ieeta instituto de engenharia electrónica e telemática de aveiro

universidade de aveiro

APLICAÇÕES DE INTELIGÊNCIA ARTIFICIAL
APPLICATIONS OF ARTIFICIAL INTELLIGENCE

LECTURE 3: CLASSIFICATION

Petia Georgieva
(petia@ua.pt)

universidade de aveiro

LECTURE OUTLINE

1. **Logistic Regression (Log Reg) – linear classifier**
   - Sigmoid function model
   - Logistic regression cost (loss) function
   - Decision boundary
   - Nonlinearly separable data
   - Regularized logistic regression

2. **NEURAL NETWORKS (NN) – NON-linear classifier**
   - Neuron model: logistic unit
   - NN - binary versus multi-class classification
   - Cost function (with or without regularization)
   - NN learning - Error Backpropagation algorithm

Classification –

LOGISTIC REGRESSION

universidade de aveiro

CLASSIFICATION

Email: Spam /Not Spam ?
Tumour: Malignant /Benign ?
Online Transactions: Fraudulent (Yes /No) ?

Binary classification:
y = 1: "positive class" (e.g. malignant tumour)
y = 0: "negative class" (e.g. benign tumour)

Probabilistic interpretation, find $0<=h(x)<=1$ such that:
if h(x) >=0.5, predict "y=1"
if h(x) <0.5, predict "y=0"

Multiclass classification (*m* classes) => y= {0, 1, 2,..}
Build *m* binary classifiers, for each classifier one of the classes has label 1 all other classes take label 0 .

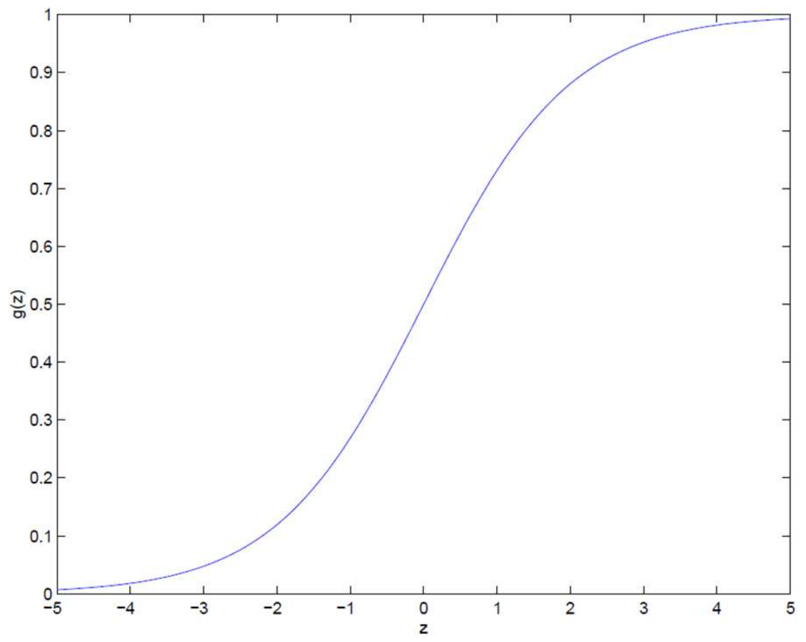
Logistic Regression

Given labelled data of m examples and n features
Labels {0,1} => binary classification
x –vector of features; θ – vector of model parameters;
h(x) – logistic (sigmoid) function model

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

$$z = \theta^T x = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \ldots .\theta_n x_n$$

Logistic (sigmoid) function



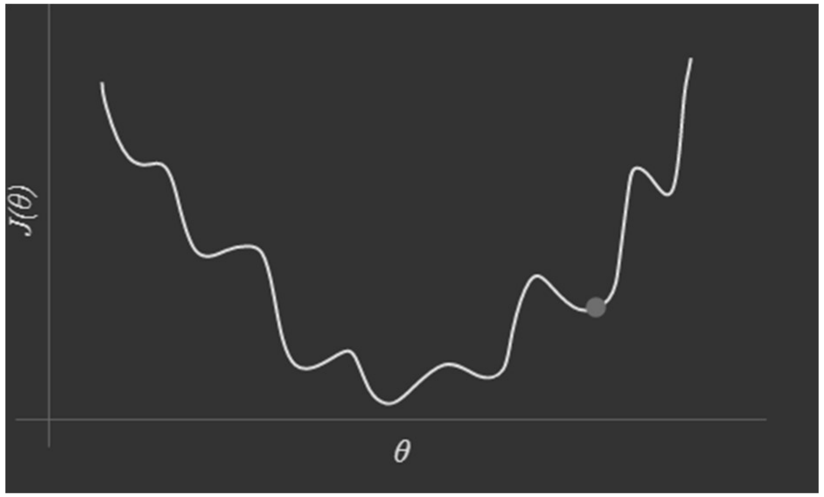
Logistic Regression Cost Function

Linear regression model => $h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \dots \theta_n x_n = \vec{\theta}^T \vec{x}$

Linear Regression cost function => $J = \frac{1}{2m}\sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)^2$

Nonlinear logistic (sigmoid) model => $h_\theta(x) = \frac{1}{1+e^{-\theta^T x}}$

If we use the same cost function as with linear regression, but now the hypothesis is a nonlinear function, *J(θ)* will be a non-convex function (has many local minima)=> **not efficient for optimization !**



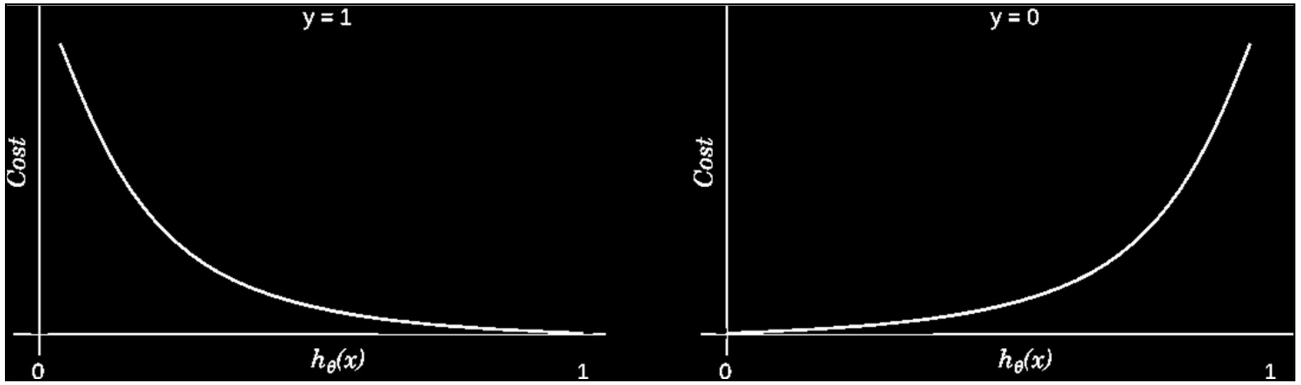
Logistic Regression Cost Function

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m} \mathrm{Cost}(h_\theta(x^{(i)}), y^{(i)})$$

$$\mathrm{Cost}(h_\theta(x), y) = \begin{cases} -\log(h_\theta(x)) & \text{if } y = 1 \\ -\log(1 - h_\theta(x)) & \text{if } y = 0 \end{cases}$$

Note: $y = 0$ or 1 always

$-\log(h_\theta(x))$ $\quad$ $-\log(1-h_\theta(x))$



LogReg cost function combined into one expression :
(also known as binary Cross-Entropy or Log Loss function)

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

Log Reg with gradient descent learning

Inicialize model parameters *(e.g. θ =0)*
Repeat until J converge {

Compute LogReg Model prediction =>
(different from linear regression model)

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

Compute LogReg cost function =>
(different from linear regression cost function)

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

Goal =>

$$\min_\theta J(\theta)$$

Compute cost function gradients =>
(same as linear regression gradients)

$$\frac{\partial J(\theta)}{\partial \theta_j} = \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)}$$

Update parameters =>
(same as linear regression parameter update)

$$\theta_j := \theta_j - \alpha\frac{\partial}{\partial\theta_j}J(\theta)$$

}

universidade de aveiro

Optimization algorithms

Gradient descent (learned in class) -
updates the parameters in the opposite direction of the gradient (in direction of decreasing gradient).

