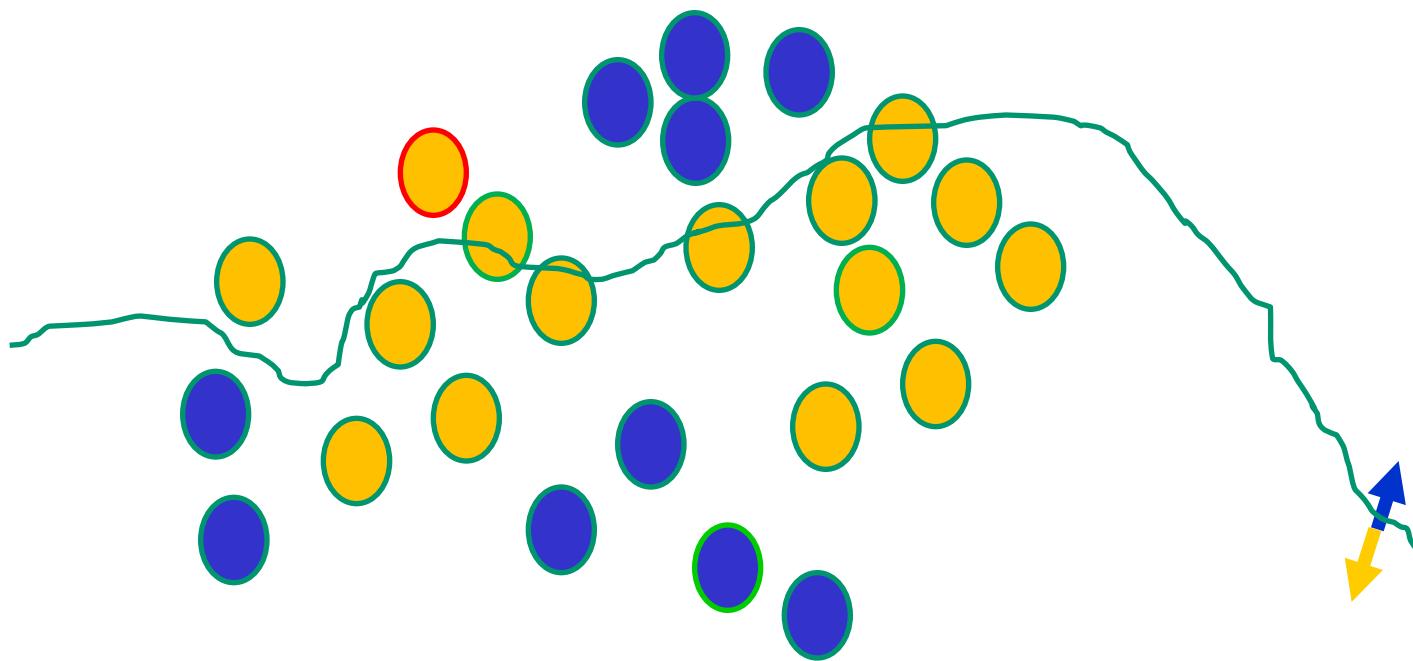
# The decision boundary perspective...

Present a training instance / adjust the weights



# The decision boundary perspective...

Present a training instance / adjust the weights



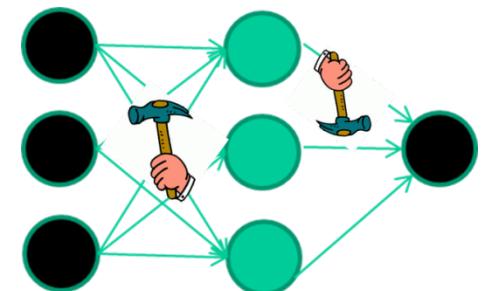
# The decision boundary perspective...

Eventually ....



# Observations:

- 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



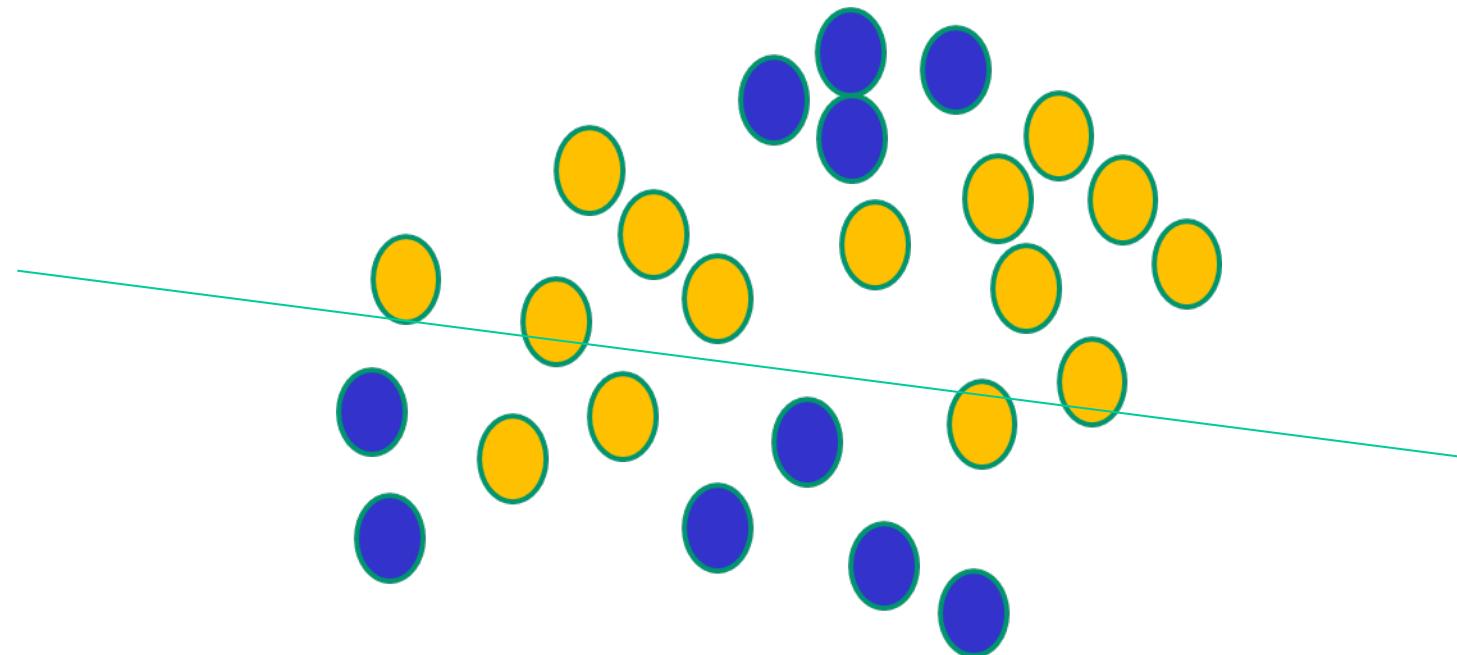
# Some other points:

**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.

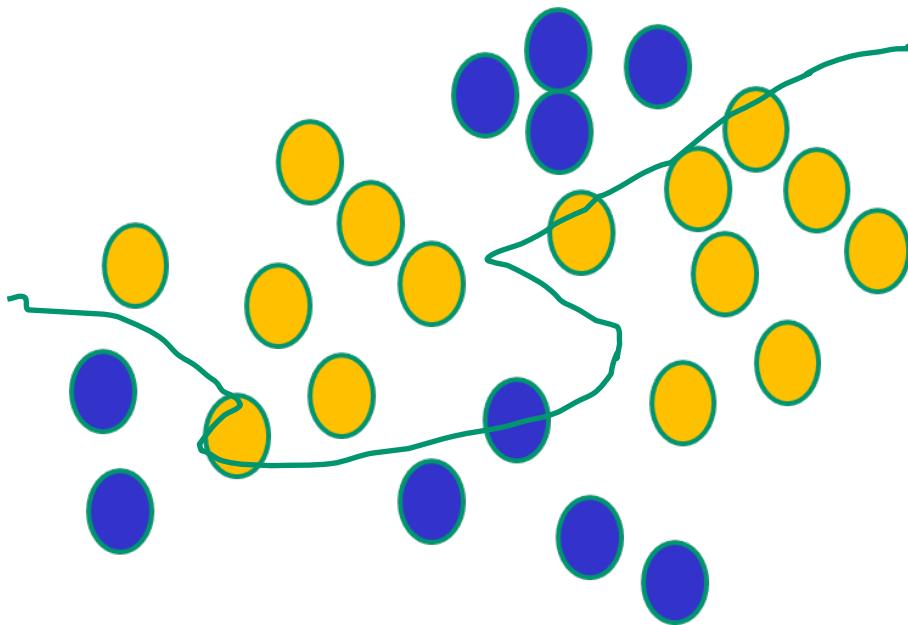
# Some other ‘by the way’ points

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



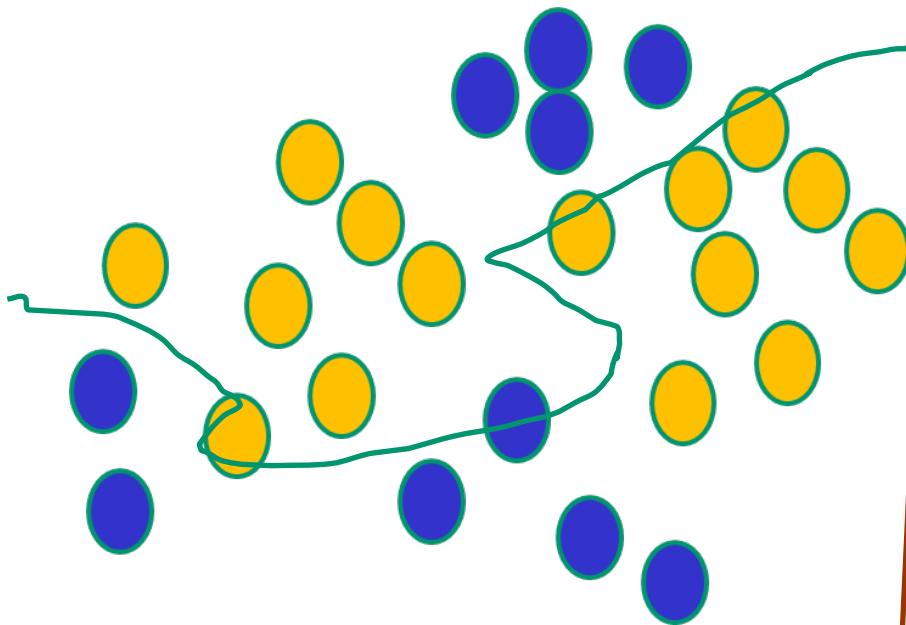
# Some other ‘by the way’ points

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

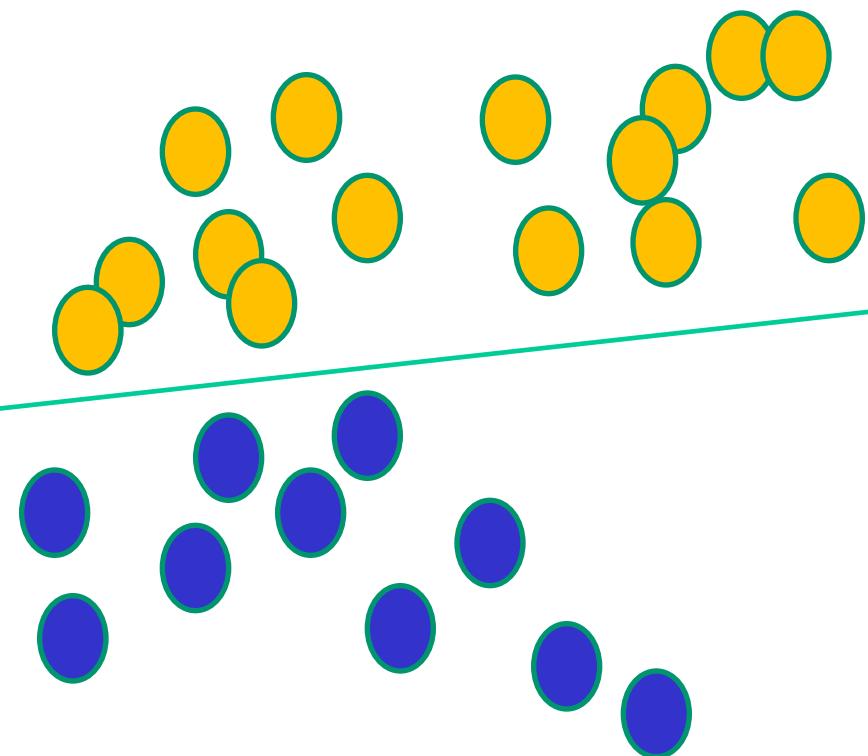


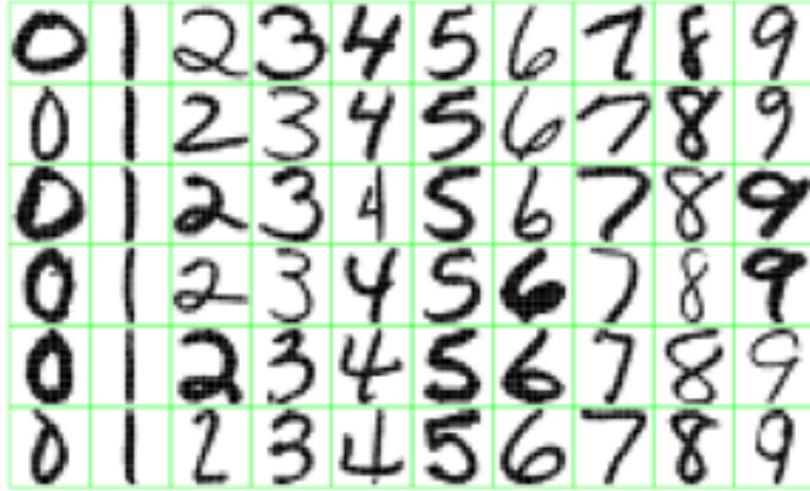
# Some other ‘by the way’ points

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



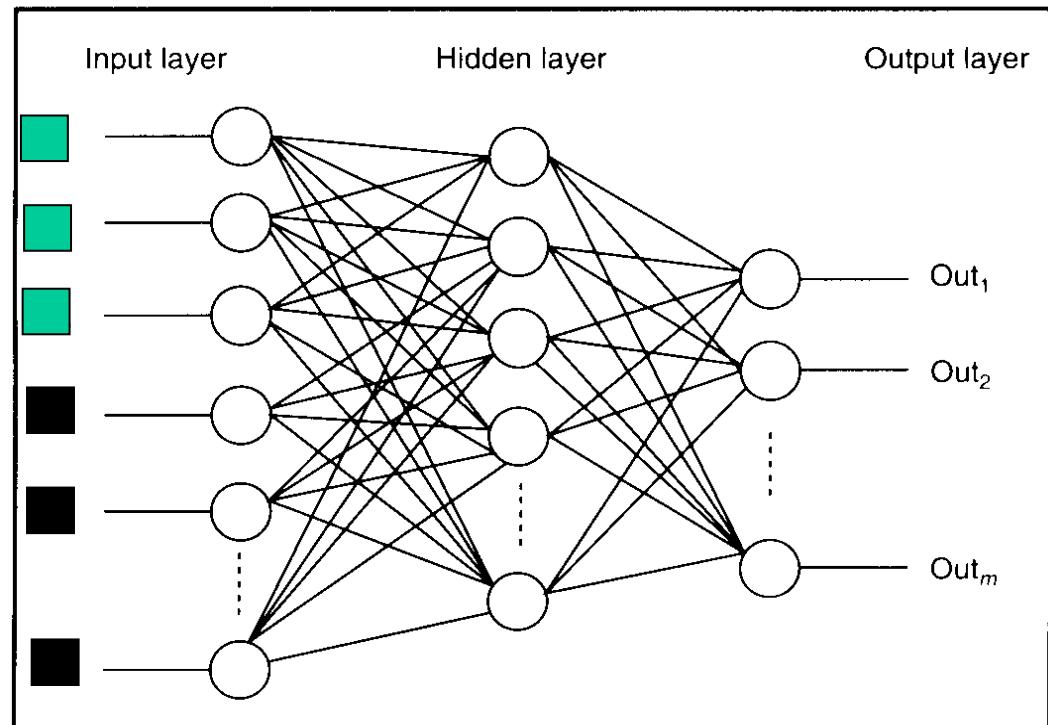
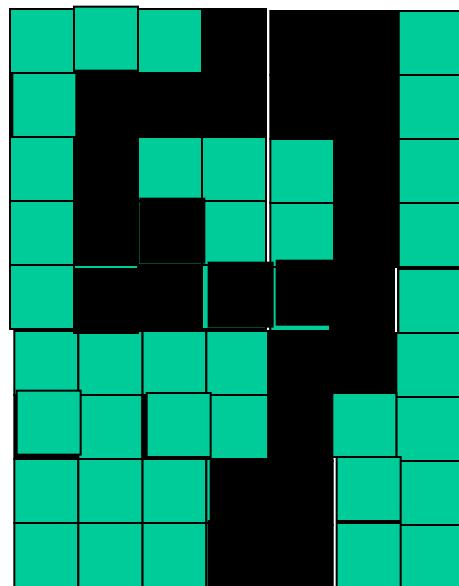
SVMs only draw straight lines, but they transform the data first in a way that makes that OK

