HALMSTAD UNIVERSITY

Learning Systems (DT8008)

Artificial Neural Networks

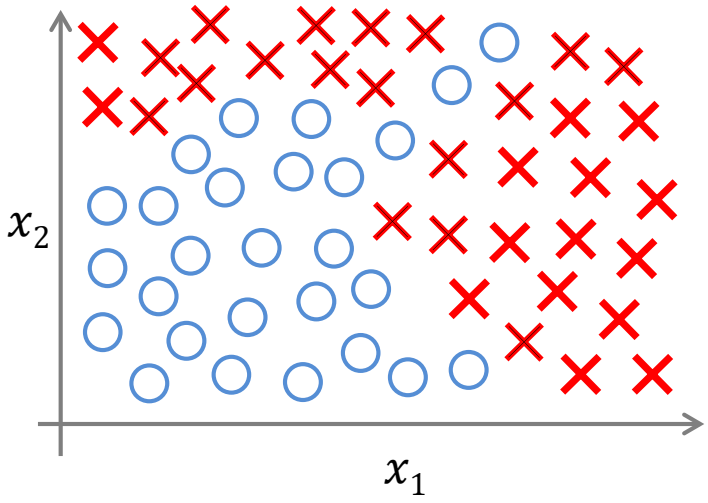
Dr. Mohamed-Rafik Bouguelia
mohamed-rafik.bouguelia@hh.se

Halmstad University

Nonlinear hypotheses (Motivation)

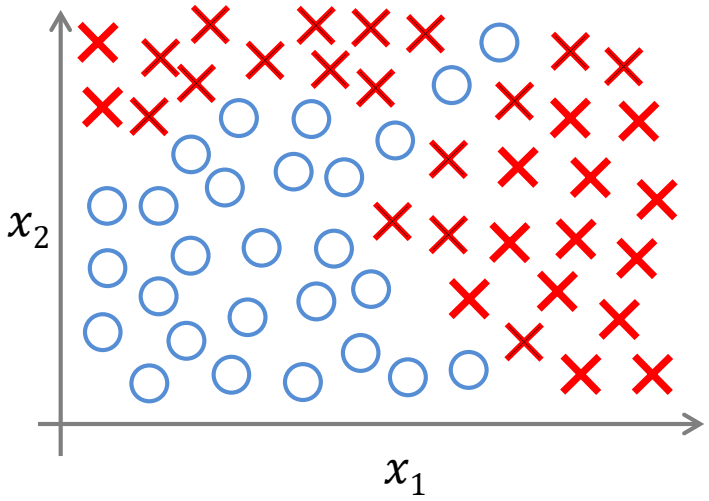
Nonlinear classification

- Consider a supervised learning classification problem.
  - Suppose that you want to apply logistic regression to this dataset.



Nonlinear classification

- Consider a supervised learning classification problem.
  - Suppose that you want to apply logistic regression to this dataset.

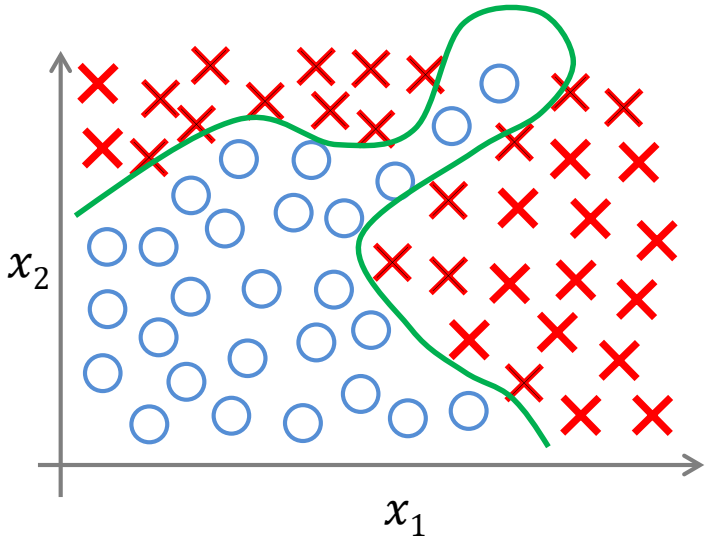


- One thing you could do is to apply logistic regression with a lot of nonlinear features, e.g :

$$g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1 x_2 + \theta_4 x_1^2 x_2 + \theta_5 x_1^3 x_2 + \theta_6 x_1 x_2^2 + \dots)$$

Nonlinear classification

- Consider a supervised learning classification problem.
  - Suppose that you want to apply logistic regression to this dataset.



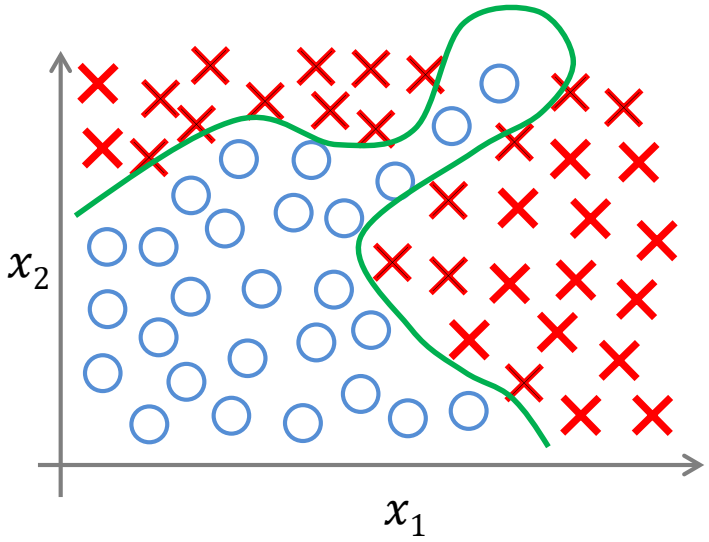
- One thing you could do is to apply logistic regression with a lot of nonlinear features, e.g :

$$g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1 x_2 + \theta_4 x_1^2 x_2 + \theta_5 x_1^3 x_2 + \theta_6 x_1 x_2^2 + \dots)$$

- If you include enough polynomial terms, then maybe you get a hypothesis that separates the positive and negative examples.

Nonlinear classification

- Consider a supervised learning classification problem.
  - Suppose that you want to apply logistic regression to this dataset.



- One thing you could do is to apply logistic regression with a lot of nonlinear features, e.g :

$$g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1 x_2 + \theta_4 x_1^2 x_2 + \theta_5 x_1^3 x_2 + \theta_6 x_1 x_2^2 + \dots)$$

- If you include enough polynomial terms, then maybe you can a hypothesis that separates the positive and negative examples.

- But what if you had much more than two features in your dataset:

$x_1 =$ size
$x_2 =$ # bedrooms
$x_3 =$ # floors
$x_4 =$ age
$\cdots$
x_{100}

Just considering 2nd order polynomial terms, there will be a lot of them (in the order of $O(d^2)$) :
$x_1^2, x_1 x_2, x_1 x_3, \dots, x_1 x_{100}, x_2^2, x_2 x_3, \dots, x_3^2, \dots, x_{100}^2$ ...
Overfitting and **computationally expensive**.

You can include a subset of these polynomial terms (e.g. $x_1^2, x_2^2, \dots, x_{100}^2$), but maybe not enough to fit a complex dataset (i.e. to have a sufficiently complex decision boundary) …

Example of a high number of features d

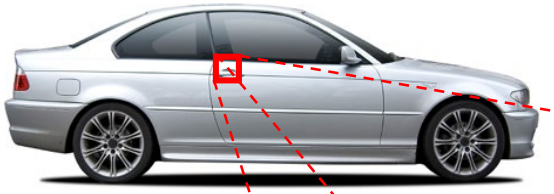
- For many real world problems, the original number of features d will be pretty large.

Example: what is this?

Example of a high number of features d

- For many real world problems, the original number of features d will be pretty large.

Example: what is this?



But the camera sees this:

194	210	201	212	199	213	215	195	178	158	182	209
180	189	190	221	209	205	191	167	147	115	129	163
114	126	140	188	176	165	152	140	170	106	78	88
87	103	115	154	143	142	149	153	173	101	57	57
102	112	106	131	122	138	152	147	128	84	58	66
94	95	79	104	105	124	129	113	107	87	69	67
68	71	69	98	89	92	98	95	89	88	76	67
41	56	68	99	63	45	60	82	58	76	75	65
20	43	69	75	56	41	51	73	55	70	63	44
50	50	57	69	75	75	73	74	53	68	59	37
72	59	53	66	84	92	84	74	57	72	63	42
67	61	58	65	75	78	76	73	59	75	69	50

Based on these pixel values, you want the ML model to tell you that this is a "*door handle of a car*"

Example of a high number of features d






Cars



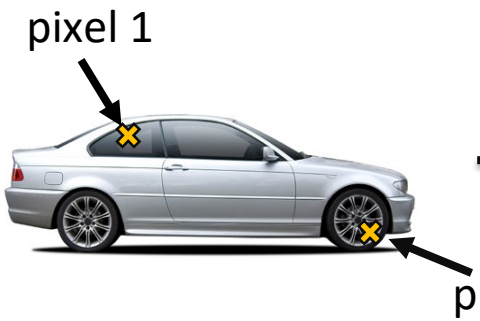



Not a car

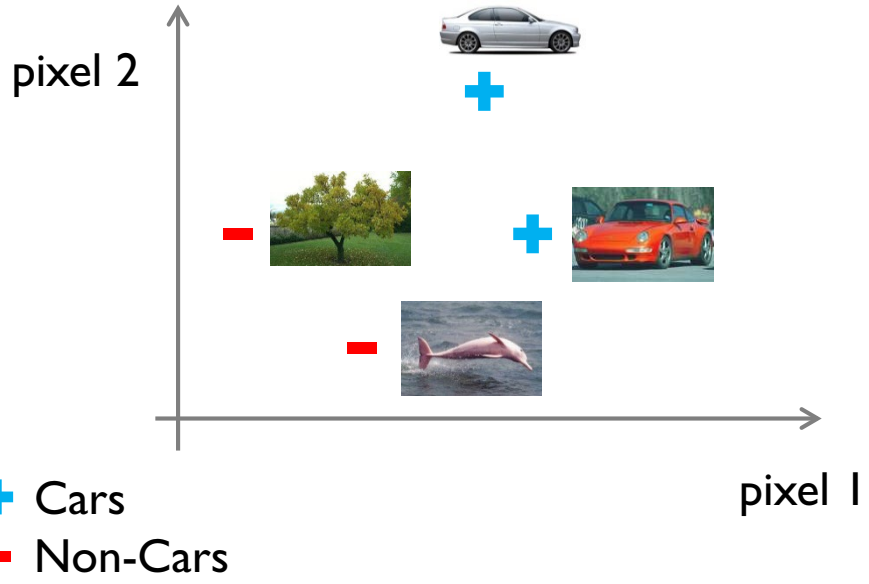
Testing:

What is this?

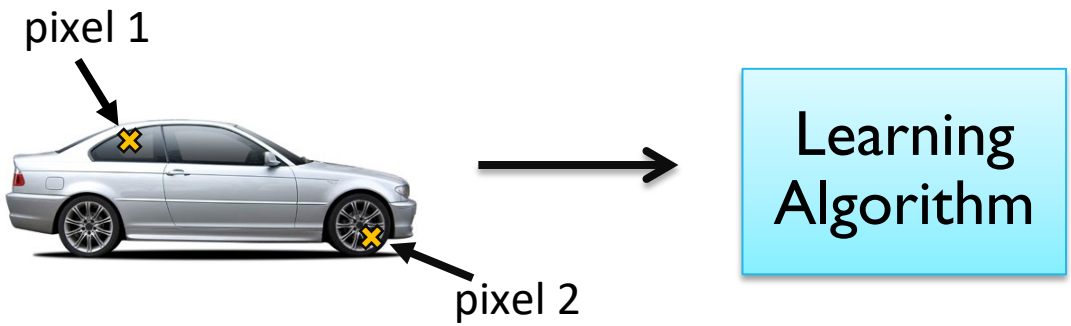
Example of a high number of features d

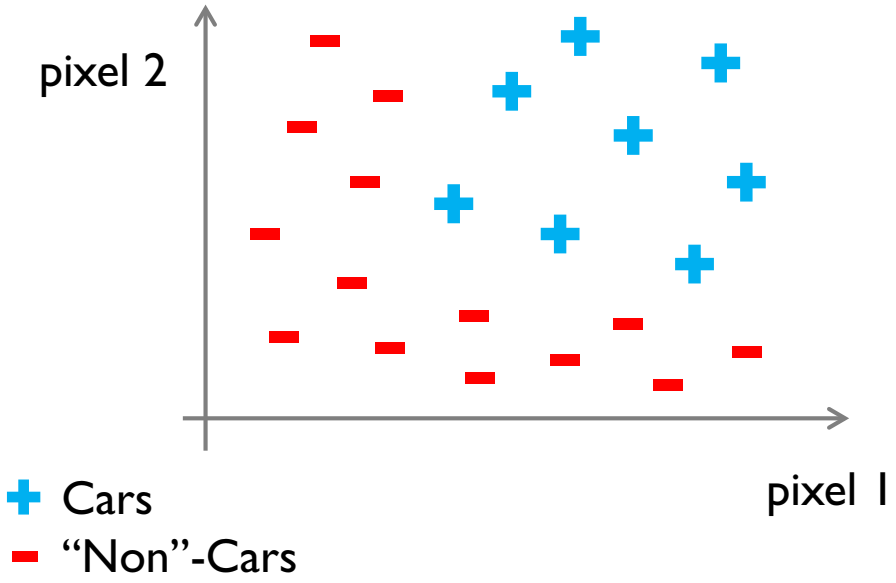


Learning Algorithm

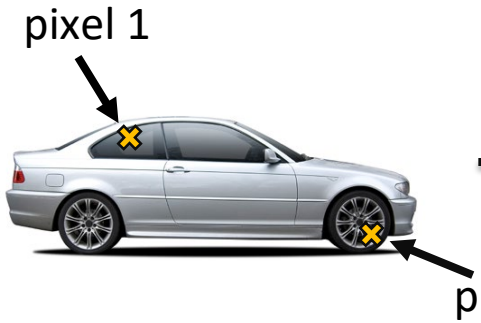


Example of a high number of features d

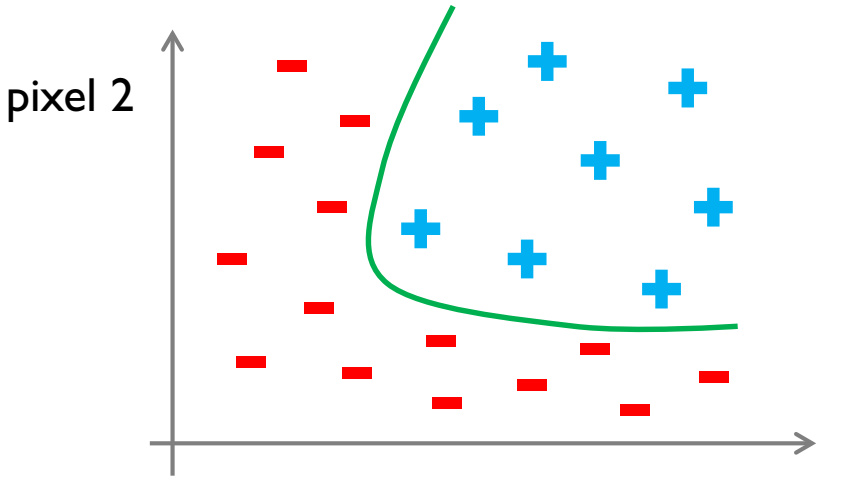




Example of a high number of features d



Learning Algorithm



+ Cars
− "Non"-Cars

50×50 pixel images→ 2500 pixels
$d = 2500$ (7500 if RGB)

$$x = \begin{bmatrix} \text{pixel 1 intensity} \\ \text{pixel 2 intensity} \\ \vdots \\ \text{pixel 2500 intensity} \end{bmatrix}$$

Quadratic features ($x_i \times x_j$): ≈3 million features

Nonlinear classification

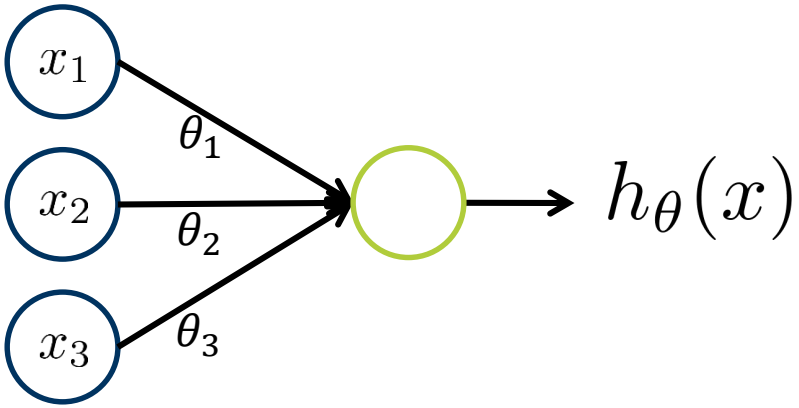
- So a simple method (e.g. logistic regression) together with adding polynomial features, is **not** a good way to learn complex nonlinear hypotheses where the original number of features d is large; as we will end un with way too many features.

- **Neural networks** turn out to be a much better way to learn complex nonlinear hypotheses, even when d is large.

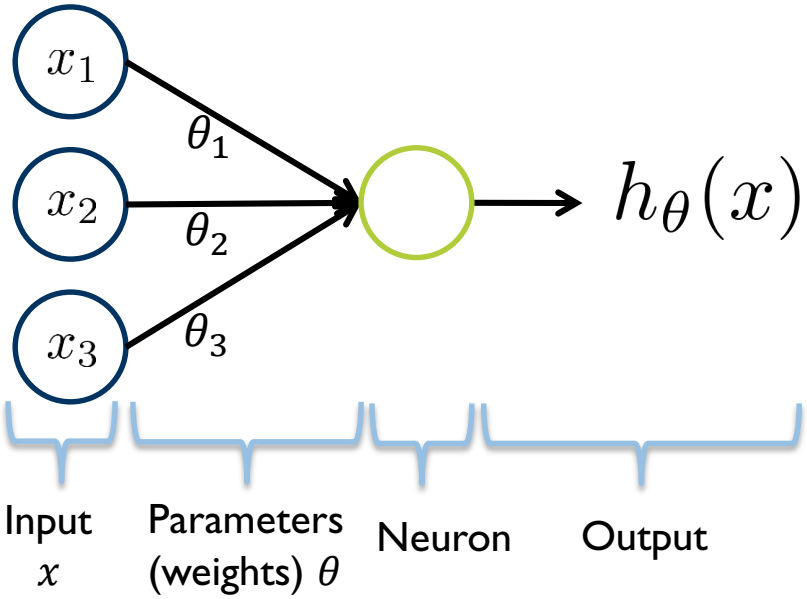
HALMSTAD UNIVERSITY

Neural Network Model Representation

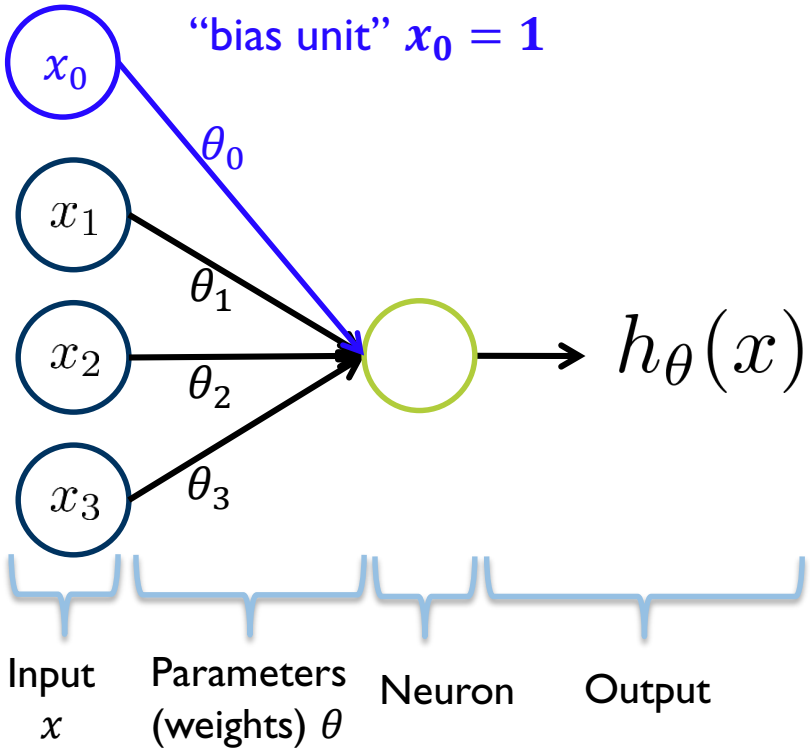
Single Neuron



Single Neuron



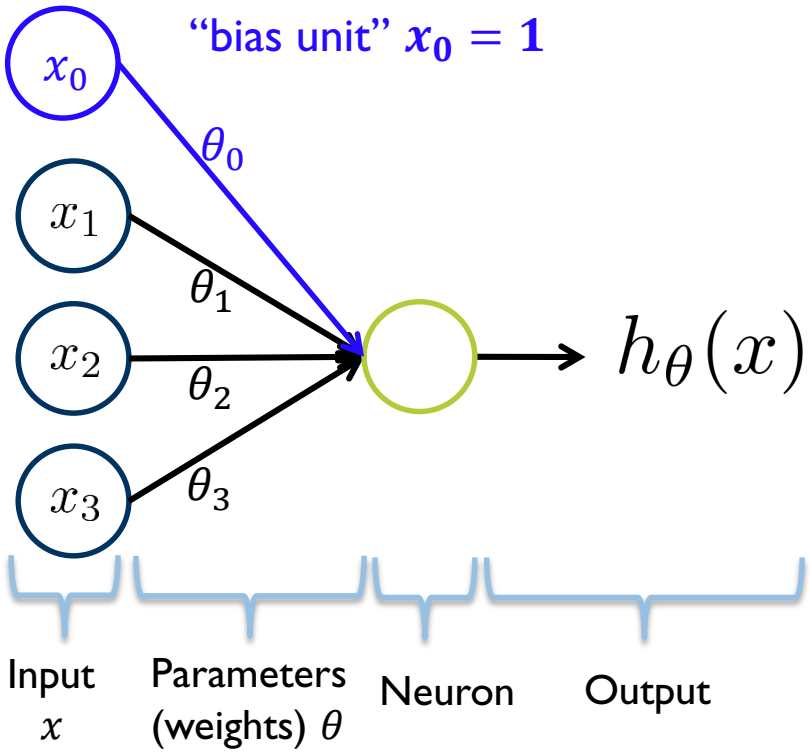
Single Neuron



$$x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \quad \theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix}$$

- The neuron applies an activation function on $\theta^T x$. So $h_\theta(x) = g(\theta^T x)$

Single Neuron



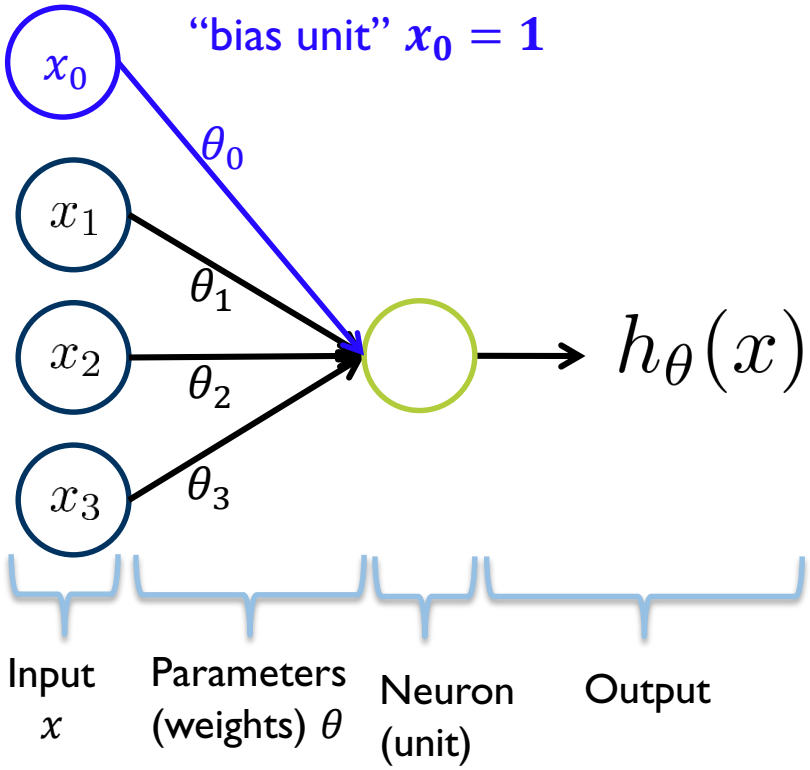
$$x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \quad \theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix}$$

- The neuron applies an activation function on $\theta^T x$. So $h_\theta(x) = g(\theta^T x)$
- If our activation function happens to be the sigmoid (logistic) function, then:

$$h_\theta(x) = \frac{1}{1 + e^{-\theta^T x}}$$

Does this remind you something?