2. Other optimization algorithms

- Conjugate gradient
- AdaGrad, RMSProp
- Stochastic gradient descent with momentum
- ADAM (combination of RMSProp and stochastic optimization)
- BFGS (Broyden–Fletcher–Goldfarb–Shanno)
- Quasi-Newton methods (approximate the second derivative)

Characteristics

- Adaptive learning rate (alfa);
- Often faster than gradient descent; better convergence;
- Approximate (estimate) the true gradient over a mini-batch and not over the whole data;
- More complex algorithms

Logistic regression - example

Training data: applicant's scores on two exams and the admission decision (historical data). Build a logistic regression model to predict whether a student gets admitted into a university.

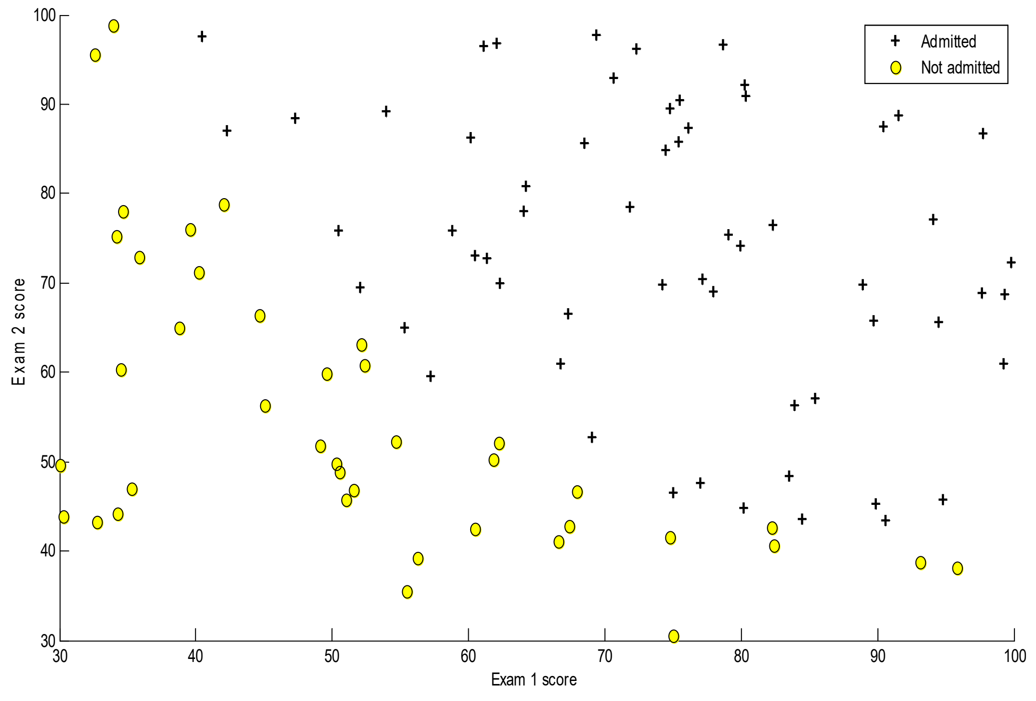


Fig. 1 Scatter plot of training data

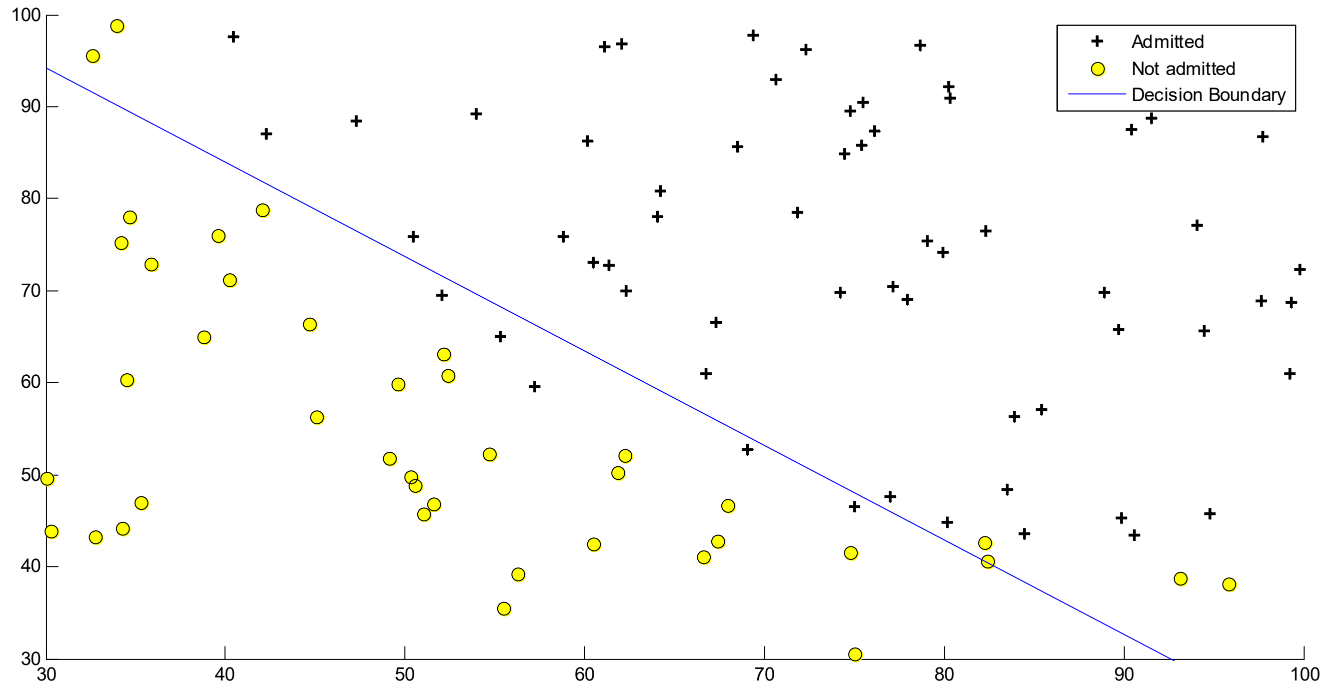
Logistic regression - example

$$z = \theta^T x = \theta_0 + \theta_1 x_1 + \theta_2 x_2 = 0 \Rightarrow \text{decision boundary}$$

$$if \quad z > 0 \Rightarrow g(z) > 0.5 \Rightarrow \text{predict class} = 1$$

$$if \quad z < 0 \Rightarrow g(z) < 0.5 \Rightarrow \text{predict class} = 0$$

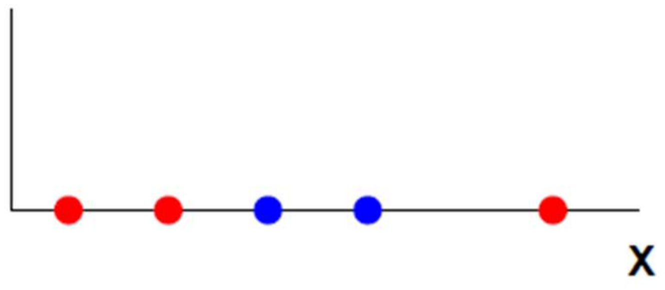
Scatter plot of training data and linear decision boundary with the optimized parameters

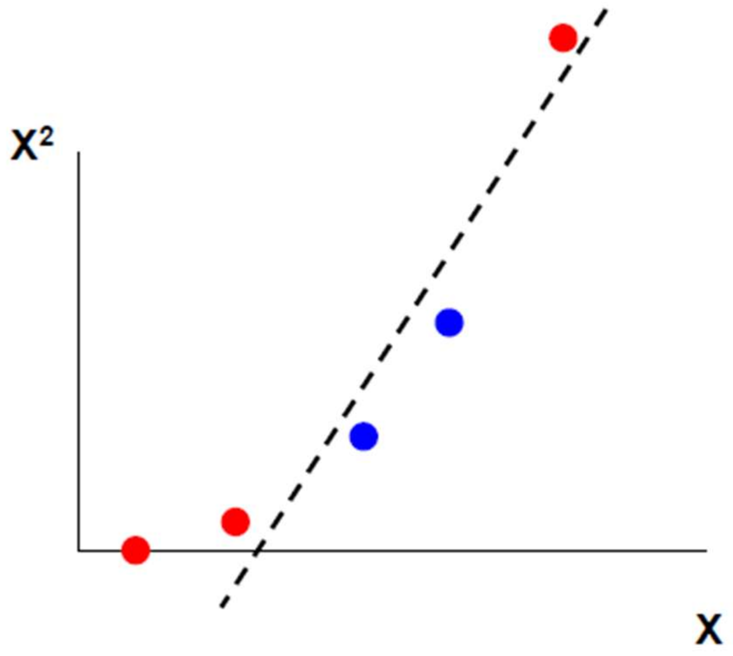


Nonlinearly Separable Data

Linear classifier cannot classify these examples.

