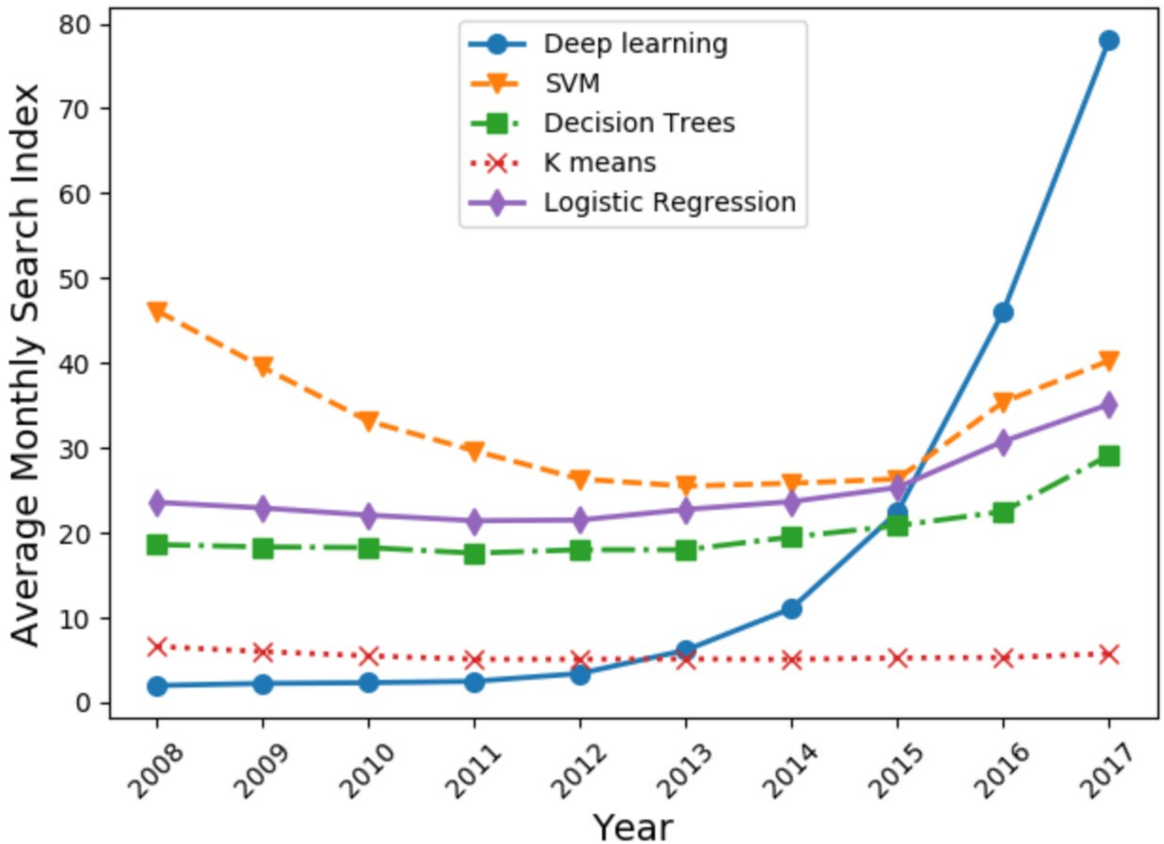
Lecture 4
Deep Learning

CS 180 – Intelligent Systems

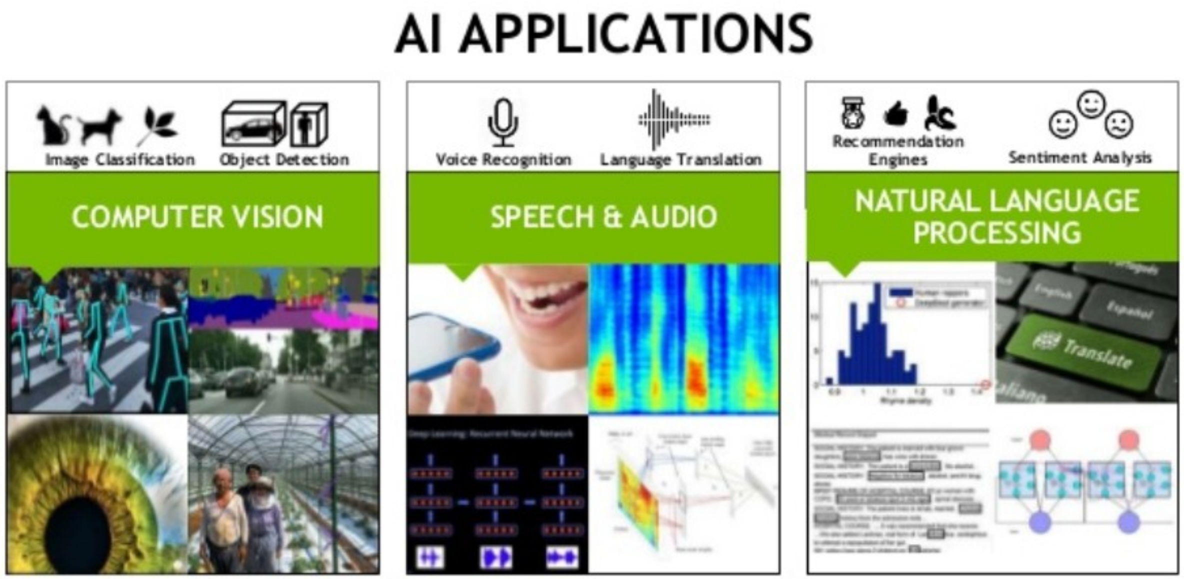
Dr. Victor Chen

Spring 2021

Deep learning gaining popularity



Deep learning today

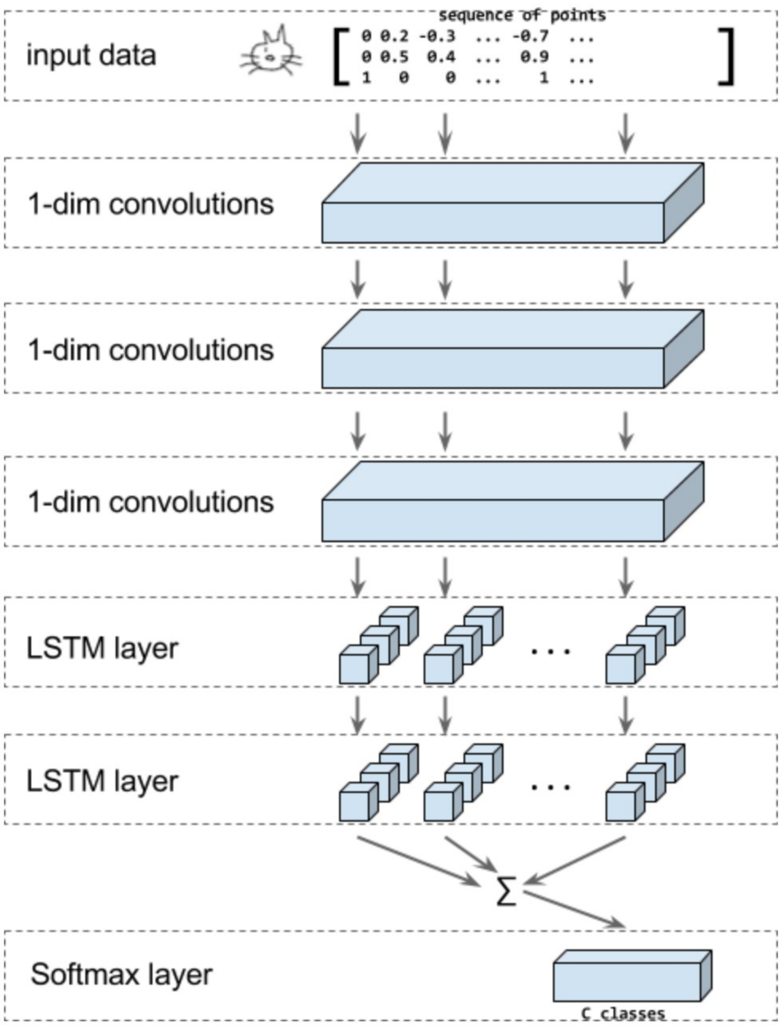


https://experiments.withgoogle.com/collection/ai

Demo

https://quickdraw.withgoogle.com/

The model takes sequences of strokes as input.



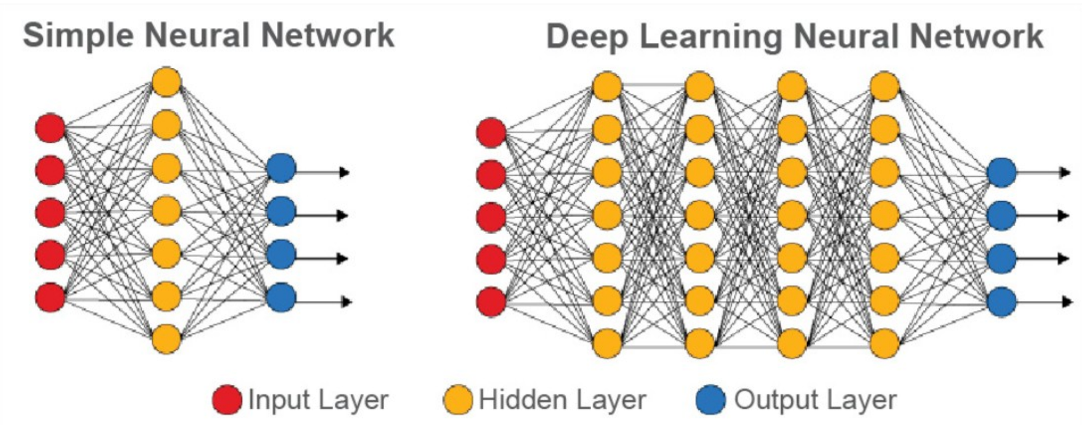
Deep learning in a nutshell

"Regular" neural networks usually have

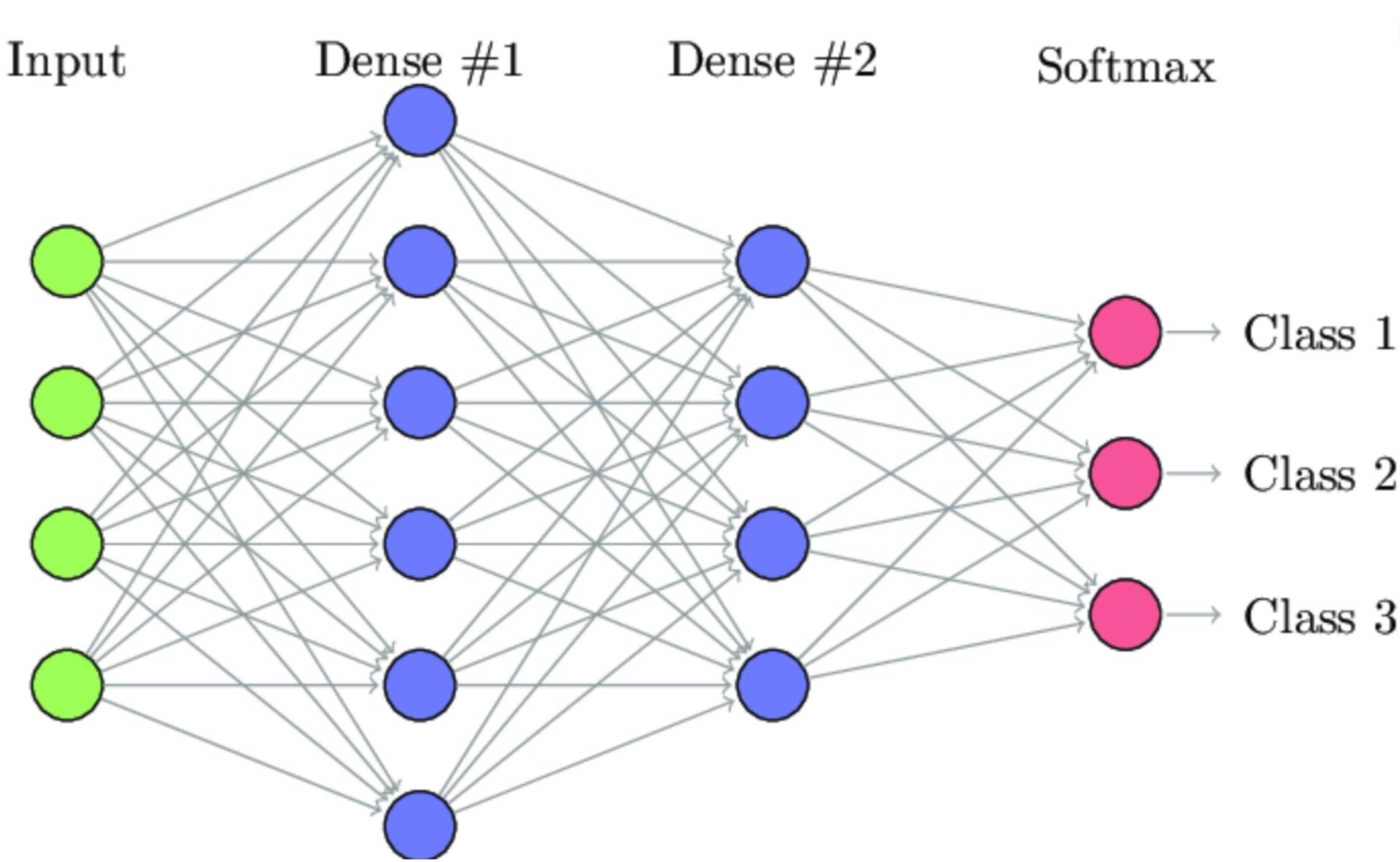
- one to two hidden layers
- are used for SUPERVISED learning.

"Deep" neural networks have

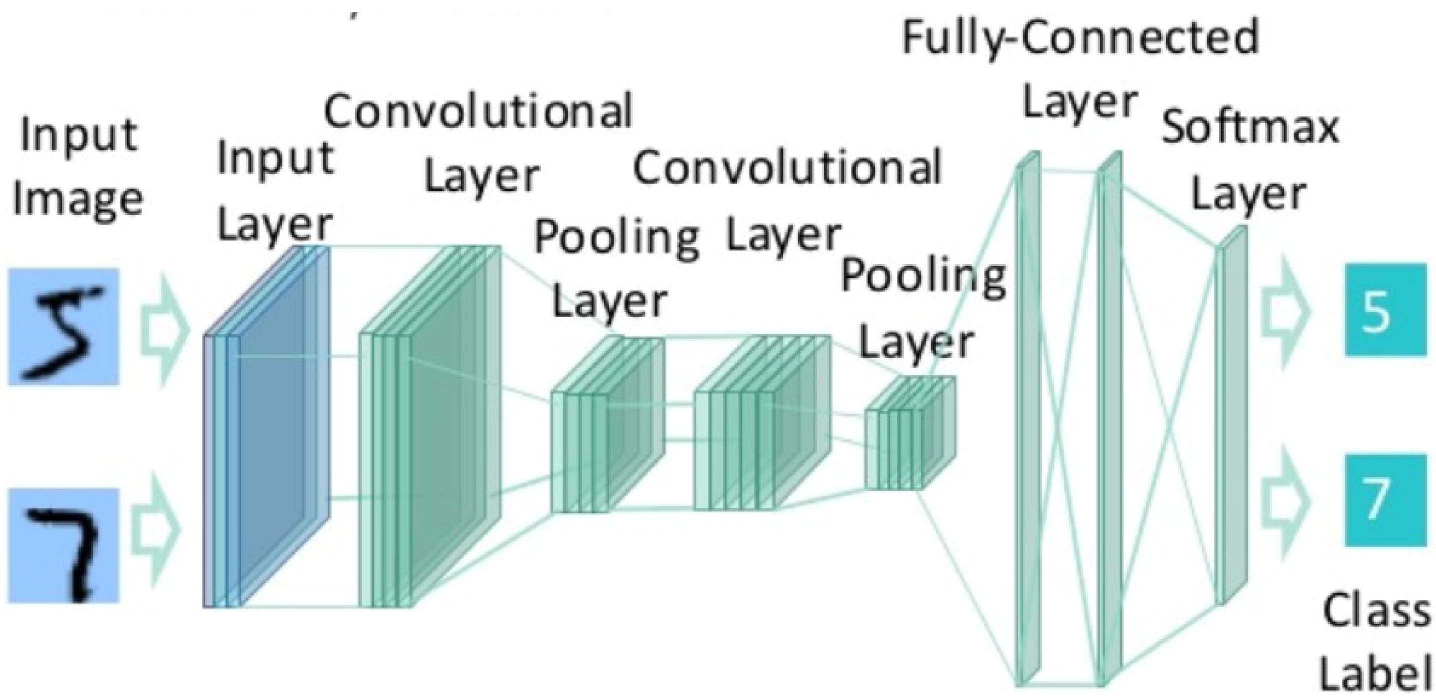
- more hidden layers
- can be used for both UNSUPERVISED and SUPERVISED learning.



Regular neural network (dense/fully-connect neural network)



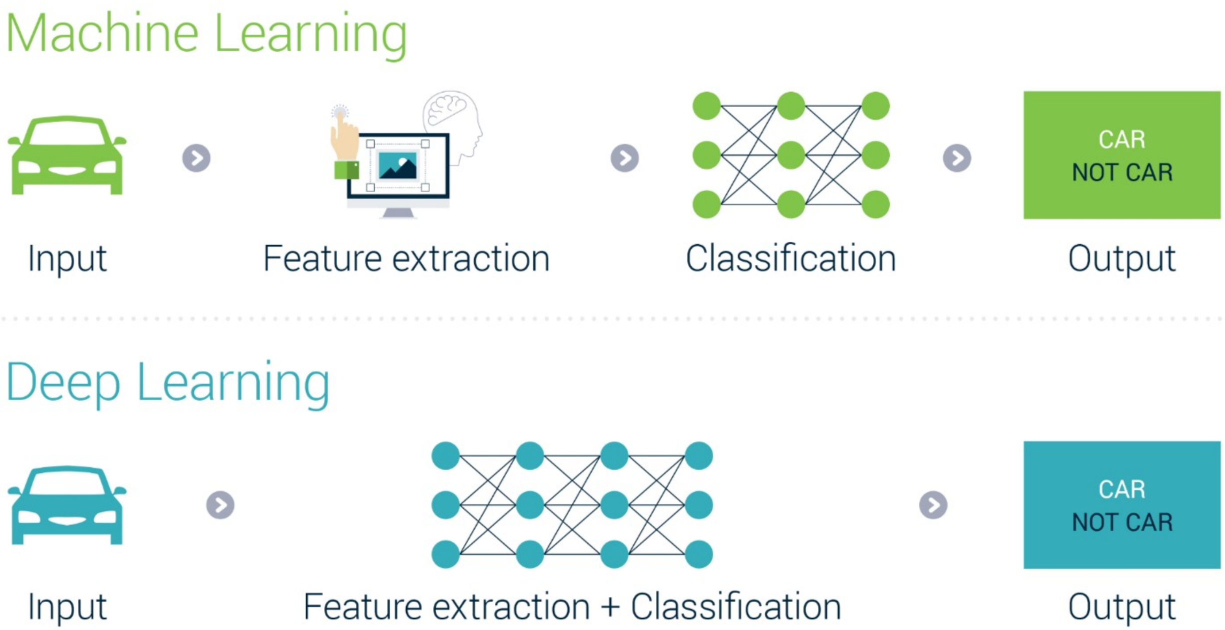
A deep network for recognizing handwritten digits



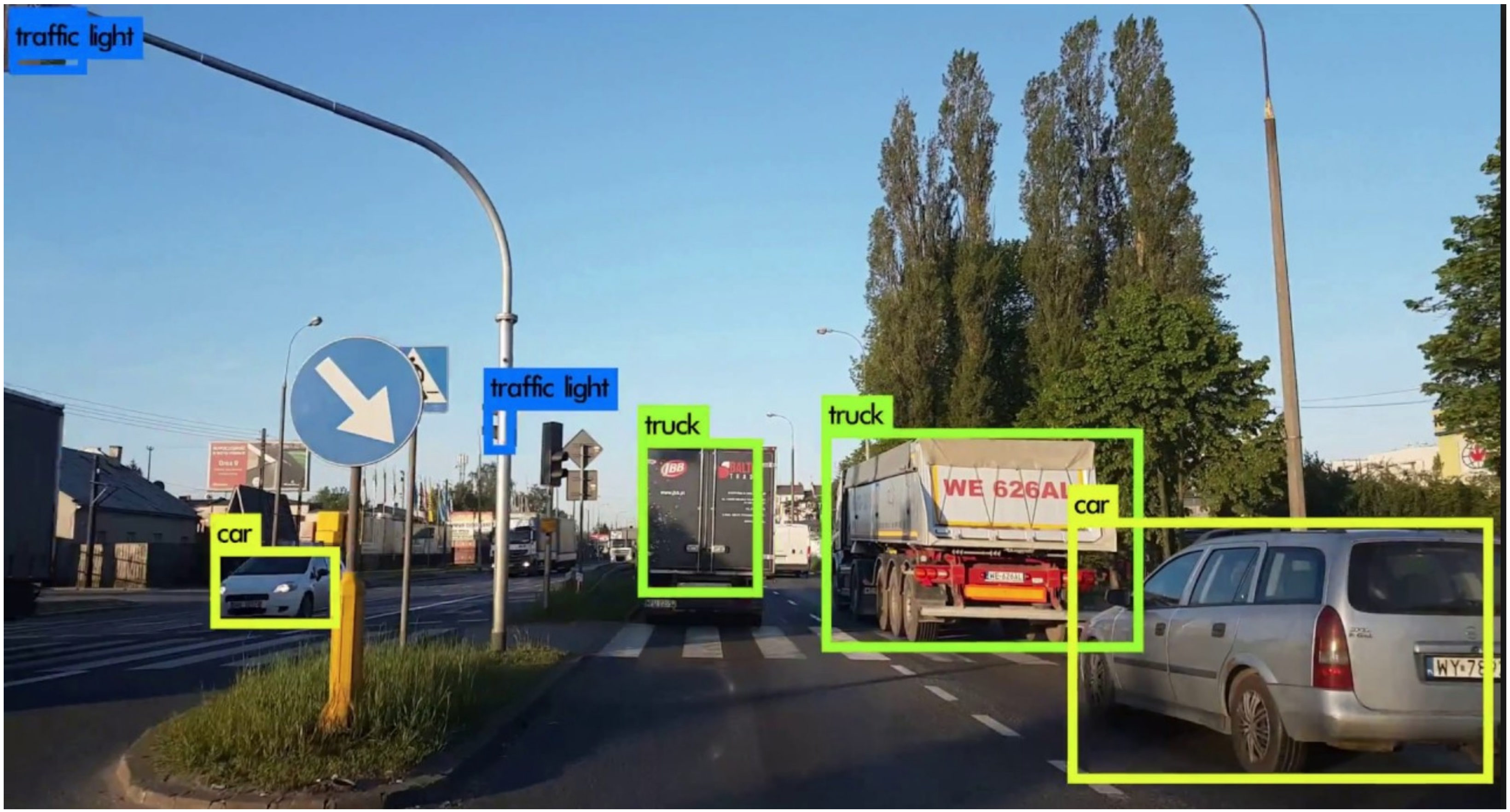
Why deep learning is generally better?

