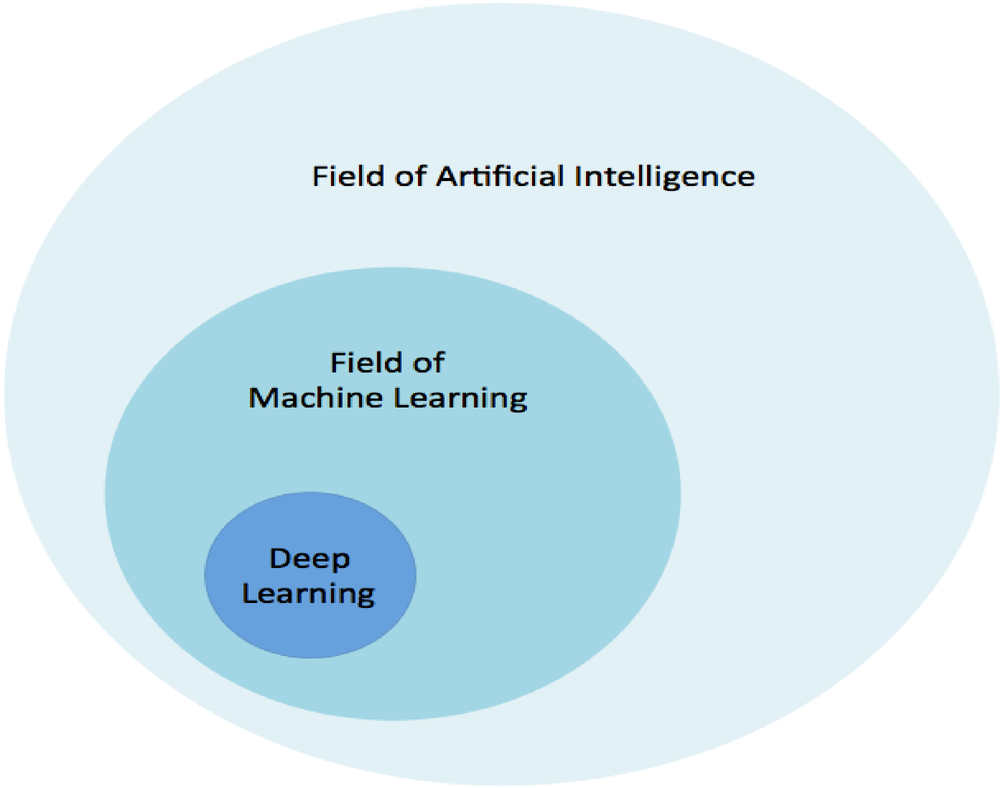
Microsoft

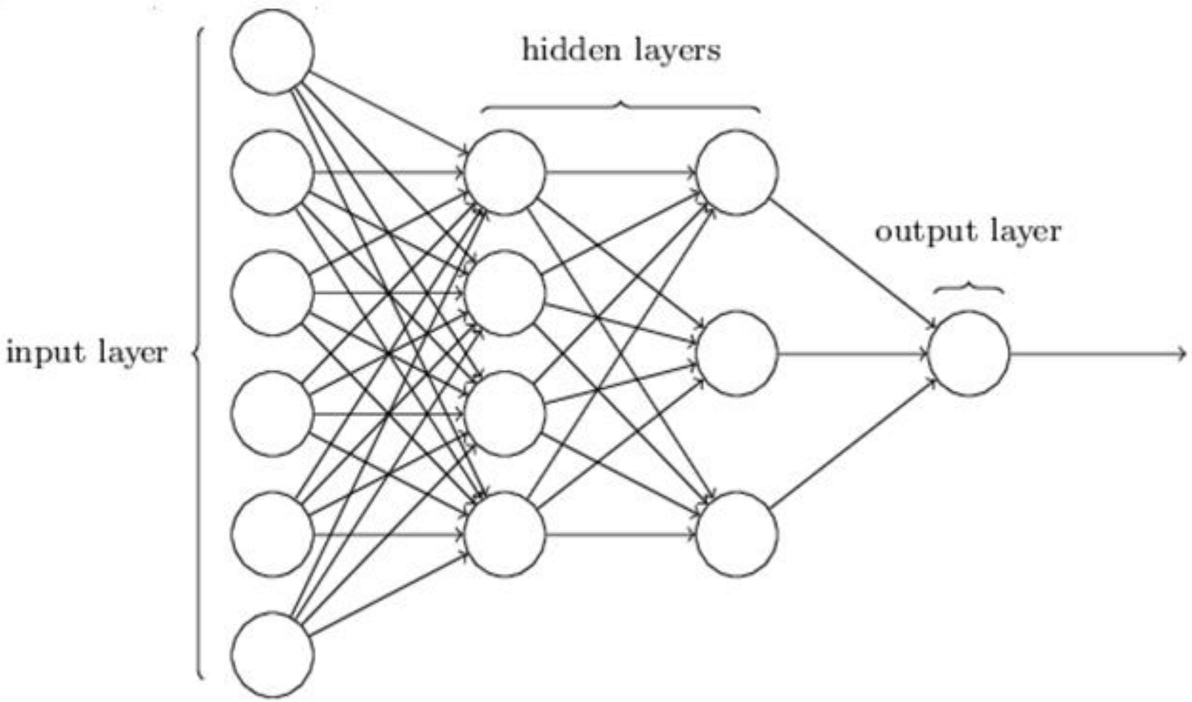
Deep Learning

Mithun Prasad, PhD
miprasad@Microsoft.com

What Is Deep Learning?

1. Based on Algorithms that attempt to model high level abstractions in data
2. Deep learning is synonymous with artificial neural network (ANN)
3. The “deep” in deep learning refers to the depth of the network. An ANN can be very shallow





Microsoft

Why Is Deep Learning Popular?

- DL models has been here for a long time
  - Fukushima (1980) – Neo-Cognitron
  - LeCun (1989) – Convolutional Neural Network

- DL popularity grew recently
  - With growth of Big Data
  - With the advent of powerful GPUs

Motivation: Why Go Deep

- Deep Architectures can be representationally efficient
  Fewer computational units for same function

- Deep Representations might allow for a hierarchy or representation
  Allows non-local generalization
  Comprehensibility

- Multiple levels of latent variables allow combinatorial sharing of statistical strength

- Deep architectures work well (vision, audio, NLP, etc.) !

Different Levels Of Abstraction

Hierarchical Learning

- Natural progression from low level to high level structure as seen in natural complexity

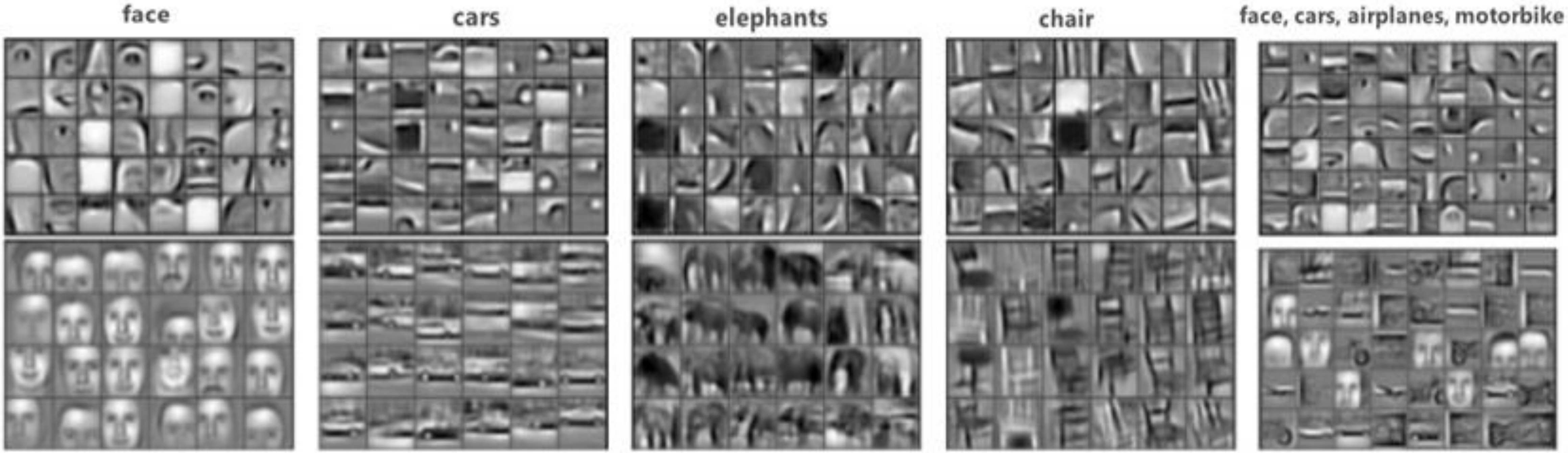
- Easier to monitor what is being learnt and to guide the machine to better subspaces

- A good lower level representation can be used for many distinct tasks

Compositional Data

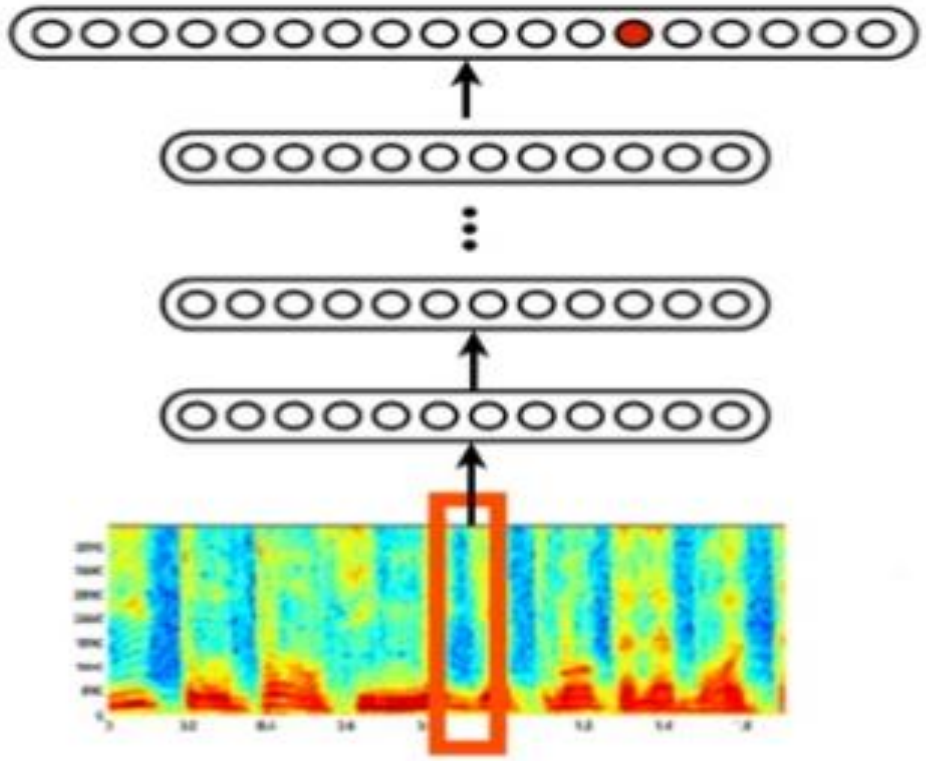
NATURAL DATA
IS COMPOSITIONAL.

Compositional Data



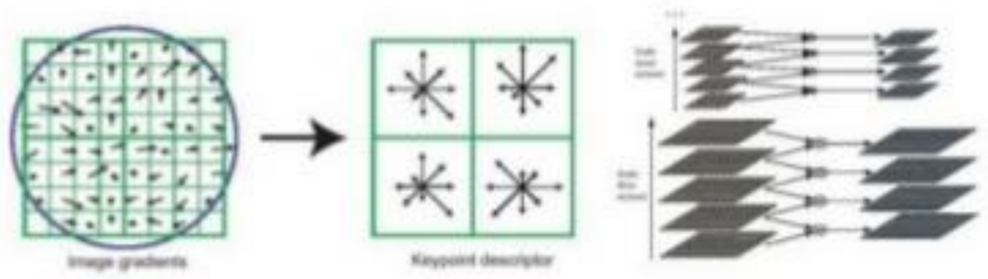
Compositional Data

Sound

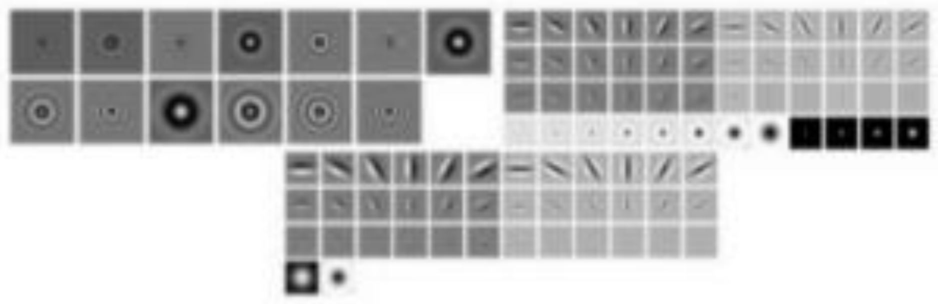


Microsoft

Traditional vs Deep Learning



SIFT



Textons

Traditional vs Deep Learning

Feature extractors, required:

- Expert knowledge
- Time-consuming hand-tuning
- In industrial applications, this 90% of the time
- Sometimes are problem specific

But, what if we could learn feature extractors ?

Traditional vs Deep Learning

Traditional ML requires manual feature extraction/engineering

Deep learning can automatically learn features in data

Feature extraction for unstructured data is very difficult

Deep learning is largely a "black box" technique, updating learned weights at each layer

Deep Learning Begins With A Little Function

It all starts with a humble linear function called a perceptron.

$$
\begin{array}{rl}
 & \text{weight1} \times \text{input1} \\
 & \text{weight2} \times \text{input2} \\
+ & \text{weight3} \times \text{input3} \\
\hline
 & \text{sum}
\end{array}
$$

Perceptron:
If sum > threshold: output 1
Else: output 0

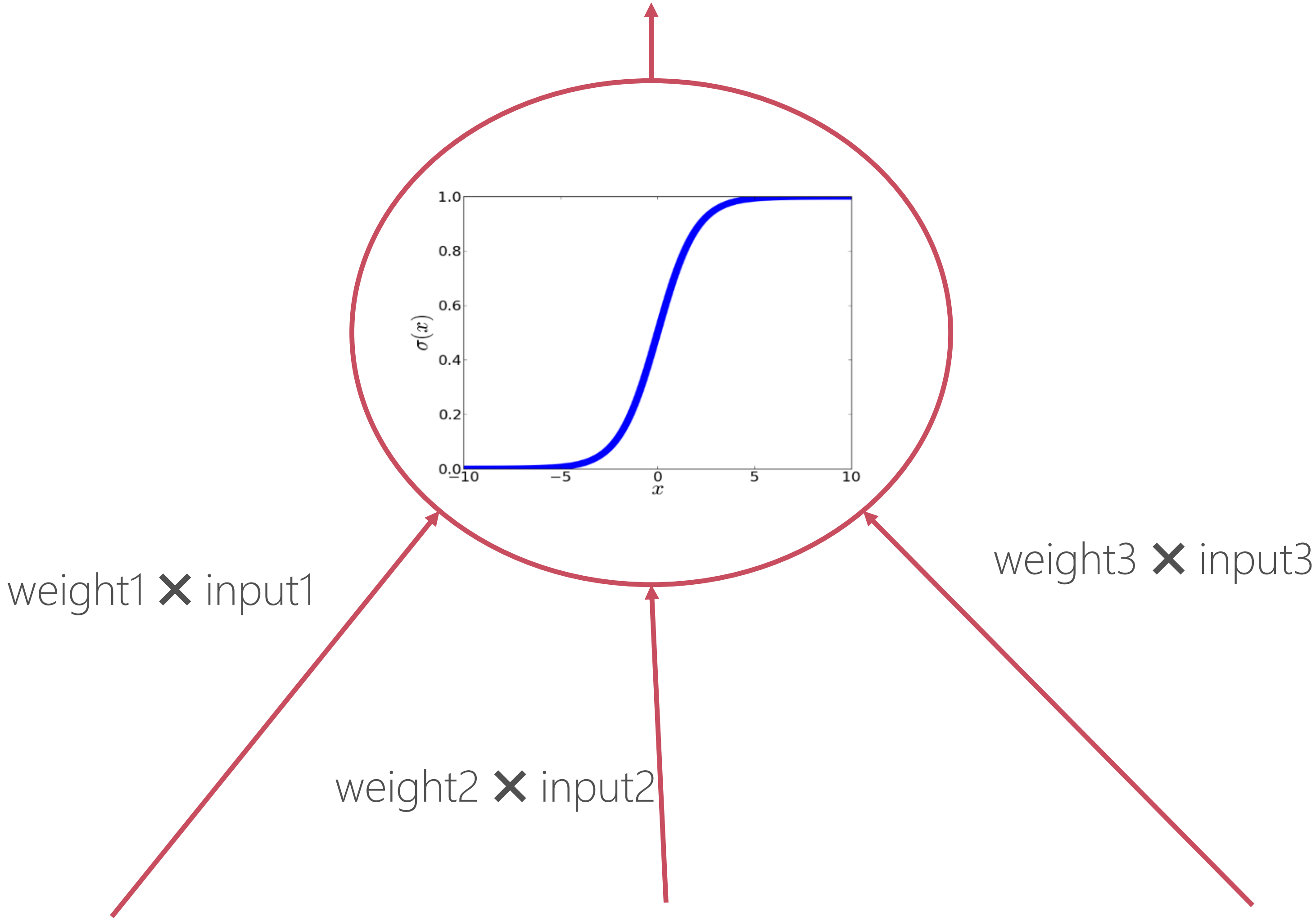
Example: The inputs can be your data. Question: Should I buy this car?

$$
\begin{array}{rl}
 & 0.2 \times \text{gas mileage} \\
 & 0.3 \times \text{horsepower} \\
+ & 0.5 \times \text{num cup holders} \\
\hline
 & \text{sum}
\end{array}
$$

Perceptron:
If sum < threshold: buy
Else: walk

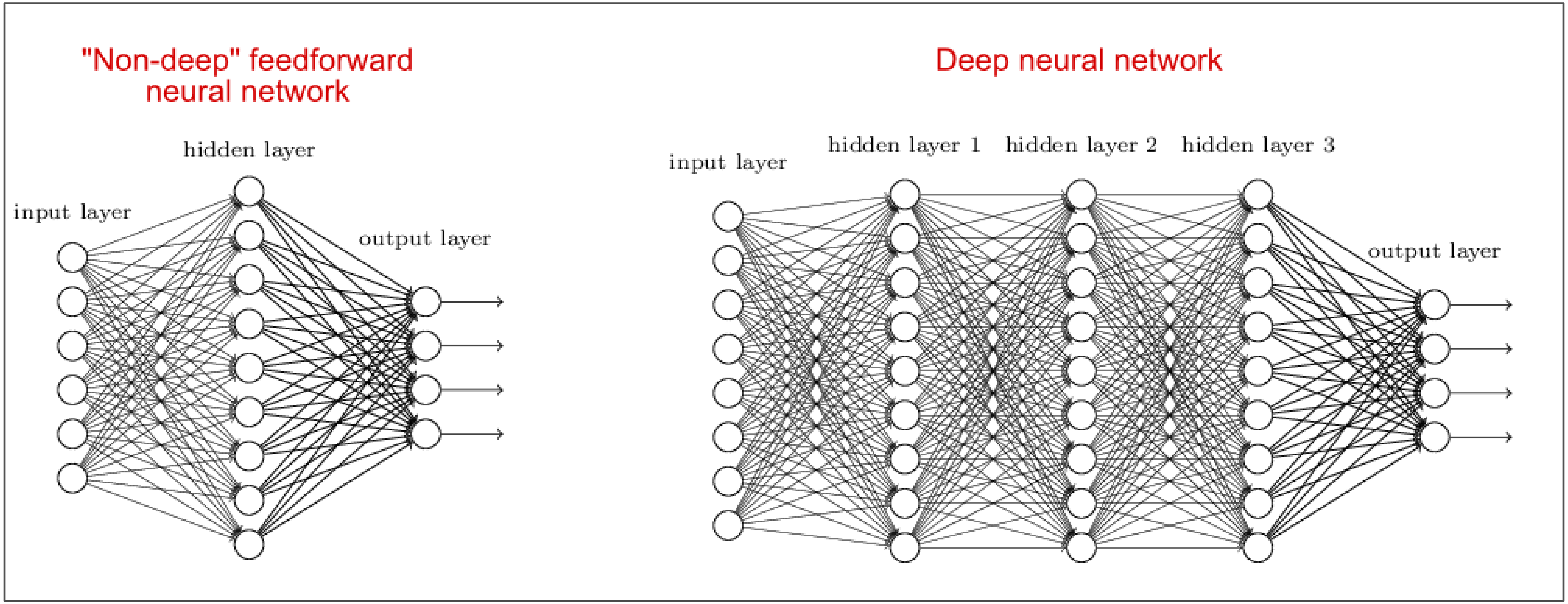
These Little Functions Are Chained Together

- Deep learning comes from chaining a bunch of these little functions together
- Chained together, they are called neurons