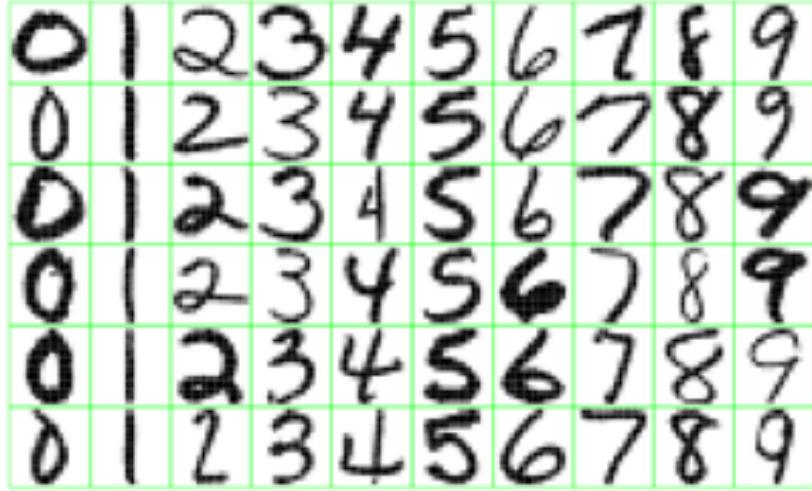




# Feature detectors

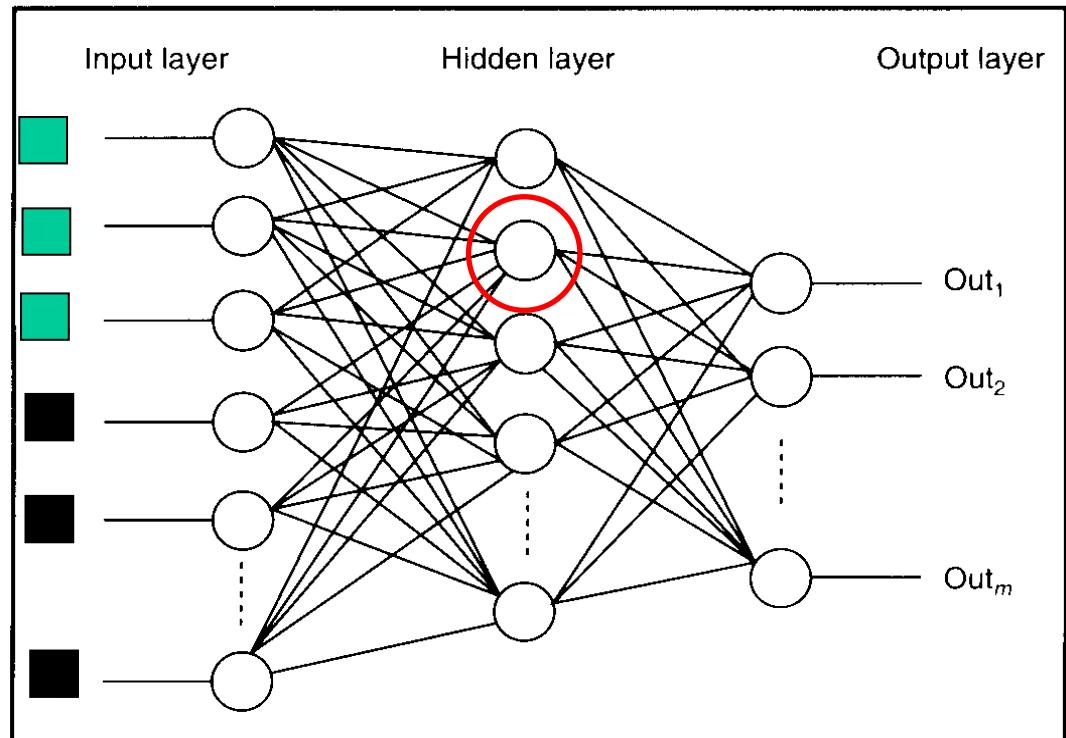
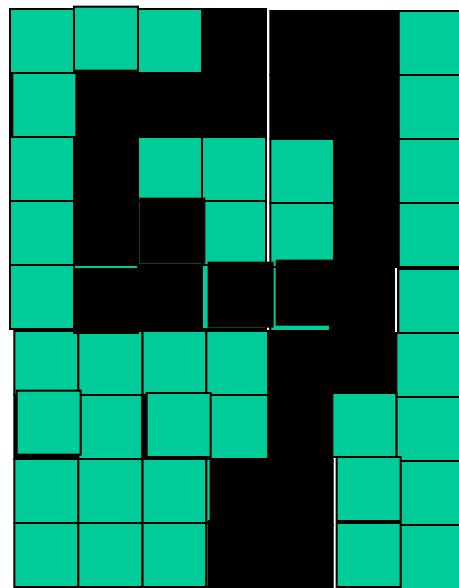
Figure 1.2: Examples of handwritten digits from U.S. postal envelopes.



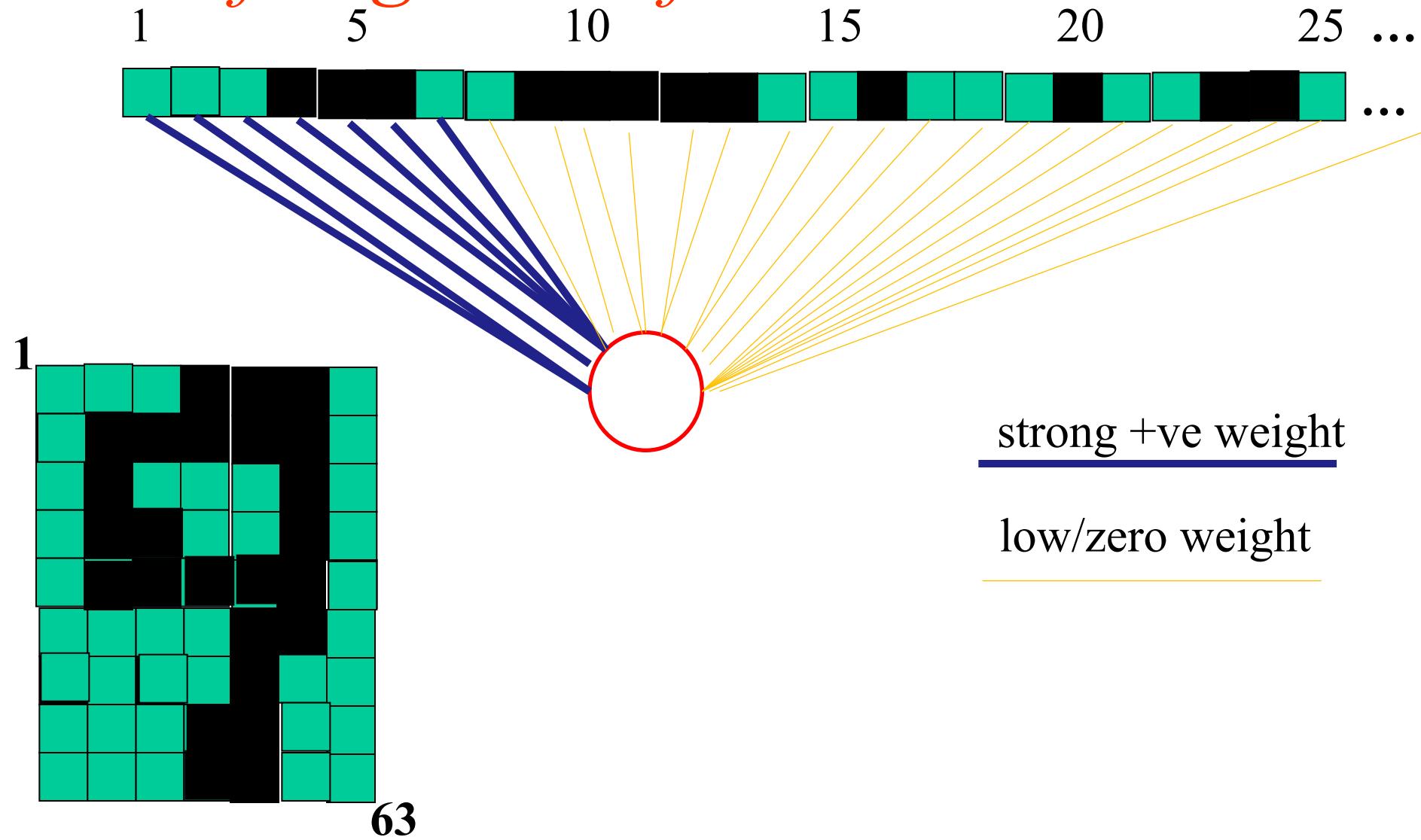


*what is this  
unit doing?*

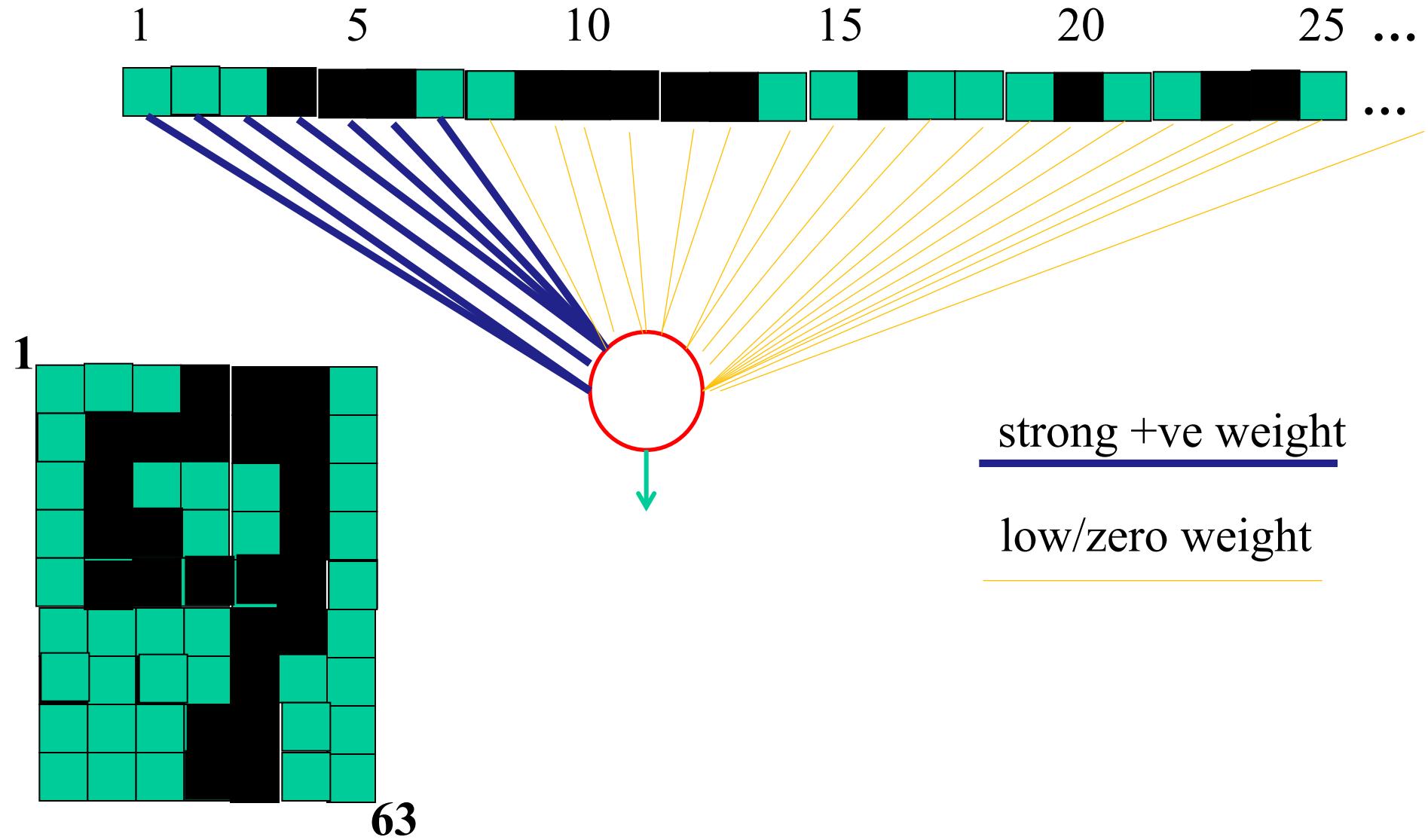
Figure 1.2: Examples of handwritten digits from U.S. postal envelopes.