And now ?





$z = \theta^T x = \theta_0 + \theta_1 x + \theta_2 x^2 = 0 =>$

Nonlinear decision boundary (in the original feature space x)

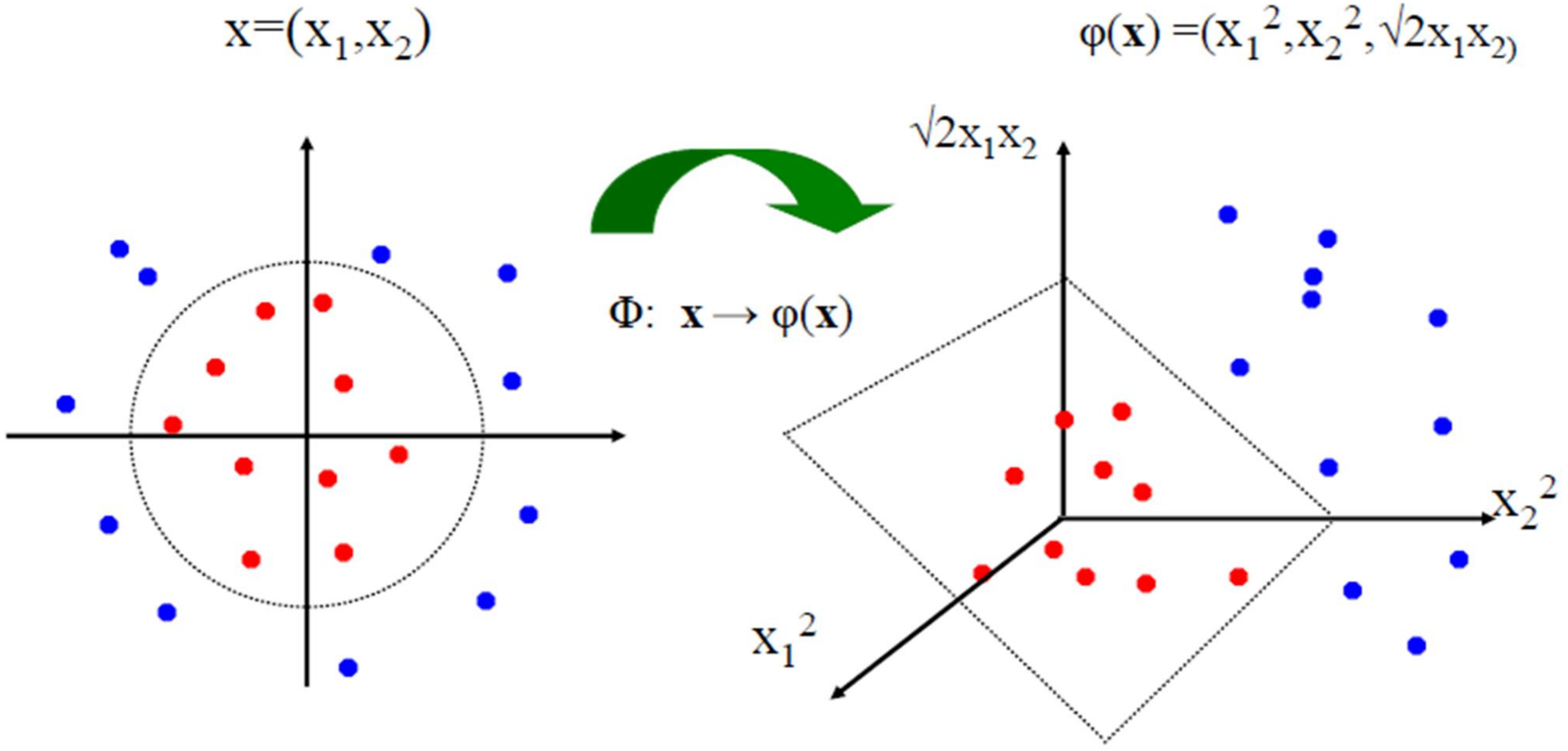
Linear decision boundary (in the extended feature space x, x^2)

$if \quad z > 0 => g(z) > 0.5 =>$ predict class $= 1$

$if \quad z < 0 => g(z) < 0.5 =>$ predict class $= 0$

Nonlinearly Separable Data

- The original input space (x) can be mapped to some higher-dimensional feature space (φ(**x**))where the training set is separable:

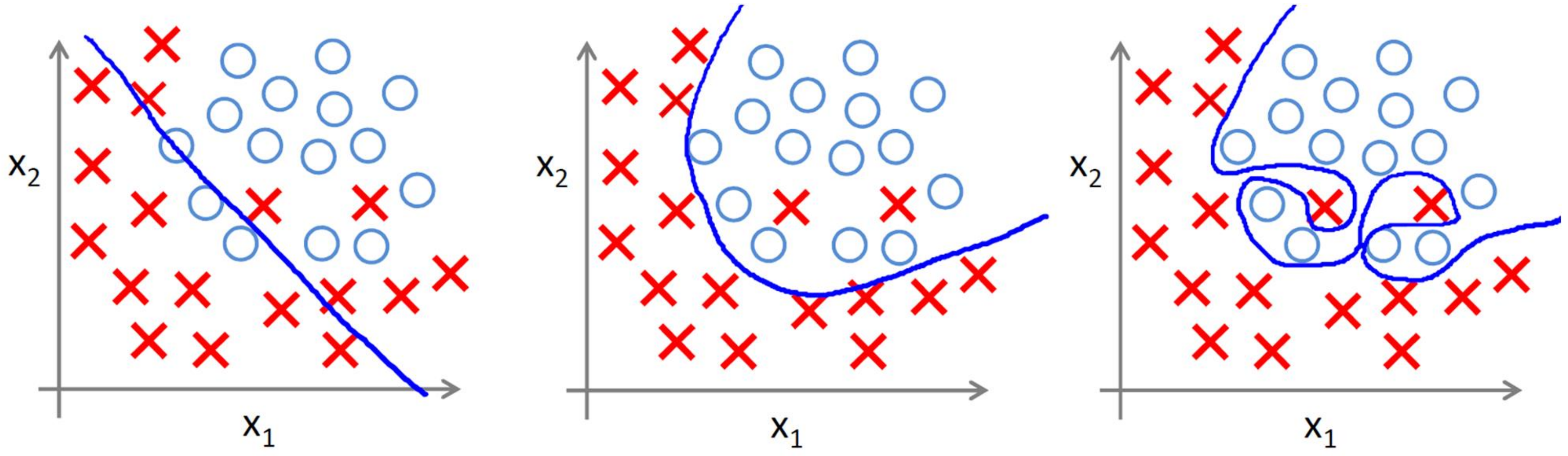


$x=(x_1,x_2)$
$\varphi(\mathbf{x}) = (x_1^2, x_2^2, \sqrt{2}x_1x_2)$
$\Phi\colon\ \mathbf{x} \rightarrow \varphi(\mathbf{x})$
$\sqrt{2}x_1x_2$
x_2^2
x_1^2

This slide is courtesy of *www.iro.umontreal.ca/~pift6080/documents/papers/**svm**_tutorial.**ppt***

Overfitting problem

<u>Overfitting:</u> If we have too many features, the learned model may fit the training data very well but fail to generalize to new examples.



underfit- high bias model ok model overfit – high variance

Regularization

Regularization is a popular method in ML to prevent overfitting by reducing the model parameters θ towards zero

1 Ridge Regression (L2 norm)

— Keep all the features, but reduces the magnitude of θ.
— Works well when each of the features contributes a bit to predict y.