Single Neuron



$$x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \quad \theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix}$$

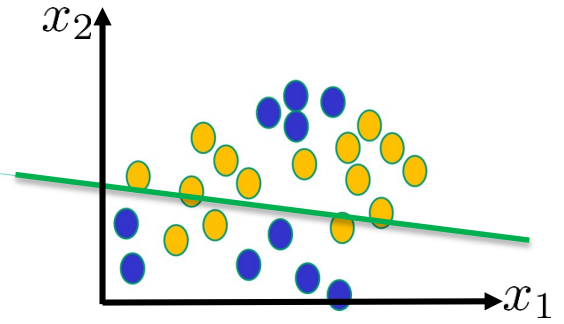
- The neuron applies an activation function on $\theta^T x$. So $h_\theta(x) = g(\theta^T x)$
- If our activation function happens to be the sigmoid (logistic) function, then:

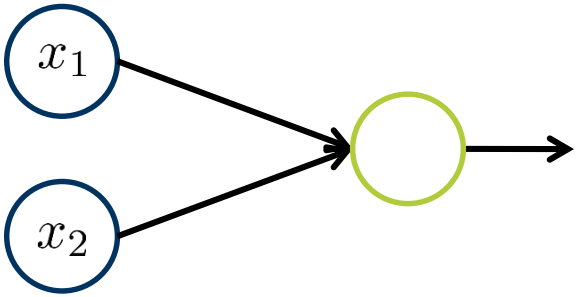
$$h_\theta(x) = \frac{1}{1 + e^{-\theta^T x}}$$

Same as a simple logistic regression (linear model).

Neural Network Representation

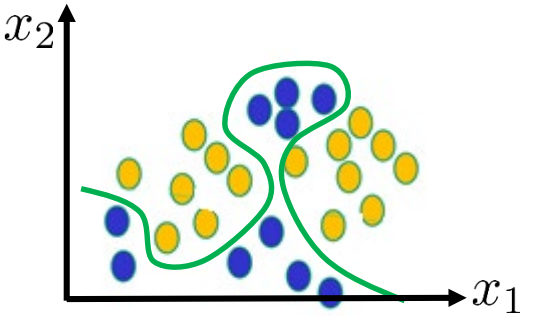
- Single neuron
  - With a single neuron, our model is only a linear model (similar to e.g. logistic regression)



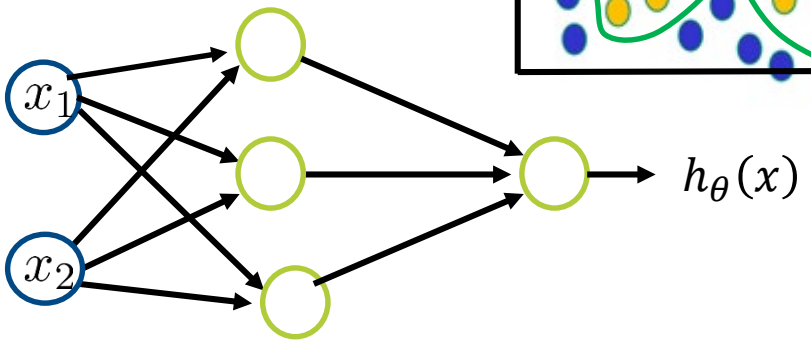


$$h_\theta(x) = \frac{1}{1 + e^{-\theta^T x}}$$

- Neural network (with one hidden layer)
  - Gives a nonlinear model.

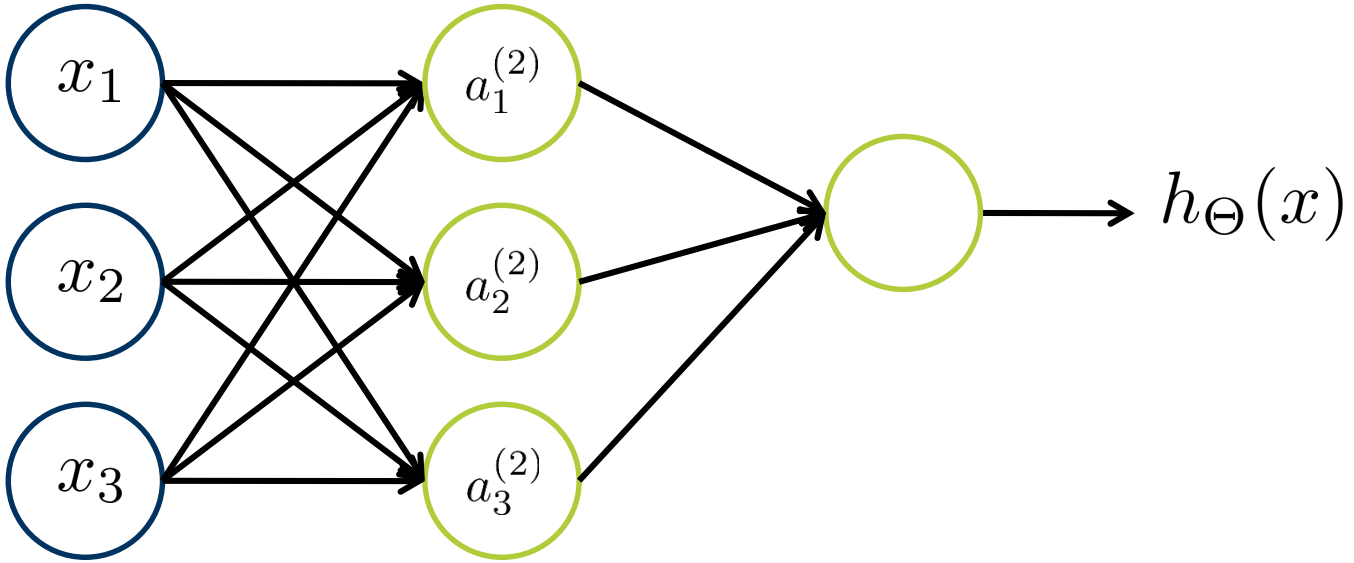


E.g. ANN with an input $x \in R^2$ (2 features) and a hidden layer with 3 neurons (units).



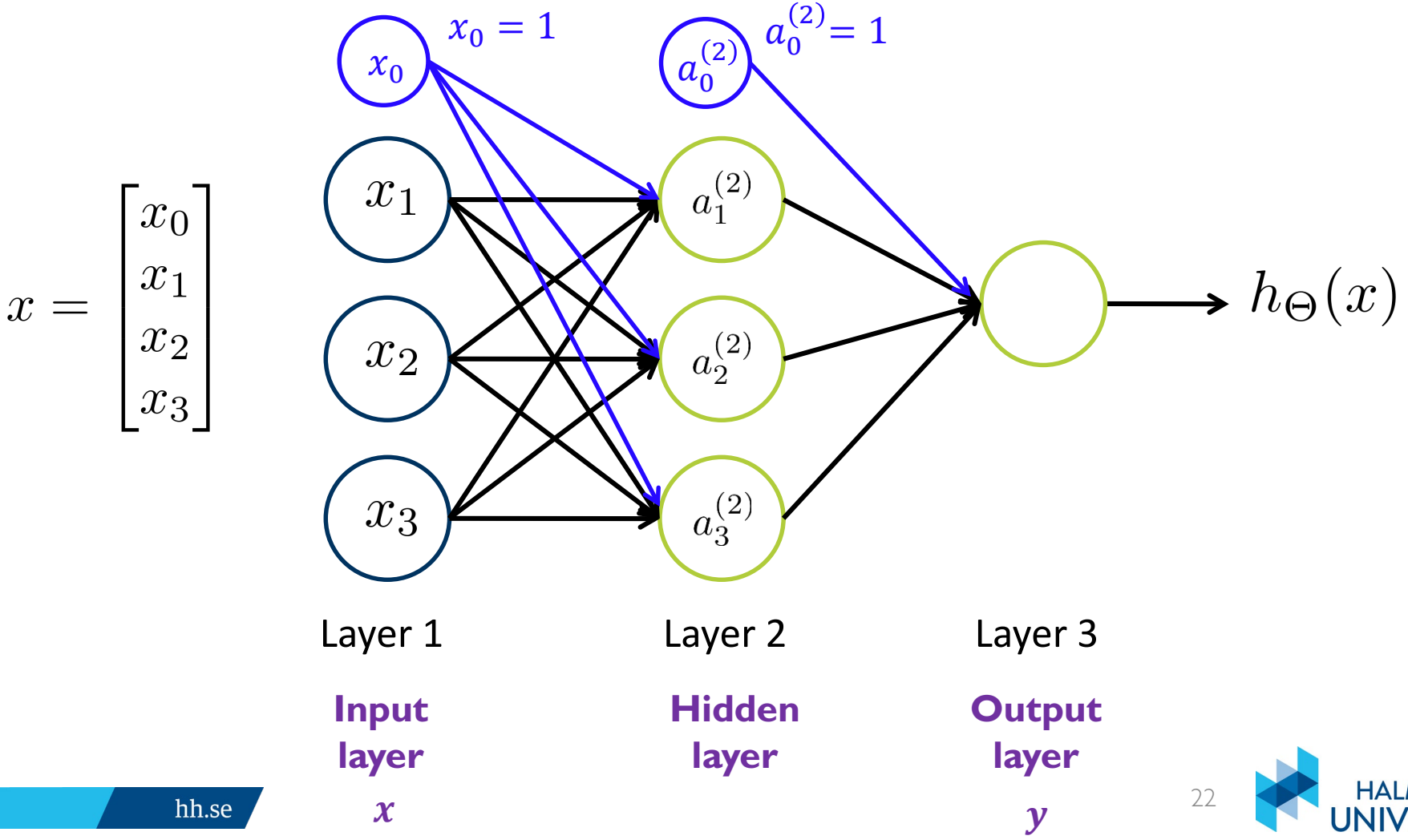
Neural Network Representation

- Neural network with one hidden layer:

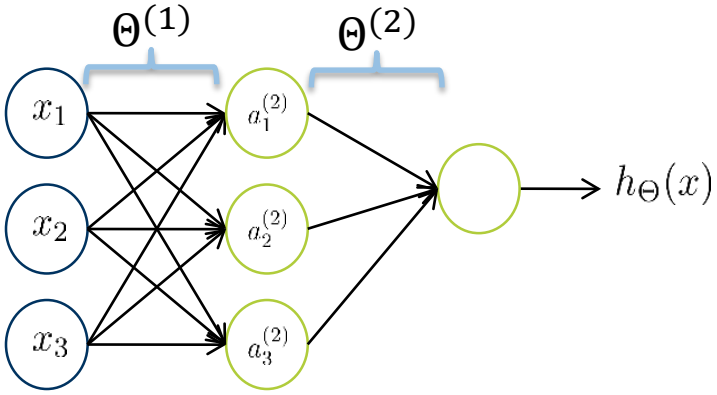


Neural Network Representation

- Neural network with one hidden layer:



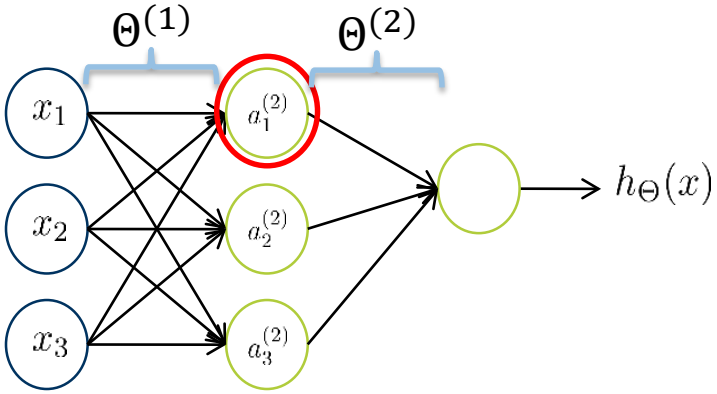
Neural Network Representation



$a_i^{(j)}$ = "activation" of unit i in layer j

$\Theta^{(j)}$ = matrix of weights (parameters) controlling function mapping from layer j to layer $j+1$

Neural Network Representation



$a_i^{(j)}$ = "activation" of unit i in layer j

$\Theta^{(j)}$ = matrix of weights (parameters) controlling function mapping from layer j to layer $j+1$

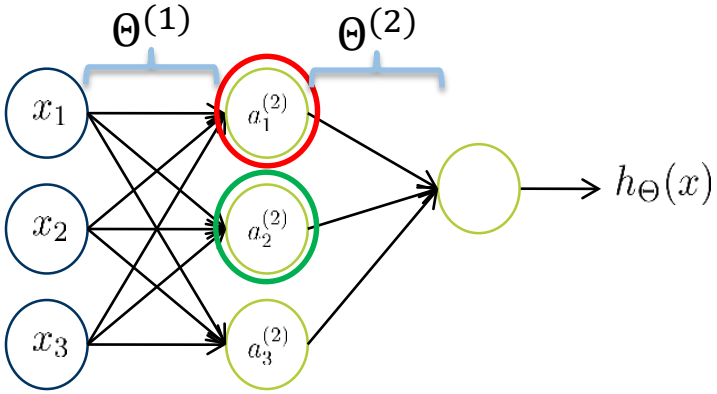
$$a_1^{(2)} = g(\Theta_{10}^{(1)} x_0 + \Theta_{11}^{(1)} x_1 + \Theta_{12}^{(1)} x_2 + \Theta_{13}^{(1)} x_3)$$

where g(.) is the activation function, e.g:

- Sigmoid: $g(z) = \frac{1}{1+e^{-z}}$
- ReLU: $g(z) = \max(0, z)$

Note: for hidden units, it recommended to not use the sigmoid (we will see later why).

Neural Network Representation



$a_i^{(j)}$ = "activation" of unit i in layer j

$\Theta^{(j)}$ = matrix of weights (parameters) controlling function mapping from layer j to layer $j+1$

$$a_1^{(2)} = g(\Theta_{10}^{(1)} x_0 + \Theta_{11}^{(1)} x_1 + \Theta_{12}^{(1)} x_2 + \Theta_{13}^{(1)} x_3)$$

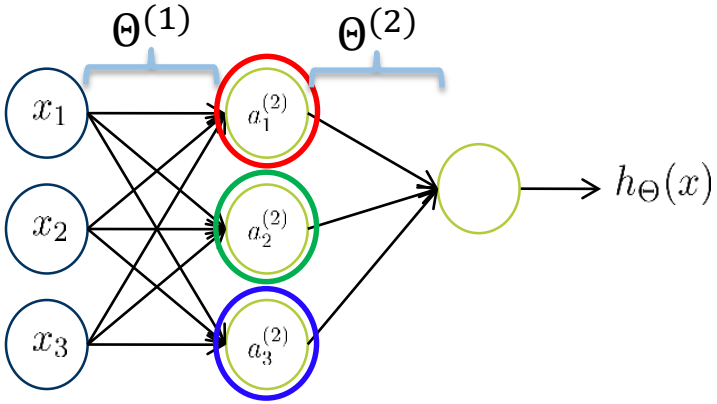
$$a_2^{(2)} = g(\Theta_{20}^{(1)} x_0 + \Theta_{21}^{(1)} x_1 + \Theta_{22}^{(1)} x_2 + \Theta_{23}^{(1)} x_3)$$

where g(.) is the activation function, e.g:

- Sigmoid: $g(z) = \frac{1}{1+e^{-z}}$
- ReLU: $g(z) = \max(0, z)$

Note: for hidden units, it recommended to not use the sigmoid (we will see later why).

Neural Network Representation



$a_i^{(j)}$ = "activation" of unit i in layer j

$\Theta^{(j)}$ = matrix of weights (parameters) controlling function mapping from layer j to layer $j+1$

$$a_1^{(2)} = g(\Theta_{10}^{(1)}x_0 + \Theta_{11}^{(1)}x_1 + \Theta_{12}^{(1)}x_2 + \Theta_{13}^{(1)}x_3)$$

$$a_2^{(2)} = g(\Theta_{20}^{(1)}x_0 + \Theta_{21}^{(1)}x_1 + \Theta_{22}^{(1)}x_2 + \Theta_{23}^{(1)}x_3)$$

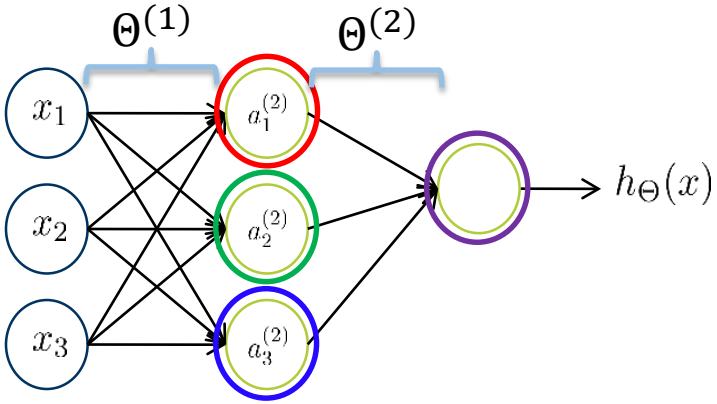
$$a_3^{(2)} = g(\Theta_{30}^{(1)}x_0 + \Theta_{31}^{(1)}x_1 + \Theta_{32}^{(1)}x_2 + \Theta_{33}^{(1)}x_3)$$

where g(.) is the activation function, e.g:

- Sigmoid: $g(z) = \frac{1}{1+e^{-z}}$
- ReLU: $g(z) = \max(0, z)$

Note: for hidden units, it recommended to not use the sigmoid (we will see later why).

Neural Network Representation



$a_i^{(j)}$ = "activation" of unit i in layer j

$\Theta^{(j)}$ = matrix of weights (parameters) controlling function mapping from layer j to layer $j+1$

$$a_1^{(2)} = g(\Theta_{10}^{(1)}x_0 + \Theta_{11}^{(1)}x_1 + \Theta_{12}^{(1)}x_2 + \Theta_{13}^{(1)}x_3)$$

$$a_2^{(2)} = g(\Theta_{20}^{(1)}x_0 + \Theta_{21}^{(1)}x_1 + \Theta_{22}^{(1)}x_2 + \Theta_{23}^{(1)}x_3)$$

$$a_3^{(2)} = g(\Theta_{30}^{(1)}x_0 + \Theta_{31}^{(1)}x_1 + \Theta_{32}^{(1)}x_2 + \Theta_{33}^{(1)}x_3)$$

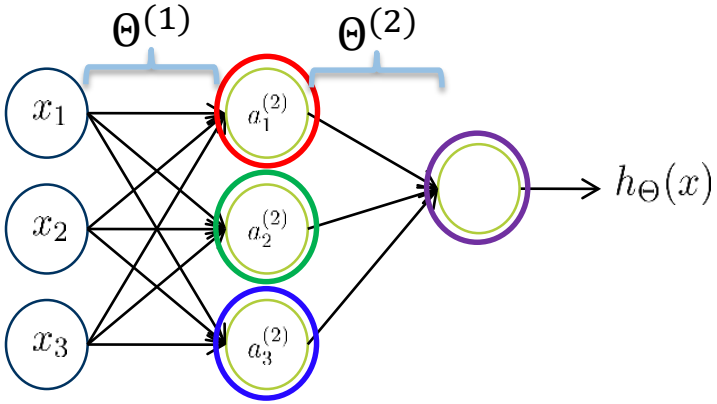
$$h_\Theta(x) = a_1^{(3)} = g(\Theta_{10}^{(2)}a_0^{(2)} + \Theta_{11}^{(2)}a_1^{(2)} + \Theta_{12}^{(2)}a_2^{(2)} + \Theta_{13}^{(2)}a_3^{(2)})$$

where g(.) is the activation function, e.g:

- Sigmoid: $g(z) = \frac{1}{1+e^{-z}}$
- ReLU: $g(z) = \max(0, z)$

Note: for hidden units, it recommended to not use the sigmoid (we will see later why).

Neural Network Representation



$a_i^{(j)}$ = "activation" of unit i in layer j

$\Theta^{(j)}$ = matrix of weights (parameters) controlling function mapping from layer j to layer $j+1$

$$a_1^{(2)} = g(\Theta_{10}^{(1)}x_0 + \Theta_{11}^{(1)}x_1 + \Theta_{12}^{(1)}x_2 + \Theta_{13}^{(1)}x_3)$$

$$a_2^{(2)} = g(\Theta_{20}^{(1)}x_0 + \Theta_{21}^{(1)}x_1 + \Theta_{22}^{(1)}x_2 + \Theta_{23}^{(1)}x_3)$$

$$a_3^{(2)} = g(\Theta_{30}^{(1)}x_0 + \Theta_{31}^{(1)}x_1 + \Theta_{32}^{(1)}x_2 + \Theta_{33}^{(1)}x_3)$$

$$h_\Theta(x) = a_1^{(3)} = g(\Theta_{10}^{(2)}a_0^{(2)} + \Theta_{11}^{(2)}a_1^{(2)} + \Theta_{12}^{(2)}a_2^{(2)} + \Theta_{13}^{(2)}a_3^{(2)})$$

where g(.) is the activation function, e.g:

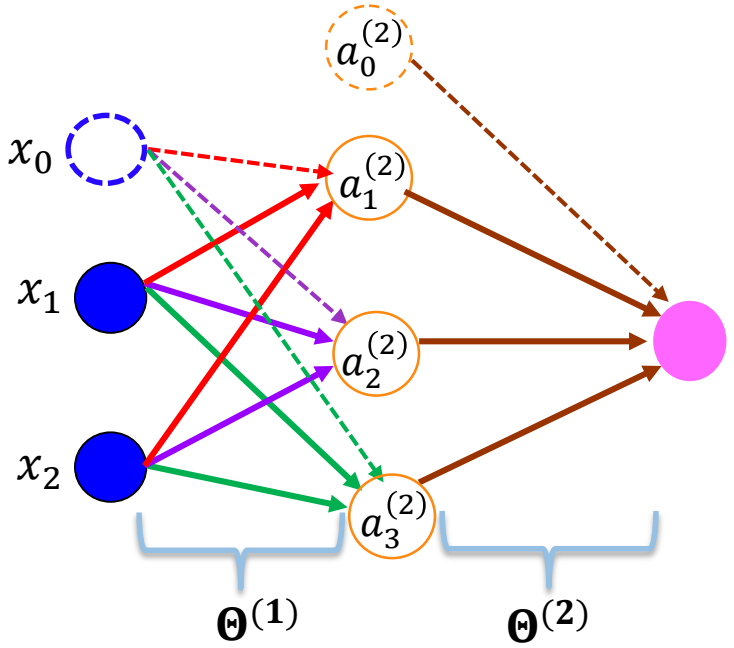
- Sigmoid: $g(z) = \frac{1}{1+e^{-z}}$
- ReLU: $g(z) = \max(0, z)$

Note: for hidden units, it recommended to not use the sigmoid (we will see later why).

If network has s_j units in layer j, s_{j+1} units in layer $j+1$, then $\Theta^{(j)}$ will be of dimension $s_{j+1} \times (s_j + 1)$.