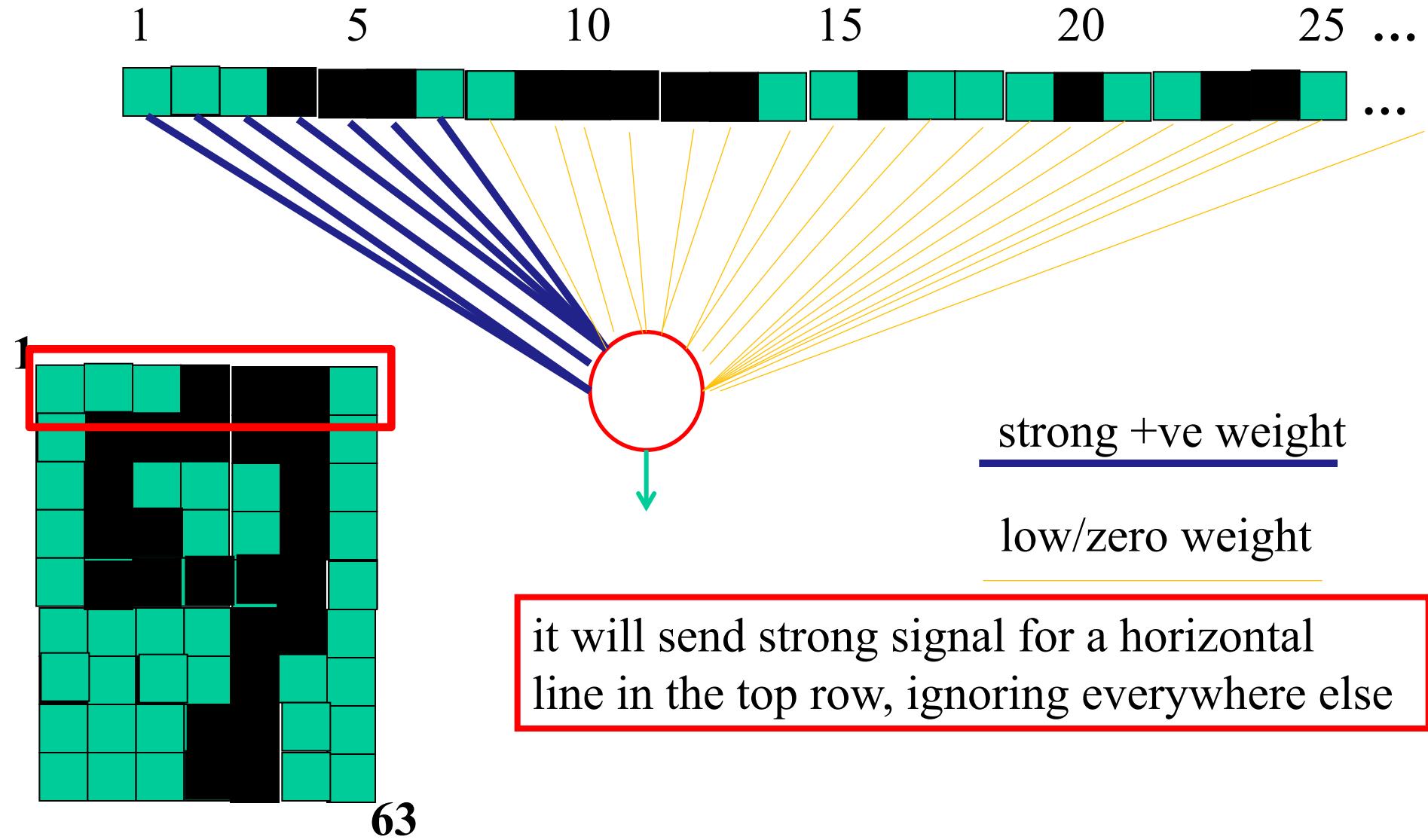
# Hidden layer units become *self-organised feature detectors*



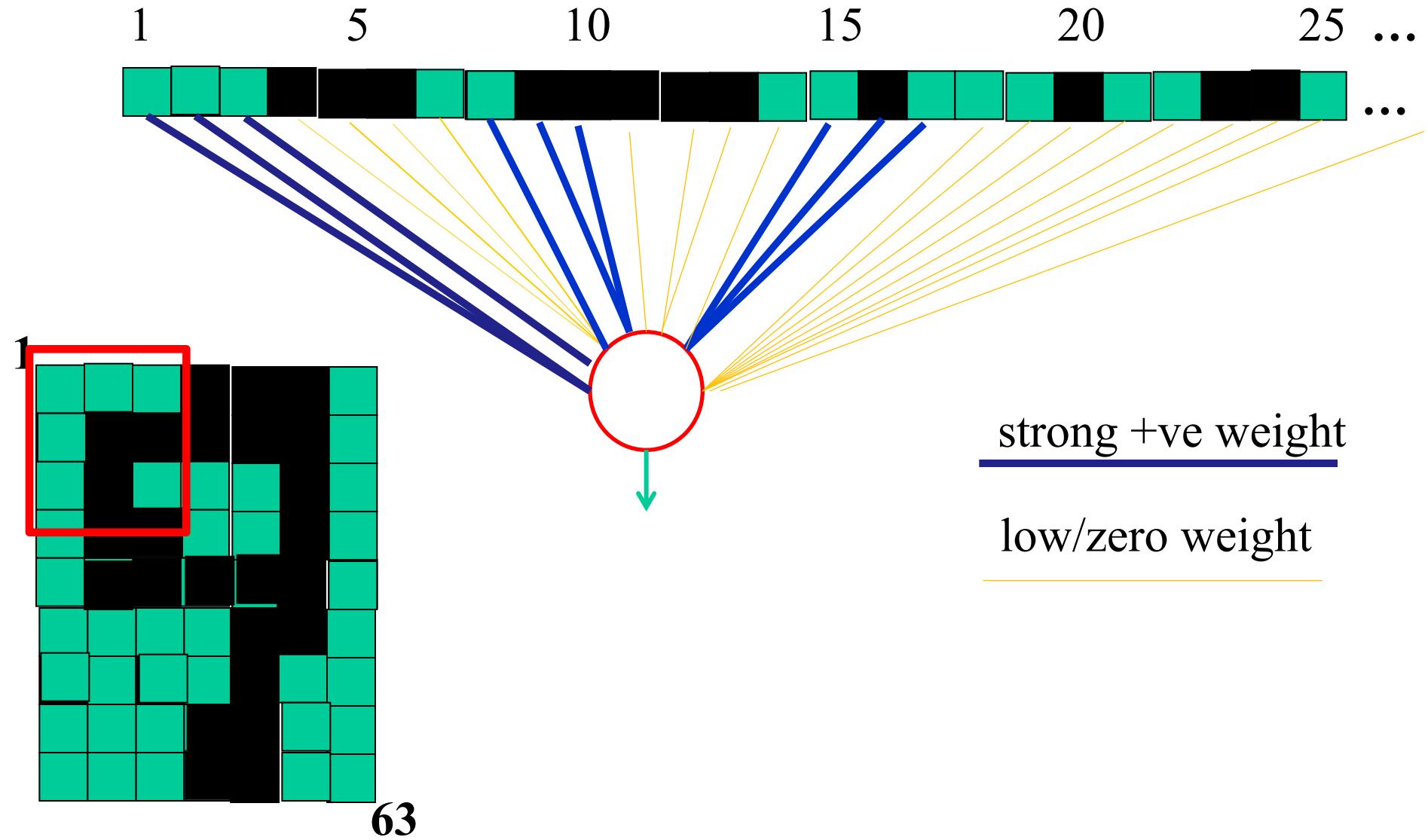
# What does this unit detect?



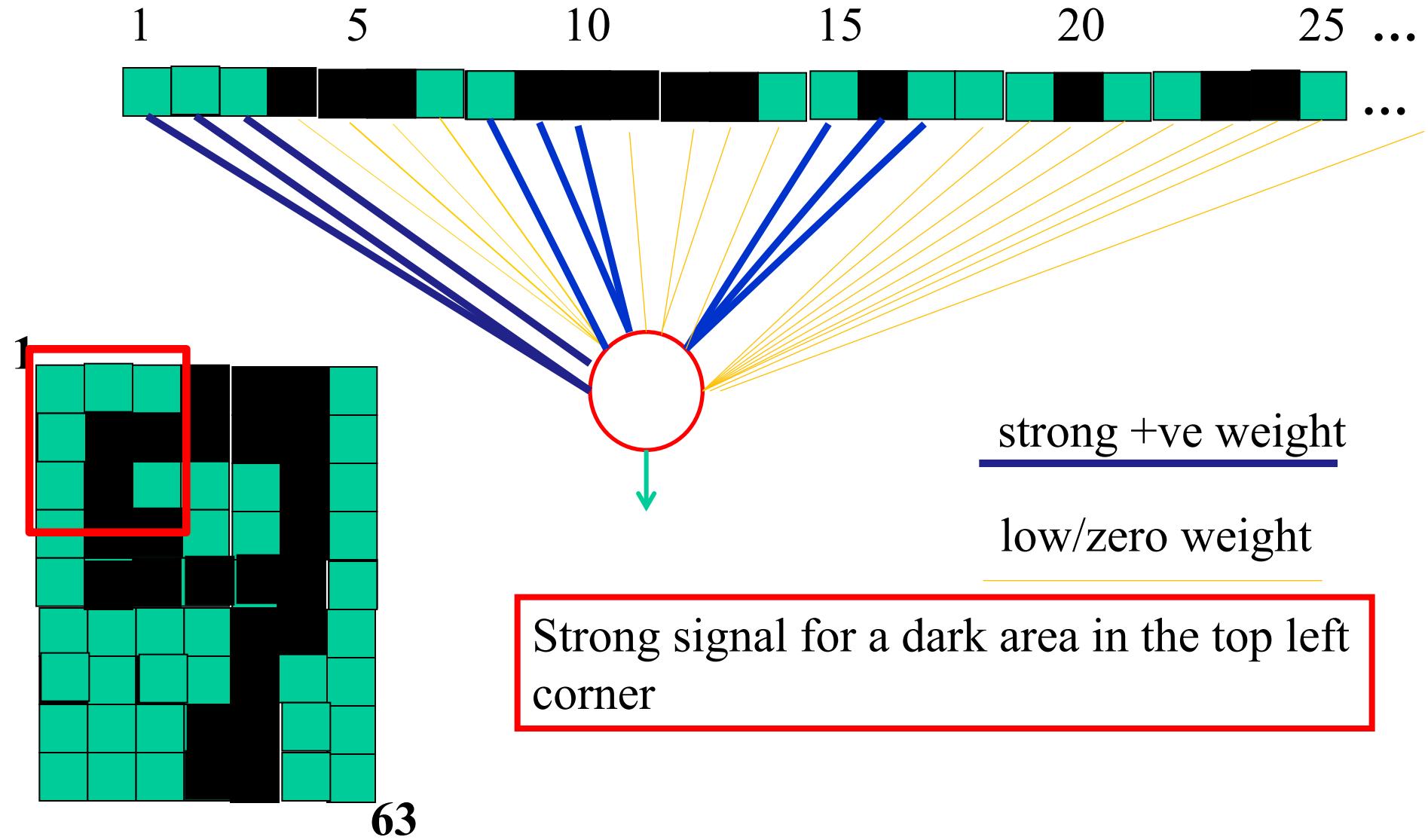
# What does this unit detect?



# What does this unit detect?



# What does this unit detect?



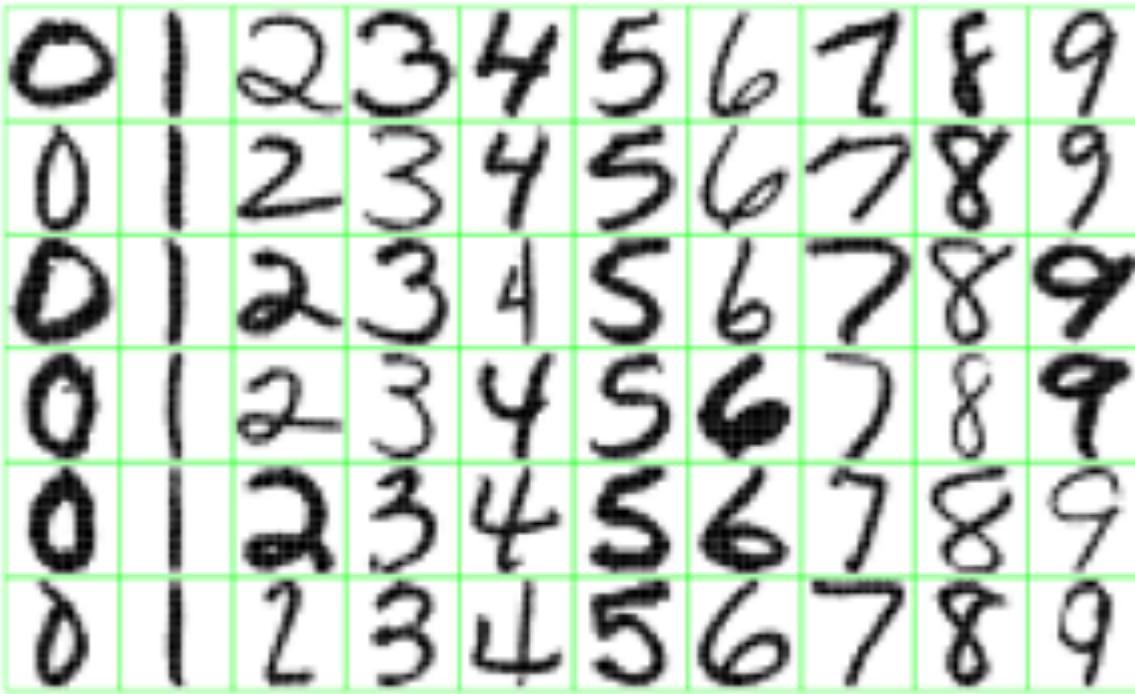


Figure 1.2: Examples of handwritten digits from U.S. postal envelopes.

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

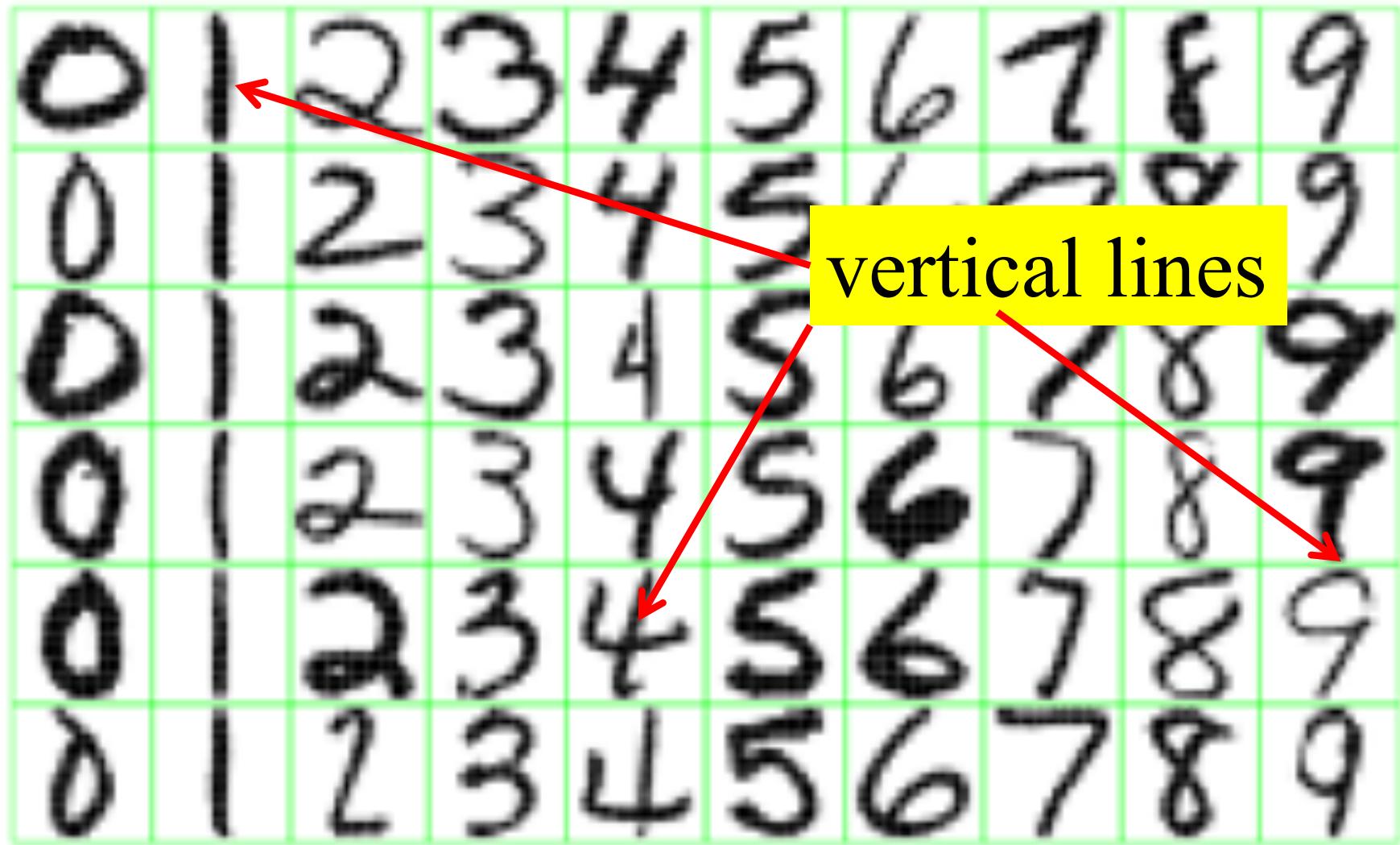


Figure 1.2: Examples of handwritten digits from U.S. postal envelopes.