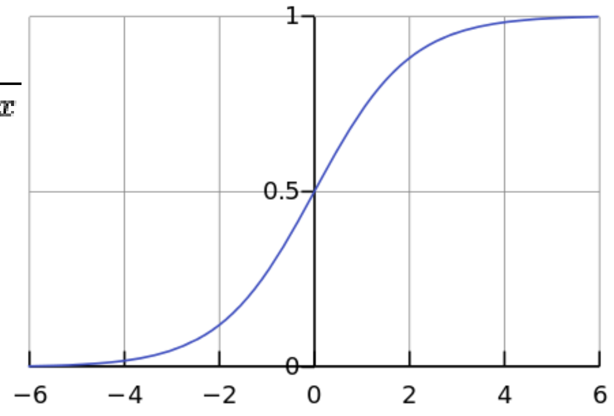


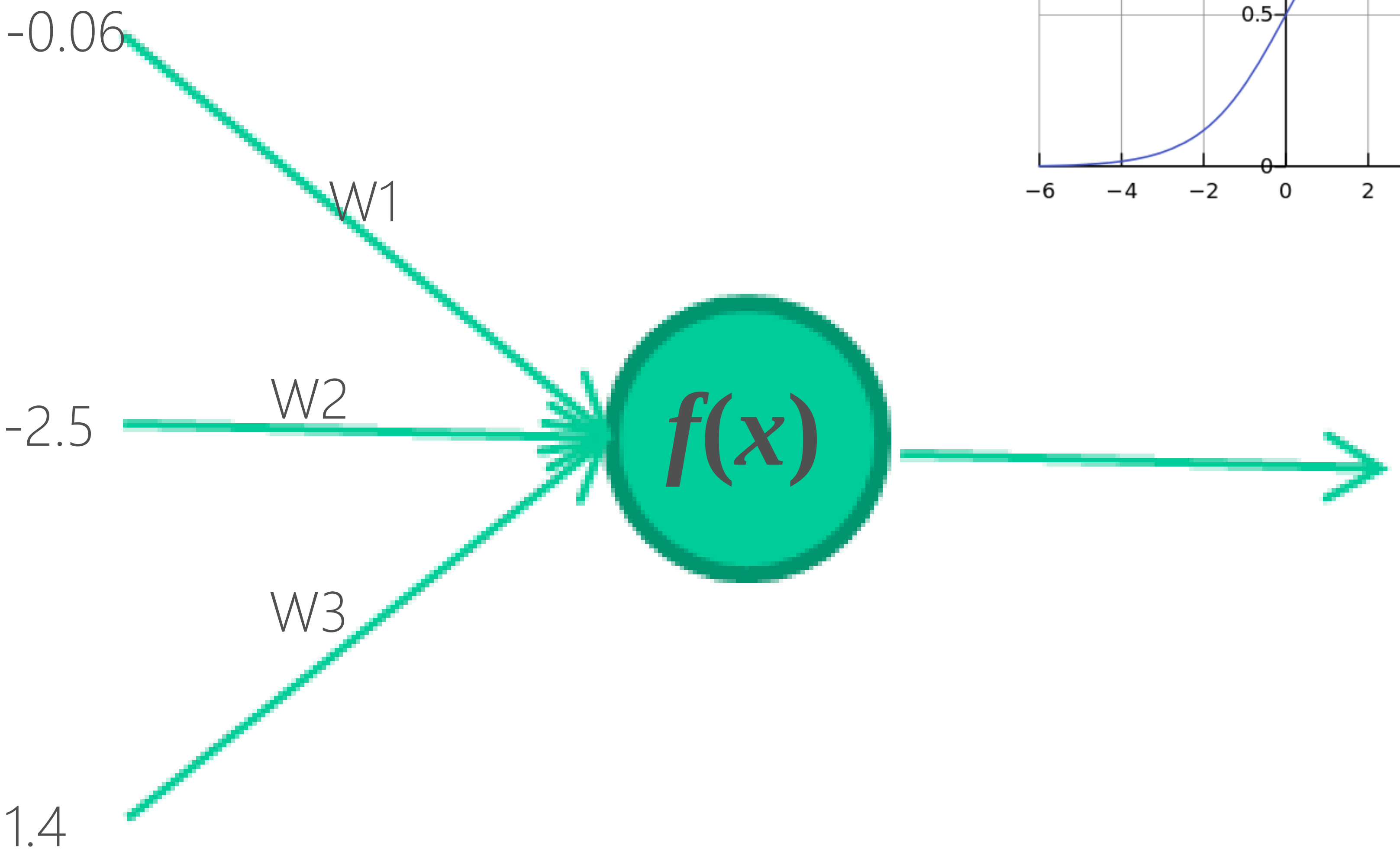
Microsoft

Deep Neural Network (DNN)

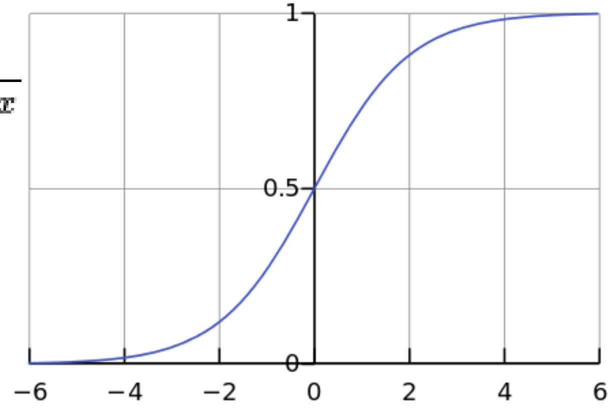


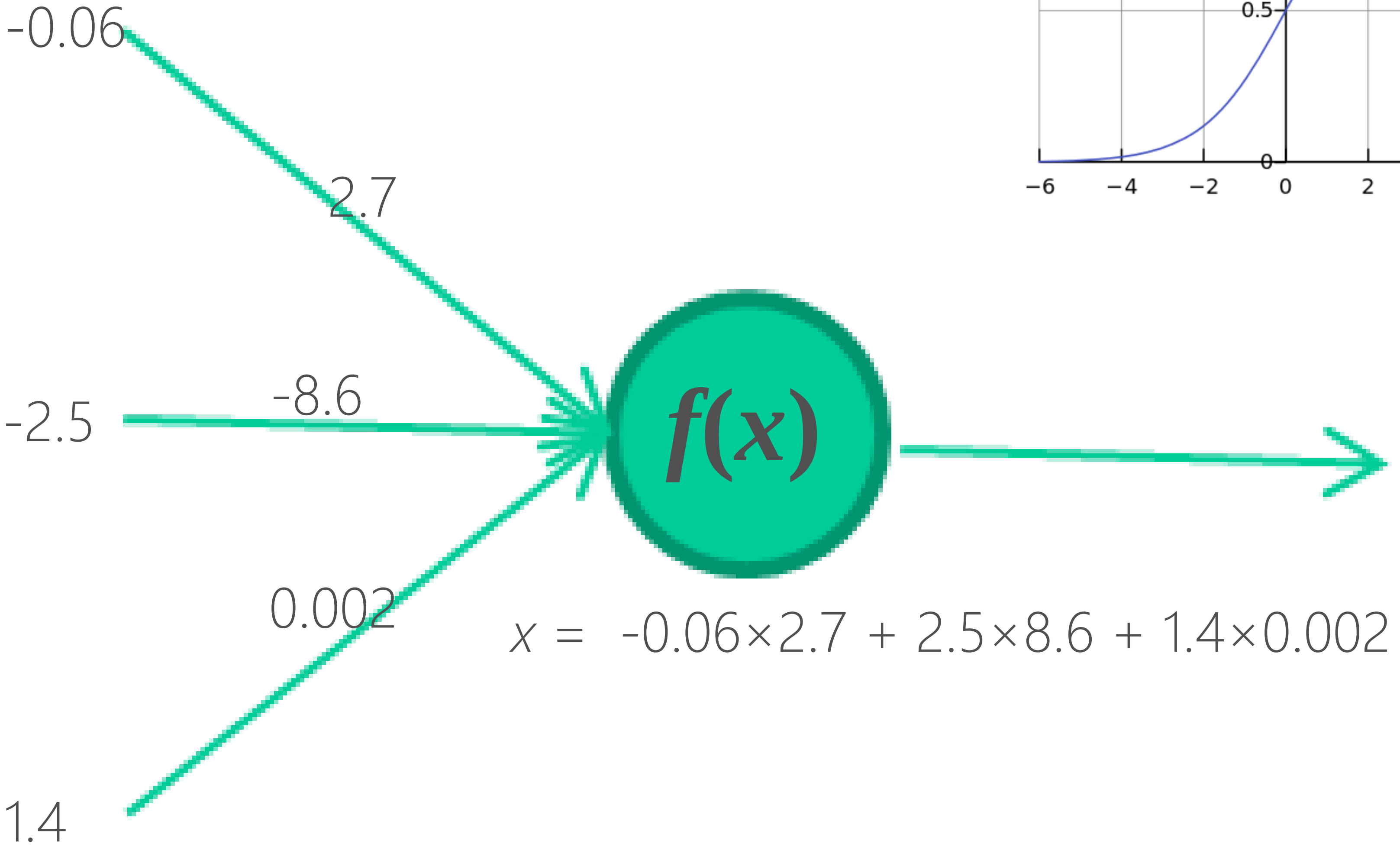
$$f(x) = \frac{1}{1 + e^{-x}}$$





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



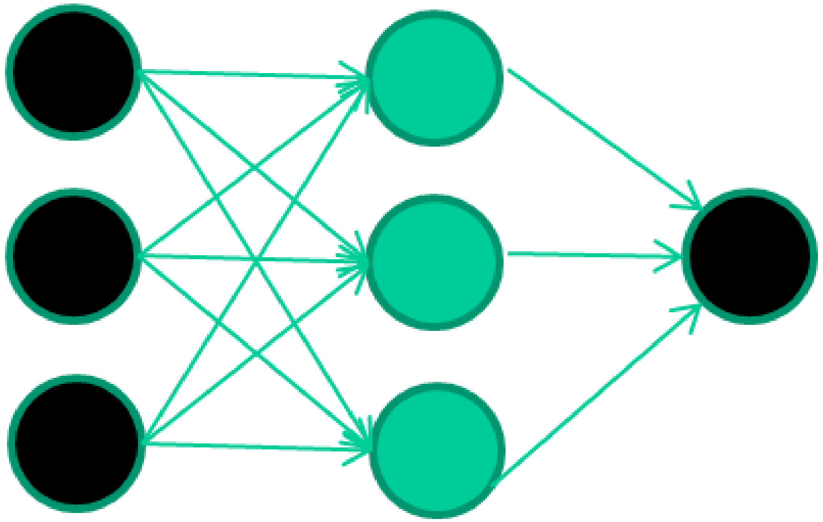


$$x = -0.06\times2.7 + 2.5\times8.6 + 1.4\times0.002 = 21.34$$

Microsoft

A dataset

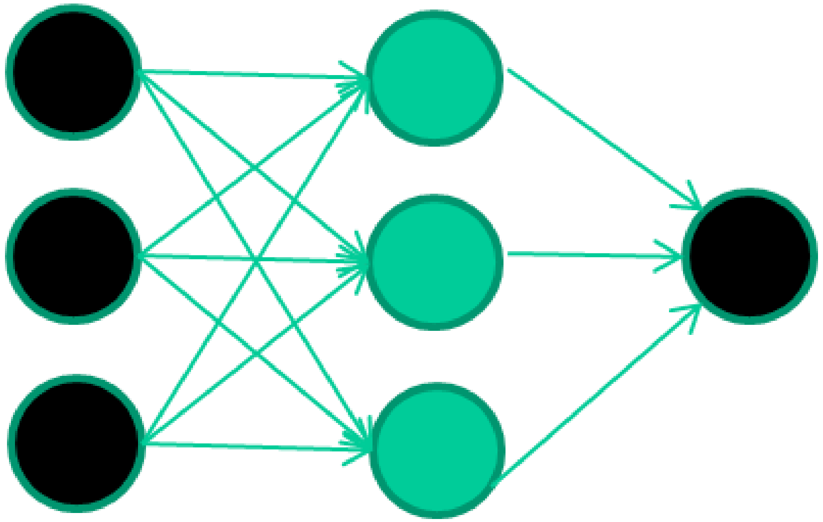
Fields			*class*
1.4	2.7	1.9	0
3.8	3.4	3.2	0
6.4	2.8	1.7	1
4.1	0.1	0.2	0
etc ...			



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

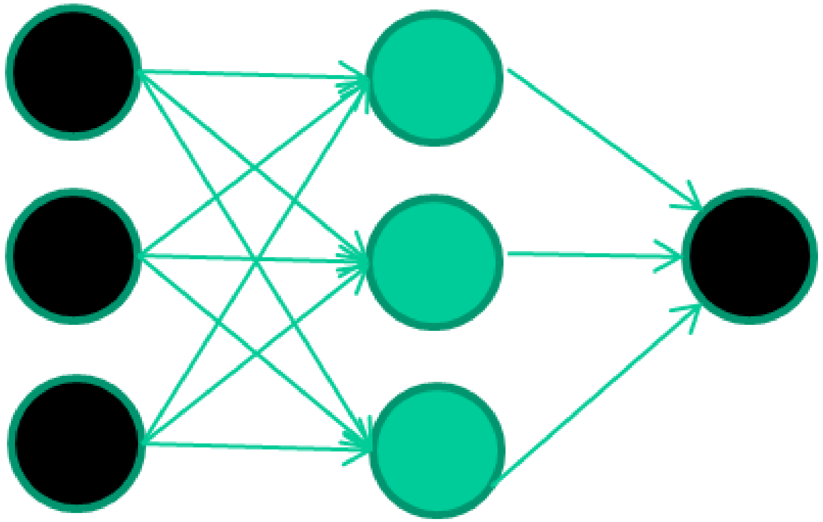
etc ...



Training the neural network

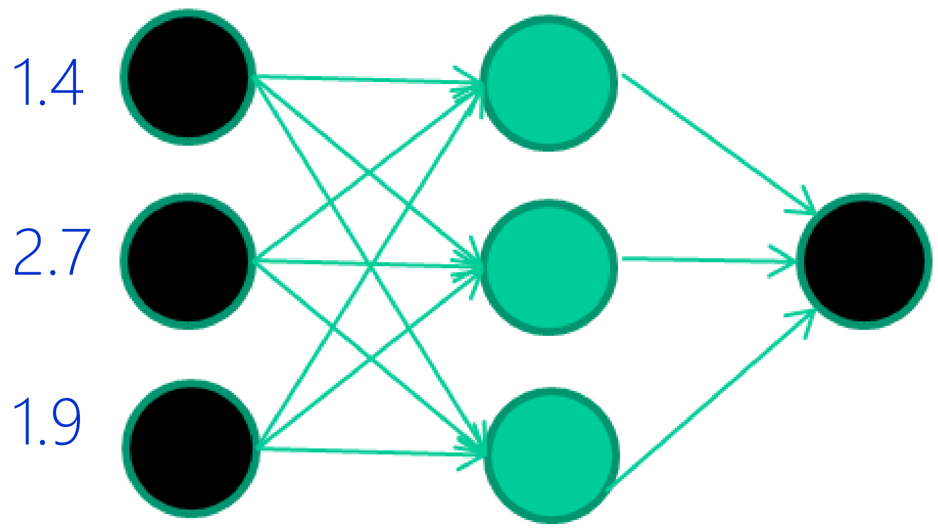
Fields			*class*
1.4	2.7	1.9	0
3.8	3.4	3.2	0
6.4	2.8	1.7	1
4.1	0.1	0.2	0
etc ...			

Initialise with random weights



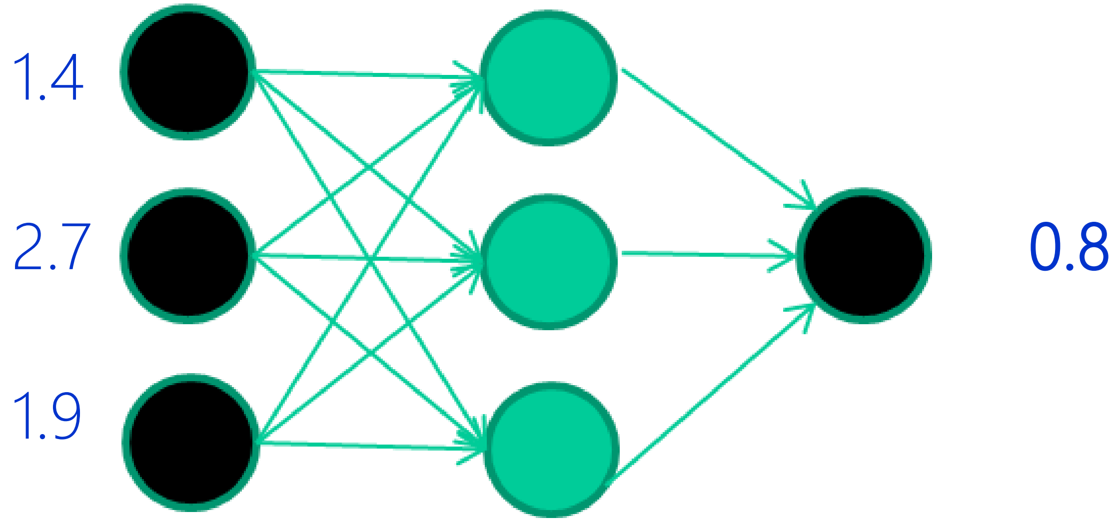
Fields			class
1.4	2.7	1.9	0
3.8	3.4	3.2	0
6.4	2.8	1.7	1
4.1	0.1	0.2	0
etc ...			

Present a training pattern



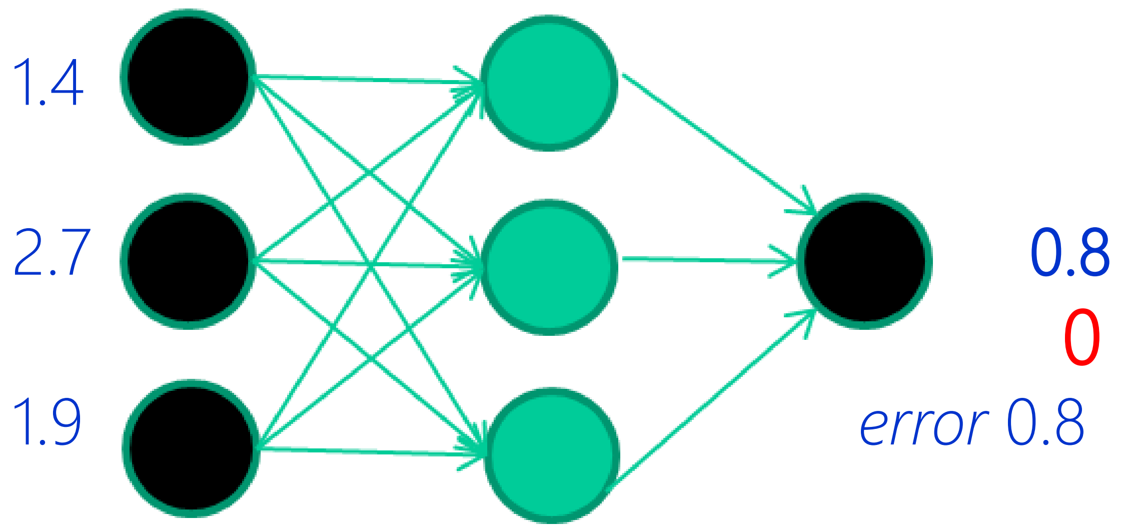
Fields			*class*
1.4	2.7	1.9	0
3.8	3.4	3.2	0
6.4	2.8	1.7	1
4.1	0.1	0.2	0
etc …			

Feed it through to get output



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 ...			

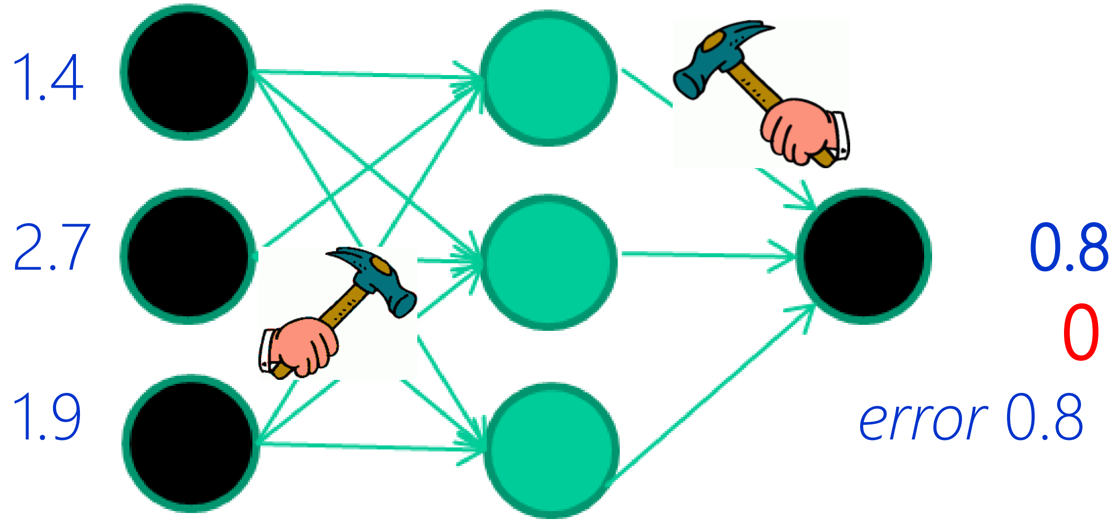
Compare with target output



Microsoft

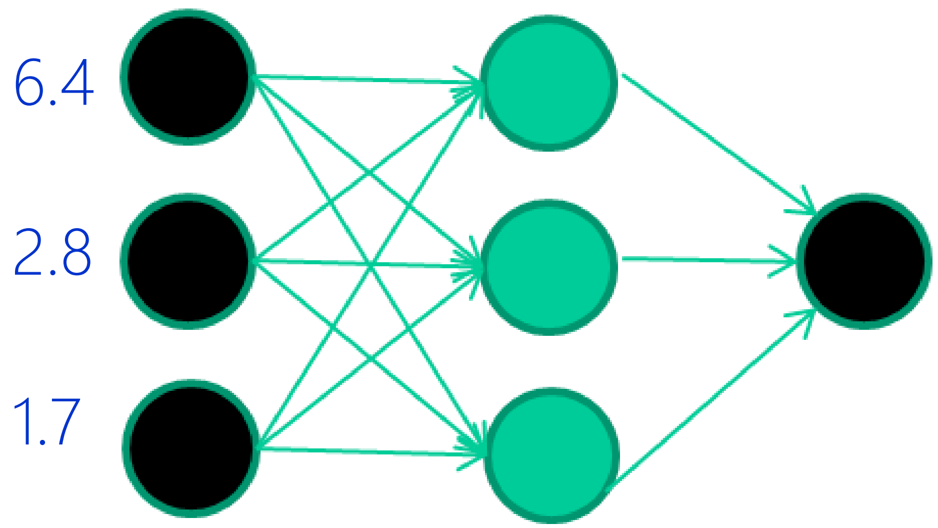
Fields			*class*
1.4	2.7	1.9	0
3.8	3.4	3.2	0
6.4	2.8	1.7	1
4.1	0.1	0.2	0
etc …			

Adjust weights based on error



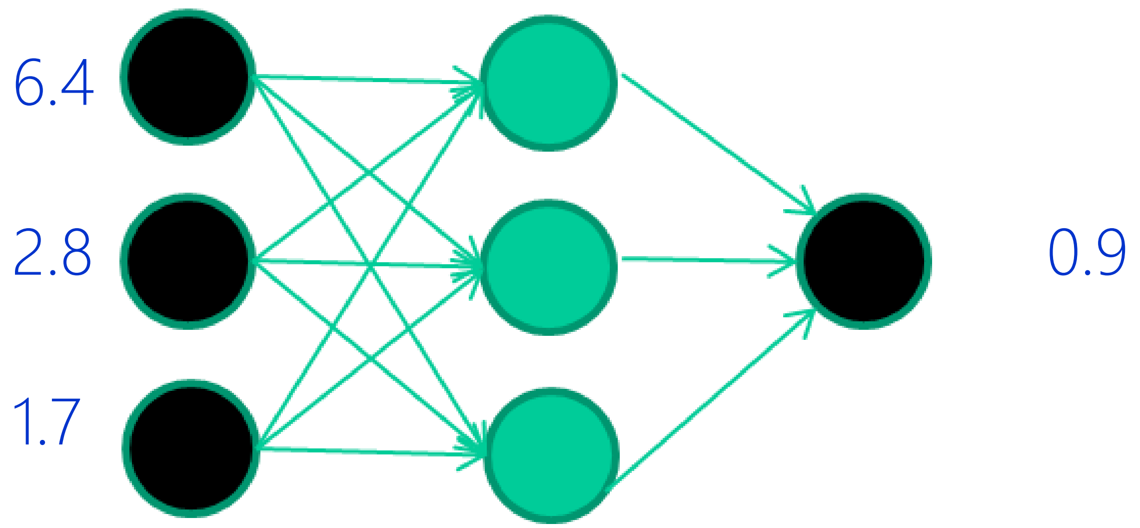
Present a training pattern

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 ...			



