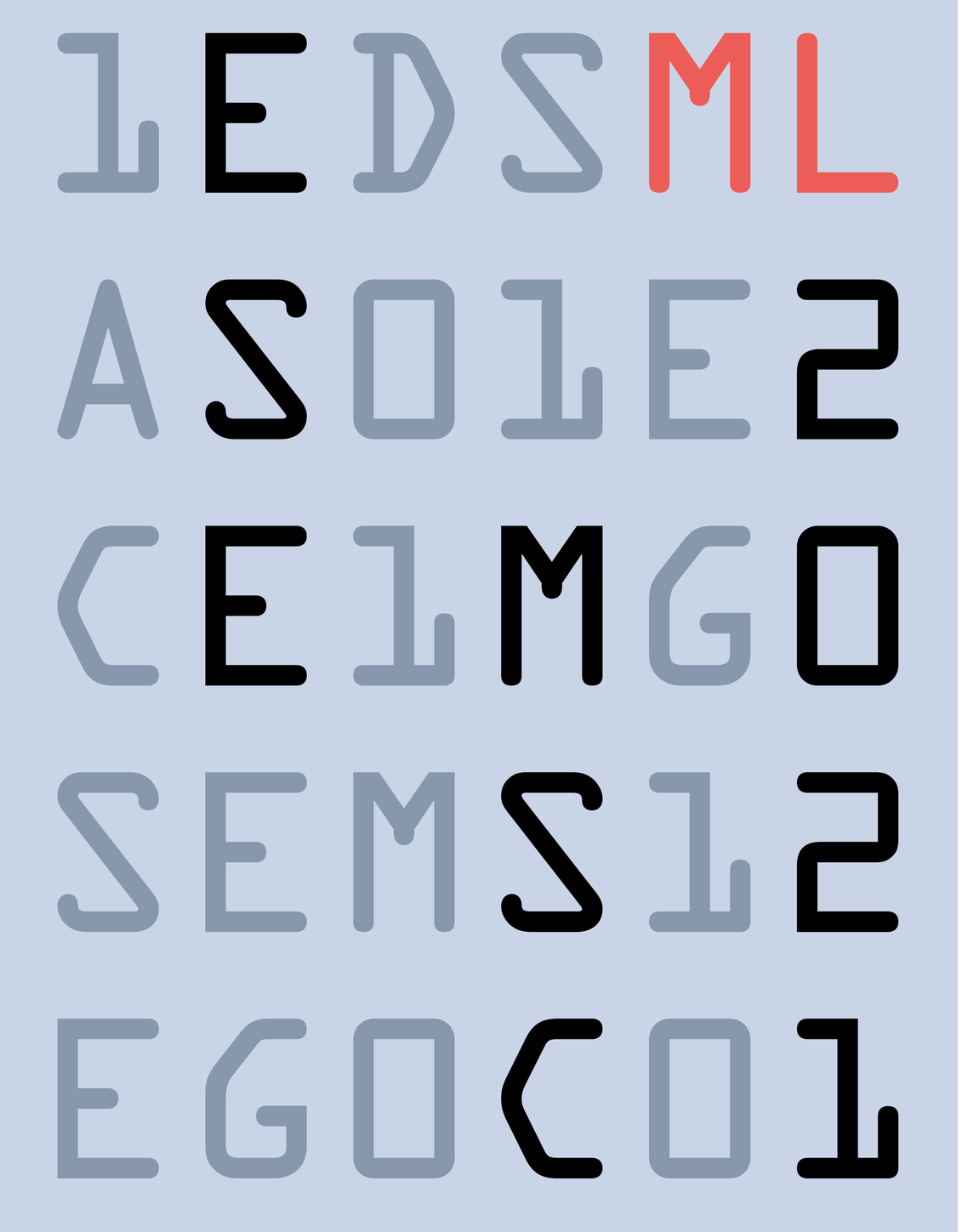
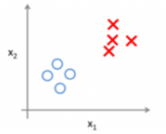


1-Introduction to Machine Learning [RECAP]

Lluis Guasch

SUPERVISED VS UNSUPERVISED LEARNING

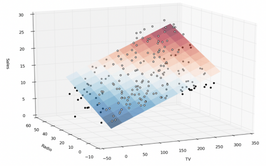
Summary:



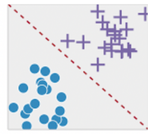
SUPERVISED LEARNING

Training Set: Data x with labels y

- **Regression**: label is a real number/vector

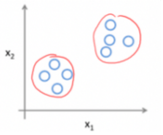


- **Classification**: label is a class or integer number/vector

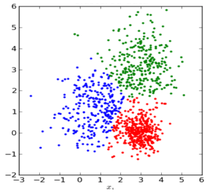


UNSUPERVISED LEARNING

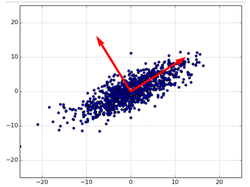
Training Set: Just Data x



- **Clustering**: organise data in groups



- **Dimensionality reduction**: compress data



Supervised Learning Recap

Linear regression:

$$h_{\boldsymbol{\theta}}(\mathbf{x}) = \boldsymbol{\theta}_0 + \boldsymbol{\theta}_1 \mathbf{x}$$

Cost or loss function:

$$J(\boldsymbol{\theta}_0, \boldsymbol{\theta}_1) = \frac{1}{2m} \sum_{i=1}^{m} (y^i - h(x^i))^2 = \frac{1}{2m} \sum_{i=1}^{m} (y^i - \boldsymbol{\theta}_0 - \boldsymbol{\theta}_1 x^i)^2$$

Solution:

$$\theta_0^* = \frac{\left(\frac{1}{m}\sum_{i=1}^{m} x^{(i)2}\right)\left(\frac{1}{m}\sum_{i=1}^{m} y^{(i)}\right) - \left(\frac{1}{m}\sum_{i=1}^{m} x^{(i)}\right)\left(\frac{1}{m}\sum_{i=1}^{m} x^{(i)} y^{(i)}\right)}{\left(\frac{1}{m}\sum_{i=1}^{m} x^{(i)2}\right) - \left(\frac{1}{m}\sum_{i=1}^{m} x^{(i)}\right)^2} \quad \theta_1^* = \frac{\frac{1}{m}\sum_{i=1}^{m} x^{(i)} y^{(i)} - \left(\frac{1}{m}\sum_{i=1}^{m} x^{(i)}\right)\left(\frac{1}{m}\sum_{i=1}^{m} y^{(i)}\right)}{\left(\frac{1}{m}\sum_{i=1}^{m} x^{(i)2}\right) - \left(\frac{1}{m}\sum_{i=1}^{m} x^{(i)}\right)^2}$$

Logistic regression:

Cost function using for logistic regression:

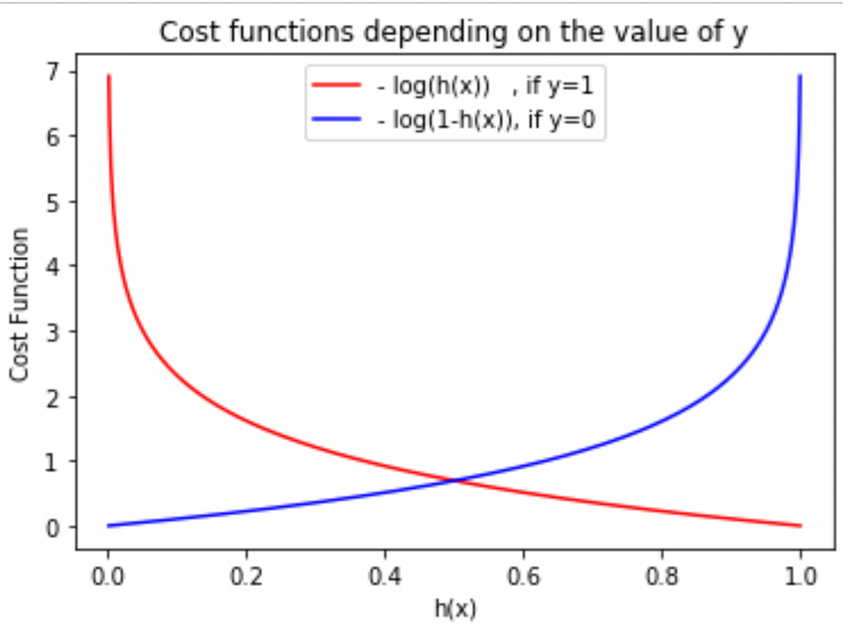
For one data point , how to measure the discrepancy between the actual data value y (equal to 0 or 1) and the estimated probability $h_{\boldsymbol{\theta}}(\boldsymbol{x})$?

if $(y = 1)$

$$C(h_{\boldsymbol{\theta}}(\boldsymbol{x}), y) = -\log(h_{\boldsymbol{\theta}}(\boldsymbol{x}))$$

if $(y = 0)$

$$C(h_{\boldsymbol{\theta}}(\boldsymbol{x}), y) = -\log(1 - h_{\boldsymbol{\theta}}(\boldsymbol{x}))$$



Cost or loss function:

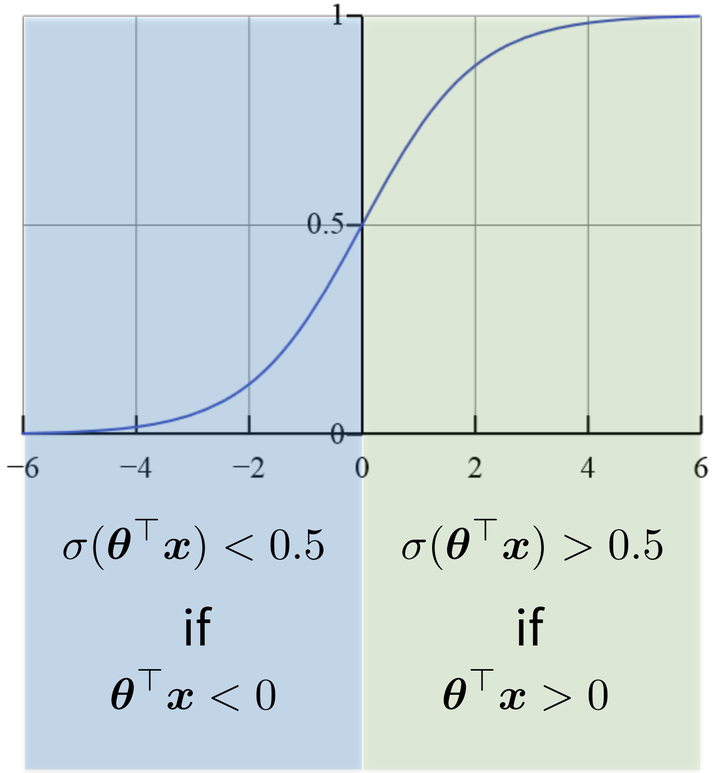
$$C(h_{\boldsymbol{\theta}}(\boldsymbol{x}), y) = -y\log(h_{\boldsymbol{\theta}}(\boldsymbol{x})) - (1 - y)\log(1 - h_{\boldsymbol{\theta}}(\boldsymbol{x}))$$

SUPERVISED LEARNING RECAP

Logistic regression:

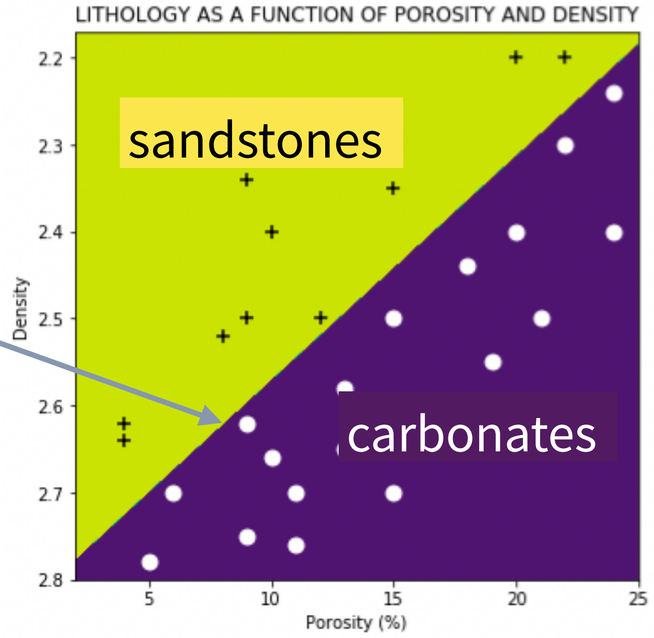
Definition of the **Decision Boundary**:

$$P(y = 1|\boldsymbol{\theta}, \boldsymbol{x}) = \sigma(\boldsymbol{\theta}^\top \boldsymbol{x})$$



Decision boundary satisfies:

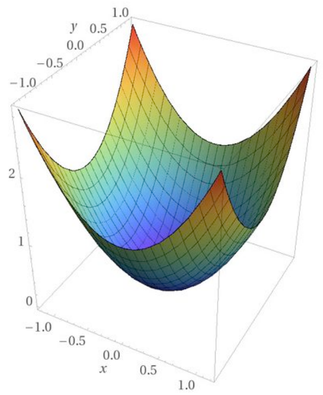
$$\boldsymbol{\theta}^\top \boldsymbol{x} = 0$$



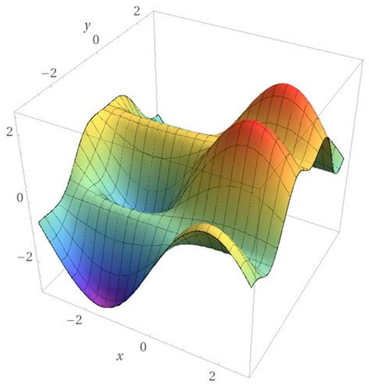
Supervised Learning Recap

Linear regression and logistic regression can be interpreted as the simplest forms of machine learning.

They both rely on assuming a linear relation between data and labels, but in the real world data and labels are not necessarily linearly related.



cost functions



Supervised Learning Recap

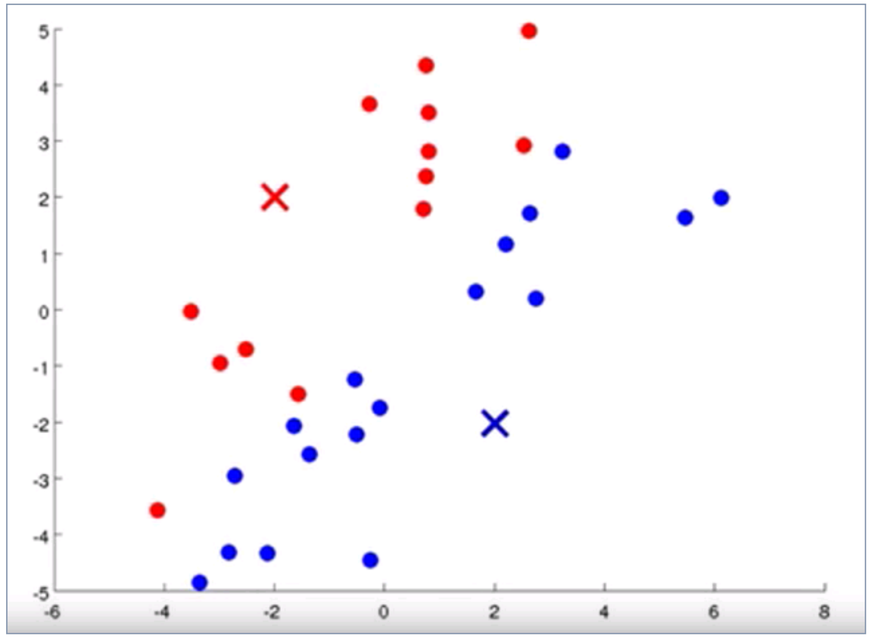
Even if we use **higher order polynomials** to describe the relations between input data and labels, we are still imposing structure on the shape of this relation.

The idea behind machine learning is to relax these imposed constraints and create a hypothesis function (a network) that can adapt (**learn**) its parameters to capture the relation between the data and labels with more flexibility.

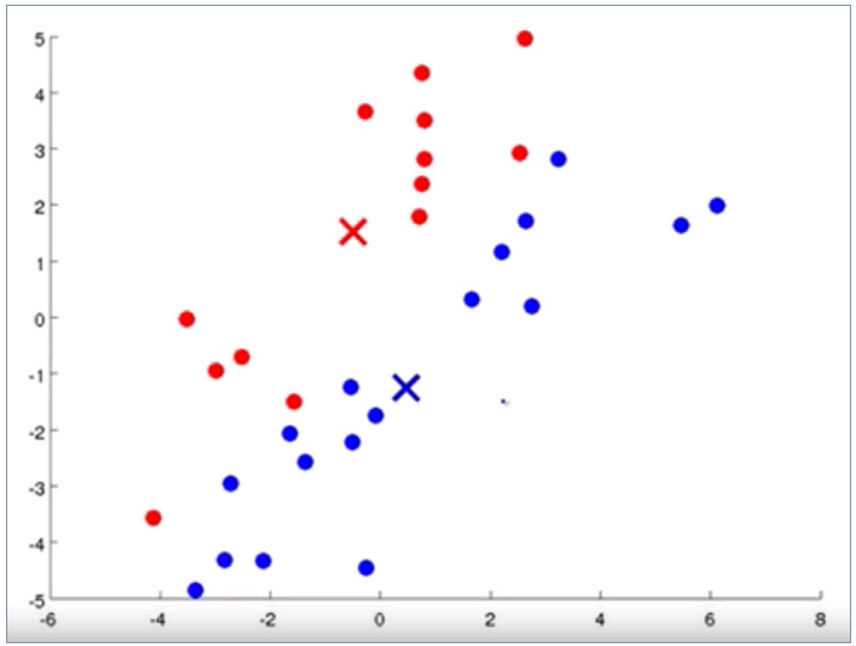
1EDSML
ASO1E2
CE1MGO
SEMS12
EGOCO1

k-means recap:

Visual description of the k-means algorithm:



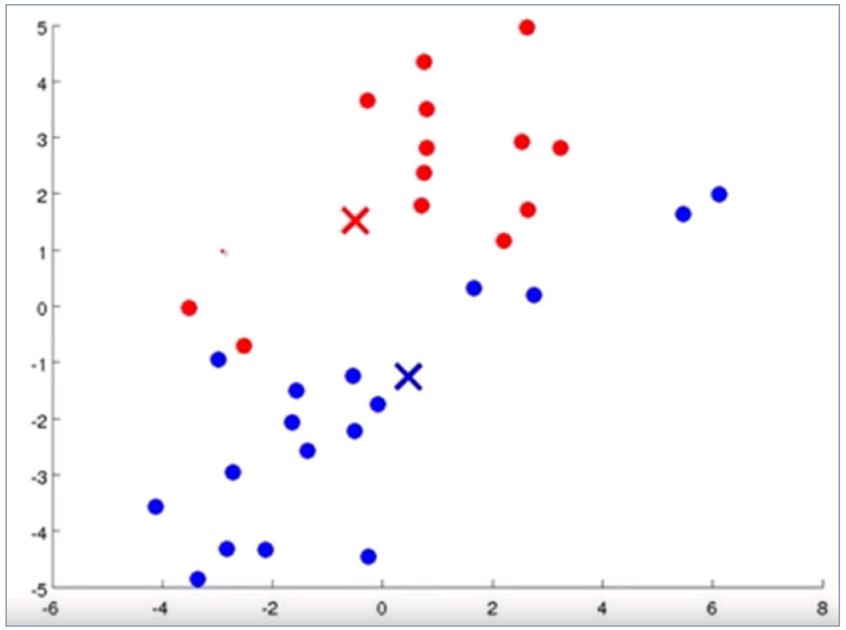
cluster assignment 1



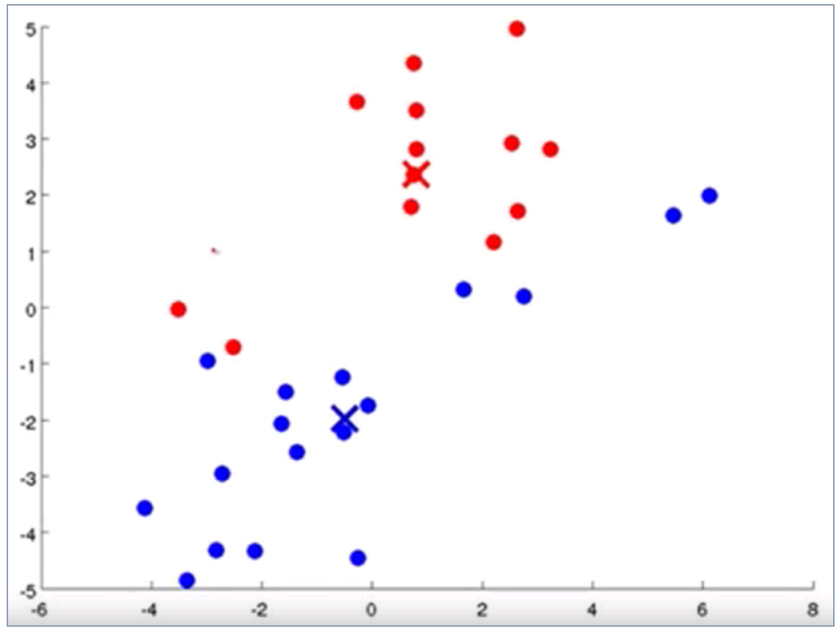
centroid assignment 1

k-means recap:

Visual description of the k-means algorithm:



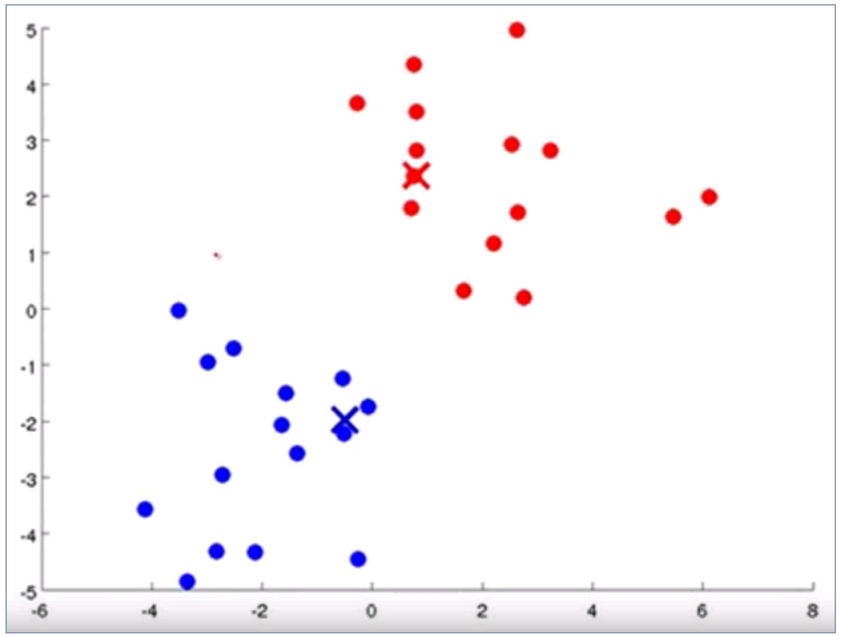
cluster assignment 2



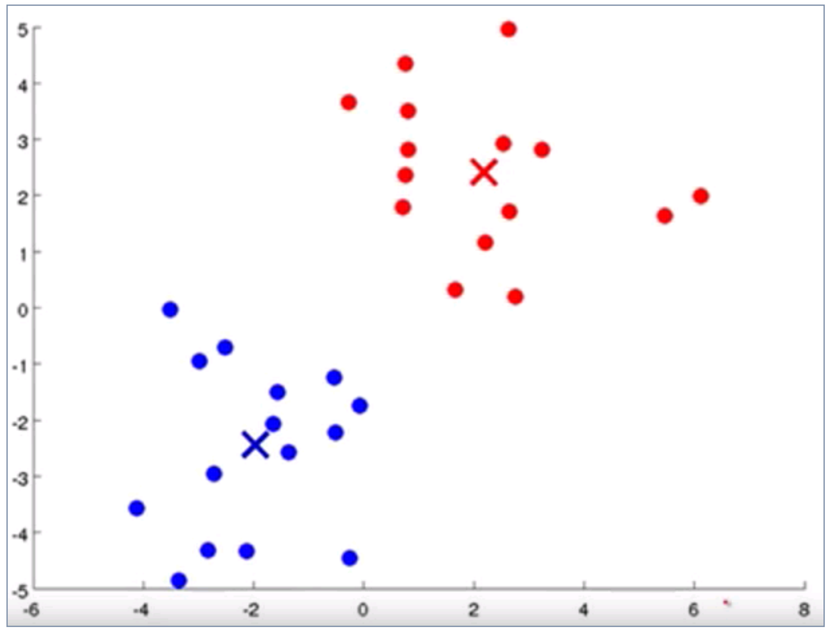
centroid assignment 2

k-means recap:

Visual description of the k-means algorithm:



cluster assignment 3



final centroids
(**in this example**, they don't change position anymore)

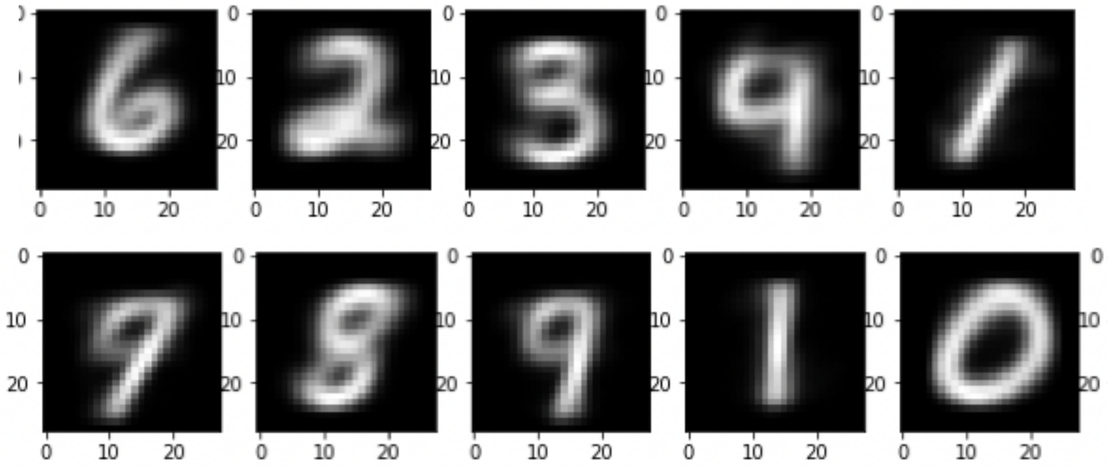
k-means recap:

Cost function:

$$J\left(c^{(1)}, \dots, c^{(m)}, \mu_1, \dots, \mu_k\right) = \frac{1}{m}\sum_{i=1}^{m}\left\|x^{(i)} - \mu_{c^{(i)}}\right\|^2$$

- $c^{(1)}, \dots, c^{(m)}$: Classes associated with each data point
- $\mu_1, \dots, \mu_k$: Class centroids
- $x^{(i)}$: Data Points
- $\mu_{c^{(i)}}$: Centroid of $x^{(i)}$'s class