Feature detection using deep learning

“The hidden layers” can (1) detect “true underlying” features and (2) train model on those features



Object detection example



Feature detectors

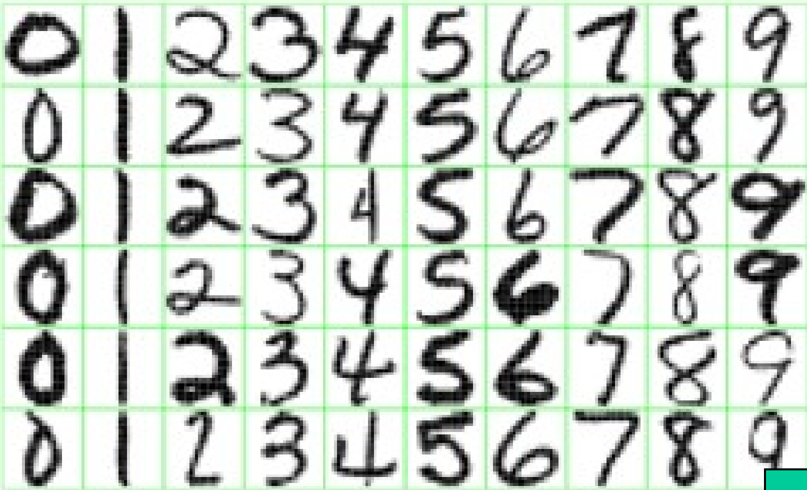
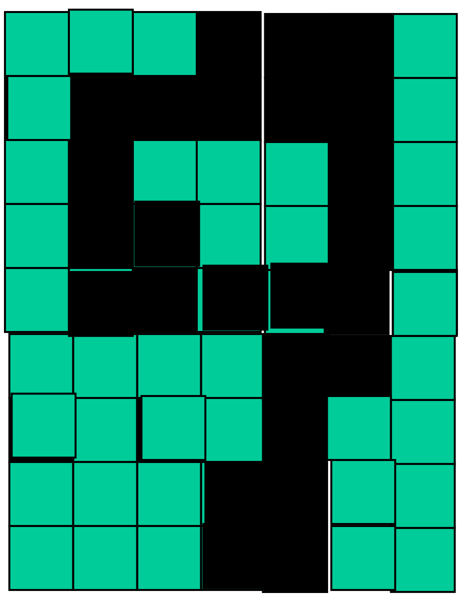
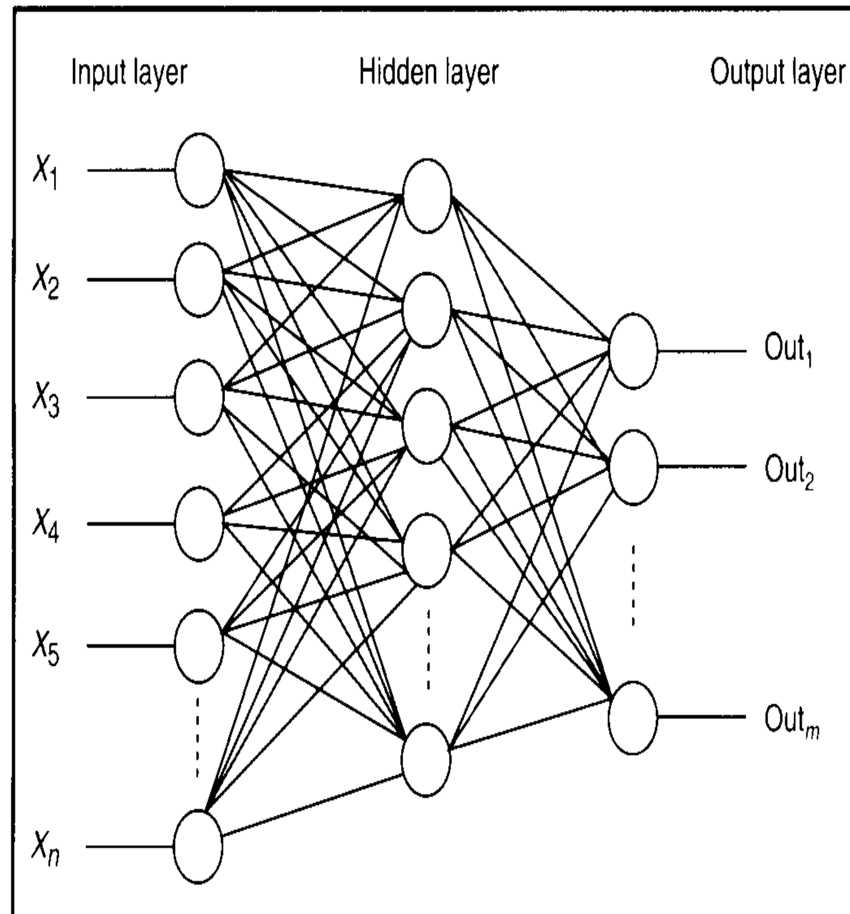


Figure 1.2: *Examples of handwritten digits from U.S. postal envelopes.*





What is this neuron doing?

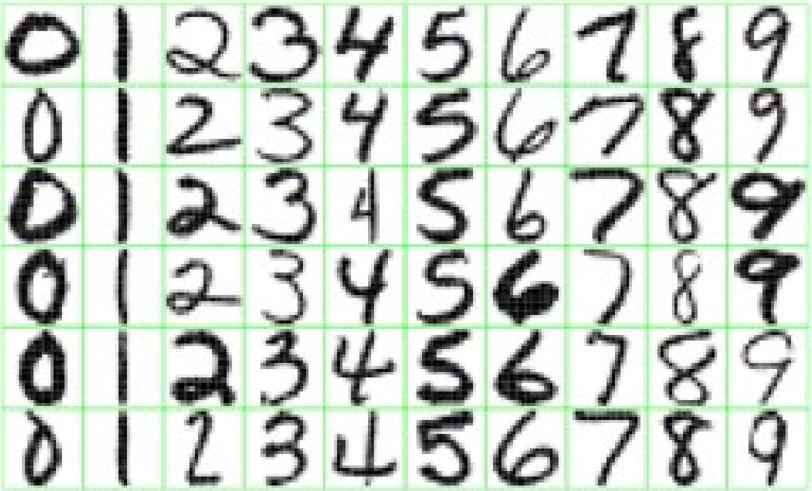
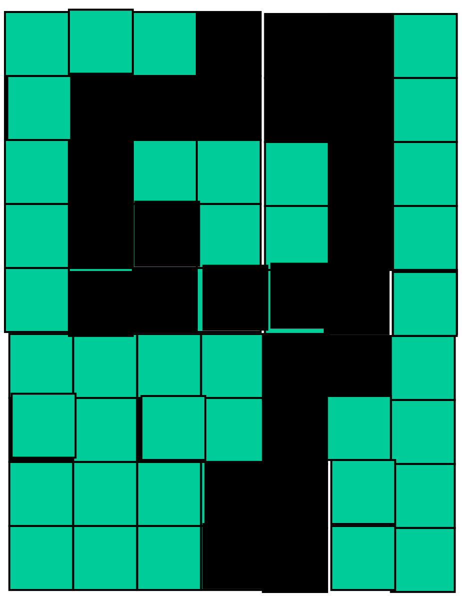
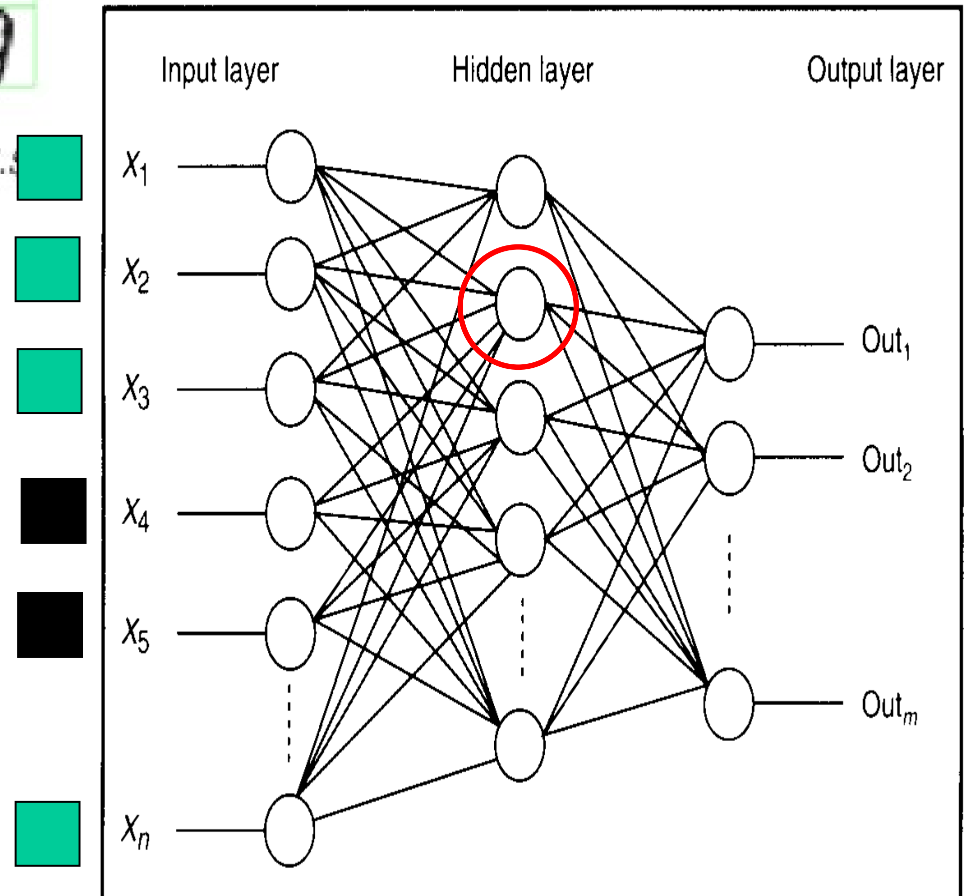
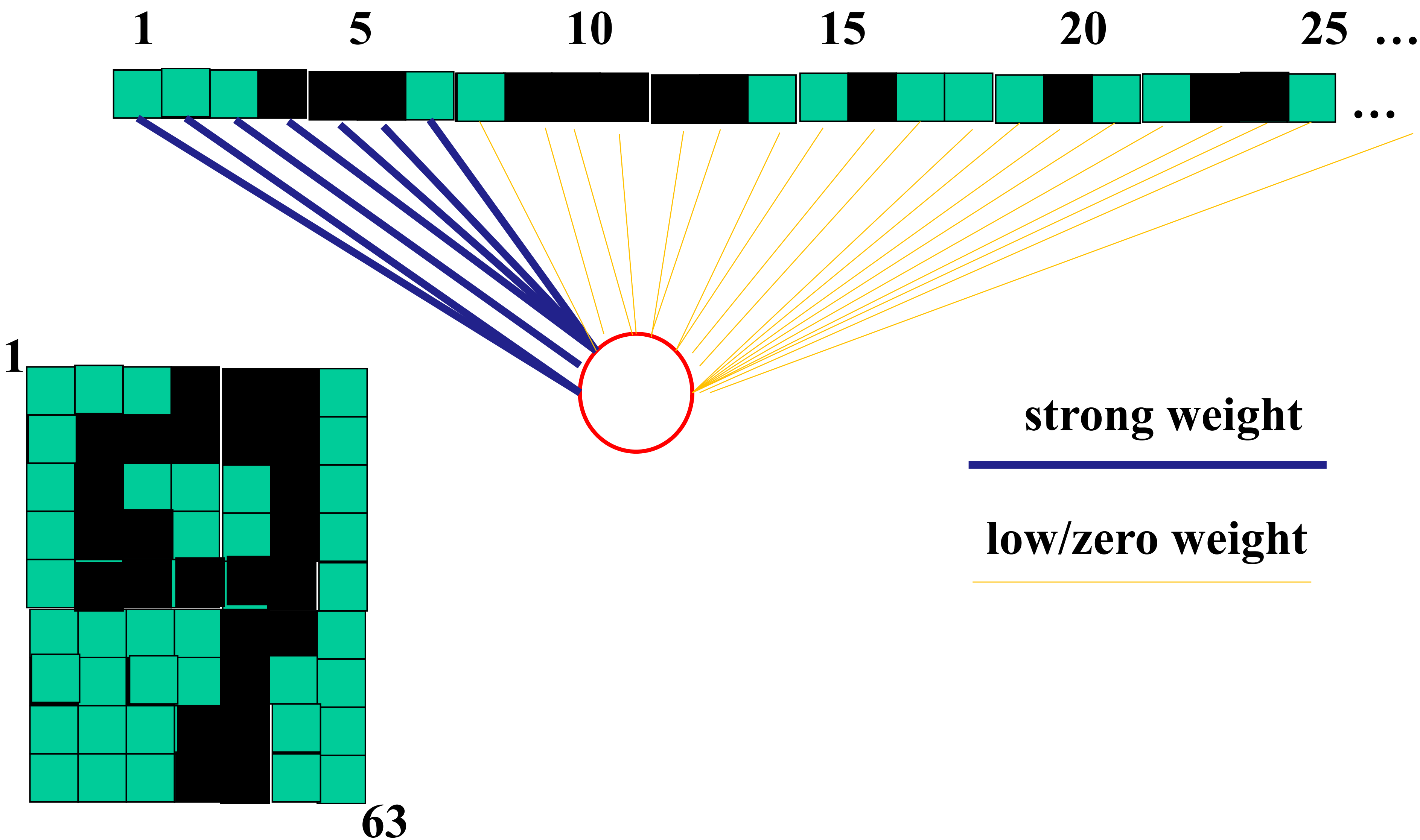


Figure 1.2: *Examples of handwritten digits from U.S*
postal envelopes.

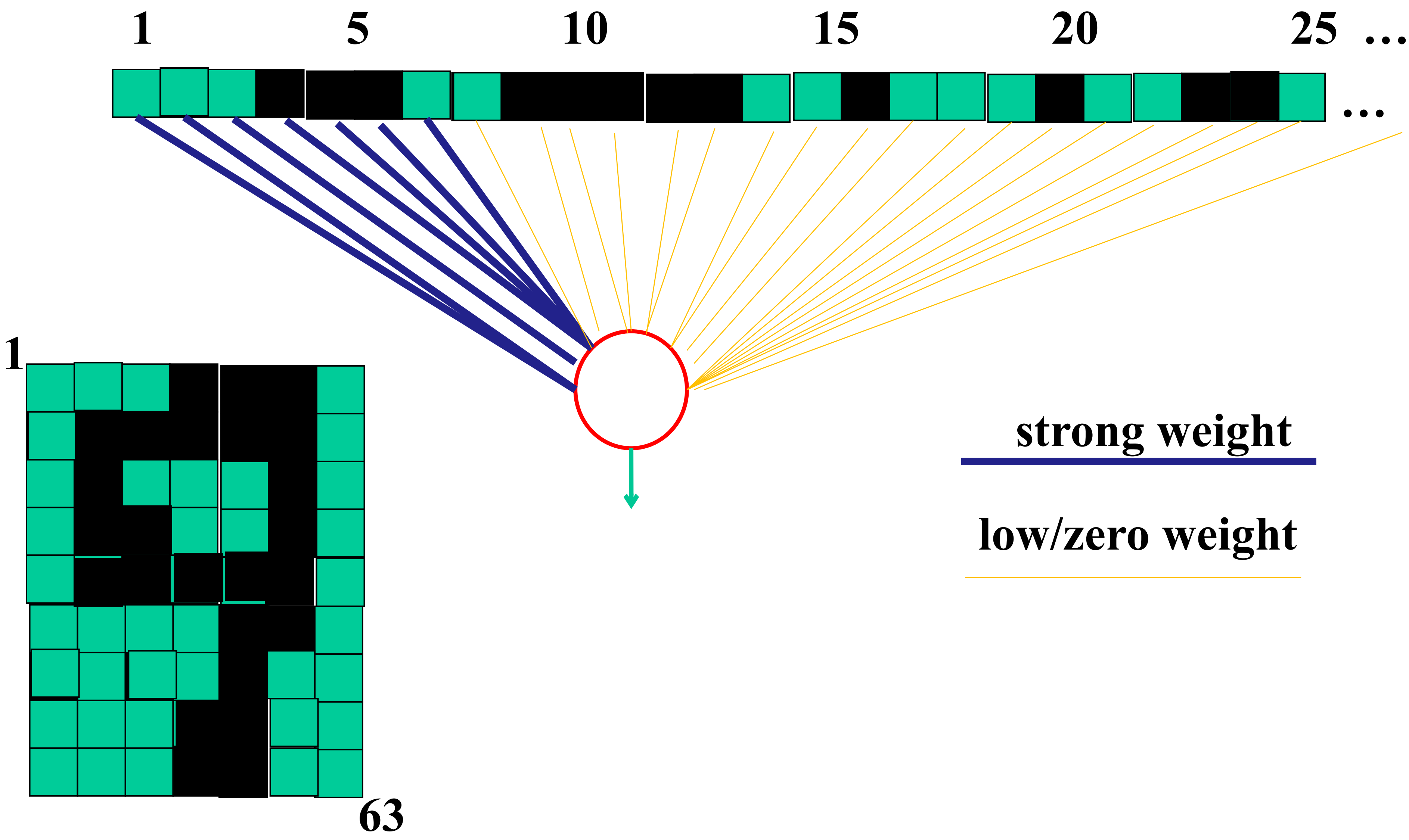




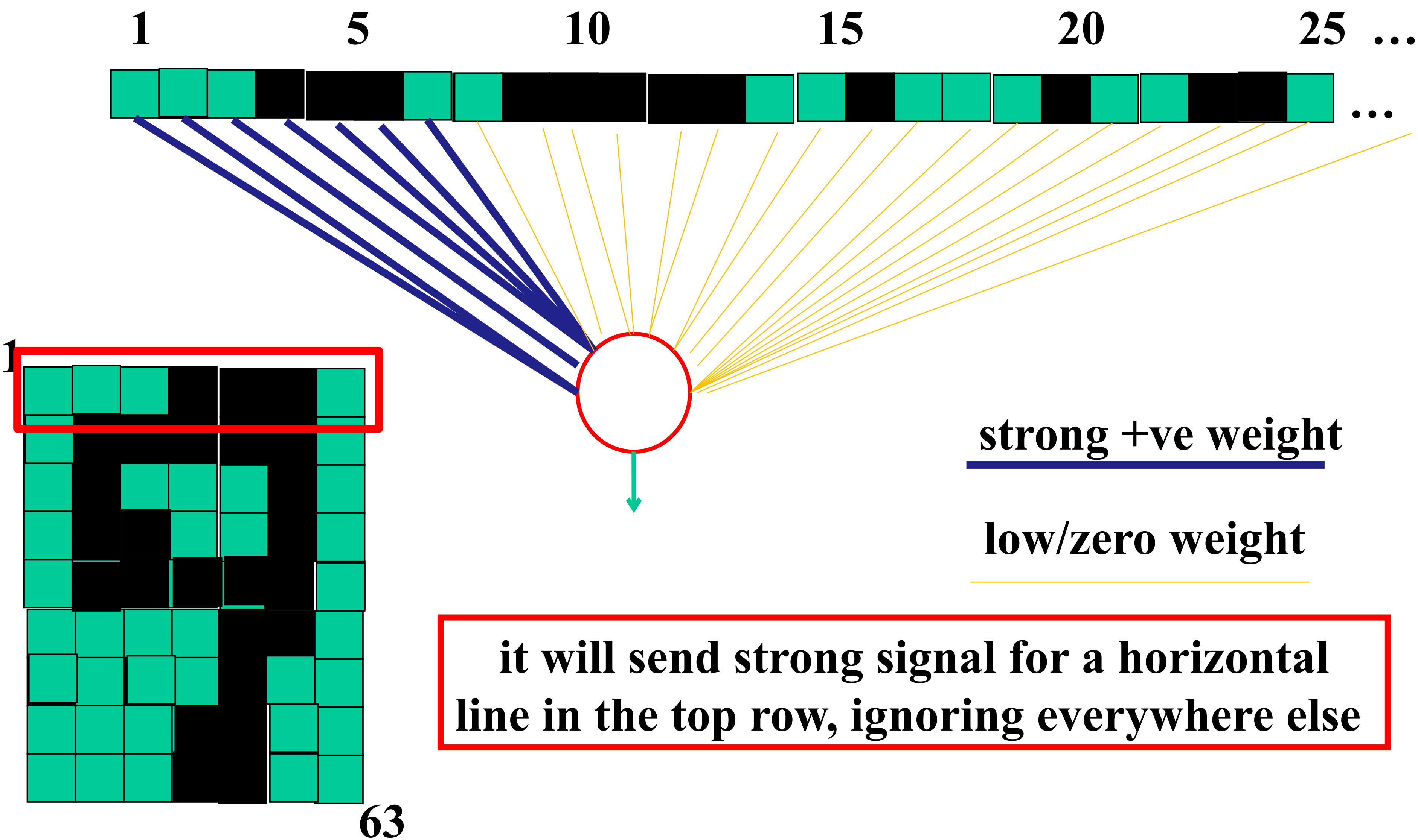
Hidden layer neurons become *feature detectors*