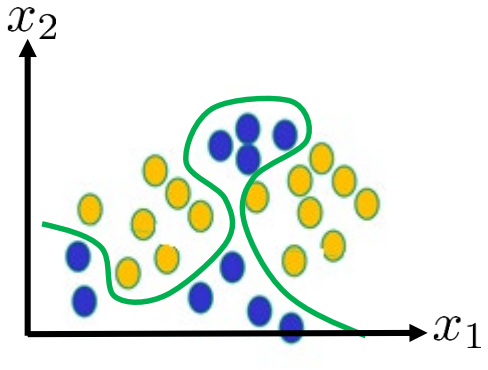
Example: $\Theta^{(1)} \in \mathbb{R}^{3\times 4}$

Neural Network Representation



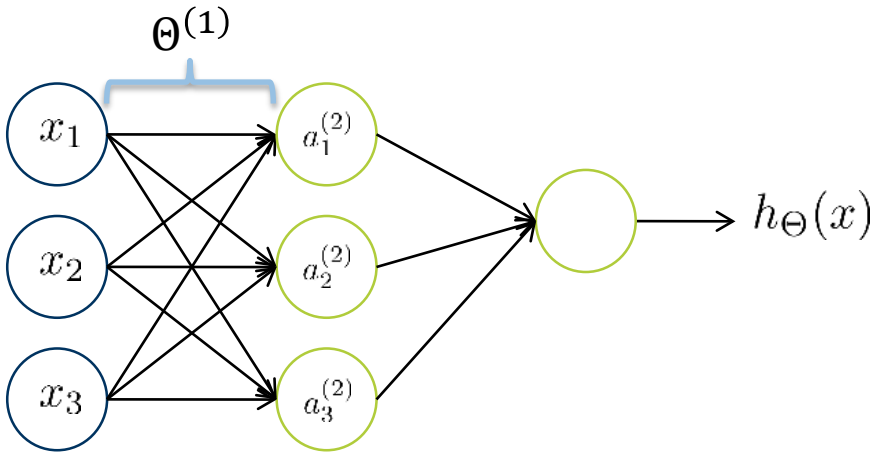
- In this example:
  - 2d input feature-vector x
  - Matrix $\Theta^{(1)}$ of dimension 3×3
    - Row 1: $\Theta_{10}^{(1)}, \Theta_{11}^{(1)}, \Theta_{12}^{(1)}$ (in red)
    - Row 2: $\Theta_{20}^{(1)}, \Theta_{21}^{(1)}, \Theta_{22}^{(1)}$ (in purple)
    - Row 3: $\Theta_{30}^{(1)}, \Theta_{31}^{(1)}, \Theta_{32}^{(1)}$ (in green)
  - For nodes in the hidden layer, you apply the sigmoid activation function $g(..)$ on each element of the vector: $\Theta^{(1)}x$:
    - The 1st unit : $a_1^{(2)} = g(\Theta_1^{(1)}.x)$
    - The 2nd unit: $a_2^{(2)} = g(\Theta_2^{(1)}.x)$
    - The 3rd unit: $a_3^{(2)} = g(\Theta_3^{(1)}.x)$

    $= \Theta^{(1)}x = a^{(2)} \in \mathbb{R}^3$



  - Matrix $\Theta^{(2)}$ of dimension 1×4
    - One row: $\Theta_{10}^{(2)}, \Theta_{11}^{(2)}, \Theta_{12}^{(2)}, \Theta_{13}^{(2)}$ (in brown)
  - The output layer gives a scalar value $\hat{y} = g(\Theta^{(2)}.a^{(2)})$

Feedforward propagation (vectorized implementation)



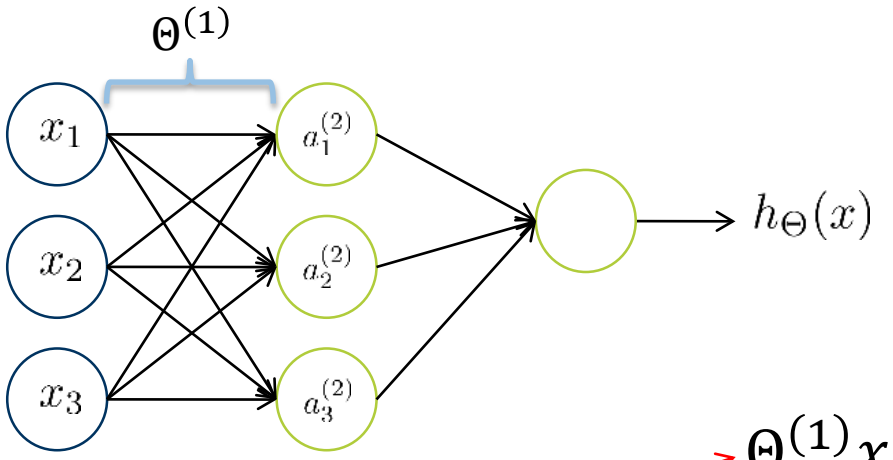
$$a_1^{(2)} = g(\Theta_{10}^{(1)} x_0 + \Theta_{11}^{(1)} x_1 + \Theta_{12}^{(1)} x_2 + \Theta_{13}^{(1)} x_3)$$

$$a_2^{(2)} = g(\Theta_{20}^{(1)} x_0 + \Theta_{21}^{(1)} x_1 + \Theta_{22}^{(1)} x_2 + \Theta_{23}^{(1)} x_3)$$

$$a_3^{(2)} = g(\Theta_{30}^{(1)} x_0 + \Theta_{31}^{(1)} x_1 + \Theta_{32}^{(1)} x_2 + \Theta_{33}^{(1)} x_3)$$

$$h_\Theta(x) = g(\Theta_{10}^{(2)} a_0^{(2)} + \Theta_{11}^{(2)} a_1^{(2)} + \Theta_{12}^{(2)} a_2^{(2)} + \Theta_{13}^{(2)} a_3^{(2)})$$

Feedforward propagation (vectorized implementation)



$\Theta^{(1)}x$

$$a_1^{(2)} = g(\Theta_{10}^{(1)}x_0 + \Theta_{11}^{(1)}x_1 + \Theta_{12}^{(1)}x_2 + \Theta_{13}^{(1)}x_3)$$

$$a_2^{(2)} = g(\Theta_{20}^{(1)}x_0 + \Theta_{21}^{(1)}x_1 + \Theta_{22}^{(1)}x_2 + \Theta_{23}^{(1)}x_3)$$

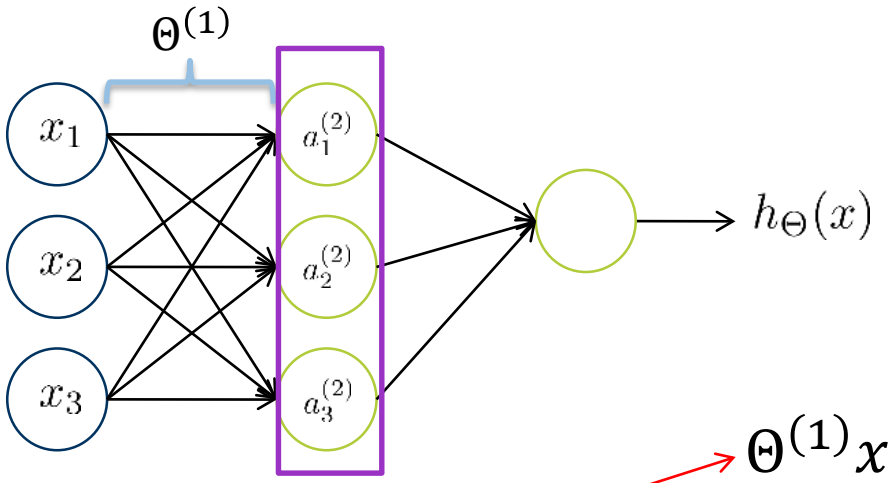
$$a_3^{(2)} = g(\Theta_{30}^{(1)}x_0 + \Theta_{31}^{(1)}x_1 + \Theta_{32}^{(1)}x_2 + \Theta_{33}^{(1)}x_3)$$

$$h_\Theta(x) = g(\Theta_{10}^{(2)}a_0^{(2)} + \Theta_{11}^{(2)}a_1^{(2)} + \Theta_{12}^{(2)}a_2^{(2)} + \Theta_{13}^{(2)}a_3^{(2)})$$

$\Theta^{(1)} \in \mathbb{R}^{3\times 4}$

$$x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

Feedforward propagation (vectorized implementation)



$\Theta^{(1)}x$

$$a_1^{(2)} = g(\Theta_{10}^{(1)}x_0 + \Theta_{11}^{(1)}x_1 + \Theta_{12}^{(1)}x_2 + \Theta_{13}^{(1)}x_3)$$

$$a_2^{(2)} = g(\Theta_{20}^{(1)}x_0 + \Theta_{21}^{(1)}x_1 + \Theta_{22}^{(1)}x_2 + \Theta_{23}^{(1)}x_3)$$

$$a_3^{(2)} = g(\Theta_{30}^{(1)}x_0 + \Theta_{31}^{(1)}x_1 + \Theta_{32}^{(1)}x_2 + \Theta_{33}^{(1)}x_3)$$

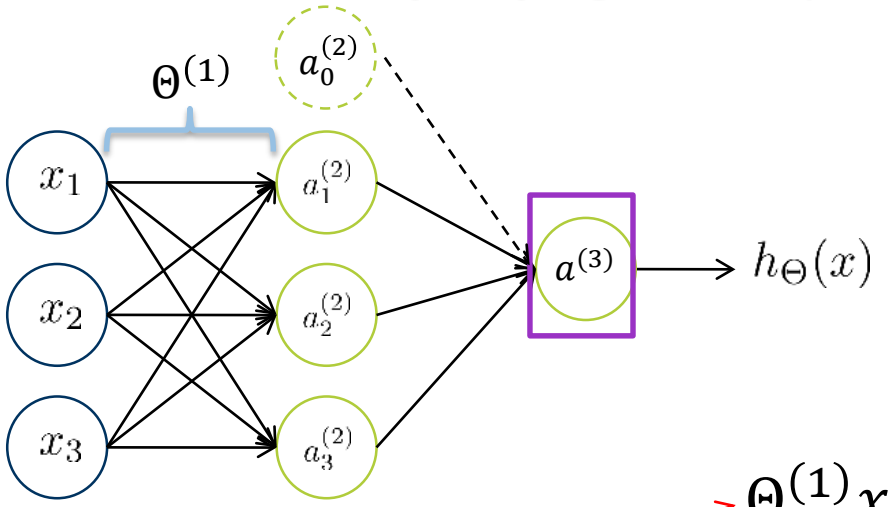
$$h_\Theta(x) = g(\Theta_{10}^{(2)}a_0^{(2)} + \Theta_{11}^{(2)}a_1^{(2)} + \Theta_{12}^{(2)}a_2^{(2)} + \Theta_{13}^{(2)}a_3^{(2)})$$

$\Theta^{(1)} \in \mathbb{R}^{3\times 4}$

$a^{(2)} = g(\Theta^{(1)}x)$

$$x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

Feedforward propagation (vectorized implementation)



$\Theta^{(1)} x$

$$a_1^{(2)} = g(\Theta_{10}^{(1)} x_0 + \Theta_{11}^{(1)} x_1 + \Theta_{12}^{(1)} x_2 + \Theta_{13}^{(1)} x_3)$$

$$a_2^{(2)} = g(\Theta_{20}^{(1)} x_0 + \Theta_{21}^{(1)} x_1 + \Theta_{22}^{(1)} x_2 + \Theta_{23}^{(1)} x_3)$$

$$a_3^{(2)} = g(\Theta_{30}^{(1)} x_0 + \Theta_{31}^{(1)} x_1 + \Theta_{32}^{(1)} x_2 + \Theta_{33}^{(1)} x_3)$$

$$h_\Theta(x) = g(\Theta_{10}^{(2)} a_0^{(2)} + \Theta_{11}^{(2)} a_1^{(2)} + \Theta_{12}^{(2)} a_2^{(2)} + \Theta_{13}^{(2)} a_3^{(2)})$$

$\Theta^{(2)} a^{(2)}$

$\Theta^{(1)} \in \mathbb{R}^{3\times 4}$

$$x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

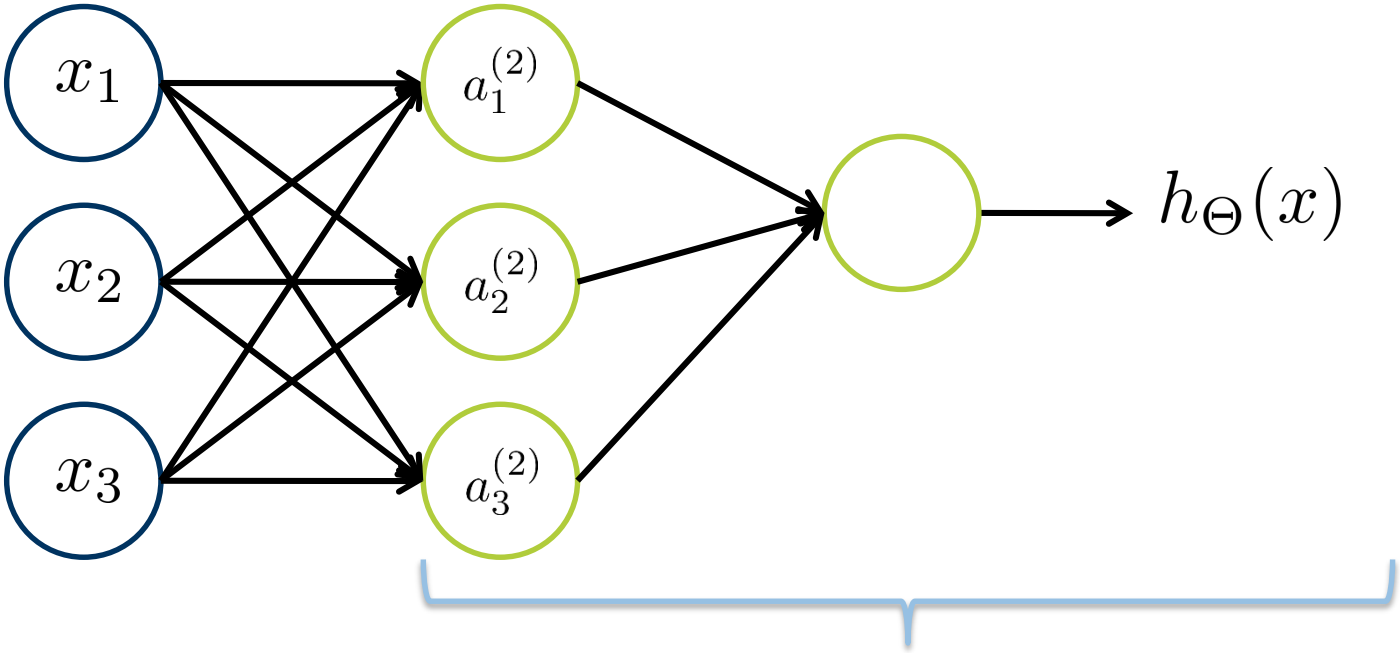
$a^{(2)} = g(\Theta^{(1)} x)$

Add $a_0^{(2)} = 1$ to $a^{(2)}$

$\Theta^{(2)} \in \mathbb{R}^{1\times 4}$

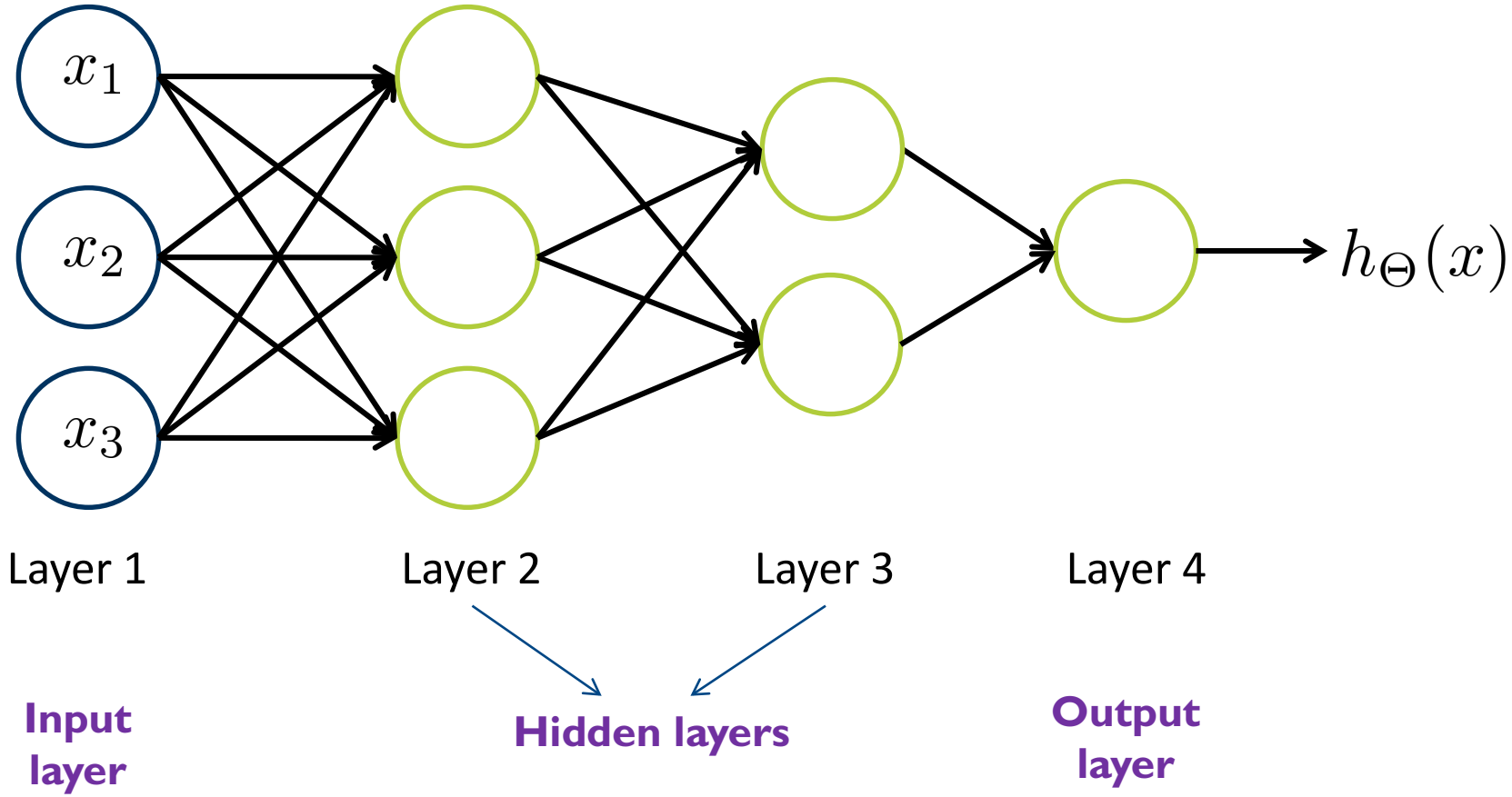
$a^{(3)} = h_\Theta(x) = g(\Theta^{(2)} a^{(2)})$

Neural Network learns its own features



This part of the network is simply doing logistic regression, except that instead of using the original input features x_1, x_2, x_3, it uses the new features $a_1^{(2)}, a_2^{(2)}, a_3^{(2)}$. These new features are themselves learned from the input.

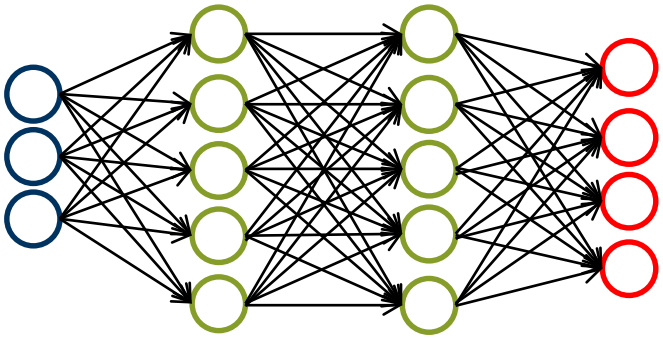
Other Neural Network Architectures



Multiple output units
(multi-class classification)

Suppose we have a multi-class classification problem where you want to learn to classify an image in one of the 4 classes: "Pedestrian", "Car", "Motorcycle", "Truck"

di bo



$$h_\Theta(x) \in \mathbb{R}^4$$

output

Pedestrian

$$\begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Car

$$\begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$$

Motorcycle

$$\begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}$$

Truck

$$\begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

- Each prediction $h_\Theta(x)$ is in $\mathbb{R}^4$.
- Each output $y^{(i)}$ in the training set should be in $\mathbb{R}^4$.

A Neural Networks is a Universal Approximation Function

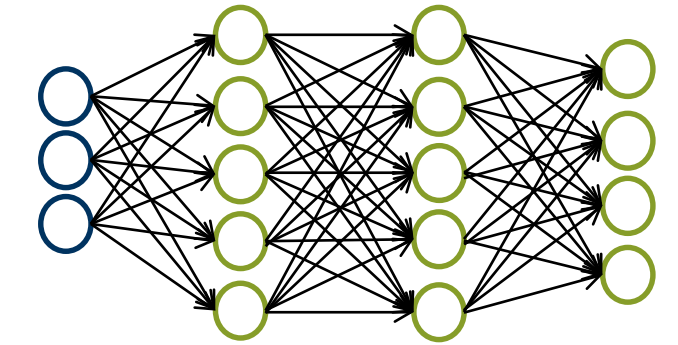
pho quat

- A neural network with at least one hidden layer can, *in theory*, learn/approximate perfectly any function.
  - A set of parameters (weights) that can produce the outputs from the inputs, exists.
  - The problem is finding them …

- *In practice,* usually the more units/layers you have, the better, but it becomes more computationally expensive (time consuming) to train such **deep** neural networks.
  - The number of layers and the number of units in each layer, are hyperparameters ...

Cost Function

Some notations



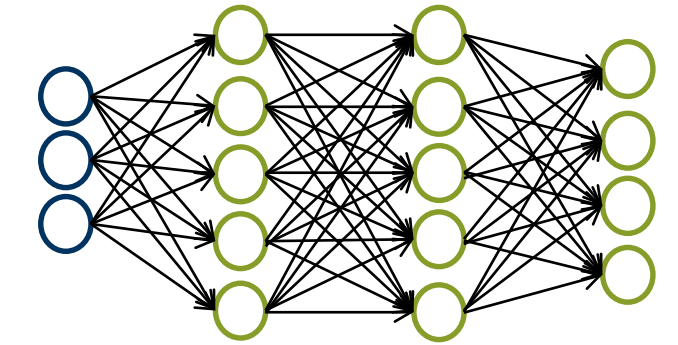
$$\{(x^{(1)}, y^{(1)}), (x^{(1)}, y^{(1)}), \dots, (x^{(n)}, y^{(n)})\}$$

$L =$ total no. of layers in network

$s_l =$ no. of units (not counting bias unit) in layer l

- What is the value of L in the above example ?
- What is the value of s_1, s_2, s_3, s_4 in the above example ?

Some notations



$$\{(x^{(1)}, y^{(1)}), (x^{(1)}, y^{(1)}), \dots, (x^{(n)}, y^{(n)})\}$$

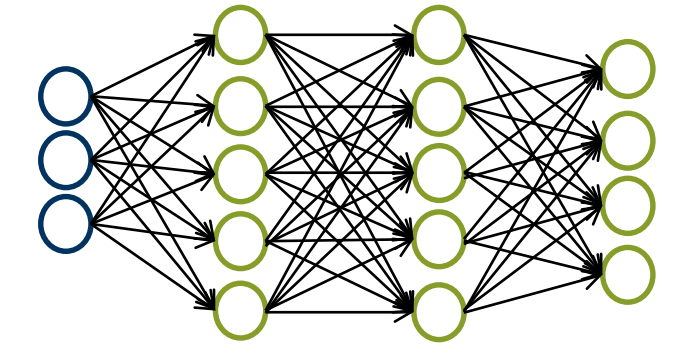
$L =$ total no. of layers in network

$s_l =$ no. of units (not counting bias unit) in layer l

- What is the value of L in the above example ?
  ➔ $L = 4$

- What is the value of s_1, s_2, s_3, s_4 in the above example ?
  ➔ $s_1 = 3,\ s_2 = 5,\ s_3 = 5,\ s_4 = s_L = 4$

Some notations



$\{(x^{(1)}, y^{(1)}), (x^{(1)}, y^{(1)}), \ldots, (x^{(n)}, y^{(n)})\}$

$L =$ total no. of layers in network

$s_l =$ no. of units (not counting bias unit) in layer l

Binary classification

$y = 0$ or 1

… or regression

1 output unit ($s_L = 1$)

$h_\Theta(x) \in \mathbb{R}$

Multi-class classification (K classes)

$y \in \mathbb{R}^K$ E.g. $\begin{bmatrix}1\\0\\0\\0\end{bmatrix}$, $\begin{bmatrix}0\\1\\0\\0\end{bmatrix}$, $\begin{bmatrix}0\\0\\1\\0\end{bmatrix}$, $\begin{bmatrix}0\\0\\0\\1\end{bmatrix}$

pedestrian car motorcycle truck

K output units ($s_L = K$)

$h_\Theta(x) \in \mathbb{R}^K$ $K \geq 3$

Cost function

Logistic regression:

$$E(\theta) = -\frac{1}{n}\left[\sum_{i=1}^{n} y^{(i)} \log h_\theta(x^{(i)}) + (1 - y^{(i)}) \log(1 - h_\theta(x^{(i)}))\right] + \frac{\lambda}{2\,n}\sum_{j=1}^{d} \theta_j^2$$

Neural network:

$$h_\Theta(x) \in \mathbb{R}^K \quad (h_\Theta(x))_i = i^{th} \text{ output}$$

$$E(\Theta) = -\frac{1}{n}\left[\sum_{i=1}^{n}\sum_{k=1}^{K} y_k^{(i)} \log(h_\Theta(x^{(i)}))_k + (1 - y_k^{(i)}) \log(1 - (h_\Theta(x^{(i)}))_k)\right]$$

$$+\frac{\lambda}{2\,n}\sum_{l=1}^{L-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}(\Theta_{ji}^{(l)})^2$$

Backpropagation

lan truyen nguoc

Gradient Computation

$$E(\Theta) = -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)} \log(h_\Theta(x^{(i)}))_k + (1 - y_k^{(i)}) \log(1 - (h_\Theta(x^{(i)}))_k)\right]$$

$$+\frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}(\Theta_{ji}^{(l)})^2$$

We need to minimize $E(\Theta)$.