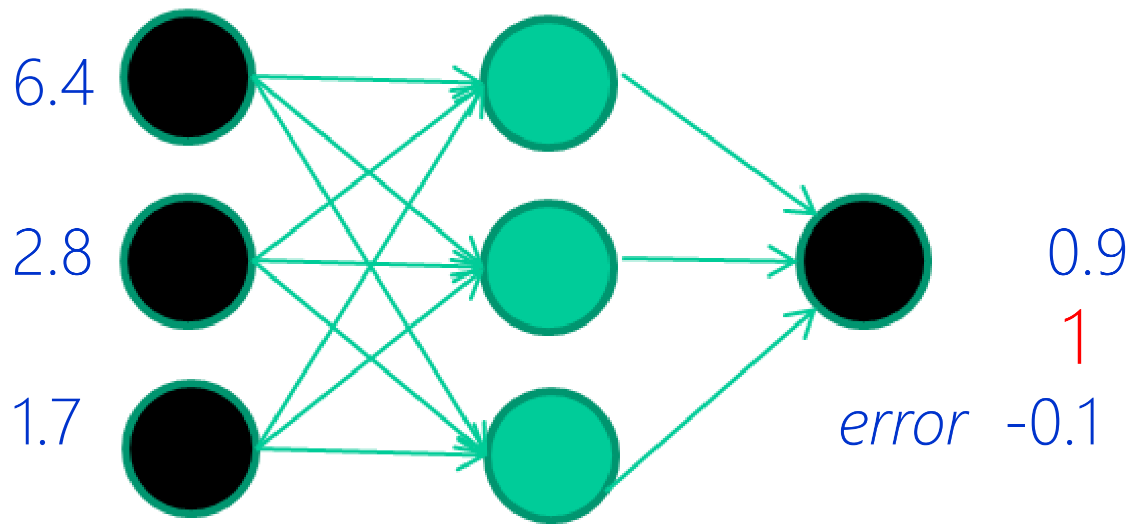
Fields			*class*
1.4	2.7	1.9	0
3.8	3.4	3.2	0
6.4	2.8	1.7	1
4.1	0.1	0.2	0
etc ...			

Feed it through to get output



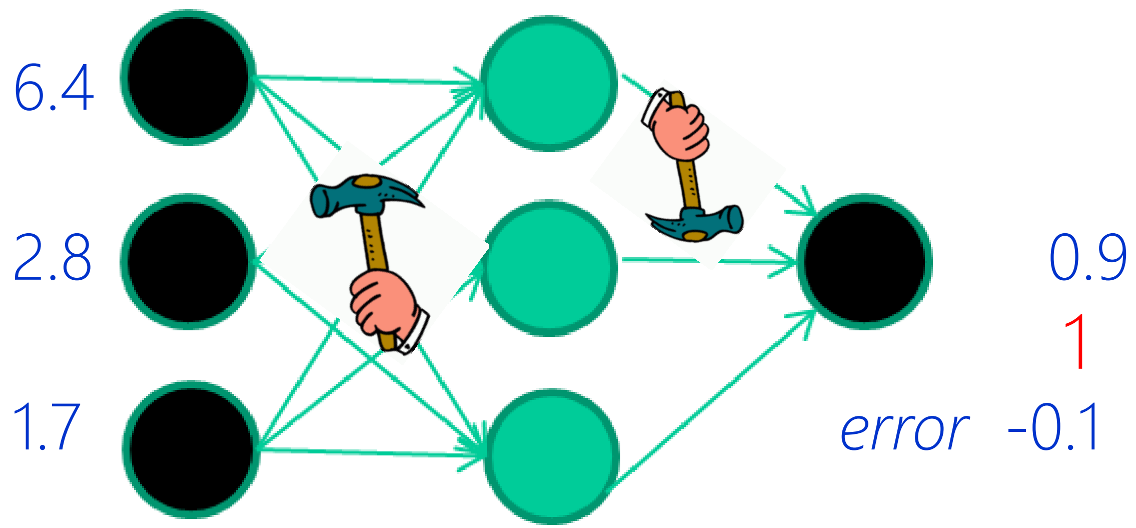
Fields			*class*
1.4	2.7	1.9	0
3.8	3.4	3.2	0
6.4	2.8	1.7	1
4.1	0.1	0.2	0
etc ...			

Compare with target output



And so on ....

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 ...			

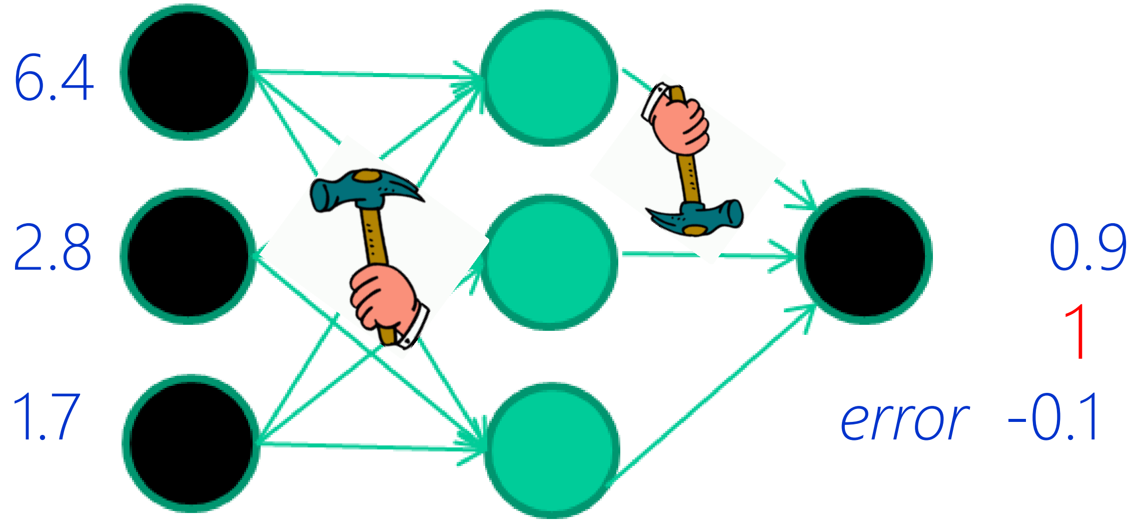


Repeat this thousands, maybe millions of times – each time taking a random training instance, and making slight weight adjustments

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

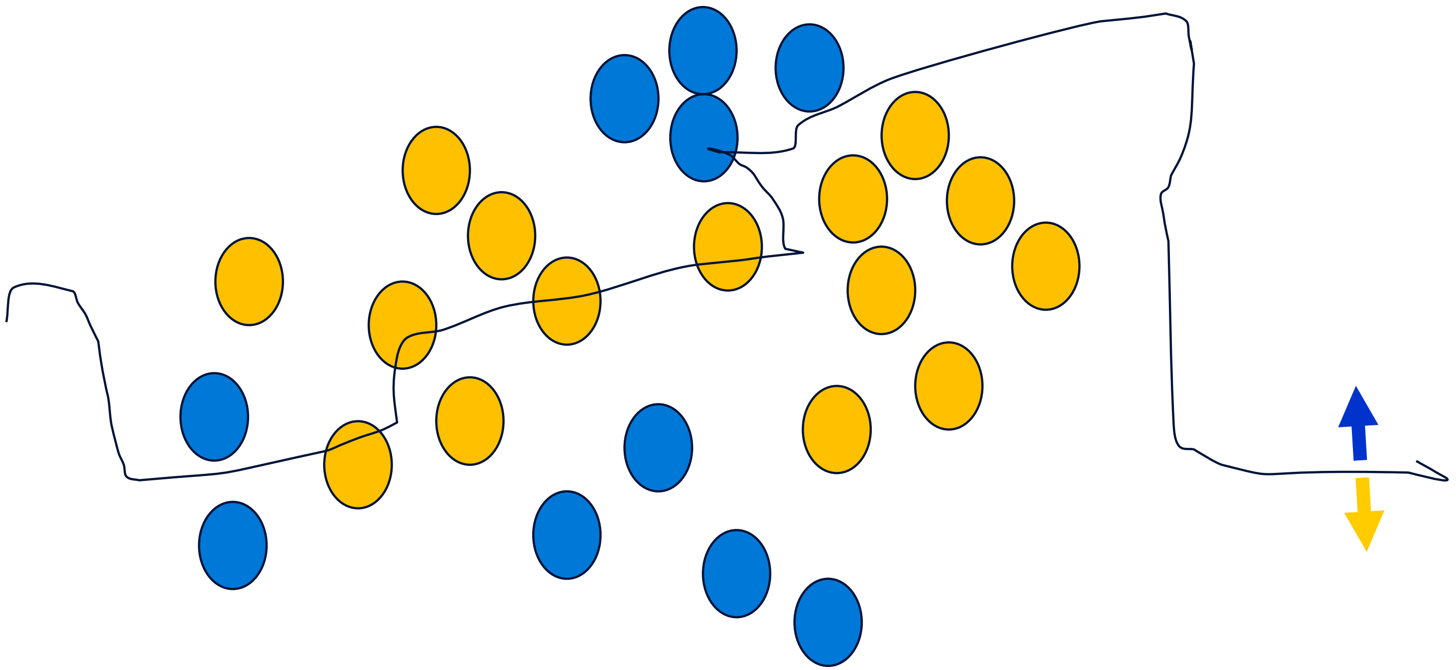
Adjust weights based on error

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 ...			



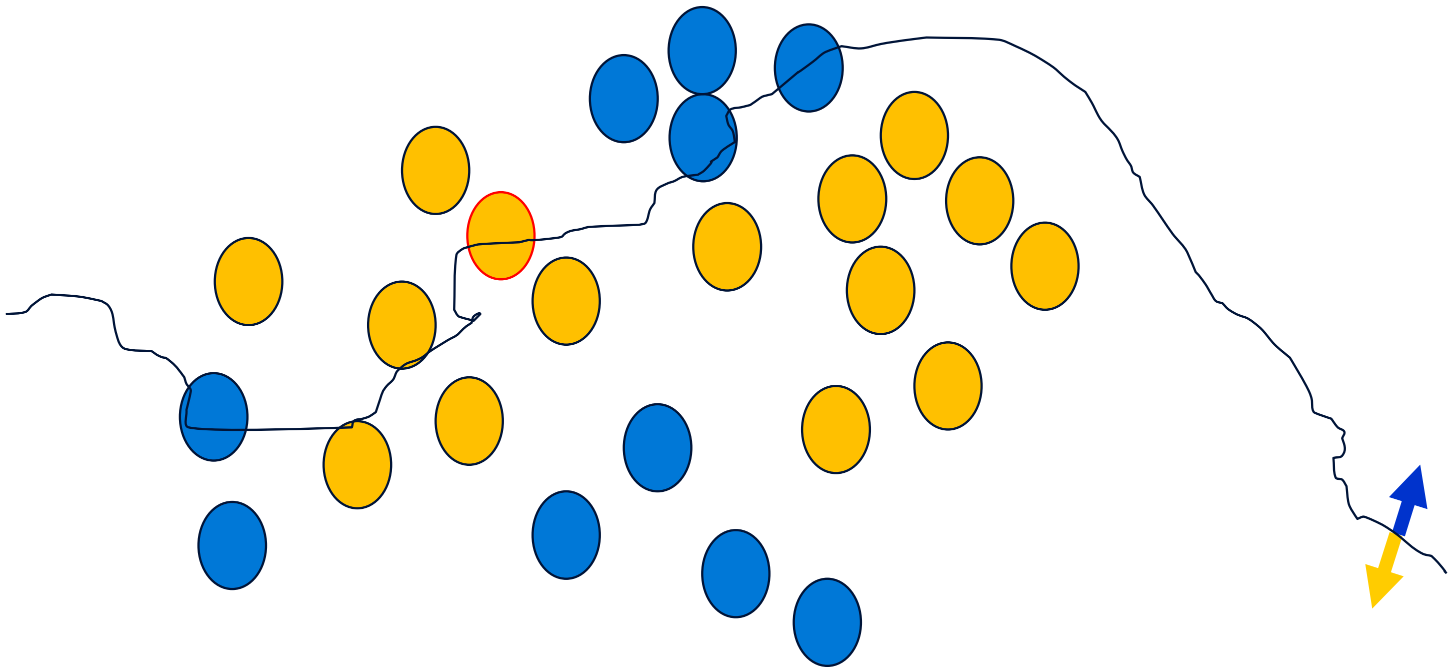
The Decision Boundary Perspective…

Initial random weights



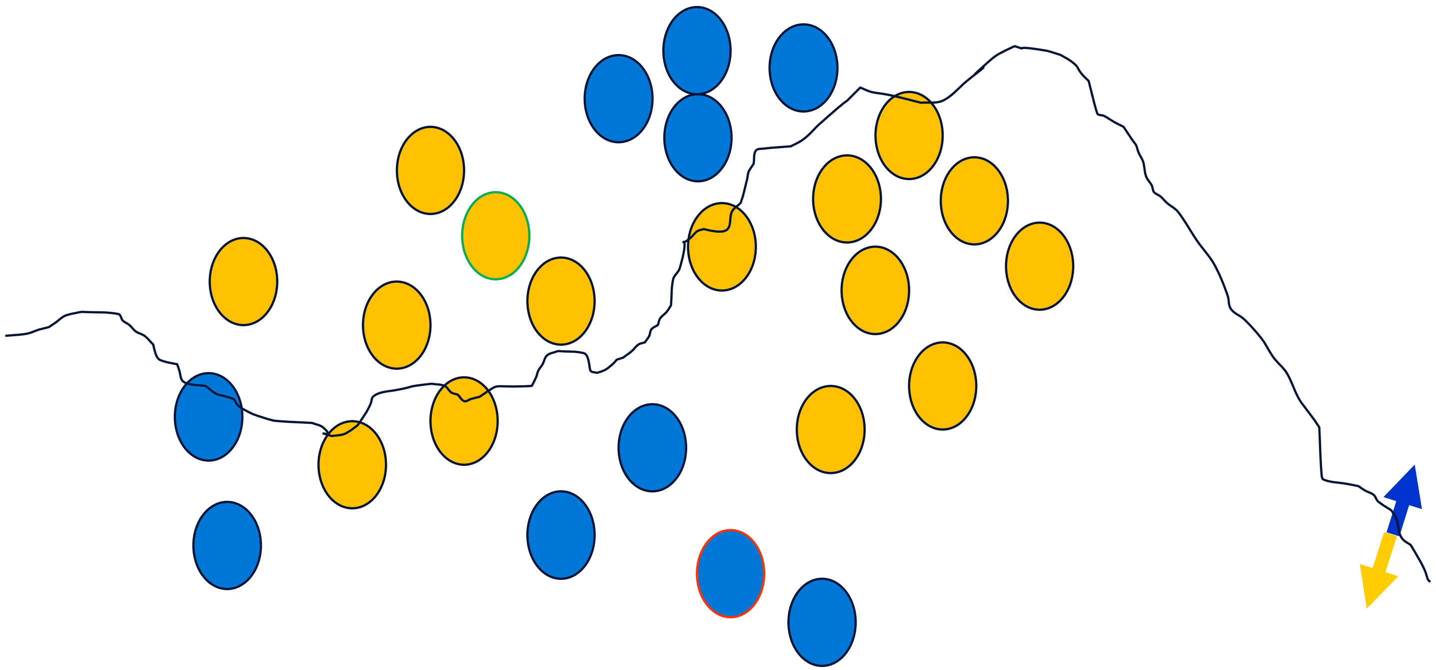
The Decision Boundary Perspective…

Present a training instance / adjust the weights



The Decision Boundary Perspective…

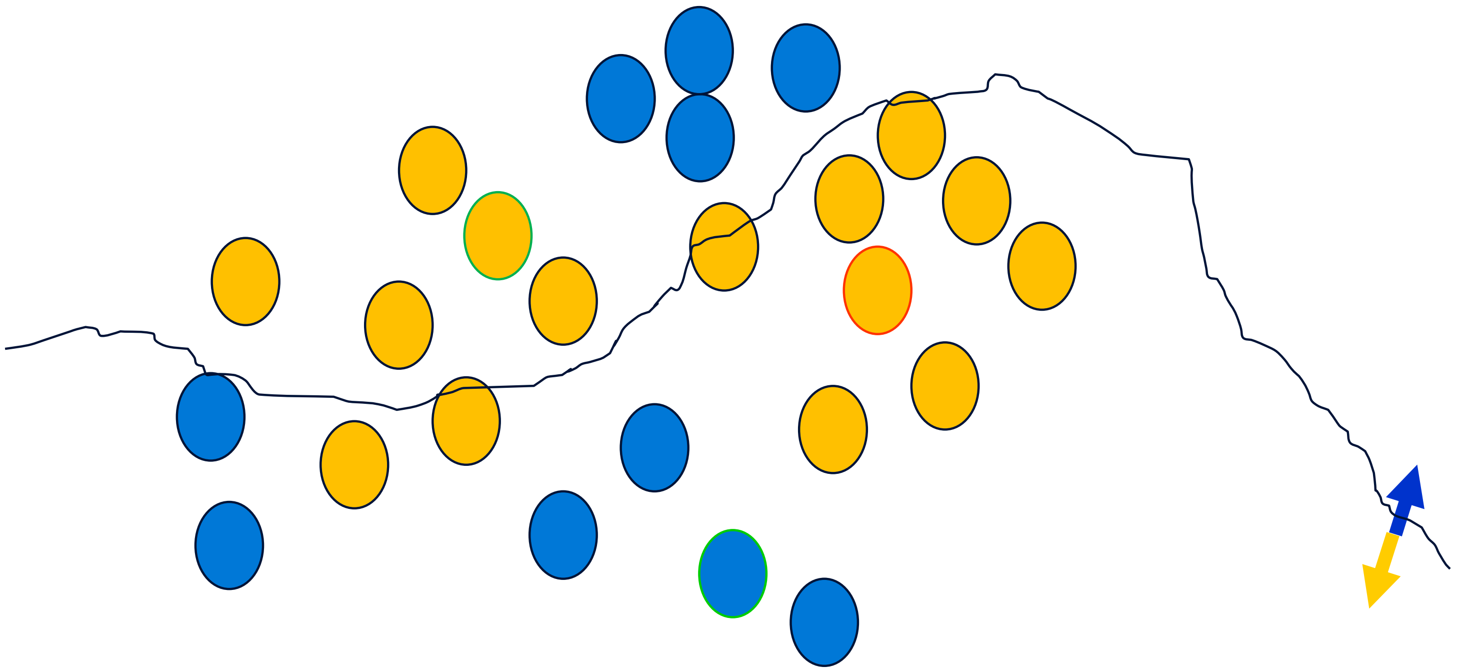
Present a training instance / adjust the weights