Hidden layer neurons become *feature detectors*

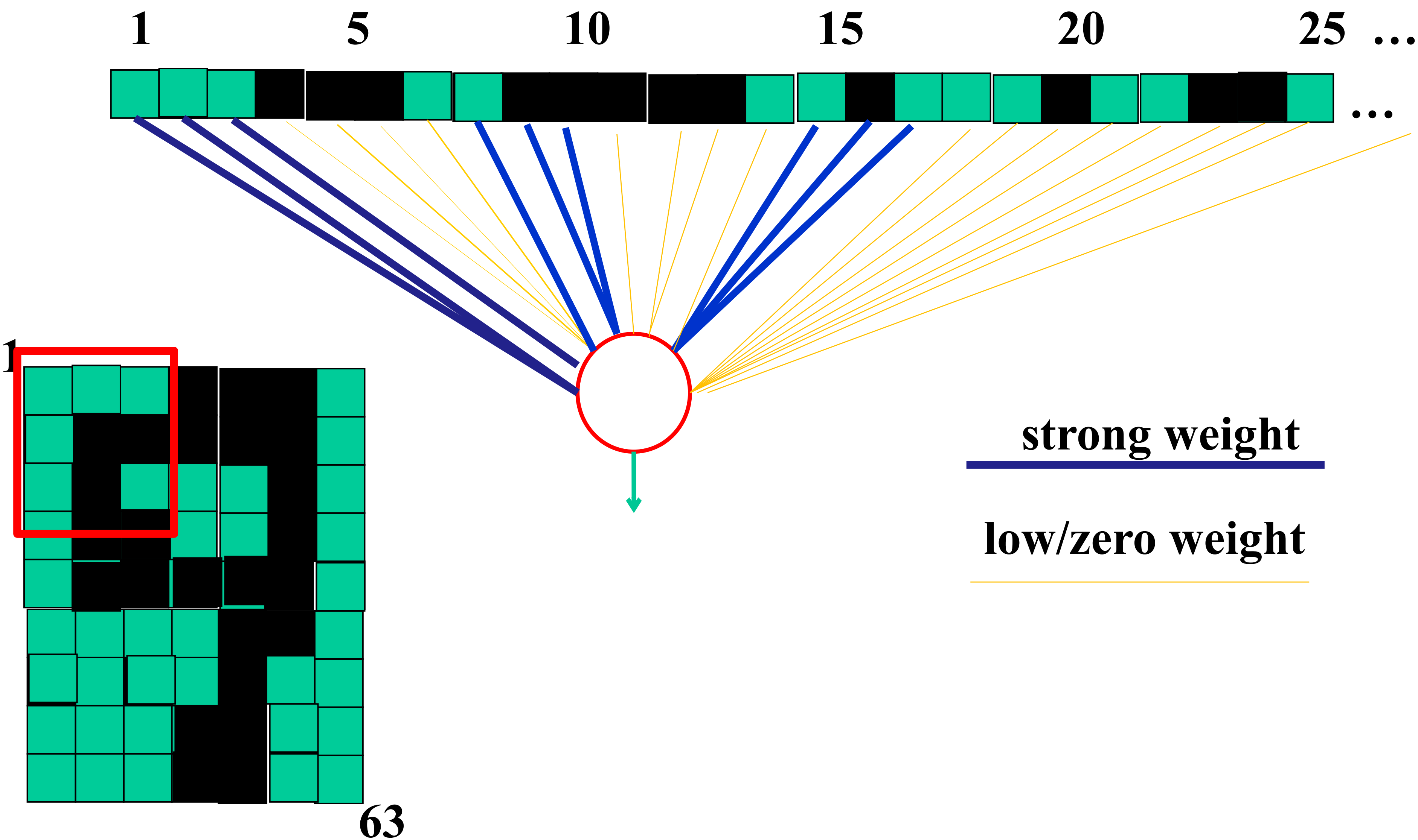


Hidden layer neurons become *feature detectors*

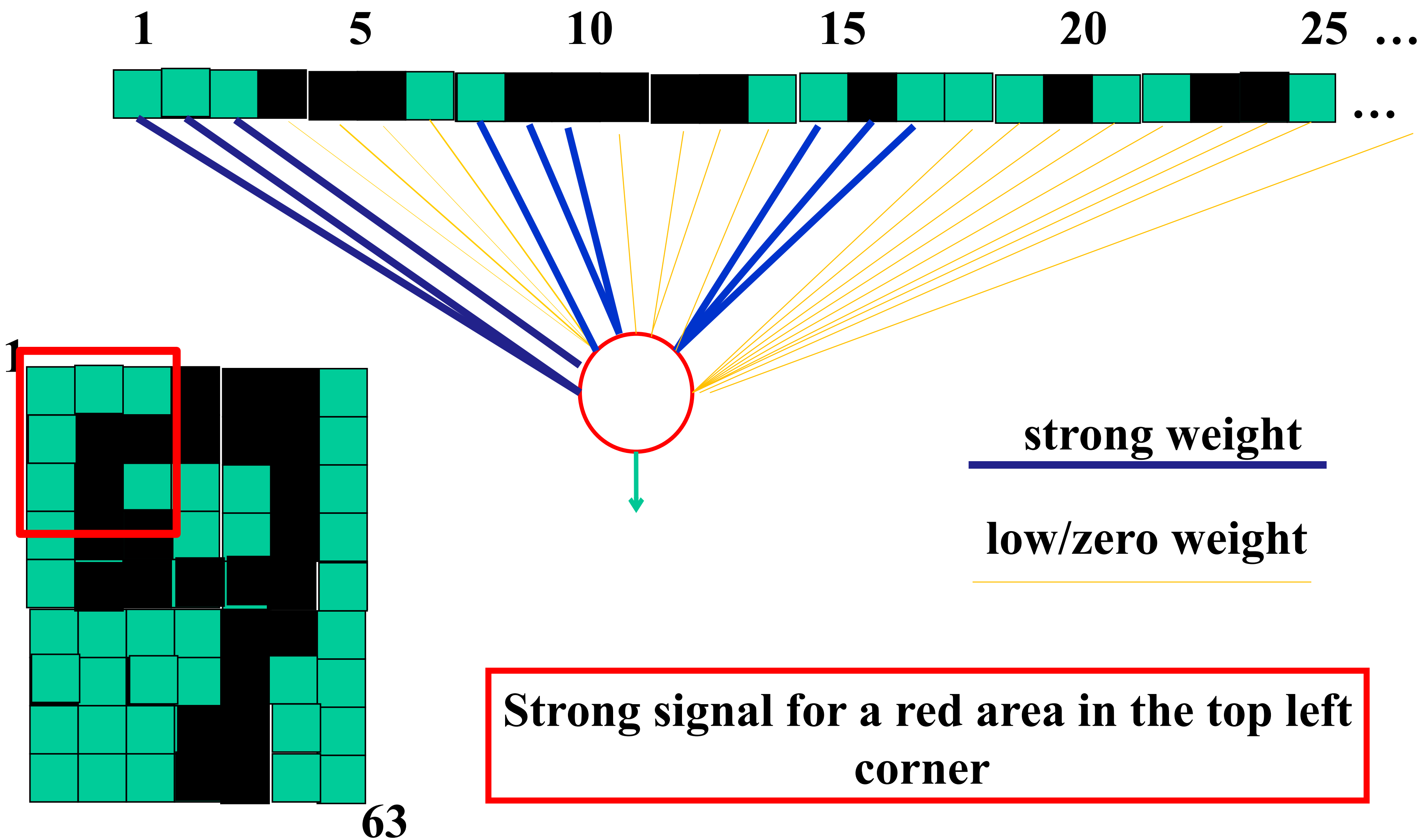


it will send strong signal for a horizontal line in the top row, ignoring everywhere else

Another neuron



Another neuron



Strong signal for a red area in the top left corner

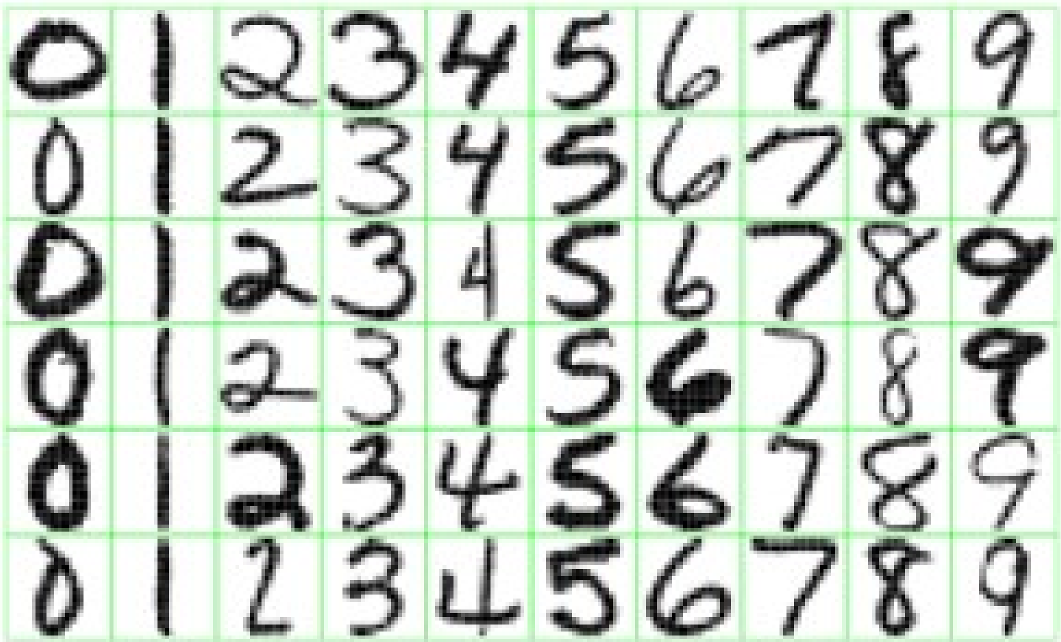


Figure 1.2: *Examples of handwritten digits from U.S. postal envelopes.*

What features do you think would make a good model?

1

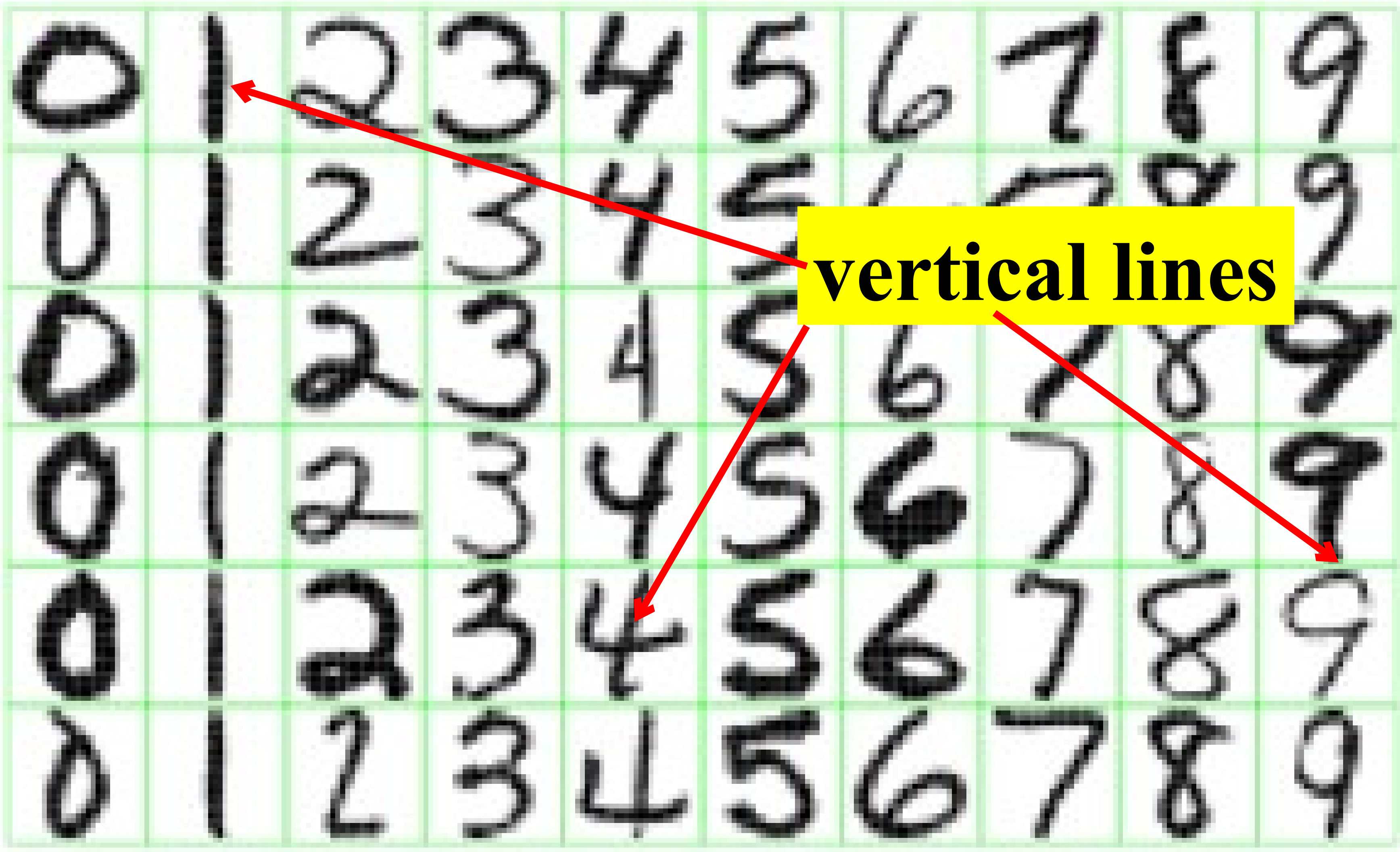


Figure 1.2: *Examples of handwritten digits from U.S. postal envelopes.*

1

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

1

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

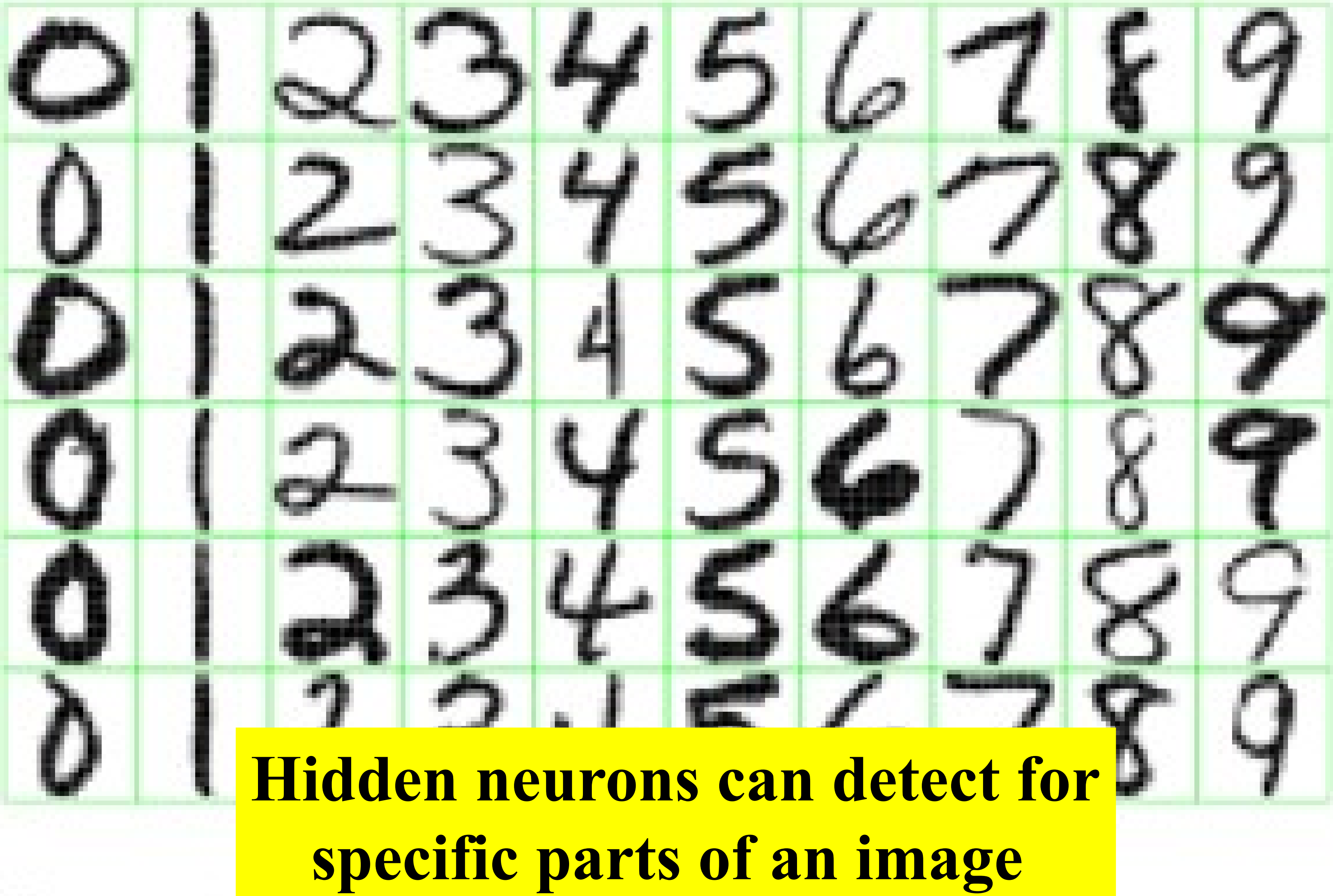
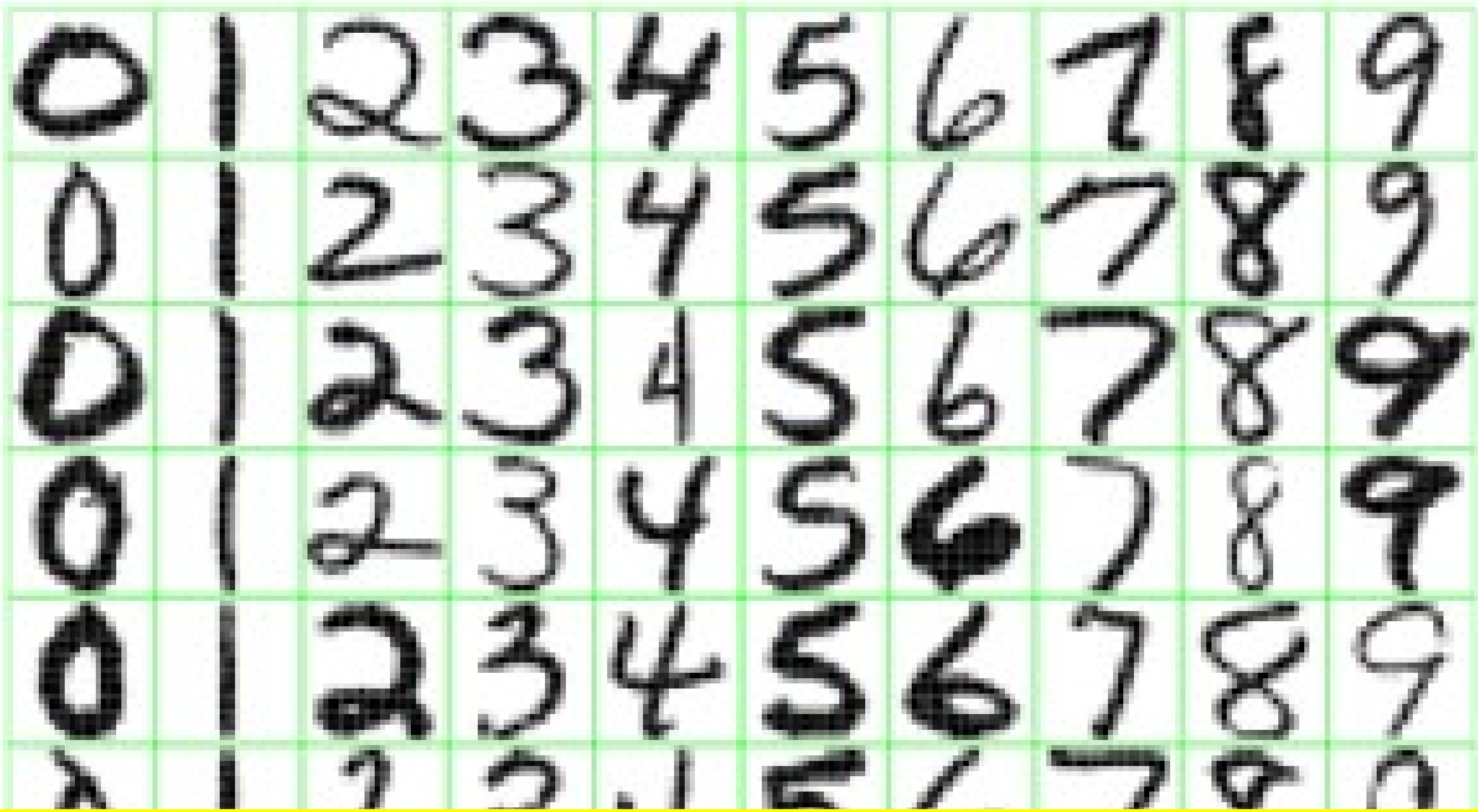