2 Lasso Regression (L1 norm)

- May shrink some coefficients of θ to exactly zero.
- Serve as a feature selection tools (reduces the number of features).

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

Regularized Logistic Regression

Unregularized Log Reg cost function:

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

Regularized Log Reg cost function (ridge regression)

λ is the regularization parameter (hyper-parameter) that needs to be selected

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

Small λ => lower bias, higher variance

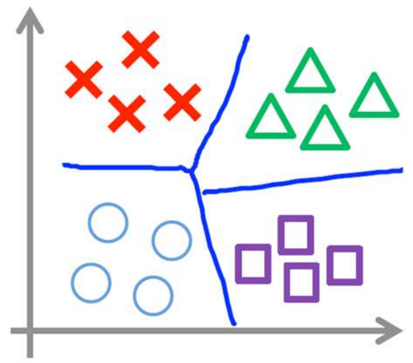
High λ => higher bias, lower variance

universidade de aveiro

Multiclass Classification

Exs. *Email classification* (work, friends, family)

Medical diagnosis (not ill, cold, flu) ; *Weather* (sunny, cloudy, rain, snow)



One-versus-all strategy :

For K classes train K binary classifiers:

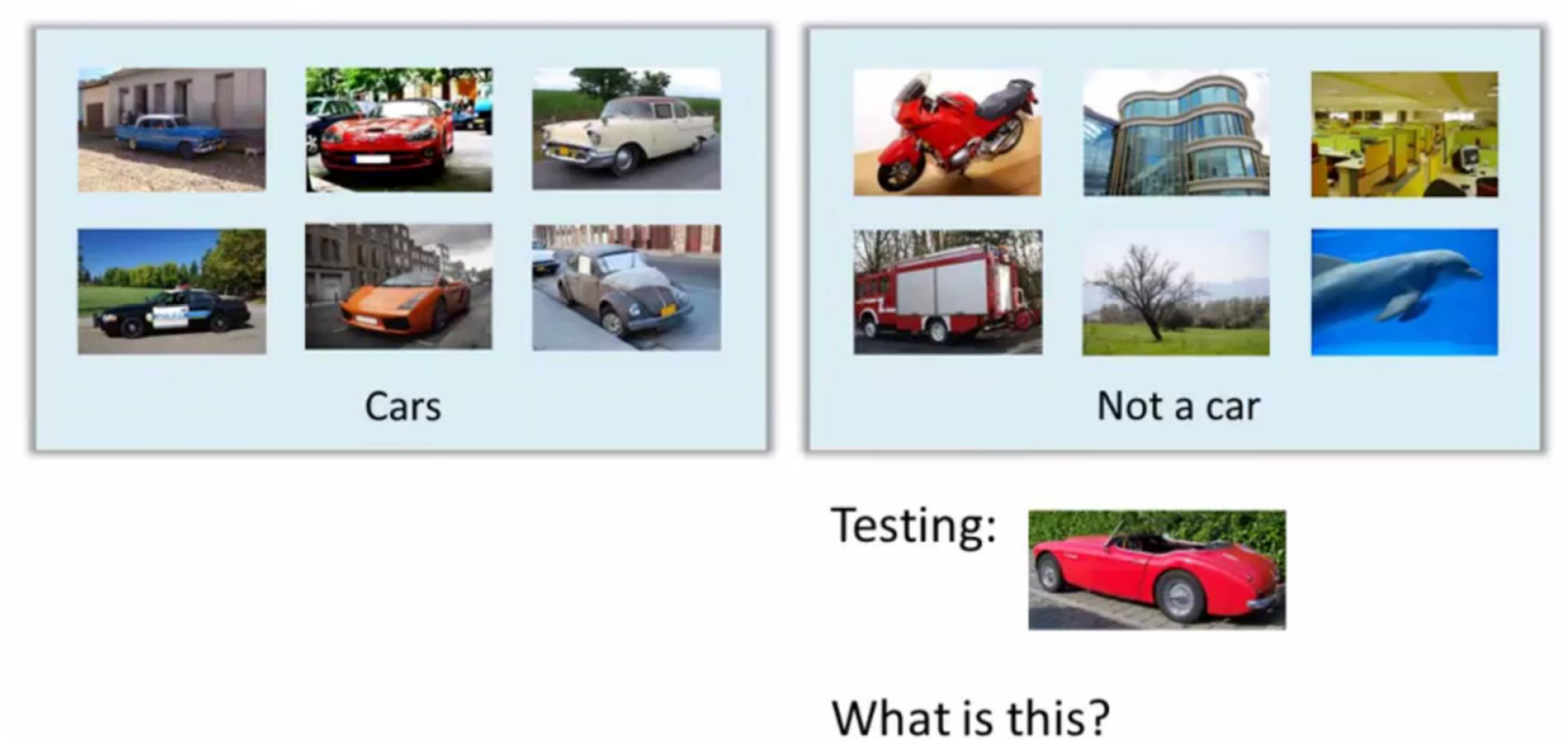
```
for c=1:K
    Make y_binary=1 (only for examples of class c)
         y_binary=0 (for examples of all other classes)
    theta=Train classifier with training data X and output y_binary.
    Save the learned parameters of all classifiers in one matrix
    where each raw is the learned paramenters of one classifier:
   theta_all(c,:)=theta
end
```

New example: winner-takes-all strategy, the binary classifier with the highest output score assigns the class.

Classification –

NEURAL NETWORKS (NN)

Computer vision: car detection



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

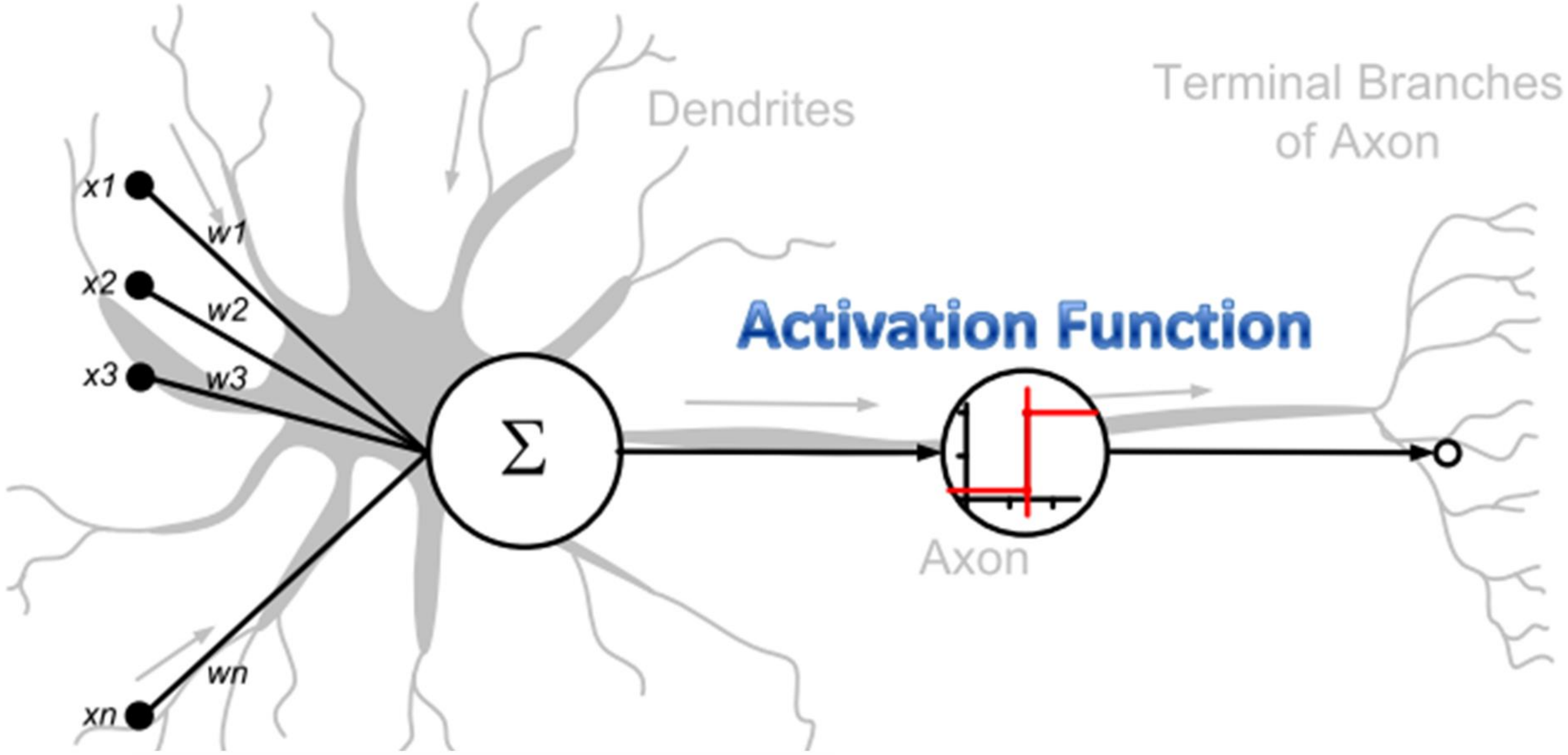
If using quadratic features => 3 million features