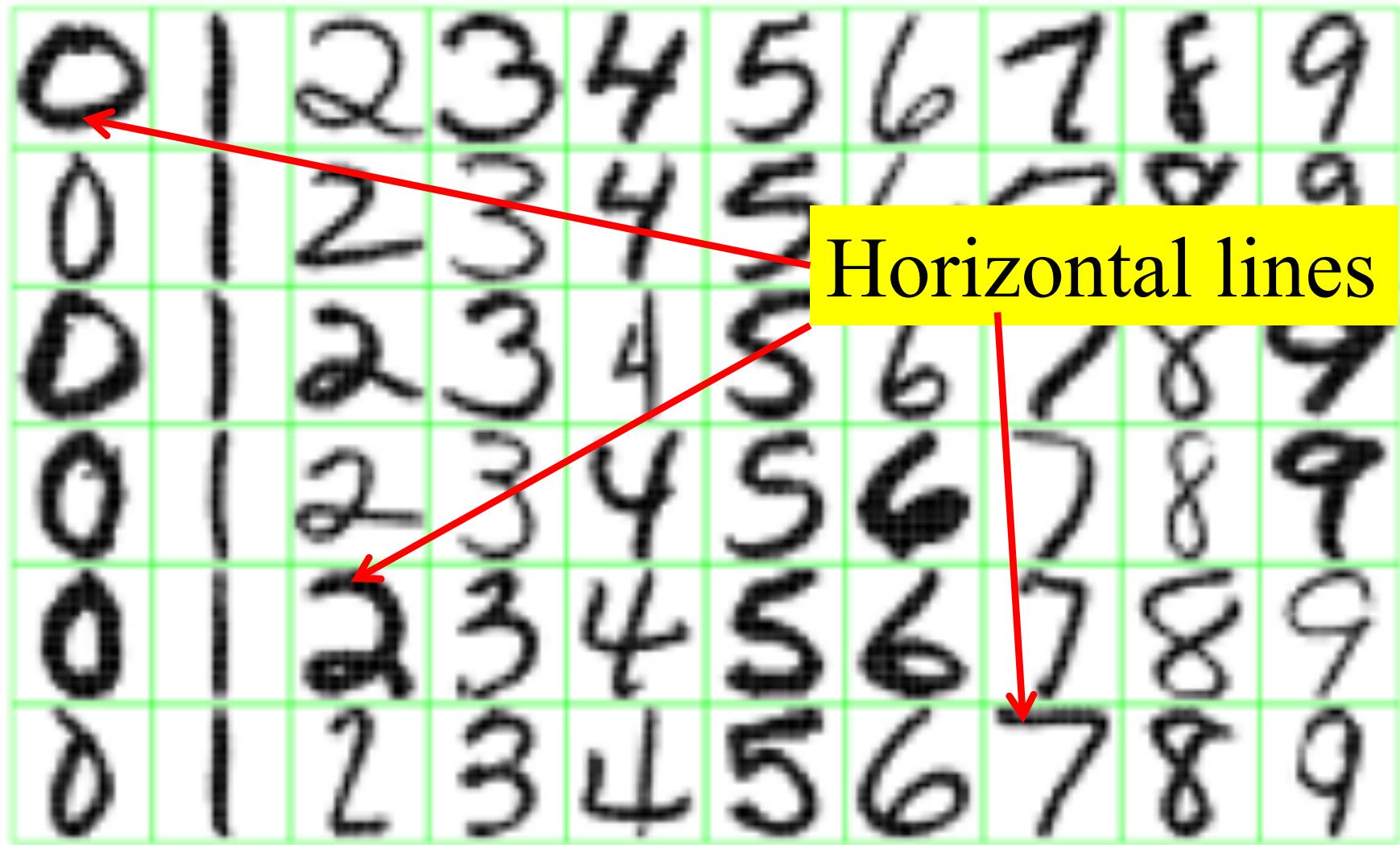


Figure 1.2: Examples of handwritten digits from U.S. postal envelopes.

1

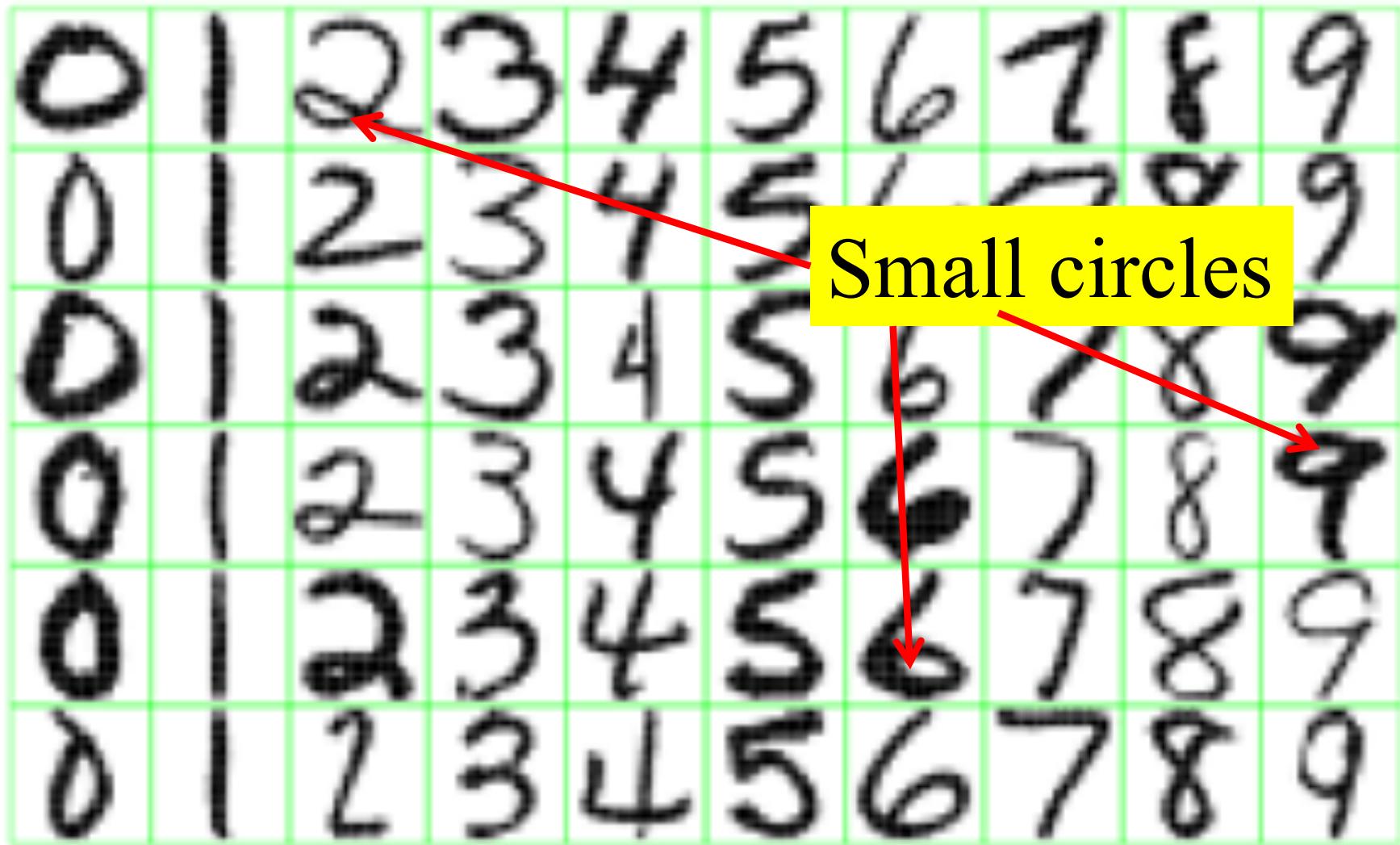
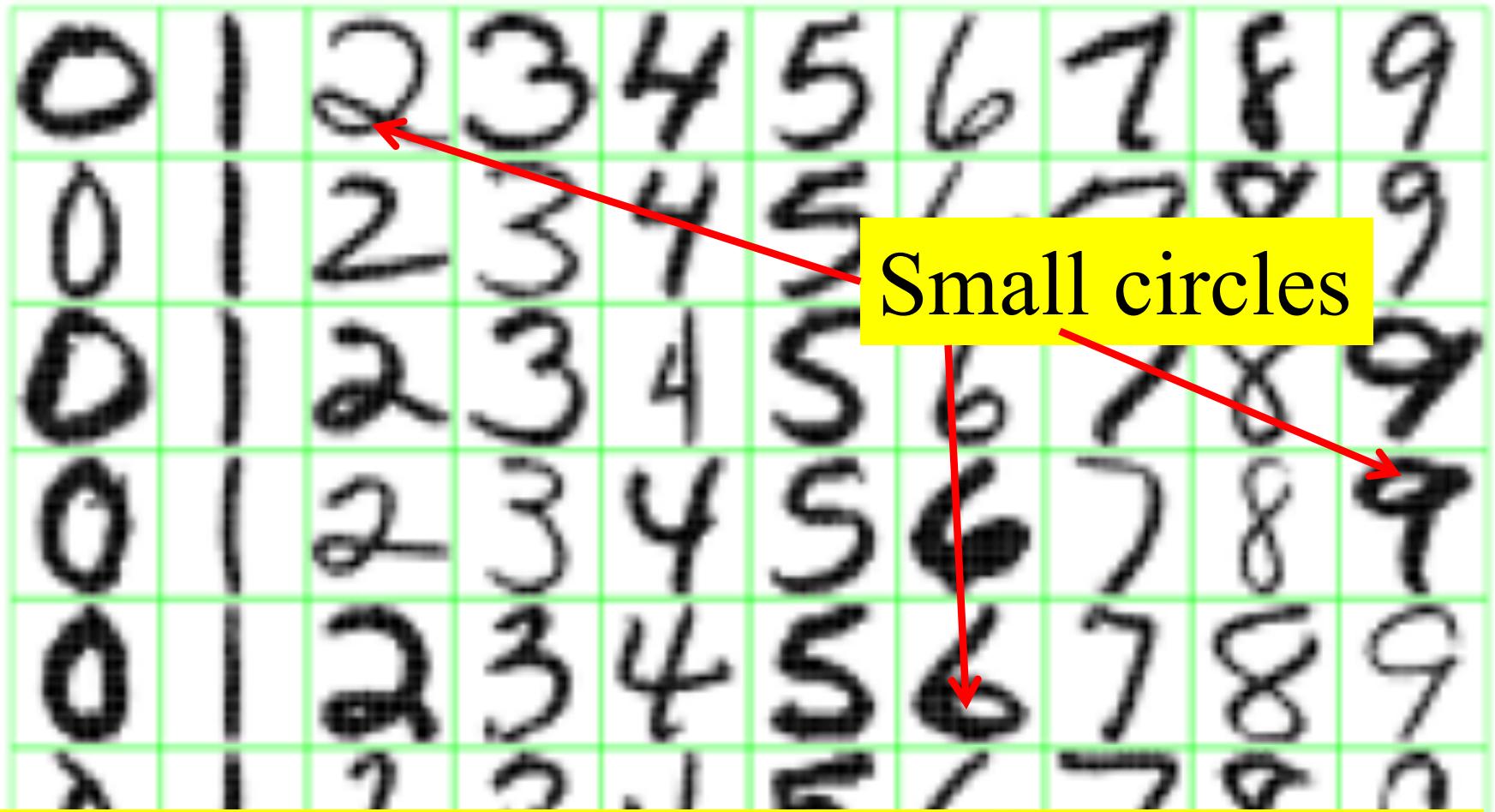
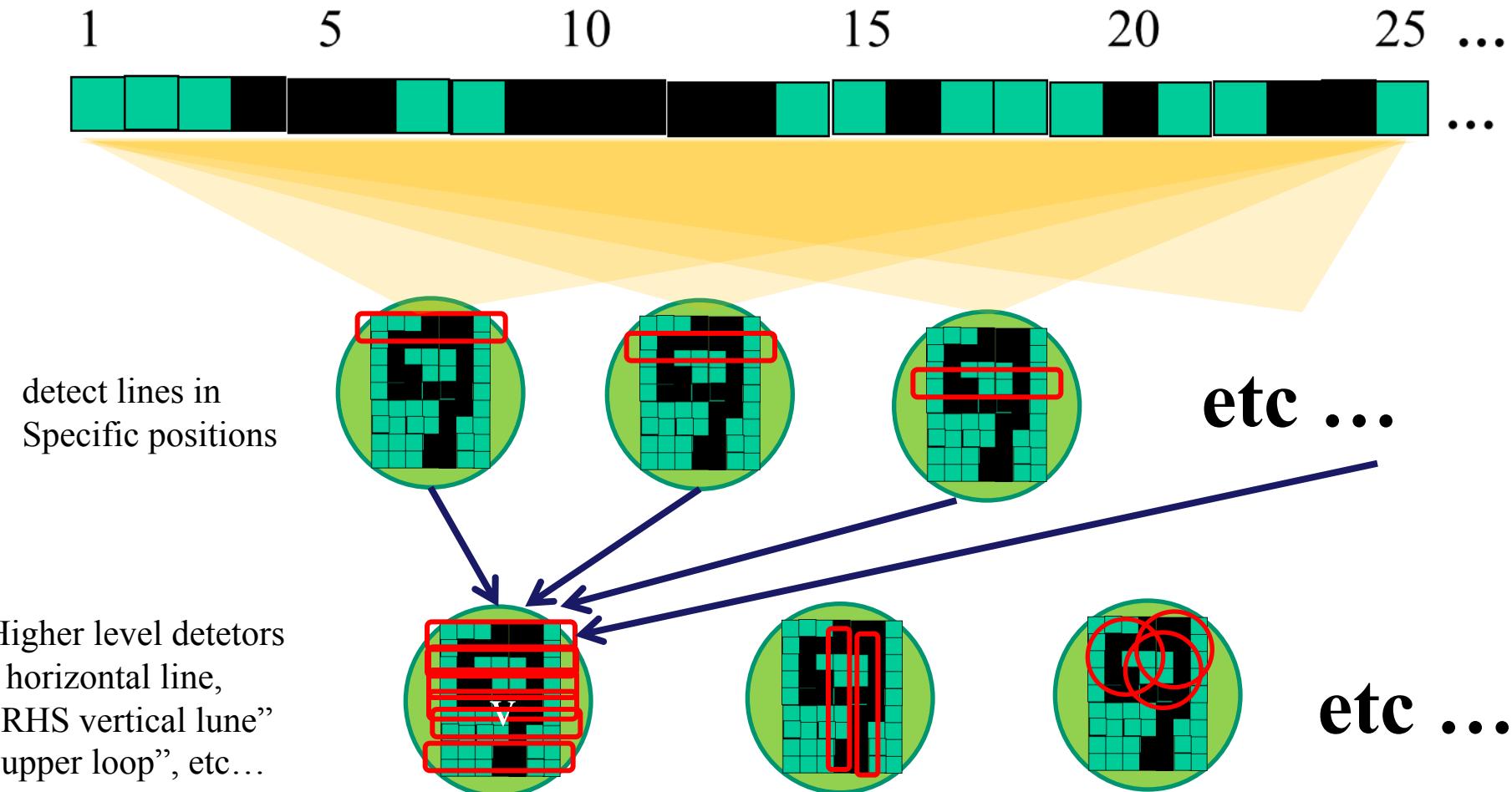


Figure 1.2: Examples of handwritten digits from U.S. postal envelopes.