Microsoft

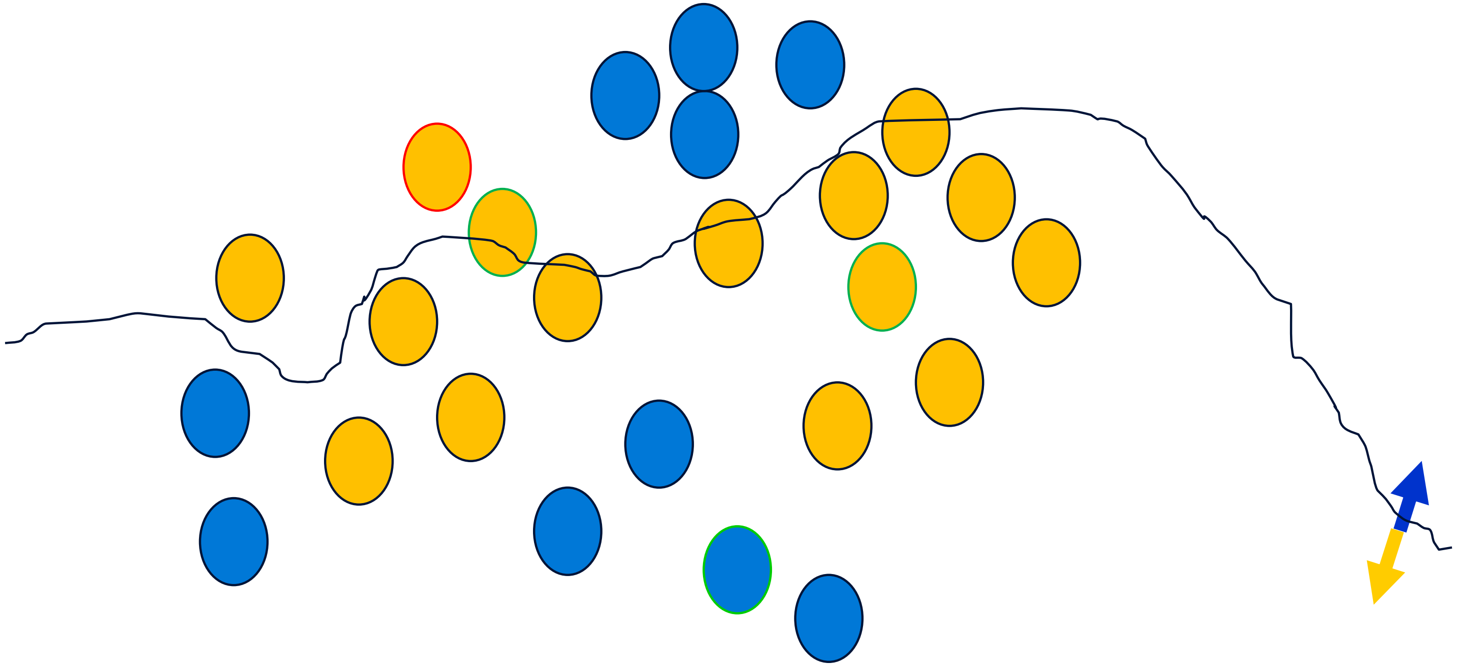
The Decision Boundary Perspective…

Present a training instance / adjust the weights



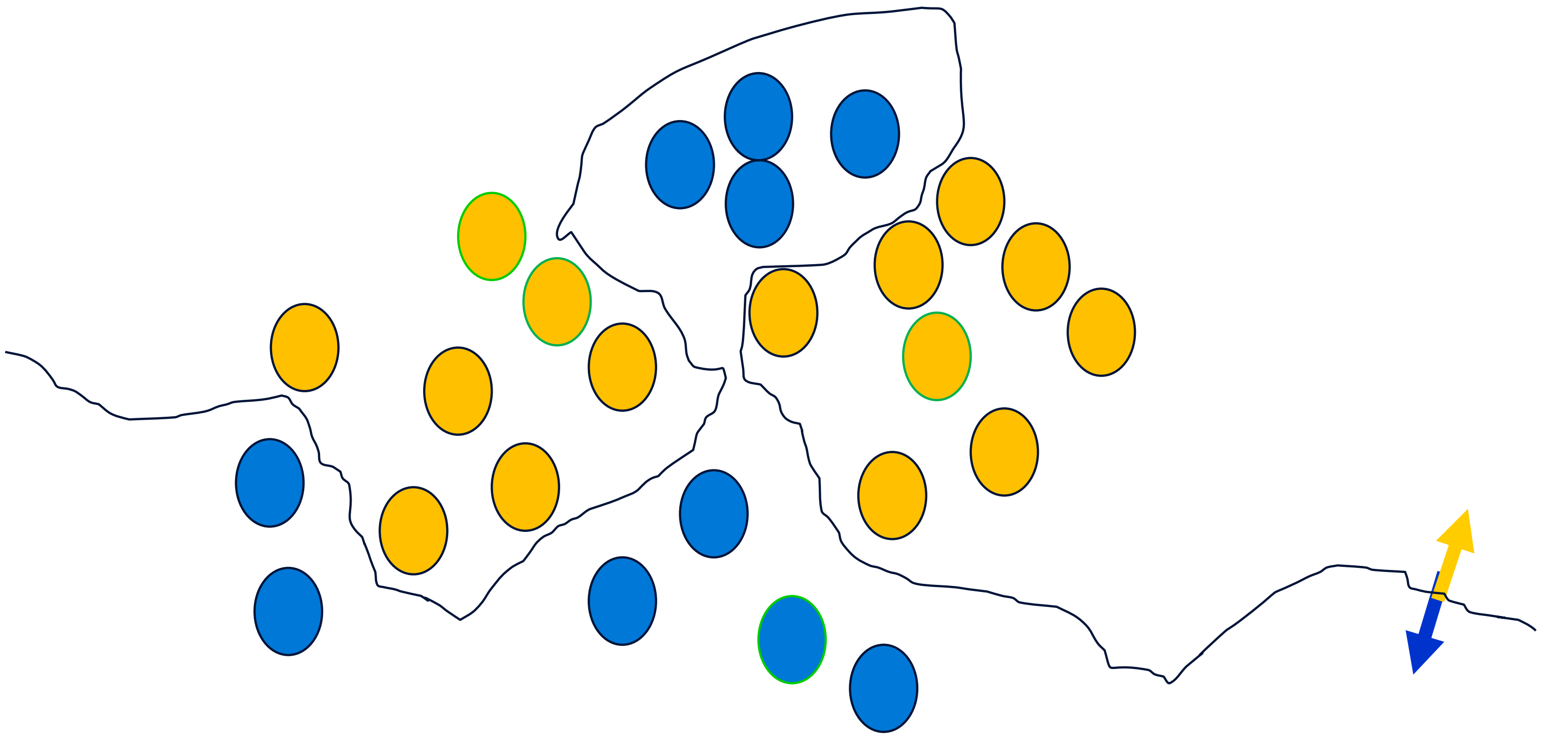
The Decision Boundary Perspective…

Present a training instance / adjust the weights



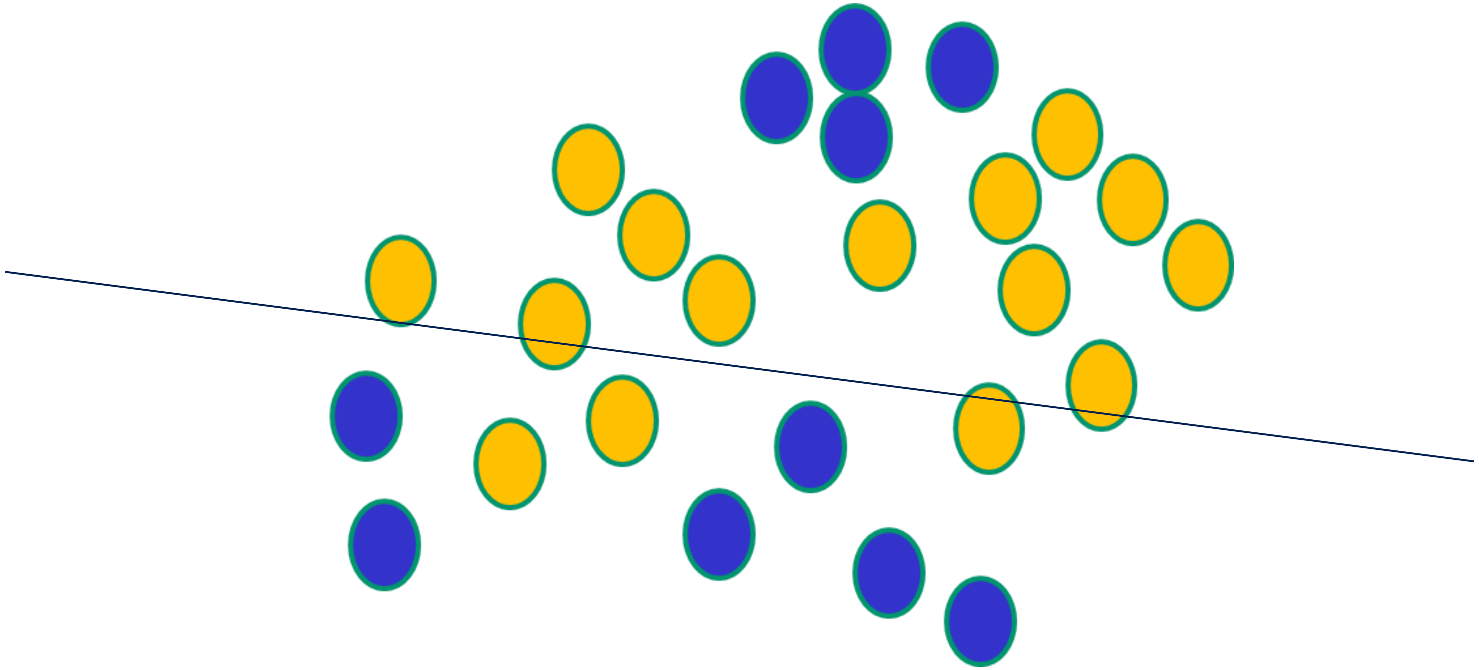
The Decision Boundary Perspective…

Eventually ….



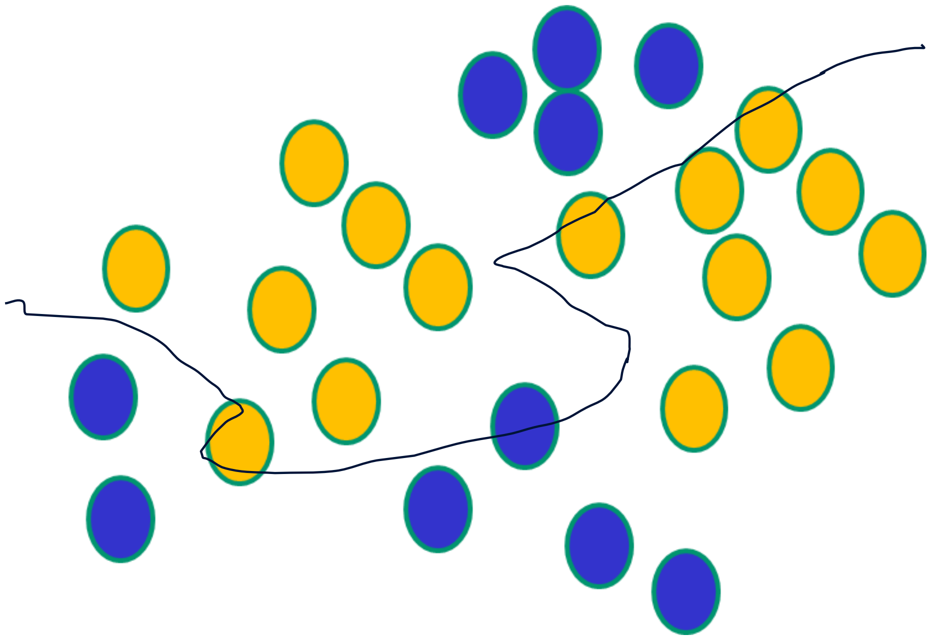
Some Other 'By The Way' Points

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



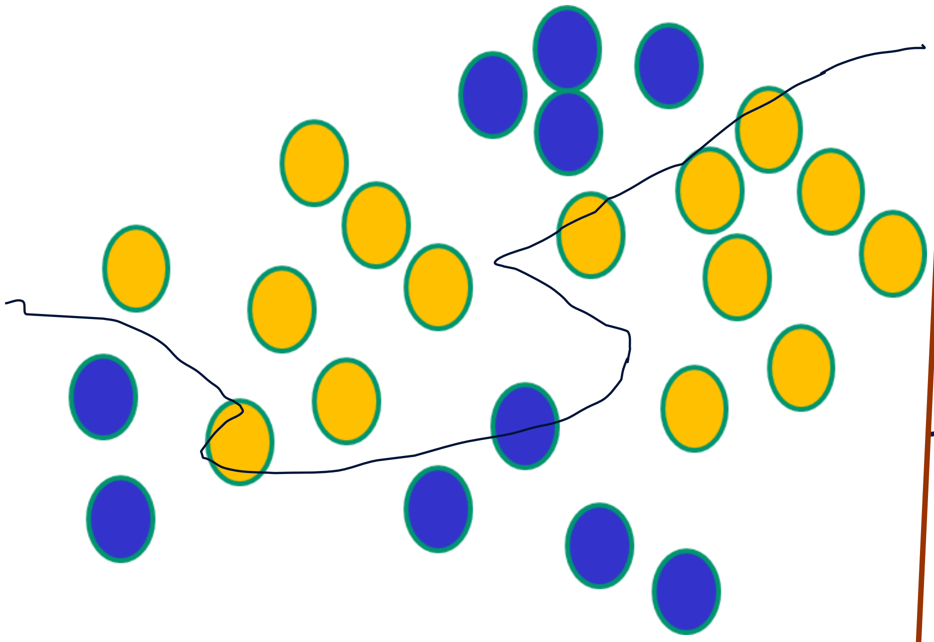
Some Other 'By The Way' Points

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

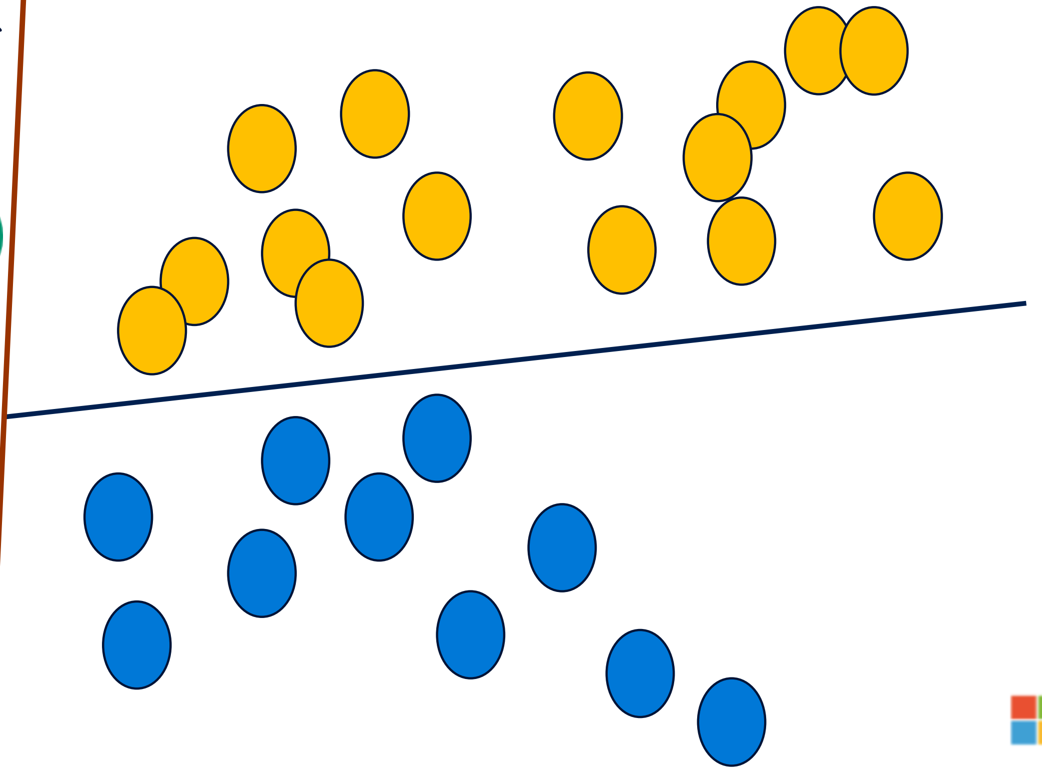


Some Other 'By The Way' Points

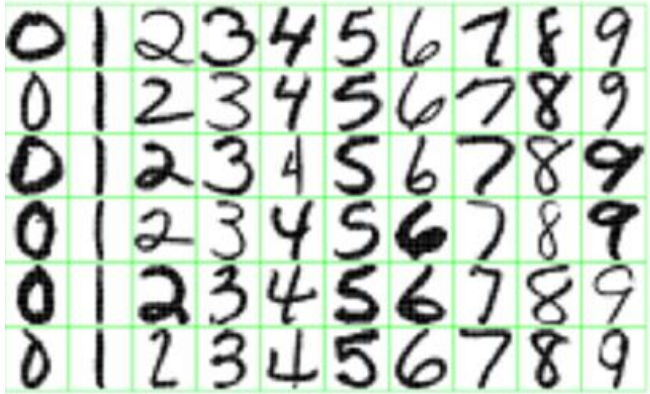
NNs use nonlinear *f*(x) so they can draw complex boundaries, but keep the data unchanged

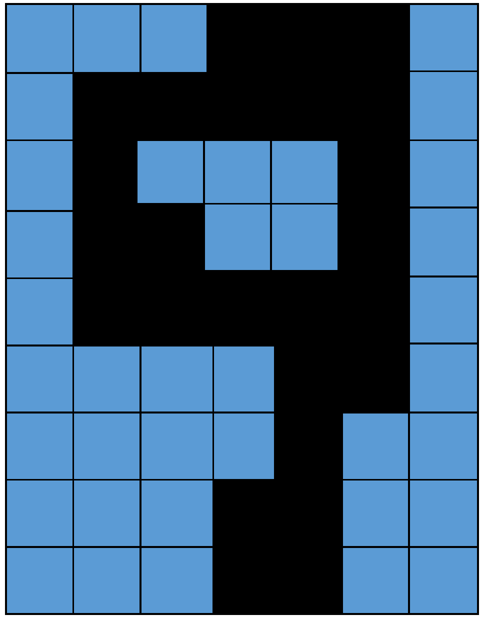


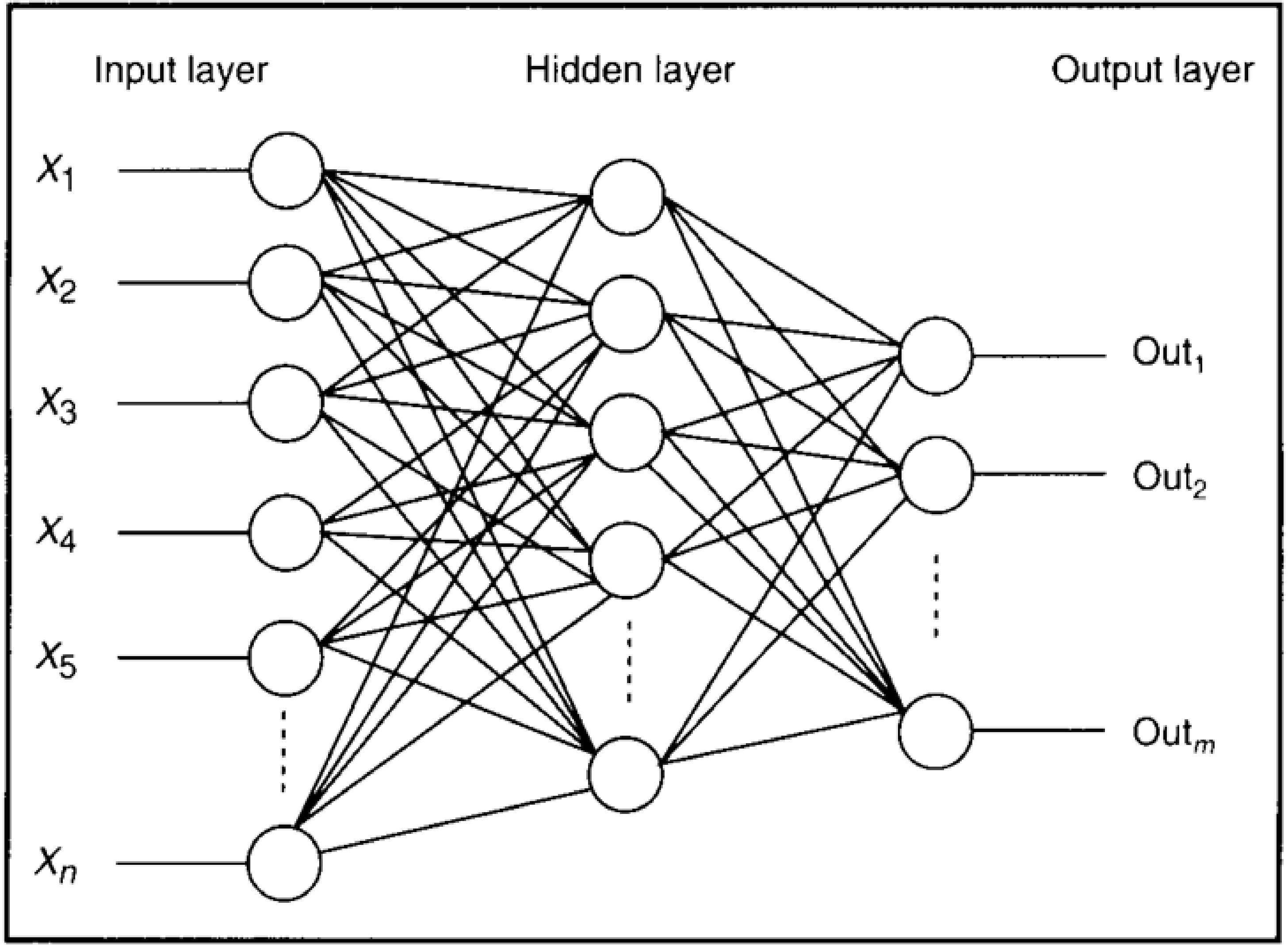
SVMs only draw straight lines, but they transform the data first in a way that makes that OK