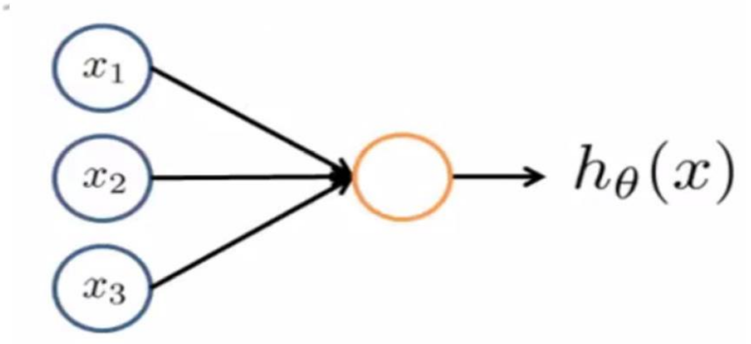
Logistic regression is not suitable for such complex nonlinear models.

Neural Networks fit better complex nonlinear models.

Neuron model

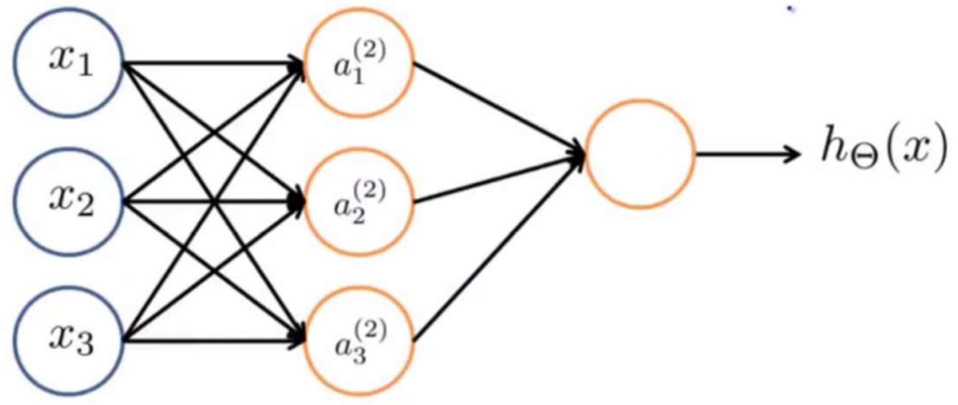
Origins: NN models inspired by biological neuron structures and computations.





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 $j+1$

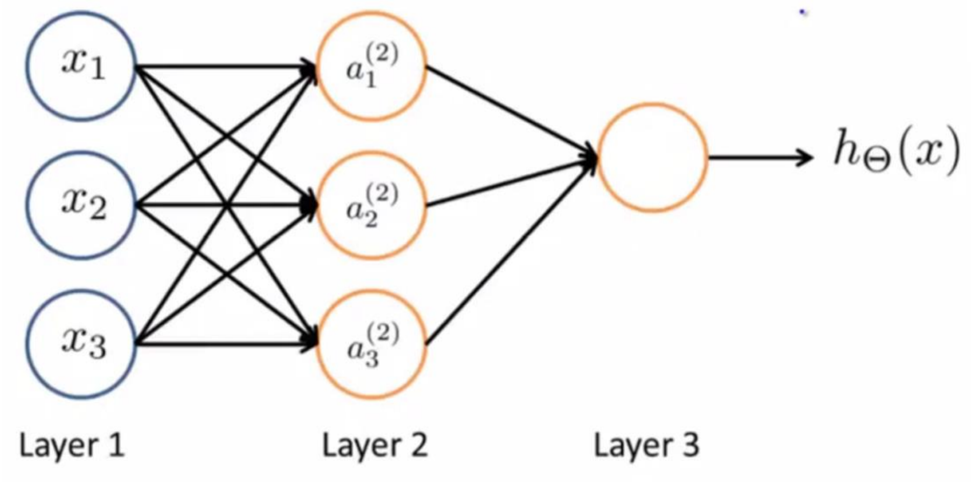
Layer 1 Layer 2 Layer 3

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

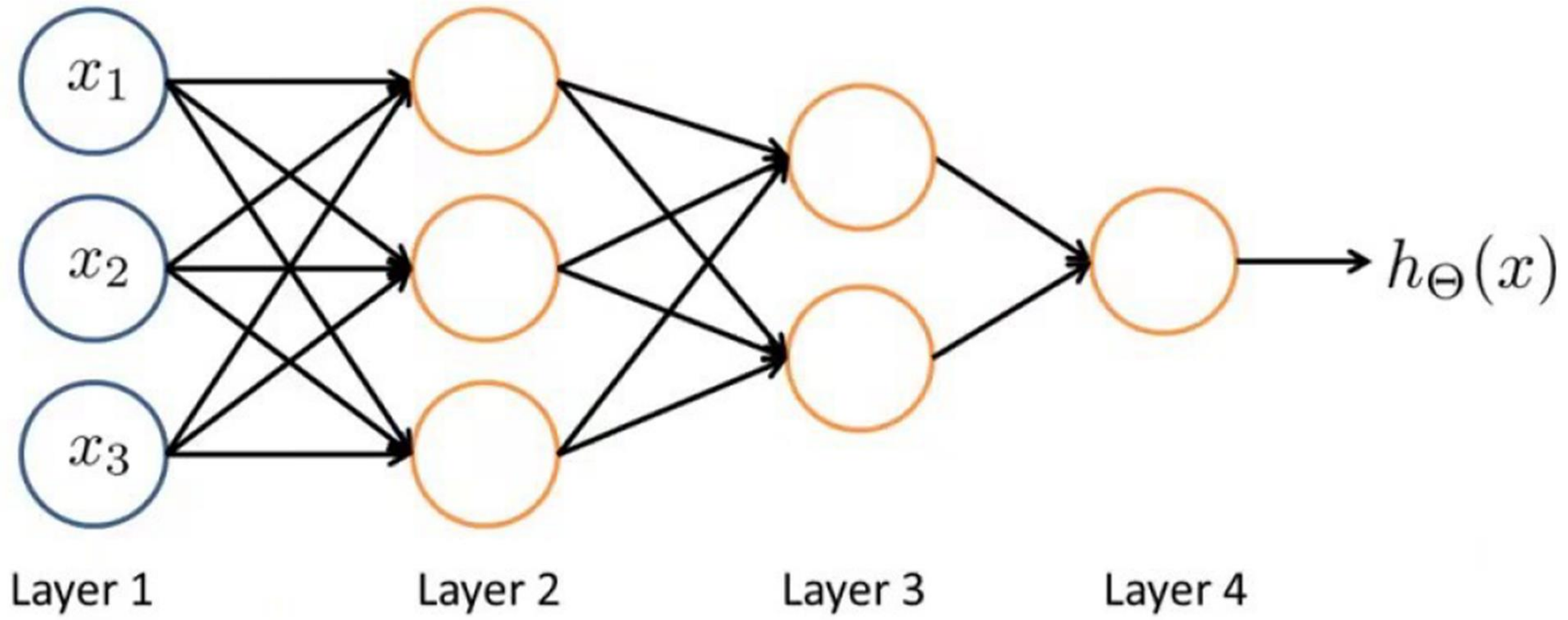


$$x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \qquad \underline{z^{(2)}} = \begin{bmatrix} z_1^{(2)} \\ z_2^{(2)} \\ z_3^{(2)} \end{bmatrix}$$