To do so, we need to compute the partial derivatives $\frac{\partial}{\partial \Theta_{ij}^{(l)}} E(\Theta)$.

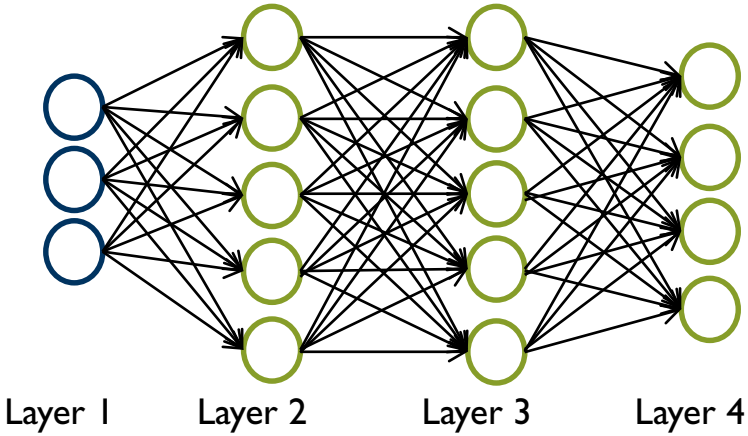
Gradient Computation

The first thing to do, is forward propagation

Given one training example (x, y) :

Forward propagation:

$$
\begin{aligned}
a^{(1)} &= x \\
z^{(2)} &= \Theta^{(1)} a^{(1)} \\
a^{(2)} &= g(z^{(2)}) \quad (\text{add } a_0^{(2)}) \\
z^{(3)} &= \Theta^{(2)} a^{(2)} \\
a^{(3)} &= g(z^{(3)}) \quad (\text{add } a_0^{(3)}) \\
z^{(4)} &= \Theta^{(3)} a^{(3)} \\
a^{(4)} &= h_\Theta(x) = g(z^{(4)})
\end{aligned}
$$



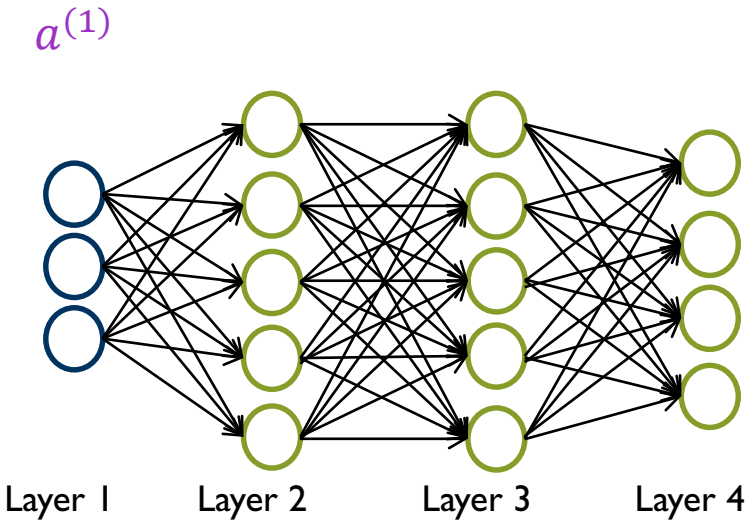
Gradient Computation

The first thing to do, is forward propagation

Given one training example (x, y) :

Forward propagation:

$$a^{(1)} = x$$
$$z^{(2)} = \Theta^{(1)} a^{(1)}$$
$$a^{(2)} = g(z^{(2)}) \quad (\text{add } a_0^{(2)})$$
$$z^{(3)} = \Theta^{(2)} a^{(2)}$$
$$a^{(3)} = g(z^{(3)}) \quad (\text{add } a_0^{(3)})$$
$$z^{(4)} = \Theta^{(3)} a^{(3)}$$
$$a^{(4)} = h_\Theta(x) = g(z^{(4)})$$



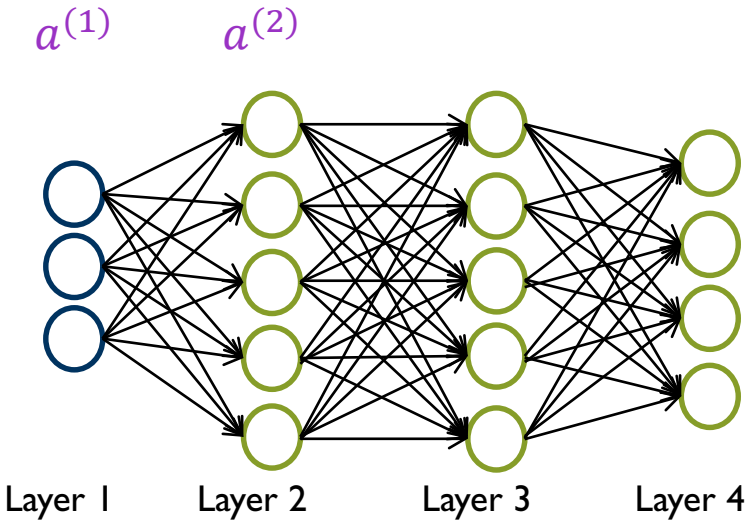
Gradient Computation

The first thing to do, is forward propagation

Given one training example (x, y) :

Forward propagation:

$$
\begin{aligned}
a^{(1)} &= x \\
z^{(2)} &= \Theta^{(1)} a^{(1)} \\
a^{(2)} &= g(z^{(2)}) \quad (\text{add } a_0^{(2)}) \\
z^{(3)} &= \Theta^{(2)} a^{(2)} \\
a^{(3)} &= g(z^{(3)}) \quad (\text{add } a_0^{(3)}) \\
z^{(4)} &= \Theta^{(3)} a^{(3)} \\
a^{(4)} &= h_\Theta(x) = g(z^{(4)})
\end{aligned}
$$



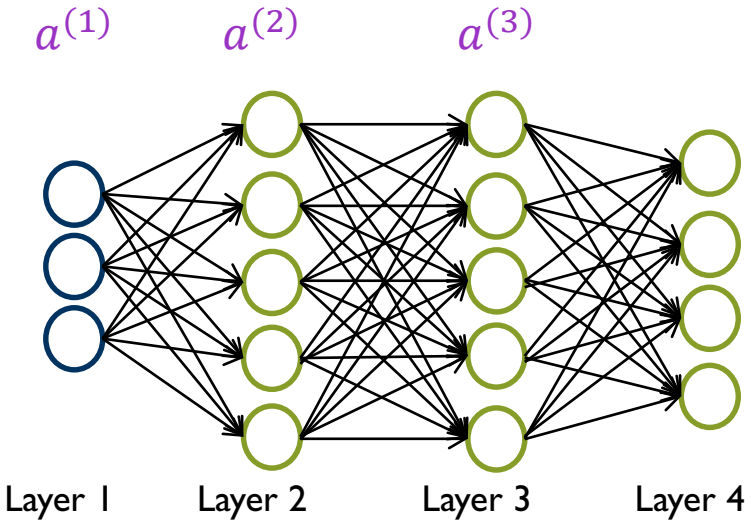
Gradient Computation

The first thing to do, is forward propagation

Given one training example (x, y) :

Forward propagation:

$$a^{(1)} = x$$
$$z^{(2)} = \Theta^{(1)} a^{(1)}$$
$$a^{(2)} = g(z^{(2)}) \quad (\text{add } a_0^{(2)})$$
$$z^{(3)} = \Theta^{(2)} a^{(2)}$$
$$a^{(3)} = g(z^{(3)}) \quad (\text{add } a_0^{(3)})$$
$$z^{(4)} = \Theta^{(3)} a^{(3)}$$
$$a^{(4)} = h_\Theta(x) = g(z^{(4)})$$



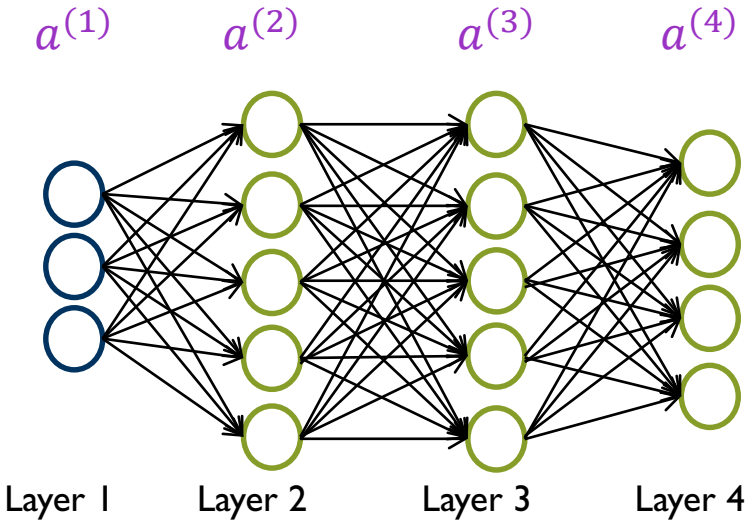
Gradient Computation

The first thing to do, is forward propagation

Given one training example (x, y) :

Forward propagation:

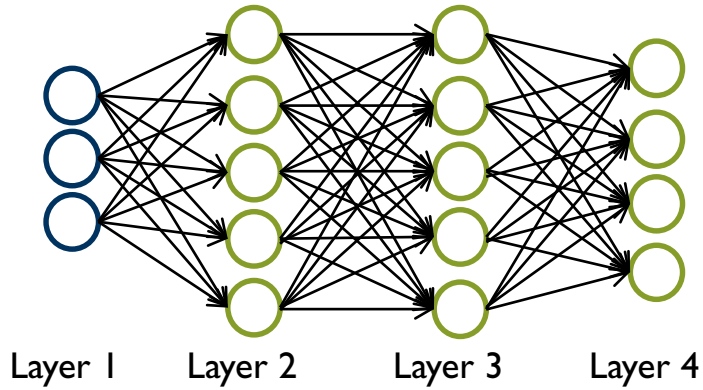
$$a^{(1)} = x$$
$$z^{(2)} = \Theta^{(1)} a^{(1)}$$
$$a^{(2)} = g(z^{(2)}) \quad (\text{add } a_0^{(2)})$$
$$z^{(3)} = \Theta^{(2)} a^{(2)}$$
$$a^{(3)} = g(z^{(3)}) \quad (\text{add } a_0^{(3)})$$
$$z^{(4)} = \Theta^{(3)} a^{(3)}$$
$$a^{(4)} = h_\Theta(x) = g(z^{(4)})$$



Gradient Computation: Backpropagation

Intuition: $\delta_j^{(l)} =$ "error" of node j in layer l .

truc giac

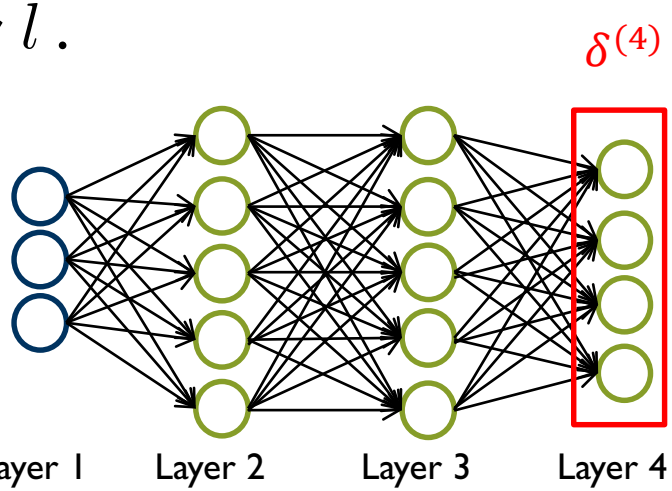


For each output unit in layer $L = 4$)

$$\delta_j^{(4)} = a_j^{(4)} - y_j$$

Gradient Computation: Backpropagation

Intuition: $\delta_j^{(l)} =$ "error" of node j in layer l.



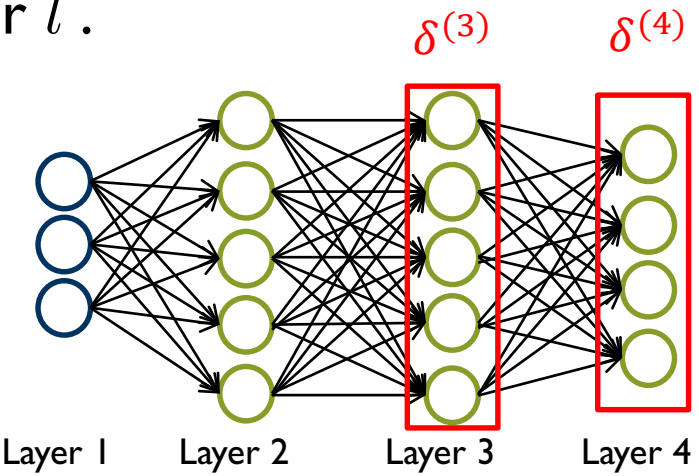
For each output unit in layer $L = 4$)

$$\delta_j^{(4)} = a_j^{(4)} - y_j$$

Vectorized version:
$\delta^{(4)} = a^{(4)} - y$

Gradient Computation: Backpropagation

Intuition: $\delta_j^{(l)} =$ "error" of node j in layer l.



For each output unit in layer $L = 4$)

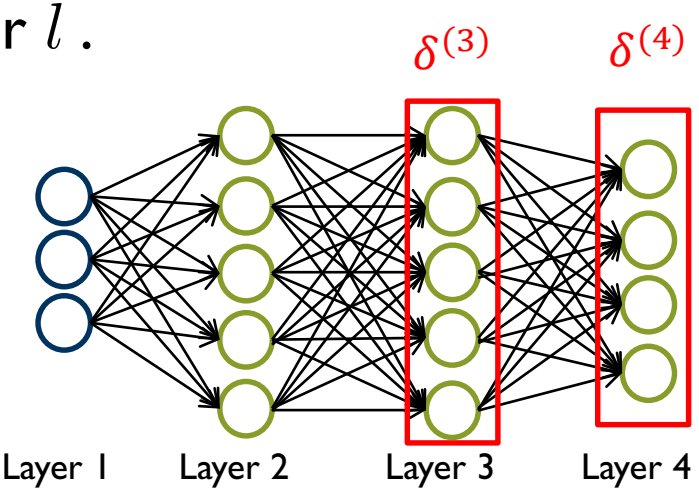
$$\delta_j^{(4)} = a_j^{(4)} - y_j$$

Vectorized version:
$\delta^{(4)} = a^{(4)} - y$

$$\delta^{(3)} = (\Theta^{(3)})^T \delta^{(4)} .* g'(z^{(3)})$$

Gradient Computation: Backpropagation

Intuition: $\delta_j^{(l)} =$ "error" of node j in layer l.



For each output unit in layer $L = 4$)

$$\delta_j^{(4)} = a_j^{(4)} - y_j$$

Vectorized version:
$\delta^{(4)} = a^{(4)} - y$

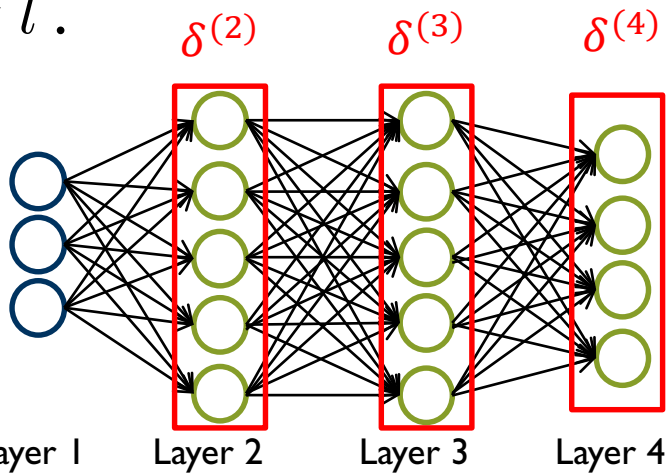
$$\delta^{(3)} = (\Theta^{(3)})^T \delta^{(4)} .* g'(z^{(3)})$$

Example:
$a^{(3)} * (1 - a^{(3)})$

that's if the activation g(.) is the sigmoid function.

Gradient Computation: Backpropagation

Intuition: $\delta_j^{(l)} =$ "error" of node j in layer l.



For each output unit in layer $L = 4$)

$$\delta_j^{(4)} = a_j^{(4)} - y_j$$

Vectorized version:
$\delta^{(4)} = a^{(4)} - y$

$$\delta^{(3)} = (\Theta^{(3)})^T \delta^{(4)} .* g'(z^{(3)})$$

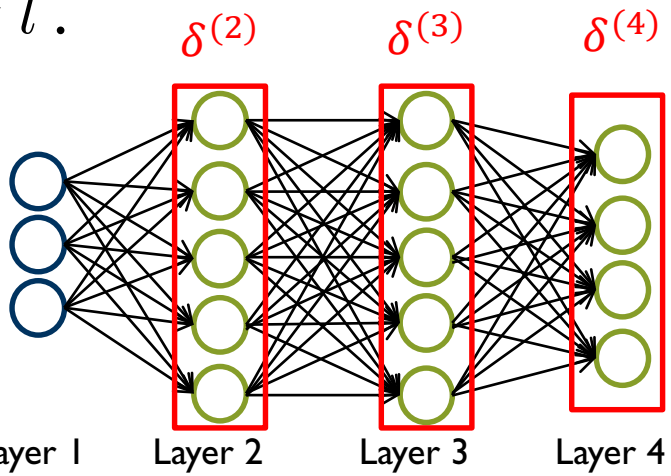
$$\delta^{(2)} = (\Theta^{(2)})^T \delta^{(3)} .* g'(z^{(2)})$$

Example:
$a^{(3)} * (1 - a^{(3)})$
$a^{(2)} * (1 - a^{(2)})$

that's if the activation g(.) is the sigmoid function.

Gradient Computation: Backpropagation

Intuition: $\delta_j^{(l)} =$ "error" of node j in layer l.



For each output unit in layer $L = 4$)

$$\delta_j^{(4)} = a_j^{(4)} - y_j$$

Vectorized version:
$\delta^{(4)} = a^{(4)} - y$

$$\delta^{(3)} = (\Theta^{(3)})^T \delta^{(4)} .* g'(z^{(3)})$$

$$\delta^{(2)} = (\Theta^{(2)})^T \delta^{(3)} .* g'(z^{(2)})$$

Example:
$a^{(3)} * (1 - a^{(3)})$
$a^{(2)} * (1 - a^{(2)})$

that's if the activation g(.) is the sigmoid function.

This symbol denotes the elementwise multiplication

Backpropagation algorithm

Training set $\{(x^{(1)}, y^{(1)}), \ldots, (x^{(n)}, y^{(n)})\}$

Set $\triangle_{ij}^{(l)} = 0$ (for all l, i, j).

For $i = 1$ to m

Set $a^{(1)} = x^{(i)}$

Perform forward propagation to compute $a^{(l)}$ for $l = 2, 3, \ldots, L$

Using $y^{(i)}$, compute $\delta^{(L)} = a^{(L)} - y^{(i)}$

Compute $\delta^{(L-1)}, \delta^{(L-2)}, \ldots, \delta^{(2)}$

$\triangle_{ij}^{(l)} := \triangle_{ij}^{(l)} + a_j^{(l)} \delta_i^{(l+1)}$

$$D_{ij}^{(l)} := \frac{1}{n} \triangle_{ij}^{(l)} + \lambda \Theta_{ij}^{(l)} \text{ if } j \neq 0$$

$$D_{ij}^{(l)} := \frac{1}{n} \triangle_{ij}^{(l)} \text{ if } j = 0$$

$$\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta) = D_{ij}^{(l)}$$

Backpropagation algorithm

Training set $\{(x^{(1)}, y^{(1)}), \ldots, (x^{(n)}, y^{(n)})\}$ The training dataset

Set $\triangle_{ij}^{(l)} = 0$ (for all l, i, j).

For $i = 1$ to m

Set $a^{(1)} = x^{(i)}$

Perform forward propagation to compute $a^{(l)}$ for $l = 2, 3, \ldots, L$

Using $y^{(i)}$, compute $\delta^{(L)} = a^{(L)} - y^{(i)}$

Compute $\delta^{(L-1)}, \delta^{(L-2)}, \ldots, \delta^{(2)}$

$\triangle_{ij}^{(l)} := \triangle_{ij}^{(l)} + a_j^{(l)} \delta_i^{(l+1)}$

$$D_{ij}^{(l)} := \frac{1}{n} \triangle_{ij}^{(l)} + \lambda \Theta_{ij}^{(l)} \text{ if } j \neq 0$$

$$D_{ij}^{(l)} := \frac{1}{n} \triangle_{ij}^{(l)} \text{ if } j = 0$$

$$\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta) = D_{ij}^{(l)}$$

Backpropagation algorithm

Training set $\{(x^{(1)}, y^{(1)}), \ldots, (x^{(n)}, y^{(n)})\}$

Set $\triangle_{ij}^{(l)} = 0$ (for all l, i, j).

Used to compute $\frac{\partial}{\partial \Theta_{ij}^{(l)}} E(\Theta)$

For $i = 1$ to m

Set $a^{(1)} = x^{(i)}$

Perform forward propagation to compute $a^{(l)}$ for $l = 2, 3, \ldots, L$

Using $y^{(i)}$, compute $\delta^{(L)} = a^{(L)} - y^{(i)}$

Compute $\delta^{(L-1)}, \delta^{(L-2)}, \ldots, \delta^{(2)}$

$\triangle_{ij}^{(l)} := \triangle_{ij}^{(l)} + a_j^{(l)} \delta_i^{(l+1)}$

$$D_{ij}^{(l)} := \frac{1}{n} \triangle_{ij}^{(l)} + \lambda \Theta_{ij}^{(l)} \text{ if } j \neq 0$$

$$D_{ij}^{(l)} := \frac{1}{n} \triangle_{ij}^{(l)} \text{ if } j = 0$$

$$\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta) = D_{ij}^{(l)}$$

Backpropagation algorithm

Training set $\{(x^{(1)}, y^{(1)}), \ldots, (x^{(n)}, y^{(n)})\}$

Set $\triangle_{ij}^{(l)} = 0$ (for all l, i, j).

For $i = 1$ to m

Do the following for each training example

Set $a^{(1)} = x^{(i)}$

Perform forward propagation to compute $a^{(l)}$ for $l = 2, 3, \ldots, L$

Using $y^{(i)}$, compute $\delta^{(L)} = a^{(L)} - y^{(i)}$

Compute $\delta^{(L-1)}, \delta^{(L-2)}, \ldots, \delta^{(2)}$

$\triangle_{ij}^{(l)} := \triangle_{ij}^{(l)} + a_j^{(l)} \delta_i^{(l+1)}$

$$D_{ij}^{(l)} := \frac{1}{n} \triangle_{ij}^{(l)} + \lambda \Theta_{ij}^{(l)} \text{ if } j \neq 0$$

$$D_{ij}^{(l)} := \frac{1}{n} \triangle_{ij}^{(l)} \text{ if } j = 0$$

$$\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta) = D_{ij}^{(l)}$$

Backpropagation algorithm

Training set $\{(x^{(1)}, y^{(1)}), \ldots, (x^{(n)}, y^{(n)})\}$

Set $\triangle_{ij}^{(l)} = 0$ (for all l, i, j).

For $i = 1$ to m

Set $a^{(1)} = x^{(i)}$ The first layer's vector is just the input vector

Perform forward propagation to compute $a^{(l)}$ for $l = 2, 3, \ldots, L$

Using $y^{(i)}$, compute $\delta^{(L)} = a^{(L)} - y^{(i)}$

Compute $\delta^{(L-1)}, \delta^{(L-2)}, \ldots, \delta^{(2)}$

$\triangle_{ij}^{(l)} := \triangle_{ij}^{(l)} + a_j^{(l)} \delta_i^{(l+1)}$

$$D_{ij}^{(l)} := \frac{1}{n} \triangle_{ij}^{(l)} + \lambda \Theta_{ij}^{(l)} \text{ if } j \neq 0$$

$$D_{ij}^{(l)} := \frac{1}{n} \triangle_{ij}^{(l)} \text{ if } j = 0$$

$$\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta) = D_{ij}^{(l)}$$

Backpropagation algorithm

Training set $\{(x^{(1)}, y^{(1)}), \ldots, (x^{(n)}, y^{(n)})\}$

Set $\triangle_{ij}^{(l)} = 0$ (for all l, i, j).

For $i = 1$ to m

Set $a^{(1)} = x^{(i)}$

Perform forward propagation to compute $a^{(l)}$ for $l = 2, 3, \ldots, L$

Using $y^{(i)}$, compute $\delta^{(L)} = a^{(L)} - y^{(i)}$

Compute $\delta^{(L-1)}, \delta^{(L-2)}, \ldots, \delta^{(2)}$

$\triangle_{ij}^{(l)} := \triangle_{ij}^{(l)} + a_j^{(l)} \delta_i^{(l+1)}$

$$D_{ij}^{(l)} := \frac{1}{n} \triangle_{ij}^{(l)} + \lambda \Theta_{ij}^{(l)} \text{ if } j \neq 0$$

$$D_{ij}^{(l)} := \frac{1}{n} \triangle_{ij}^{(l)} \text{ if } j = 0$$

$$\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta) = D_{ij}^{(l)}$$

Backpropagation algorithm

Training set $\{(x^{(1)}, y^{(1)}), \ldots, (x^{(n)}, y^{(n)})\}$

Set $\triangle_{ij}^{(l)} = 0$ (for all l, i, j).