Feature Detectors

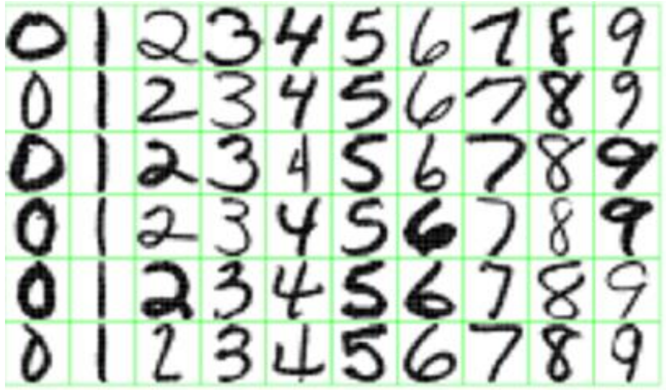


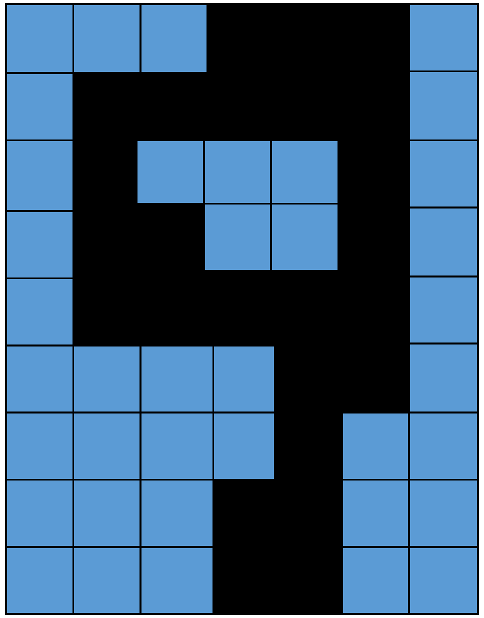


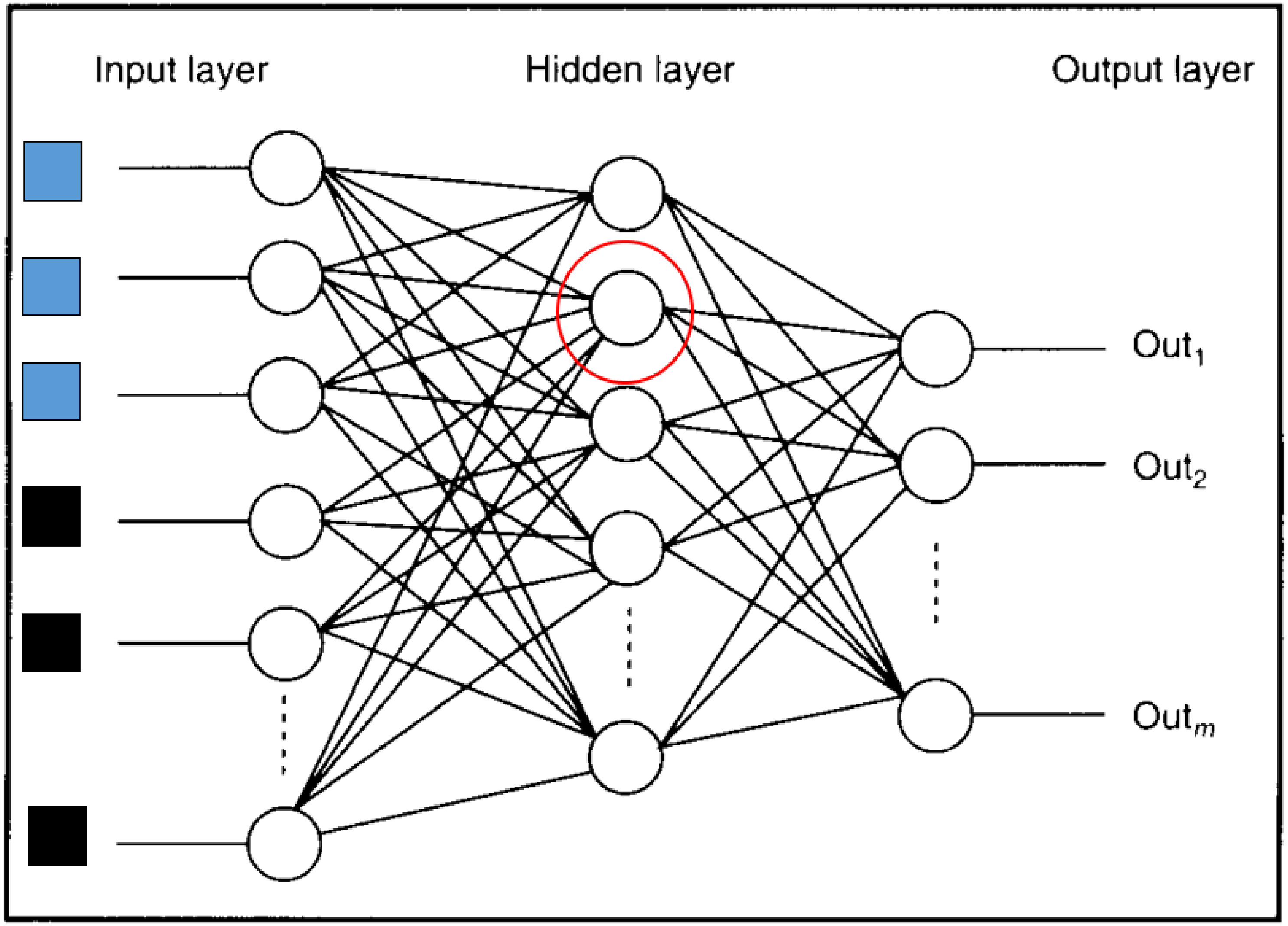


What Is This Unit Doing?

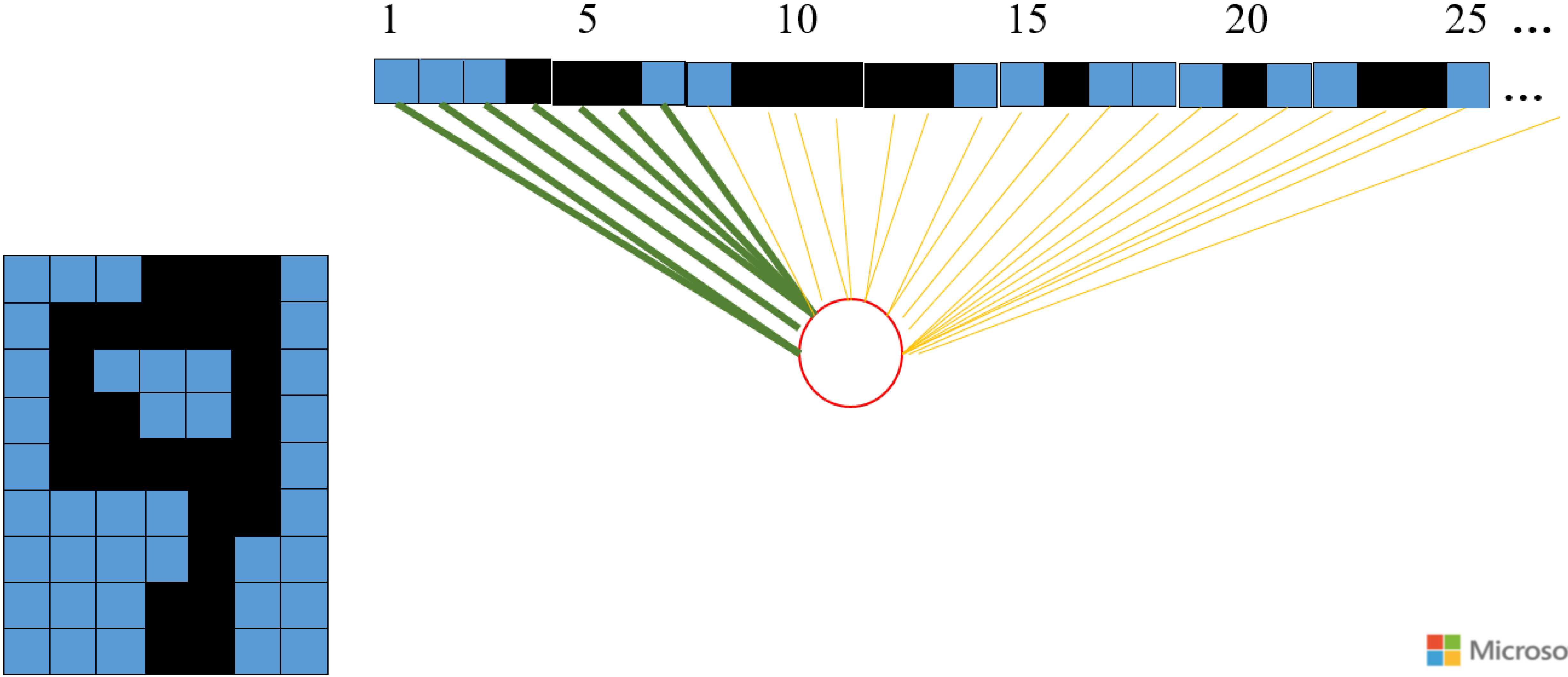
Microsoft



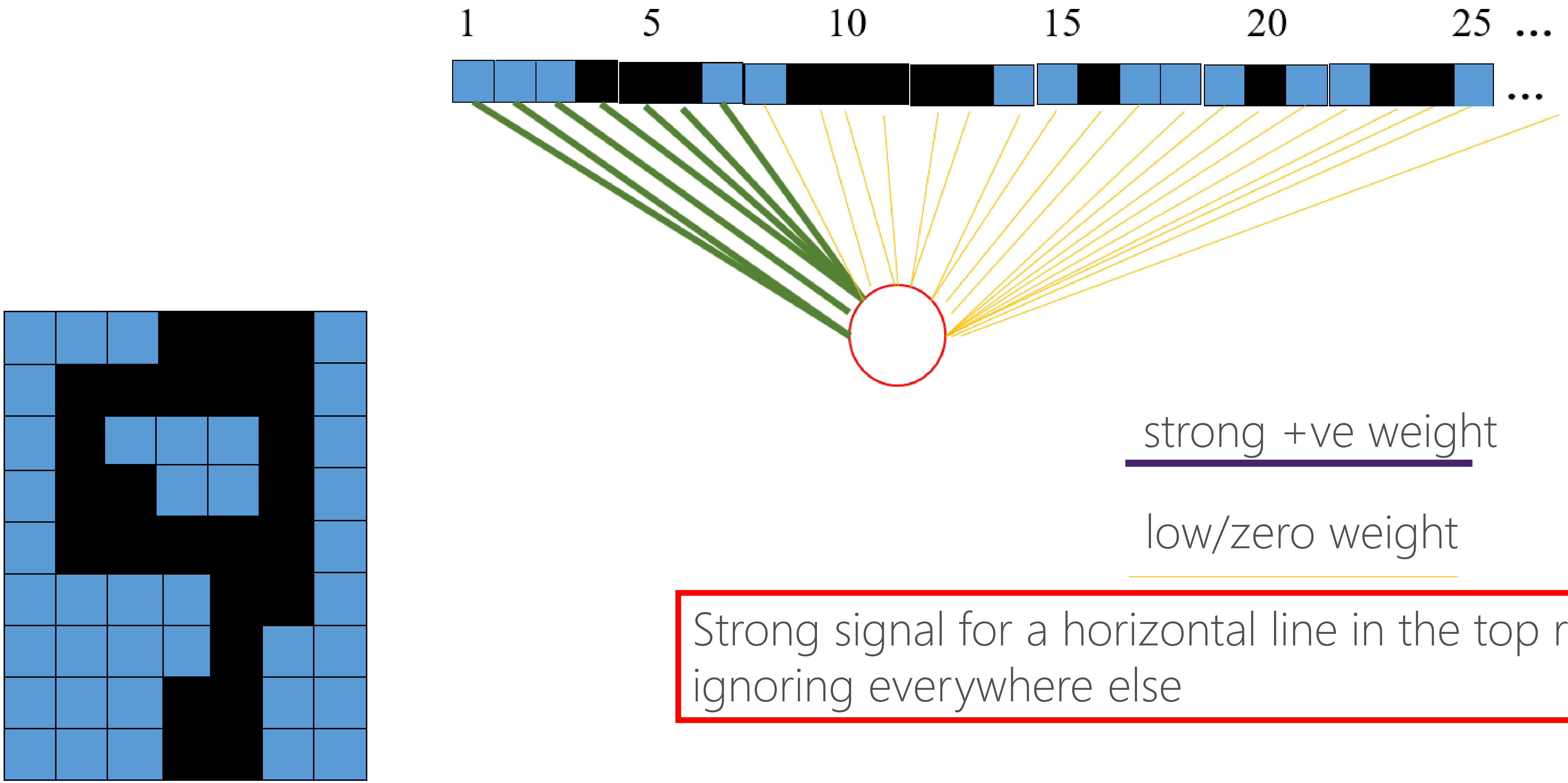




Hidden Layer Units Become Self-Organised Feature Detectors



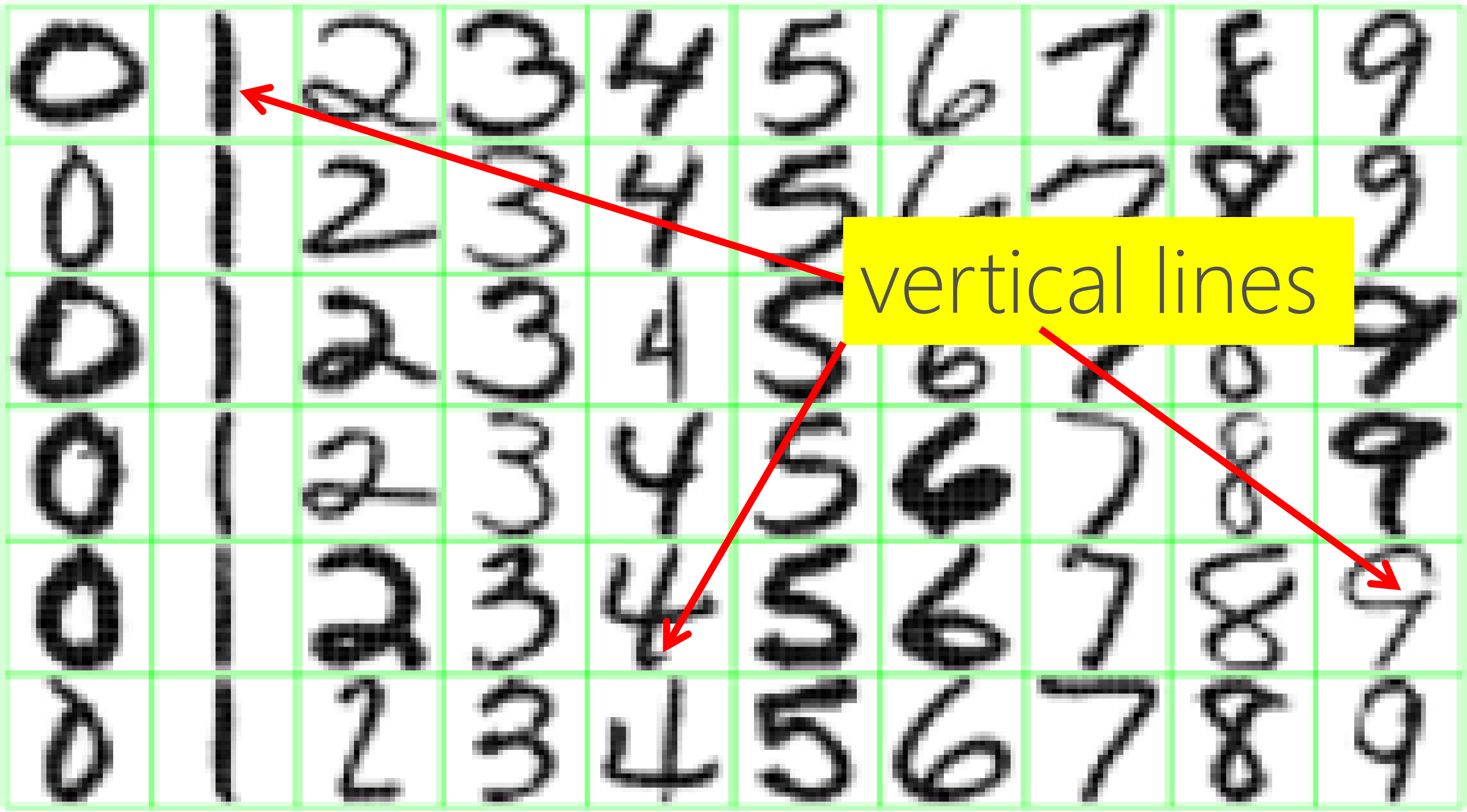
What Does This Unit Detect?