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

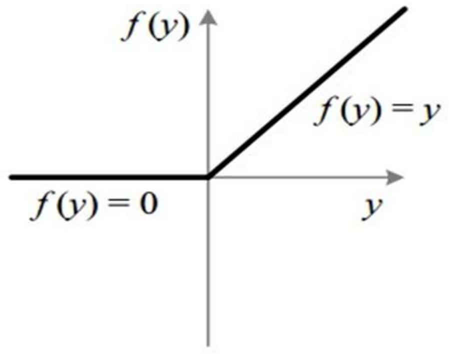
$$a_1^{(2)} = g(\Theta_{10}^{(1)} x_0 + \Theta_{11}^{(1)} x_1 + \Theta_{12}^{(1)} x_2 + \Theta_{13}^{(1)} x_3)$$
$$a_2^{(2)} = g(\Theta_{20}^{(1)} x_0 + \Theta_{21}^{(1)} x_1 + \Theta_{22}^{(1)} x_2 + \Theta_{23}^{(1)} x_3)$$
$$a_3^{(2)} = g(\Theta_{30}^{(1)} x_0 + \Theta_{31}^{(1)} x_1 + \Theta_{32}^{(1)}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

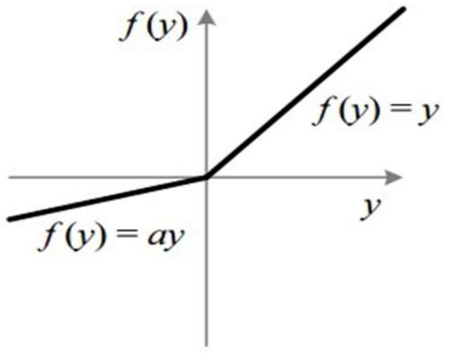
Other Network Architectures



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

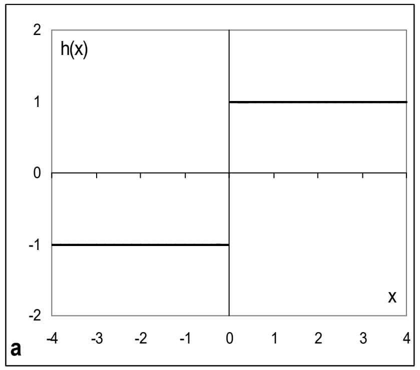
Typical Activation functions

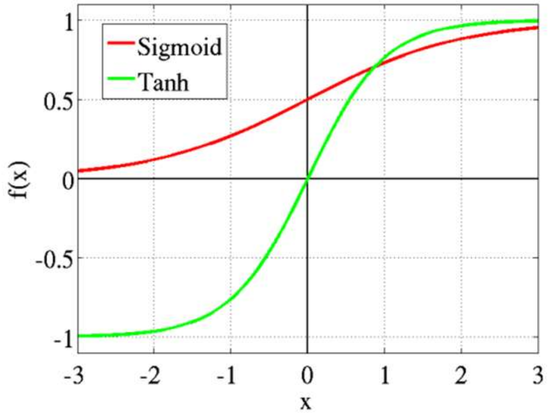


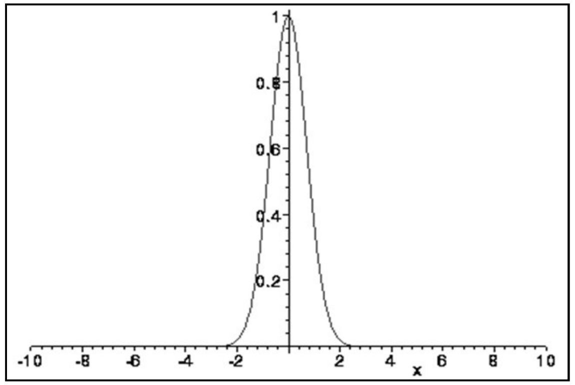


<u>ReLU (Rectified Linear Unit)</u> Leaky ReLU

Main question: Do they have gradient, how easy is to compute the gradient





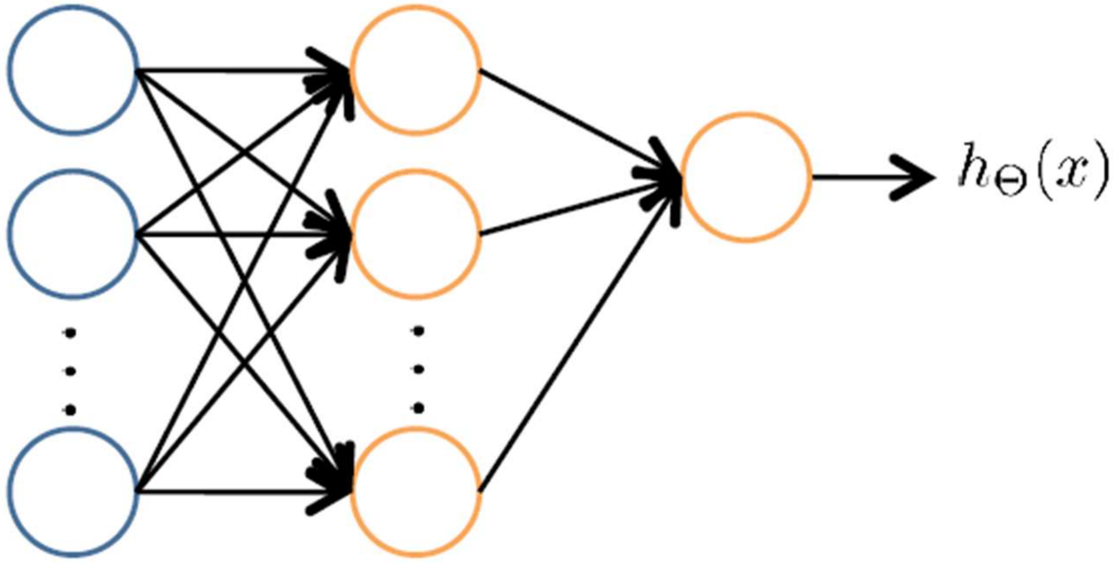


Step (heaviside)

Sigmoid (logistic) vs. Hyperbolic tangent (Tanh)

Radial Basis Function (RBF)