For $i = 1$ to m

Set $a^{(1)} = x^{(i)}$

Perform forward propagation to compute $a^{(l)}$ for $l = 2, 3, \ldots, L$

Using $y^{(i)}$, compute $\delta^{(L)} = a^{(L)} - y^{(i)}$

Compute $\delta^{(L-1)}, \delta^{(L-2)}, \ldots, \delta^{(2)}$

$\triangle_{ij}^{(l)} := \triangle_{ij}^{(l)} + a_j^{(l)} \delta_i^{(l+1)}$

As we did in a previous slide.

$$D_{ij}^{(l)} := \frac{1}{n} \triangle_{ij}^{(l)} + \lambda \Theta_{ij}^{(l)} \text{ if } j \neq 0$$

$$D_{ij}^{(l)} := \frac{1}{n} \triangle_{ij}^{(l)} \text{ if } j = 0$$

$$\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta) = D_{ij}^{(l)}$$

Backpropagation algorithm

Training set $\{(x^{(1)}, y^{(1)}), \ldots, (x^{(n)}, y^{(n)})\}$

Set $\triangle_{ij}^{(l)} = 0$ (for all l, i, j).

For $i = 1$ to m

Set $a^{(1)} = x^{(i)}$

Perform forward propagation to compute $a^{(l)}$ for $l = 2, 3, \ldots, L$

Using $y^{(i)}$, compute $\delta^{(L)} = a^{(L)} - y^{(i)}$

Compute $\delta^{(L-1)}, \delta^{(L-2)}, \ldots, \delta^{(2)}$

$$\triangle_{ij}^{(l)} := \triangle_{ij}^{(l)} + a_j^{(l)} \delta_i^{(l+1)}$$

➔ Vectorized version: $\Delta^{(l)} := \Delta^{(l)} + \delta^{(l+1)} \left(a^{(l)}\right)^T$

$$D_{ij}^{(l)} := \frac{1}{n} \triangle_{ij}^{(l)} + \lambda \Theta_{ij}^{(l)} \text{ if } j \neq 0$$

$$D_{ij}^{(l)} := \frac{1}{n} \triangle_{ij}^{(l)} \text{ if } j = 0$$

$$\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta) = D_{ij}^{(l)}$$

Backpropagation algorithm

Training set $\{(x^{(1)}, y^{(1)}), \ldots, (x^{(n)}, y^{(n)})\}$

Set $\triangle_{ij}^{(l)} = 0$ (for all l, i, j).

For $i = 1$ to m

Set $a^{(1)} = x^{(i)}$

Perform forward propagation to compute $a^{(l)}$ for $l = 2, 3, \ldots, L$

Using $y^{(i)}$, compute $\delta^{(L)} = a^{(L)} - y^{(i)}$

Compute $\delta^{(L-1)}, \delta^{(L-2)}, \ldots, \delta^{(2)}$

$\triangle_{ij}^{(l)} := \triangle_{ij}^{(l)} + a_j^{(l)} \delta_i^{(l+1)}$

Now you can use $D_{ij}^{(l)}$ as your partial derivatives in (e.g.) gradient descent (or other optimization procedures), to update the parameters $\Theta_{ij}^{(l)}$

$$D_{ij}^{(l)} := \frac{1}{n} \triangle_{ij}^{(l)} + \lambda \Theta_{ij}^{(l)} \text{ if } j \neq 0$$
$$D_{ij}^{(l)} := \frac{1}{n} \triangle_{ij}^{(l)} \text{ if } j = 0$$

$$\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta) = D_{ij}^{(l)}$$

Illustration of Backpropagation

Illustration of Backpropagation

A training 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 …			

Initialise with random weights

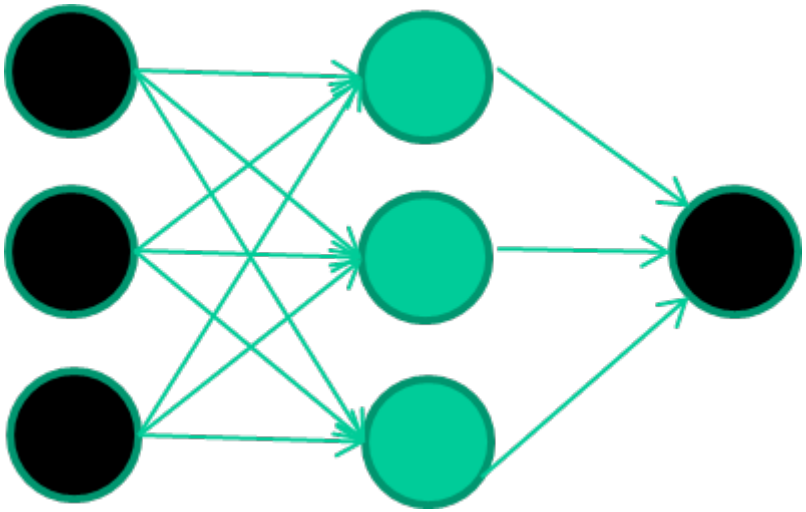


Illustration of Backpropagation

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

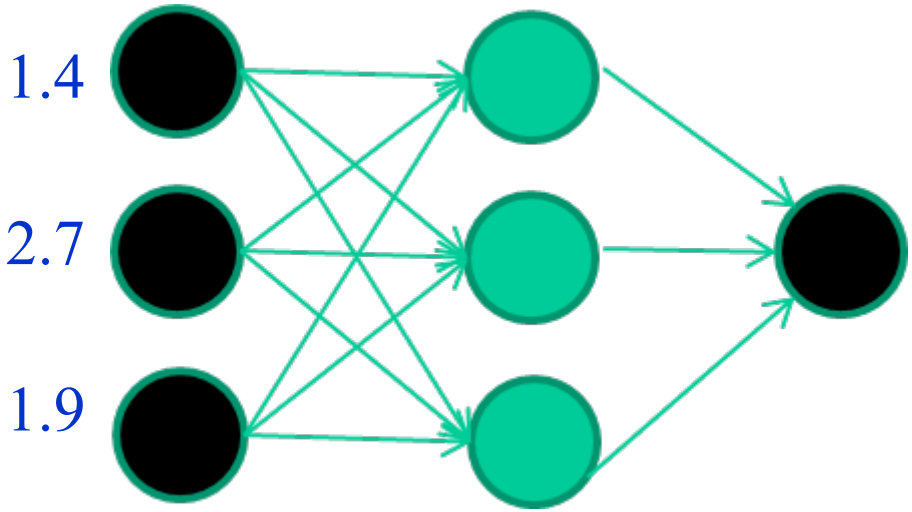


Illustration of Backpropagation

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 (Feedforward)

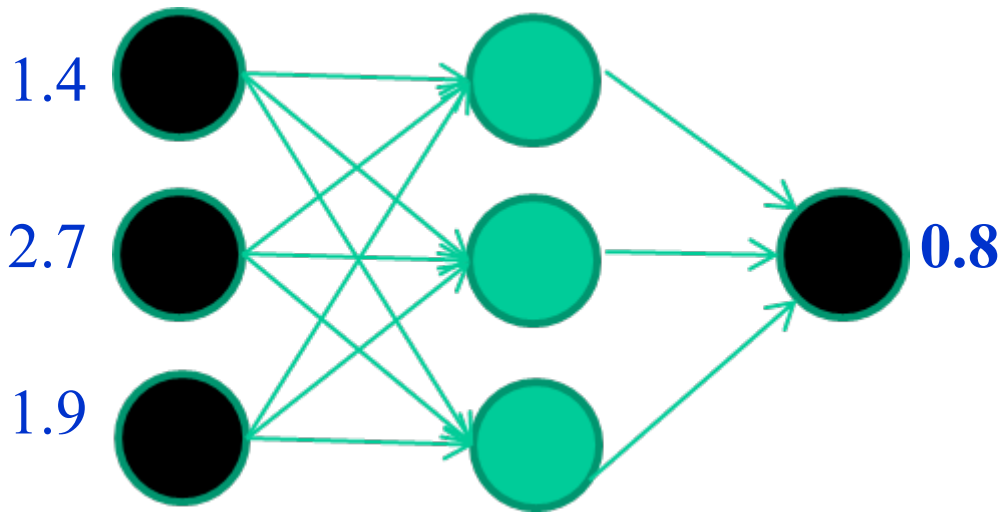


Illustration of Backpropagation

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

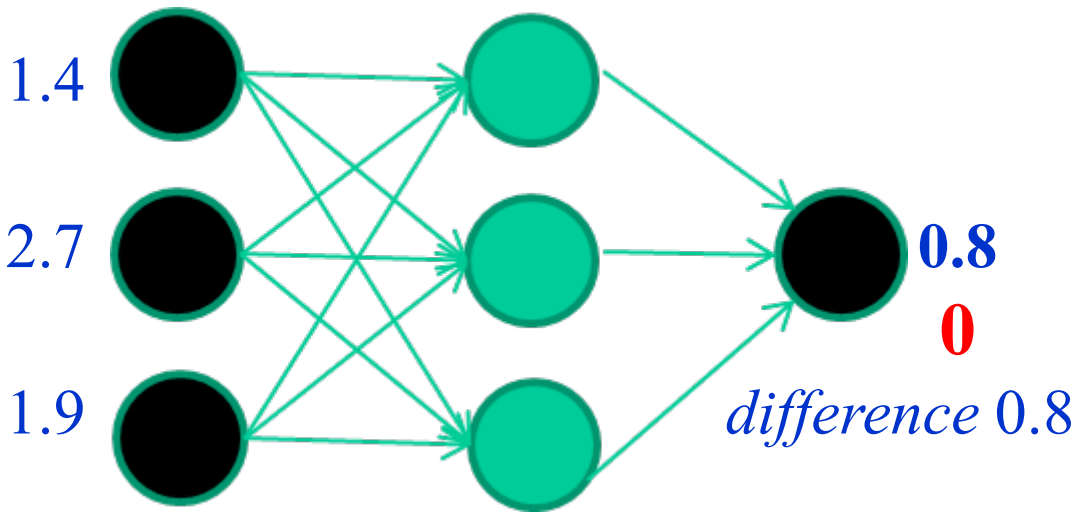


Illustration of Backpropagation

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 (Backpropagation)

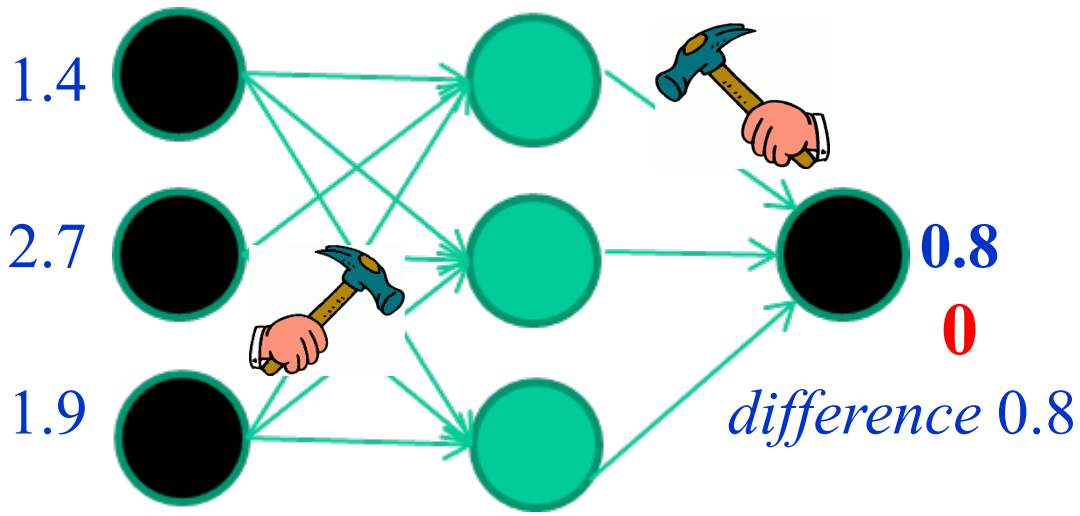


Illustration of Backpropagation

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

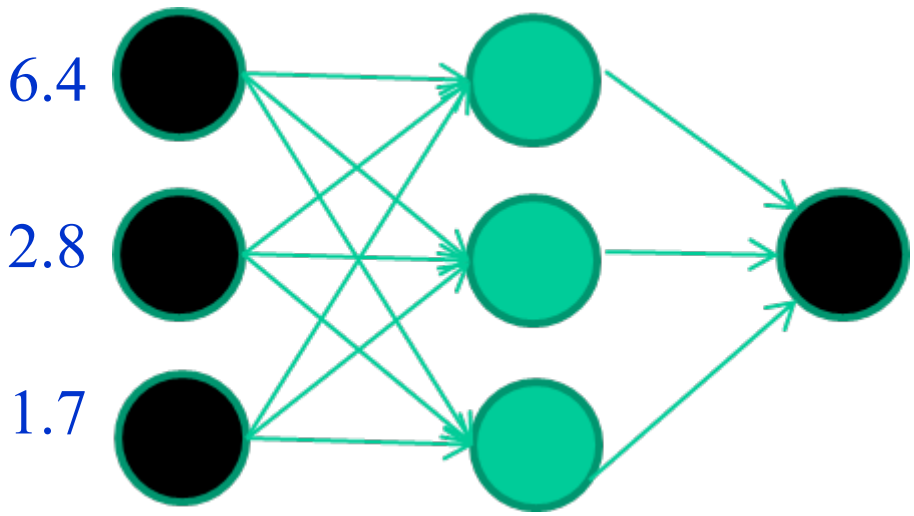


Illustration of Backpropagation

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 (Feedforward)

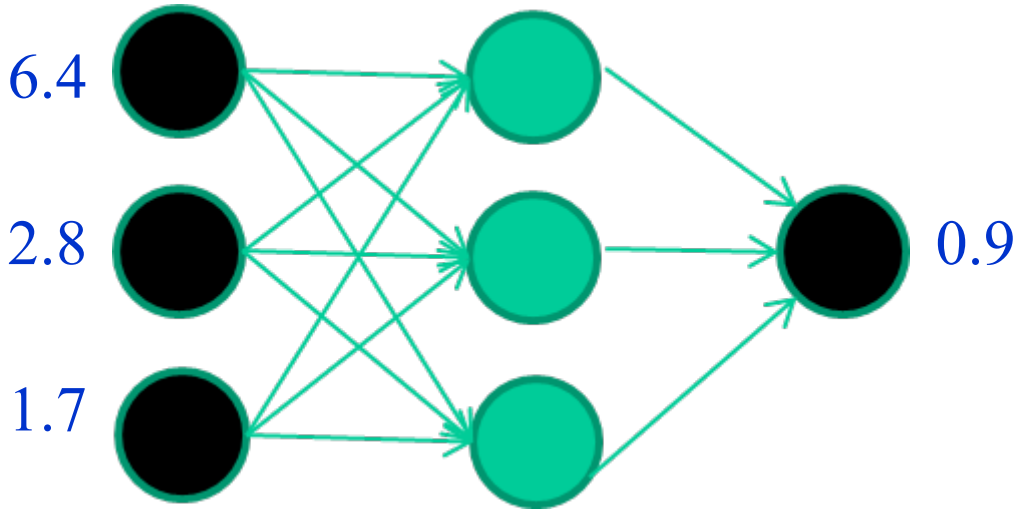


Illustration of Backpropagation

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

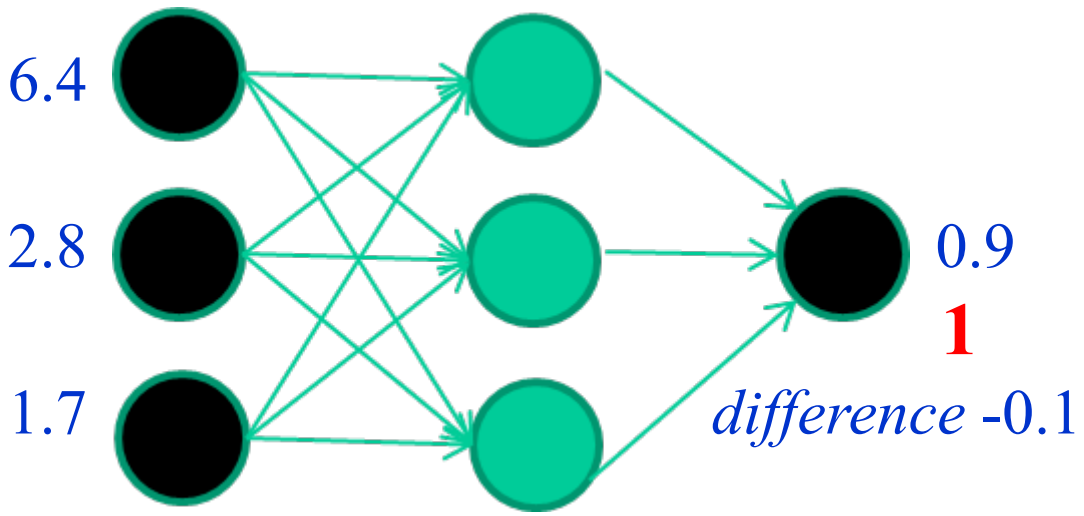


Illustration of Backpropagation

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 (Backpropagation)

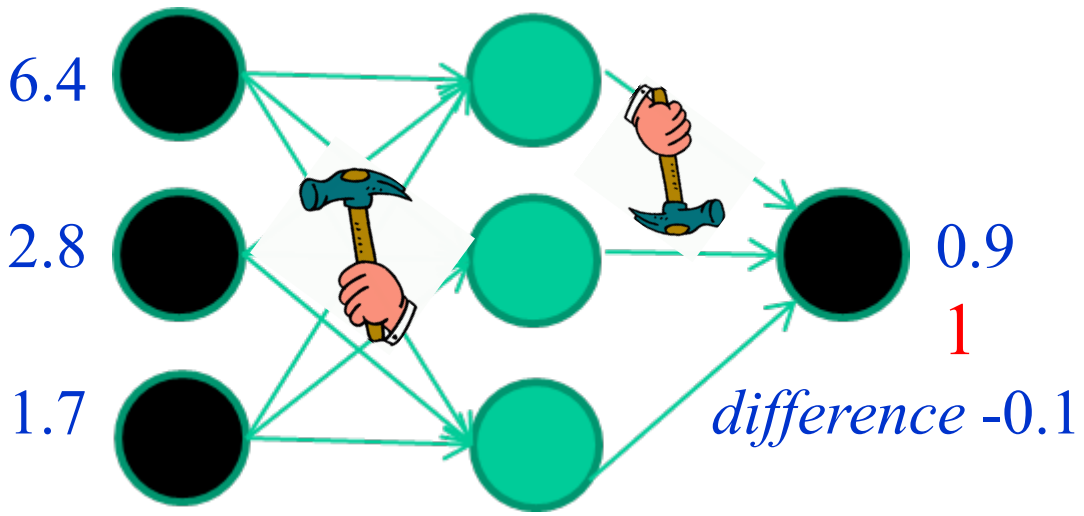


Illustration of Backpropagation

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 …

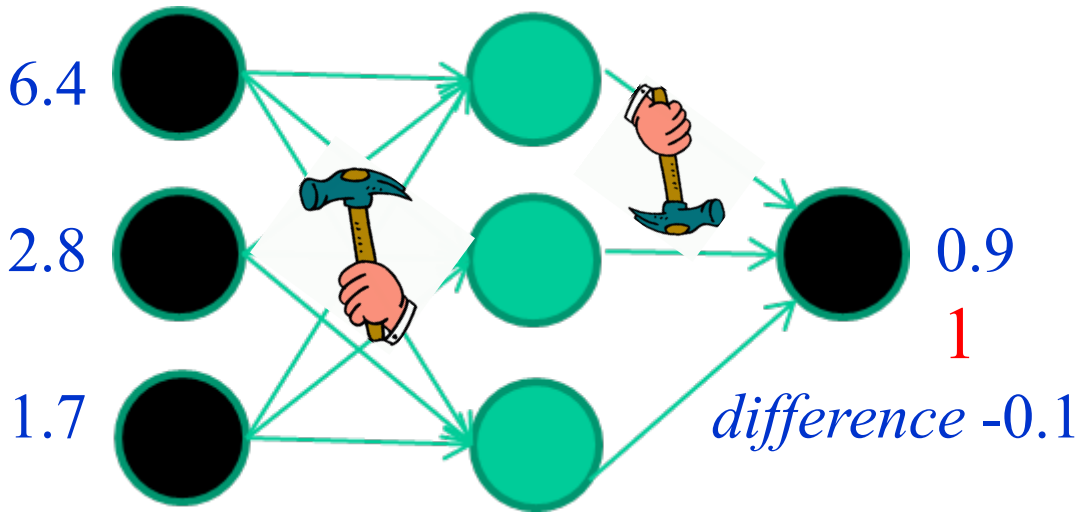


Illustration (decision boundary perspective)

Initial random weights

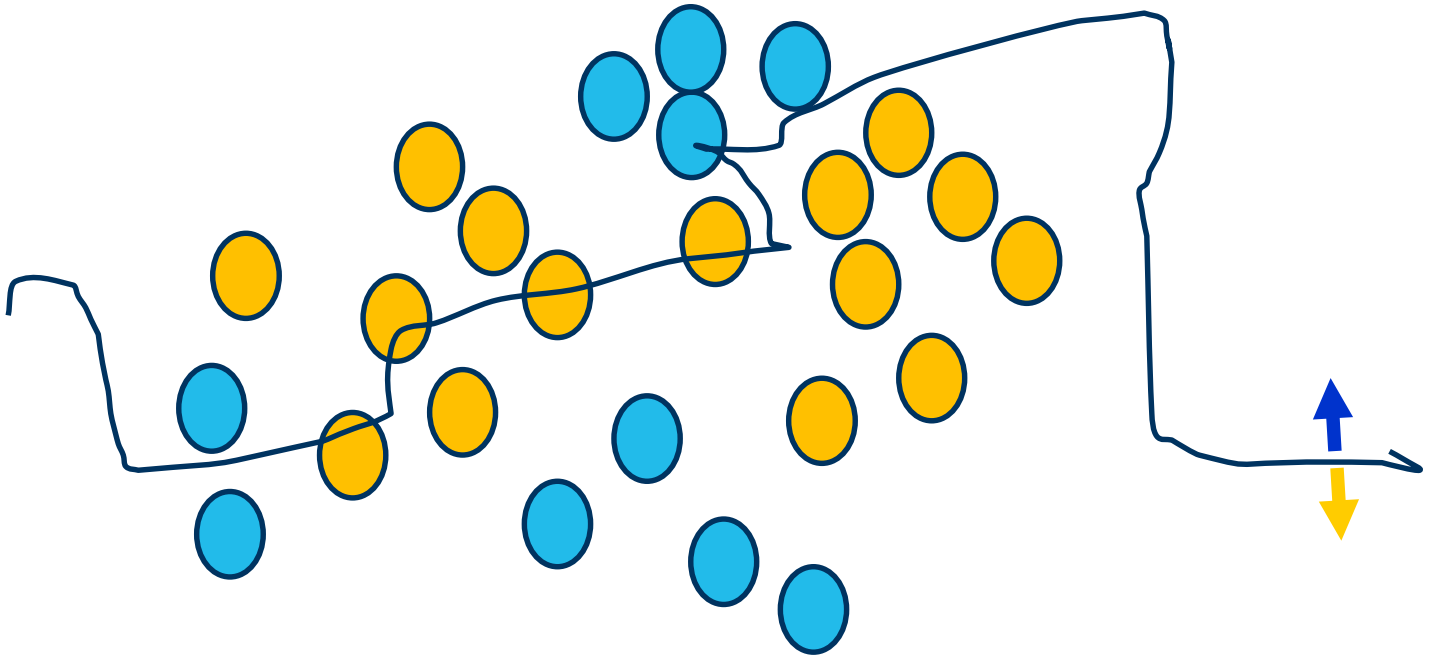


Illustration (decision boundary perspective)