Strong signal for a horizontal line in the top row, ignoring everywhere else

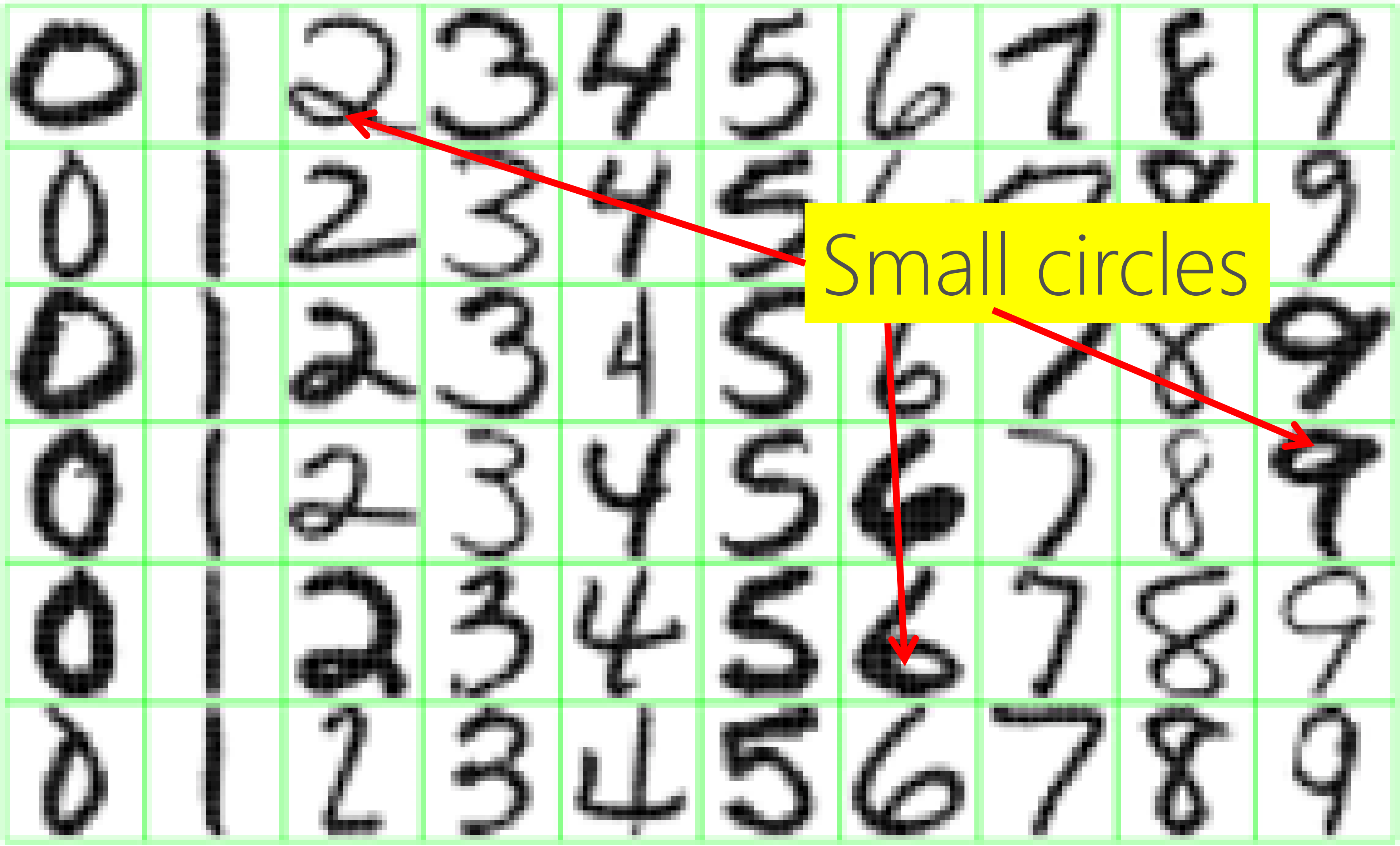
Microsoft

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

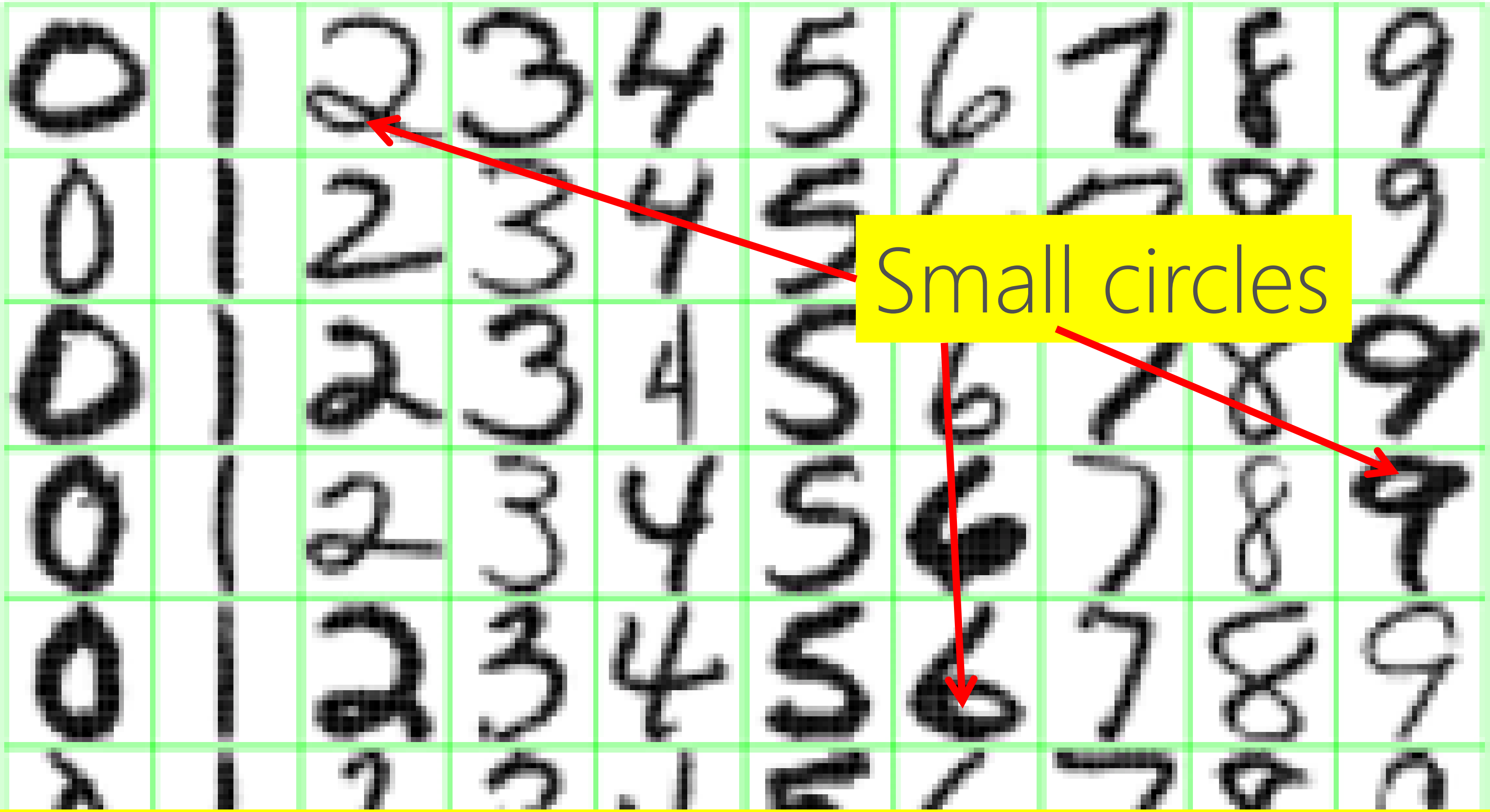


Microsoft

Microsoft

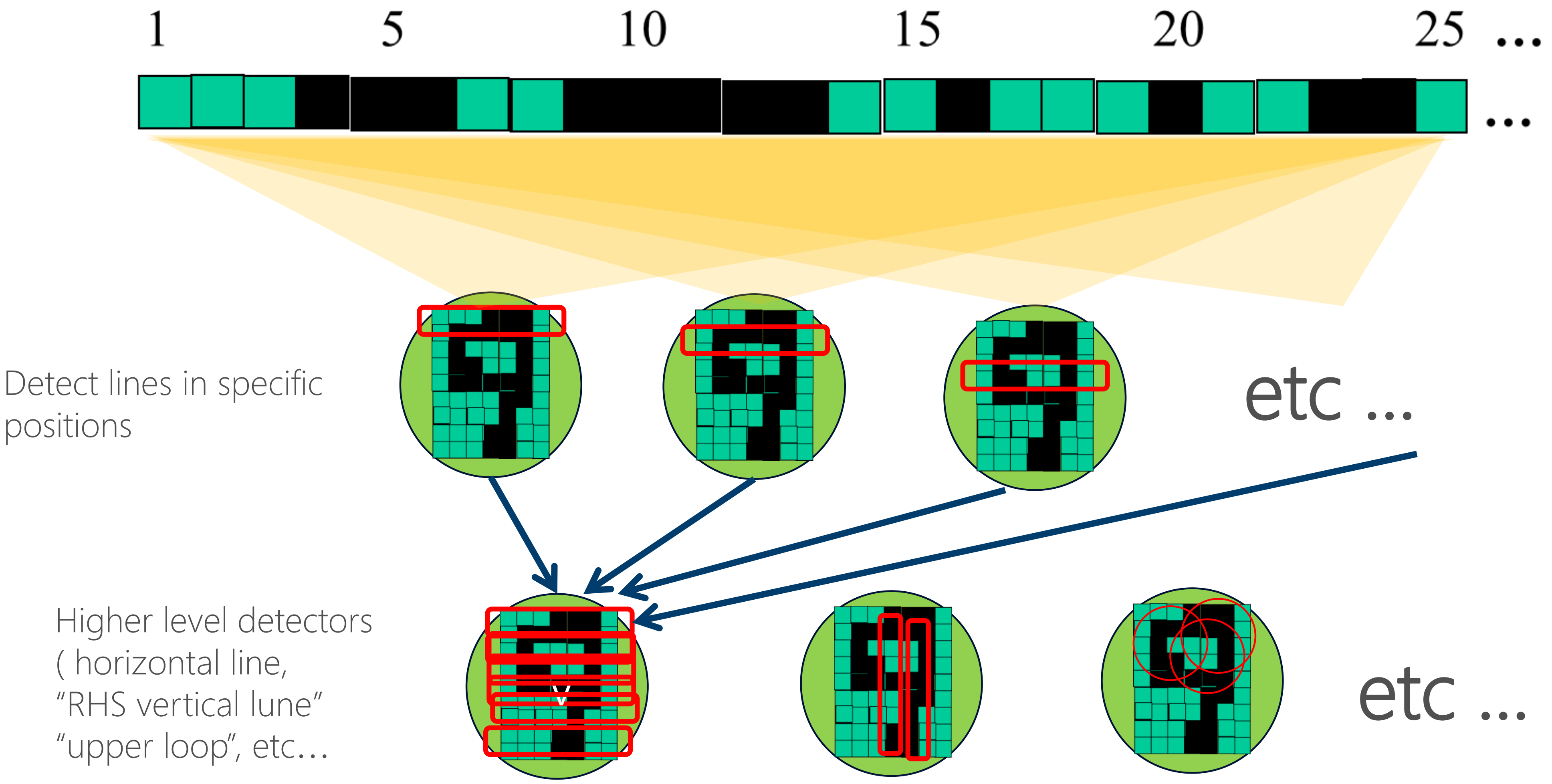


Microsoft

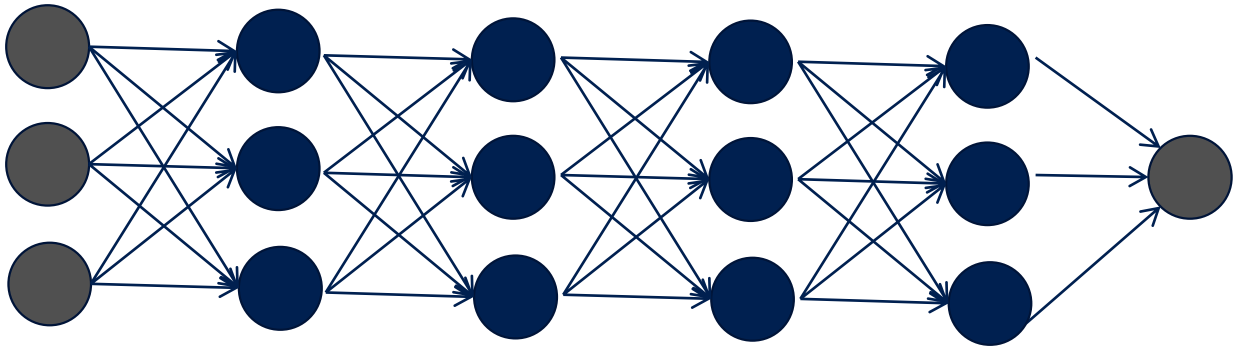


But what about position invariance ???
our example unit detectors were tied to specific parts of the image

Successive Layers Can Learn Higher-Level Features

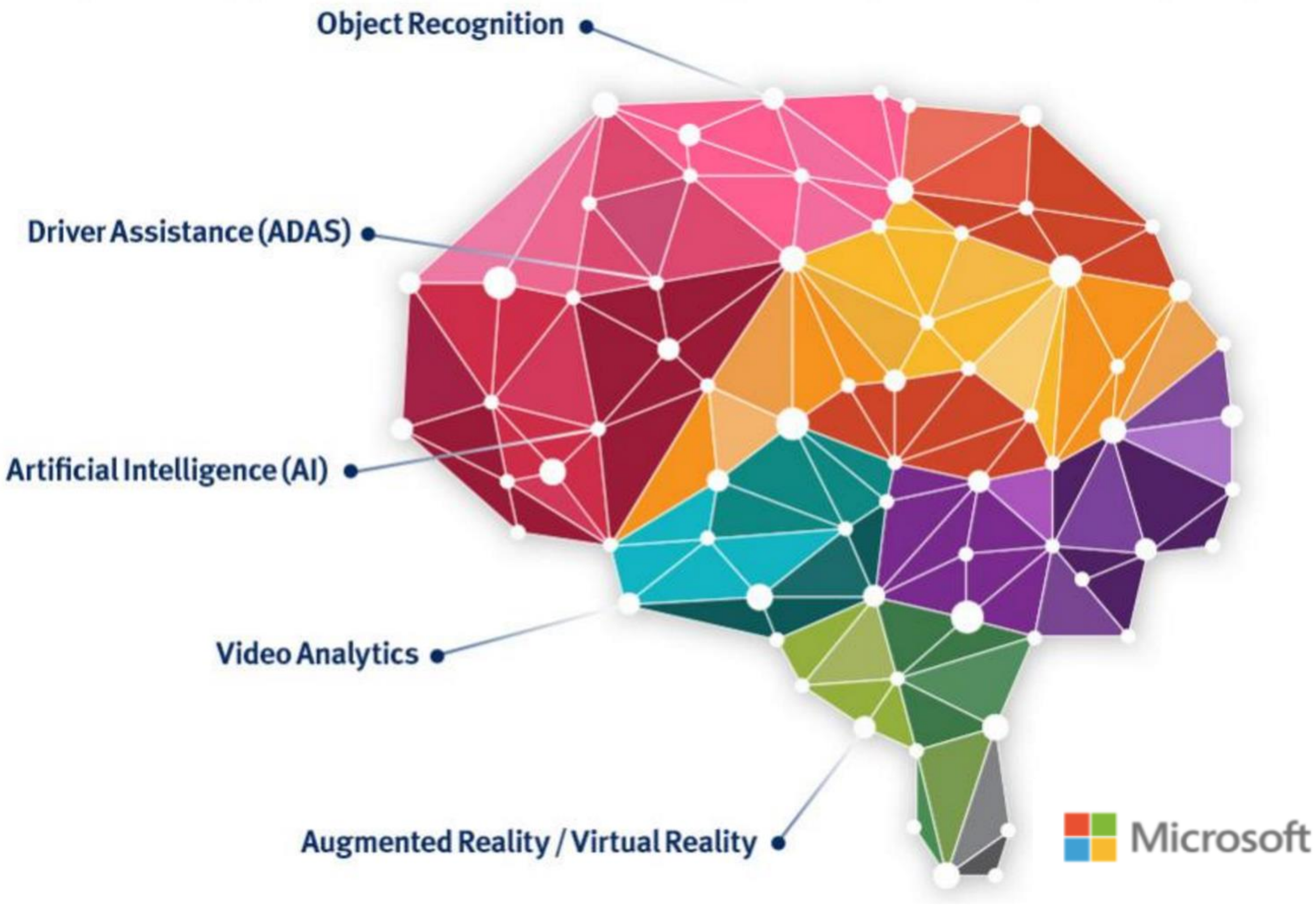


Multiple Layers Make Sense



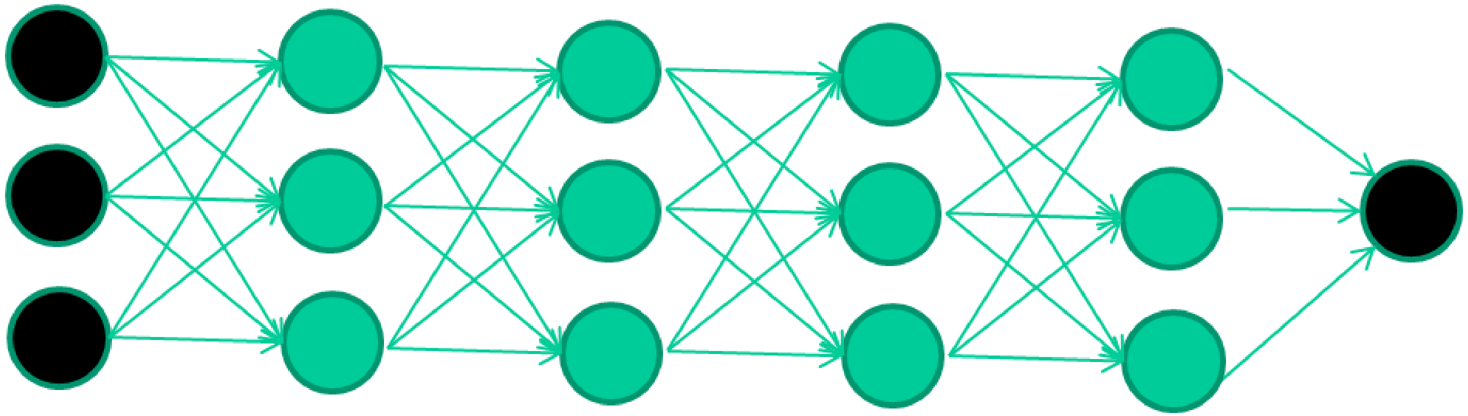
Multiple Layers Make Sense

- Deep Learning = Brain “inspired”
- Audio / Visual Cortex has multiple stages = Hierarchical

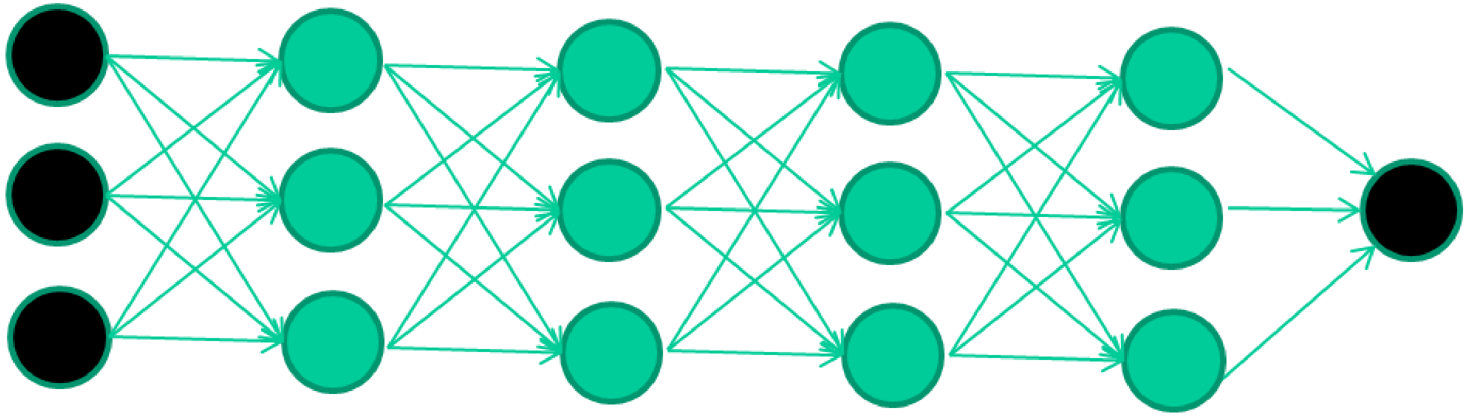


Multiple Layers Make Sense

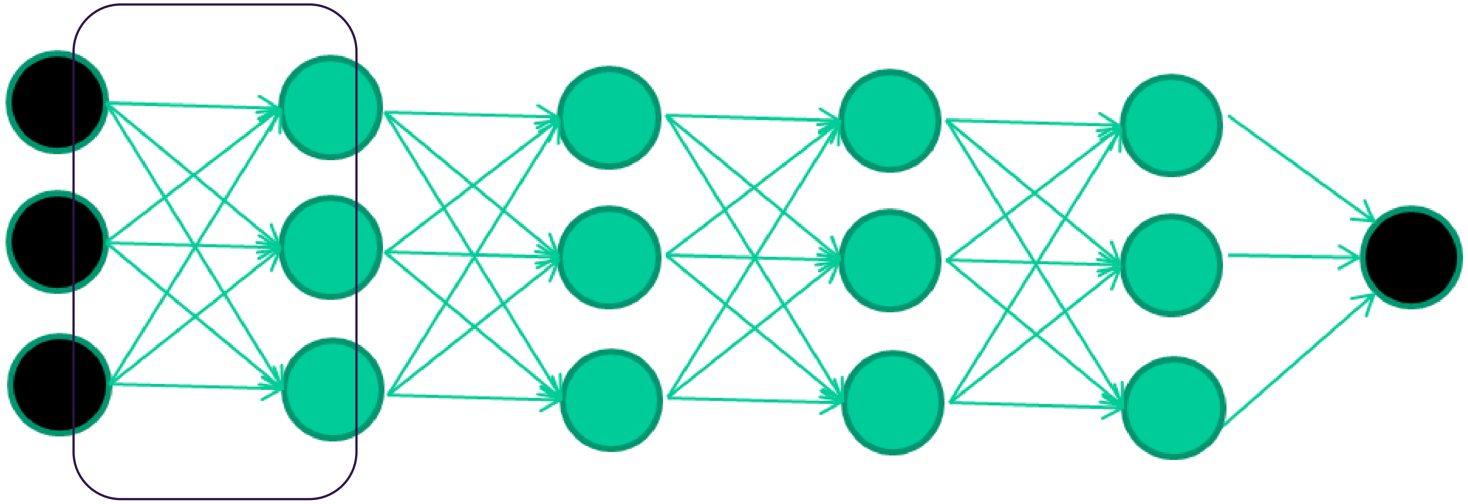
Many-layer neural network architectures should be capable of learning the true underlying features and 'feature logic', and therefore generalise very well …



