ieeta instituto de engenharia electrónica e telemática de aveiro

universidade de aveiro

Departamento de Eletrónica, Telecomunicações e Informática

Machine Learning

LECTURE 4:

NEURAL NETWORKS

Petia Georgieva
(petia@ua.pt)

universidade de aveiro

NEURAL NETWORKS- outline

1. NN - non-linear classifier
2. Neuron model: logistic unit
3. NN - binary versus multi-class classification
4. Cost function (with or without regularization)
5. NN learning - Error Backpropagation algorithm

Classification of non-linearly separable data

$x_1 =$ size of house
$x_2 =$ no. of bedrooms
$x_3 =$ no. of floors
$x_4 =$ age of house
$x_5 =$ average income in neighborhood
$x_6 =$ kitchen size
$\vdots$
x_{100}

Let we have 100 original features:

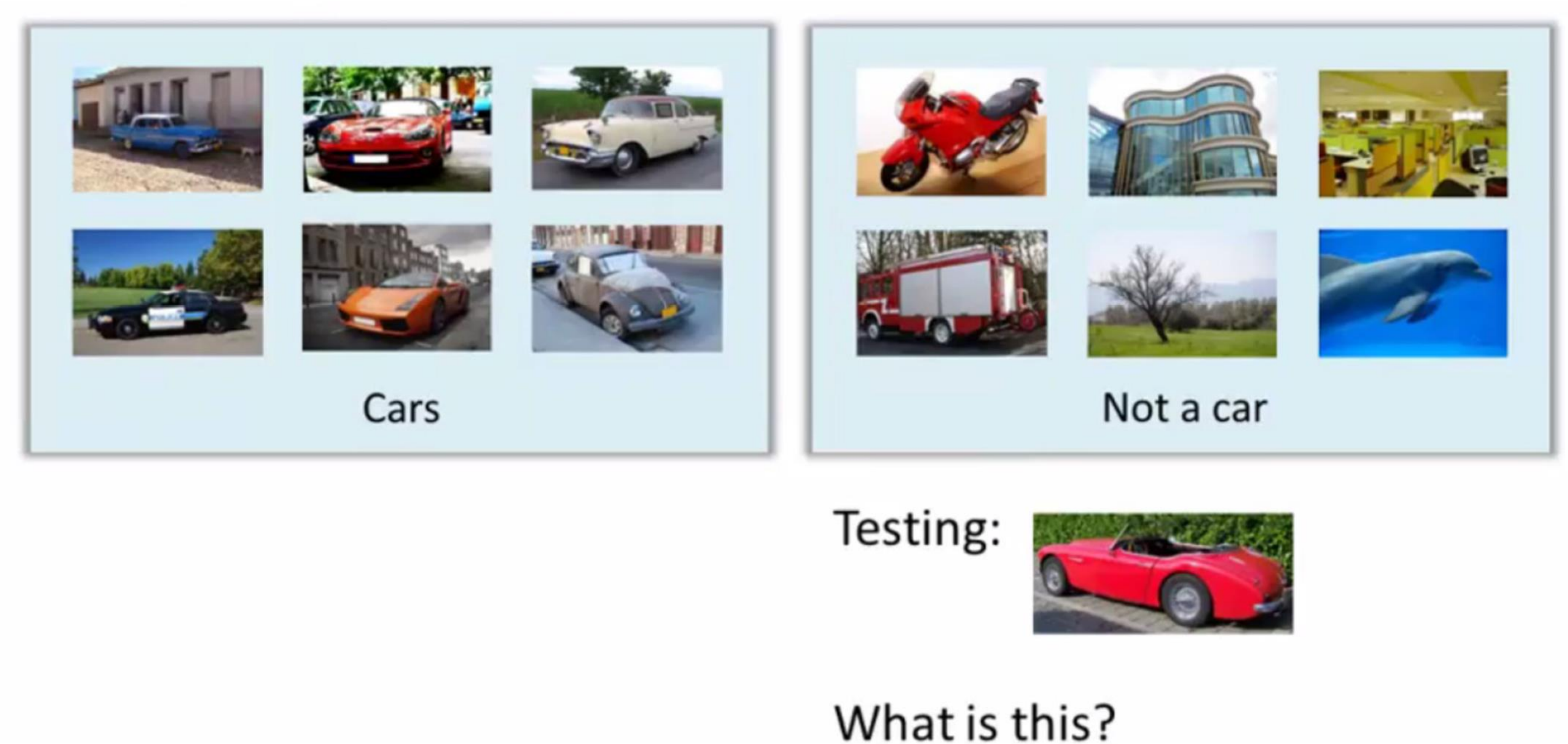
If using quadratic combinations of the features to get nonlinear decision boundary, we end up with 5000 features

If using cubic combinations of features => 170 000 features

Logistic regression is not efficient for such complex nonlinear models

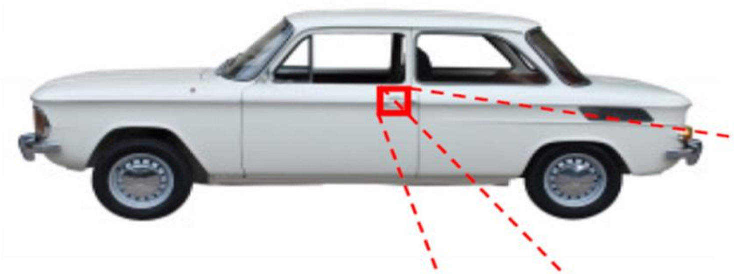
$$g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1 x_2 + \theta_4 x_1^2 x_2 + \theta_5 x_1^3 x_2 + \theta_6 x_1 x_2^2 + \ldots)$$

Computer vision: car detection



Computer vision

You see 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

For a small peace of the car image we may have too many features (pixels)

Computer vision: object detection

50 x 50 pixel images→ 2500 pixels

$n = 2500$ (7500 if RGB)

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

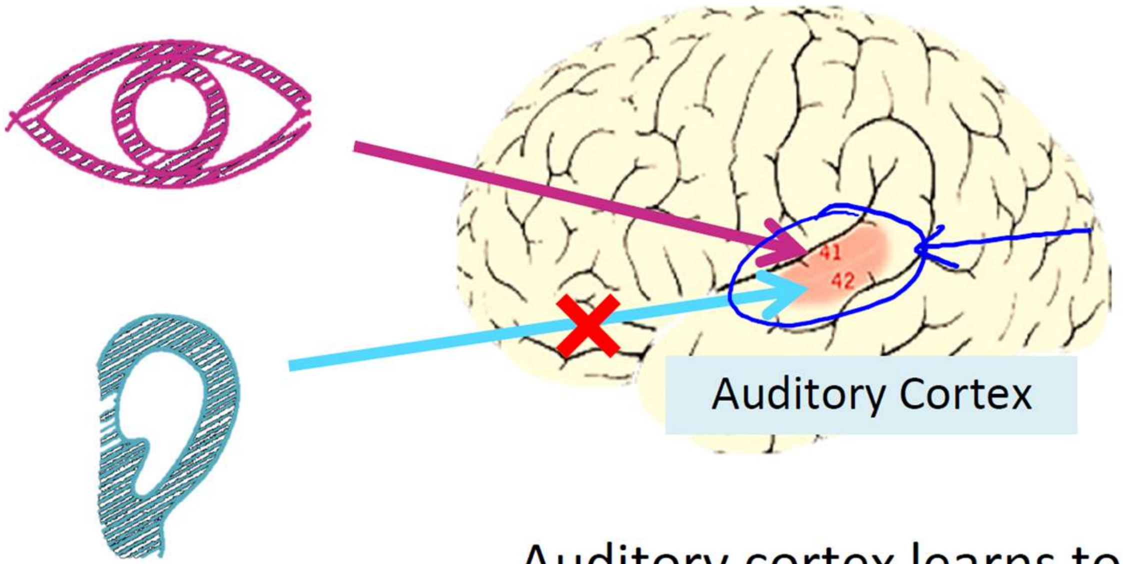
50 x 50 pixel images =>
2500 pixels (features) for a gray scale image
7500 pixels (features) for a RGB image

If using quadratic features => 3 million features

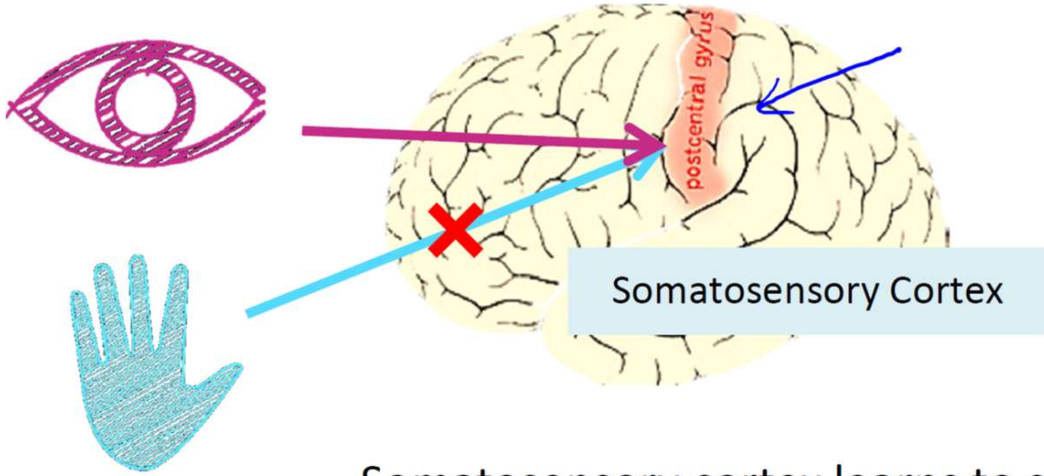
Logistic regression is not suitable for such complex nonlinear models.

Neural Networks fit better complex nonlinear models.