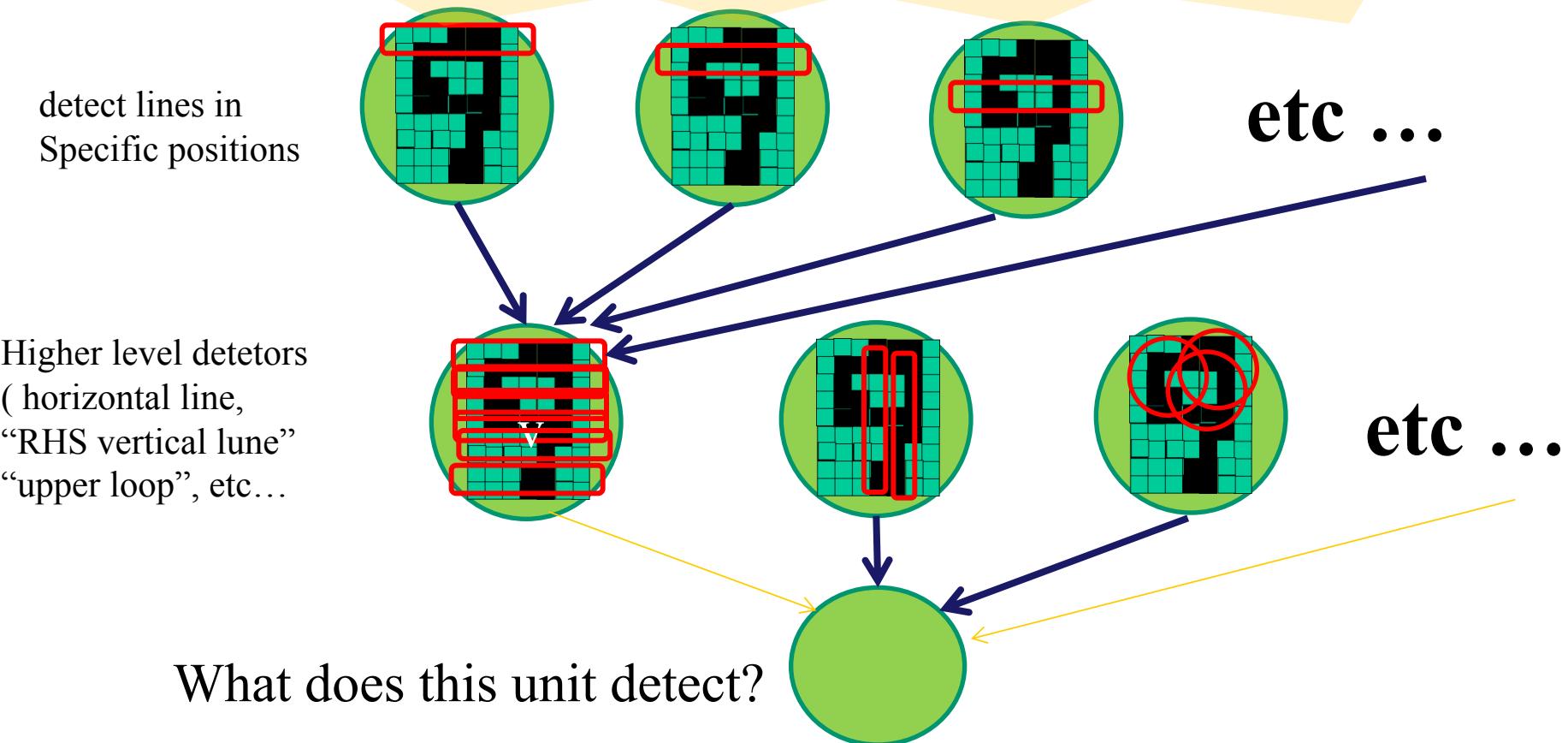


But what about position invariance ???  
our example unit detectors were tied to  
specific parts of the image

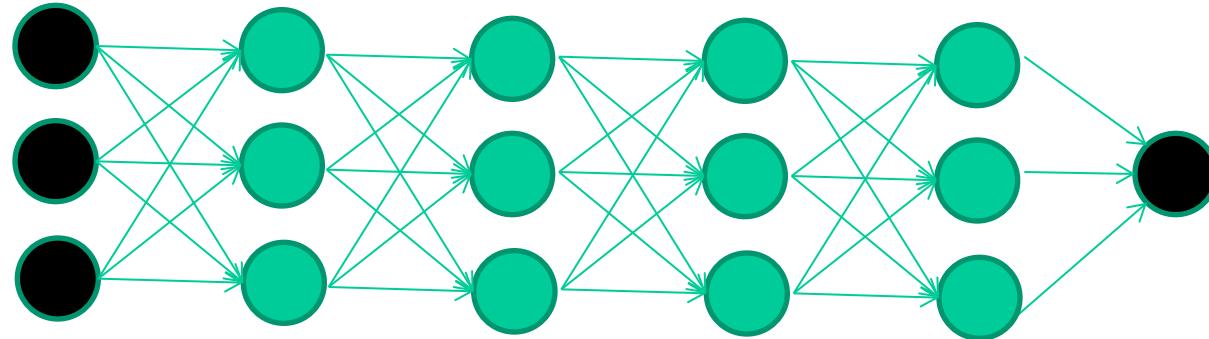
successive layers can learn higher-level features ...



successive layers can learn higher-level features ...

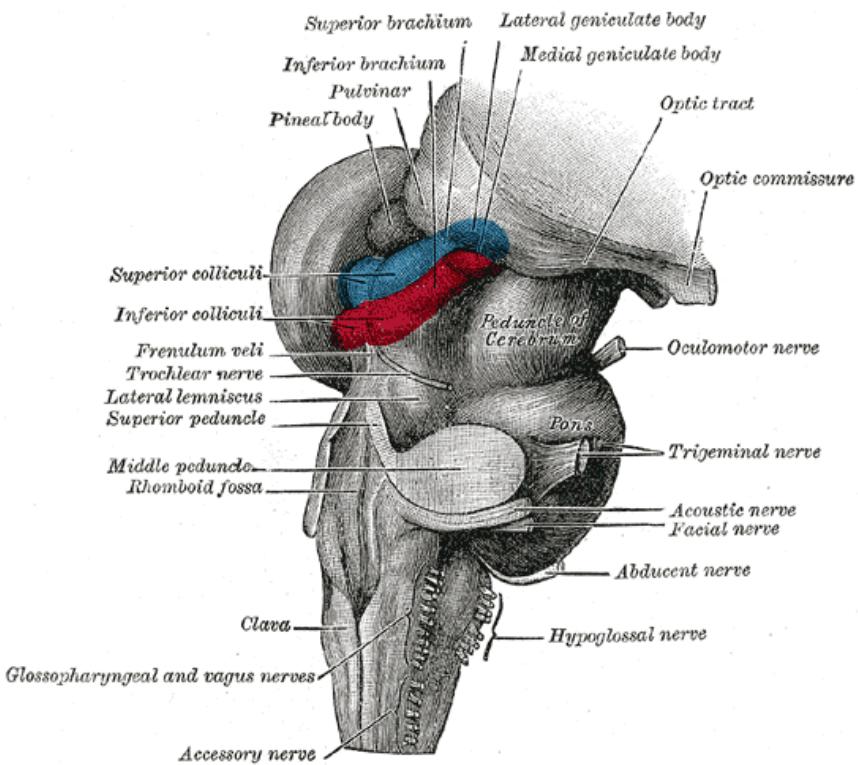
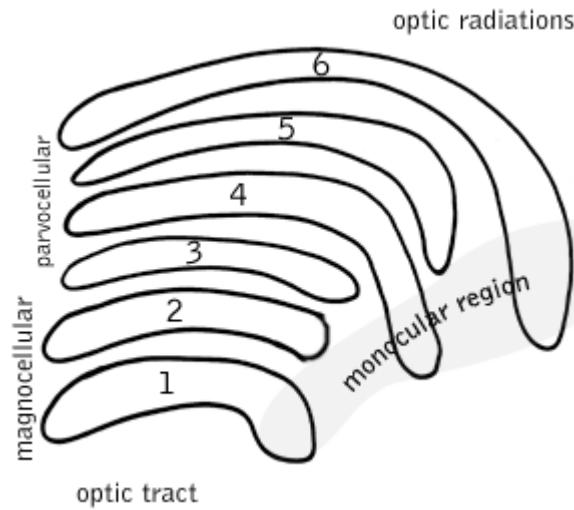


*So: multiple layers make sense*



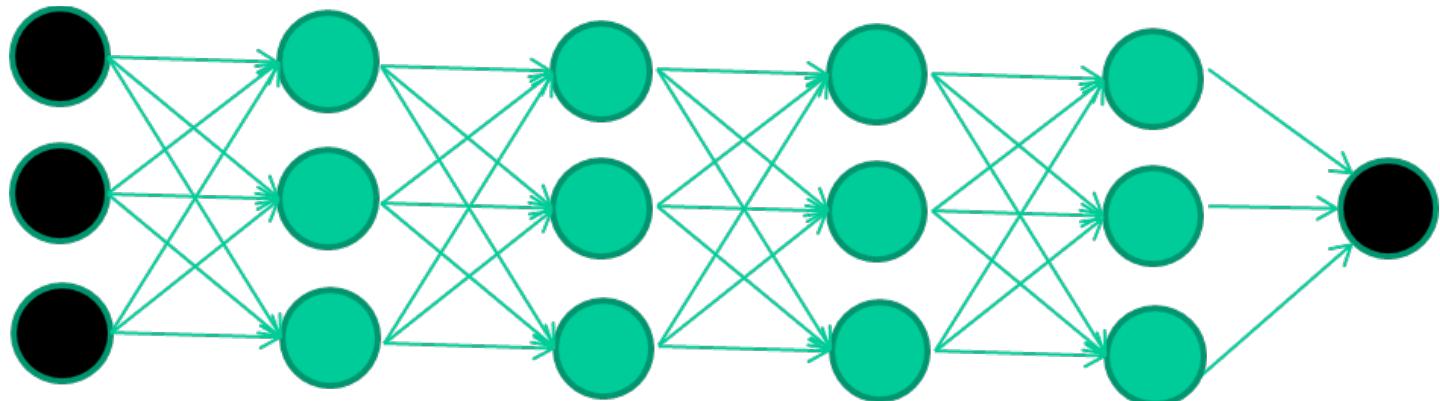
# So: multiple layers make sense

Your brain works that way

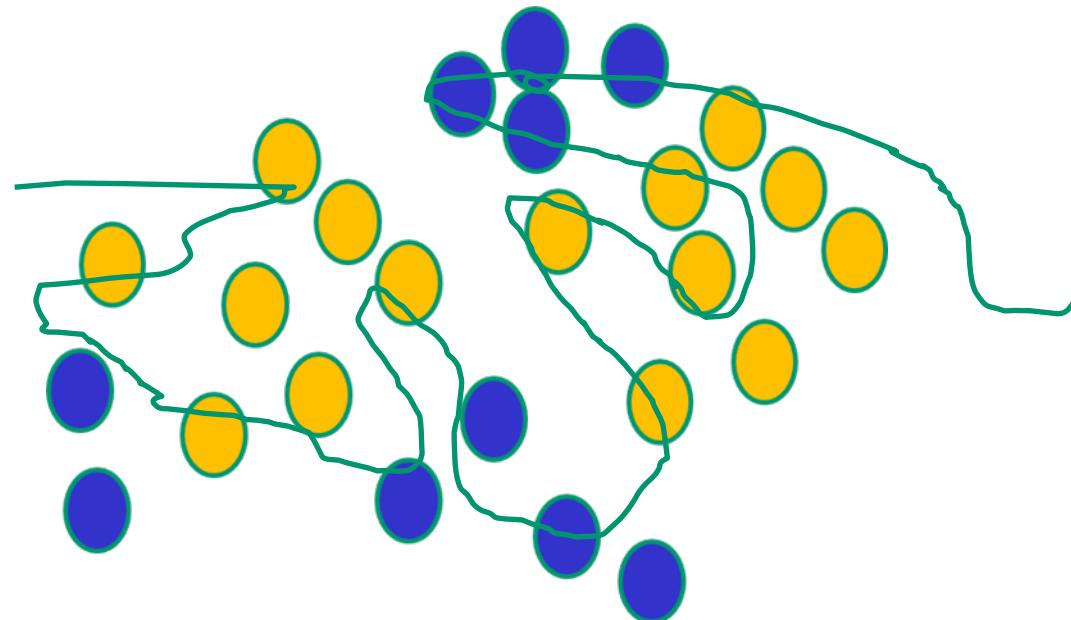
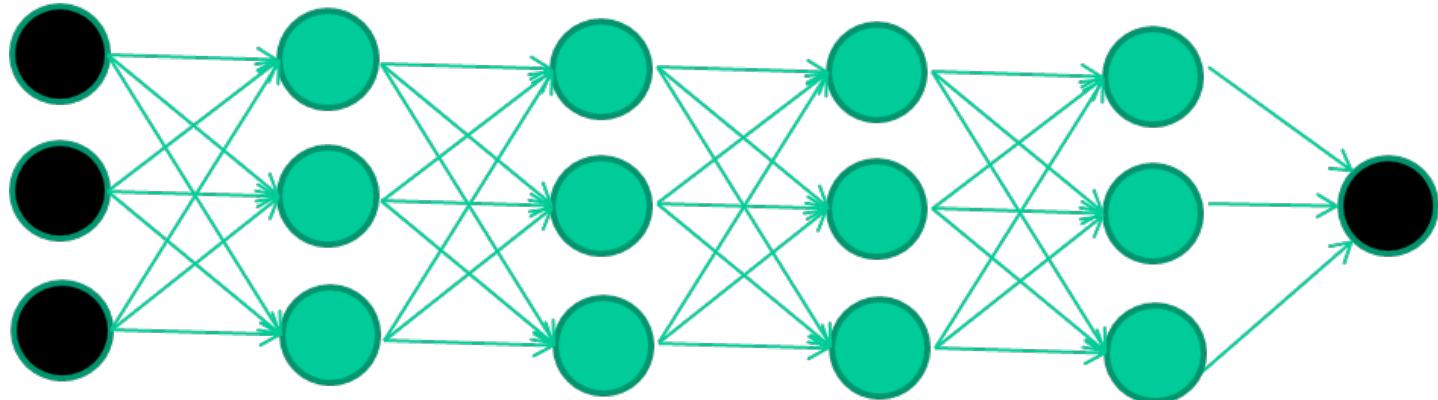