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

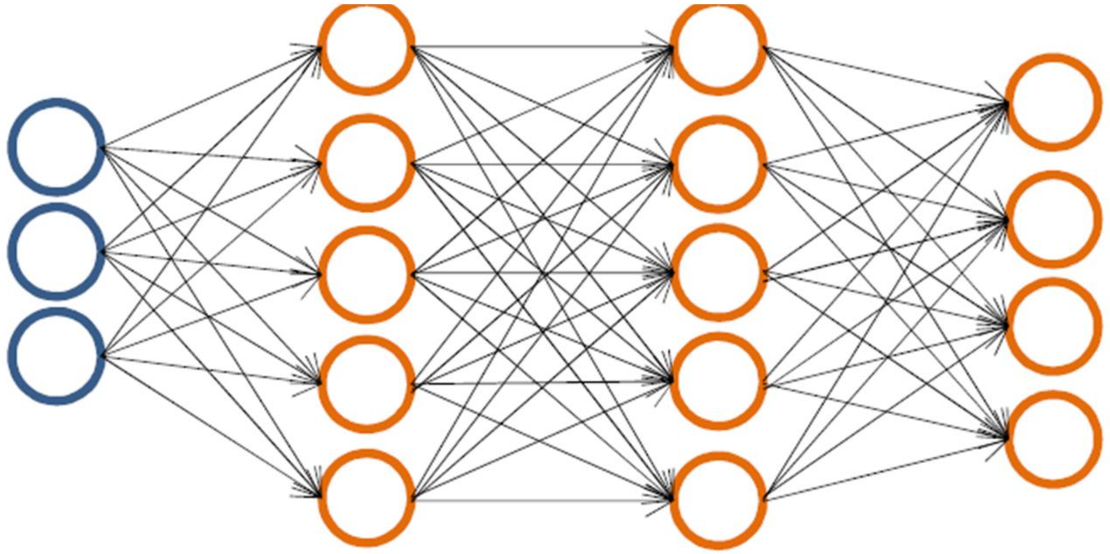
NN - multi-class classification

Pedestrian

Car

Motorcycle

Truck



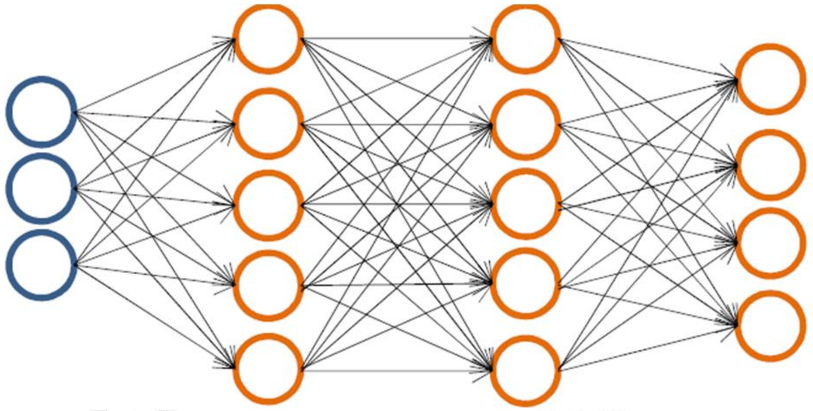
$$h_\Theta(x) \in \mathbb{R}^4$$

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

NN Cost Functions (without regularization)

NN with 1 output (logistic) unit (suitable for binary classification problems):

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

NN with K output (logistic) units (suitable for multiclass classif. problems):

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

NN with 1 output (not logistic) suitable for nonlinear regression problems:

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta\left(x^{(i)}\right) - y^{(i)} \right)^2$$

Cost Function with regularization

Regularized Logistic Regression:

Regularization teri

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

Neural Network with K output (logistic) units:

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

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

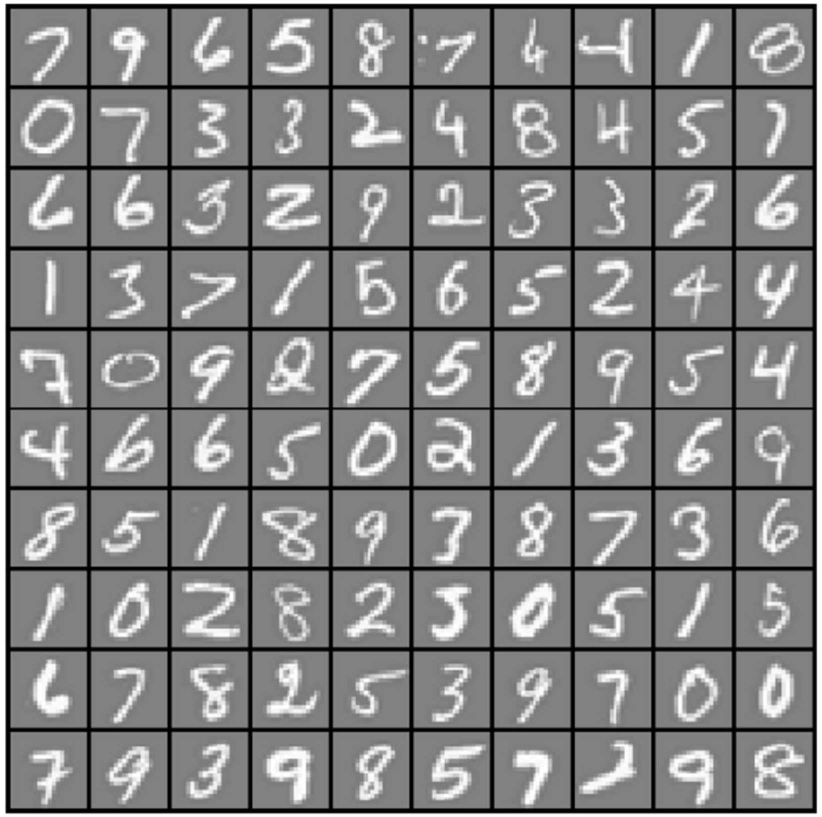
Regularization term

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

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

NN classification – MNIST problem

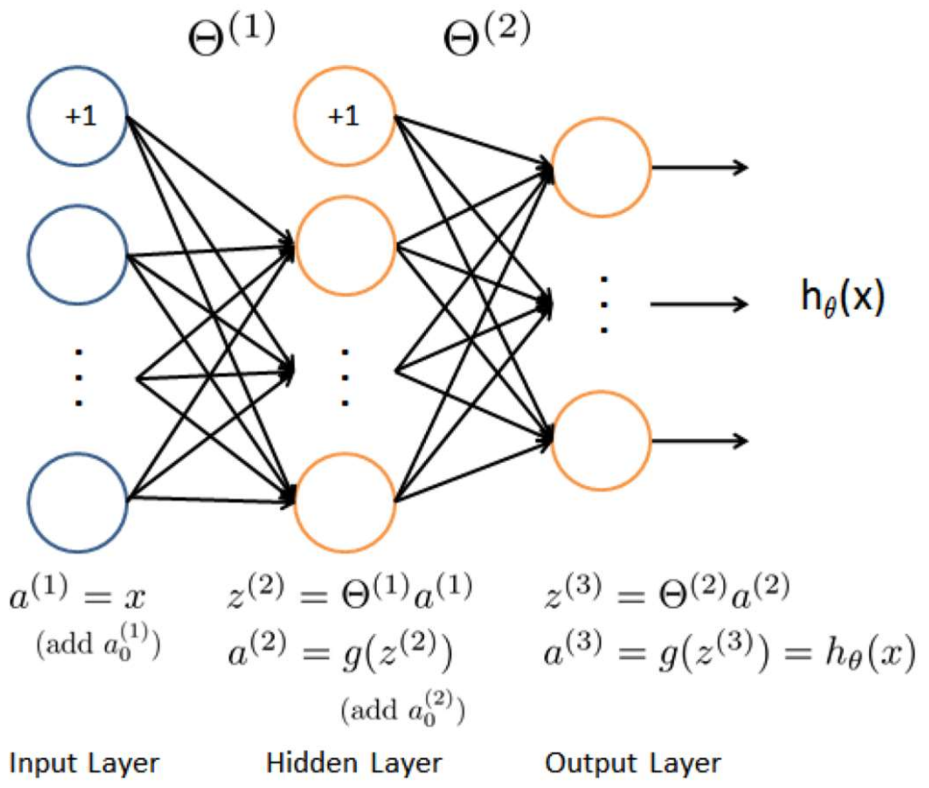
MNIST handwritten digit dataset (http://yann.lecun.com/exdb/mnist/). 5000 training examples (20x20 pixels image, indicating the grayscale color intensity). The image is transformed into a row vector (with 400 elements). This gives 5000 x 400 data matrix X (every row is a training example).



NN model - MNIST problem

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).