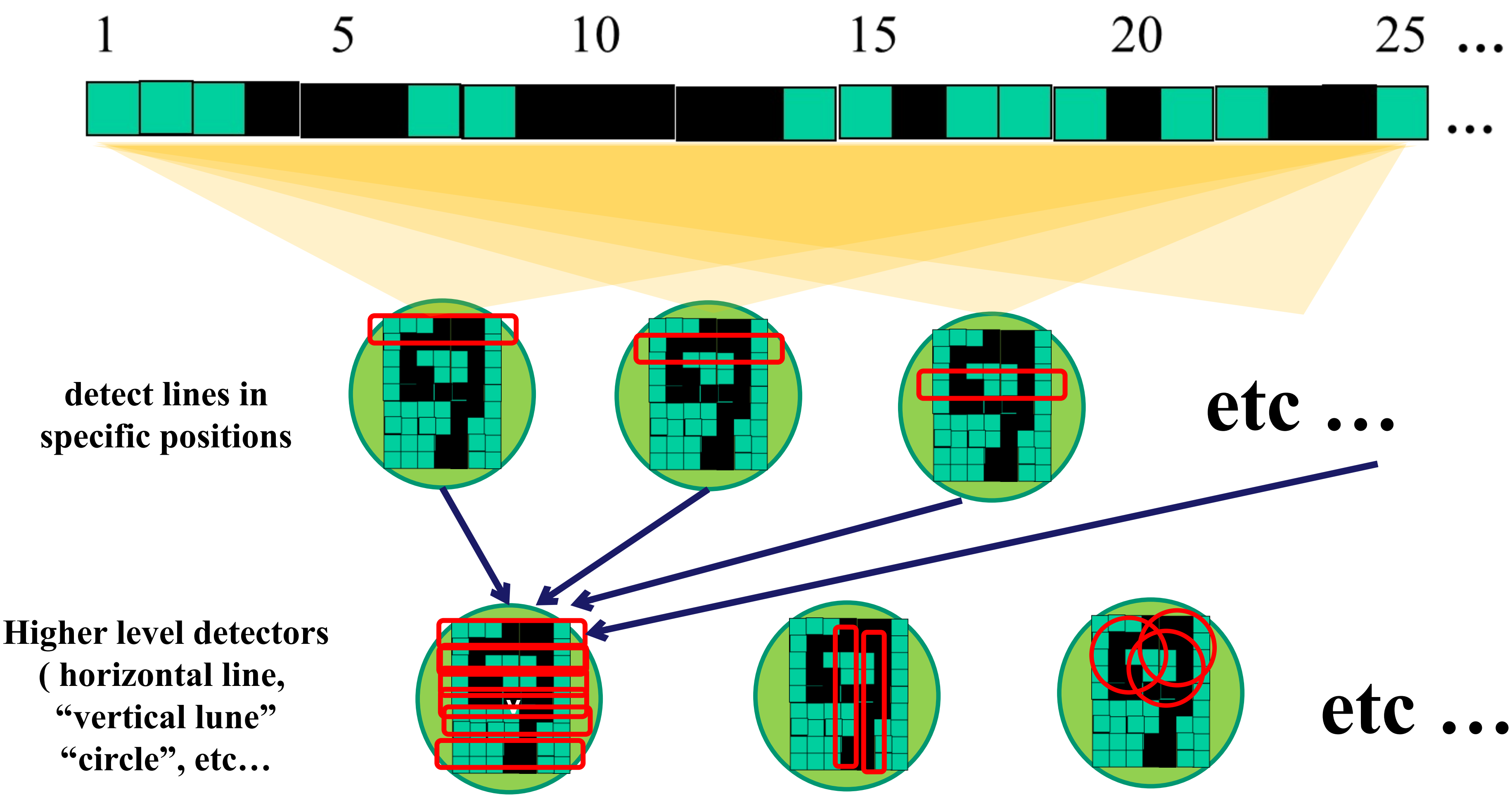
Figure 1.2: *Examples of handwritten digits from U.S. postal envelopes.*



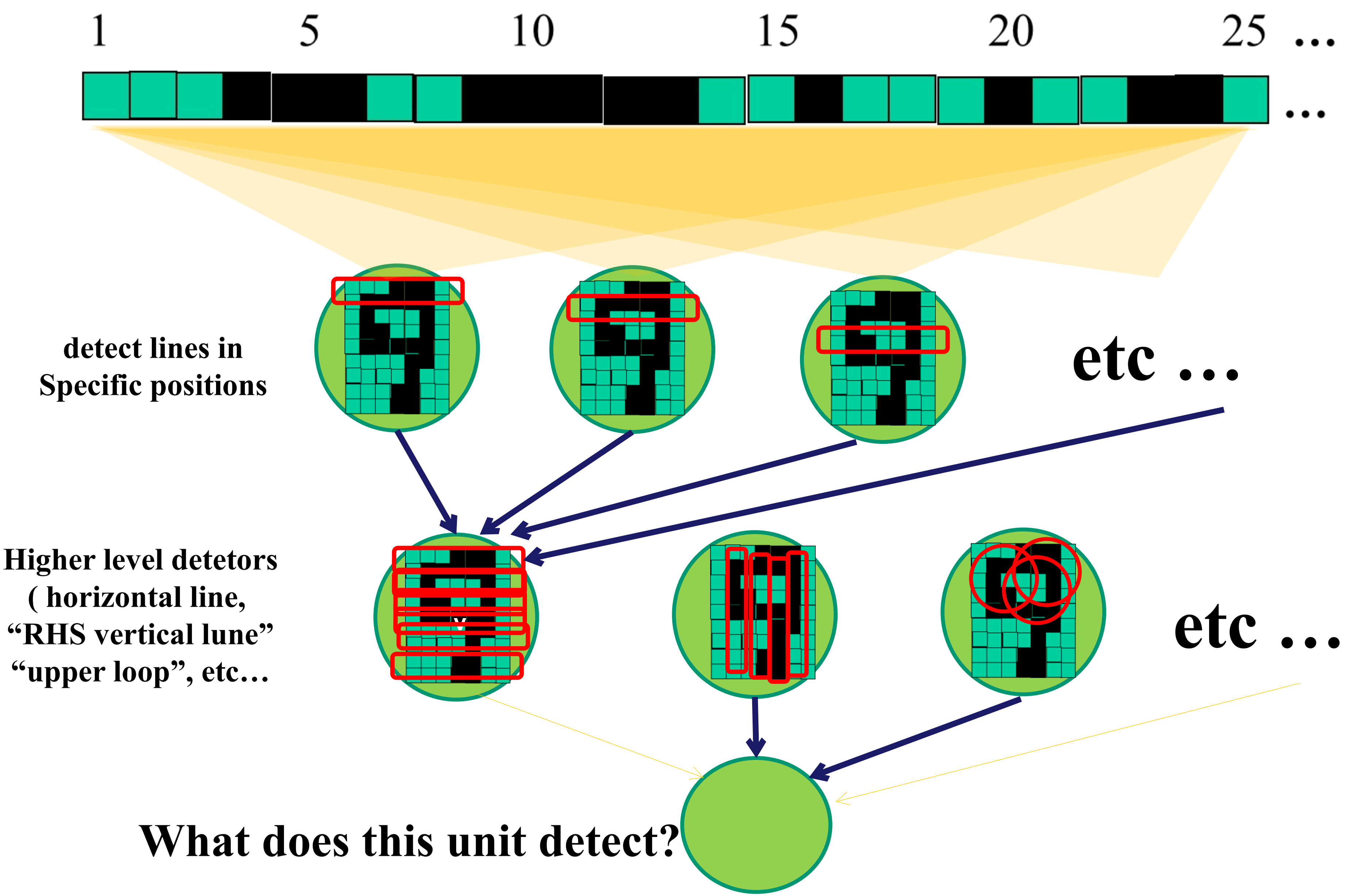
Hidden neurons can detect for
specific parts of an image
But what about correlation among different parts?

postal envelopes.

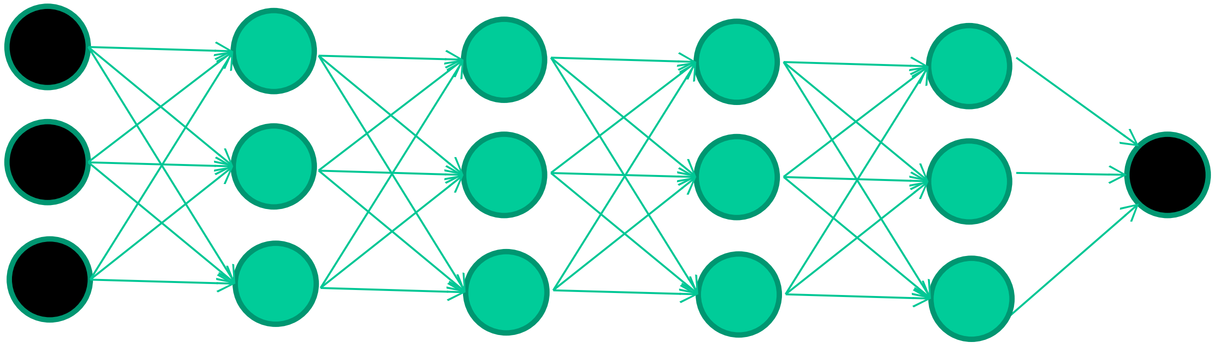
Successive layers can learn higher-level features



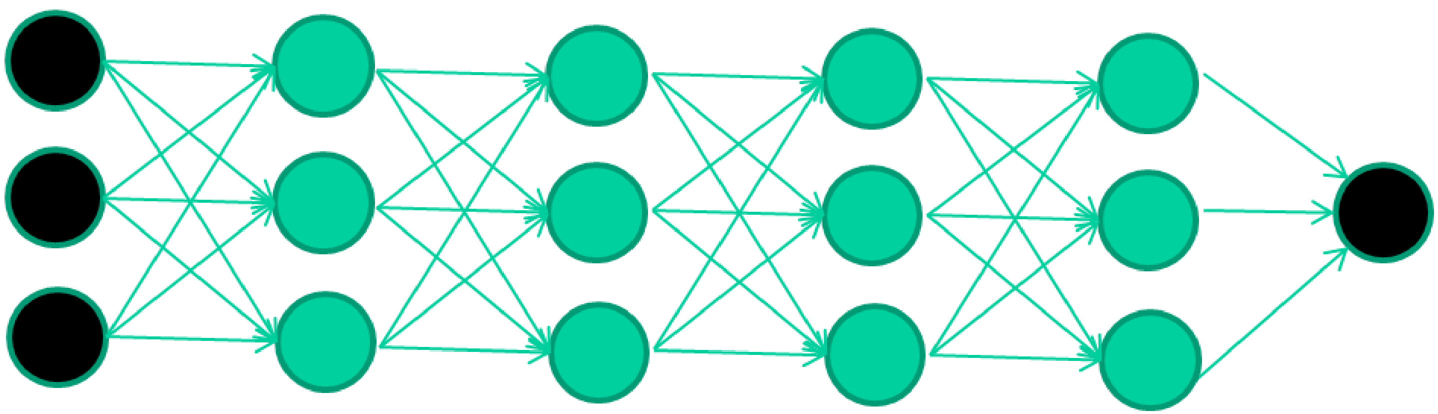
Successive layers can learn higher-level features



That is how *multiple layers make sense*

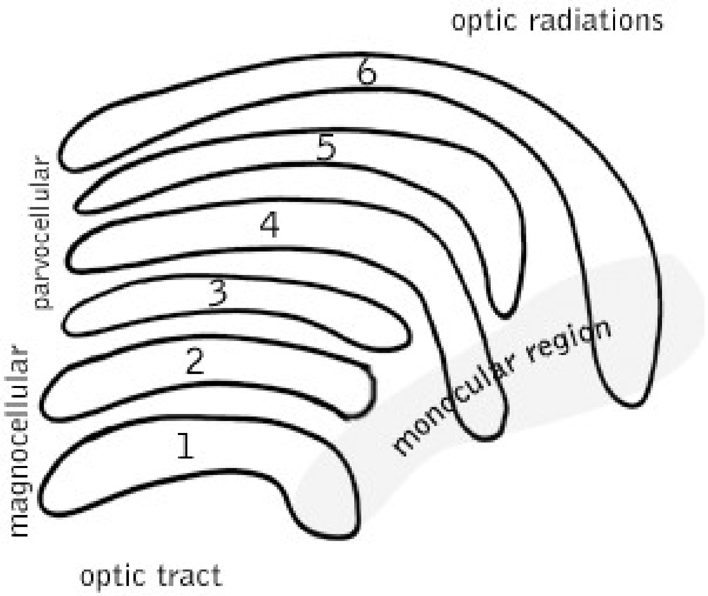


We want to **stack many hidden layers** to capture "true" features from lower levels to higher levels



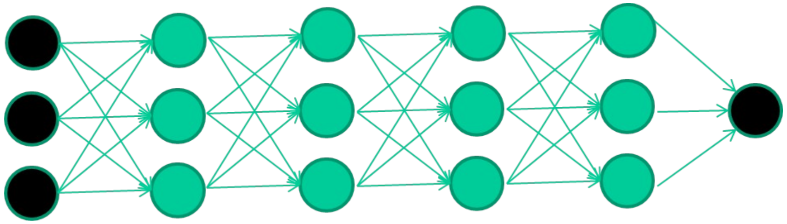
Successive layers can learn higher-level features

Your brain works that way

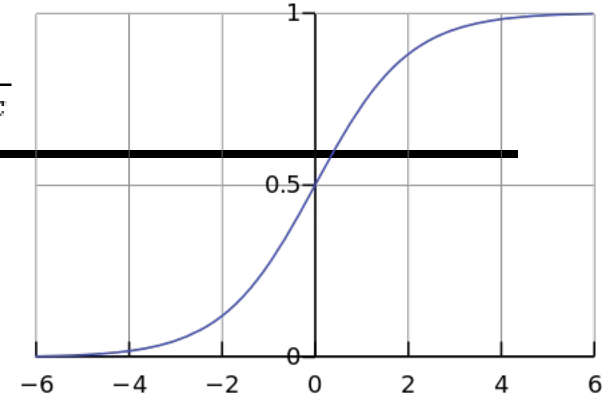


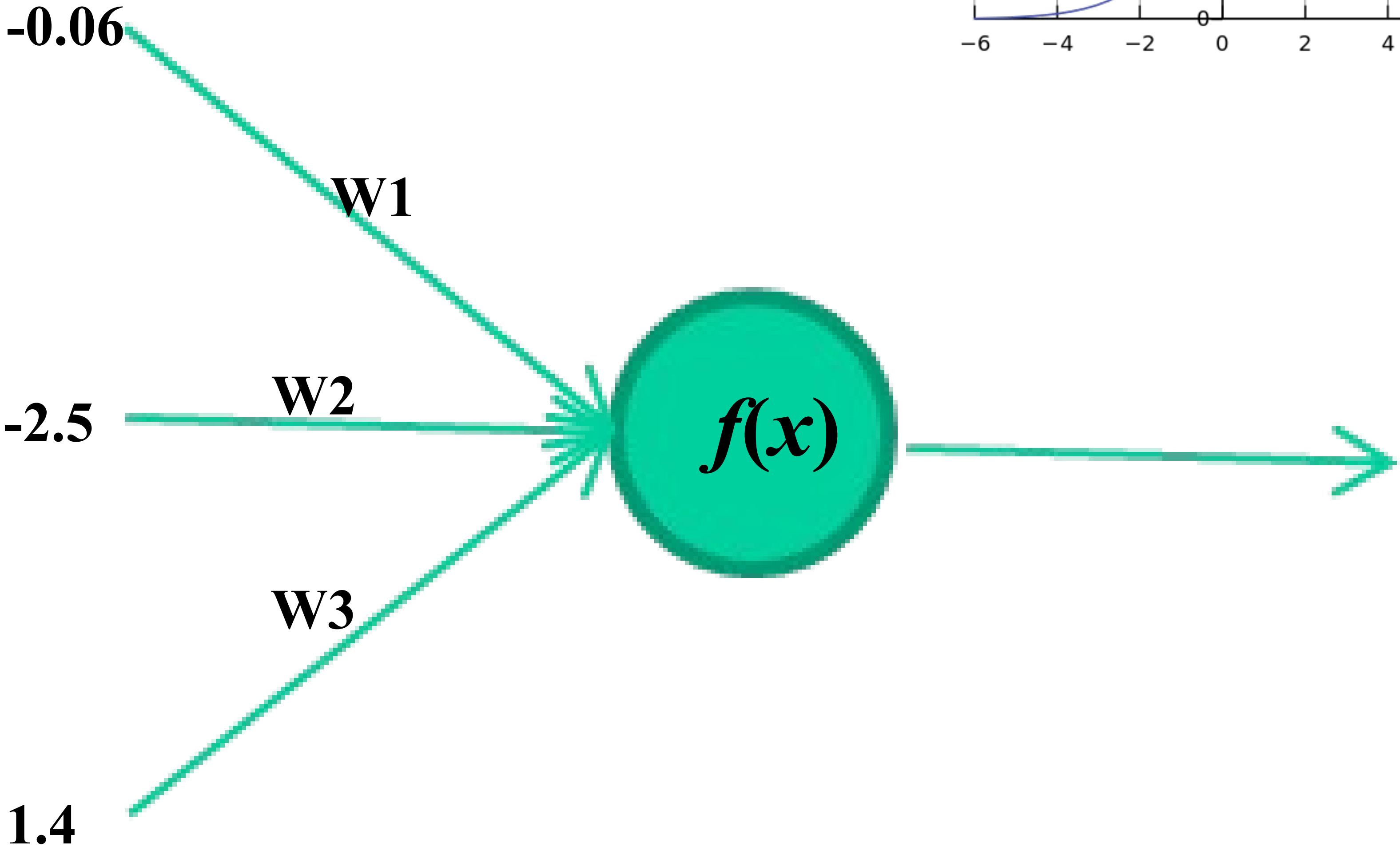
Review

How to train a neural network?