Present a training instance / adjust the weights

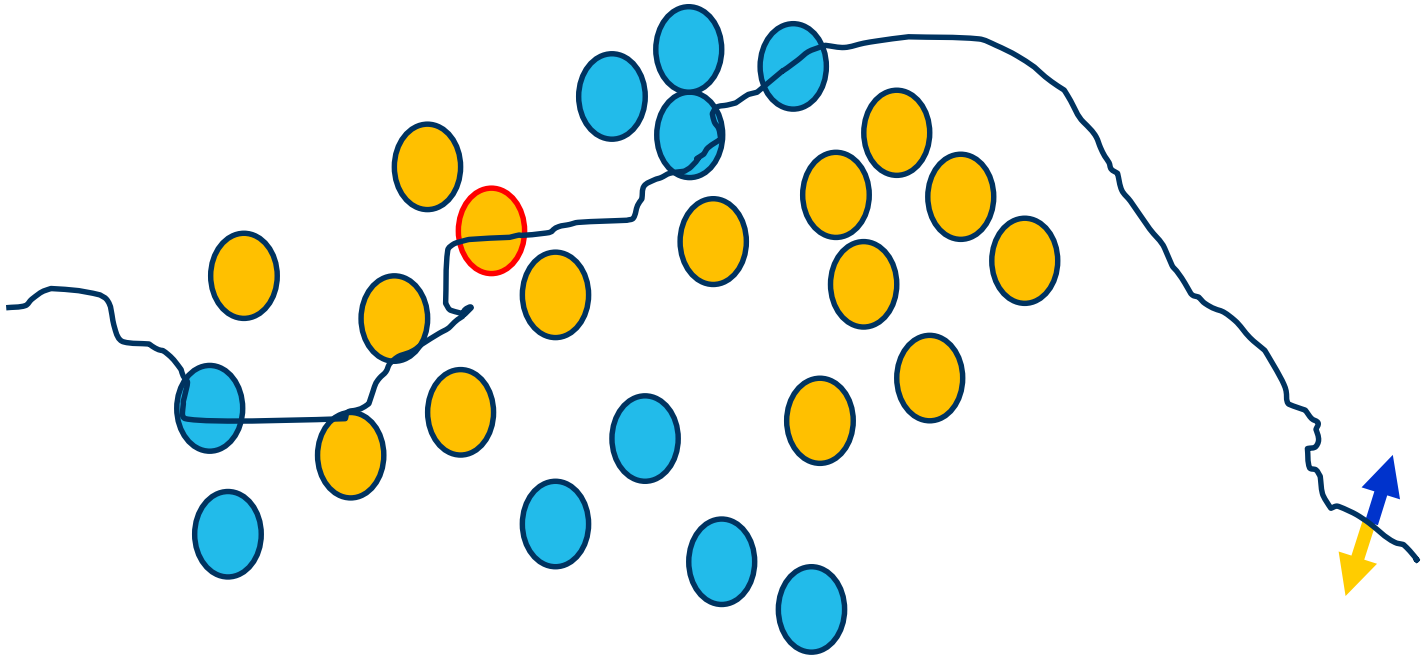


Illustration (decision boundary perspective)

Present a training instance / adjust the weights

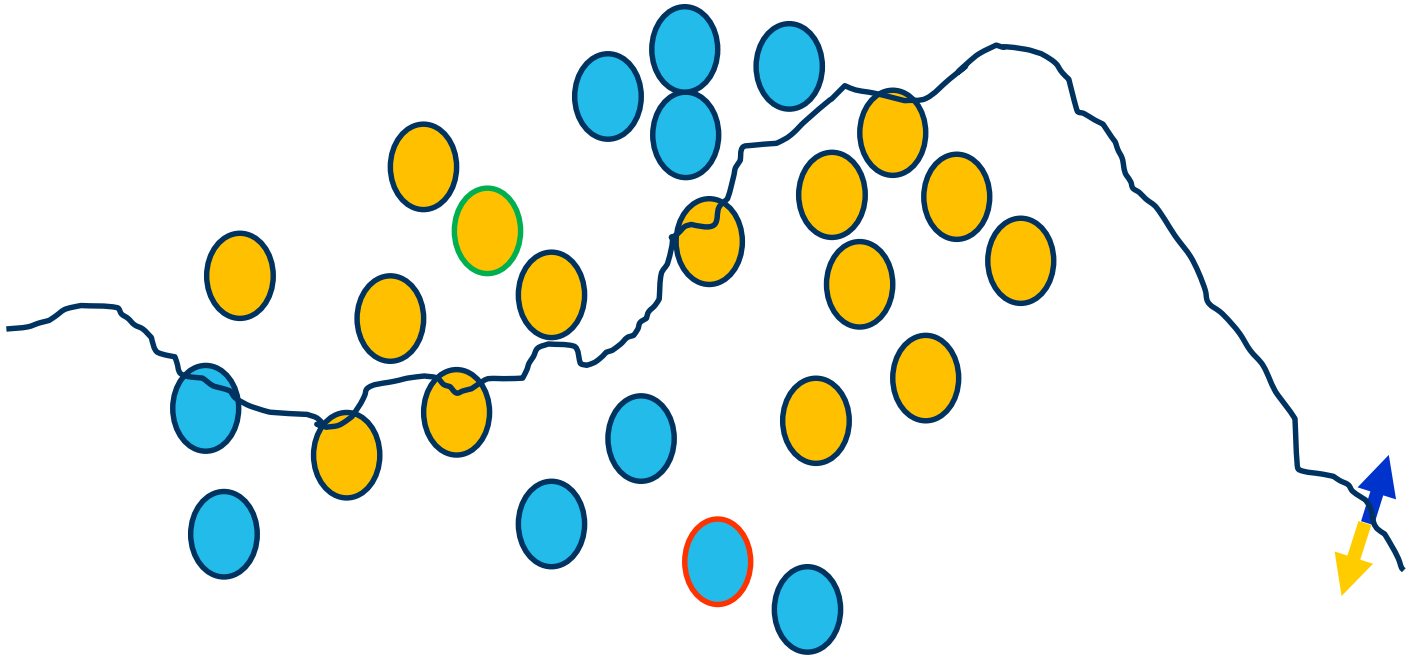


Illustration (decision boundary perspective)

Present a training instance / adjust the weights

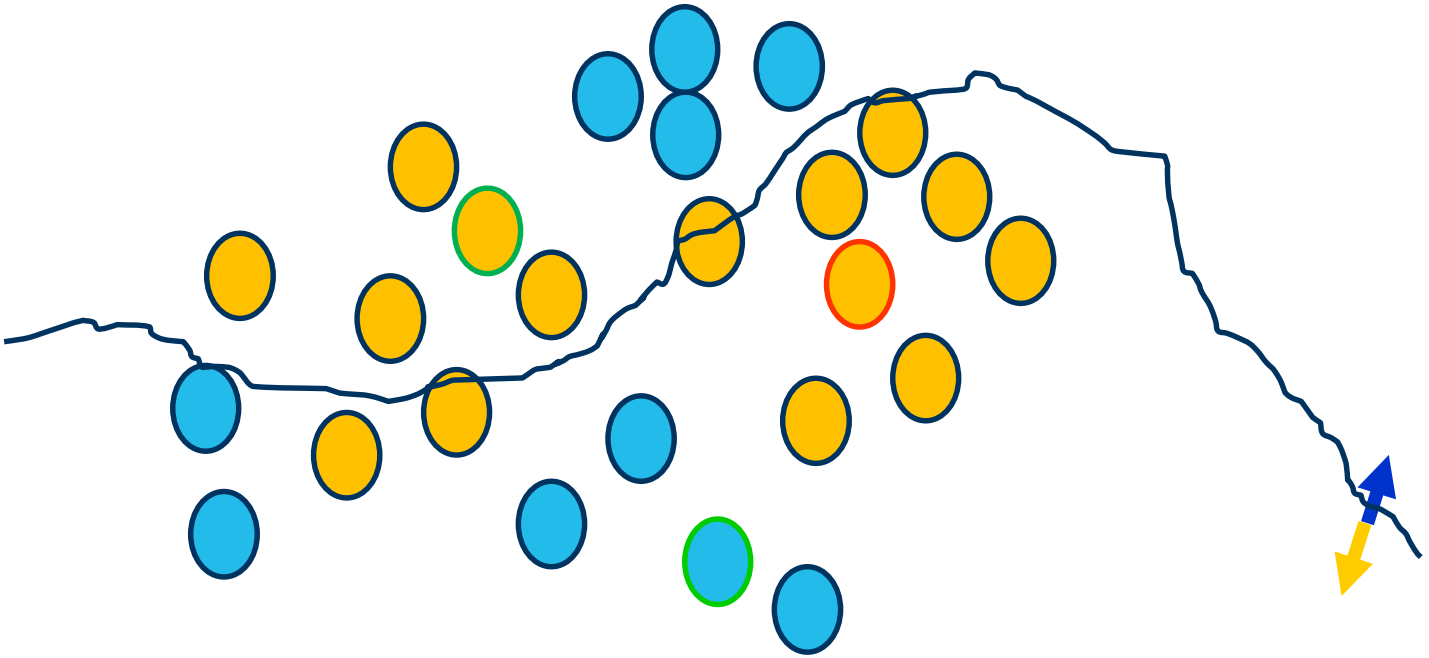


Illustration (decision boundary perspective)

Present a training instance / adjust the weights

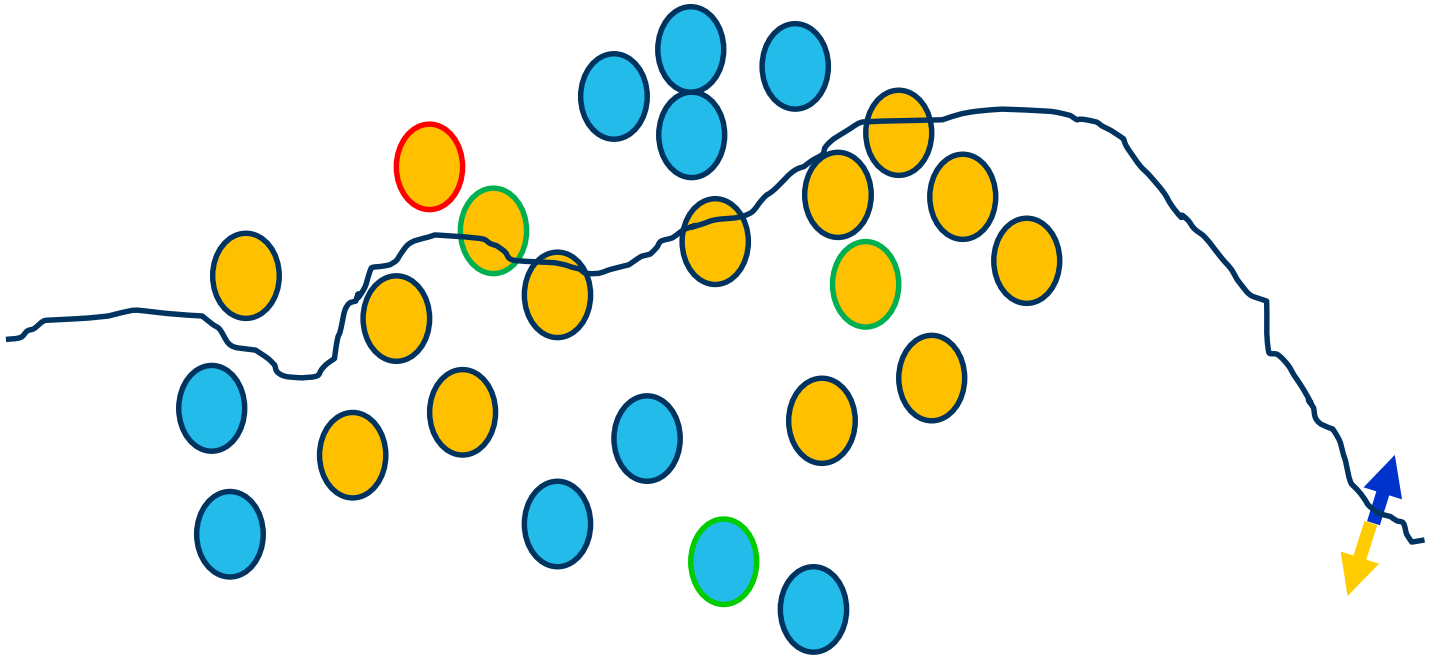
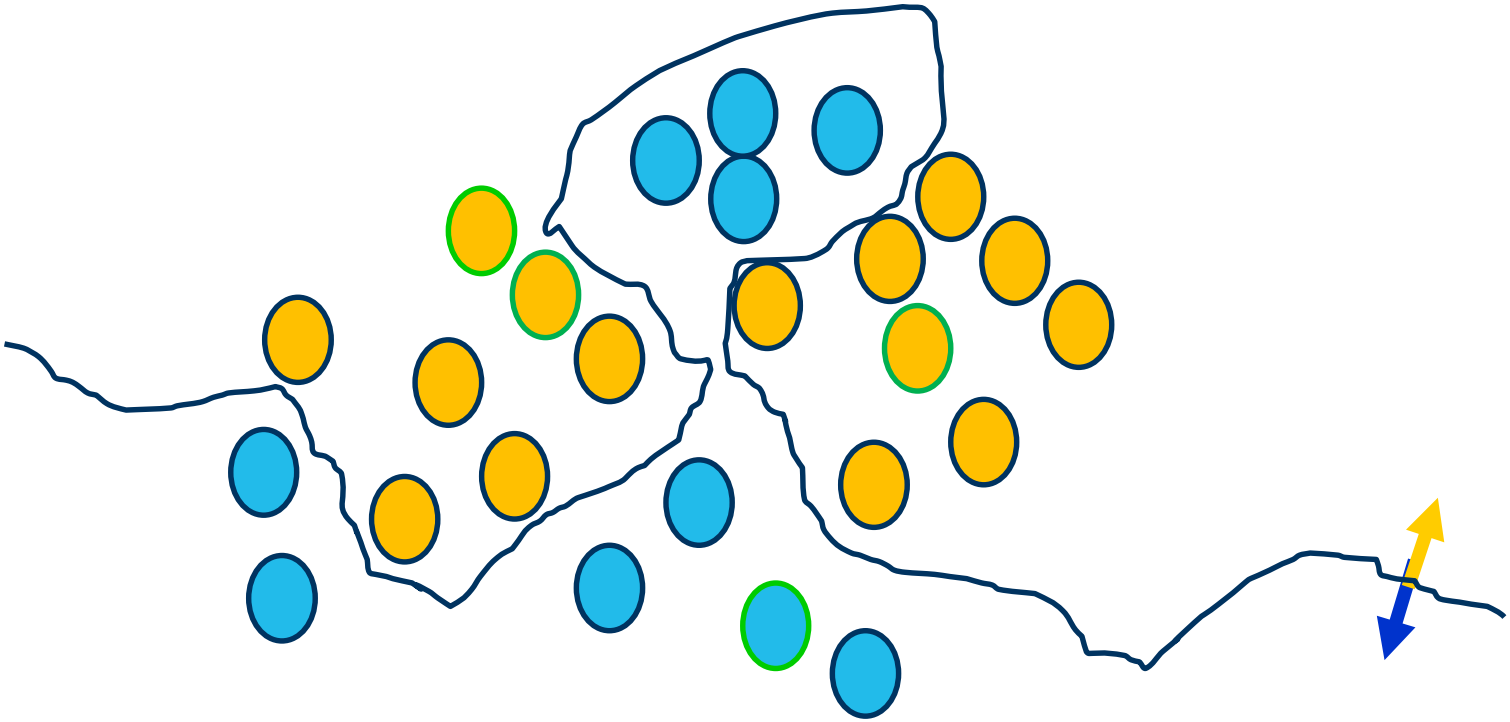


Illustration (decision boundary perspective)

Eventually …



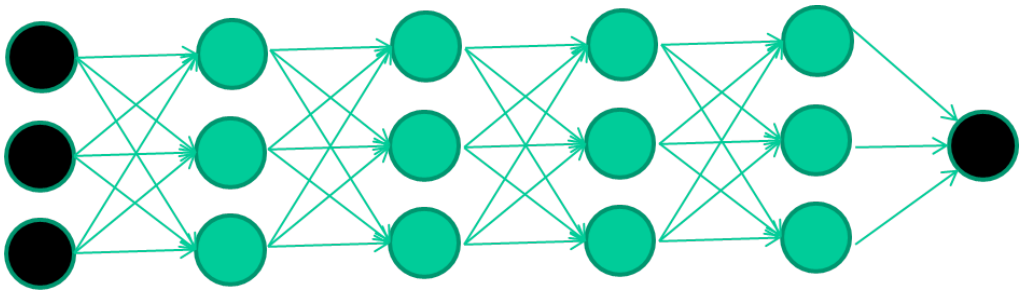
What's next ?

Deep Neural Networks (Brief overview)

Deep Learning (brief overview)

- Short answer: using a neural network with several hidden layers.
- These hidden layers do feature identification and processing.
- Various deep network architectures.

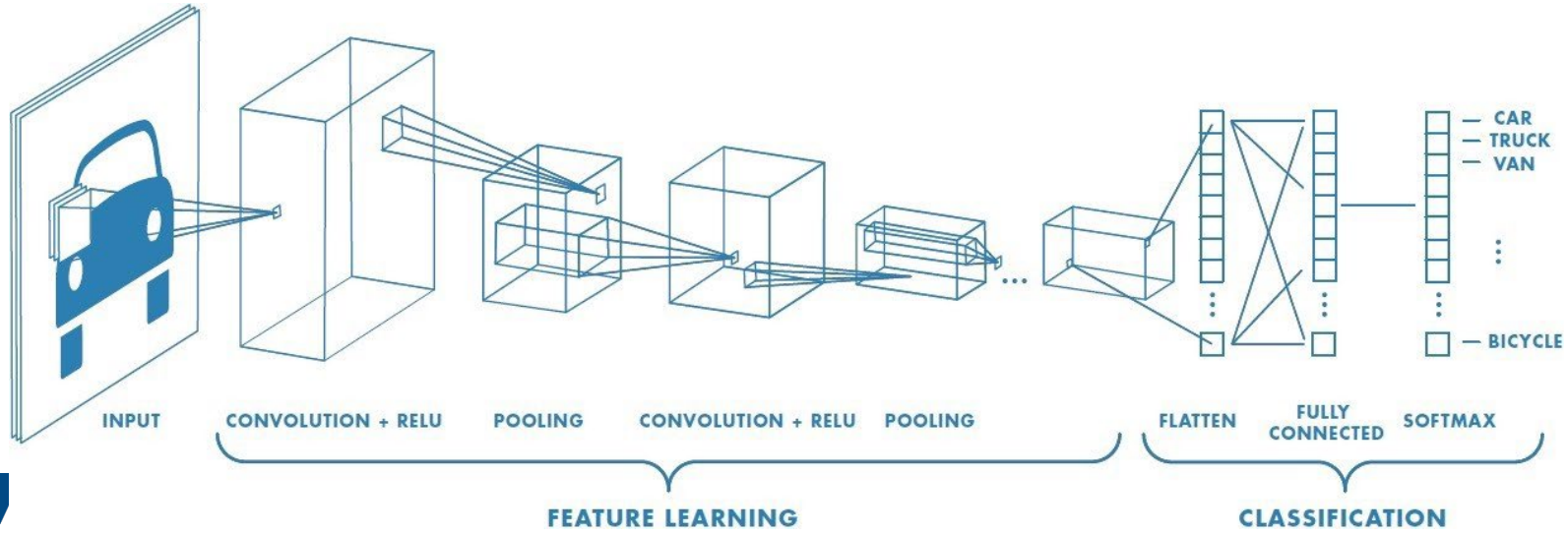
- Some examples:
  - **Feedforward (deep) neural networks**
  - Convolutional (deep) neural networks - CNN
  - Recurrent (deep) neural networks - RNN



Deep Learning (brief overview)

- Short answer: using a neural network with several hidden layers.
- These hidden layers do feature identification and processing.
- Various deep network architectures.

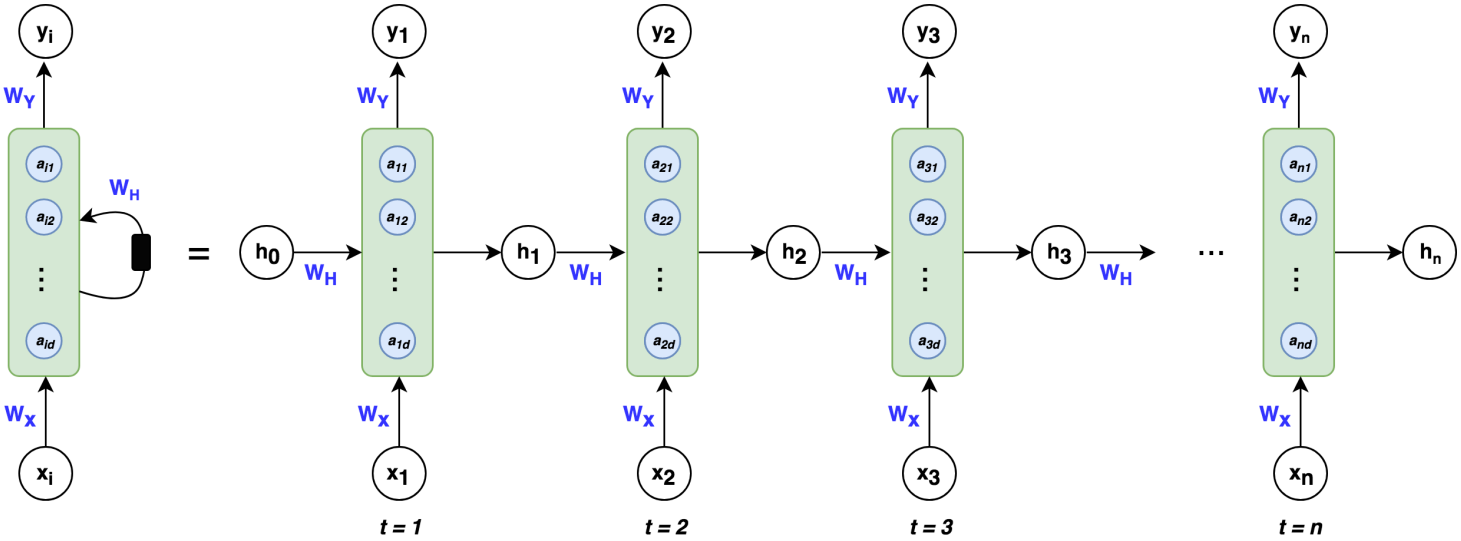
- Some examples:
  - Feedforward (deep) neural networks
  - **Convolutional (deep) neural networks - CNN**
  - Recurrent (deep) neural networks - RNN



Deep Learning (brief overview)

- Short answer: using a neural network with several hidden layers.
- These hidden layers do feature identification and processing.
- Various deep network architectures.

- Some examples:
  - Feedforward (deep) neural networks
  - Convolutional (deep) neural networks - CNN
  - **Recurrent (deep) neural networks - RNN**



Deep Learning (brief overview)

- Short answer: using a neural network with several hidden layers.
- These hidden layers do feature identification and processing.
- Various deep network architectures.

- Some examples:
  - Feedforward (deep) neural networks
  - Convolutional (deep) neural networks - CNN
  - Recurrent (deep) neural networks - RNN

You will learn more about this in the **Deep Learning course** later if you take it.

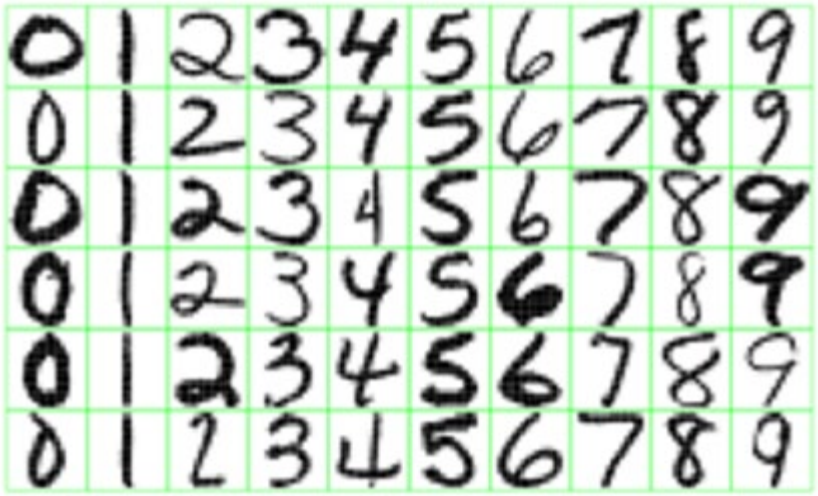
Why deep neural networks ?

- Units in hidden layers become self-organized feature detectors/extractors.

- Nodes in deeper layers learn higher level intermediate features.