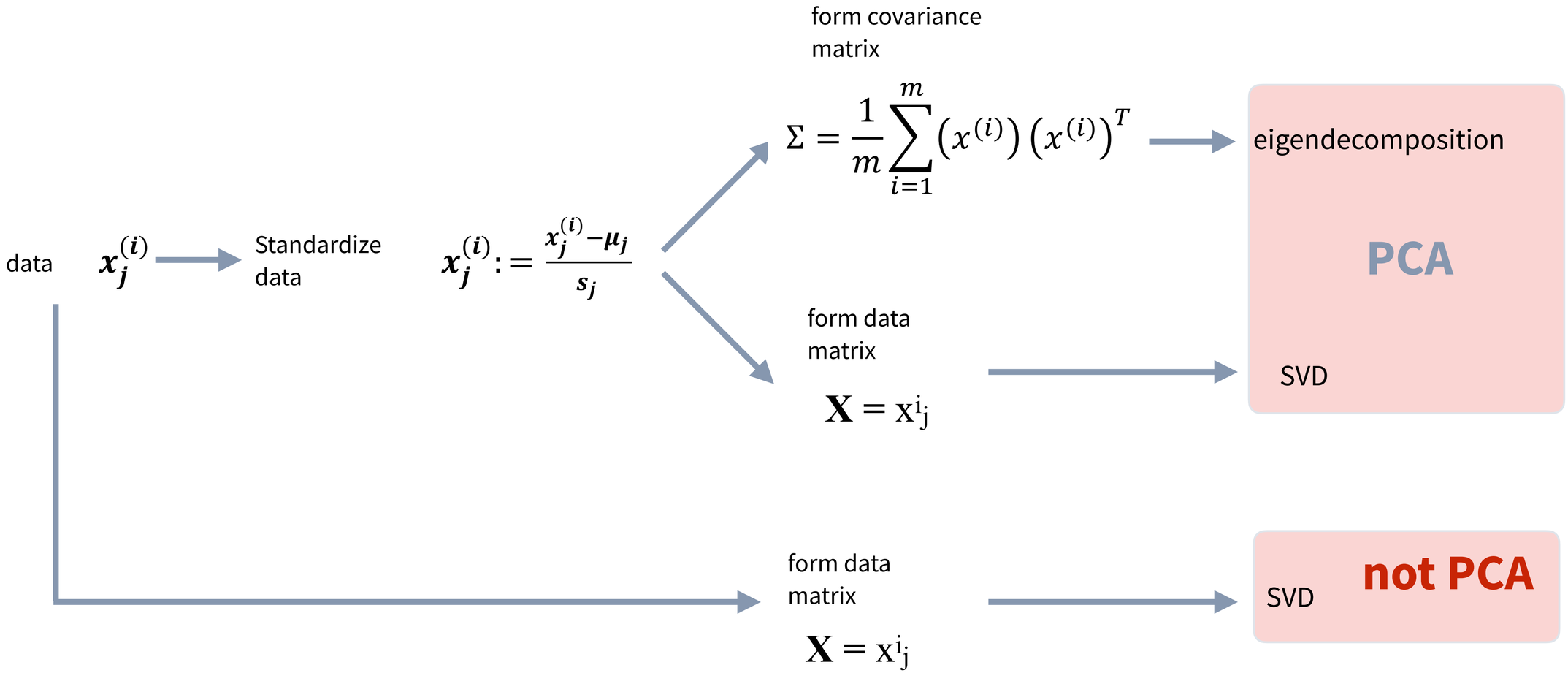
Each image is a centroid (class mean)



How can we improve this poor result? With **PCA**

PCA

The cost function in PCA is the *Rayleigh quotient* (but we won't go into any more mathematical details here). Just remember that:



PCA

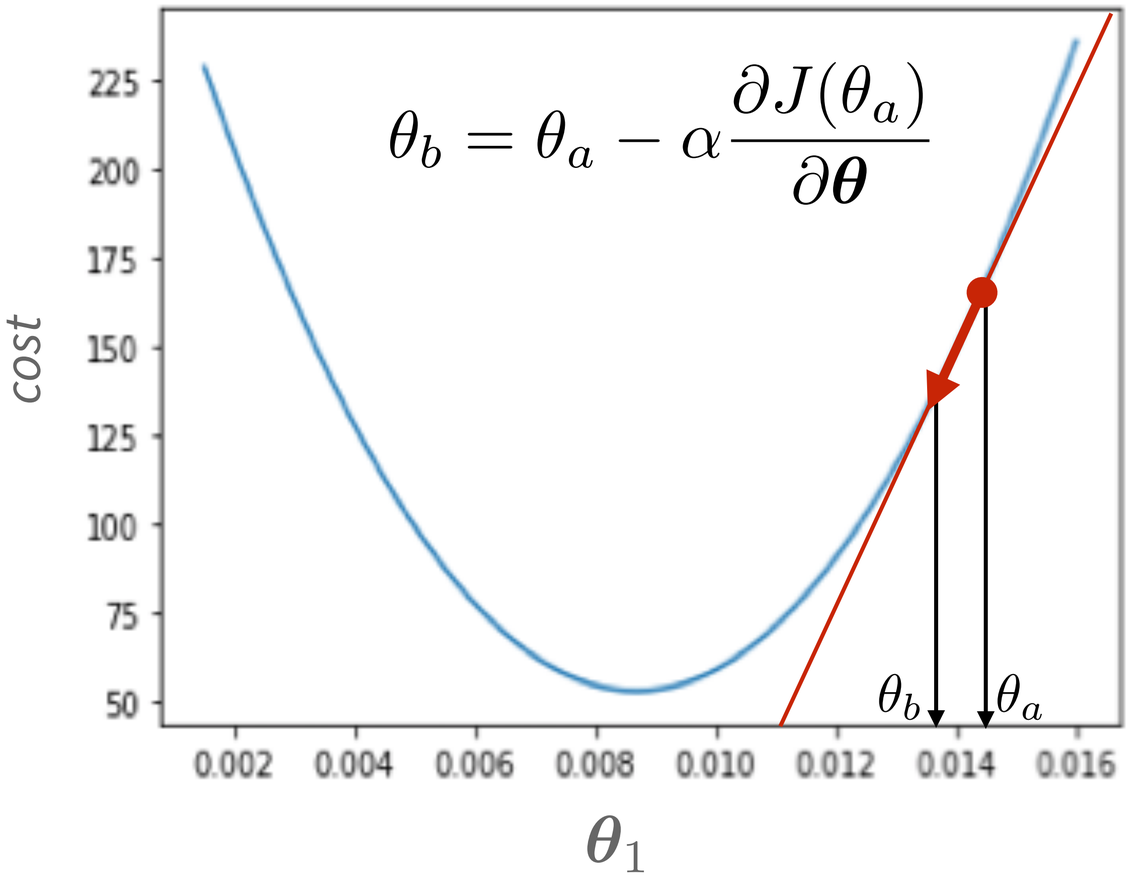
PCA is used to:

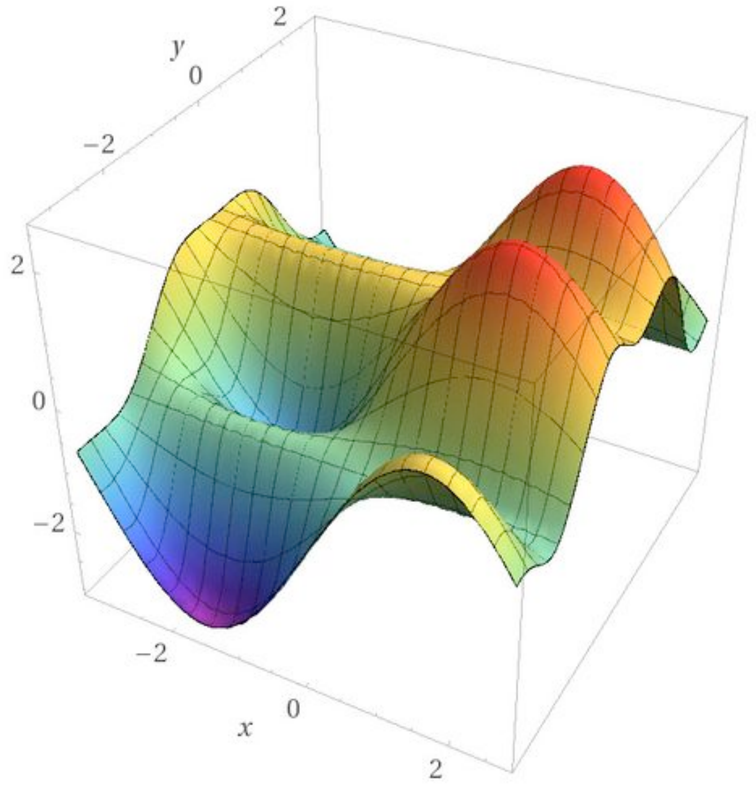
1. Reduce data dimensionality.
2. Represent data in an orthogonal basis (which means that the principal components or PCs have 0 collinearity).
3. Perfect collinearity between variables results in less available PCs.
4. Potential to be used in combined with other methods we have seen, as we saw in yesterday's practical when we applied it to regression and k-means.

Supervised VS Unsupervised Learning

Both supervised and unsupervised learning rely on defining a **cost** or **loss function** that we try to minimise using **gradient descent**.