Brain experiments
(brain can learn from any sensor wired to it)



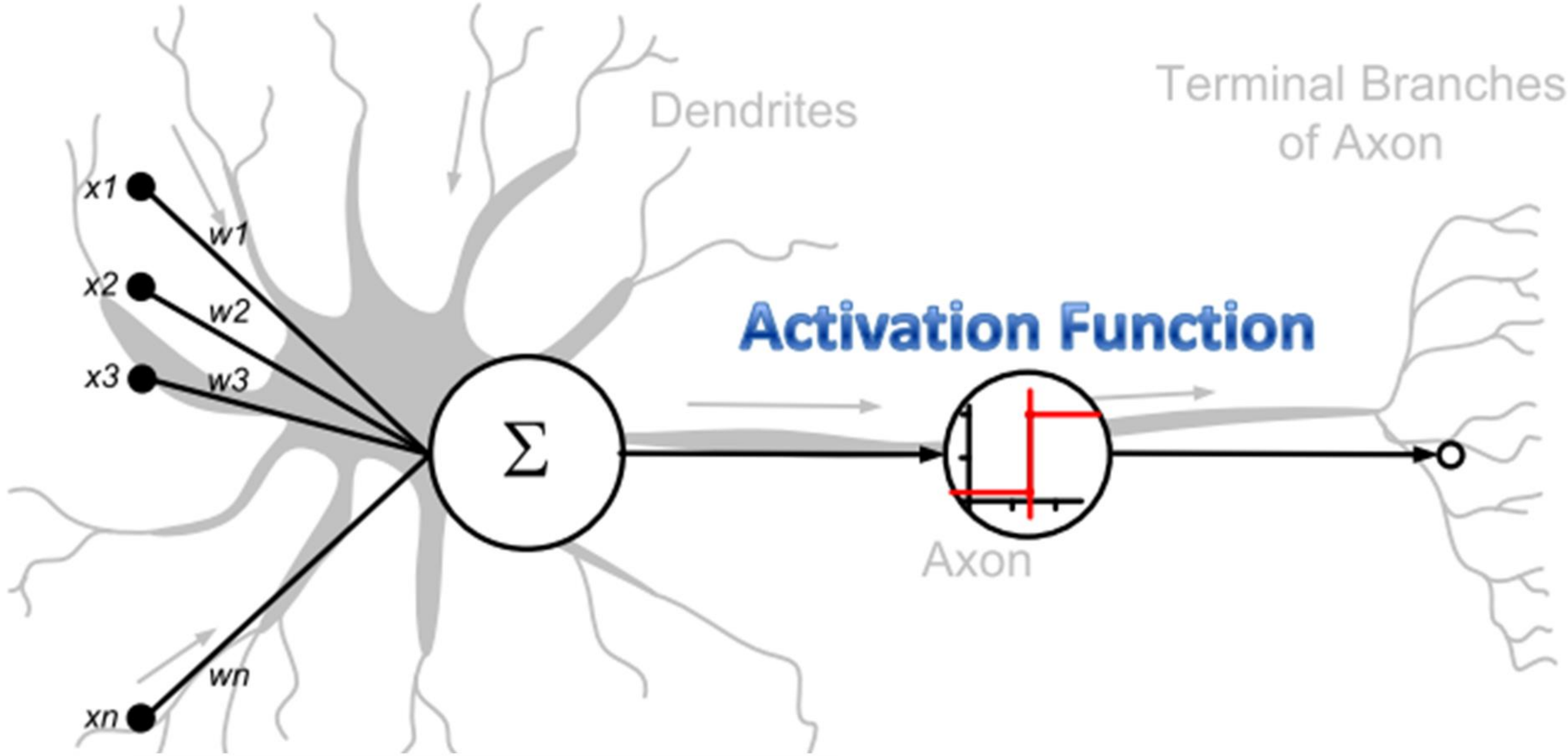
Auditory cortex learns to see

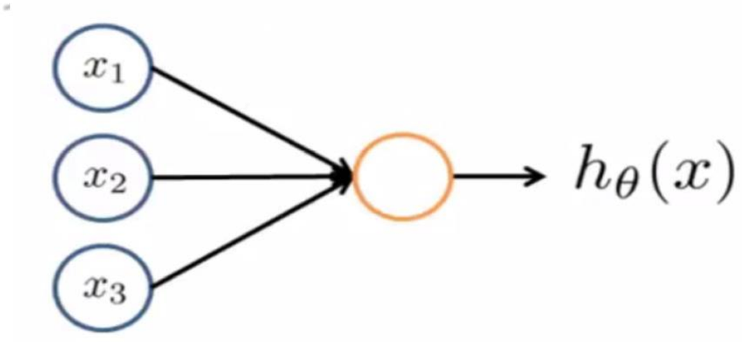


Somatosensory cortex learns to see

Neuron model

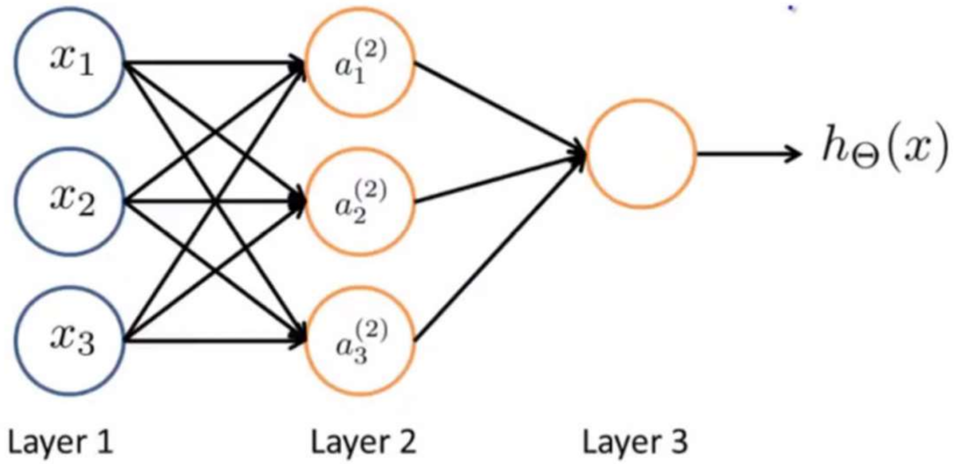
Origins: NN models inspired by biological neuron structures and computations.





Model of artificial neuron (unit, node)

Neural Network



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

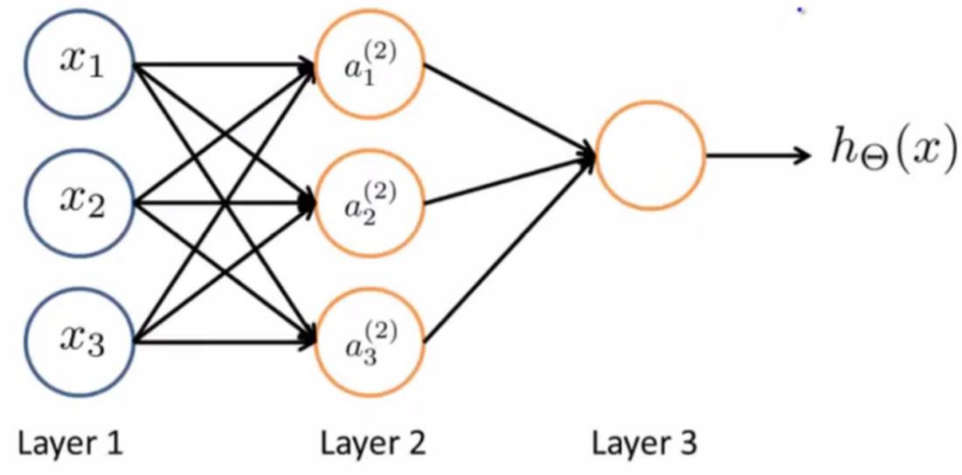
$\Theta^{(j)}$ = matrix of weights controlling function mapping from layer j to layer $j+1$

Input layer hidden layer output layer

$$a_1^{(2)} = g(\Theta_{10}^{(1)} x_0 + \Theta_{11}^{(1)} x_1 + \Theta_{12}^{(1)} x_2 + \Theta_{13}^{(1)} x_3)$$
$$a_2^{(2)} = g(\Theta_{20}^{(1)} x_0 + \Theta_{21}^{(1)} x_1 + \Theta_{22}^{(1)} x_2 + \Theta_{23}^{(1)} x_3)$$
$$a_3^{(2)} = g(\Theta_{30}^{(1)} x_0 + \Theta_{31}^{(1)} x_1 + \Theta_{32}^{(1)} x_2 + \Theta_{33}^{(1)} x_3)$$
$$h_\Theta(x) = a_1^{(3)} = g(\Theta_{10}^{(2)} a_0^{(2)} + \Theta_{11}^{(2)} a_1^{(2)} + \Theta_{12}^{(2)} a_2^{(2)} + \Theta_{13}^{(2)} a_3^{(2)})$$