# *So: multiple layers make sense*

**Many-layer neural network architectures should be capable of learning the true underlying features and ‘feature logic’, and therefore generalise very well ...**

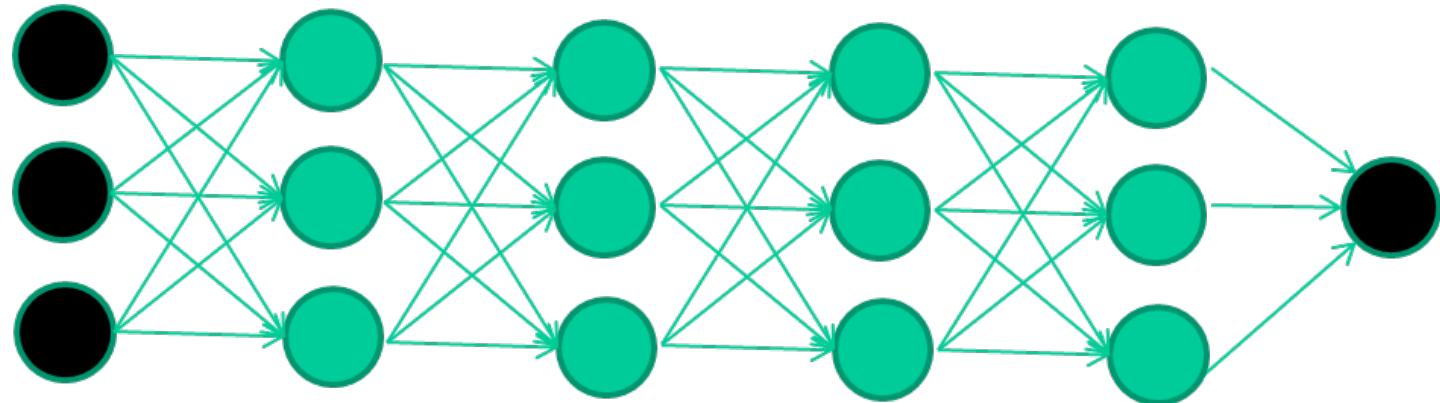


But, until very recently, our weight-learning algorithms simply did not work on multi-layer architectures

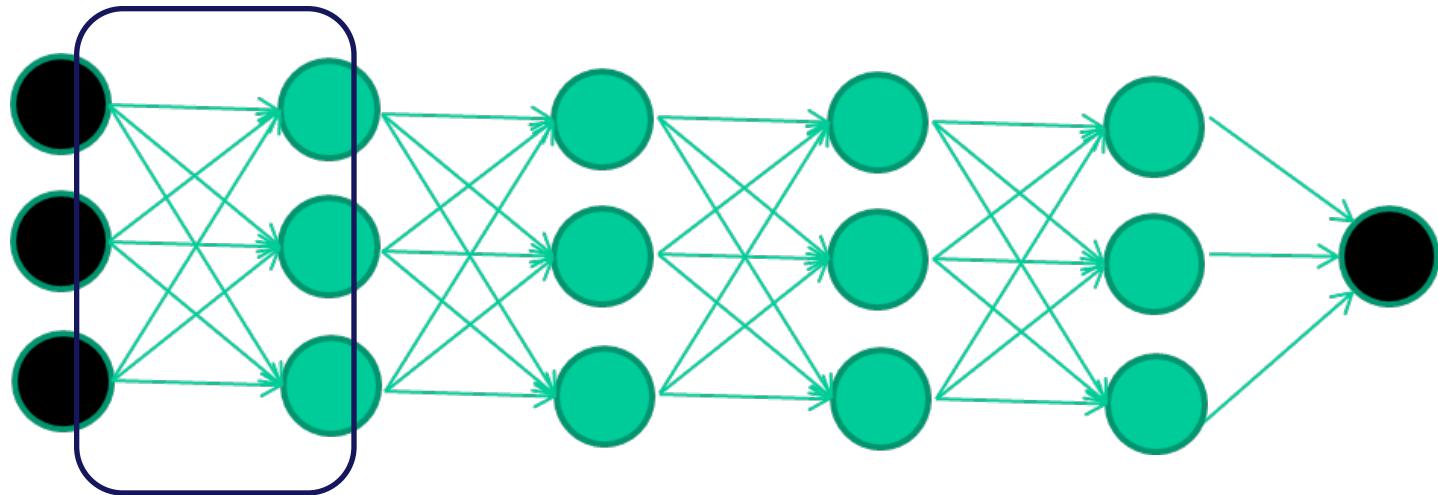


Along came deep learning ...

# The new way to train multi-layer NNs...

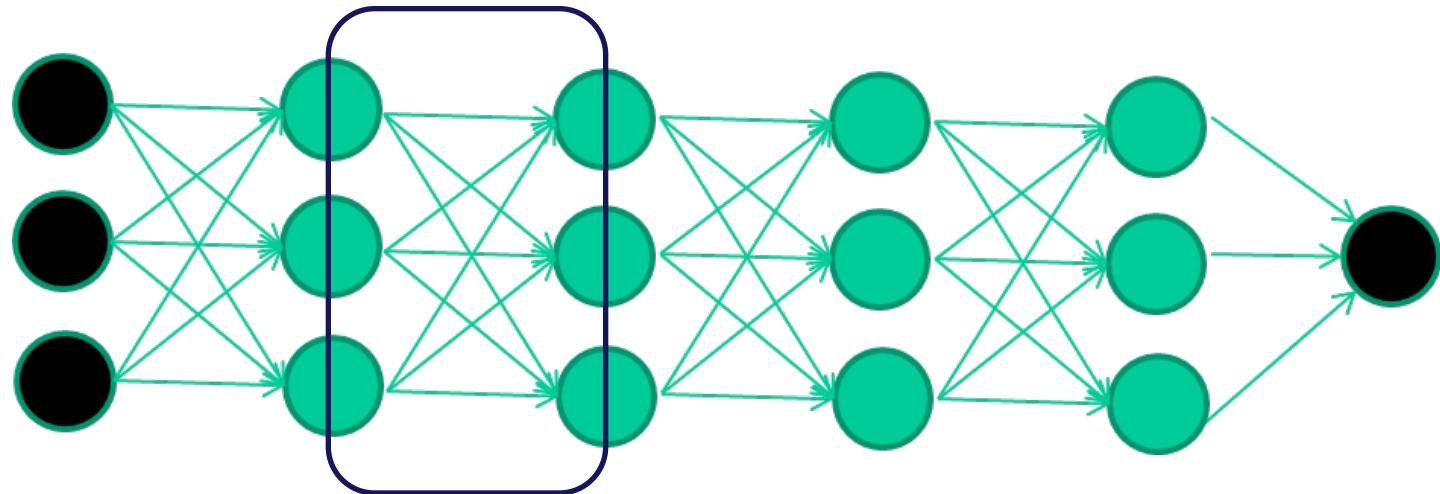


# The new way to train multi-layer NNs...



Train **this** layer first

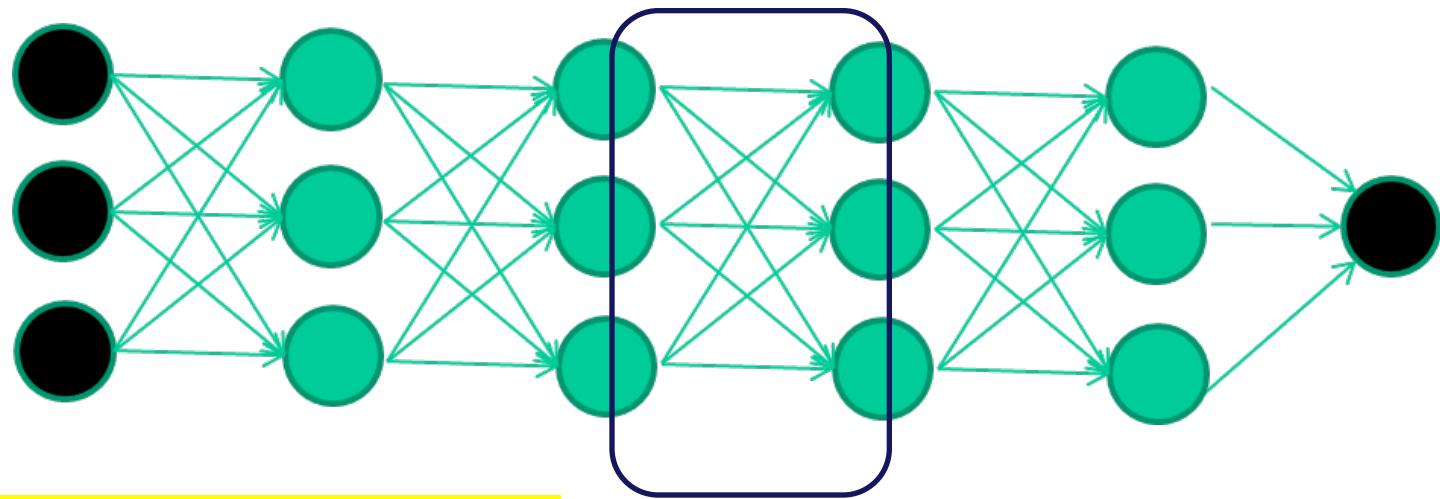
# The new way to train multi-layer NNs...