The New Way To Train Multi-Layer NNs…



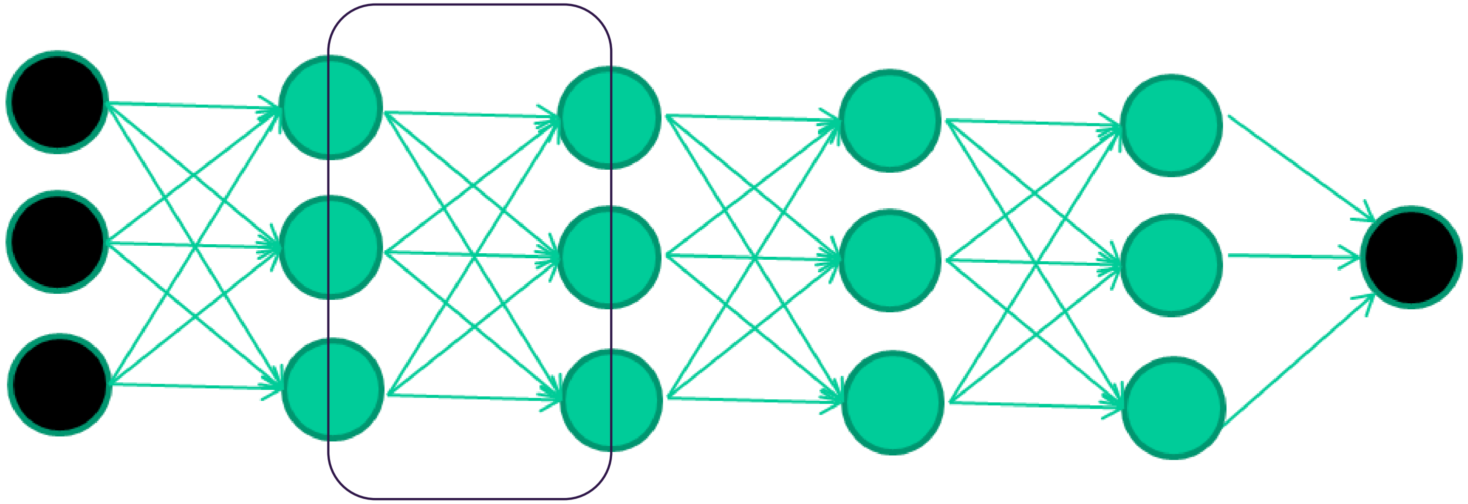
The New Way To Train Multi-Layer NNs…



Train this layer first

Microsoft

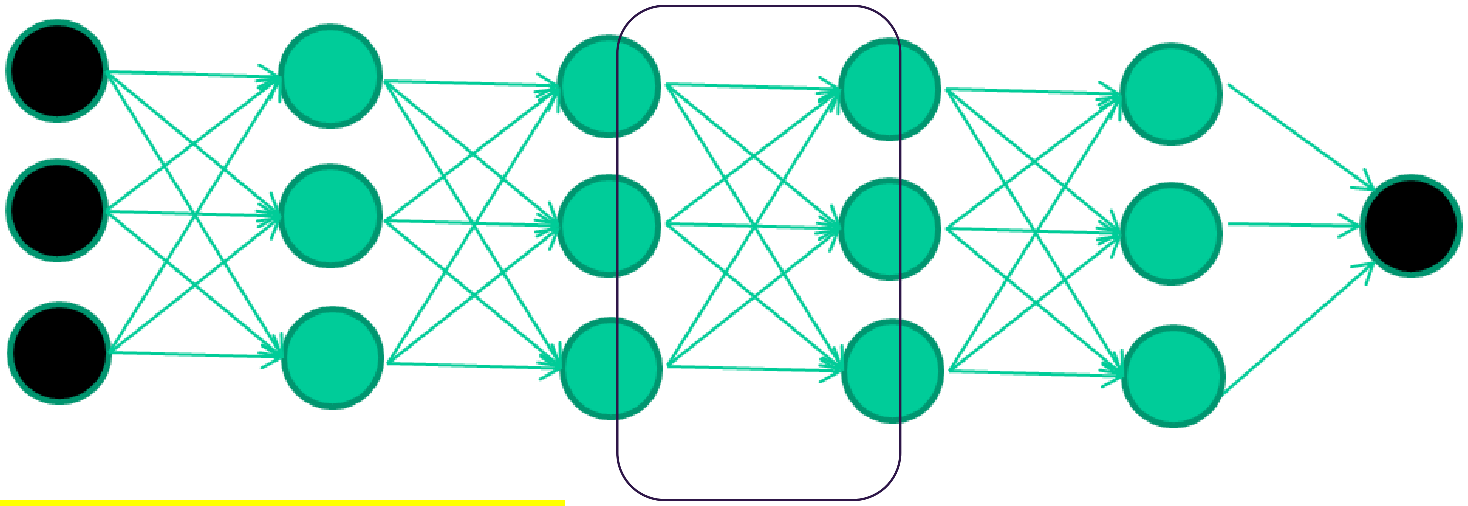
The New Way To Train Multi-Layer NNs…



Train this layer first

then this layer

The New Way To Train Multi-Layer NNs…

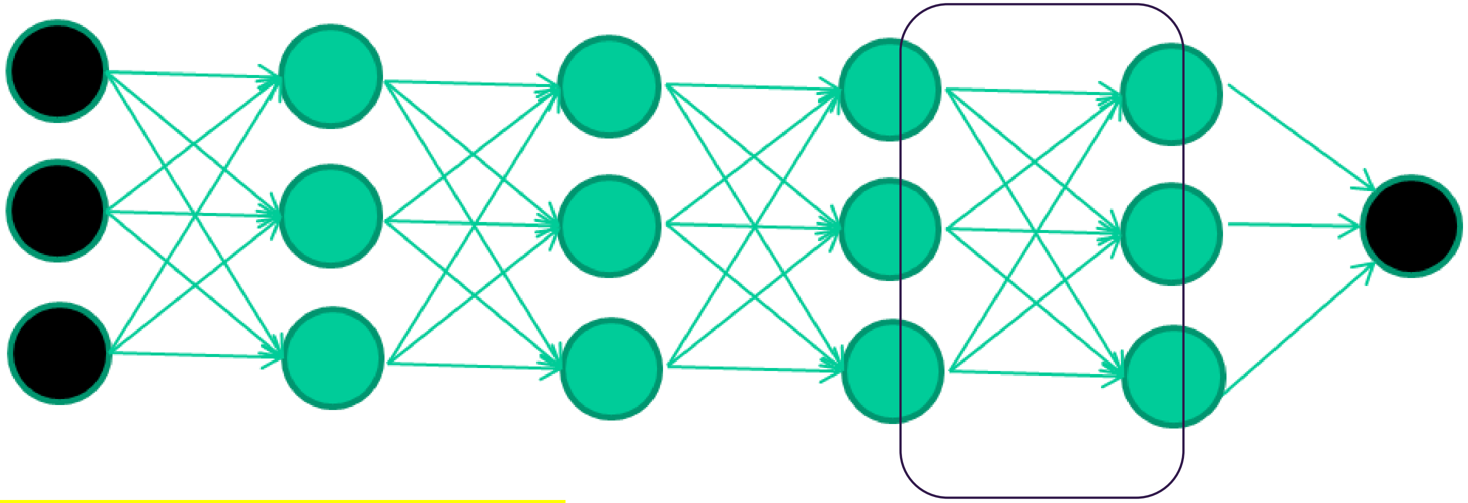


Train this layer first

then this layer

then this layer

The New Way To Train Multi-Layer NNs…



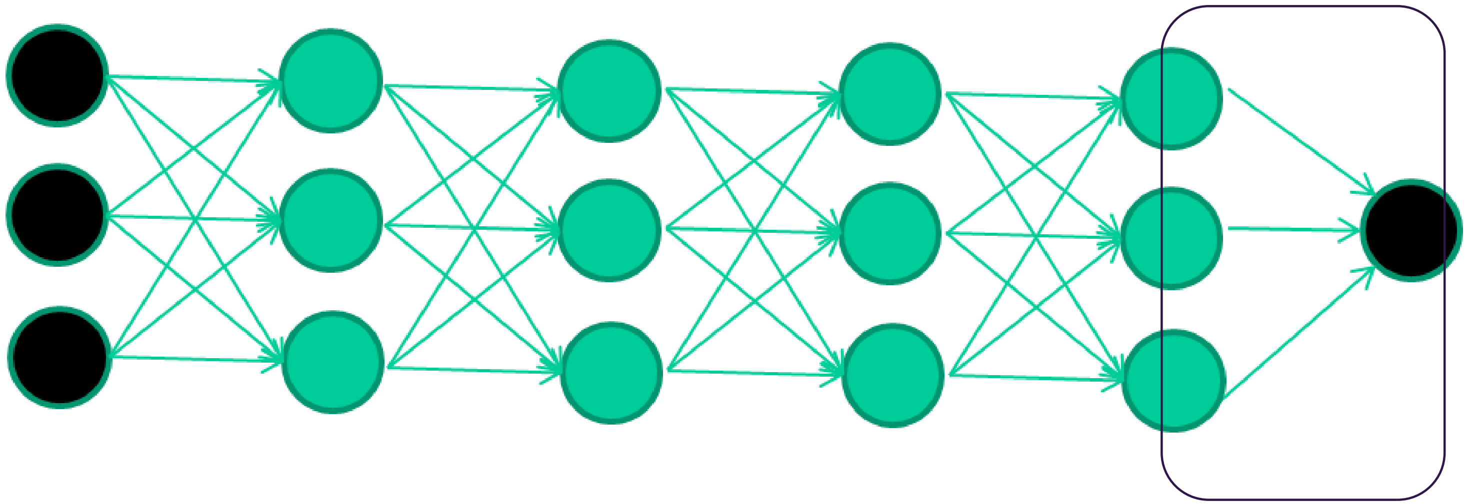
Train this layer first

then this layer

then this layer

then this layer

The New Way To Train Multi-Layer NNs…



Train this layer first

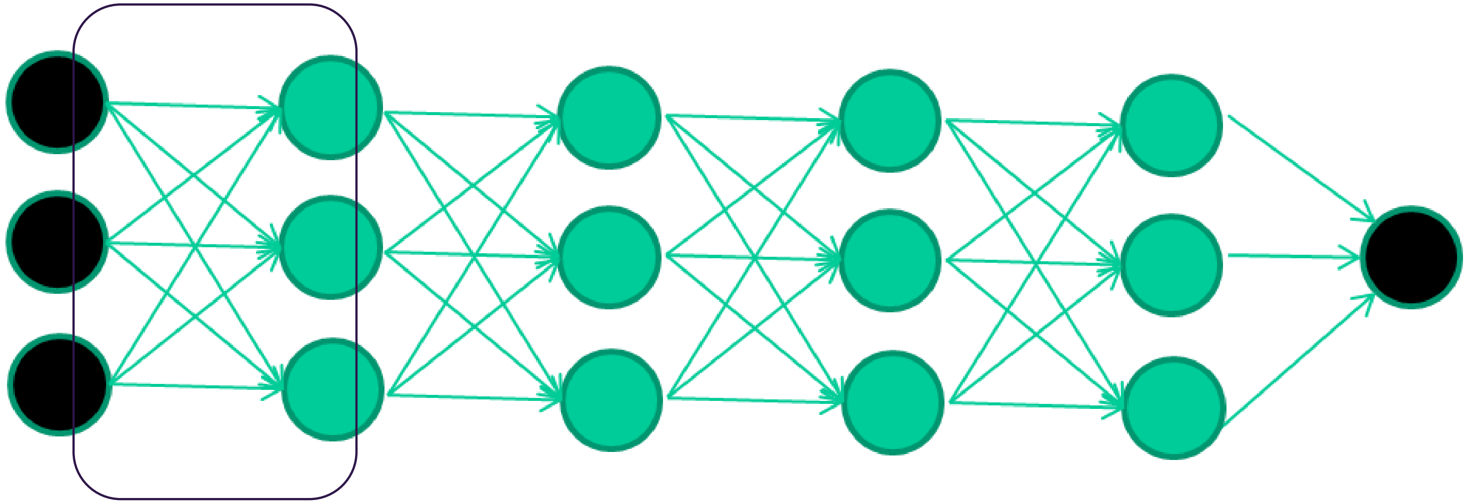
then this layer

then this layer

then this layer

finally this layer

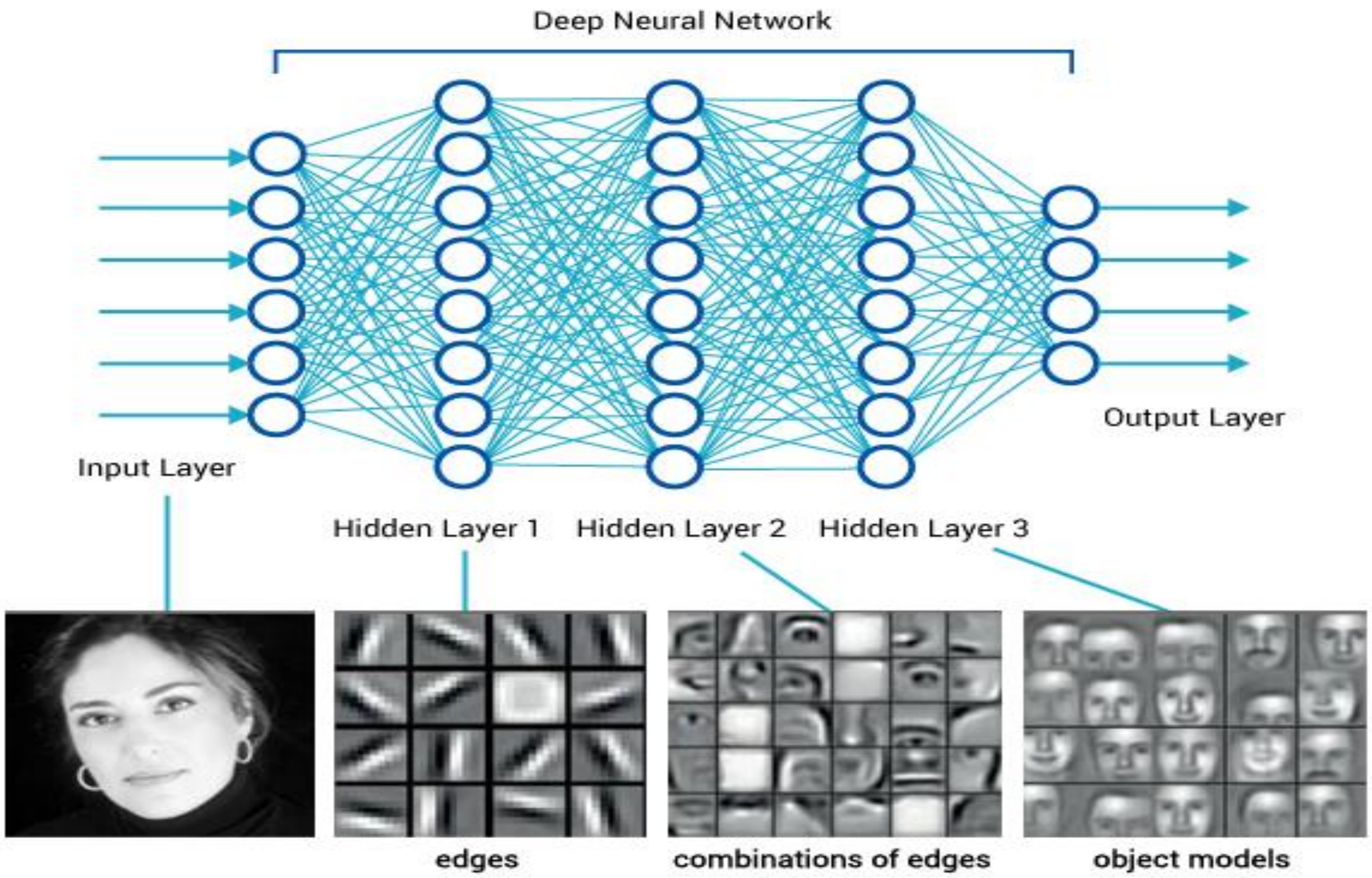
The New Way To Train Multi-Layer NNs…



Each layer can be thought of as a set of features

Idea Behind Deep Learning

- There are many types of deep learning
- Different kinds of autoencoder, variations on architectures and training algorithms, etc.
- It's a growing area



Microsoft

Common DNNs

- ❑ Deep Convolutional Neural Network (DCNN)
  - To extract representation from images
- ❑ Recurrent Neural Network (RNN)
  - To extract representation from sequential data
- ❑ Deep Belief Neural Network (DBN)
  - To extract hierarchical representation from a dataset
- ❑ Deep Reinforcement Learning (DQN)
  - To prescribe how agents should act in an environment in order to maximize future cumulative reward (e.g., a game score)

We will cover DCNN today

Microsoft

Open Source Deep Learning Frameworks

DL4J
- JVM-based
- Distrubted
- Integrates with Hadoop and Spark

Theano
- Very popular in Academia
- Fairly low level
- Interfaced with via Python and Numpy

Torch
- Lua based
- In house versions used by Facebook and Twitter
- Contains pretrained models

Microsoft

Open Source Deep Learning Frameworks

TensorFlow

- Google written successor to Theano
- Interfaced with via Python and Numpy
- Highly parallel
- Can be somewhat slow for certain problem sets

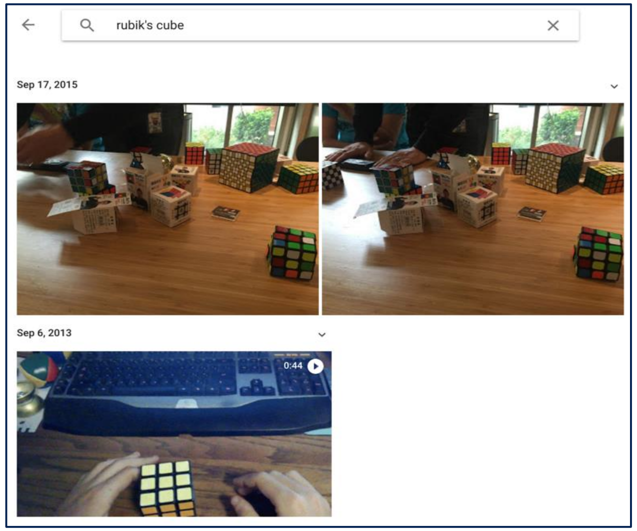
Caffe

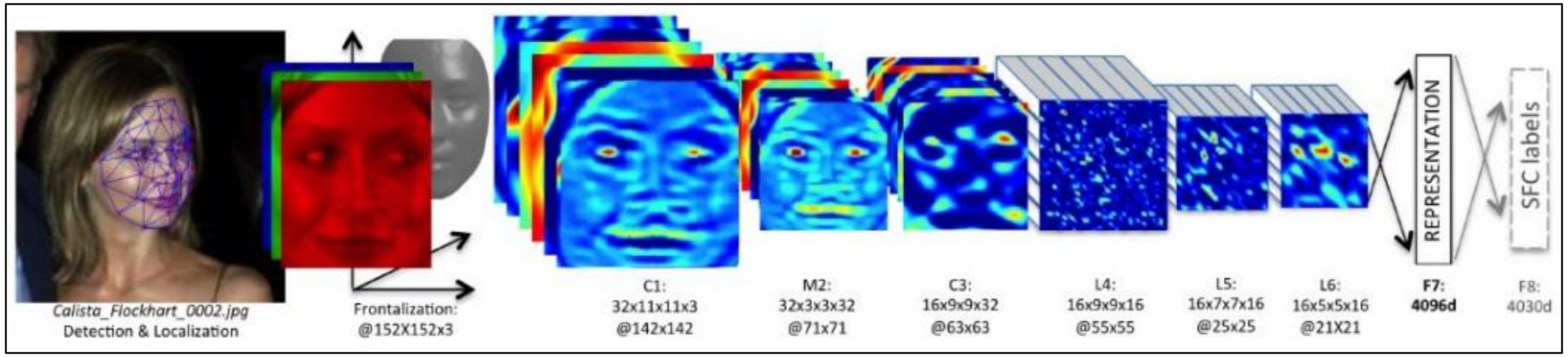
- Not general purpose. Focuses on machine-vision problems
- Implemented in C++ and is very fast
- Not easily extensible
- Has a Python interface

Microsoft

Deep Learning And Computer Vision

ConvNet





Face Verification, Taigman et al. 2014 (FAIR)

e.g. Google Photos search

[Goodfellow et al. 2014]

Self-driving cars

Image Classification

- Task of taking an input image and outputting a class
- Probability of classes that best describes the image
- For humans, effortless task

What We See

08	02	22	97	38	15	00	40	00	75	04	05	07	78	52	12	50	77	91	08
49	49	99	40	17	81	18	57	60	87	17	40	98	43	69	48	04	56	62	00
81	49	31	73	55	79	14	29	93	71	40	67	53	88	30	03	49	13	36	65
52	70	95	23	04	60	11	42	69	24	68	56	01	32	56	71	37	02	36	91
22	31	16	71	51	67	63	89	41	92	36	54	22	40	40	28	66	33	13	80
24	47	32	60	99	03	45	02	44	75	33	53	78	36	84	20	35	17	12	50
32	98	81	28	64	23	67	10	26	38	40	67	59	54	70	66	18	38	64	70
67	26	20	68	02	62	12	20	95	63	94	39	63	08	40	91	66	49	94	21
24	55	58	05	66	73	99	26	97	17	78	78	96	83	14	88	34	89	63	72
21	36	23	09	75	00	76	44	20	45	35	14	00	61	33	97	34	31	33	95
78	17	53	28	22	75	31	67	15	94	03	80	04	62	16	14	09	53	56	92
16	39	05	42	96	35	31	47	55	58	88	24	00	17	54	24	36	29	85	57
86	56	00	48	35	71	89	07	05	44	44	37	44	60	21	58	51	54	17	58
19	80	81	68	05	94	47	69	28	73	92	13	86	52	17	77	04	89	55	40
04	52	08	83	97	35	99	16	07	97	57	32	16	26	26	79	33	27	98	66
88	36	68	87	57	62	20	72	03	46	33	67	46	55	12	32	63	93	53	69
04	42	16	73	38	25	39	11	24	94	72	18	08	46	29	32	40	62	76	36
20	69	36	41	72	30	23	88	34	62	99	69	82	67	59	85	74	04	36	16
20	73	35	29	78	31	90	01	74	31	49	71	48	86	81	16	23	57	05	54
01	70	54	71	83	51	54	69	16	92	33	48	61	43	52	01	89	19	67	48

What Computers See

Microsoft

Input Image

- An image is an an array of pixel values
- A JPG color image with size 480 x 480:
  - The representative array will be 480 x 480 x 3. Each number is given a value from 0 to 255 which is the pixel intensity
- Grey scale image contains a single sample (intensity value) for each pixel
- Image Classification:
  - Given an array of numbers, produce probabilities of the image being a certain class

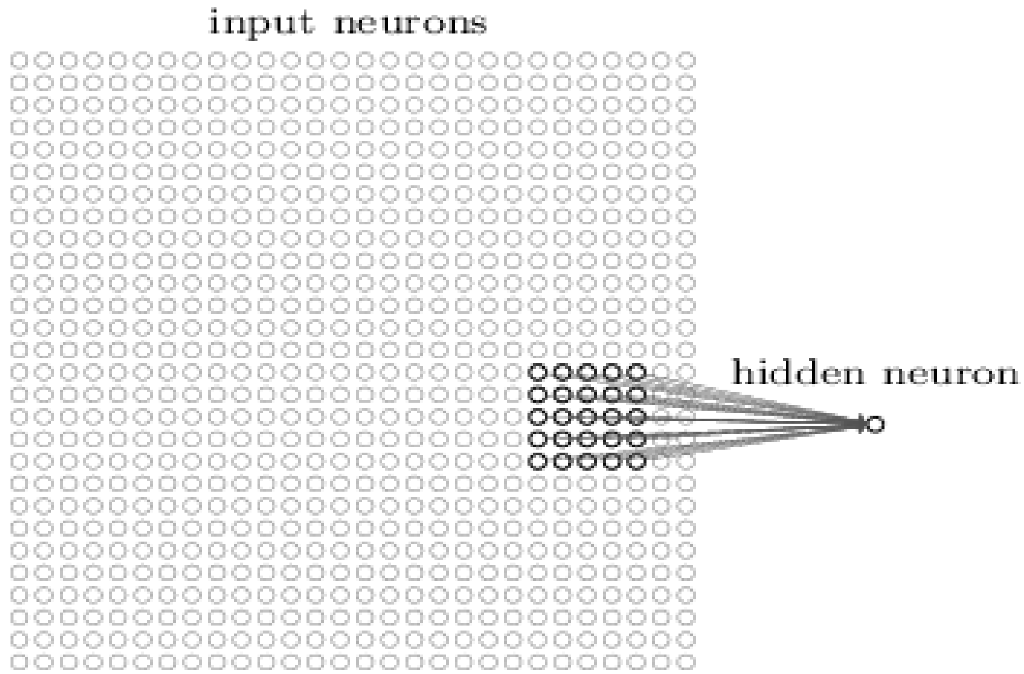
Convolutional Neural Networks

Three basic ideas:

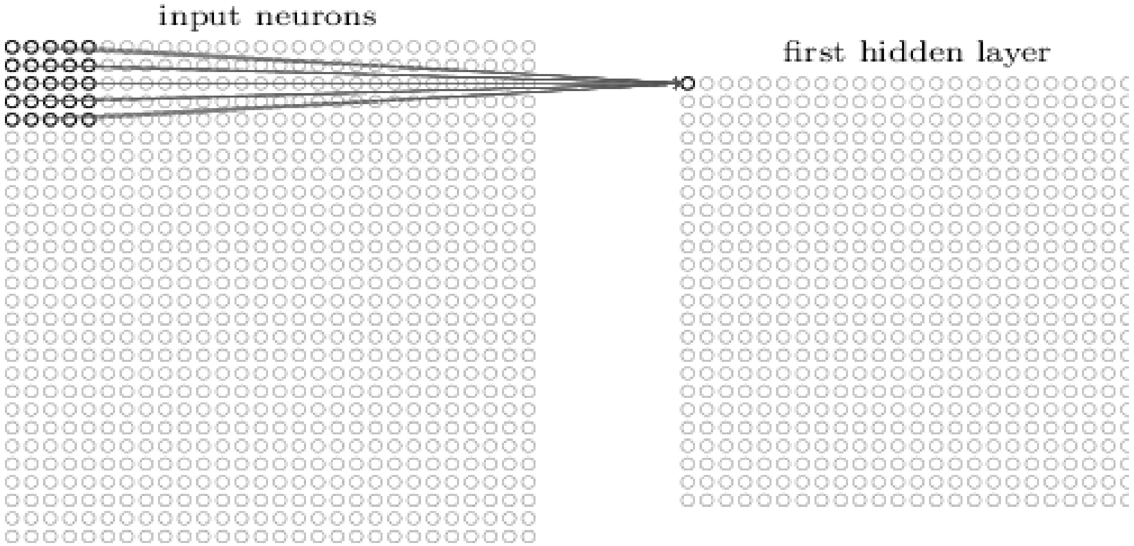
- Local receptive fields
- Shared weights
- Pooling

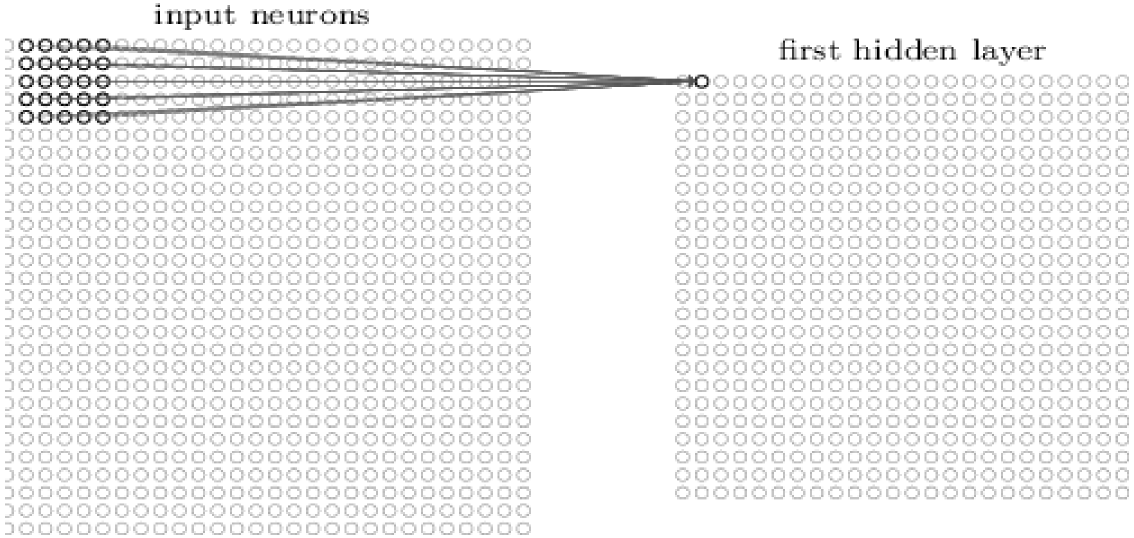
Local Receptive Fields

- Connections are from small, localized regions of the input image to hidden layers
- A little window on the input pixels
- Each neuron in the first hidden layer is connected to a small region of the input neurons. For example, a 5x5 region



Local Receptive Fields





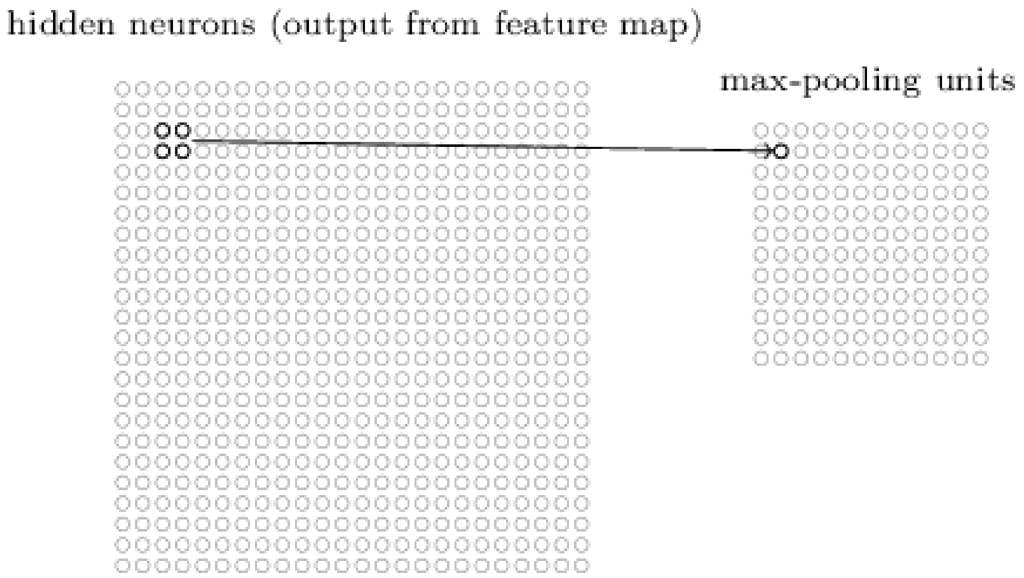
Microsoft

Pooling Layers

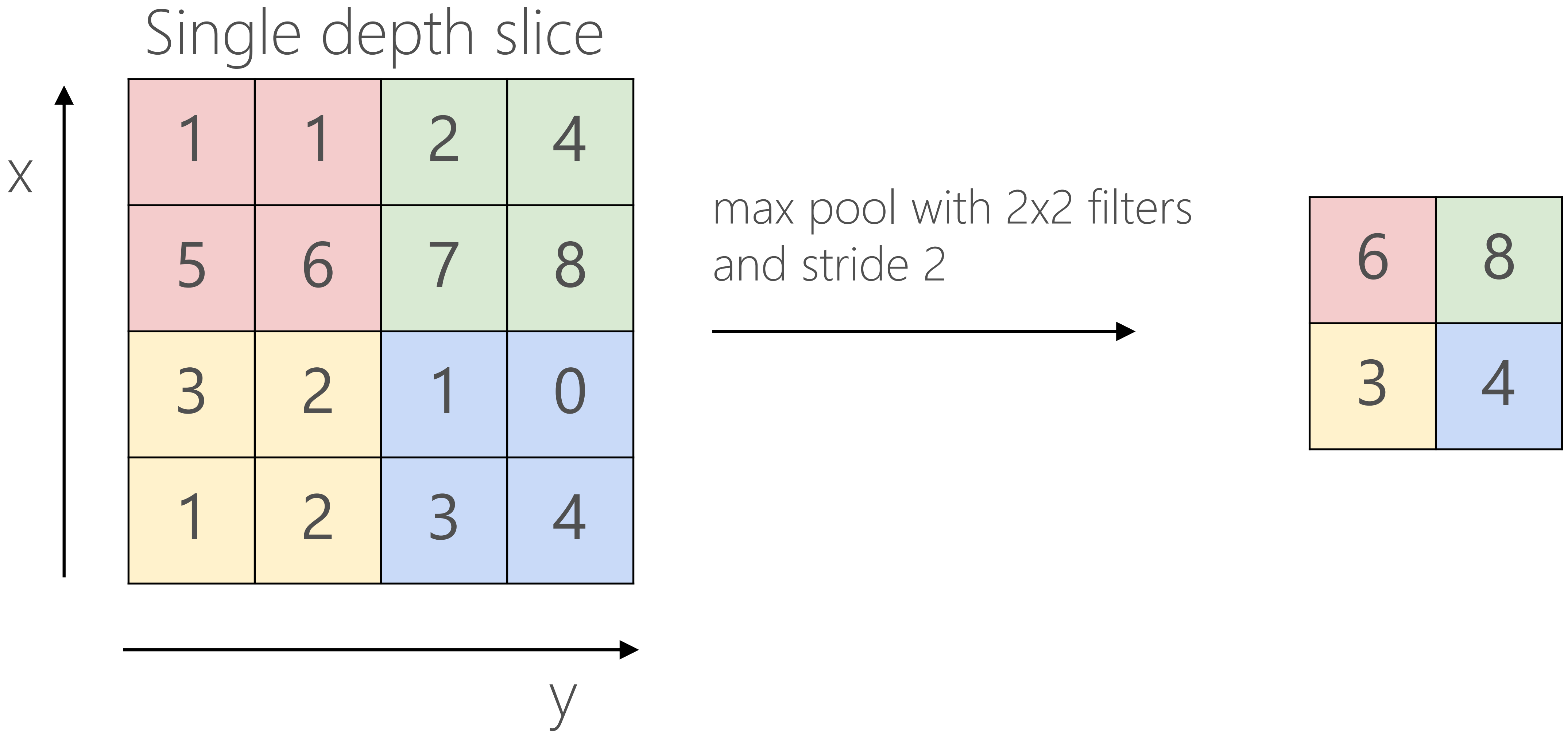
Often used immediately after convolutional layers

- Simplify the information in the output from the convolutional layer
- Takes each feature map output from the convolutional layer and prepares a condensed feature map
- Max-pooling:

  A pooling unit simply outputs the maximum activation in the $p \times p$ region

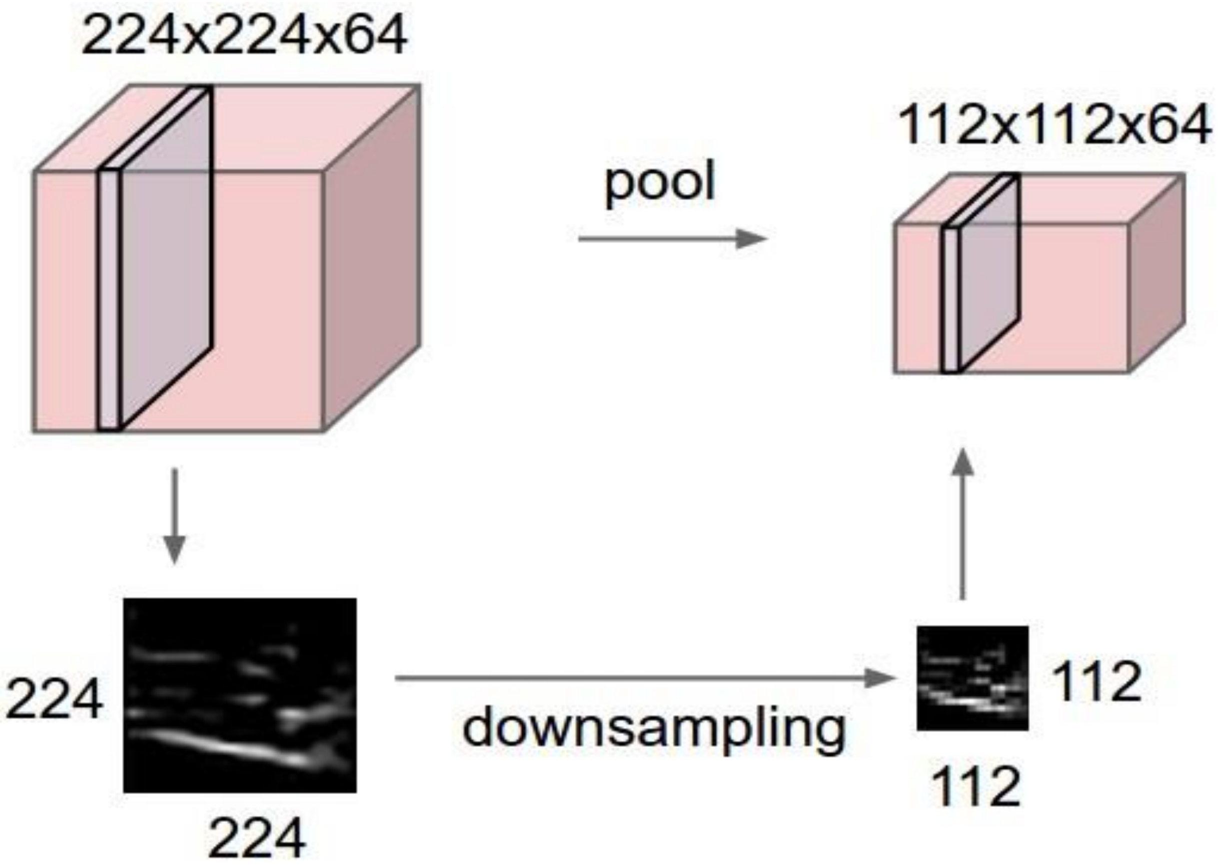


Max Pooling


Single depth slice
x
1 1 2 4
5 6 7 8
3 2 1 0
1 2 3 4
y
max pool with 2x2 filters
and stride 2
6 8
3 4

Pooling Layers

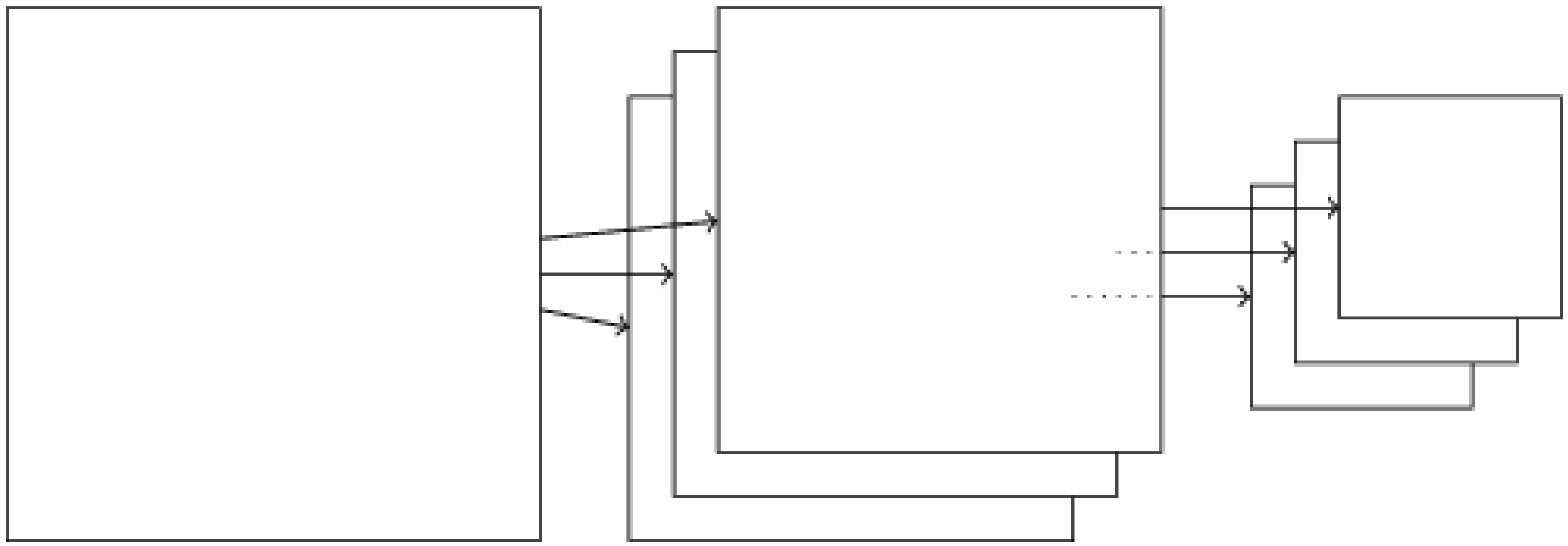
- Smaller representations and more manageable
- Operates over each activation map independently



Microsoft

Pooling Layers

Combined convolutional and max-pooling layers:



Shared Weights And Biases

1. Each hidden neuron has a bias and pxp weights connected to its local receptive field
2. The same weights and bias for each of the hidden neurons
3. In other words, for the $j,\ k^{th}$ hidden neuron, the output is:

$$\sigma(b + \sum_{l=0}^{n} \sum_{m=0}^{n} w_{l,m} a_{j+l,k+m})$$

where σ is the neural activation function - perhaps the sigmoid function
b is the shared value for the bias
$w_{l,m}$ is a $n \times n$ array of shared weights
$a_{x,y}$to denote the input activation at position x, y

Shared Weights And Biases

- Convolutional networks are well adapted to the translation invariance of images
- Greatly reduces the number of parameters involved in a convolutional network $(p \times p + b)$

Terminology

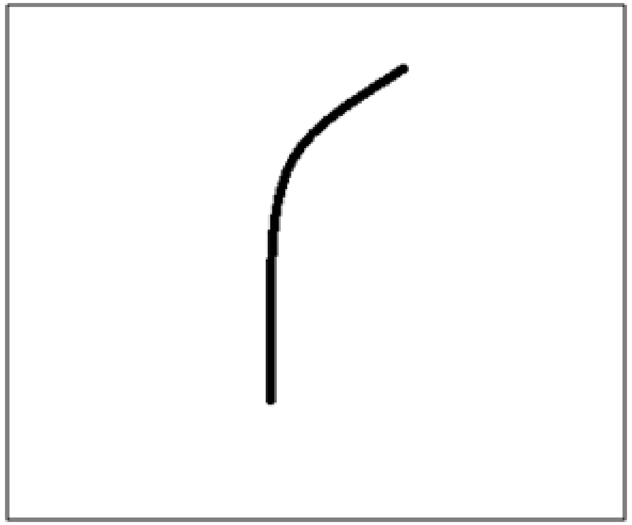
Feature map/Activation map	Map from the input layer to the hidden layer
Shared weights	Weights defining the feature map
Shared bias	Bias defining the feature map
Kernel/Filter	Shared weights and bias

First Layer – High Level Perspective

- Filters can be thought of as feature identifiers (straight edges, simple colors, and curves)
- In the simple case of a one filter convolution and a curve detector filter, activation map results in regions that are most likely curves in the picture

0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

Pixel representation of filter



Visualization of a curve detector filter

- More filters mean greater the depth of the activation map
- This results in more information about the input

Going Deeper Through The Network

- Many layers are interspersed between convolution layers (example: ReLu and Dropout)
- Introduction of nonlinearities
- Improve the robustness of the network and control overfitting

Input -> Conv -> ReLU -> Conv -> ReLU -> Pool -> ReLU -> Conv -> ReLU -> Pool ->Fully Connected

Microsoft

RelU

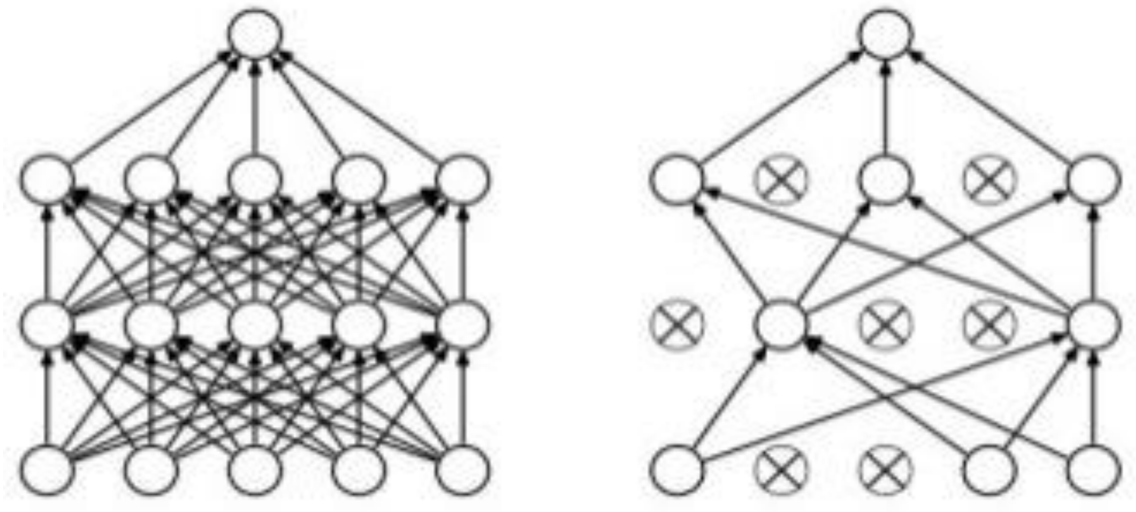
- The Rectified Linear Unit has become very popular recently

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

- Activation is simply thresholded at zero
- It was found to greatly accelerate (Krizhevsky et al.) the convergence of stochastic gradient descent compared to the sigmoid/tanh functions
- Compared to tanh/sigmoid neurons that involve expensive operations (exponentials, etc.), the ReLU is simply thresholding

Dropout

- A form of ensemble learning
- Avoids overfitting (by preventing inter-dependencies from emerging between nodes)
- Dropout - an extreme version of bagging
- At each training step, the dropout procedure creates a different network by removing some neurons randomly



Fully Connected Layer (FC layer)