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

Neural Network –vectorized implementation

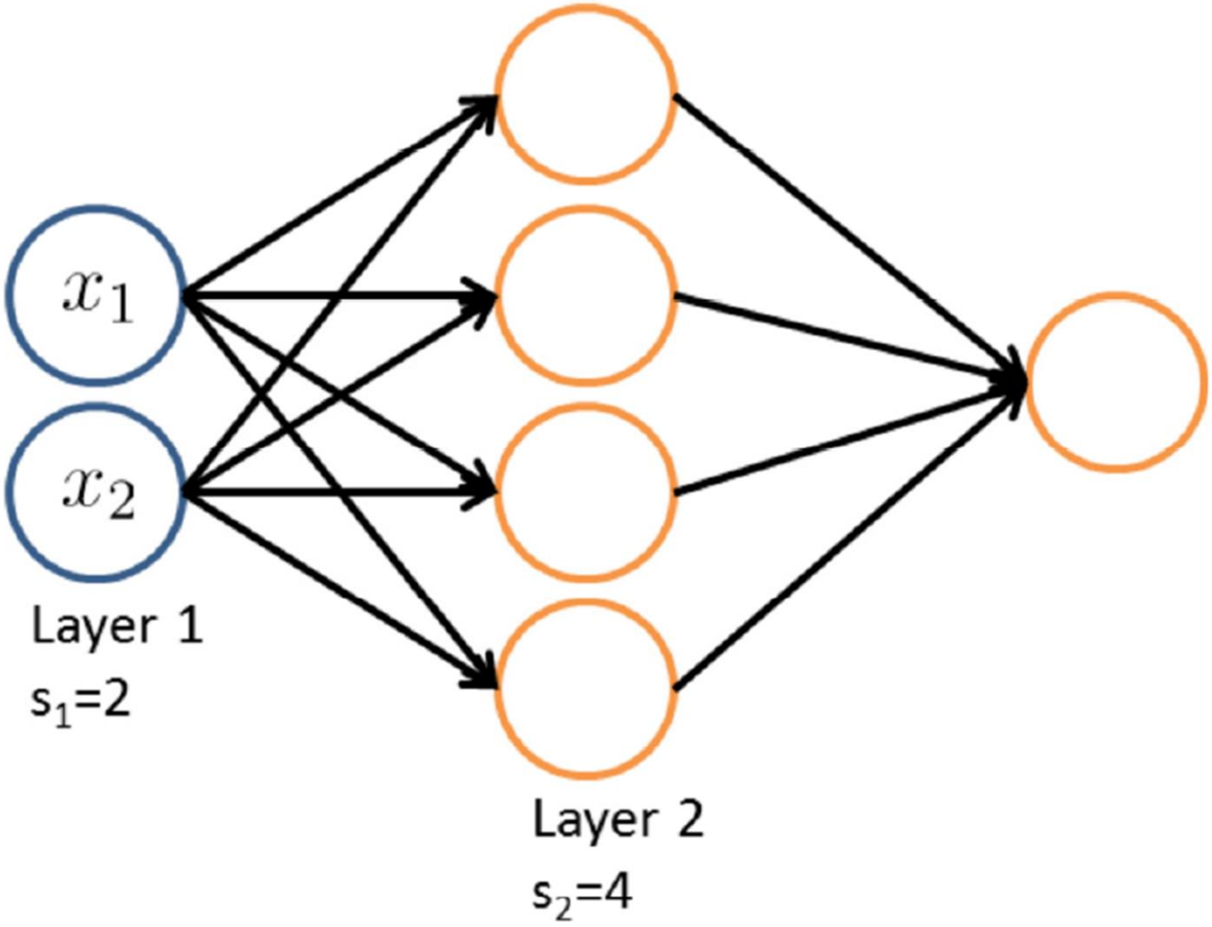


$$x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \qquad \underline{z^{(2)}} = \begin{bmatrix} z_1^{(2)} \\ z_2^{(2)} \\ z_3^{(2)} \end{bmatrix}$$

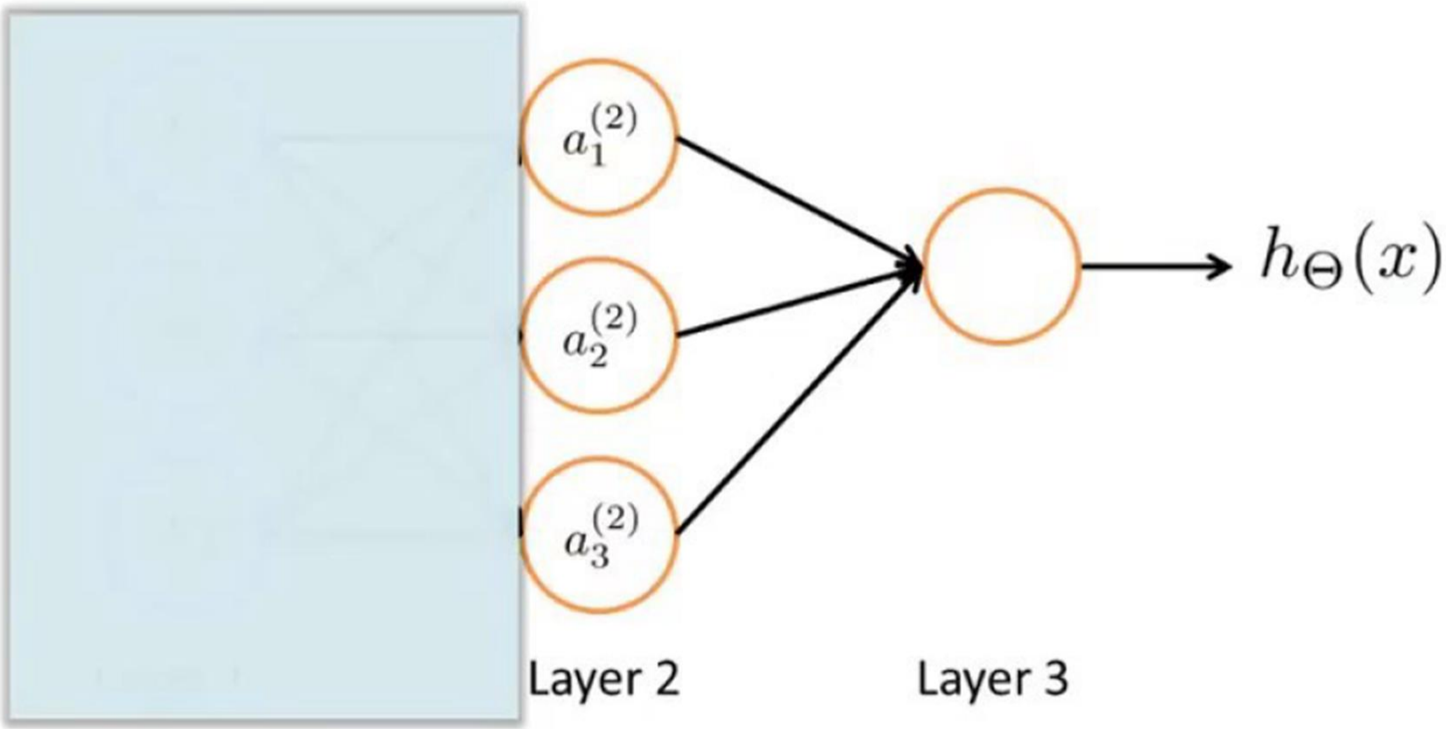
$$z^{(2)} = \Theta^{(1)} x$$
$$a^{(2)} = g(z^{(2)})$$

$$a_1^{(2)} = g(\Theta_{10}^{(1)} x_0 + \Theta_{11}^{(1)} x_1 + \Theta_{12}^{(1)} x_2 + \Theta_{13}^{(1)} x_3)$$
$$a_2^{(2)} = g(\Theta_{20}^{(1)} x_0 + \Theta_{21}^{(1)} x_1 + \Theta_{22}^{(1)} x_2 + \Theta_{23}^{(1)} x_3)$$
$$a_3^{(2)} = g(\Theta_{30}^{(1)} x_0 + \Theta_{31}^{(1)} x_1 + \Theta_{32}^{(1)} x_2 + \Theta_{33}^{(1)} x_3)$$
$$h_\Theta(x) = a_1^{(3)} = g(\Theta_{10}^{(2)} a_0^{(2)} + \Theta_{11}^{(2)} a_1^{(2)} + \Theta_{12}^{(2)} a_2^{(2)} + \Theta_{13}^{(2)} a_3^{(2)})$$

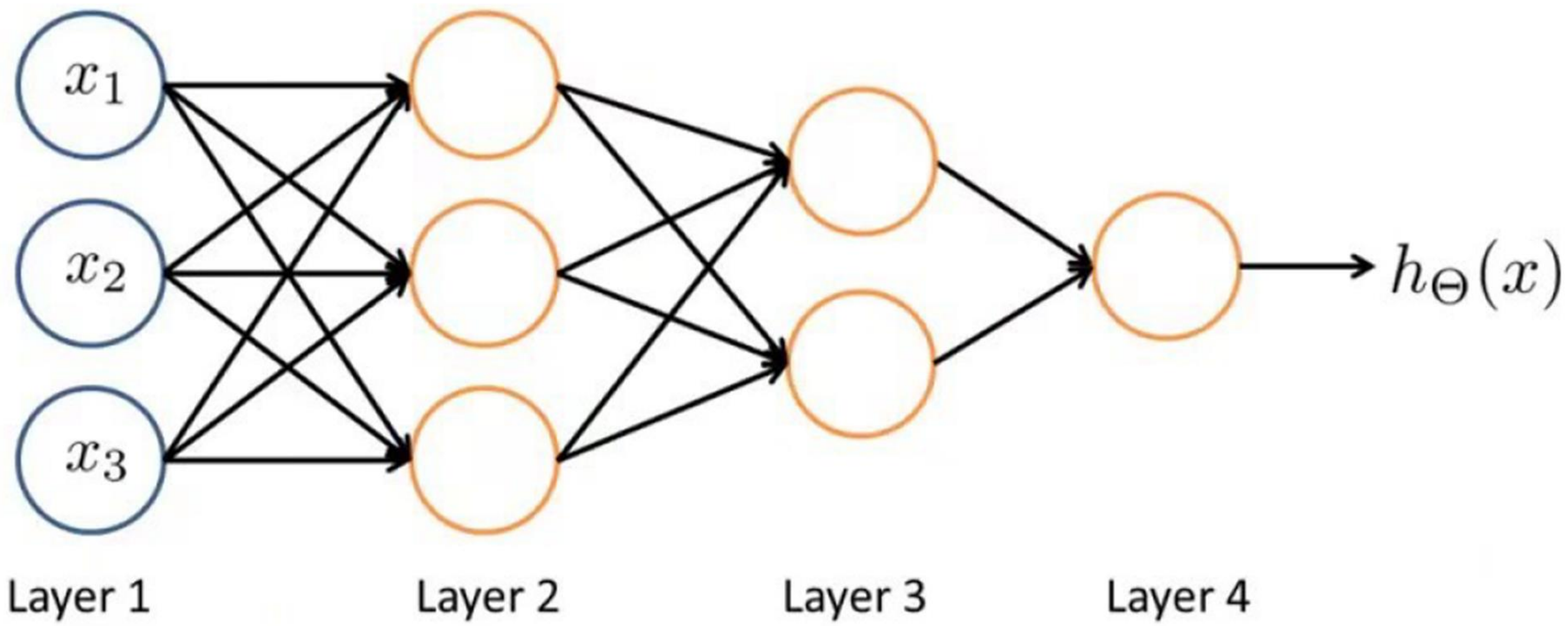
Quest: how many weight matrices has the NN and what is the dymension of each matrix ?