Train **this** layer first

then **this** layer

# The new way to train multi-layer NNs...

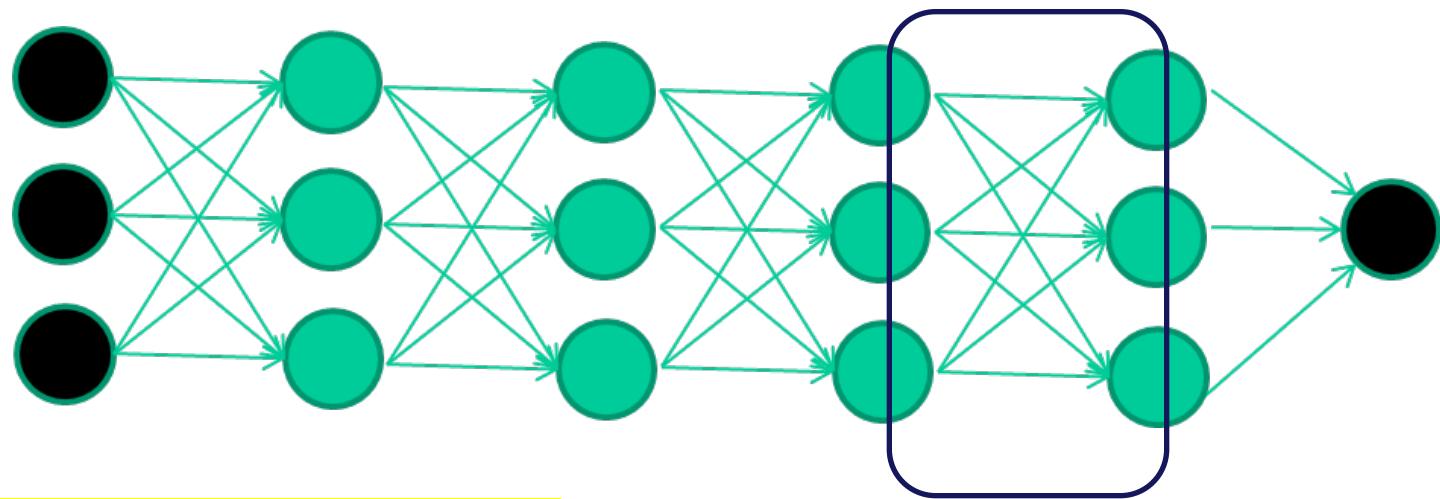


Train **this** layer first

then **this** layer

then **this** layer

# The new way to train multi-layer NNs...



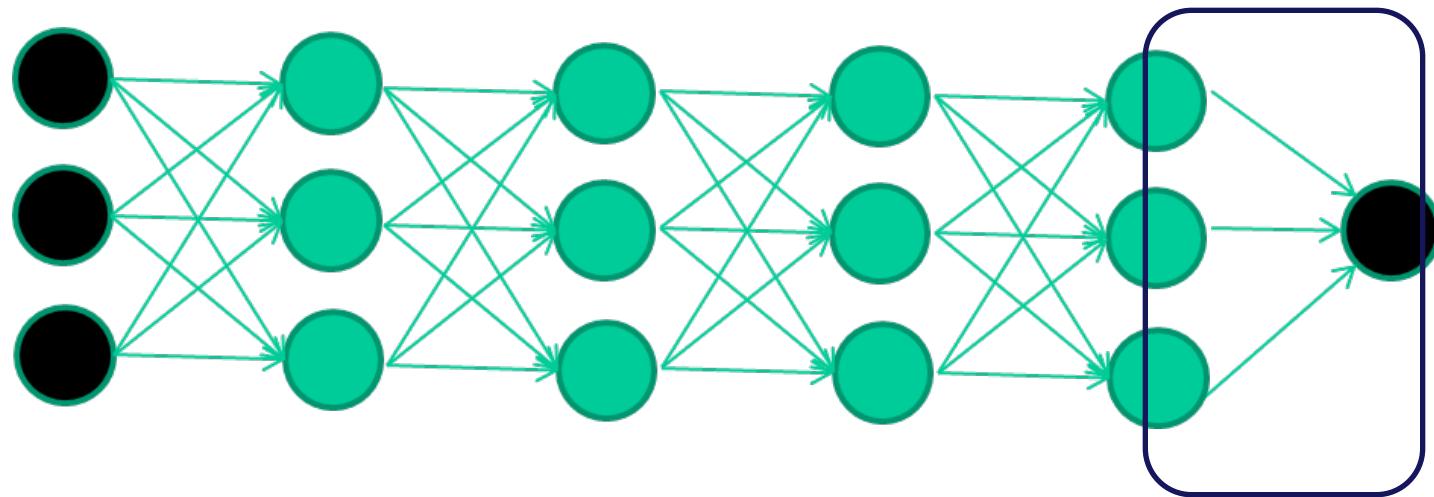
Train **this** layer first

then **this** layer

then **this** layer

then **this** layer

# The new way to train multi-layer NNs...



Train **this** layer first

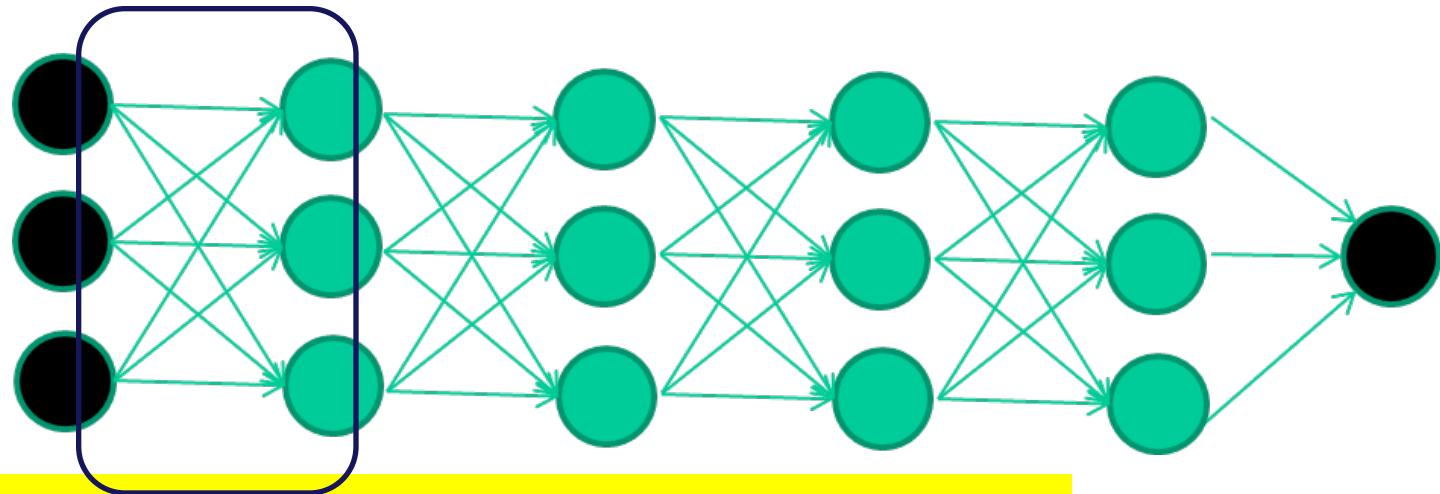
then **this** layer

then **this** layer

then **this** layer

finally **this** layer

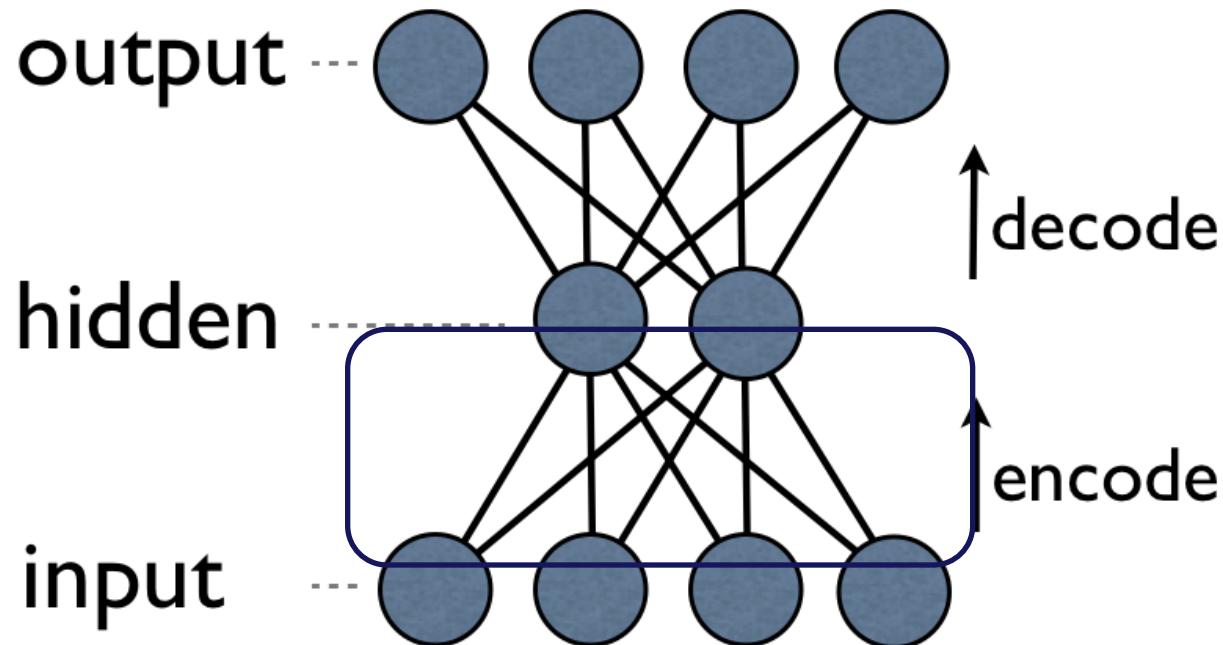
# The new way to train multi-layer NNs...



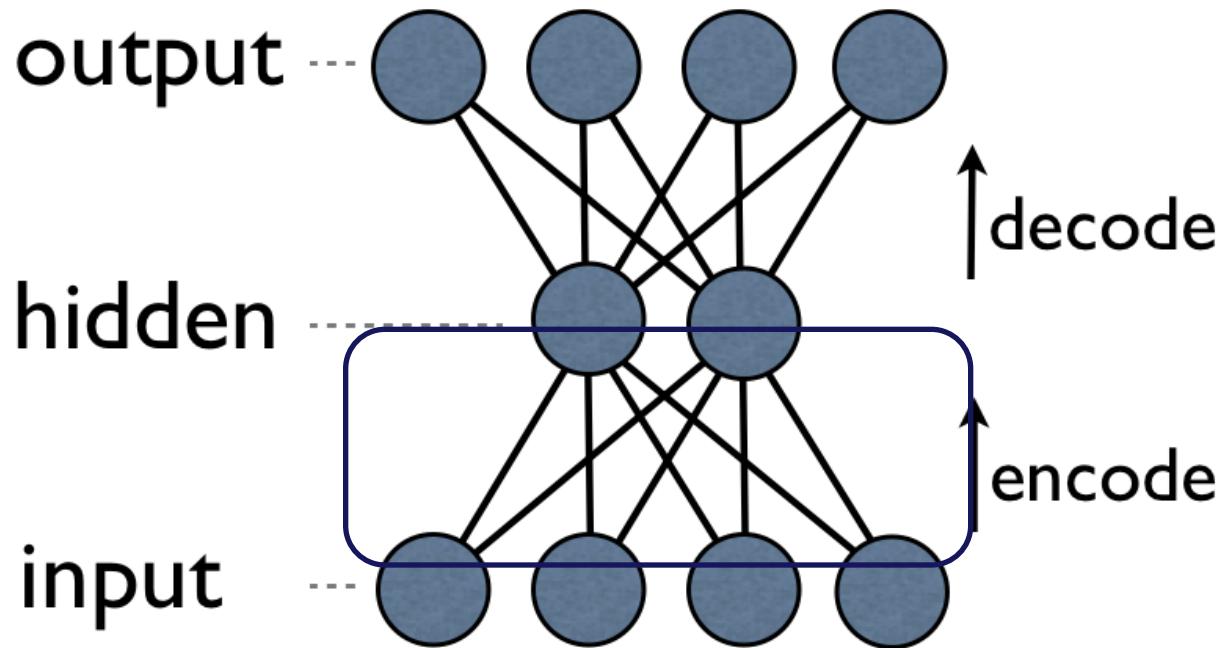
*EACH of the (non-output) layers is  
trained to be an **auto-encoder***

*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 reproduce the input

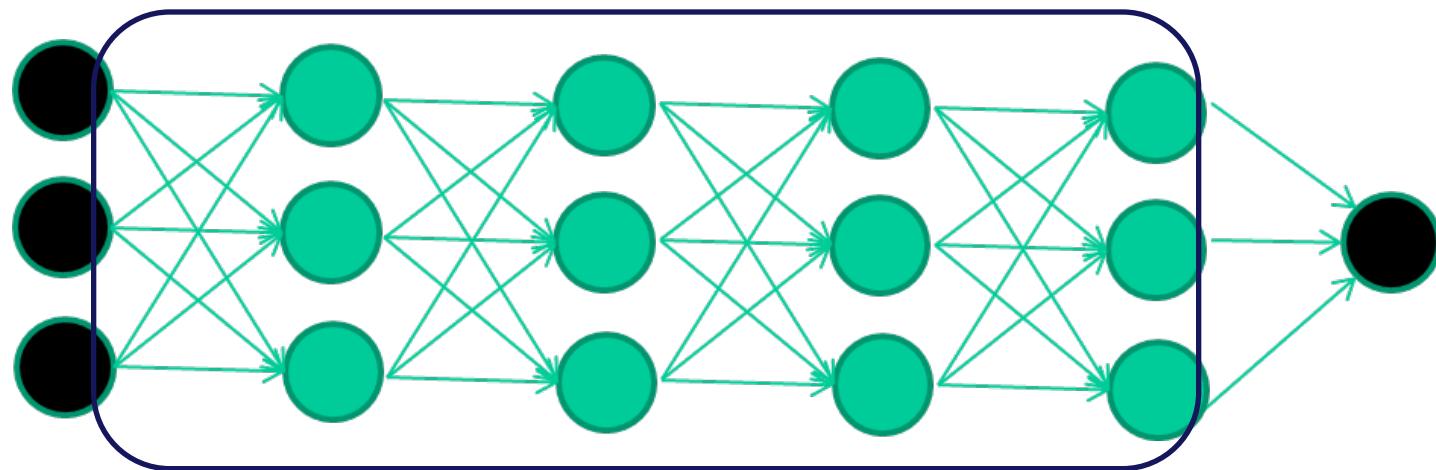


an auto-encoder is trained, with an absolutely standard weight-adjustment algorithm to reproduce the input

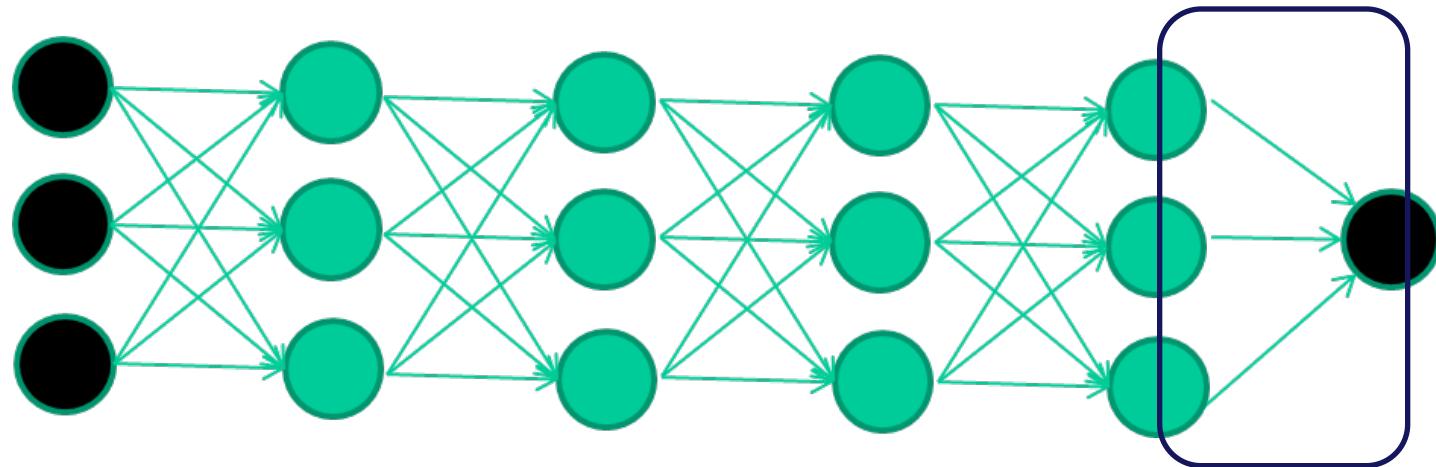


By making this happen with (many) fewer units than the inputs, this forces the ‘hidden layer’ units to become good feature detectors

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

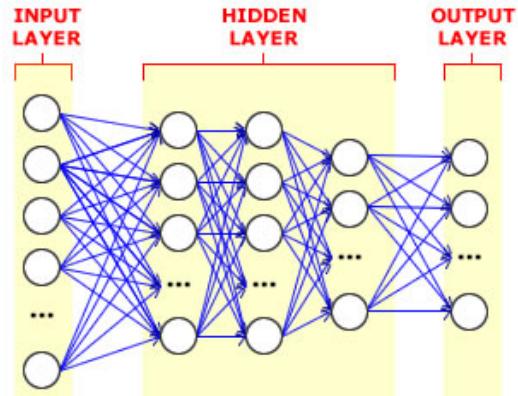


Final layer trained to predict class based  
on outputs from previous layers



# And that's that

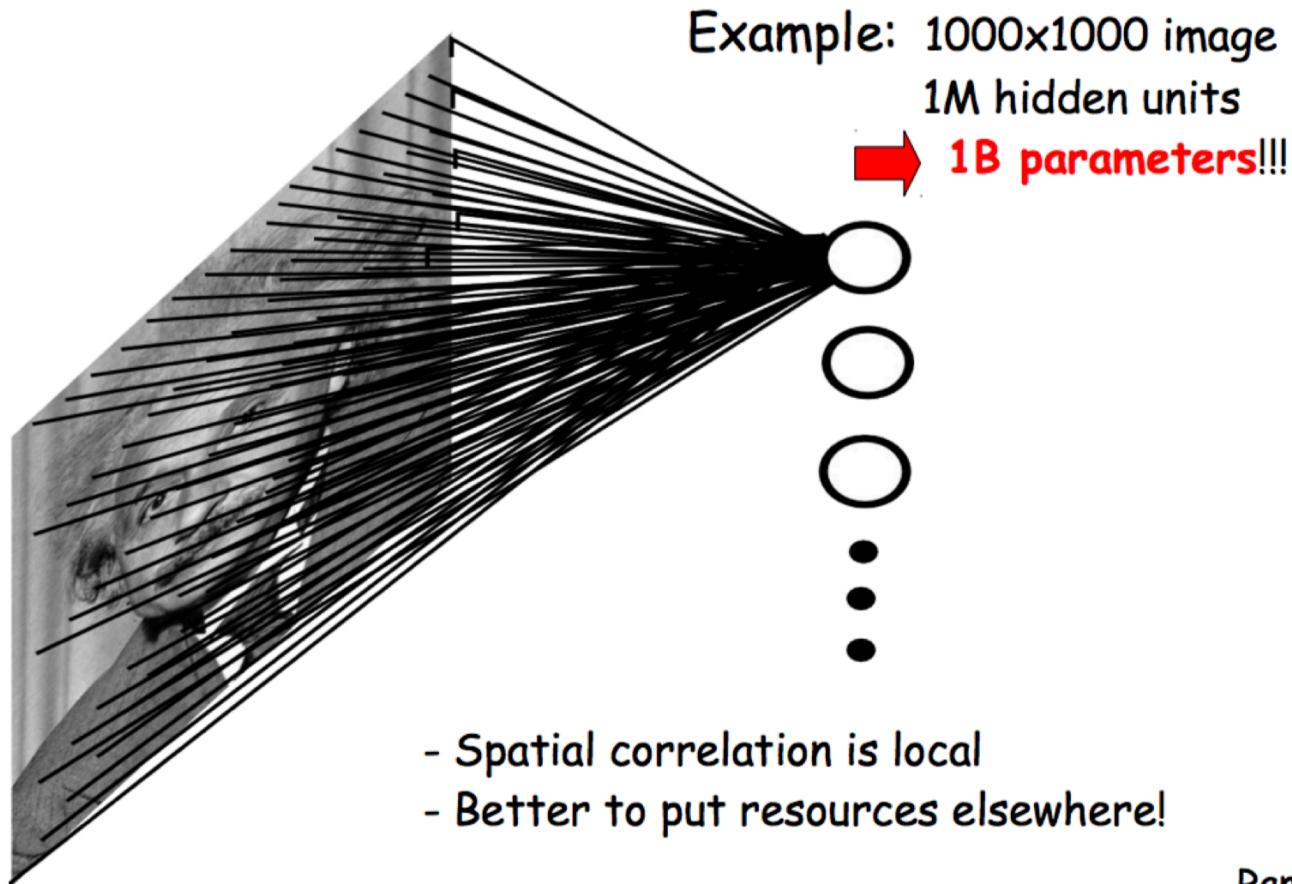
- That's the basic idea
- There are many many types of deep learning,
- different kinds of autoencoder, variations on architectures and training algorithms, etc...
- Very fast growing area ...