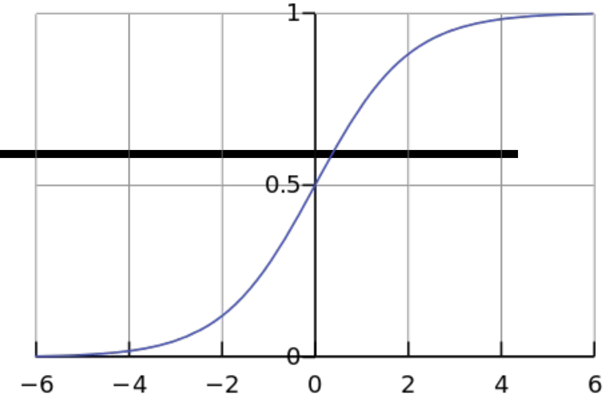


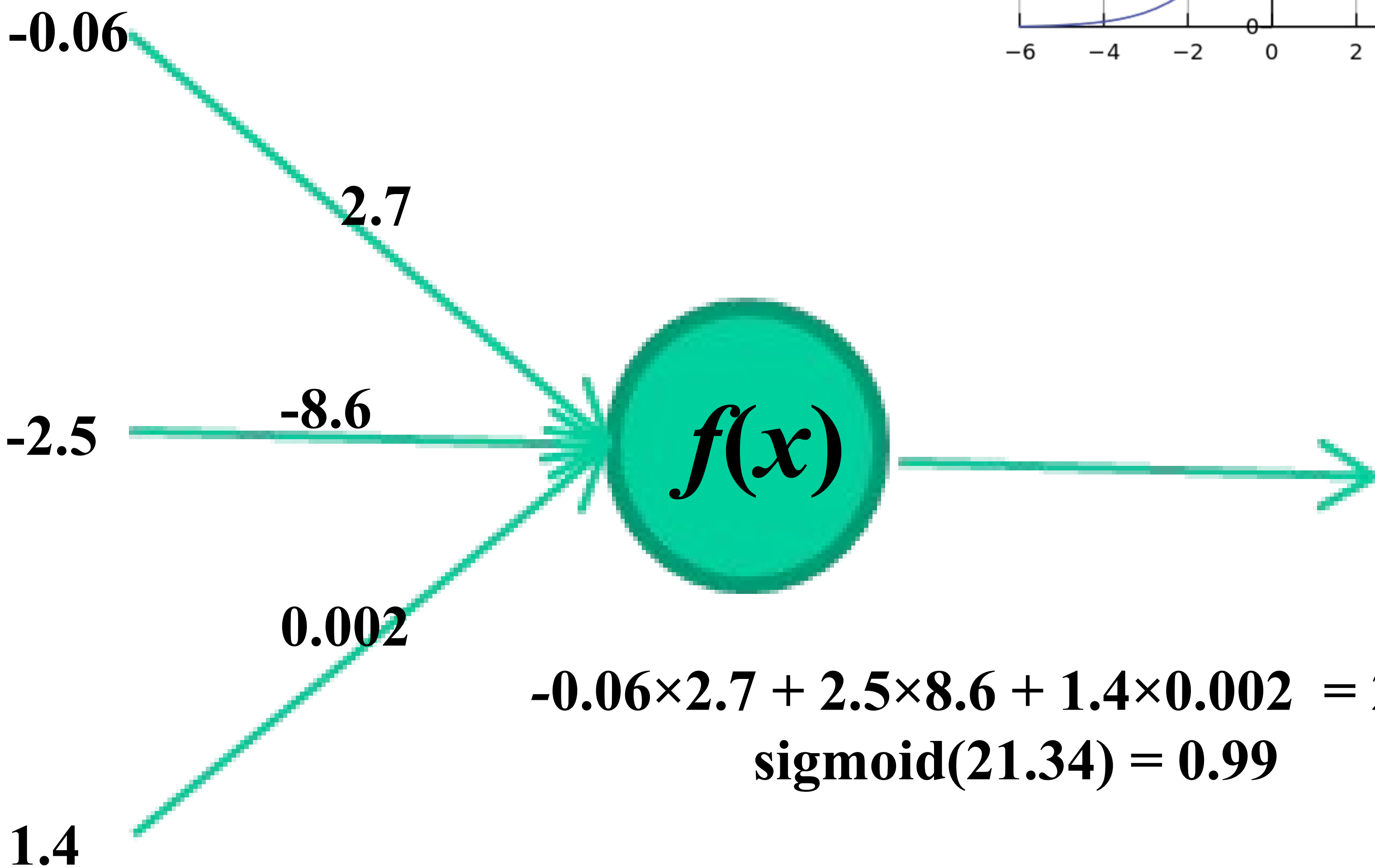
$$f(x) = \frac{1}{1+e^{-x}}$$





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





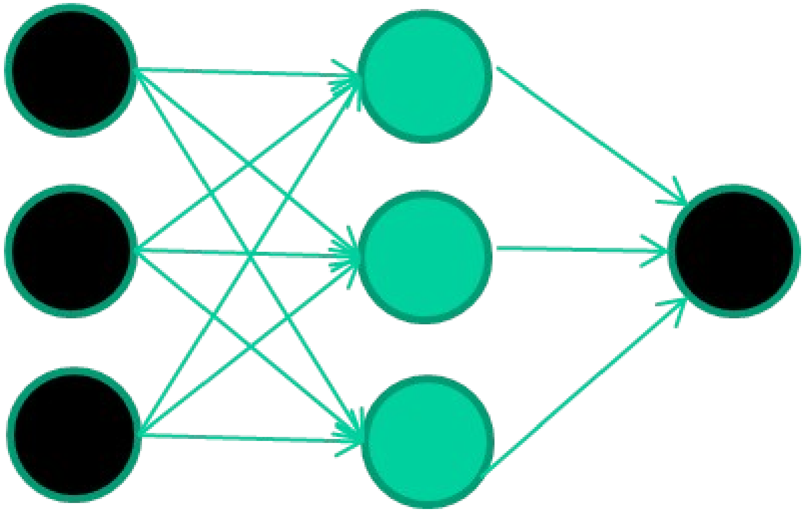
-0.06×2.7 + 2.5×8.6 + 1.4×0.002 = 21.34

sigmoid(21.34) = 0.99

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 …

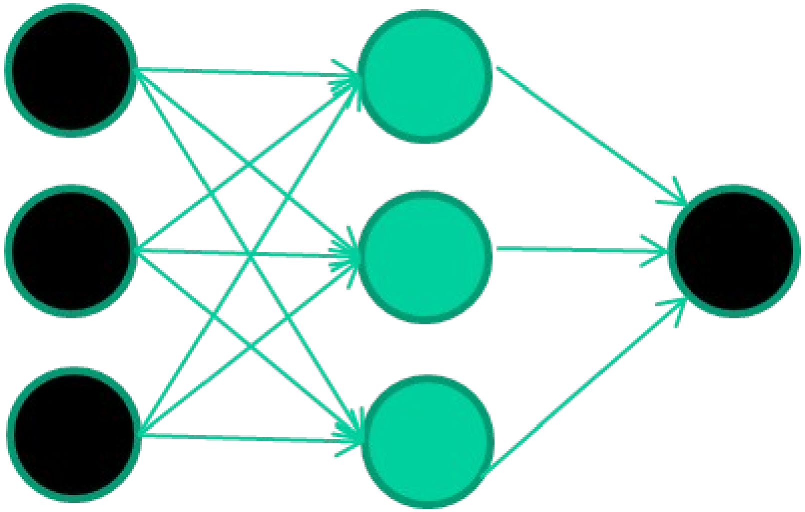


Training data

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 …

Initialize model with random weights

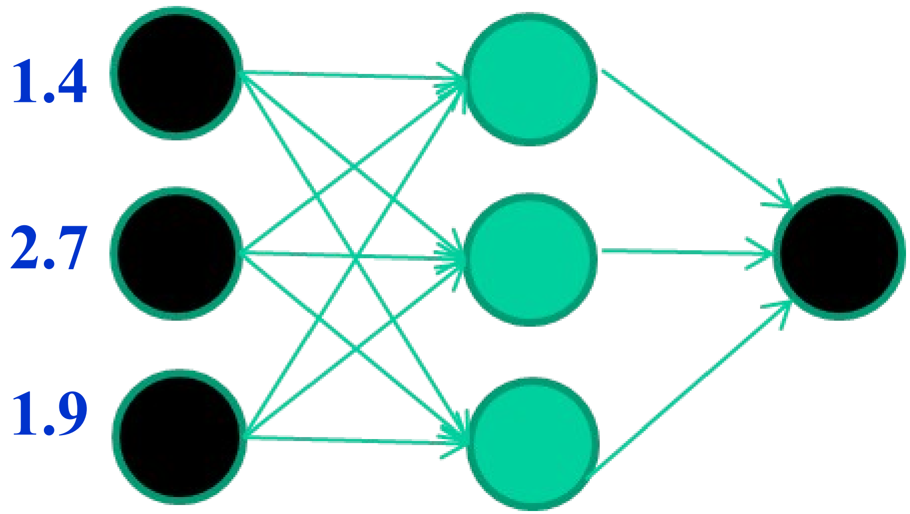


Training data

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 record

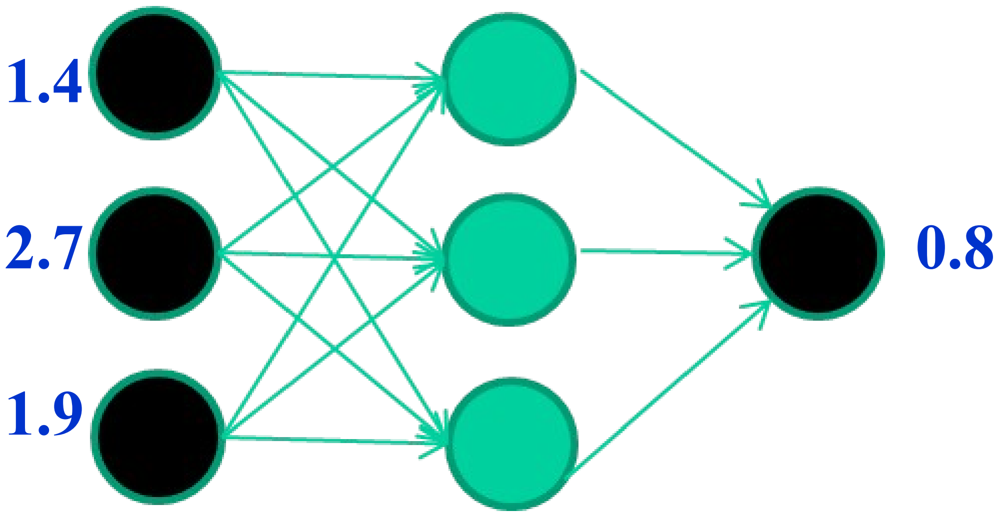


Training data

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

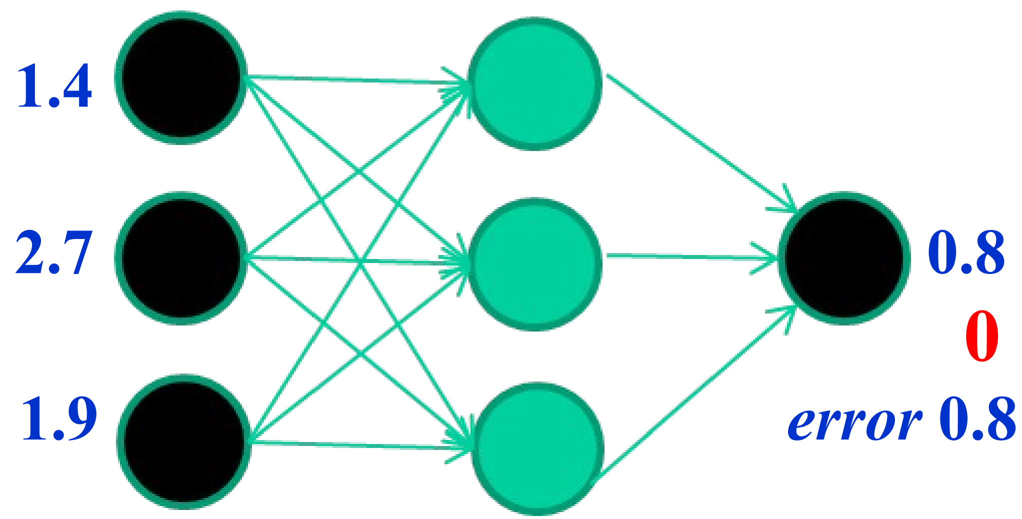


Training data

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

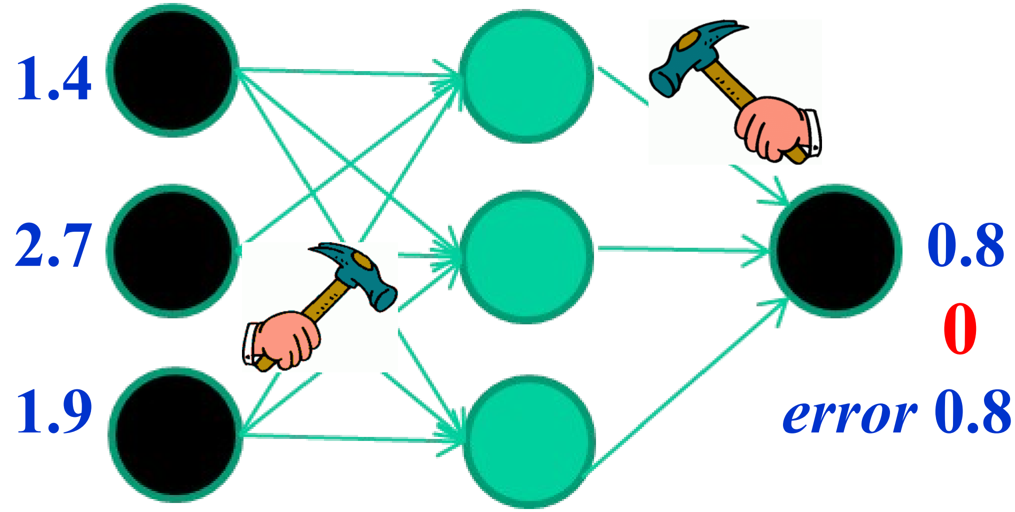


Training data

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 (gradient descent)

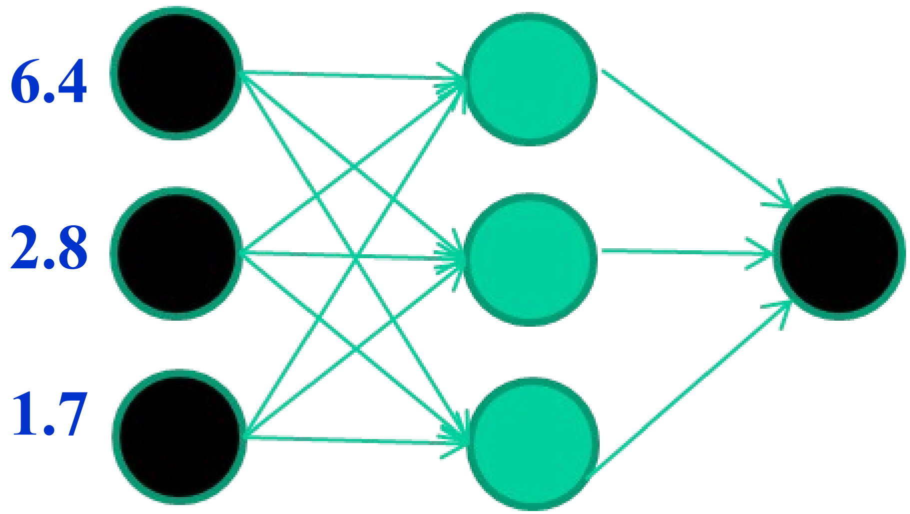


Training data

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

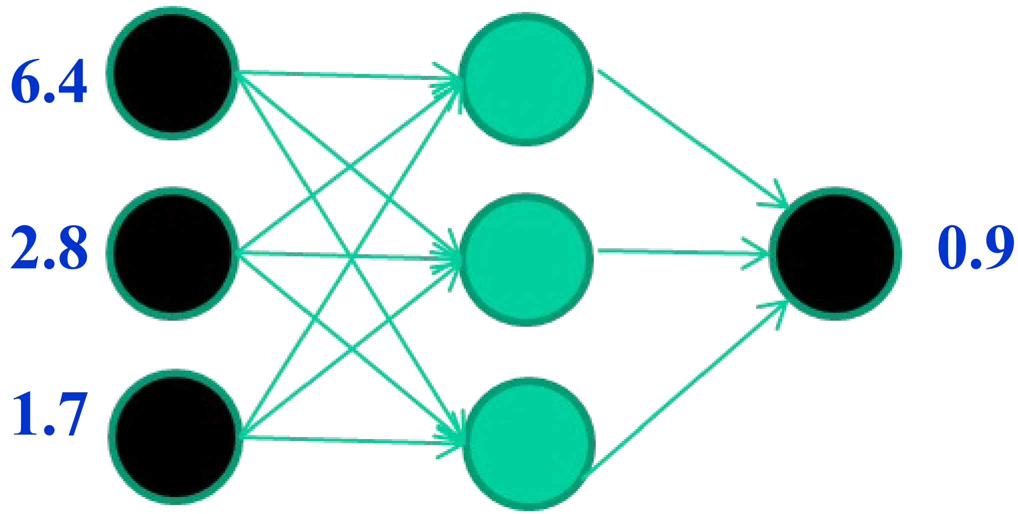


Training data

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

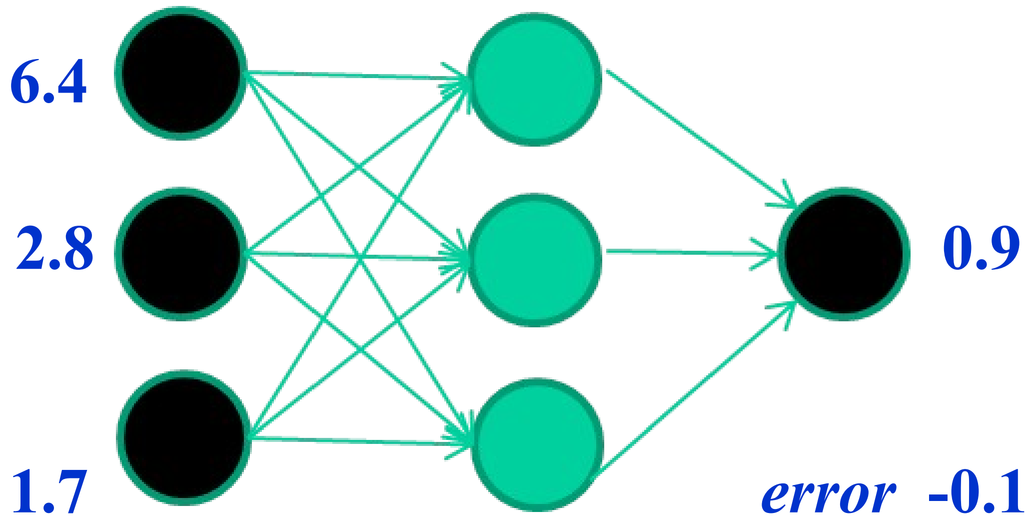


Training data

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

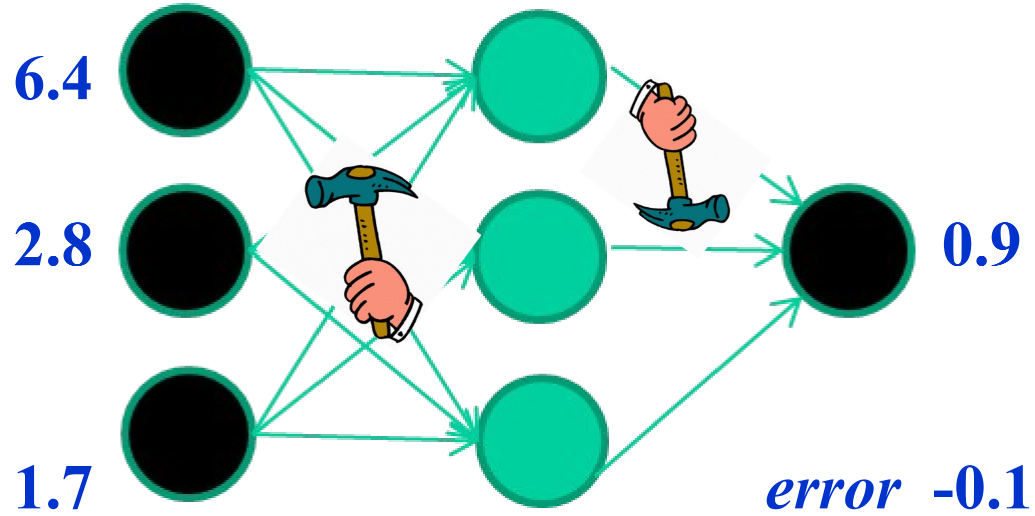


Training data

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

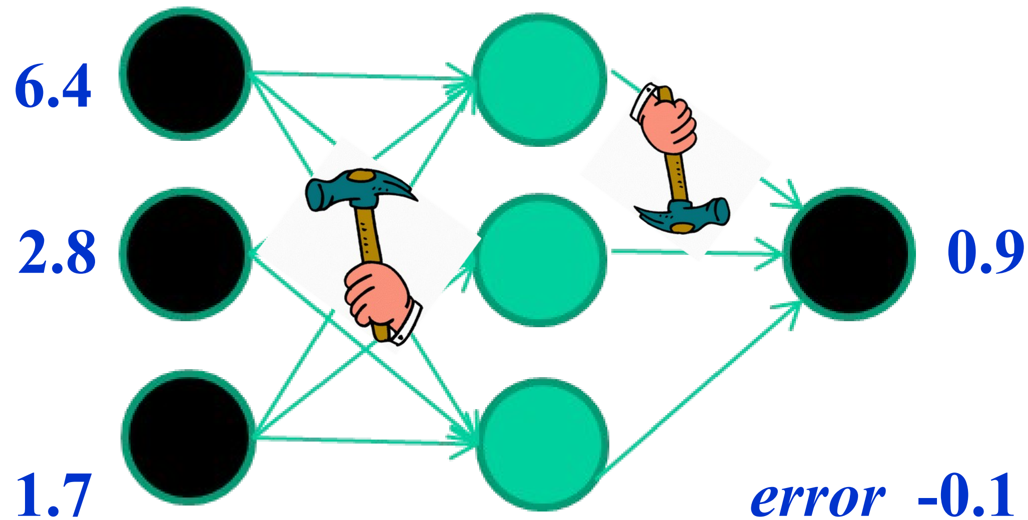


Training data

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 …

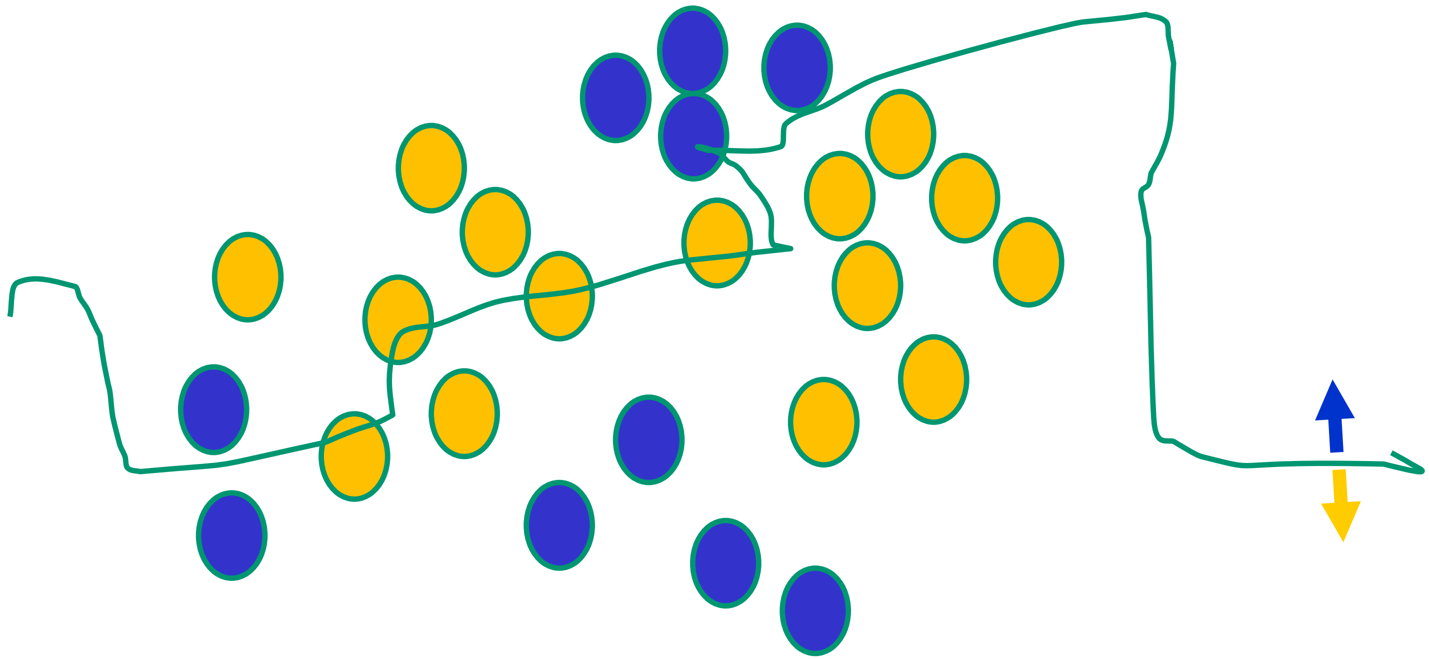
And so on ….