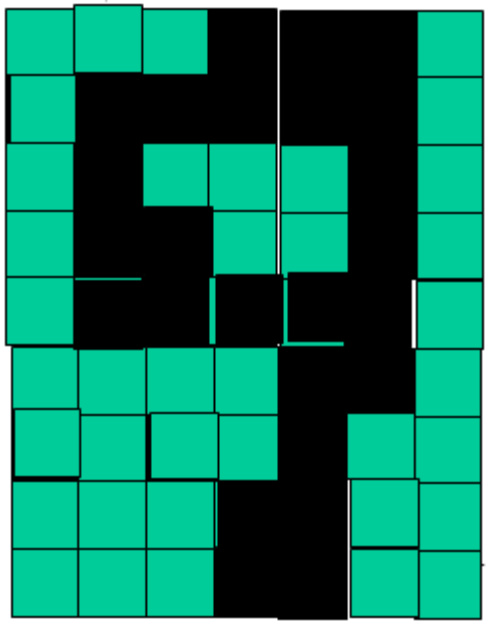
- Intermediate features are important for difficult classification tasks …

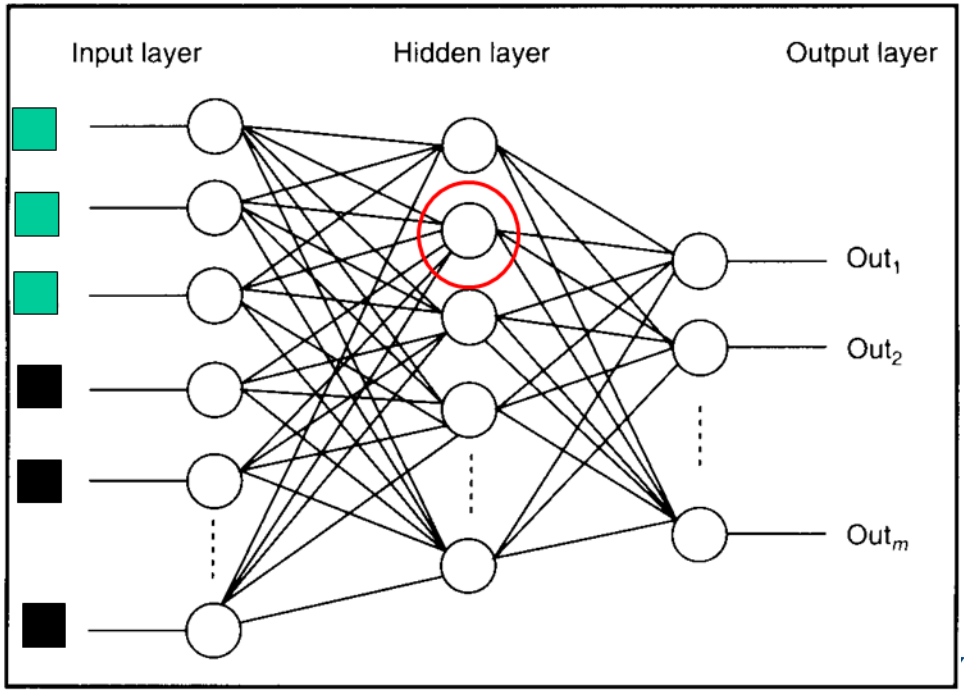
Example – why multiple layers?



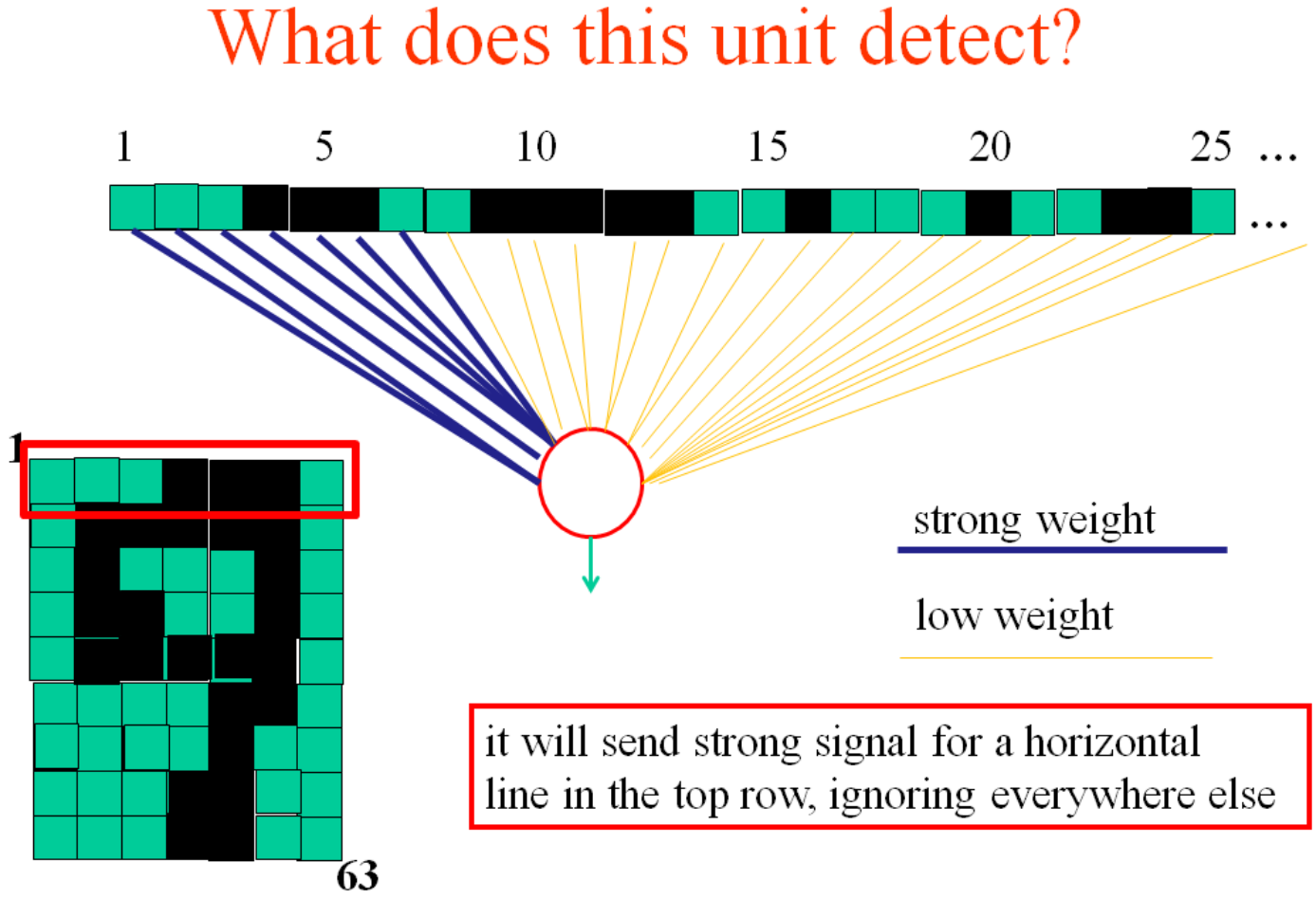
Example of handwritten digits recognition

What is this unit doing?

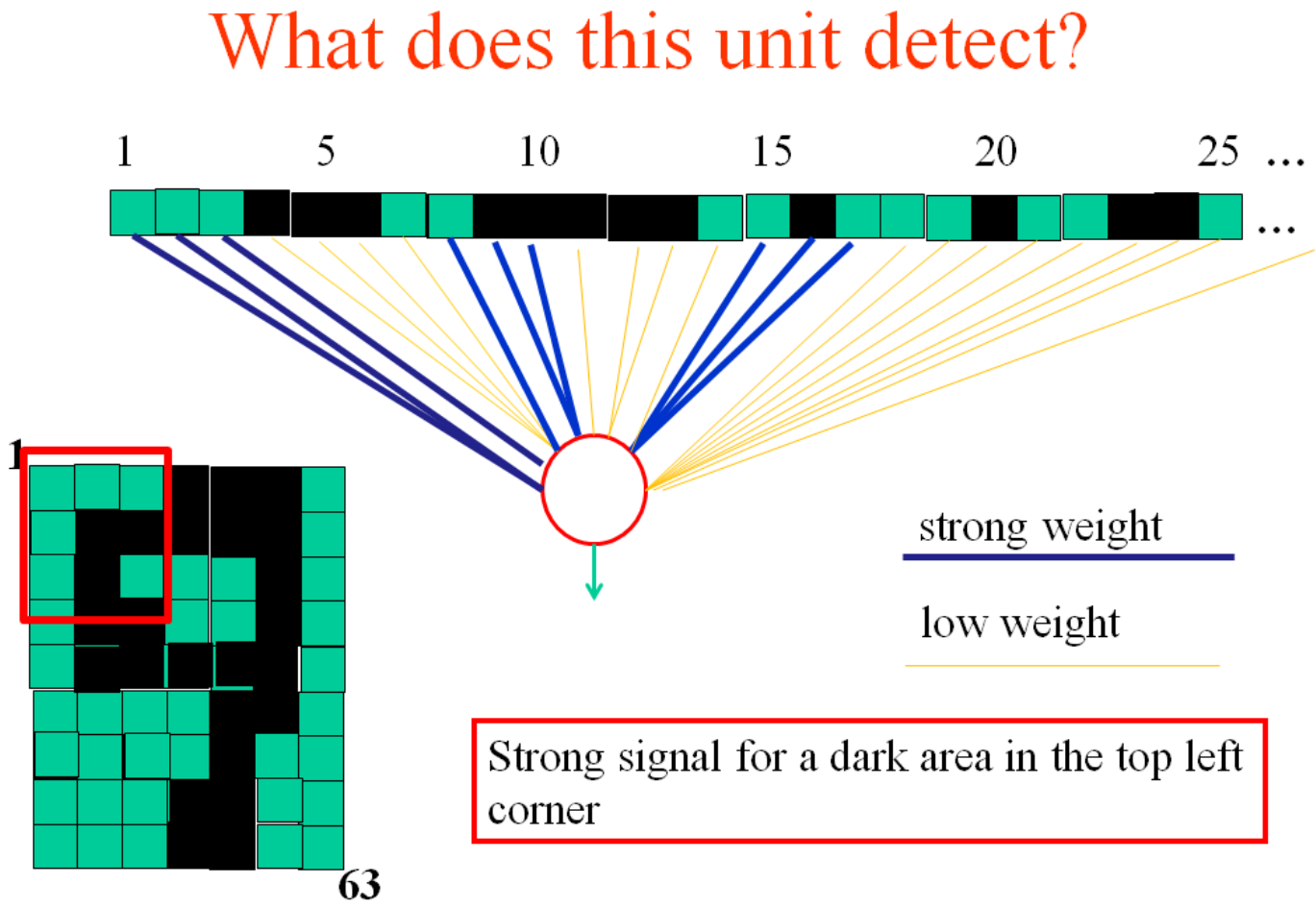




Example – why multiple layers?

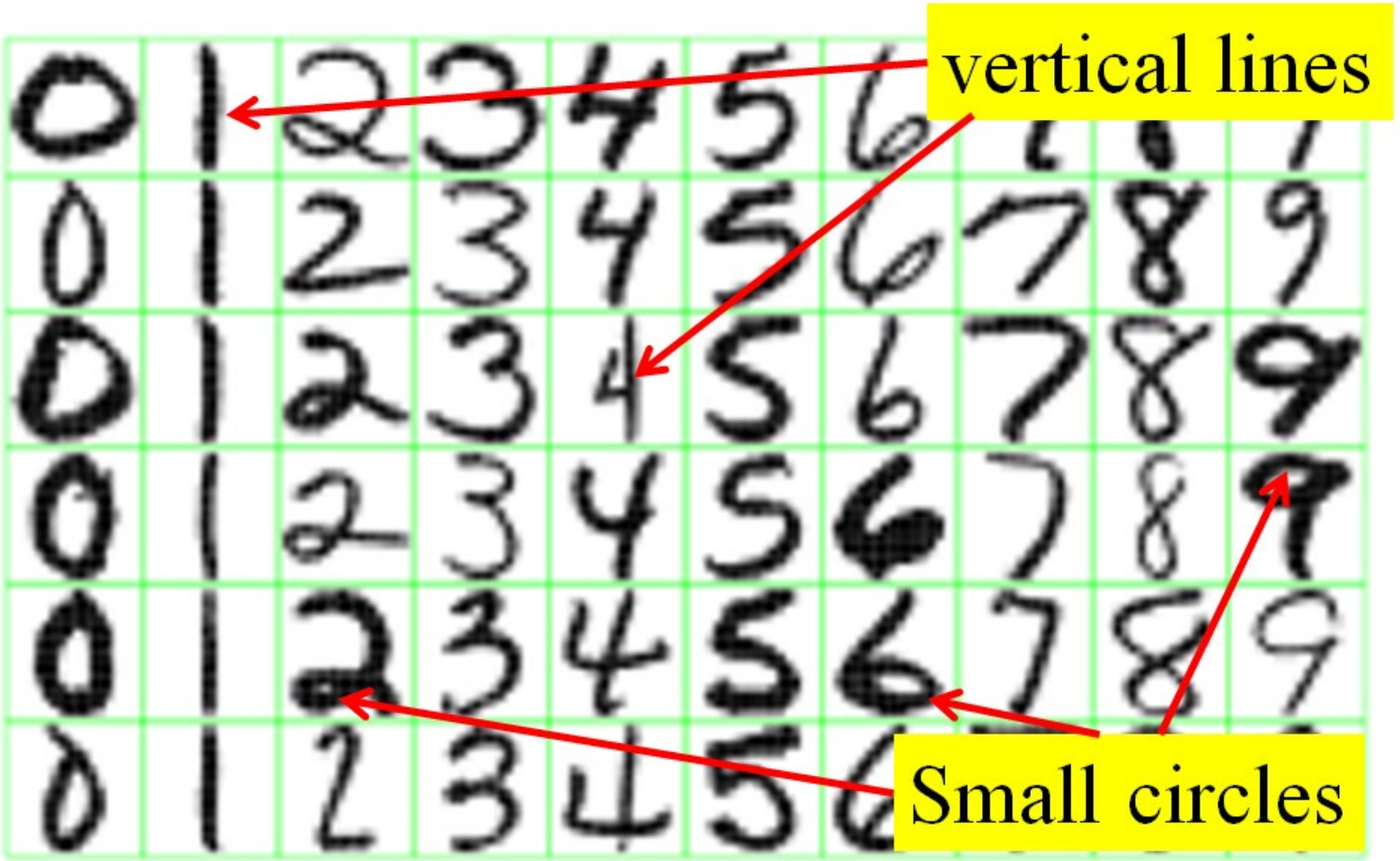


Example – why multiple layers?

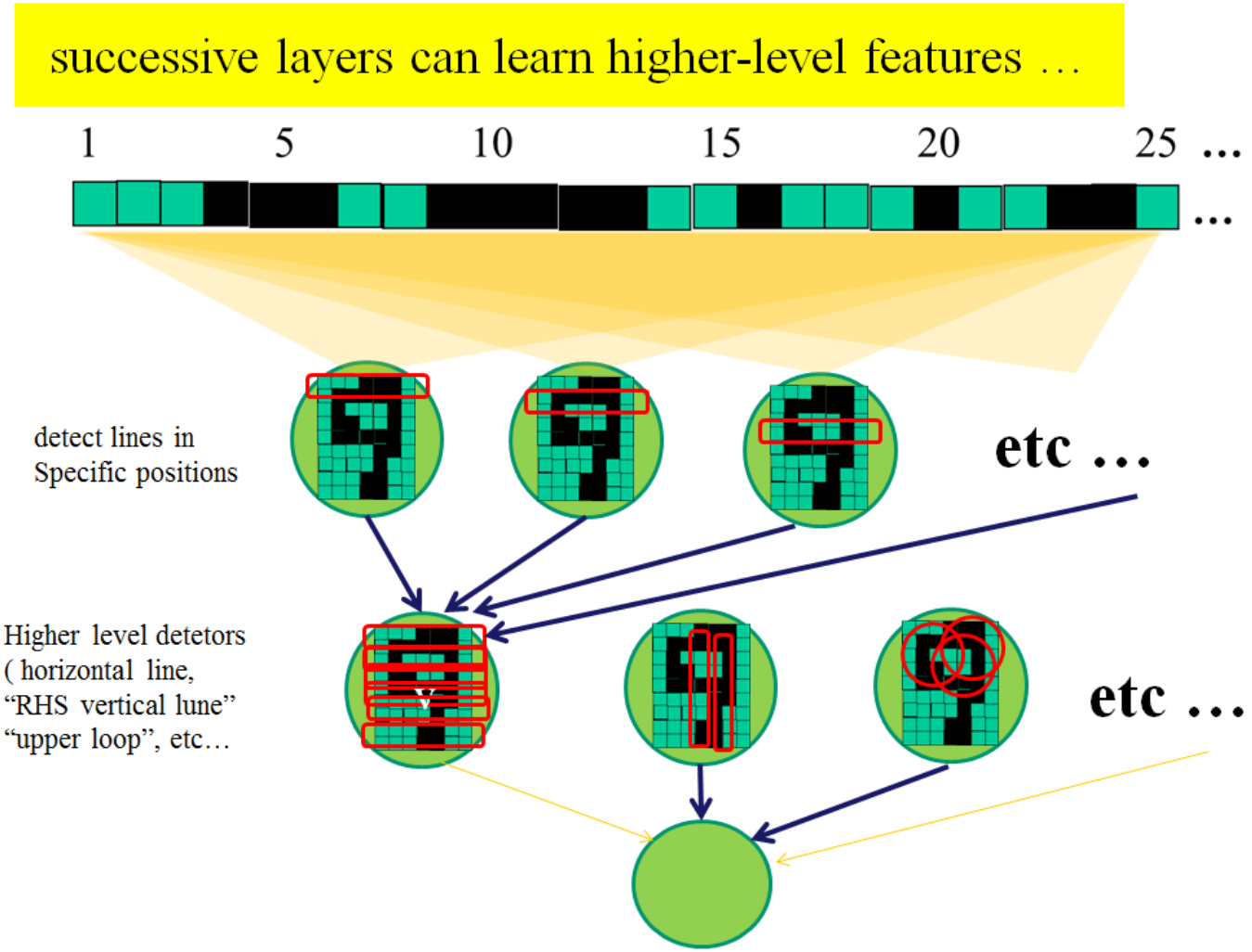


Example – why multiple layers?

These are some low level features …

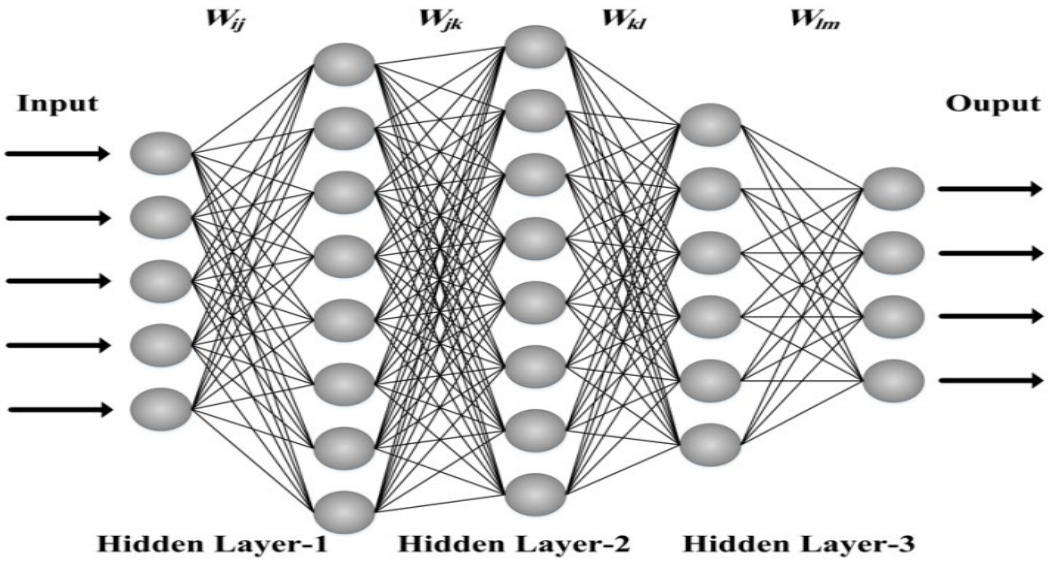


Example – why multiple layers?



So multiple layers make sense …

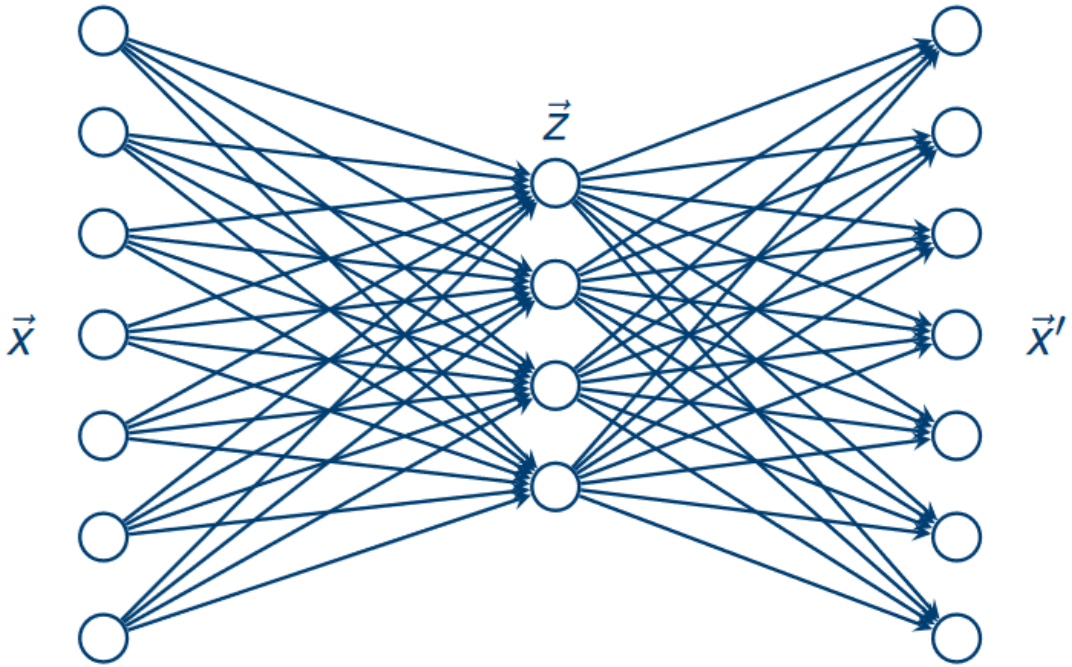
- Many-layer neural network architectures are capable of learning higher and higher level features …



More about neural networks …

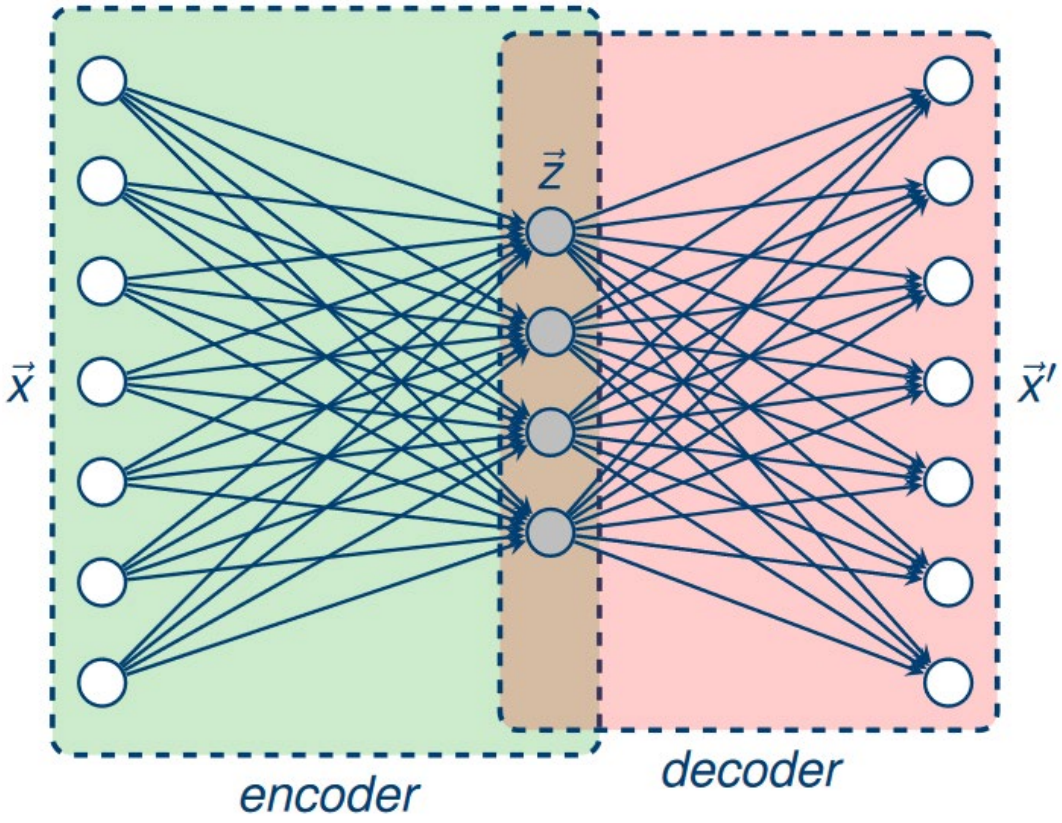
Unsupervised feature extraction with autoencoders

- Idea: what if the output of our neural network is the same as the input?