Cost function





- In simple cases like **linear regression, we do not need to use gradient descent** because we can compute the solution in one single operation (with the normal equations):

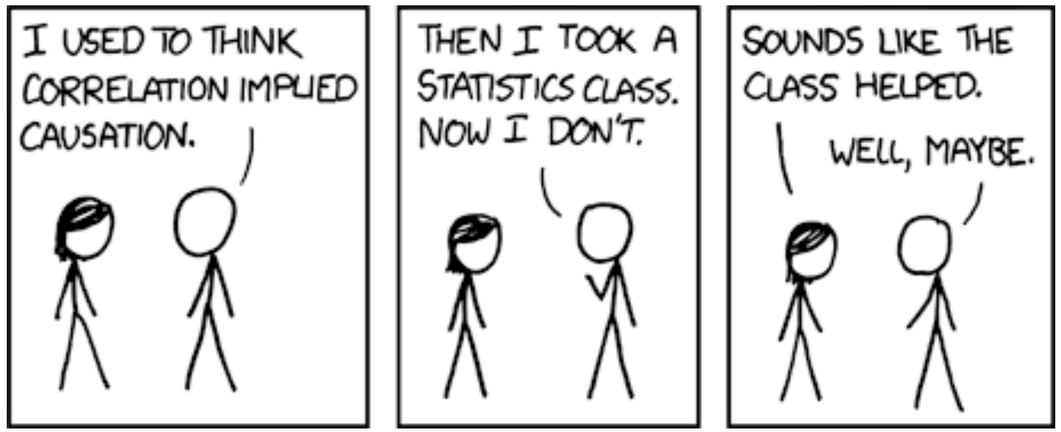
$$\boldsymbol{\theta}_1{}^* = \frac{\sum_{i=1}^{m} x^i y^i}{\sum_{i=1}^{m} (x^i)^2}$$

but when we start using networks we will always need to use some local optimisation algorithm to navigate the solution space (often referred to as **loss landscape in ML**).

- In addition, for some problems with large datasets, it may be more **computationally efficient** to apply gradient descent than solve the normal equations in linear-regression-type problems. A good discussion on the topic here:

https://stats.stackexchange.com/questions/160179/do-we-need-gradient-descent-to-find-the-coefficients-of-a-linear-regression-mode

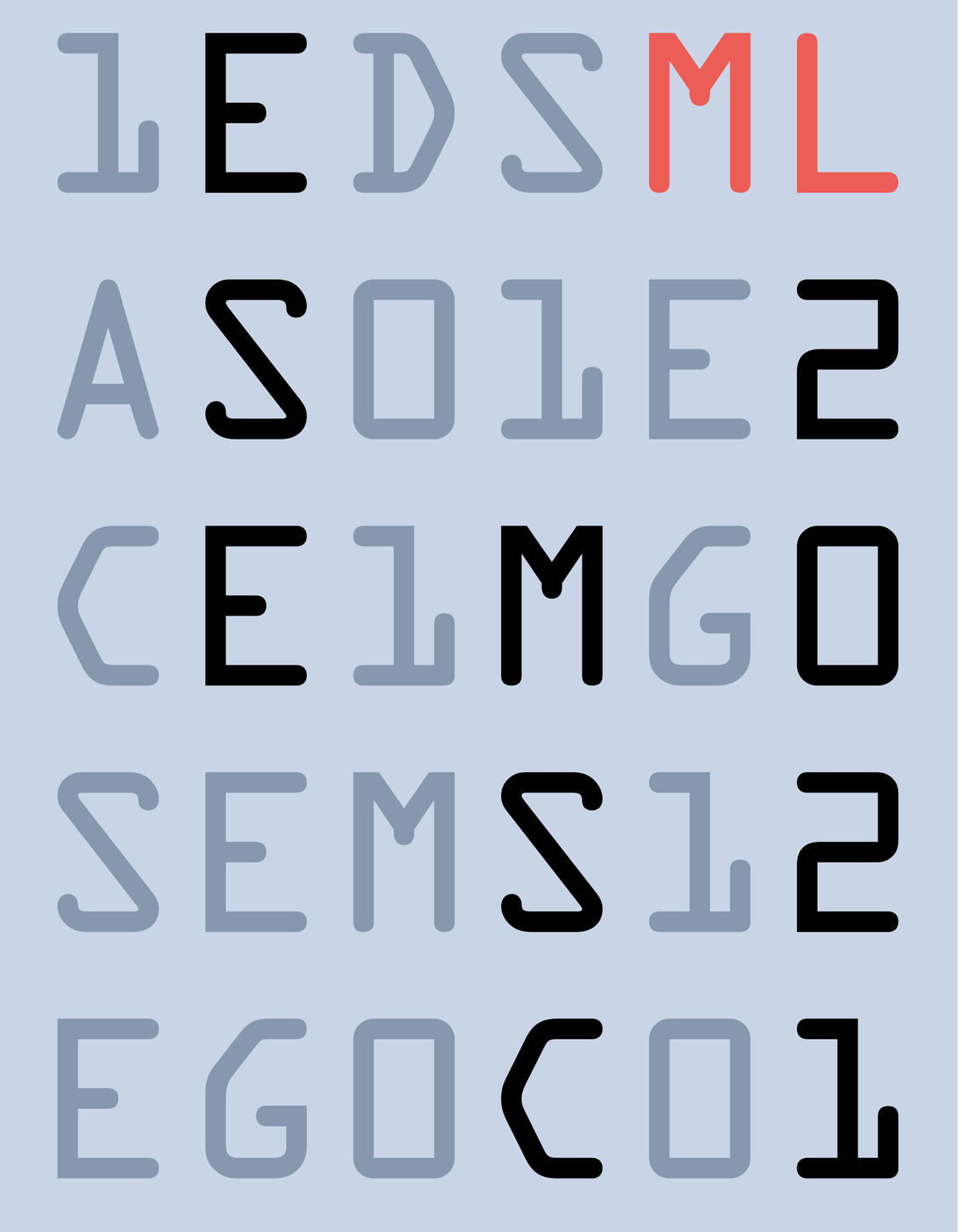
Final remark: casual vs statistical inference



In **ML** we are trying to find **statistical relations in the data.**

The interpretation of what this relations mean (causal inference) is then performed by humans.

https://stats.stackexchange.com/questions/233235/what-distinction-is-there-between-statistical-inference-and-causal-inference#:~:text=Causal%20inference%20is%20the%20process,considered%20to%20be%20causal%20explanation



2-Feed-Forward Networks

Lluis Guasch

GTAs online:
Yao
Alex

Feed-Forward Neural Networks

1. From Logistic Regression to Single Neuron Representation
2. Feed-Forward Neural Networks
3. Back-Propagation
4. Batch, Stochastic and Mini-Batch Gradient Descent
5. Some basic architectures

Objectives of the day:

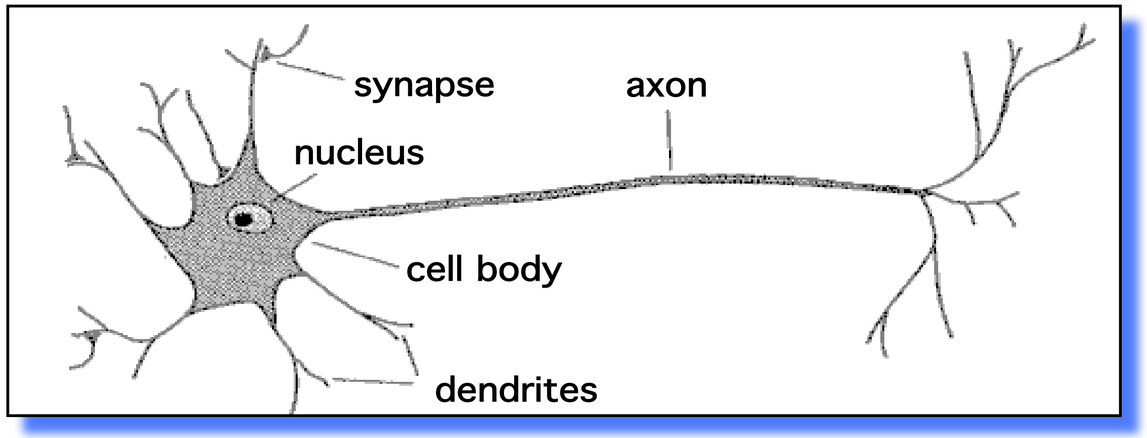
- Focus on supervised learning
- Present logistic regression as a single neuron operation
- Generalize to feed-forward neural networks
- Illustrate the backpropagation algorithm
- Introduce stochastic gradient descent
- Examples

Feed-Forward Neural Networks

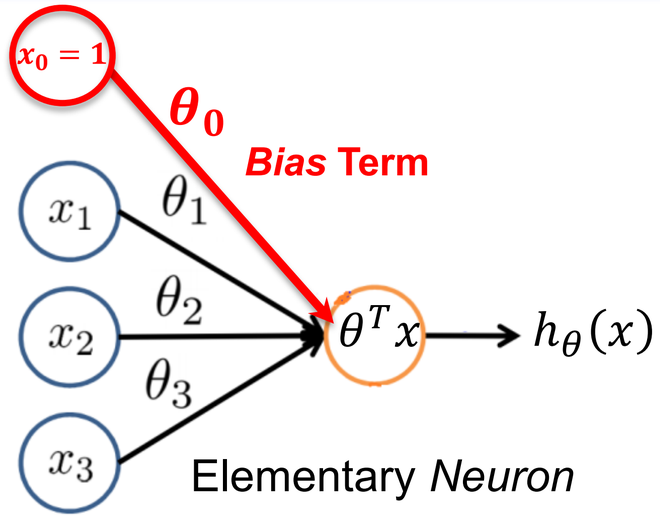
1. From Logistic Regression to Single Neuron Representation
2. Feed-Forward Neural Networks
3. Back-Propagation
4. Batch, Stochastic and Mini-Batch Gradient Descent
5. Some basic architectures