Repeat this THOUSANDS, MAYBE MILLIONS OF TIMES – each time taking a RANDOM training record, and making slight weight adjustments

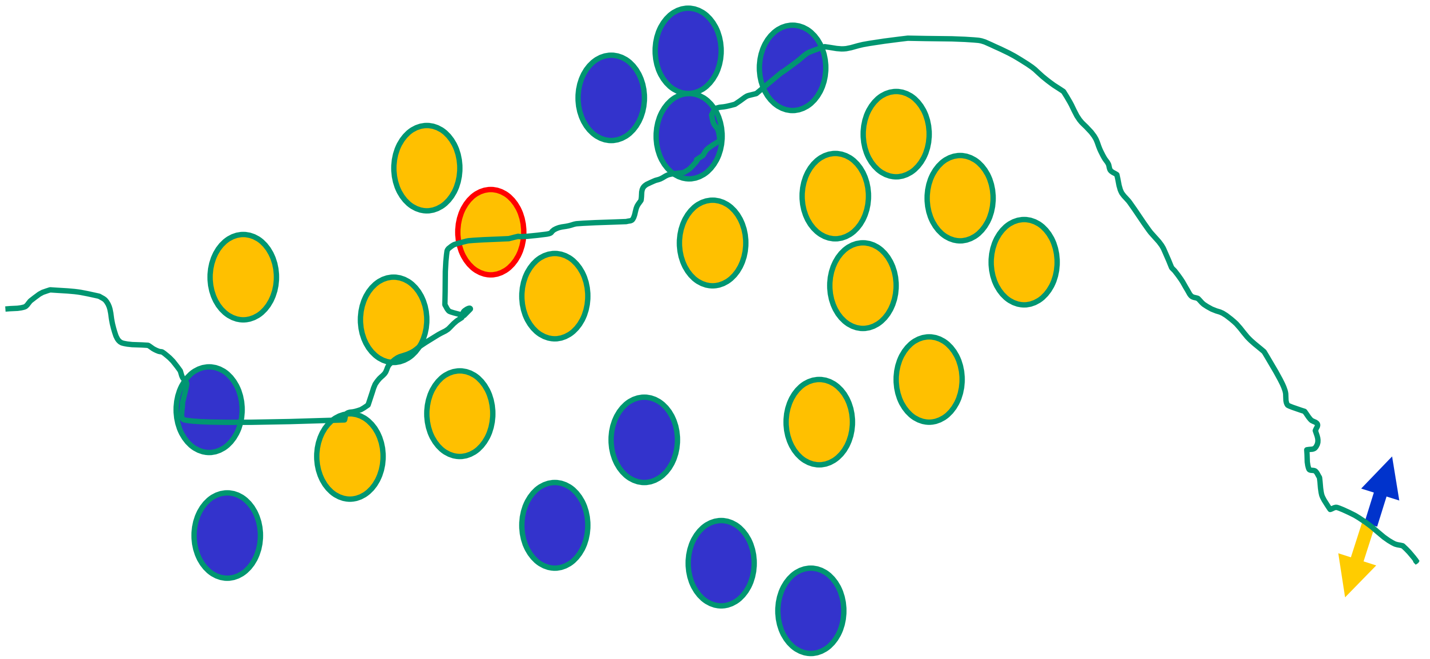
The decision boundary perspective…

Initial random weights



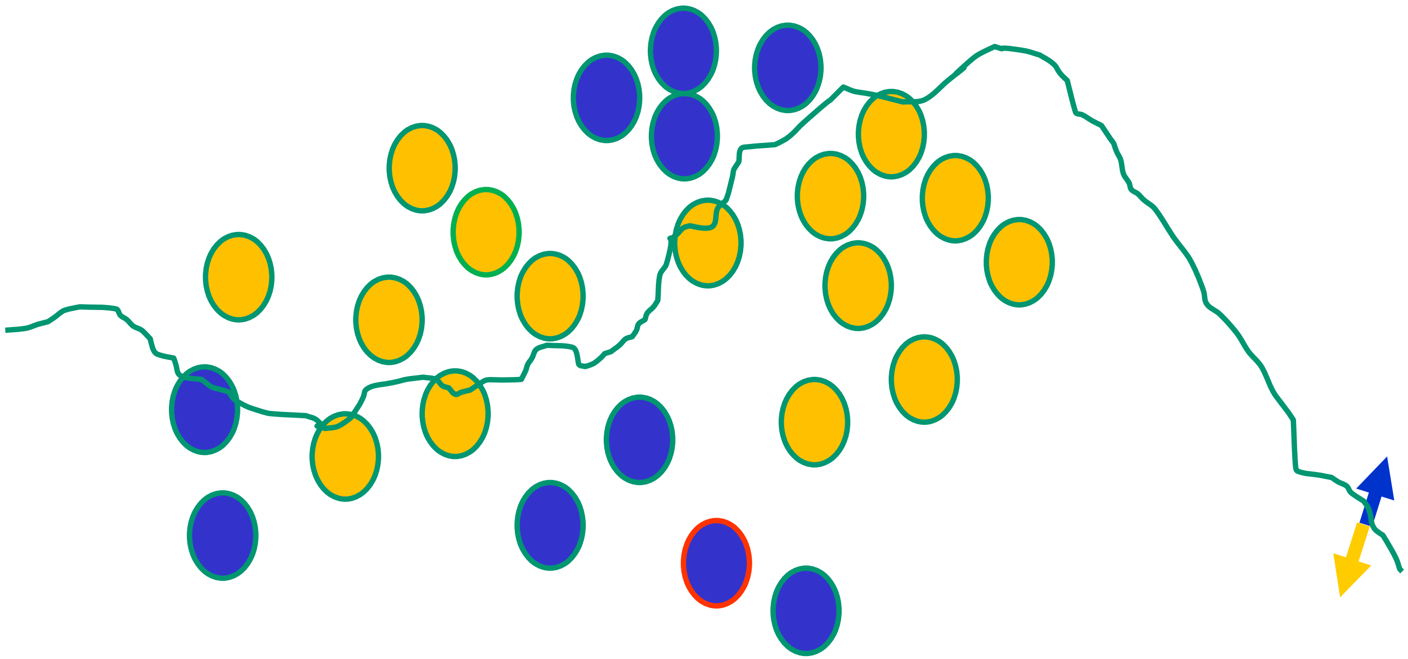
The decision boundary perspective…

Present a training instance / adjust the weights



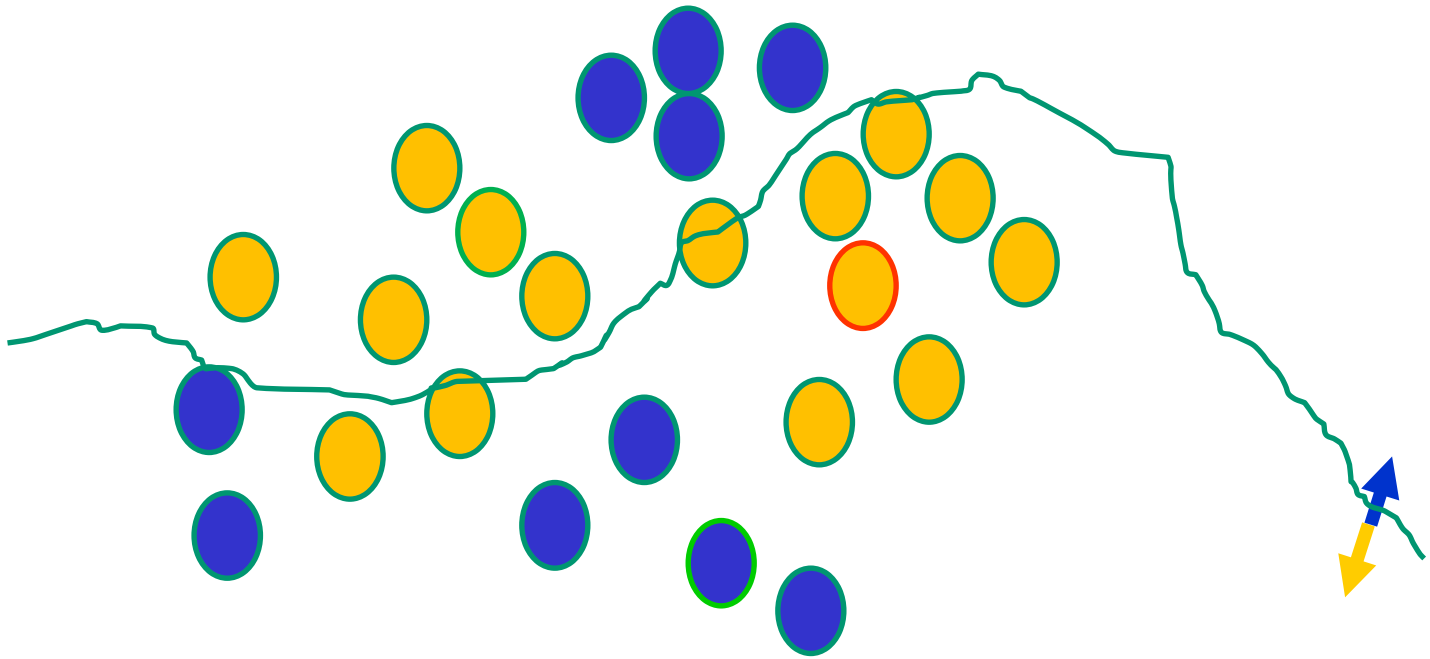
The decision boundary perspective…

Present a training instance / adjust the weights



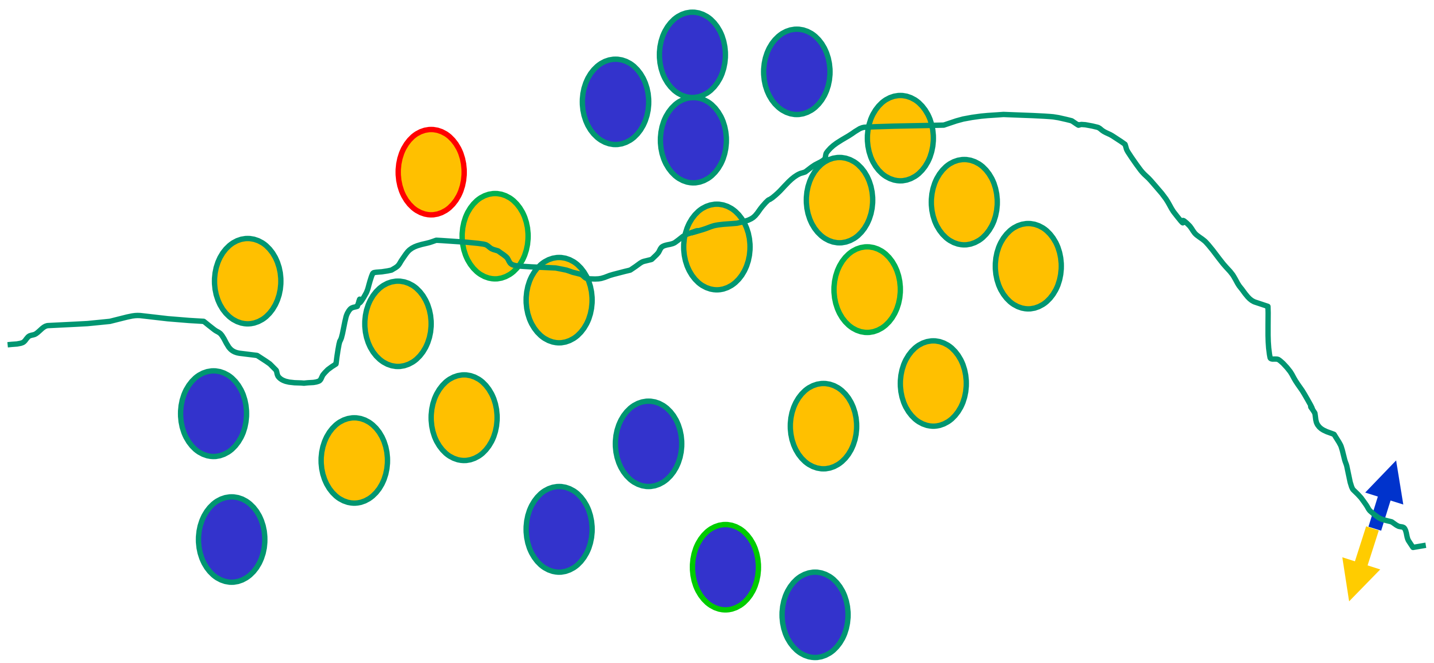
The decision boundary perspective…

Present a training instance / adjust the weights



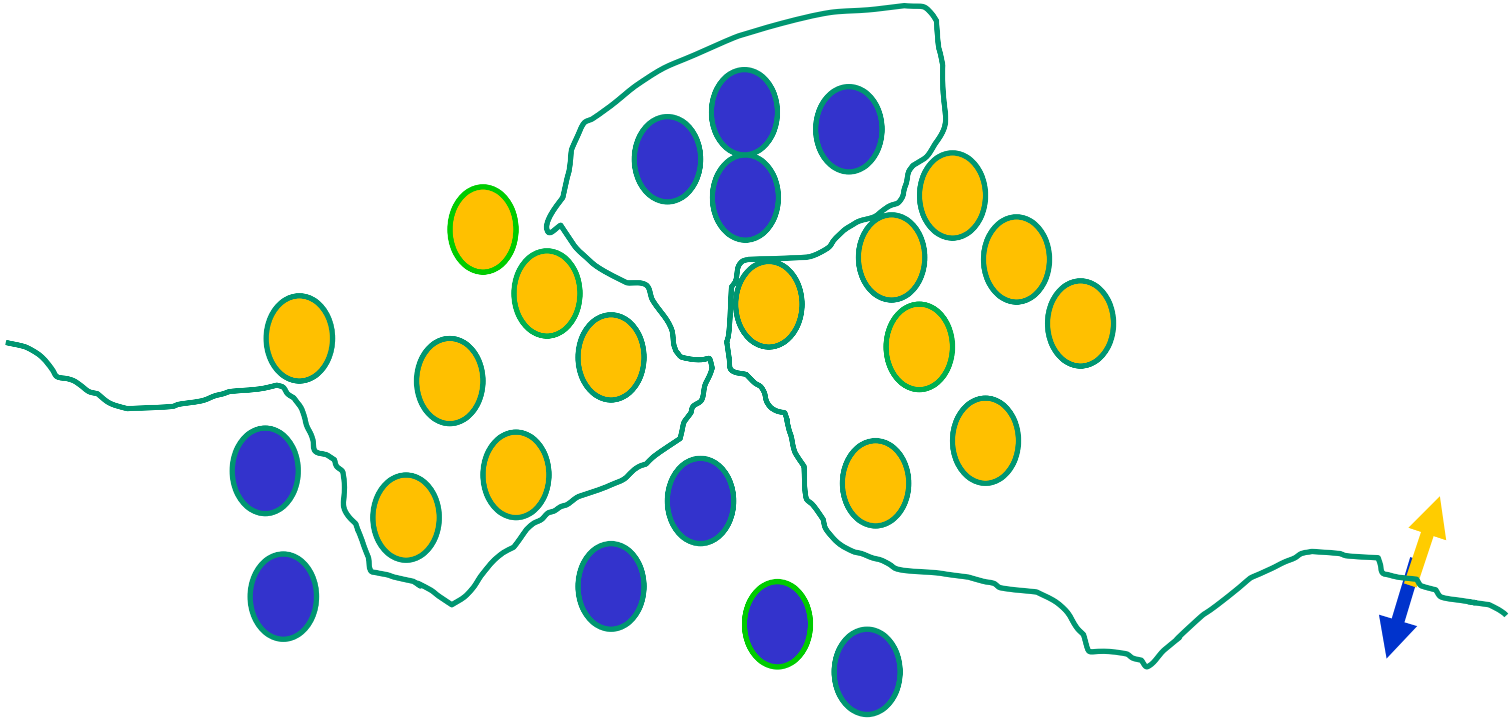
The decision boundary perspective…

Present a training instance / adjust the weights

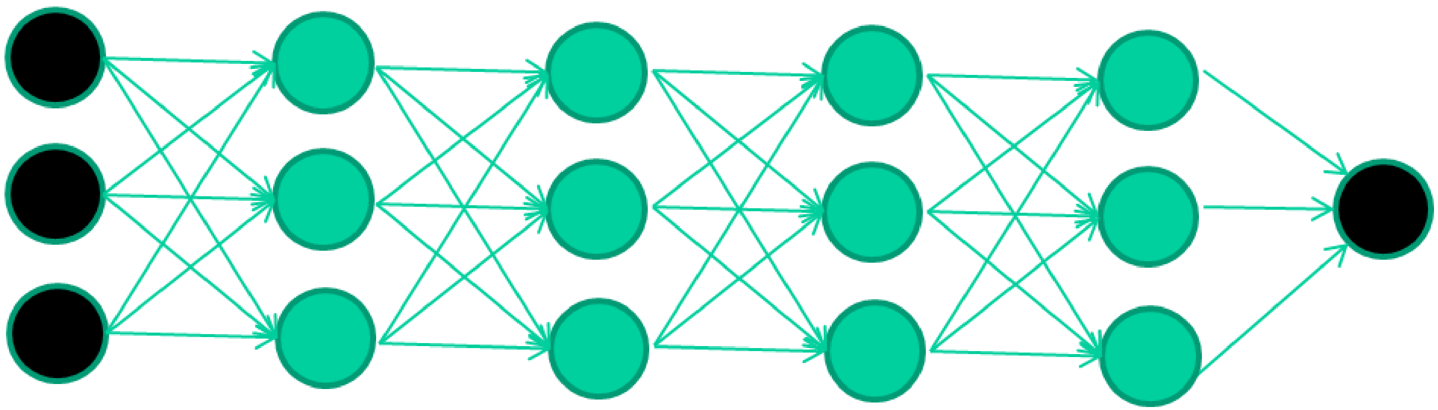


The decision boundary perspective…

Eventually ….

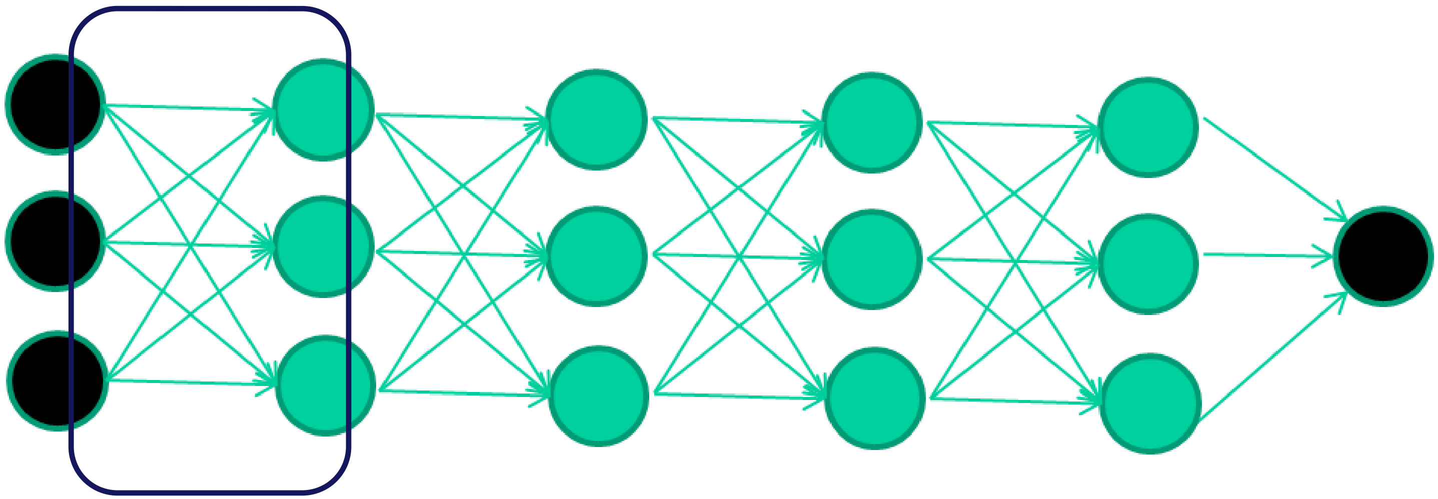


In the past, we rarely use a very deep network



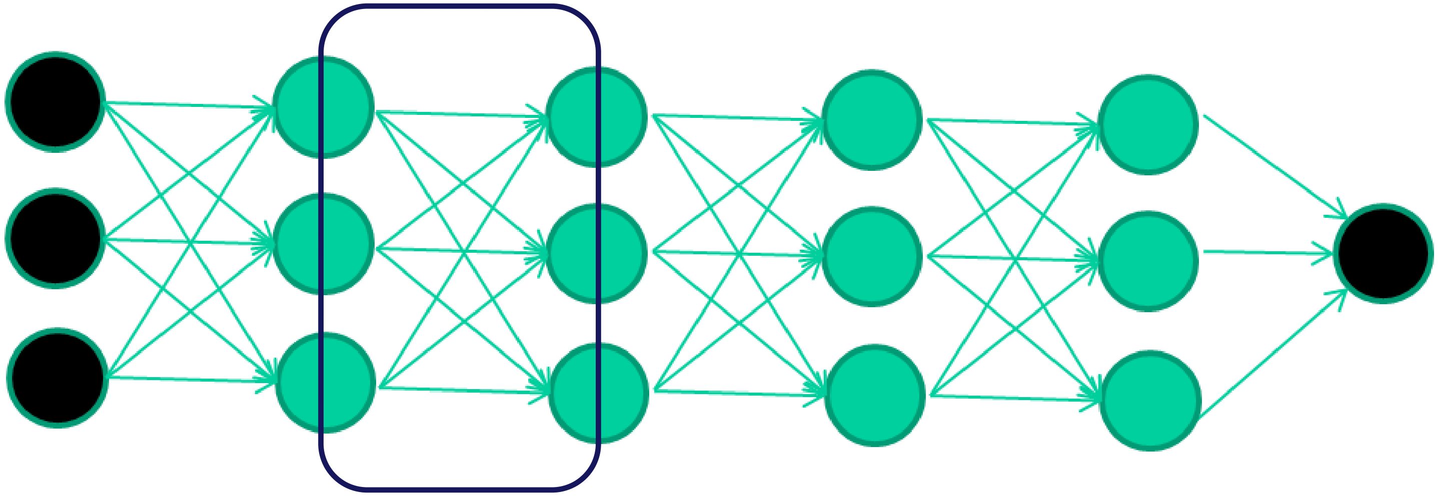
Now we have a new way to train a very deep neural networks…