Neural Network is learning its own features

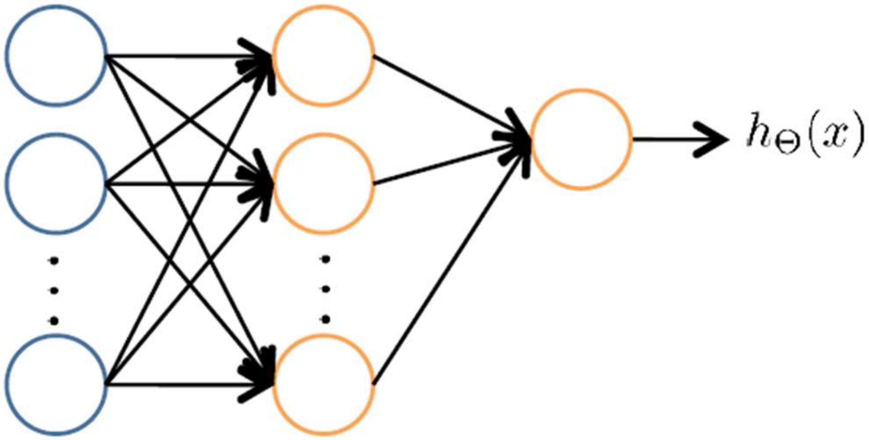


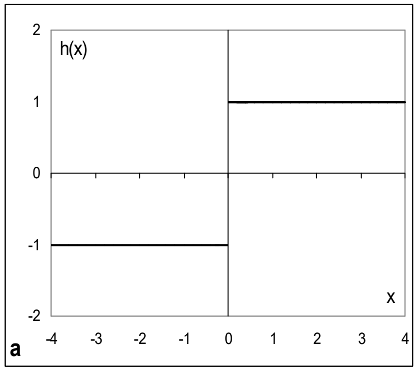
Other Network Architectures

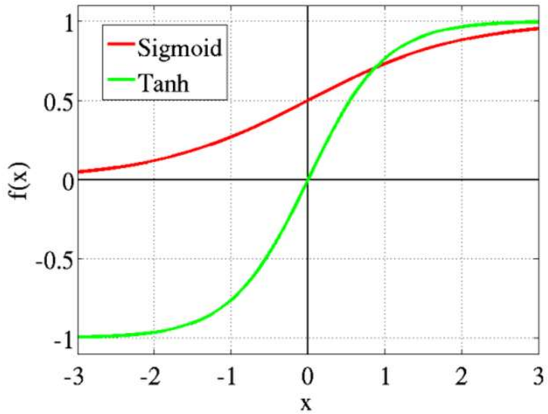


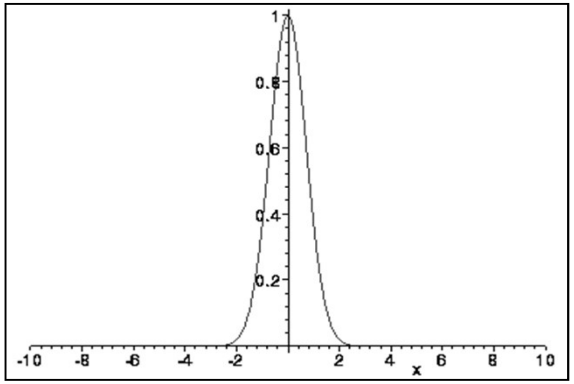
Many hidden layers can built more complex functions of the inputs (the data) => NN can learn pretty complex functions => **deep learning**

Typical Activation functions







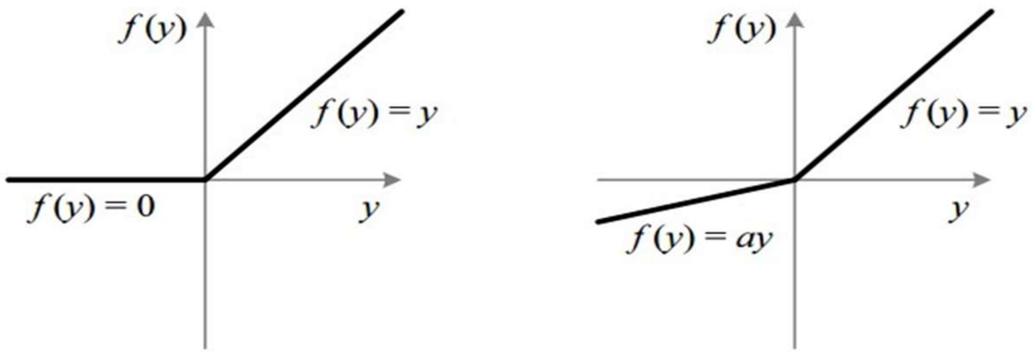


Step (heaviside)

Sigmoid (logistic) vs. Hyperbolic tangent (Tanh)

Radial Basis Function (RBF)

Typical Activation functions



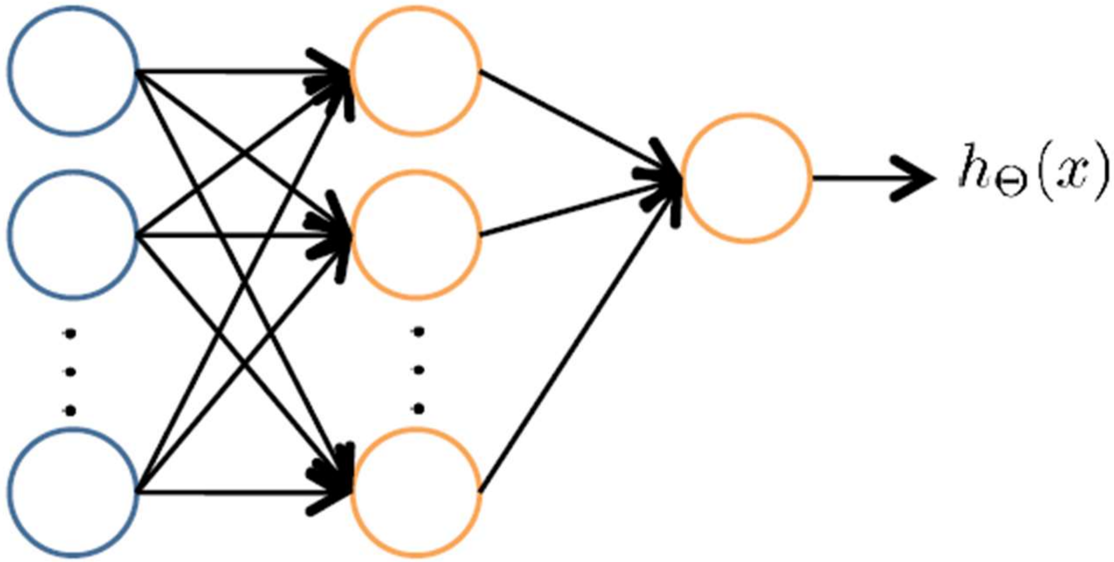
ReLU (Rectified Linear Unit) vs. Leaky ReLU

RELU:
+ Computationally efficient—allows the network to converge quickly
+ Non-linear—although it looks like a linear function, ReLU has a derivative function and allows for backpropagation.
- Dying ReLU problem—when inputs approach zero, or are negative, the gradient of the function becomes zero, the network cannot perform backpropagation and cannot learn.

Leaky ReLU:
+ Prevents dying ReLU problem—this variation of ReLU has a small positive slope in the negative area, so it does enable backpropagation, even for negative input values.
- leaky ReLU does not provide consistent predictions for negative input values.

Softmax: handles multiple classes, has as many outputs as classes. The value of each output is the probability of the class. The sum of all softmax outputs = 1.

NN - binary classification



Training set: $(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(m)}, y^{(m)})$

2 classes { 0,1 } => one output unit

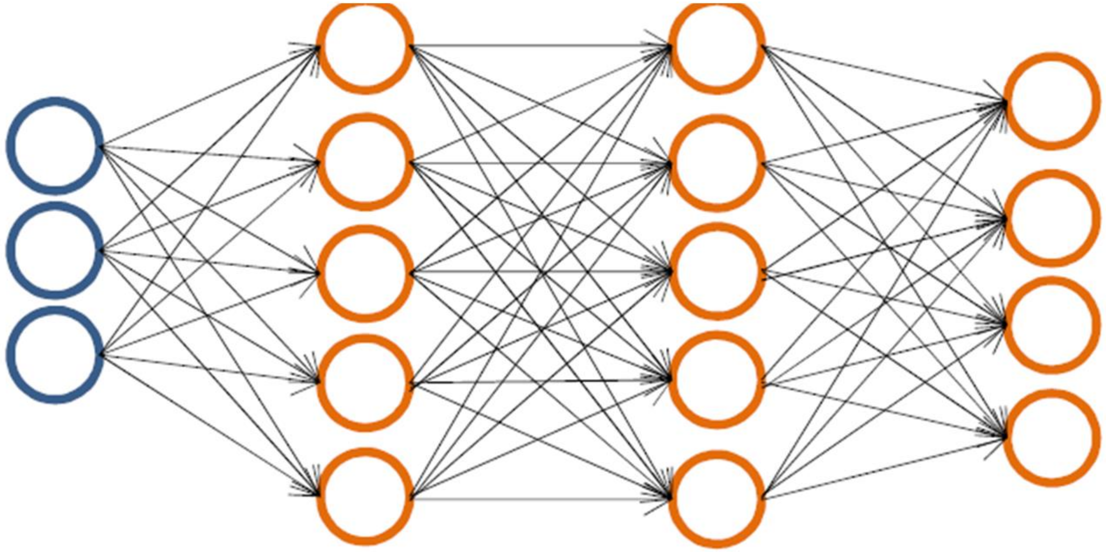
NN - multi-class classification


Pedestrian


Car


Motorcycle


Truck



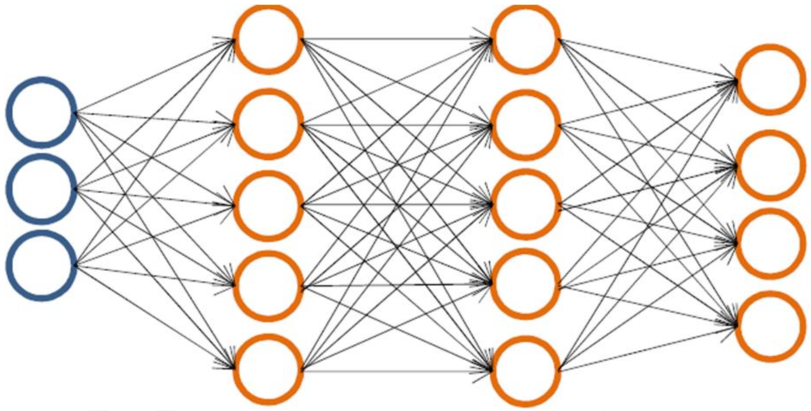
$$h_\Theta(x) \in \mathbb{R}^4$$

K classes {1,2, K} => K output units

Multiple output units: One versus all

Training set: $(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(m)}, y^{(m)})$

$y^{(i)}$ one of $\begin{bmatrix}1\\0\\0\\0\end{bmatrix}$, $\begin{bmatrix}0\\1\\0\\0\end{bmatrix}$, $\begin{bmatrix}0\\0\\1\\0\end{bmatrix}$, $\begin{bmatrix}0\\0\\0\\1\end{bmatrix}$



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

Want $h_\Theta(x) \approx \begin{bmatrix}1\\0\\0\\0\end{bmatrix}$, $h_\Theta(x) \approx \begin{bmatrix}0\\1\\0\\0\end{bmatrix}$, $h_\Theta(x) \approx \begin{bmatrix}0\\0\\1\\0\end{bmatrix}$, etc.