1EDSML
AS01E2
CE1MG0
SEMS12
EG0C01

Logistic regression analogy with human neuron:



- A neuron has
  - Branching input (dendrites)
  - Branching output (the axon)



Elementary *Neuron*

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

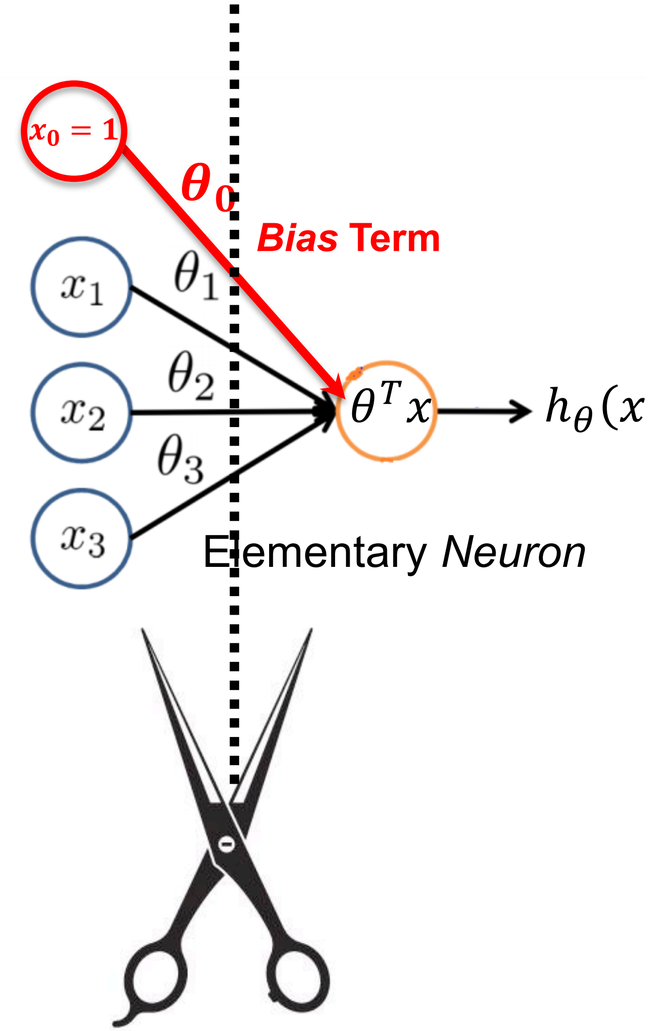
Parameters θ_i's also called *weights*

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

σ also called *activation function* g

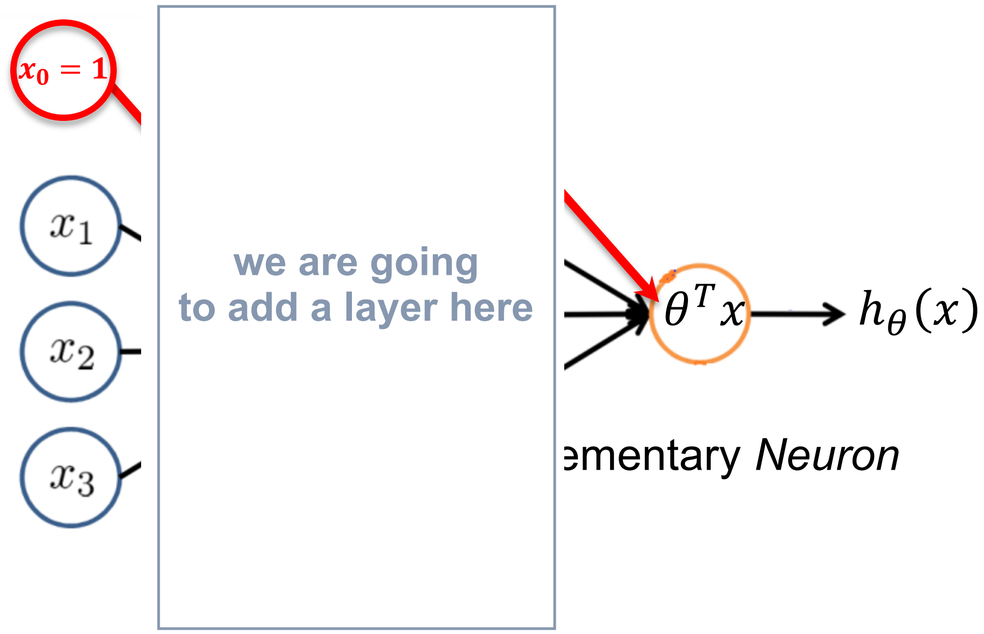
Neurons in Networks

From single neuron to feed-forward neural networks:



Neurons in Networks

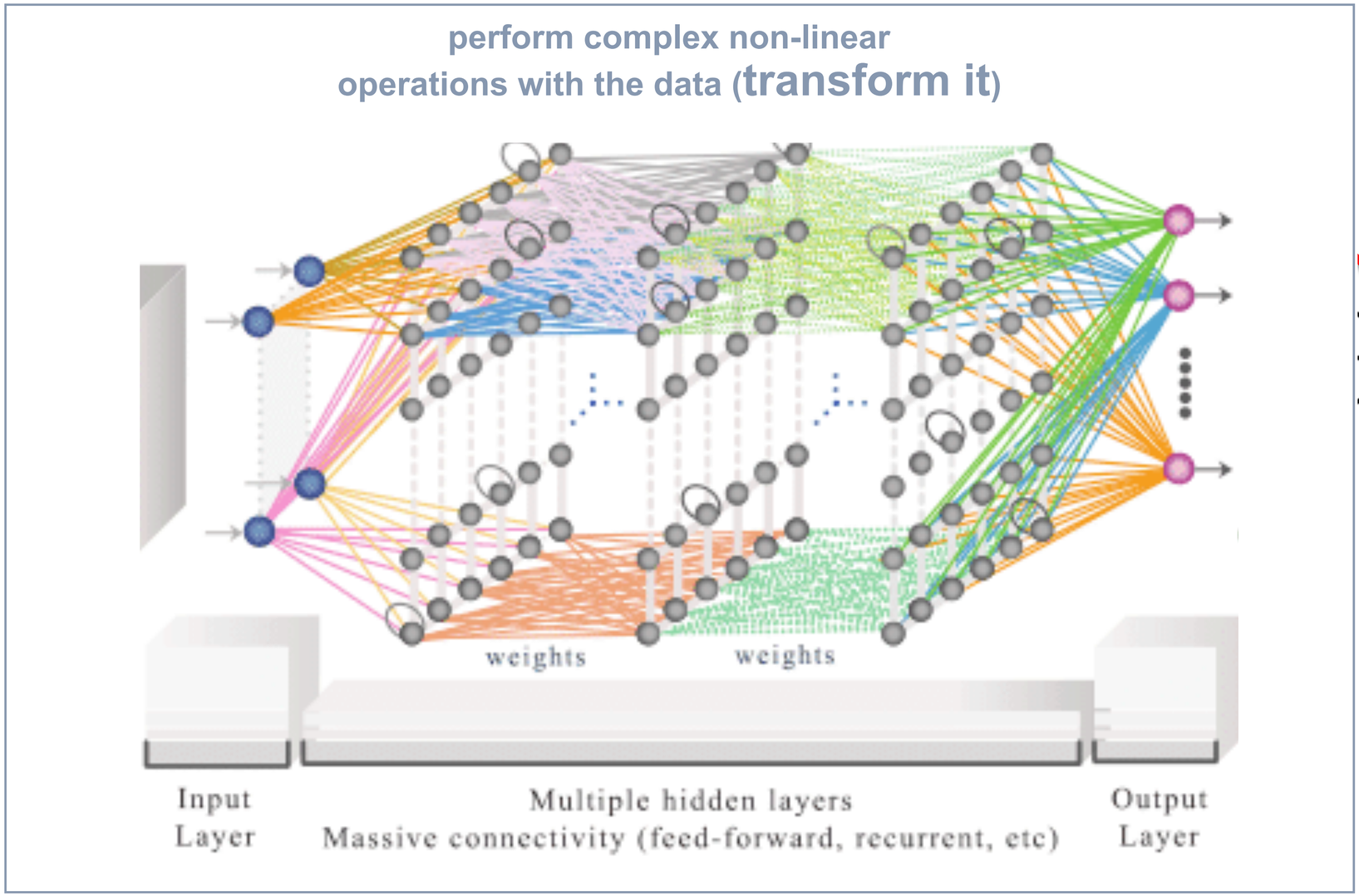
From single neuron to feed-forward neural networks:



ementary *Neuron*

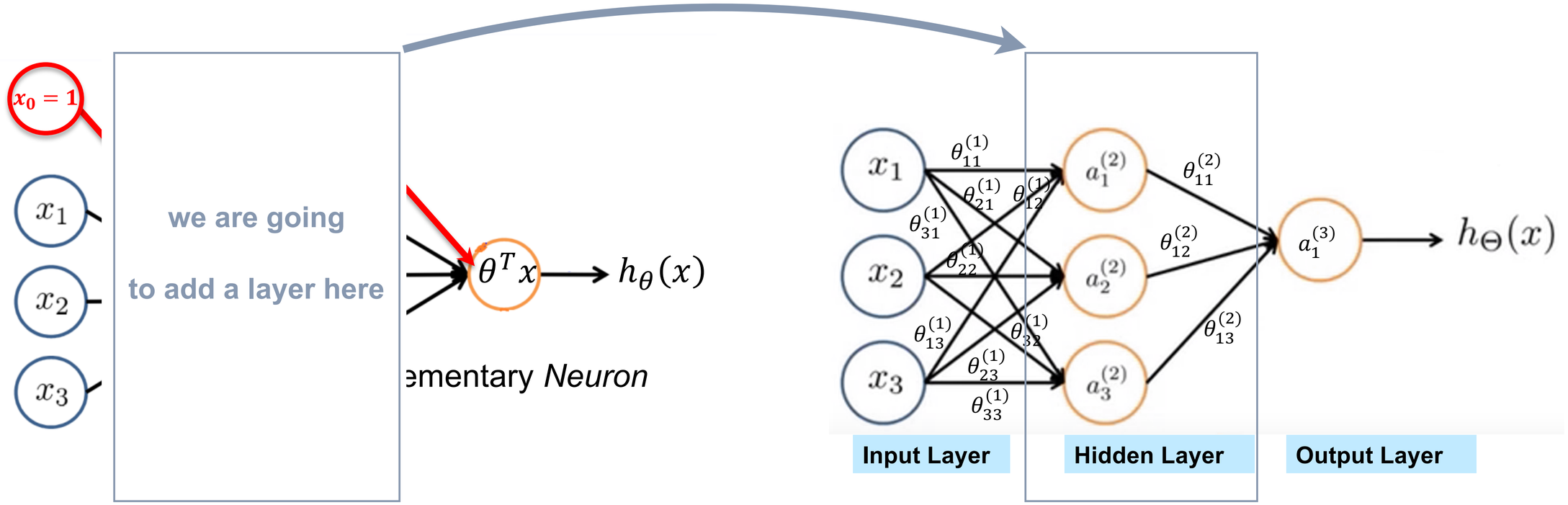
Neurons in Networks

…and ML and DL in general do basically this



1EDSML
ASO1E2
CE1MGO
SEMS12
EGOCO1

From single neuron to feed-forward neural networks:

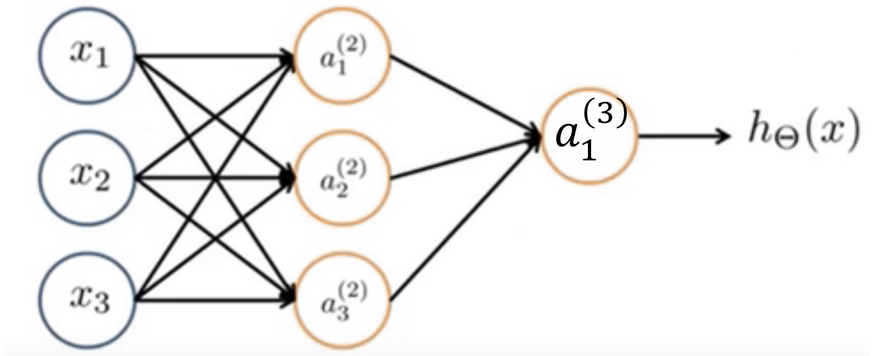


*Also called **"Fully-Connected Networks".** Historically this is a generalization of the Multi-Layer Perceptron or MLP, which was the first network capable of learning its weights (Rosenblatt, 1961).*

Feed-Forward Neural Networks

1. From Logistic Regression to Single Neuron Representation
2. **Feed-Forward Neural Networks**
3. Back-Propagation
4. Batch, Stochastic and Mini-Batch Gradient Descent
5. Some basic architectures

Simple Networks in Matrix Form



Layer $l = 1$ Layer $l = 2$ Layer $l = 3$

$a^{(l)} =$ *Activation* vector of layer l

$\theta^{(l)} =$ *Matrix of weights controlling mapping from layer l to layer l+1*
$\theta_{jk}^{(l)}$ *= weight from neuron k in layer* (l) *to neuron* j *in layer* $(l+1)$

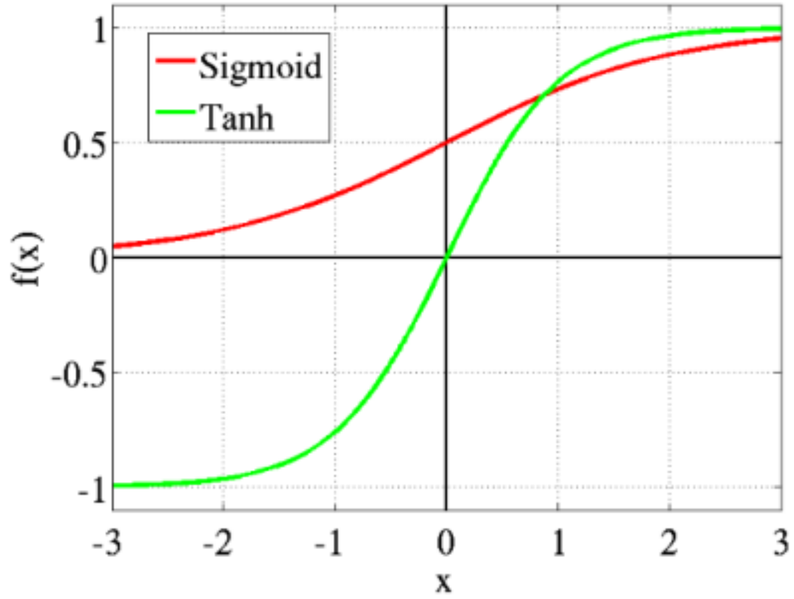
For example (if we have a bias term):

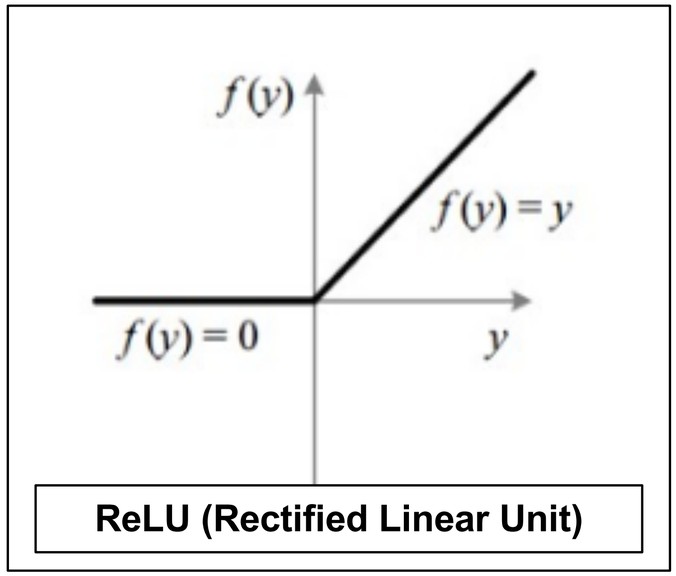
$$a^{(2)} = \begin{pmatrix} a_1^{(2)} \\ a_2^{(2)} \\ a_3^{(2)} \end{pmatrix} = g\left(\theta^{(1)} a^{(1)}\right) = g\left(\theta^{(1)} x\right) = g\left(\begin{pmatrix} \theta_{10}^{(1)} & \theta_{11}^{(1)} & \theta_{12}^{(1)} & \theta_{13}^{(1)} \\ \theta_{20}^{(1)} & \theta_{21}^{(1)} & \theta_{22}^{(1)} & \theta_{23}^{(1)} \\ \theta_{30}^{(1)} & \theta_{31}^{(1)} & \theta_{32}^{(1)} & \theta_{33}^{(1)} \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{pmatrix}\right)$$

g *is the "Activation Function"*

1EDSML
AS01E2
CE1MG0
SEMS12
EG0C01

Classic activation functions:

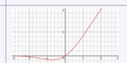
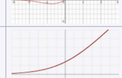
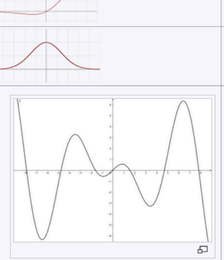




Tanh *is between -1 and +1 with mean zero (instead of mean 0.5 for sigmoid function).*
ReLU *is such that gradient does not vanish for non-zero values, most often used activation function in practice.*
Sigmoid σ *between 0 and 1, mostly used for output layer of binary classification problems, in order to allow a probabilistic interpretation of the result.*