The new way to train multi-layer NNs…



Train this layer first

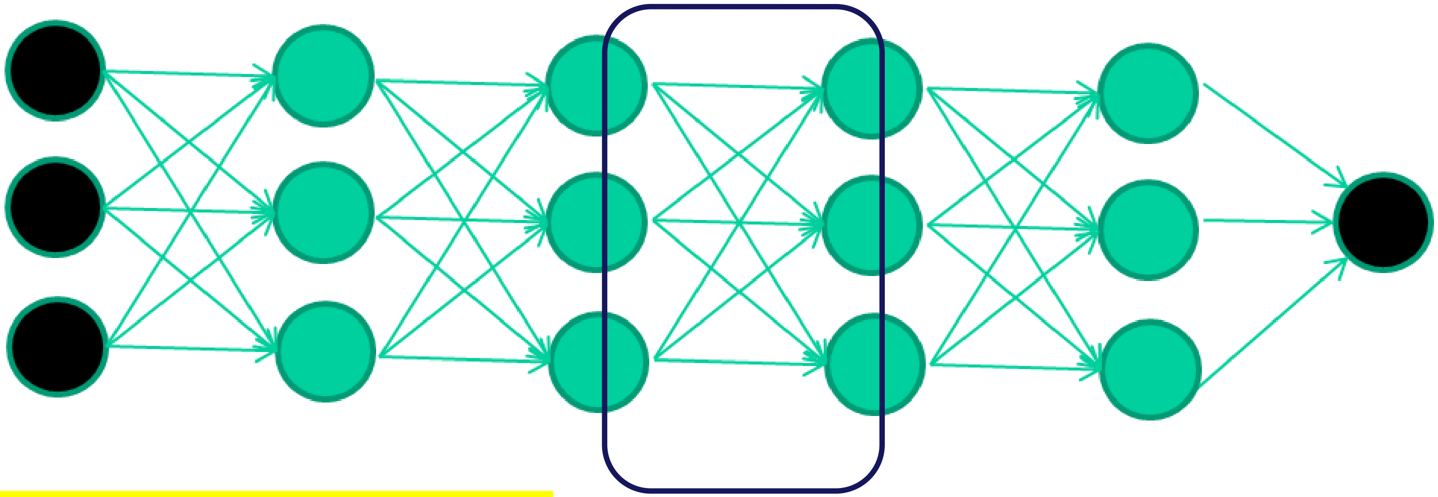
The new way to train multi-layer NNs…



Train this layer first

then this layer

The new way to train multi-layer NNs…

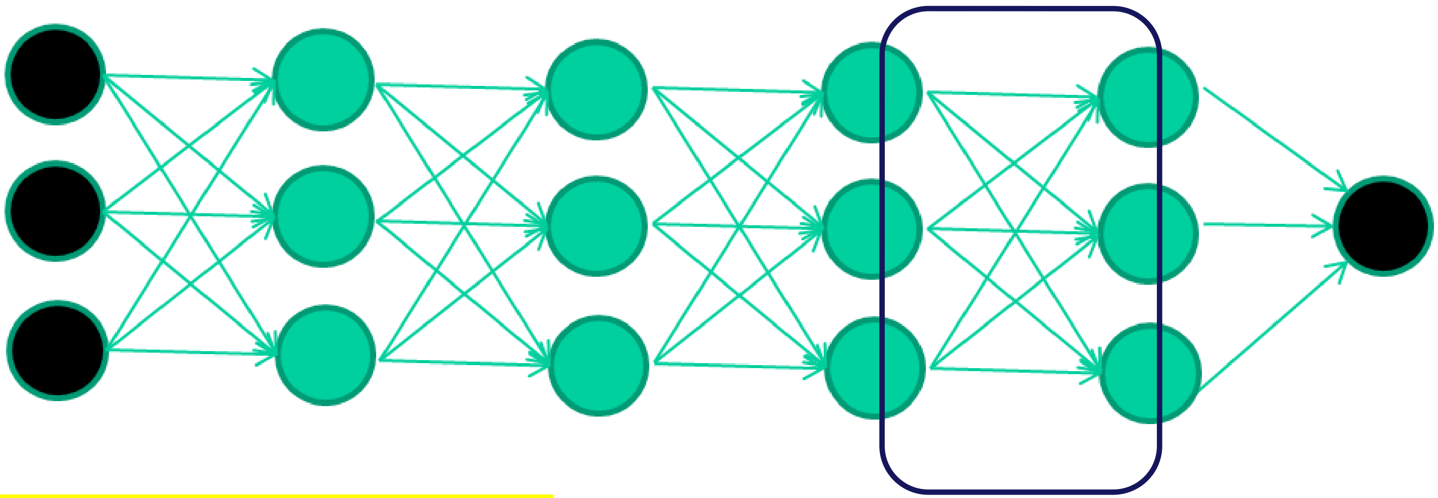


Train this layer first

then this layer

then this layer

The new way to train multi-layer NNs…



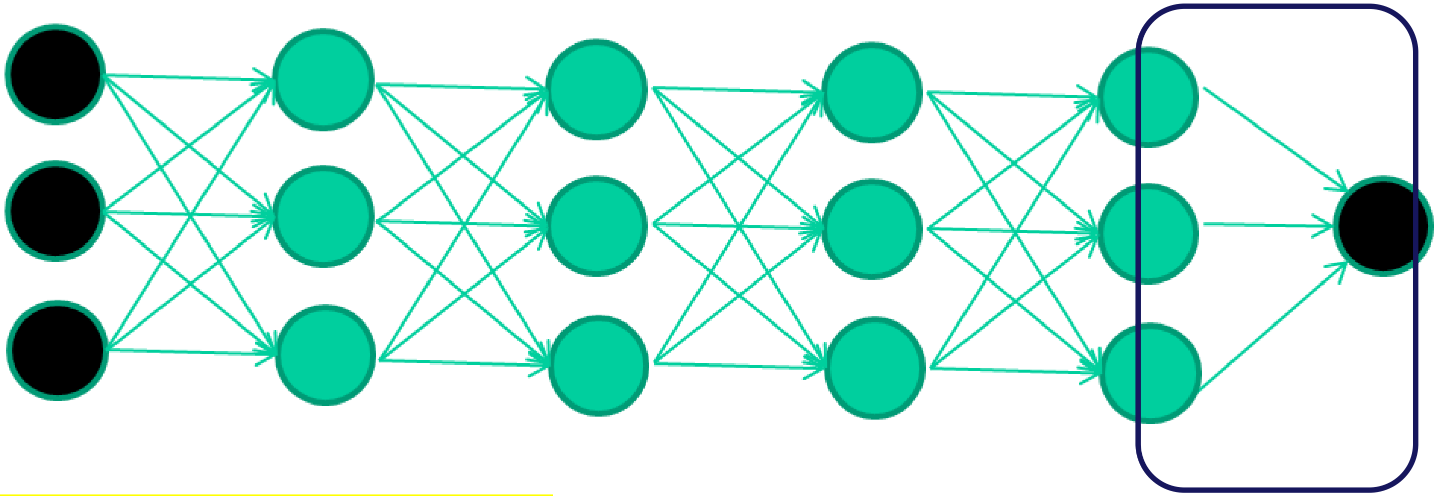
Train this layer first

then this layer

then this layer

then this layer

The new way to train multi-layer NNs…



Train this layer first

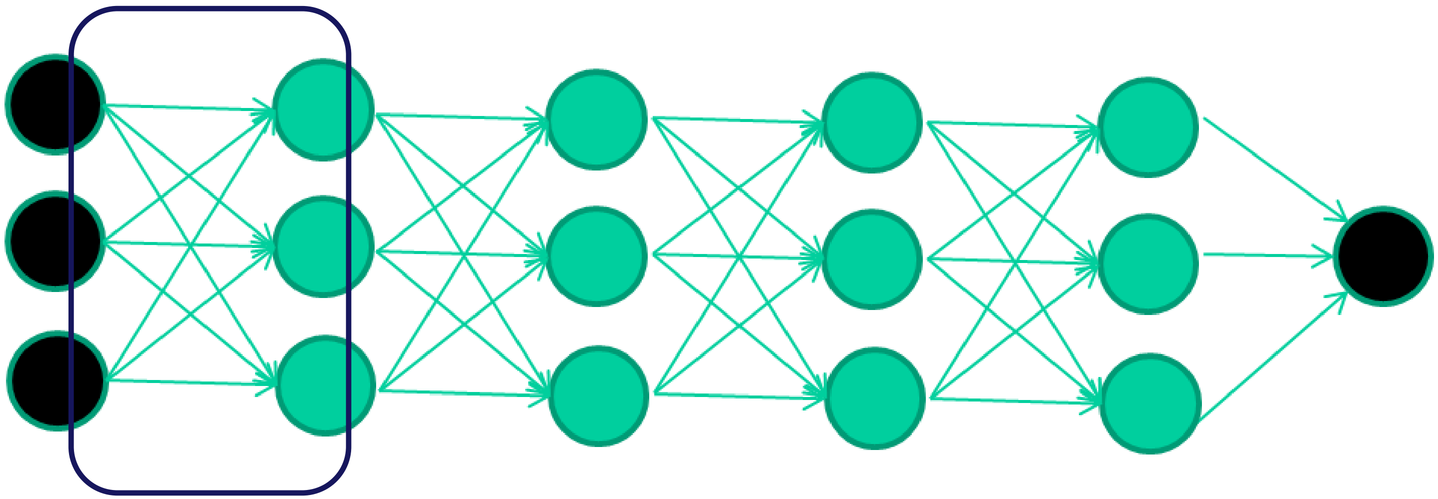
then this layer

then this layer

then this layer

finally this layer

The new way to train multi-layer NNs…

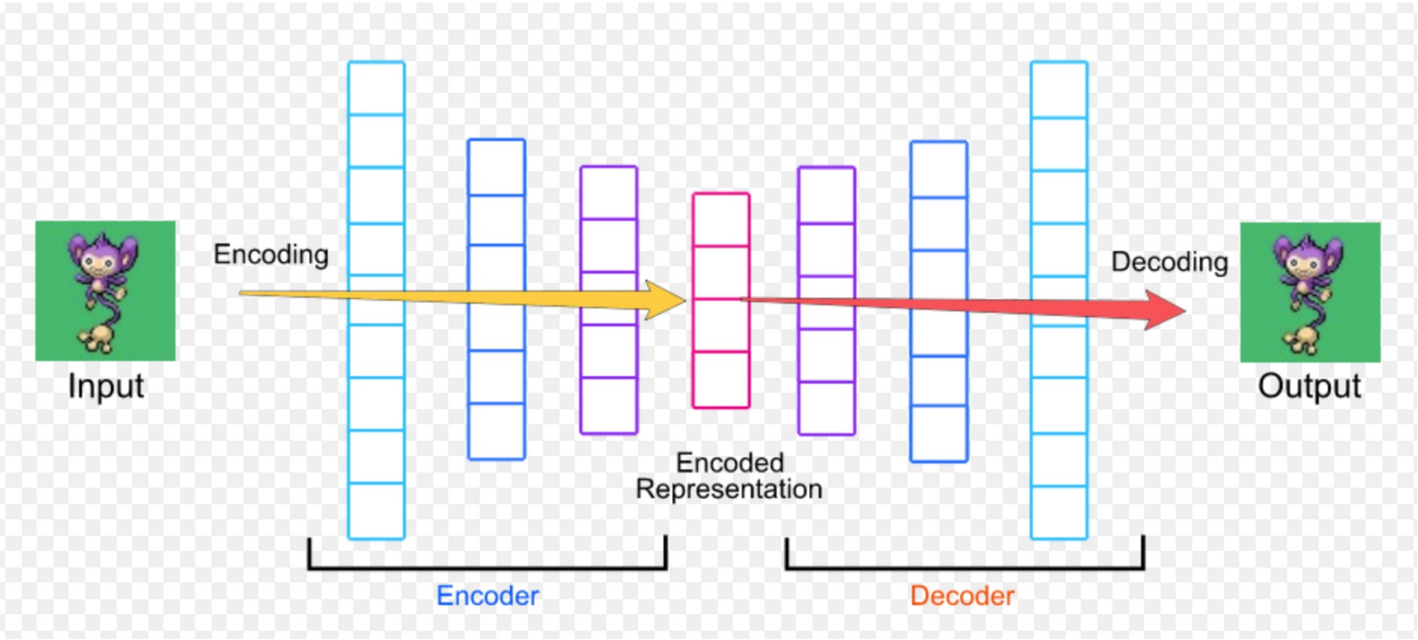


EACH of non-output layers is trained to be an

auto-encoder

Auto-encoder

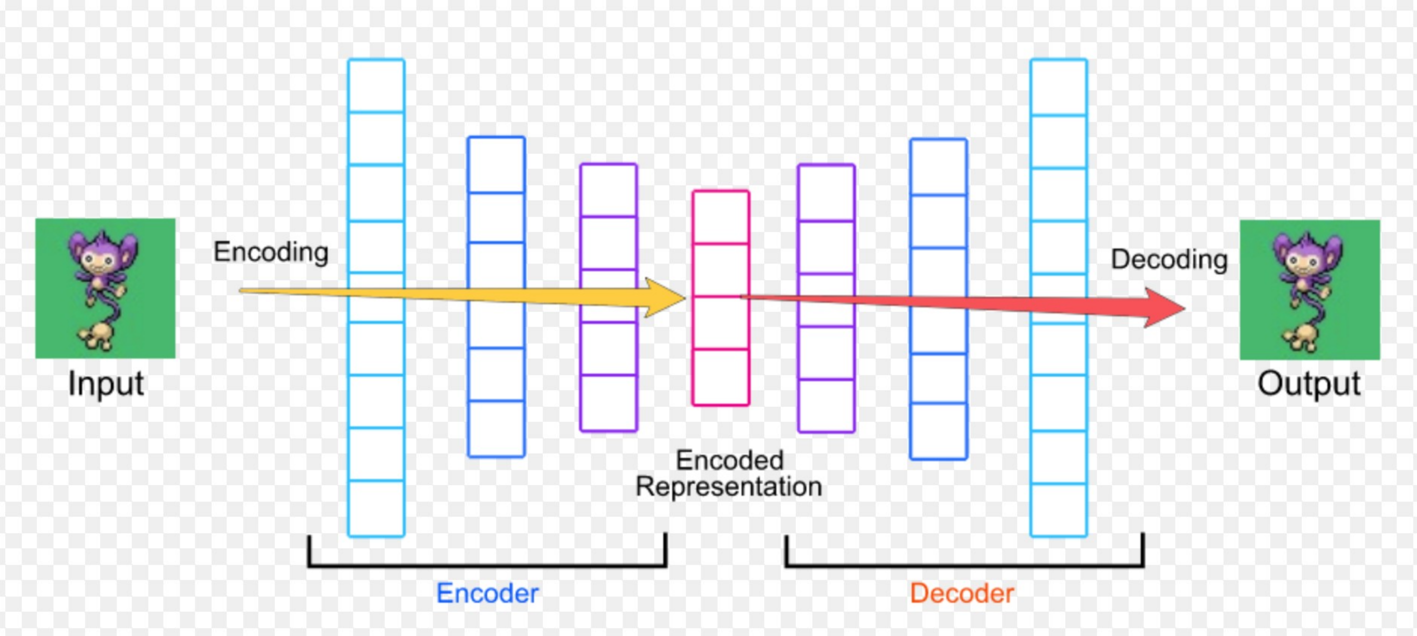
Auto-encoder *is a neural network* with the output layer having the same number of neurons as the input layer.



Auto-encoder

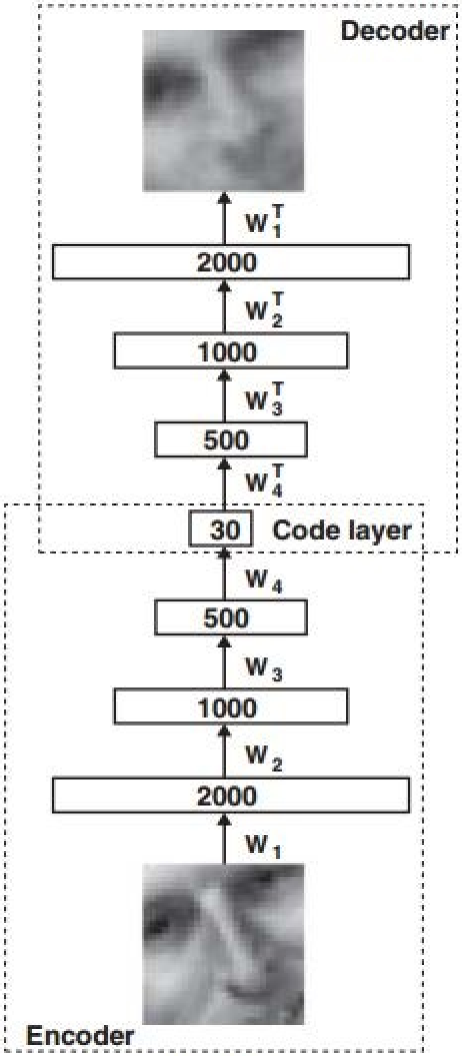
The purpose of using ***Auto-encoder*** is to reconstruct (recover) its input (X).

We want the output to be as close to input as possible



Why Auto-encoder can learn "true" features?

- The 'middle hidden layer' has fewer neurons than the inputs.
- Compress input data into a **short code** in such a way that when we uncompress that code, we can get input recovered.
- The output from the 'middle hidden layer' are "true" features (embedding of the input).
- This process forces 'middle hidden layer' **to become good feature detectors** for input
- This process can also be referred to **dimensionality reduction**.



Popular deep learning architectures

- Convolutional Neural Networks
  - Scenarios where **spatially-closer data are more correlated**
- Recurrent Neural Networks
  - Data with **temporal or sequential structures**

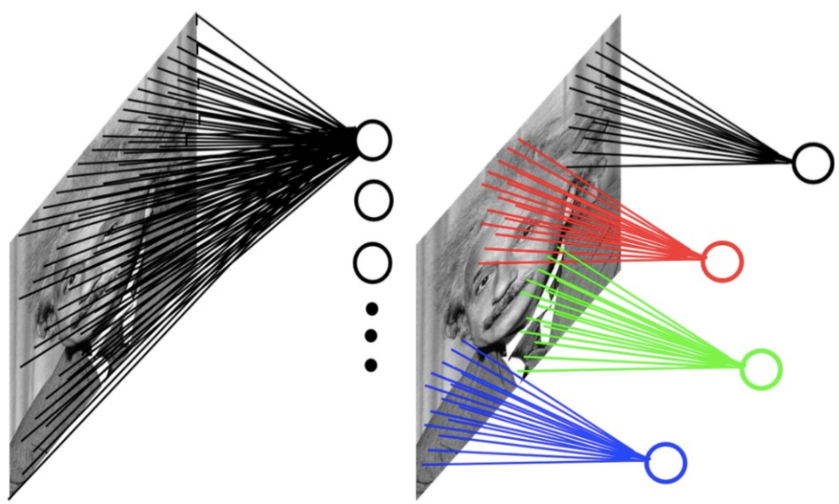
Popular deep learning architectures

- Convolutional Neural Networks
  - Scenarios where **spatially-closer data are more correlated**
- Recurrent Neural Networks
  - Data with **temporal or sequential structures**

Convolutional Neural Network (CNN)

In CNNs, hidden neurons are only connected to local receptive field.

- The number of weights (parameters) in CNNs is much smaller than a fully-connected neural network.
- CNN is widely used in computer vision.

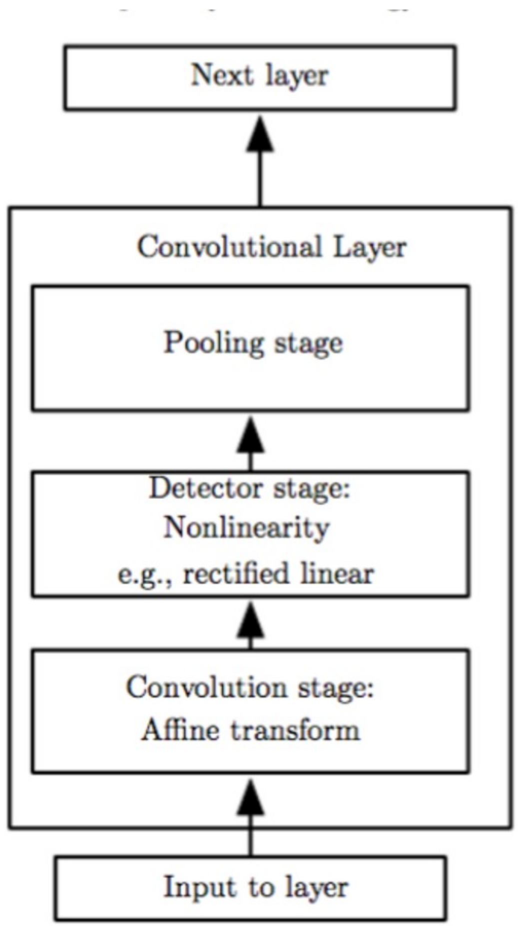


Example: 200x200 image

a) **A fully connected layer: 40,000 input units X 40,000 hidden units => 1.6 billion connections (parameters)**

b) **A CNN layer: 5x5 kernel, 100 kernel => 2,600 parameters**

Three Stages of A Convolutional Operation



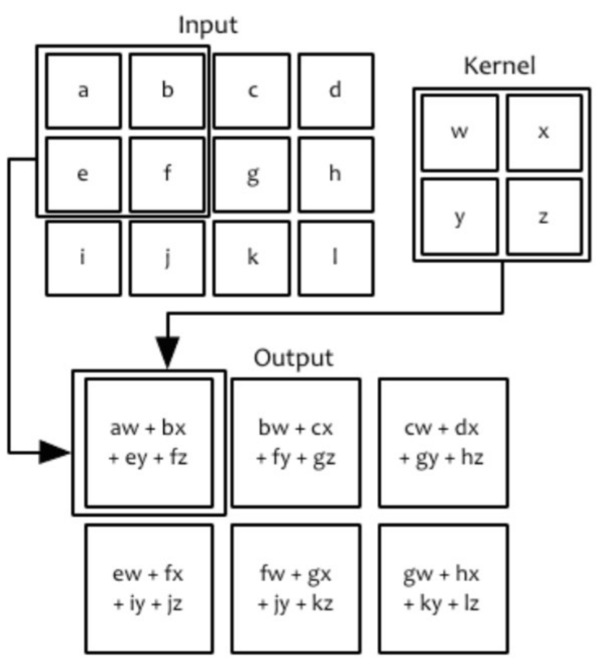
1. **Convolution**
2. **Nonlinear transformation**
3. **Pooling**

Convolution stage

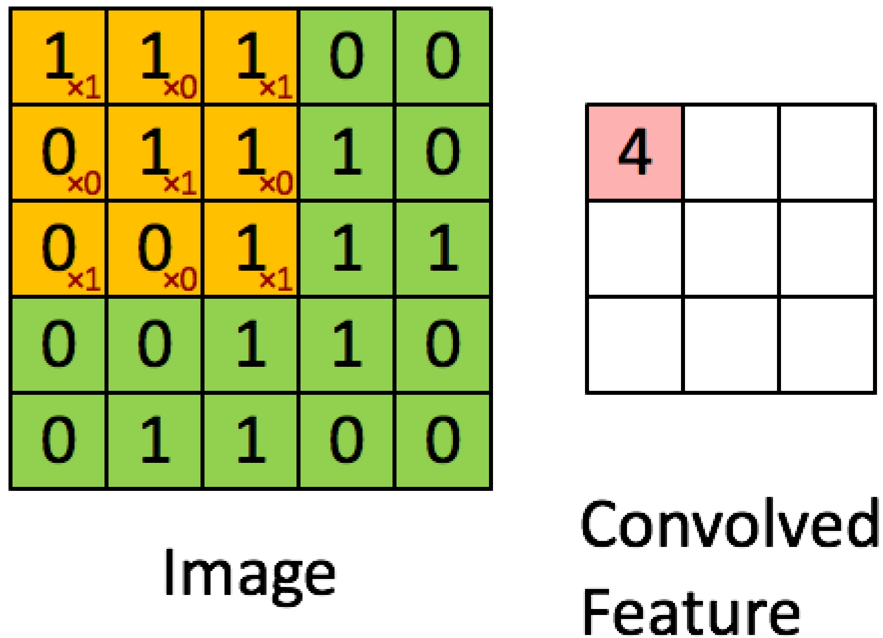
- Input: an image x (2-D array) x
- Convolution kernel (2-D array of learnable parameters): w
- Output: called feature map (2-D array of processed data): s