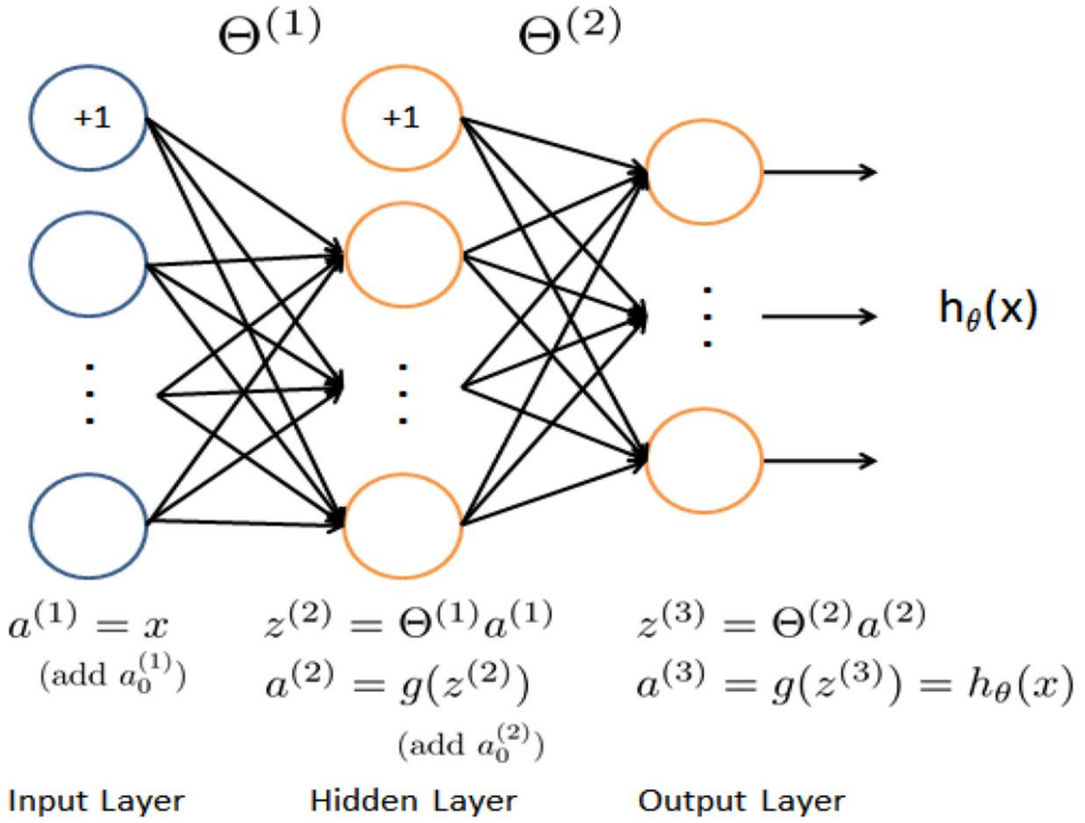
Matrix parameters: Θ_1 has size 25x401; Θ_2 has size 10x26.



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

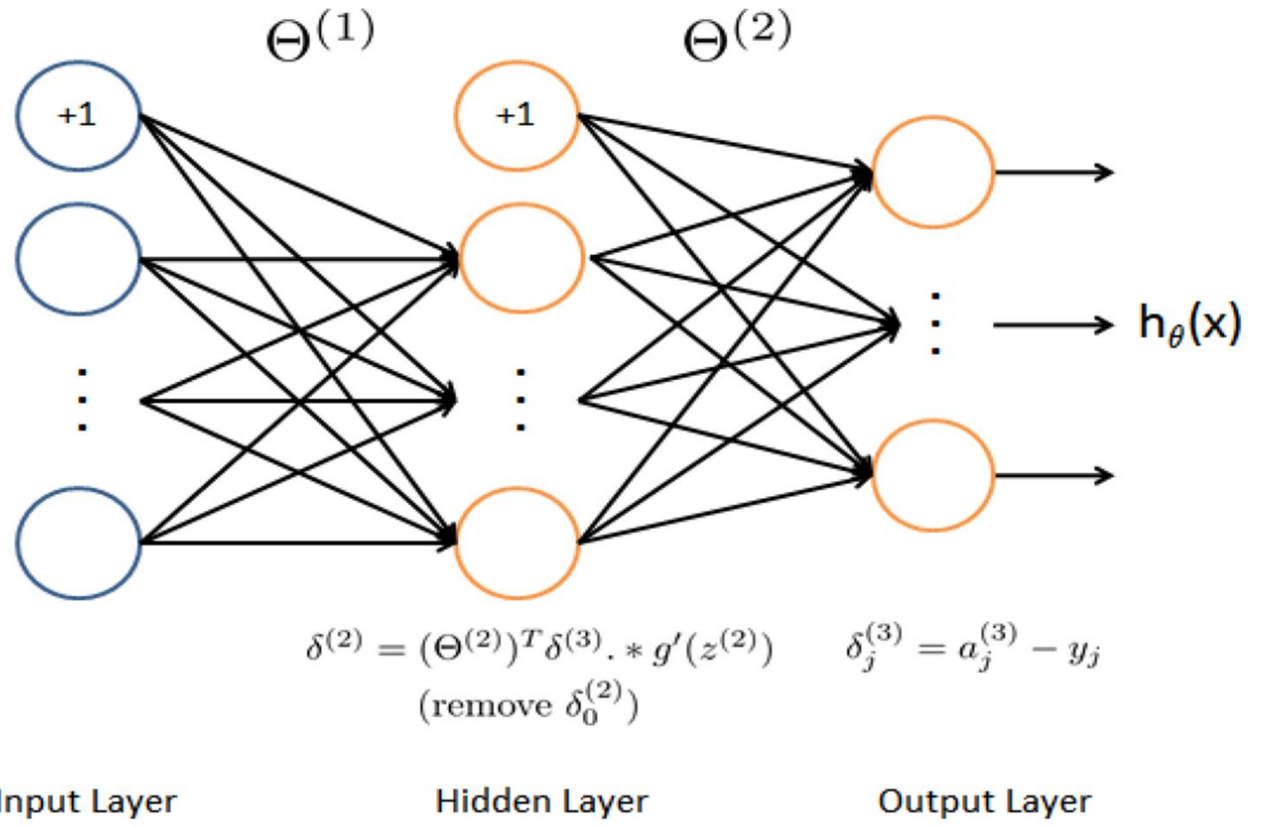
NN model learning – forward pass

- Randomly initialize the NN parameters (matrices Q_1 and Q_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)



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.



$\Theta^{(1)}$
$\Theta^{(2)}$
+1
+1
$h_\theta(x)$
$\delta^{(2)} = (\Theta^{(2)})^T \delta^{(3)} .* g'(z^{(2)})$
(remove $\delta_0^{(2)}$)
$\delta_j^{(3)} = a_j^{(3)} - y_j$
Input Layer
Hidden Layer
Output Layer

Error Backpropagation algorithm

0) Randomly initialize the parameters (Θ_1 and Θ_2)
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: $$\Theta_{ij}^{(l)} = \Theta_{ij}^{(l)} - \alpha \frac{\partial J(\Theta)}{\partial \Theta_{ij}^{(l)}}$$

universidade de aveiro

Regularized Cost Function

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

Regularization term

$$\boxed{\frac{\lambda}{2m} \left[\sum_{j=1}^{25} \sum_{k=1}^{400} (\Theta_{j,k}^{(1)})^2 + \sum_{j=1}^{10} \sum_{k=1}^{25} (\Theta_{j,k}^{(2)})^2 \right]}$$

After computing the gradient by backpropagation, add the regularization term

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

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

universidade de aveiro

NN Parameters (weights) Initialization

- Setting the weights to zero (Simplest approach)
However, by initializing every weight to zero, every neuron will have the same activations, all the calculated gradients will be the same, and consequently, each parameter will suffer the same update. Therefore, it is crucial that the initialization of the weights breaks the symmetry between different units.

- Drawn from random Gaussian distribution with mean 0 & deviation 1
Would break such symmetry but would also mean that some parameters would have much higher values than others, which would eventually lead to problems like the vanishing or exploding gradients problem.

-**Xavier/ Glorot's initialization:** drawn from uniform distribution near zero.

$$\sim U(-\frac{\sqrt{6}}{\sqrt{m}}, \frac{\sqrt{6}}{\sqrt{m}})$$

- **LeCun initialization:**

$$\sim U(-\frac{\sqrt{3}}{\sqrt{m}}, \frac{\sqrt{3}}{\sqrt{m}})$$

- **Lab Init:** Incoming and Outgoing connections (Input & Output layer size)

universidade de aveiro