when pedestrian   when car   when motorcycle

Cost Function (without regularization)

Logistic Regression:

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m}\left[-y^{(i)}\log(h_\theta(x^{(i)})) - (1-y^{(i)})\log(1-h_\theta(x^{(i)}))\right]$$

Neural Network with K output (logistic) units:

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

Cost Function with regularization

Regularized Logistic Regression:

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

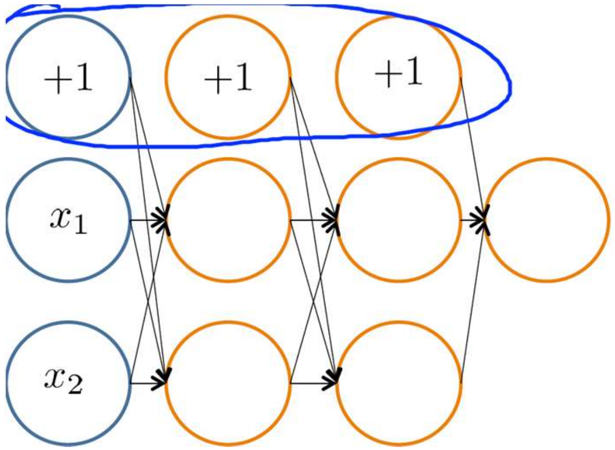
Neural Network with K output (logistic) units:

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

$$J(\Theta) = -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)} \log(h_\Theta(x^{(i)}))_k + (1-y_k^{(i)}) \log(1-(h_\Theta(x^{(i)}))_k)\right]$$
$$+\frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}(\Theta_{ji}^{(l)})^2$$

$L =$ total no. of layers in network

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



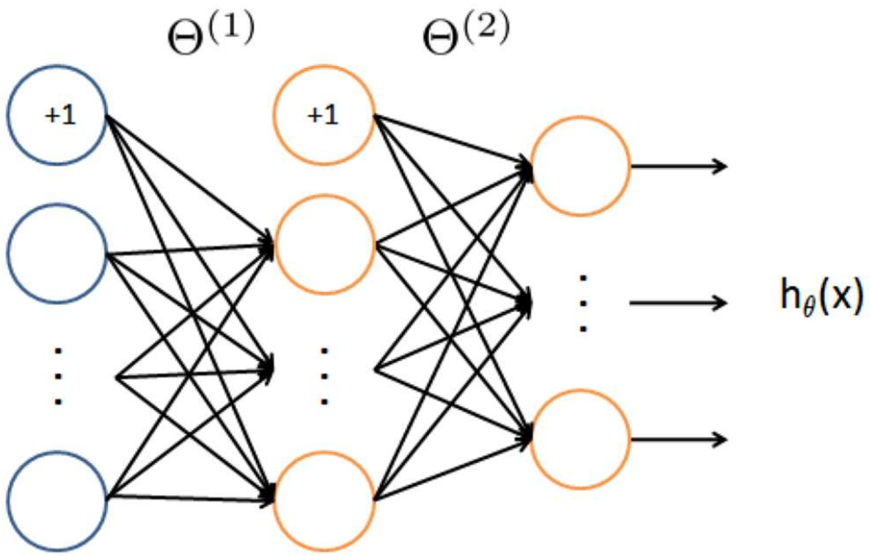
NN classification - example

MNIST handwritten digit dataset (http://yann.lecun.com/exdb/mnist/). 5000 training examples (20x20 pixels image, indicating the grayscale color intensity). The image is transformed into a row vector (with 400 elements). This gives 5000 x 400 data matrix X (every row is a training example).

NN model - example

input layer – 400 units = 20x20 pixels (input features) + 1 unit(=1, the bias)
hidden layer – 25 units + 1 unit(=1, the bias)
output layer - 10 output units (corresponding to 10 digit classes 0,1,2….9).

Matrix parameters: *Theta1* has size 25x401; *Theta2* has size 10x26.

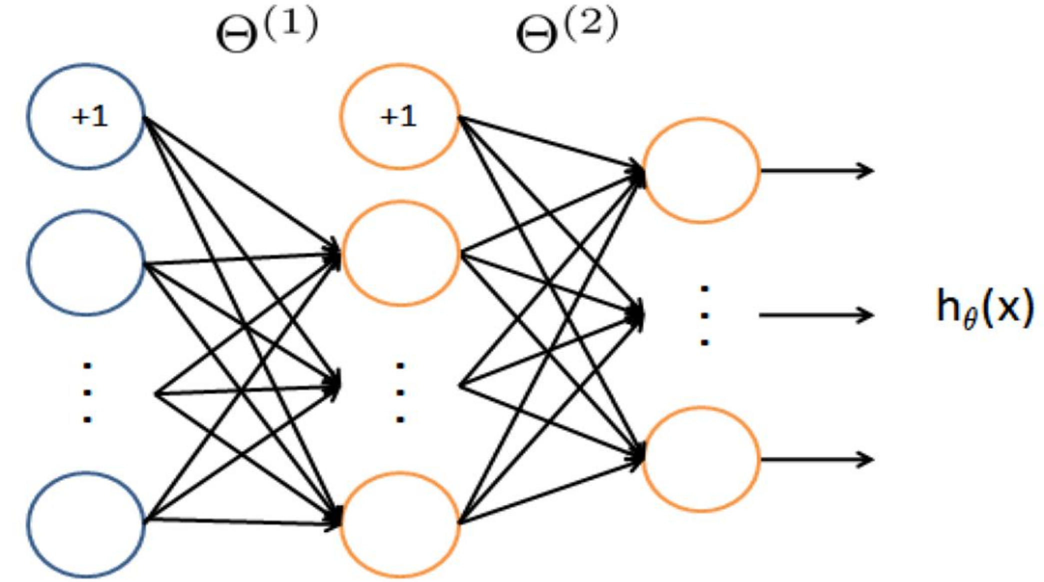


$$y = \begin{bmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \ \dots \ \text{or} \begin{bmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 1 \end{bmatrix}$$

Input Layer	Hidden Layer	Output Layer
$a^{(1)} = x$ (add $a_0^{(1)}$)	$z^{(2)} = \Theta^{(1)} a^{(1)}$ $a^{(2)} = g(z^{(2)})$ (add $a_0^{(2)}$)	$z^{(3)} = \Theta^{(2)} a^{(2)}$ $a^{(3)} = g(z^{(3)}) = h_\theta(x)$

NN model learning – forward pass

- Randomly initialize the NN parameters (matrices Theta 1 and Theta 2).
- Provide features as inputs to the NN, make a forward pass to compute all activations through the NN and the NN outputs.
- Repeat for all examples (batch training)



$a^{(1)} = x$ (add $a_0^{(1)}$)

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

$a^{(2)} = g(z^{(2)})$ (add $a_0^{(2)}$)

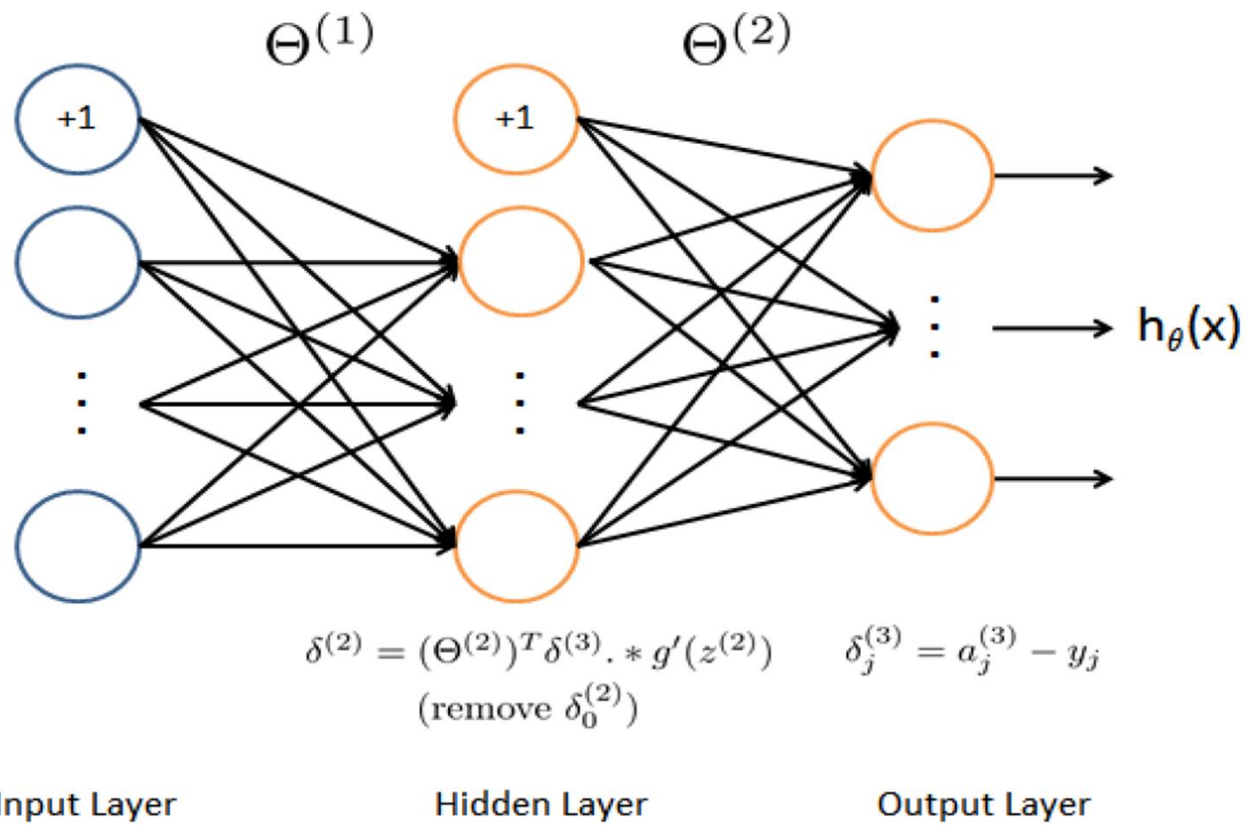
$z^{(3)} = \Theta^{(2)} a^{(2)}$

$a^{(3)} = g(z^{(3)}) = h_\theta(x)$

Input Layer | Hidden Layer | Output Layer

NN model learning -Error Backpropagation

- Compute the output error (the difference between the NN output value and the true target value).
- For all hidden layer nodes compute an “error term" that measures how much that node was “responsible" for the NN output error.
- Compute the gradient as sum of the accumulated errors for all examples.
- Update the weights.