Unsupervised feature extraction with autoencoders

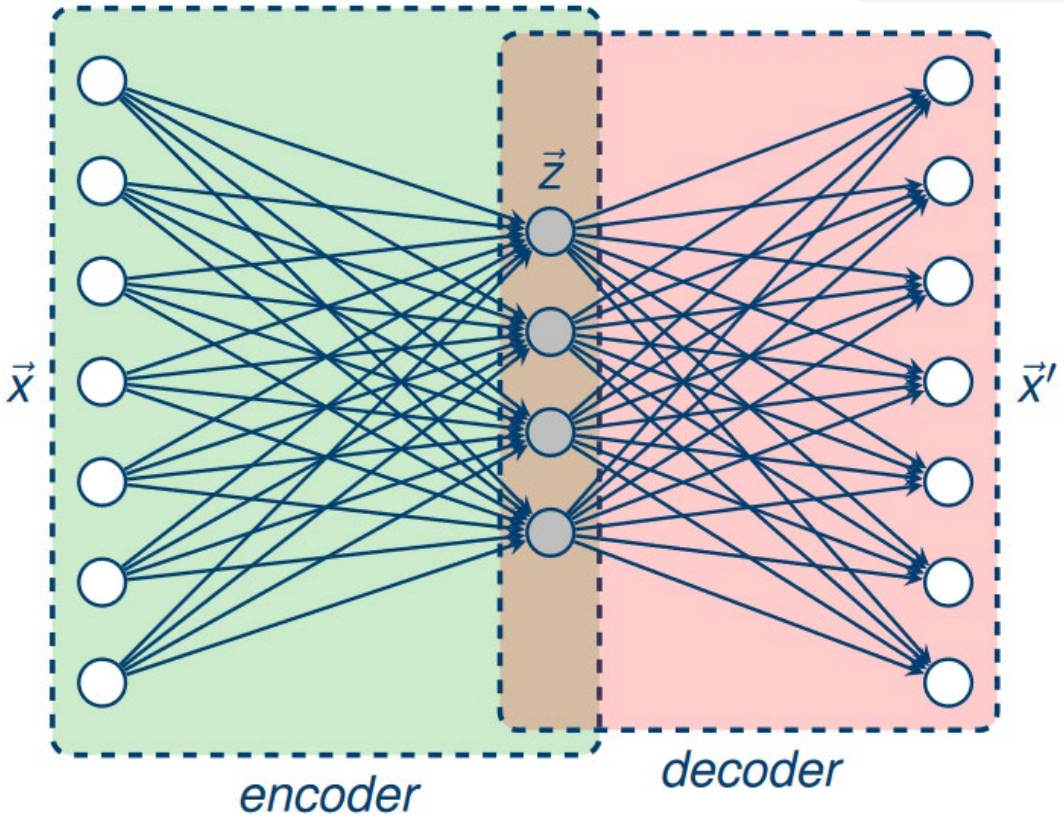
- Idea: what if the output of our neural network is the same as the input?



Unsupervised feature extraction with autoencoders

- Idea: what if the output of our neural network is the same as the input?

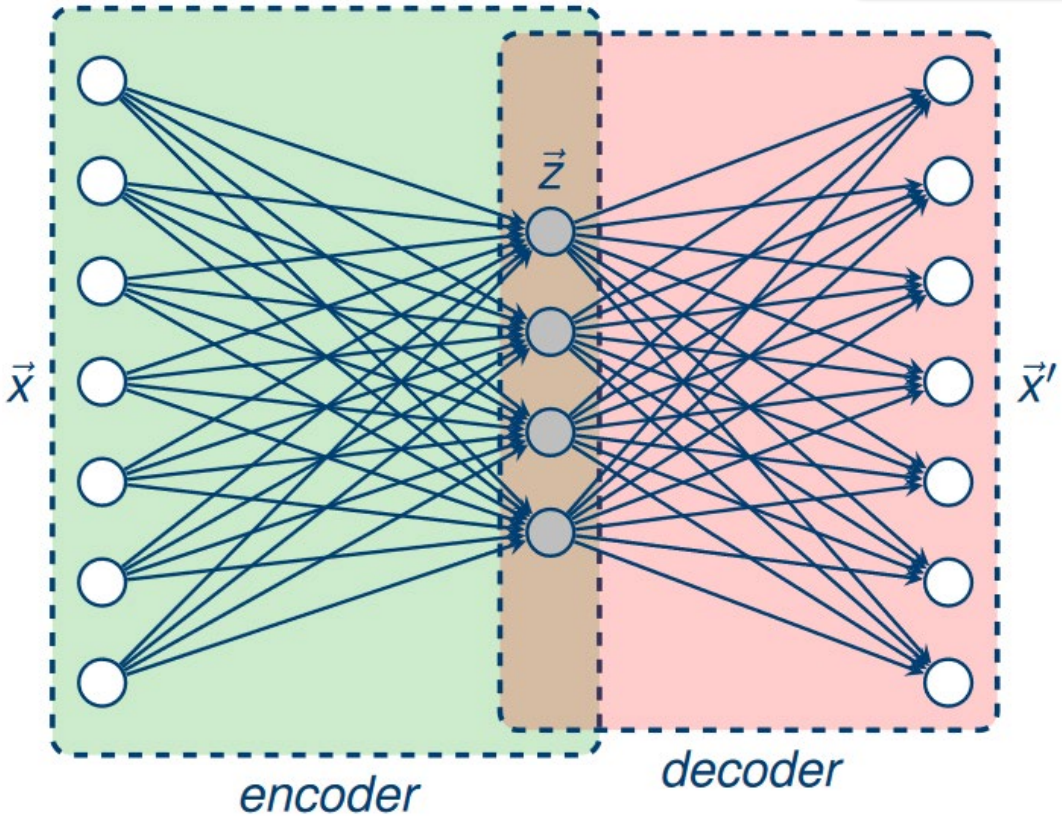
$$\underbrace{\text{decoder}\big(\underbrace{\text{encoder}(\boldsymbol{X})}_{\boldsymbol{Z}}\big)}_{\boldsymbol{X'}} \approx \boldsymbol{X}$$



Unsupervised feature extraction with autoencoders

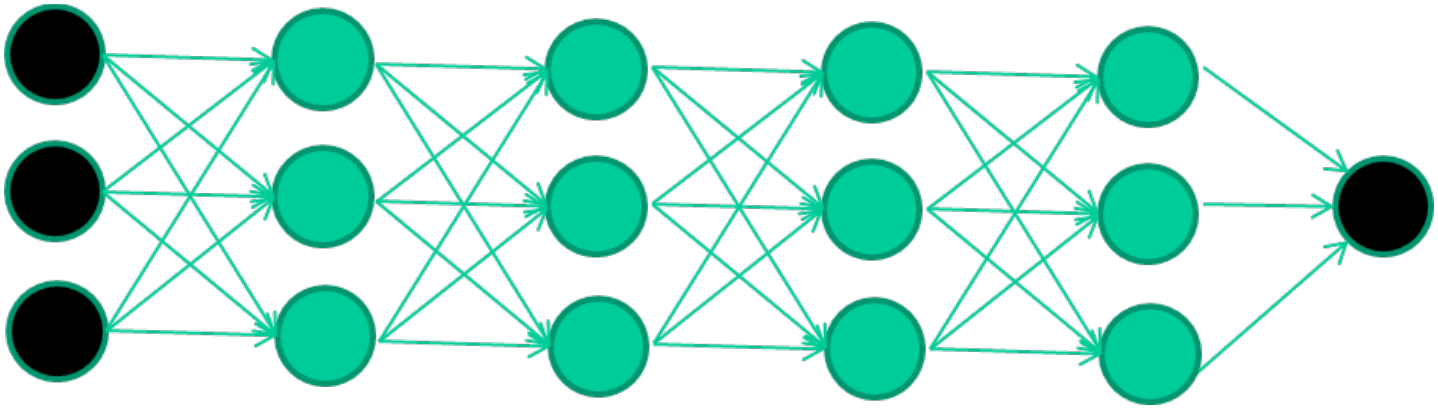
- Idea: what if the output of our neural network is the same as the input?

$$\underbrace{\text{decoder}\big(\underbrace{\text{encoder}(\boldsymbol{X})}_{\boldsymbol{Z}}\big)}_{\boldsymbol{X'}} \approx \boldsymbol{X}$$



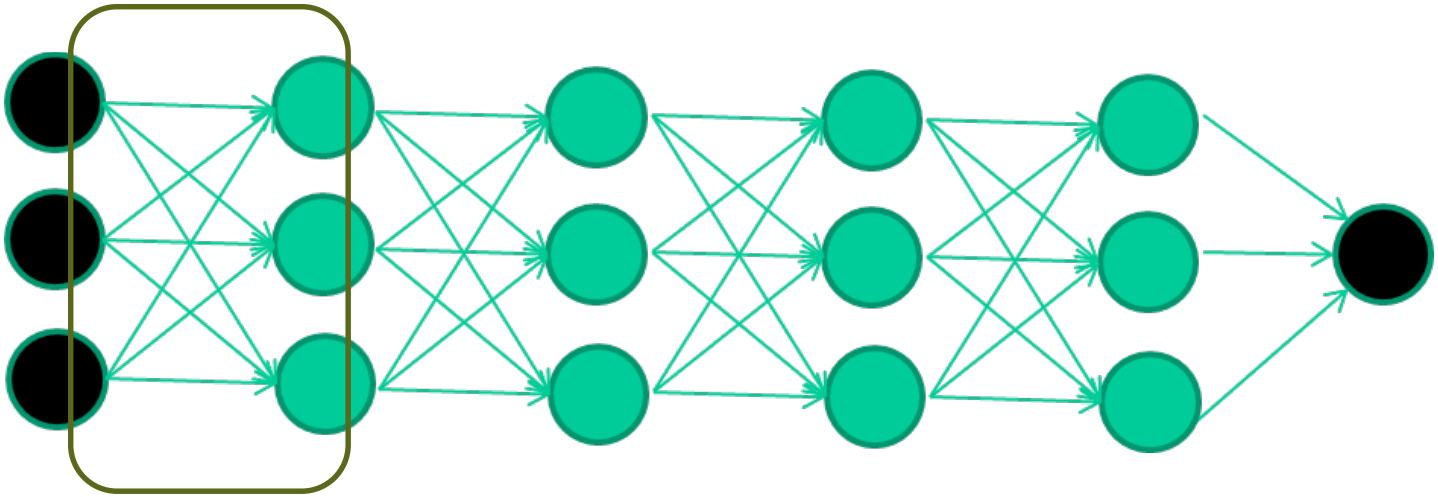
- An auto-encoder is trained as usual, but to reproduce the input.
- By making this happen with much fewer hidden units, this forces the hidden layer units to become good feature detectors.
  - We are able to reconstruct the input from the lower-dimensional (i.e. compressed) representation of the hidden layer.

Unsupervised feature learning - Example



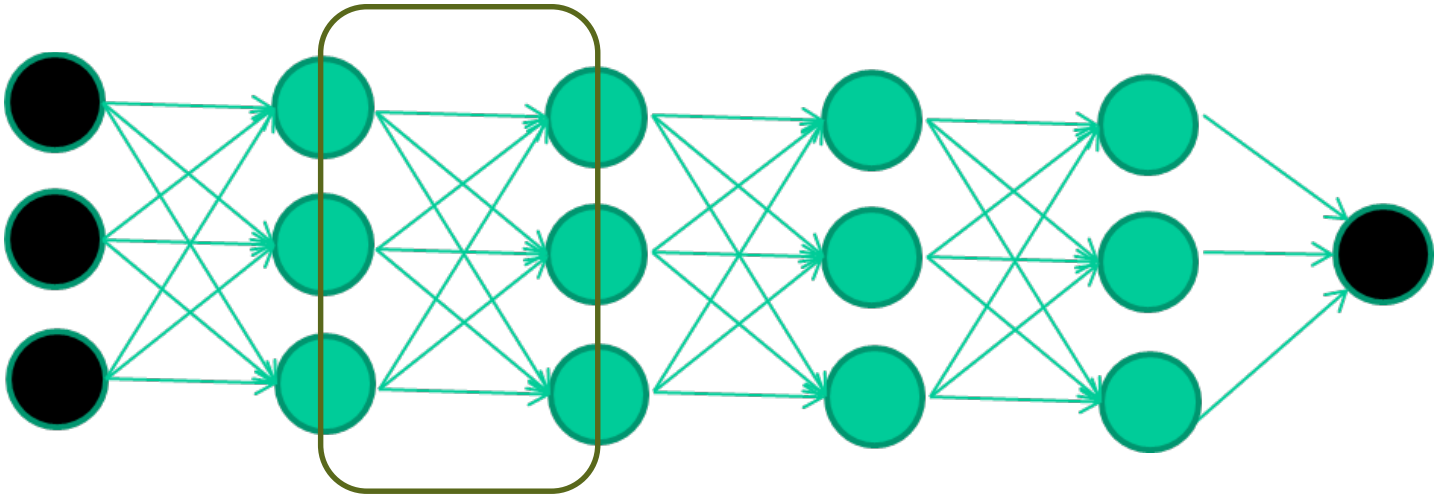
- The idea of unsupervised feature learning
- Simpler/better ways of training Deep neural networks

Unsupervised feature learning - Example



Train **this** layer first

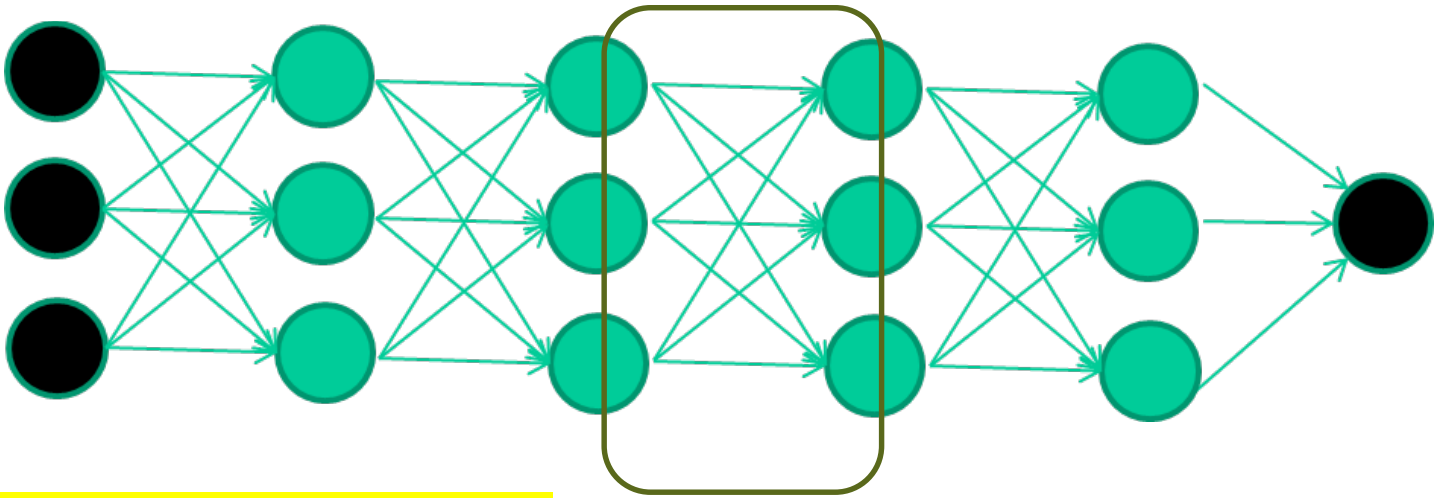
Unsupervised feature learning - Example



Train **this** layer first

then **this** layer

Unsupervised feature learning - Example

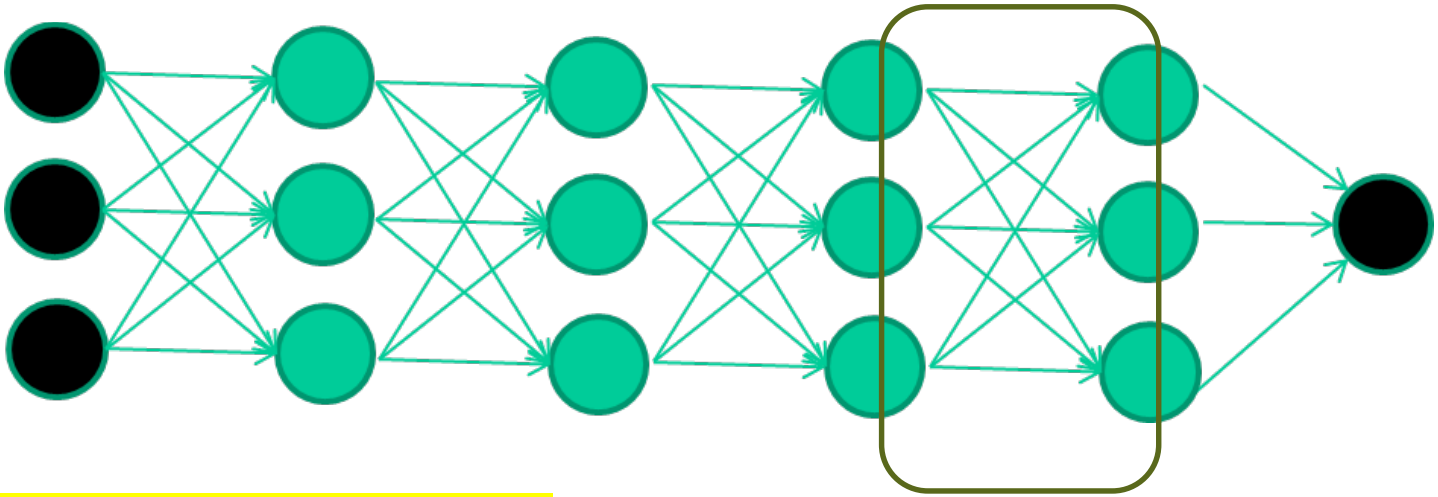


Train **this** layer first

then **this** layer

then **this** layer

Unsupervised feature learning - Example



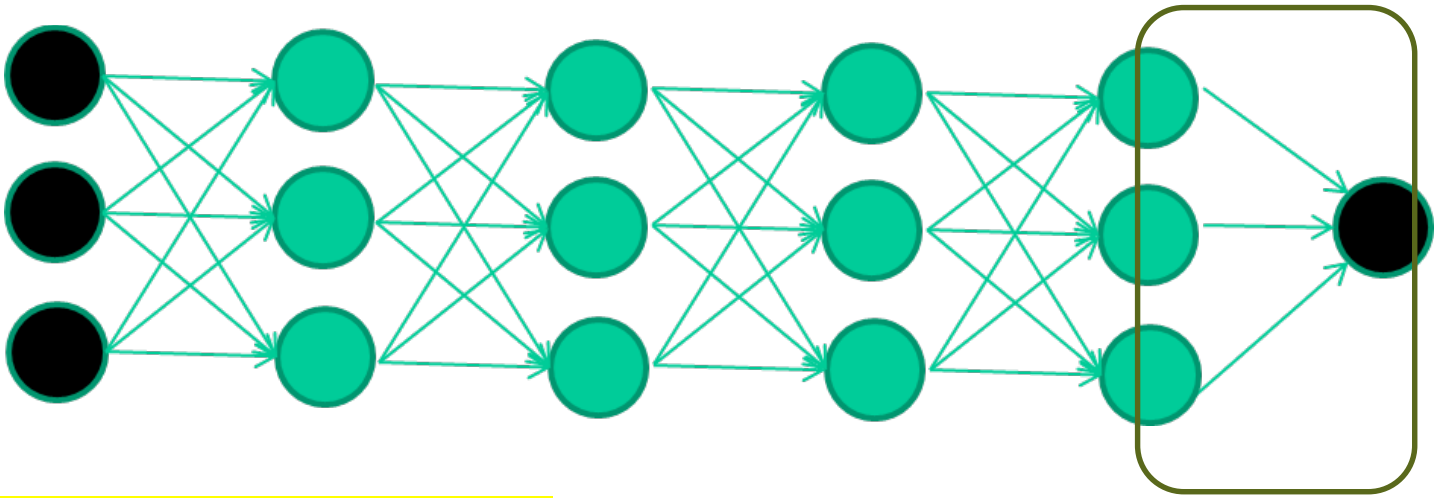
Train **this** layer first

then **this** layer

then **this** layer

then **this** layer

Unsupervised feature learning - Example



Train **this** layer first

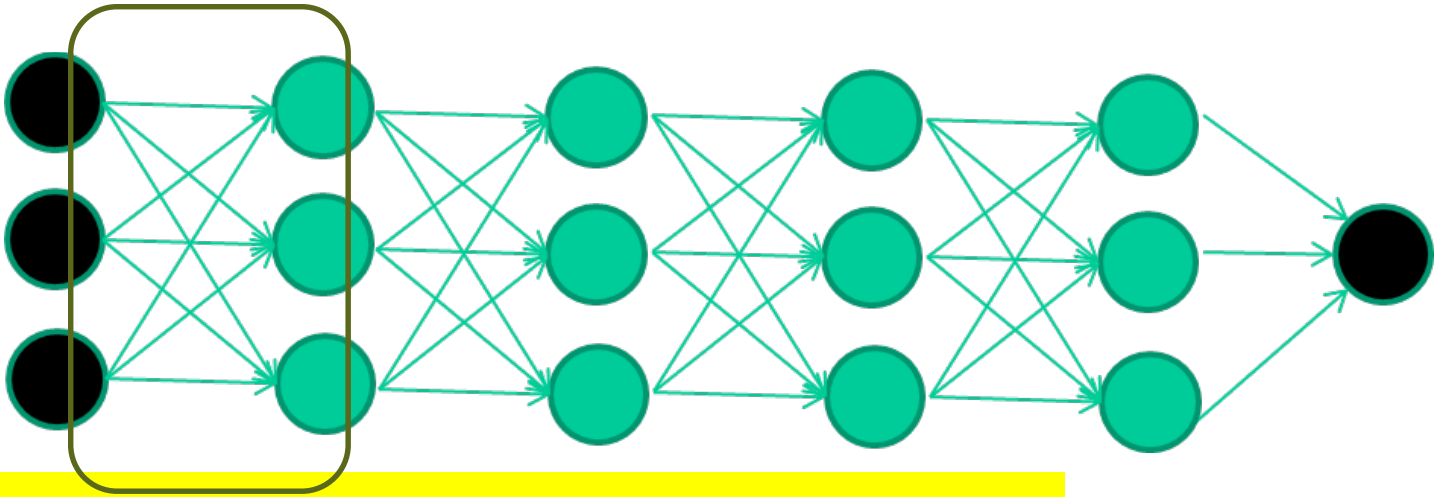
then **this** layer

then **this** layer

then **this** layer

finally **this** layer

Unsupervised feature learning - Example

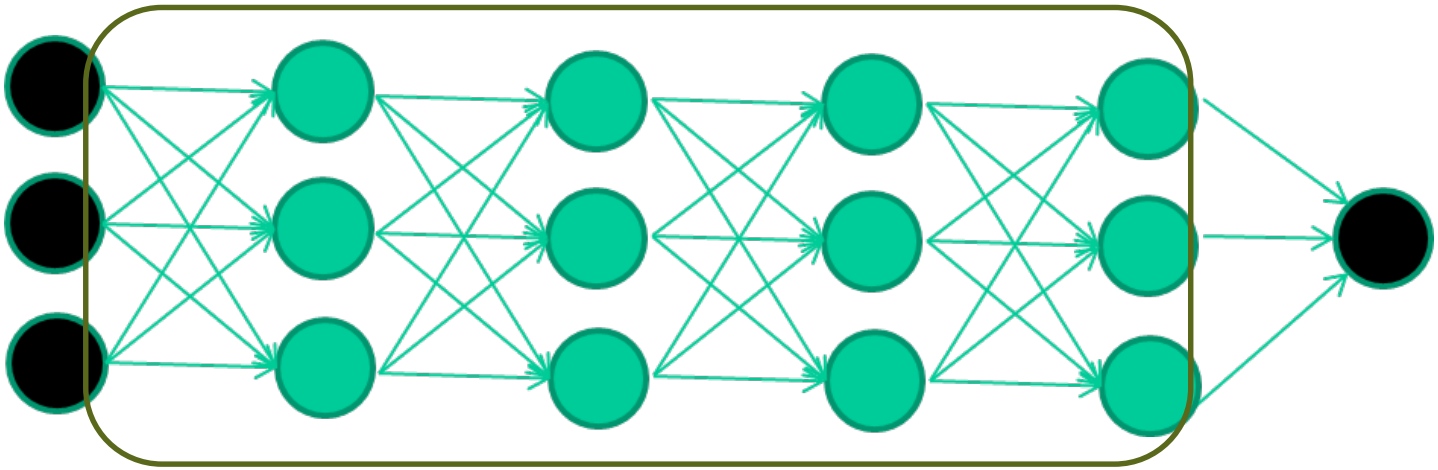


EACH of the (non-output) layers is trained to be an ***auto-encoder***

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

Unsupervised feature learning - Example

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



Unsupervised feature learning - Example

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