Convolution operation:

$$s[i,j] = (x * w)[i,j] = \sum_{m=-M}^{M} \sum_{n=-N}^{N} x[i+m, j+n]\, w[m,n]$$

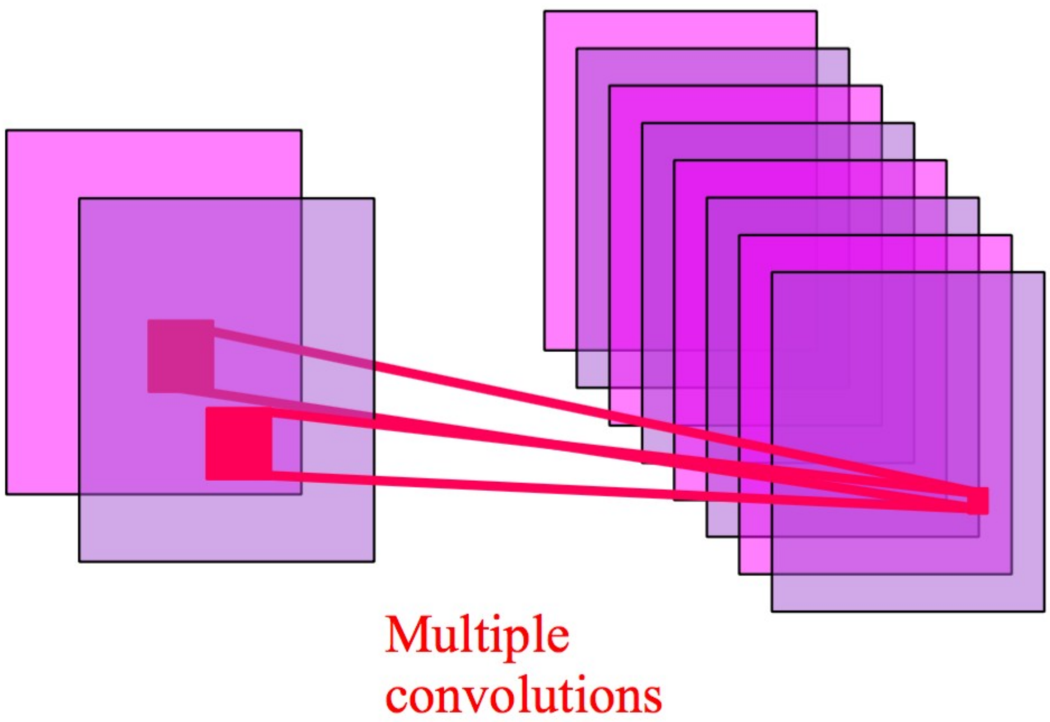


Single Convolution Kernel



Multiple Convolution Kernels

Multiple convolution kernels yield multiple feature maps, one for each kernel.



Multiple Convolution Kernels

In TesorFlow, you should provide the following parameters to define each convolutional layer:

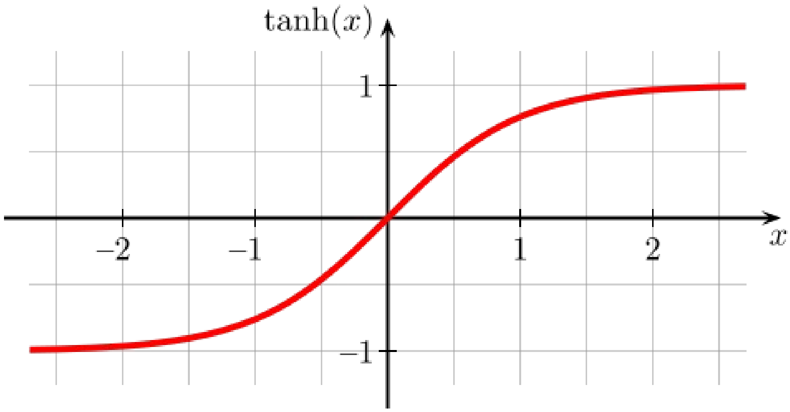
- Number of filters/kernels
- Kernel size (vertical, horizontal): the height and width of convolution window
- Strides (vertical, horizontal): default strides=(1, 1)

Demo of multiple kernels

http://setosa.io/ev/image-kernels/

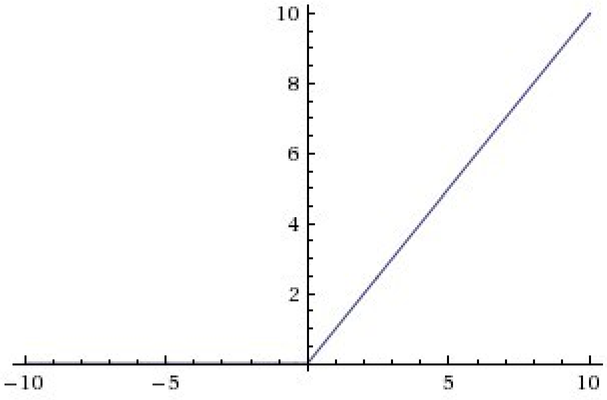
Nonlinear transformation

Tanh(x)



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

ReLU

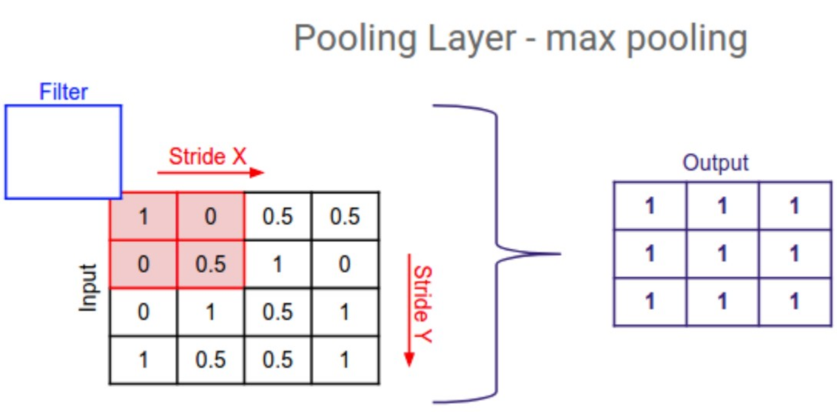


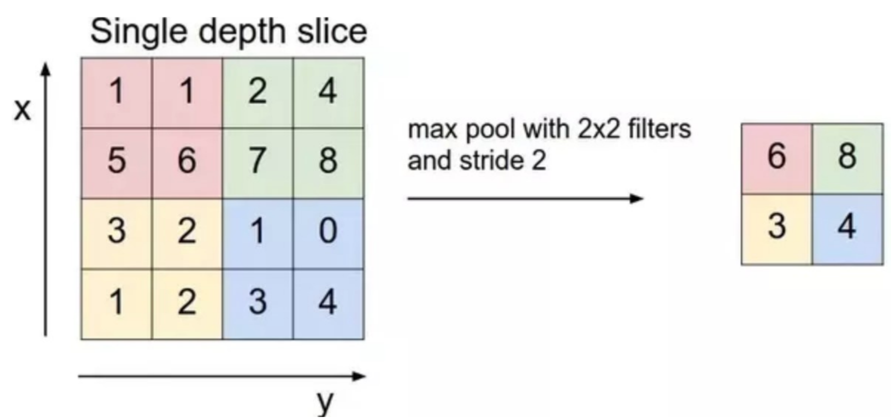
$$f(x) = max(0, x)$$

Pooling (downsampling strategy)

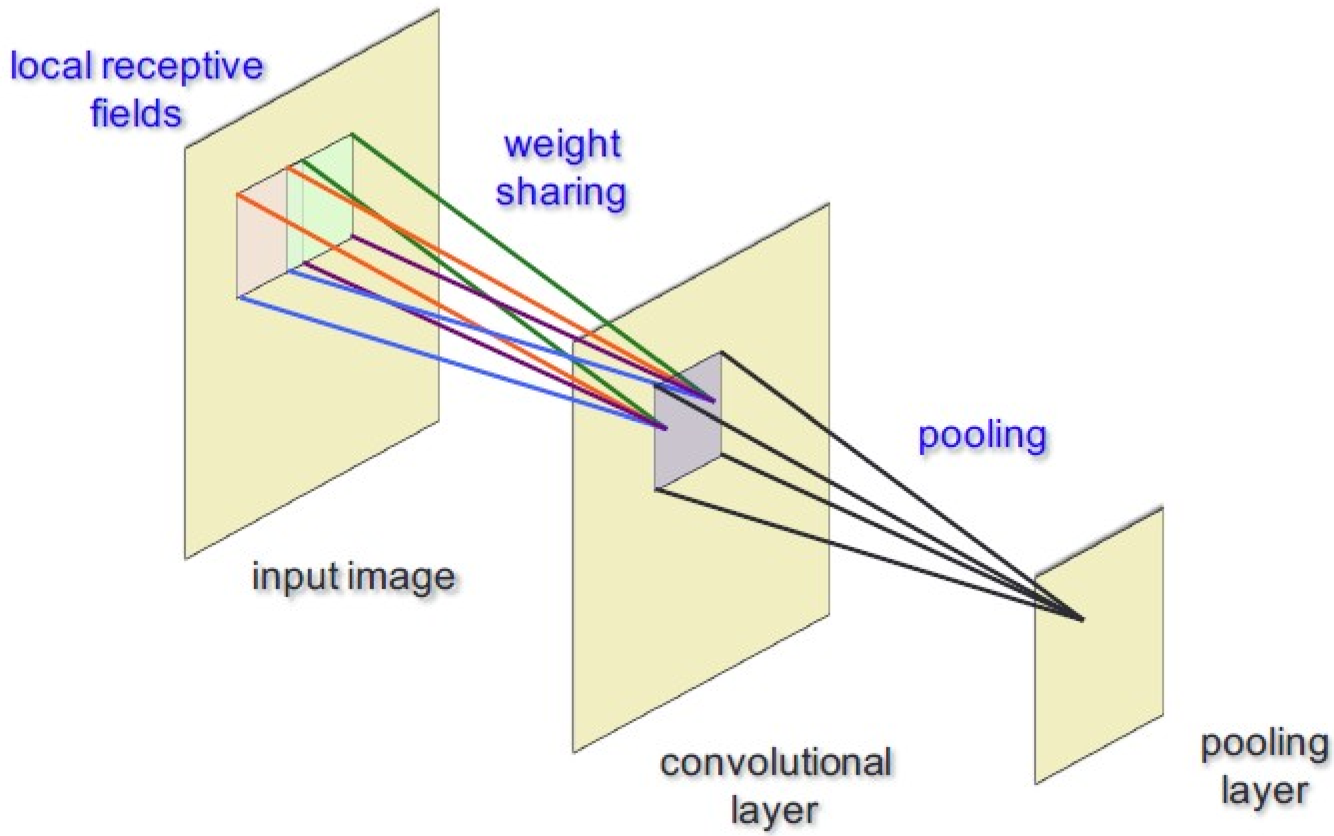
Common pooling operations:

- Max pooling: reports the maximum output within a rectangular neighborhood.
- Average pooling: reports the average output of a rectangular neighborhood.
- Pool size (vertical, horizontal)
- Stride value (vertical, horizontal)
- Padding: "valid" or "same"





Each CNN operation (Conv + Nonlinear + Pooling)



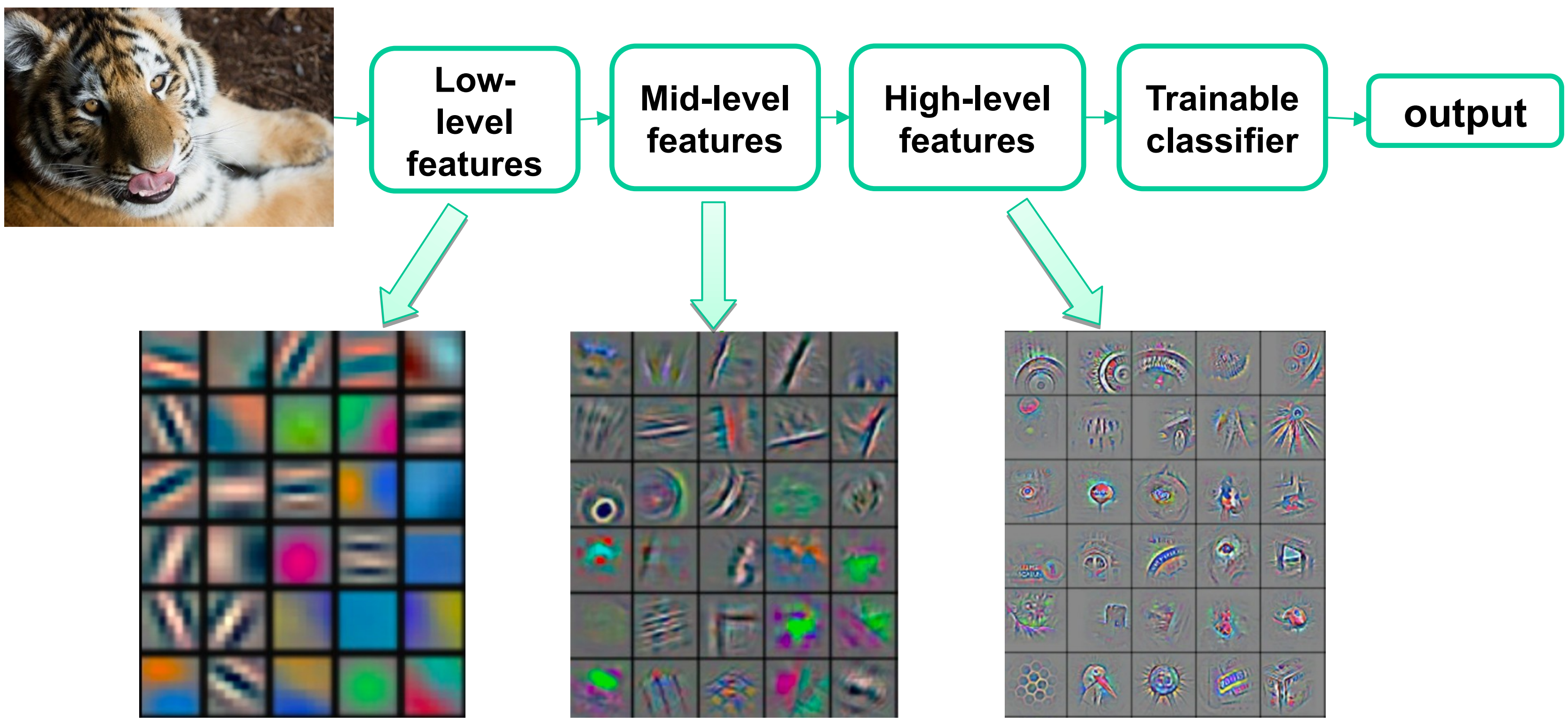
Video on CNN

Note on CNN

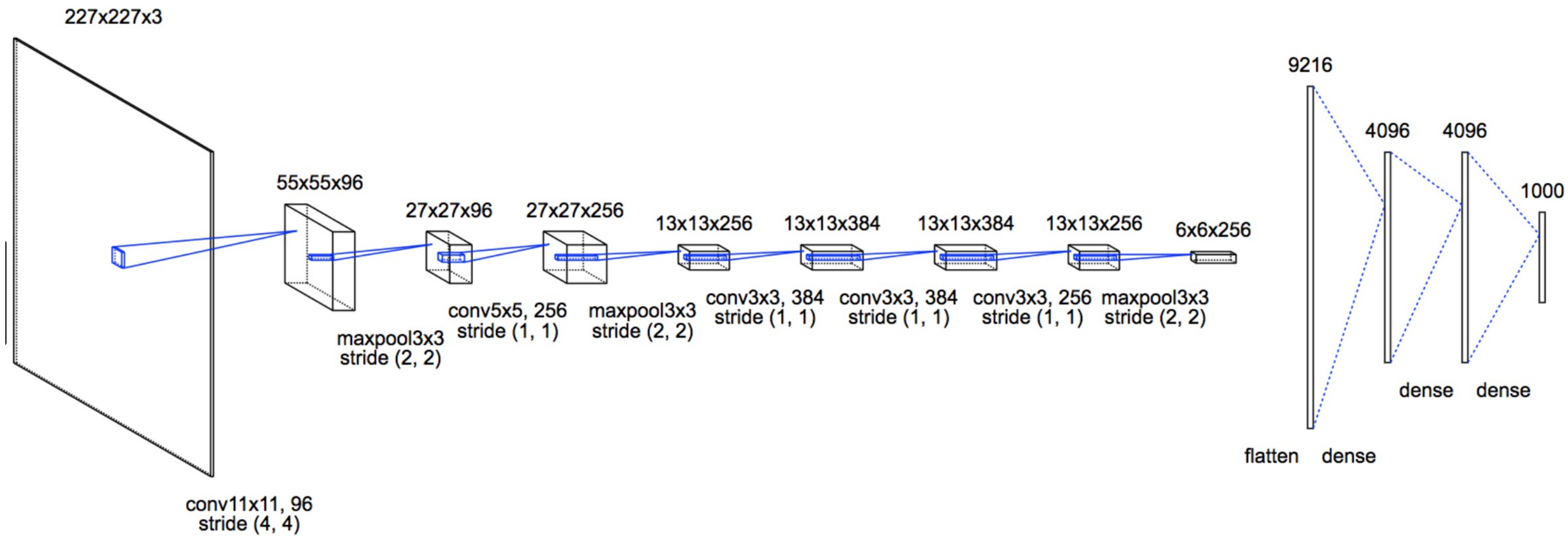
1. You may use multiple convolution kernels (hidden neurons), which give you multiple feature maps.
2. You may stack multiple convolutional layers for extracting features at different levels.
3. Higher-level layers take the feature maps from lower-level layers as input.

Deep CNN

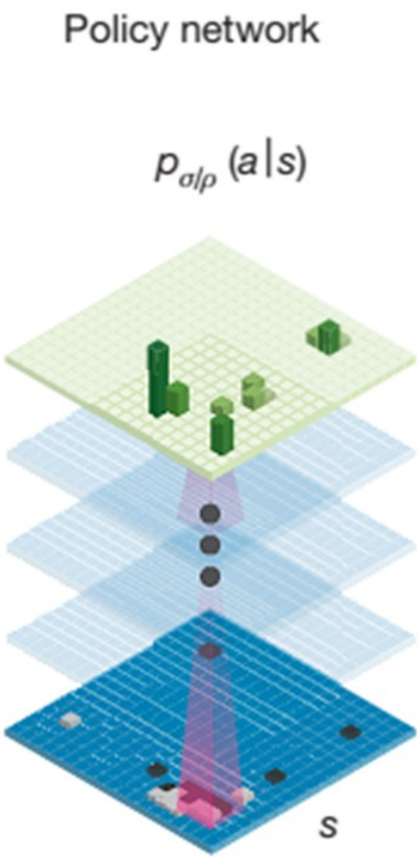
CNN extracts features automatically through multiple layers.



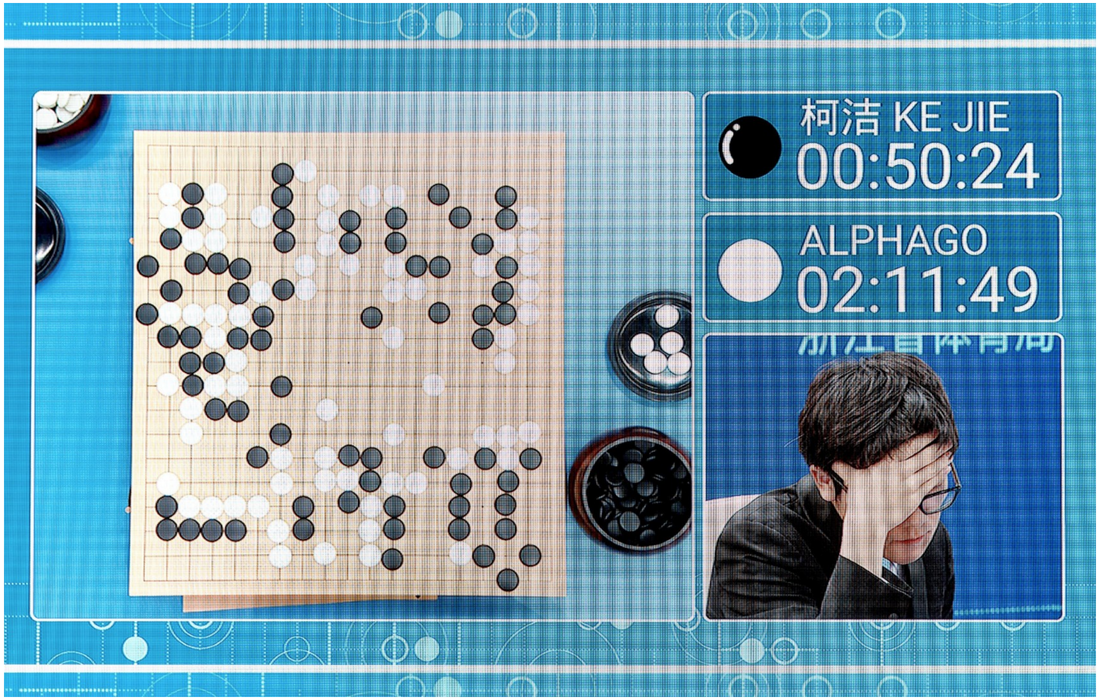
CNN for RGB Image Classification



CNN in Alpha-Go



- A CNN with 13 layers
- Input board position as image
- Output: , where is the next move



CNN for Image Classification

https://cs.stanford.edu/people/karpathy/convnetjs/

CNN talks

NVIDIA’s CES 2015 talk about their self-driving cars

YOLO