Error Backpropagation algorithm

0) Randomly initialize the parameters (Theta1 and Theta2)
1) For i=1:number of examples
2) Provide training example *i* at the NN input.
3) Perform a feedforward pass to compute z2, a2 (for the hidden layer) and z3, a3 (for the output layer)

4) For each unit k in the output layer compute: $\delta_k^{(3)} = (a_k^{(3)} - y_k)$

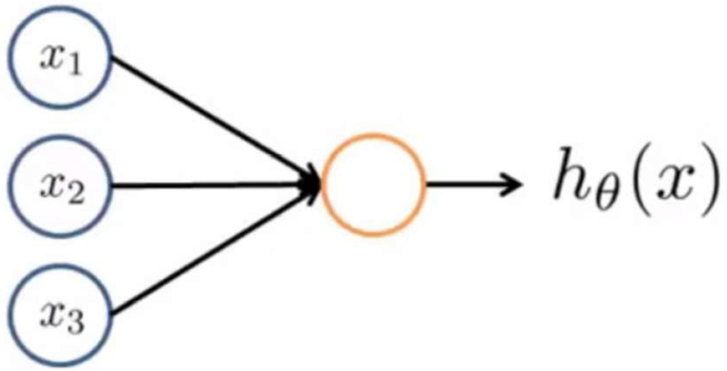
5) For the hidden layer, compute:
(***error backpropagation***)
$$\delta^{(2)} = \left(\Theta^{(2)}\right)^T \delta^{(3)} .* g'(z^{(2)})$$

6) Accumulate the gradient from this example: $\Delta^{(l)} = \Delta^{(l)} + \delta^{(l+1)}(a^{(l)})^T$

7) NN gradient (no regularization)
$$\frac{\partial J(\Theta)}{\partial \Theta_{ij}^{(l)}} = \frac{1}{m} \Delta_{ij}^{(l)}$$

8) Update NN parameters:
$$\Theta_{ij}^{(l)} = \Theta_{ij}^{(l)} - \lambda \frac{\partial J(\Theta)}{\partial \Theta_{ij}^{(l)}}$$

Sigmoid gradient



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

$$\theta^T x = \theta_0 + \sum_{j=1}^{n} \theta_j x_j$$

$$g(z) = \frac{1}{1 + e^{-z}}$$

$$g'(z) = \frac{d}{dz} g(z) = g(z)(1 - g(z))$$

Regularized Cost Function

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m}\sum_{k=1}^{K}\left[-y_k^{(i)}\log((h_\theta(x^{(i)}))_k) - (1-y_k^{(i)})\log(1-(h_\theta(x^{(i)}))_k)\right] +$$
$$\frac{\lambda}{2m}\left[\sum_{j=1}^{25}\sum_{k=1}^{400}(\Theta_{j,k}^{(1)})^2 + \sum_{j=1}^{10}\sum_{k=1}^{25}(\Theta_{j,k}^{(2)})^2\right]$$

After computing the gradient by backpropagation, add the regularization term

$$\frac{\partial}{\partial\Theta_{ij}^{(l)}}J(\Theta) = D_{ij}^{(l)} = \frac{1}{m}\Delta_{ij}^{(l)} \qquad \text{for } j = 0$$

$$\frac{\partial}{\partial\Theta_{ij}^{(l)}}J(\Theta) = D_{ij}^{(l)} = \frac{1}{m}\Delta_{ij}^{(l)} + \frac{\lambda}{m}\Theta_{ij}^{(l)} \qquad \text{for } j \geq 1$$

Adaptive learning rate

$$\theta_j = \theta_j - \alpha \frac{\partial J}{\partial \theta_j}$$

α -**Learning rate**

- **Fixed or**
- **Adaptive:**

$$\alpha^{(r+1)} = \begin{cases} b\alpha^{(r)} & if \quad J^{(r+1)} \le J^{(r)}, \quad b \ge 1\,(\text{ex.}\, b = 1.2) \\ b\alpha^{(r)} & if \quad J^{(r+1)} > J^{(r)}, \quad b < 1\,(\text{ex.}\, b = 0.2) \end{cases} \qquad \alpha^{(0)} = 0.01$$

Neural Network-Based Autonomous Driving

https://www.youtube.com/watch?v=ilP4aPDTBPE

Parameter adaptation (extra term - momentum)

$$\theta_j^{(r)} = \theta_j^{(r-1)} - \alpha \frac{\partial J}{\partial \theta_j} + \beta\left(\theta_j^{(r-1)} - \theta_j^{(r-2)}\right)$$

β **- coefficient of momentum**

- **Increase convergence rate far from minima**

- **Slow down near minima**

NN example : Back Propagation

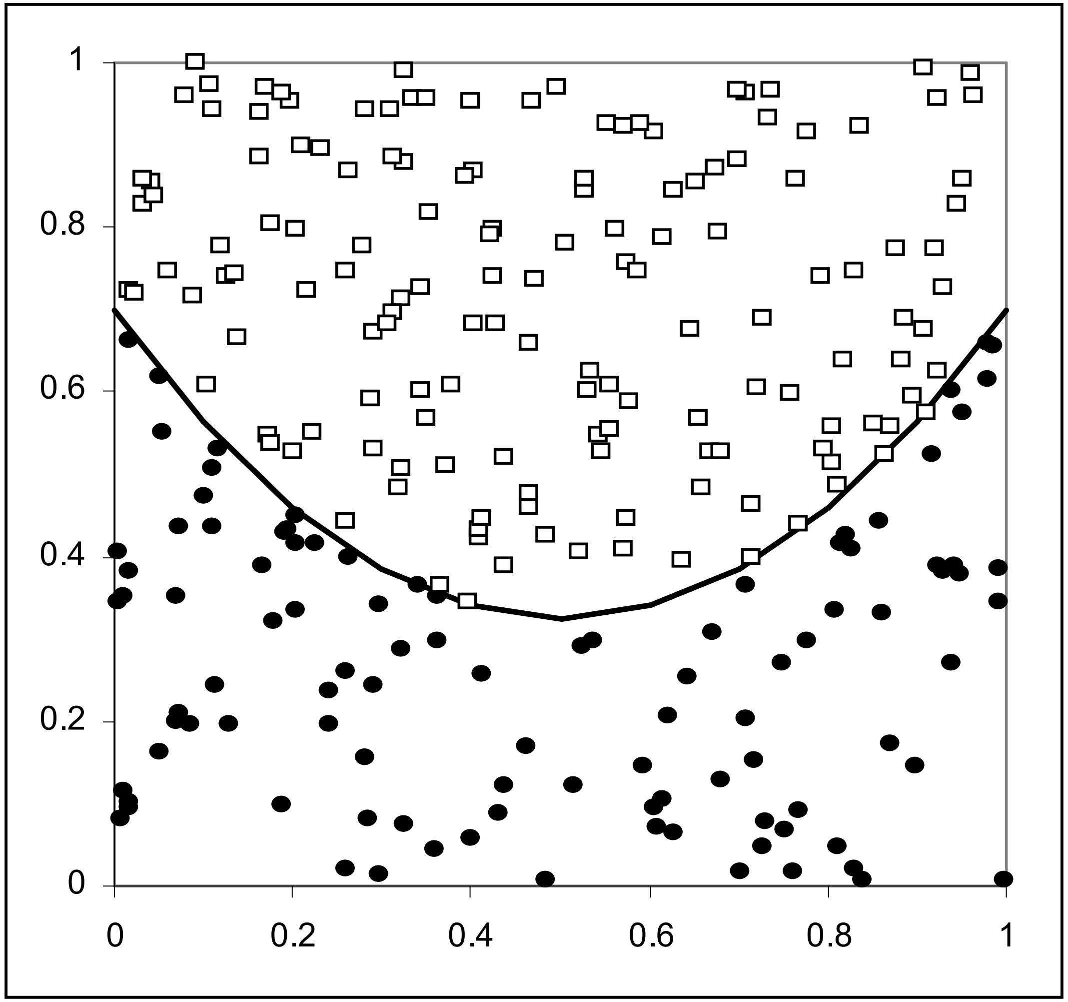
Two classes separated by a parabola

NN 2:h:1

(learn. rate 0.05)

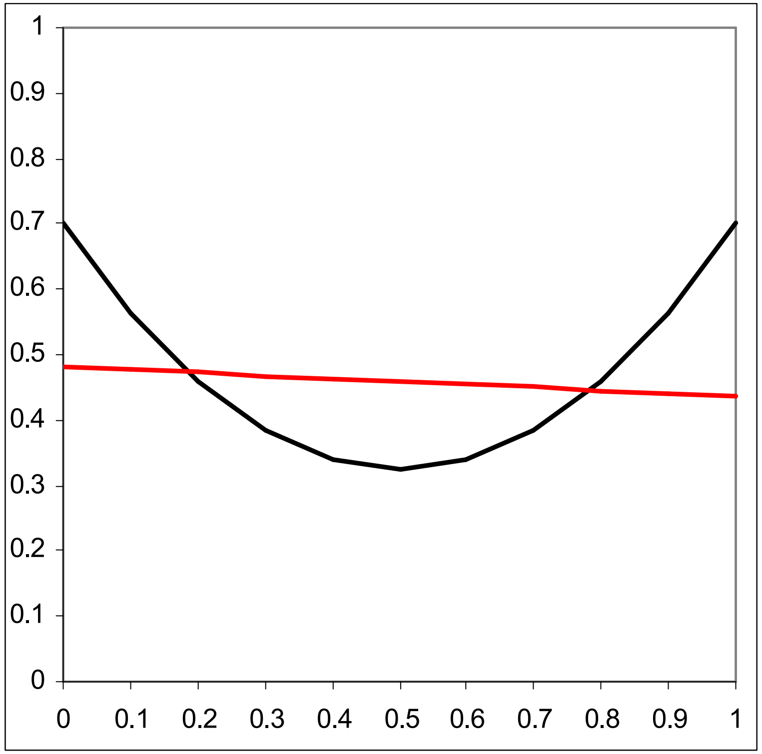
Activation func.:

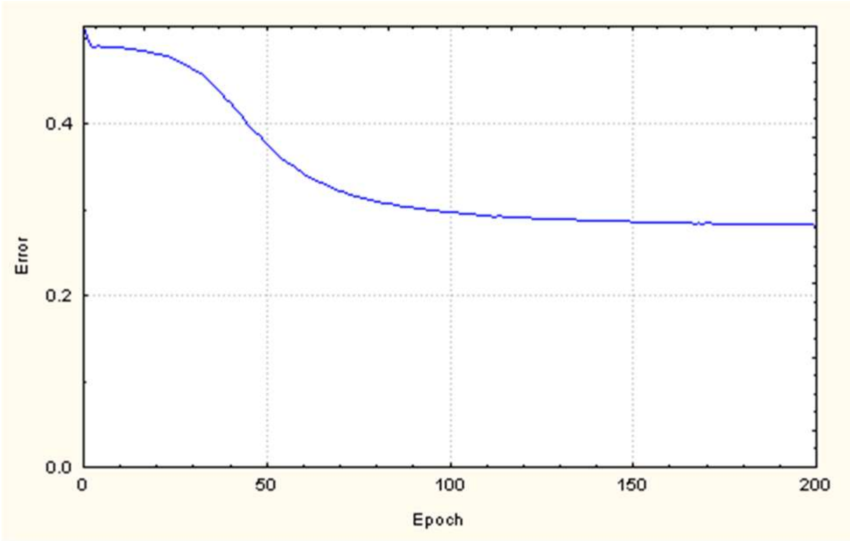
$$g(z) = \frac{1}{1+e^{-z}}$$



NN example : Back Propagation

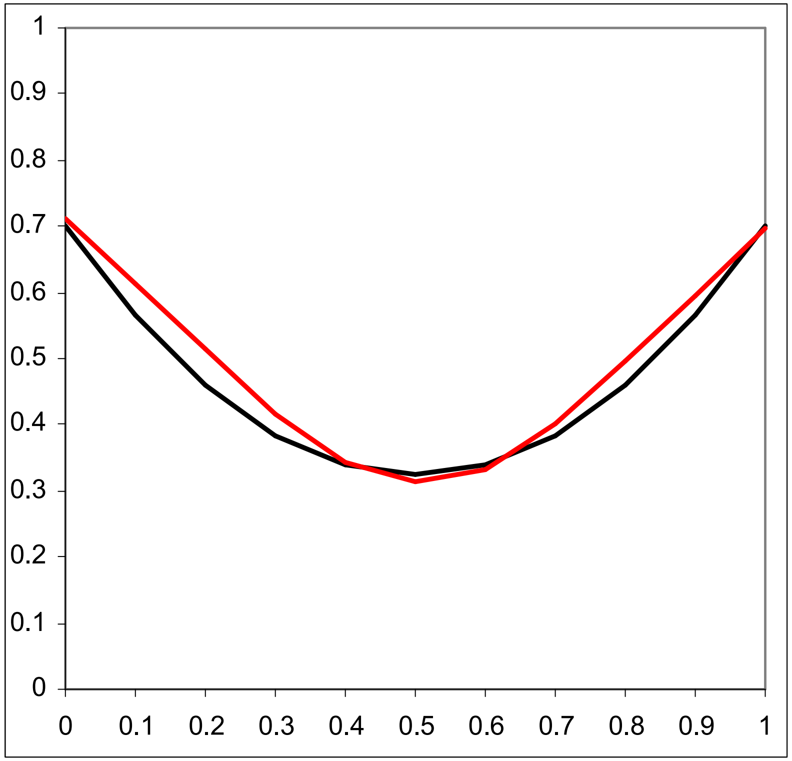
h = 1 (one hidden unit)

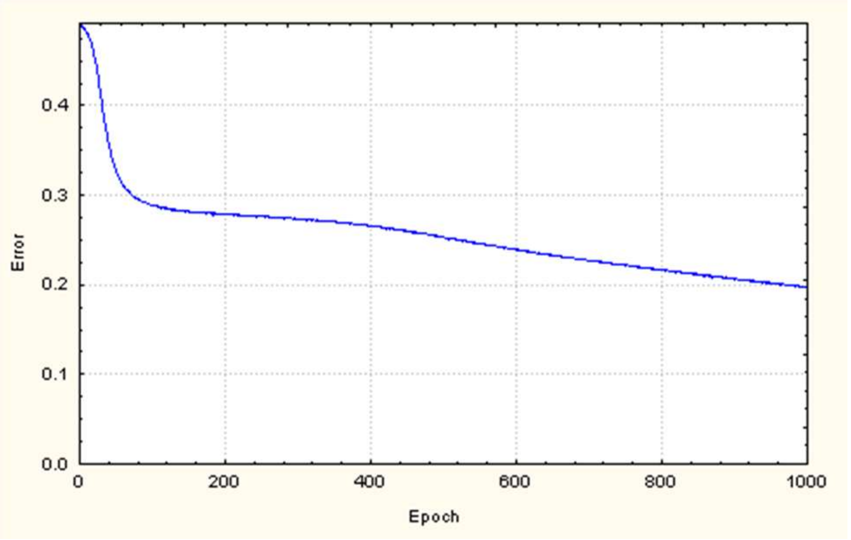




NN example : Back Propagation

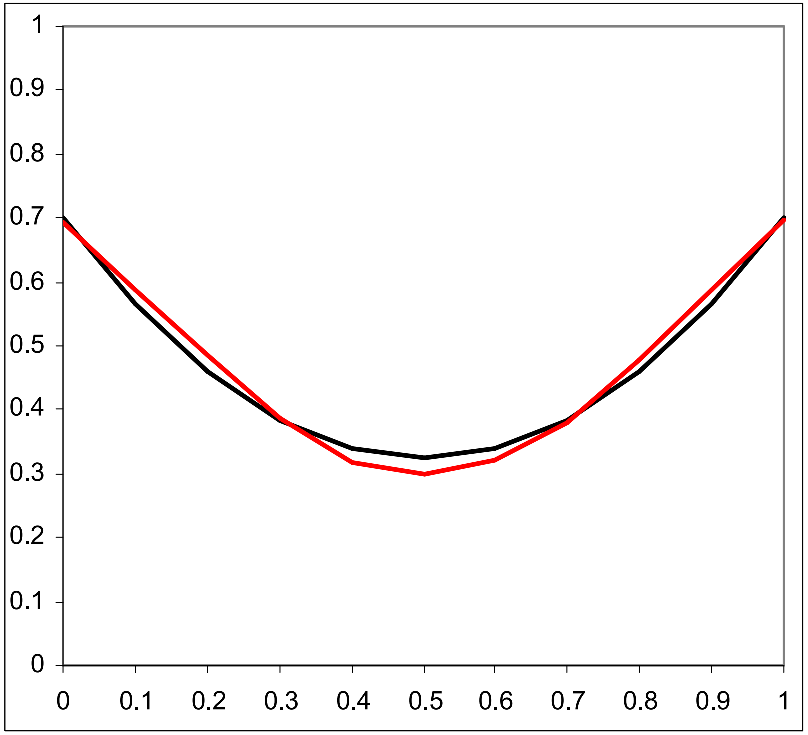
h = 2 (two hidden units)



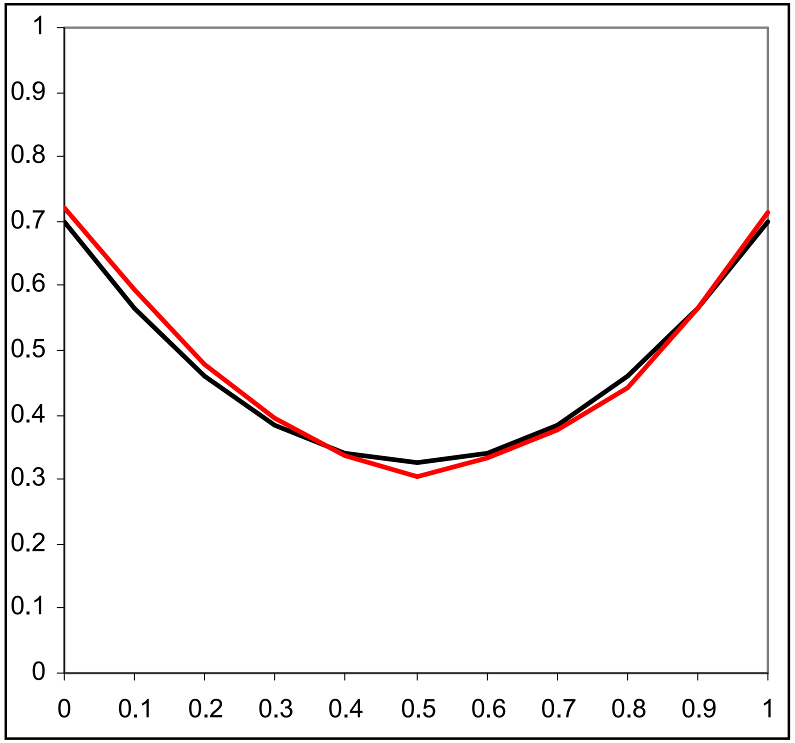


NN example : Back Propagation

(3 hidden units)