# Convolutional Neural Network

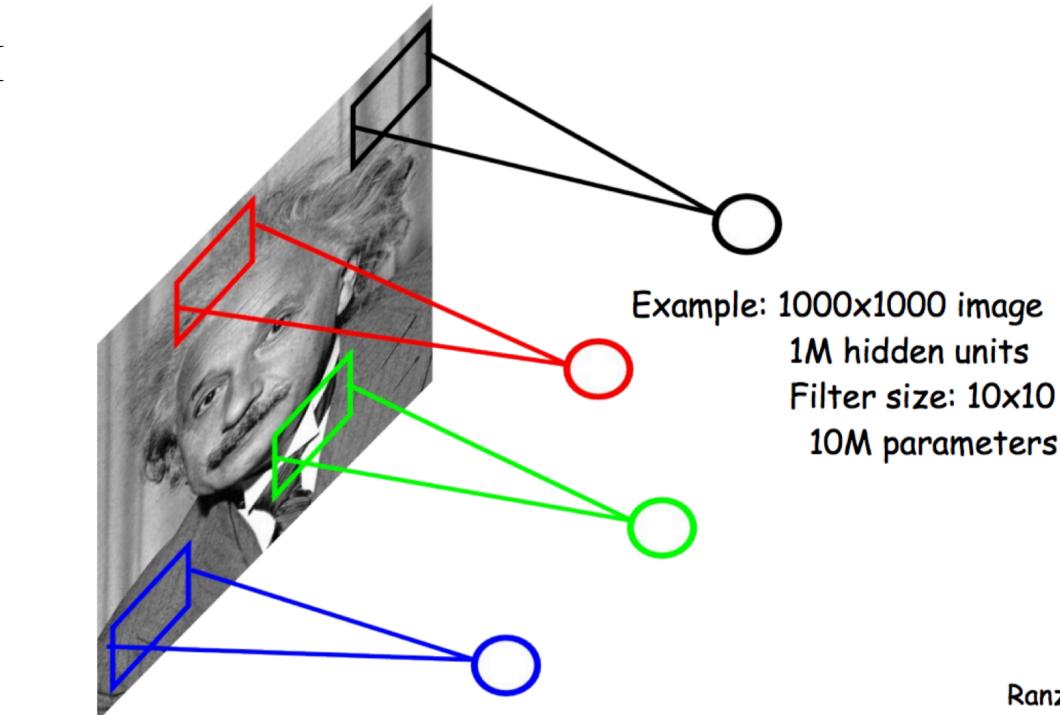
- Popular deep learning network for image classification
- One of the earliest DL Algorithms (1998)
- Basic idea:
  - multiple layers of auto-encoders in the form of convolutional layers to detect features
  - Convolutional layers cut down on weights needed to be learned

# Convolutional Neural Network



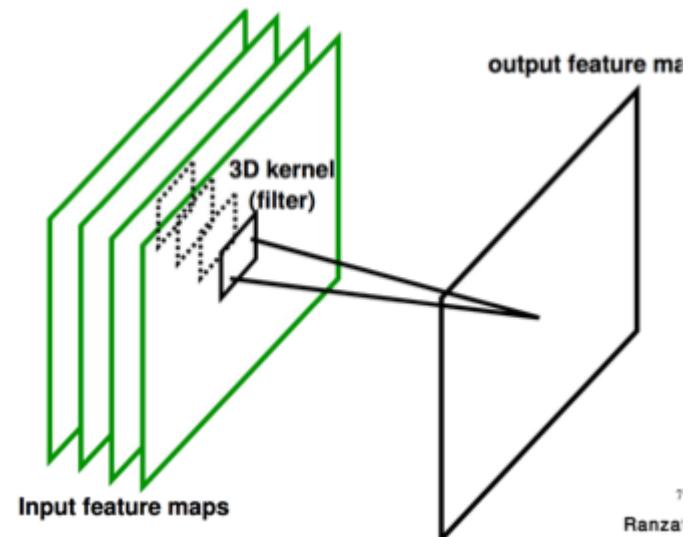
# Convolutional Neural Network

- Kernels (no not SVM kernels) but small regions of image.
- Kernels + Neurons = Convolution Layer



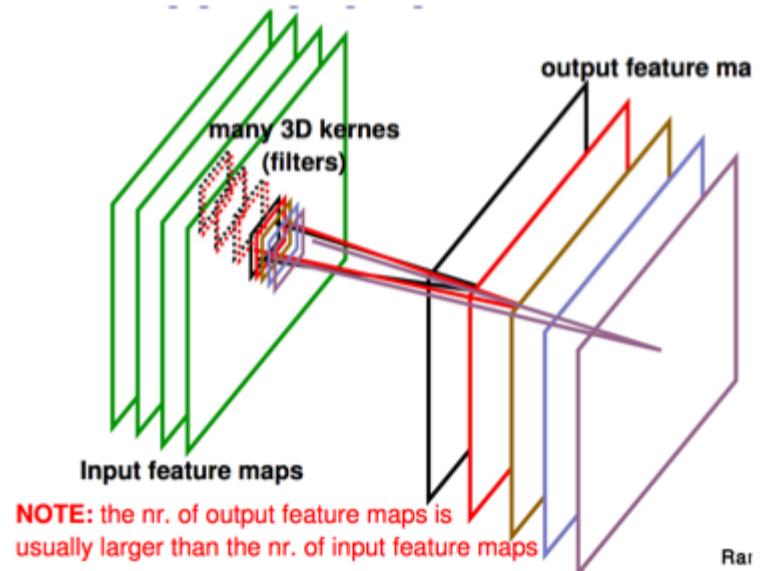
# Convolutional Neural Network

- If the input has 3 channels (R,G,B), 3 separate  $k$  by  $k$  filter is applied to each channel.
- Output of convolving 1 feature is called a *feature map*.

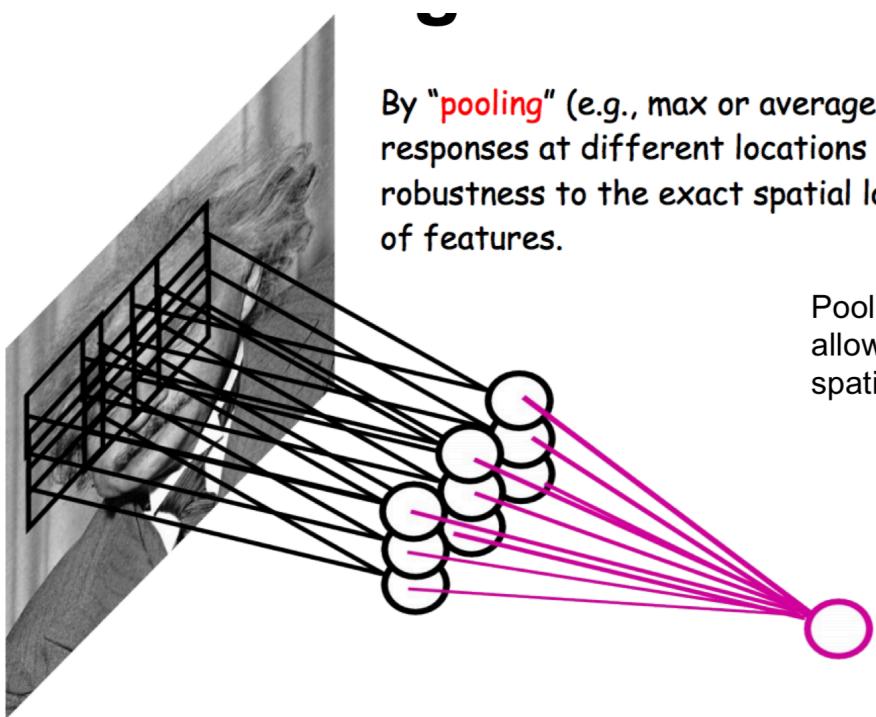


# Convolutional Neural Network

- Each filter detects features in the output of previous layer.
- So to capture different features, learn multiple filters
- This is an example of a “convolution layer”



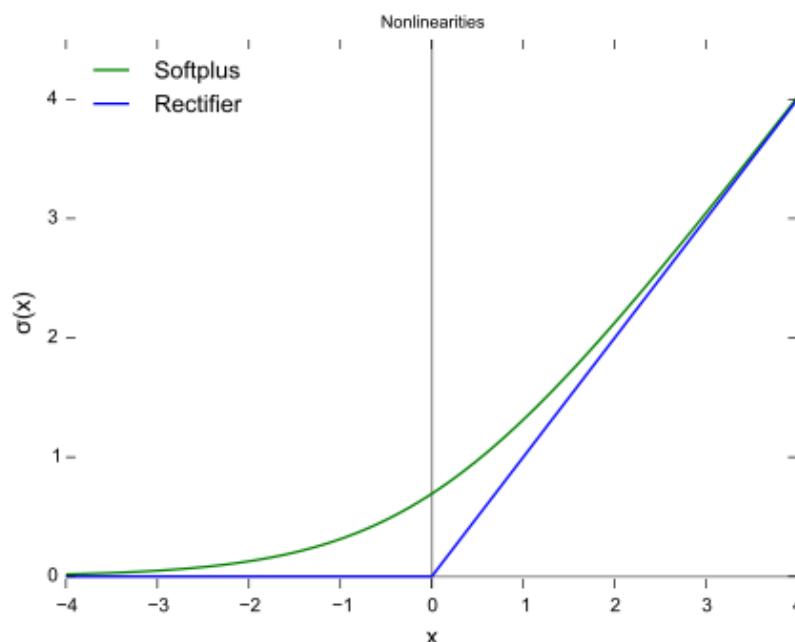
# Convolutional Neural Network



By “**pooling**” (e.g., max or average) filter responses at different locations we gain robustness to the exact spatial location of features.

Pooling also subsamples the image, allowing the next layer to look at larger spatial regions.

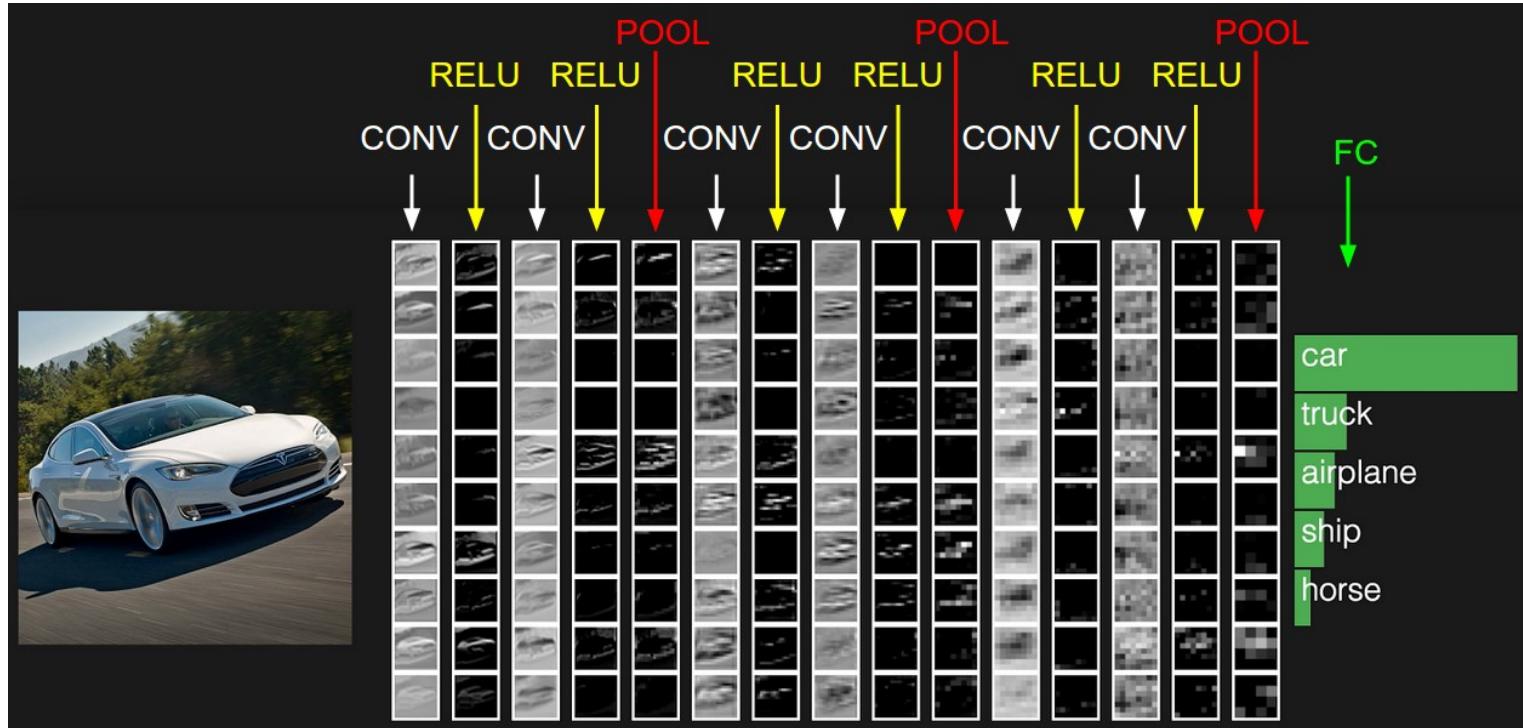
# Convolutional Neural Network



ReLU Layer (rectified linear unit) – non-linear unit

Nair, Vinod, and Geoffrey E. Hinton. "Rectified linear units improve restricted boltzmann machines." *Proceedings of the 27th international conference on machine learning (ICML-10)*. 2010.

# Convolutional Neural Network



Example of a Convolutional Neural Network

Demo: <http://cs231n.stanford.edu/>

# Convolutional Neural Network

- Nice tutorial on CNN implementation using Tensorflow and Python:
  - [https://www.tensorflow.org/tutorials/deep\\_cnn](https://www.tensorflow.org/tutorials/deep_cnn)
- No implementation of CNNs in R 😞