4 hidden units



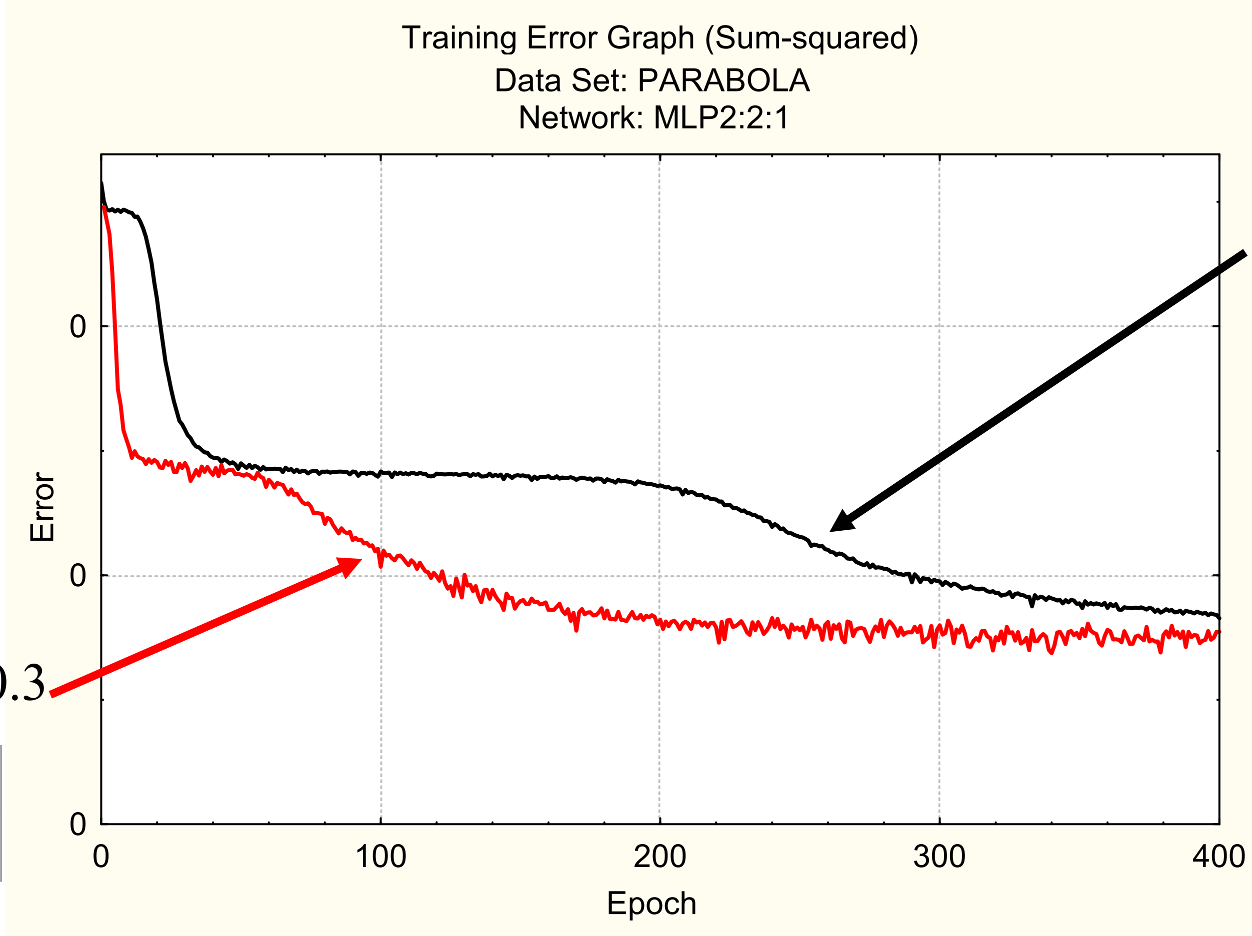
Backpropagation (BP) challenges

- Sensitivity to NN parameters (# of hidden layers, # of hidden layer units, learning rate, momentum).

- Sensitivity to the initial (random) values of weights.

- May need a large number of epochs before converging.

BP - sensitivity to learning rate



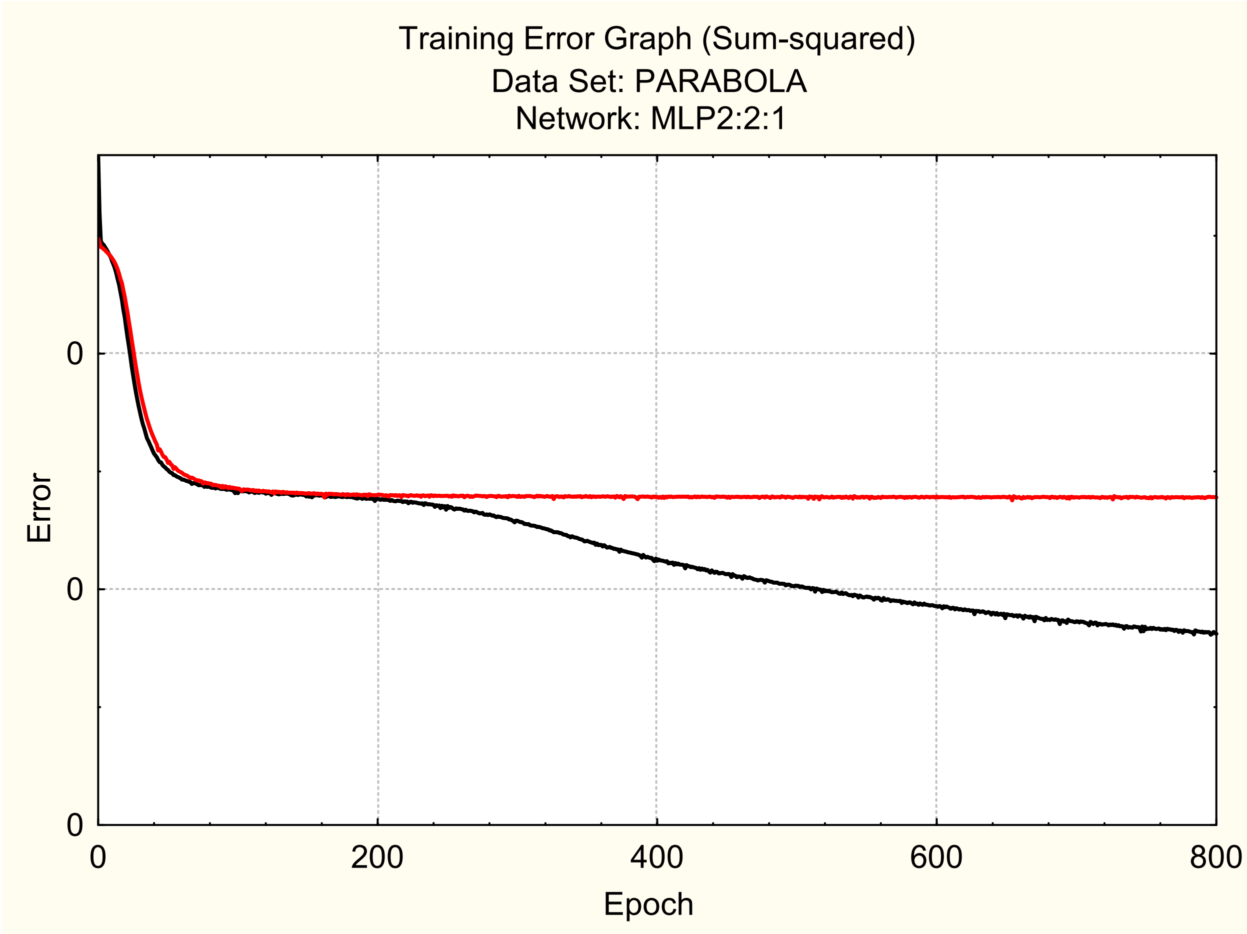
Learn. rate 0.1

99	3
3	145

Learn. rate 0.3

97	1
5	147

BP - sensitivity to initial weights

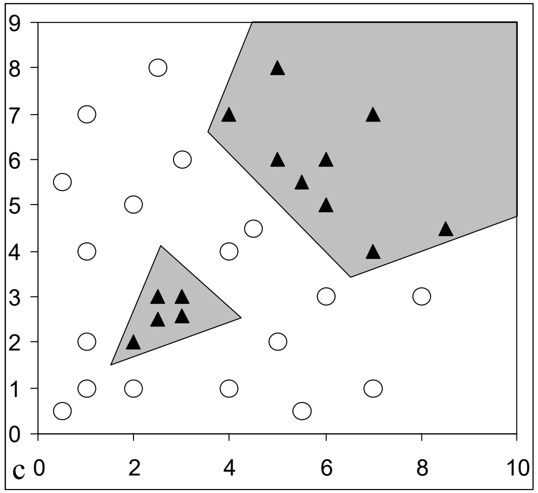


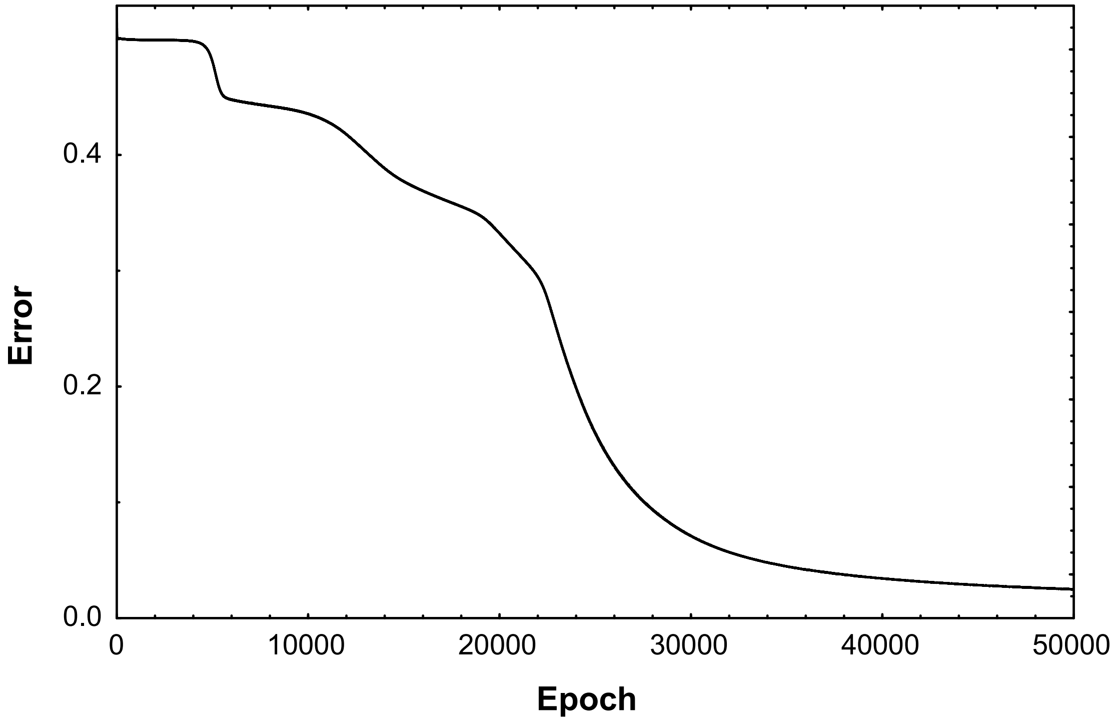
Two experiments starting with different initial weights (learn. rate 0.3)

BP - many iterations before converging

NN 2:4:1 (learn. rate 0.3)

"Sets" data





Newton's methods – alternative to gradient descent

Classical Newton's method:

$$\theta_j = \theta_j - \left(H_j\right)^{-1} \frac{\partial J}{\partial \theta_j}$$

$$\frac{\partial J}{\partial \theta_j} - \text{the gradient (Jacobian matrix, 1st derivative of cost function J)}$$

$$H_j = \frac{\left(\partial J\right)^2}{\partial \theta_j \partial \theta_j} - \text{Hessen matrix (2nd derivative of the cost function)}$$

Levenberg-Marquardt method (quasi-Newton method):

$$\theta_j = \theta_j - \left[H + \mu I\right]^{-1} \left(\frac{\partial J}{\partial \theta_j}\right)^T J$$