Activation Functions

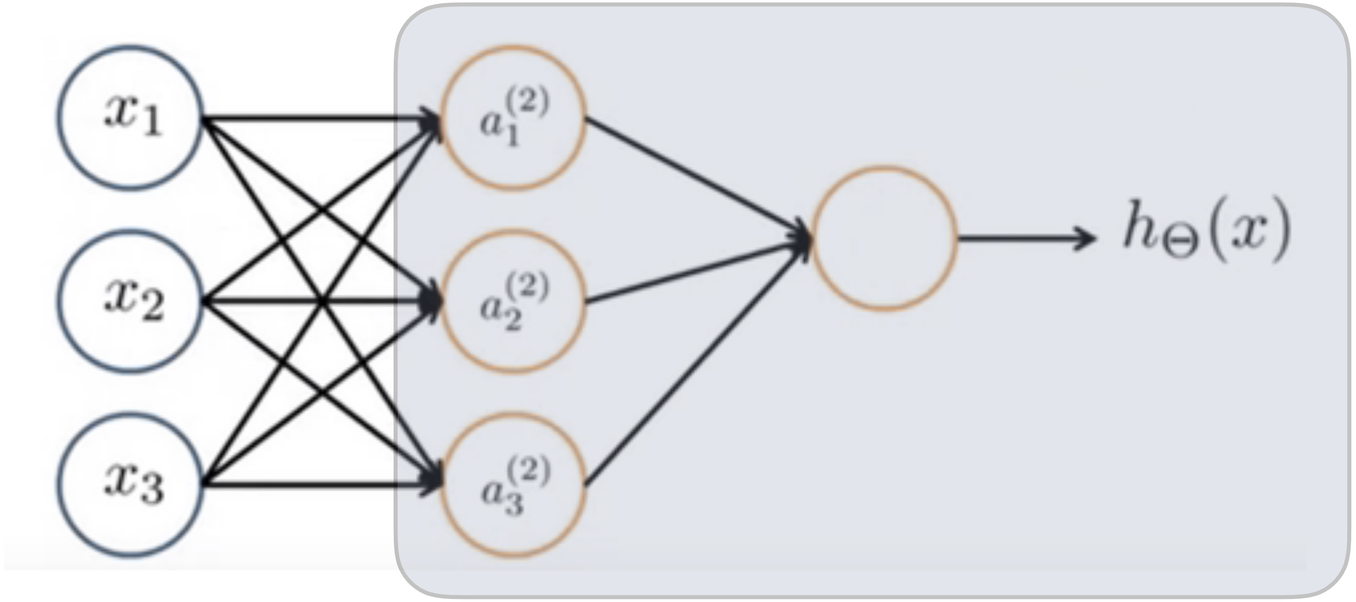
But there are many more:

Name	Plot	Function, $g(x)$	Derivative of g, $g'(x)$	Range	Order of continuity
Identity		x	1	$(-\infty, \infty)$	C^∞
Binary step		$\begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases}$	$\begin{cases} 0 & \text{if } x \neq 0 \\ \text{undefined} & \text{if } x = 0 \end{cases}$	$\{0, 1\}$	C^{-1}
Logistic, sigmoid, or soft step		$\sigma(x) = \frac{1}{1 + e^{-x}}$	$g(x)(1 - g(x))$	$(0, 1)$	C^∞
Hyperbolic tangent (tanh)		$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$	$1 - g(x)^2$	$(-1, 1)$	C^∞
Rectified linear unit (ReLU)[10]		$\begin{cases} 0 & \text{if } x \leq 0 \\ x & \text{if } x > 0 \end{cases}$ $= \max\{0, x\} = x\mathbf{1}_{x>0}$	$\begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x > 0 \\ \text{undefined} & \text{if } x = 0 \end{cases}$	$[0, \infty)$	C^0
Gaussian Error Linear Unit (GELU)[7]		$\frac{1}{2}x\left(1 + \operatorname{erf}\left(\frac{x}{\sqrt{2}}\right)\right)$ $= x\Phi(x)$	$\Phi(x) + x\phi(x)$	$(-0.17\ldots, \infty)$	C^∞
Softplus[11]		$\ln(1 + e^x)$	$\frac{1}{1 + e^{-x}}$	$(0, \infty)$	C^∞
Exponential linear unit (ELU)[12]		$\begin{cases} \alpha(e^x - 1) & \text{if } x \leq 0 \\ x & \text{if } x > 0 \end{cases}$ with parameter α	$\begin{cases} \alpha e^x & \text{if } x < 0 \\ 1 & \text{if } x > 0 \\ 1 & \text{if } x = 0 \text{ and } \alpha = 1 \end{cases}$	$(-\alpha, \infty)$	$\begin{cases} C^1 & \text{if } \alpha = 1 \\ C^0 & \text{otherwise} \end{cases}$
Scaled exponential linear unit (SELU)[13]		$\lambda\begin{cases} \alpha(e^x - 1) & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases}$ with parameters $\lambda = 1.0507$ and $\alpha = 1.67326$	$\lambda\begin{cases} \alpha e^x & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases}$	$(-\lambda\alpha, \infty)$	C^0
Leaky rectified linear unit (Leaky ReLU)[14]		$\begin{cases} 0.01x & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases}$	$\begin{cases} 0.01 & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \\ \text{undefined} & \text{if } x = 0 \end{cases}$	$(-\infty, \infty)$	C^0
Parametric rectified linear unit (PReLU)[15]		$\begin{cases} \alpha x & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases}$ with parameter α	$\begin{cases} \alpha & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases}$	$(-\infty, \infty)$	C^0
Sigmoid linear unit (SiLU,[7] Sigmoid shrinkage,[16] SiL,[17] or Swish-1[18])		$\frac{x}{1 + e^{-x}}$	$\frac{1 + e^{-x} + xe^{-x}}{(1 + e^{-x})^2}$	$[-0.278\ldots, \infty)$	C^∞
Gaussian		e^{-x^2}	$-2xe^{-x^2}$	$(0, 1]$	C^∞
Growing Cosine Unit (GCU)[2][19][1][5][3]		$xcos(x)$	$cos(x) - xsin(x)$	$(-\infty, \infty)$	C^∞

Name	Equation, $g_i(\vec{x})$	Derivatives, $\frac{\partial g_i(\vec{x})}{\partial x_j}$	Range	Order of continuity
Softmax	$\frac{e^{x_i}}{\sum_{j=1}^{J} e^{x_j}}$ for $i = 1, \ldots, J$	$g_i(\vec{x})\left(\delta_{ij} - g_j(\vec{x})\right)$[1][2]	$(0, 1)$	C^∞
Maxout[20]	$\max_i x_i$	$\begin{cases} 1 & \text{if } j = \operatorname{argmax}_i x_i \\ 0 & \text{if } j \neq \operatorname{argmax}_i x_i \end{cases}$	$(-\infty, \infty)$	C^0

https://en.wikipedia.org/wiki/Activation_function

Logistic regression is the last layer of a network:



Limitations of logistic regression

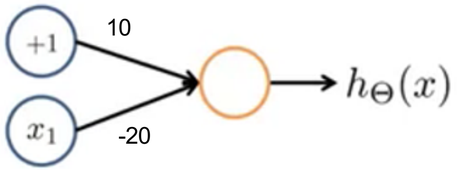
- There are some complex mathematical functions (such as the “Exclusive OR”) that Logistic Regression cannot represent

Logic Gates

Name	NOT	AND	NAND	OR	NOR	XOR	XNOR
Alg. Expr.	$\overline{A}$	AB	$\overline{AB}$	$A+B$	$\overline{A+B}$	$A \oplus B$	$\overline{A \oplus B}$
Symbol							
Truth Table	A X: 0 1; 1 0	B A X: 0 0 0; 0 1 0; 1 0 0; 1 1 1	B A X: 0 0 1; 0 1 1; 1 0 1; 1 1 0	B A X: 0 0 0; 0 1 1; 1 0 1; 1 1 1	B A X: 0 0 1; 0 1 0; 1 0 0; 1 1 0	B A X: 0 0 0; 0 1 1; 1 0 1; 1 1 0	B A X: 0 0 1; 0 1 0; 1 0 0; 1 1 1

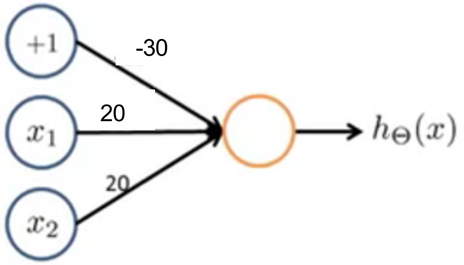
EXAMPLE: LOGICAL FUNCTIONS

Single neurons can describe some logical functions:



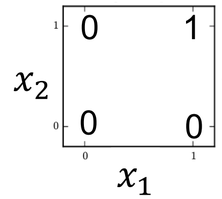
Show that $h_\theta(x)$ models the « NOT » Logical Function

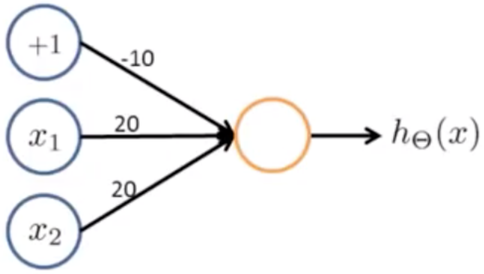
x_1	$h_\Theta(x)$
0	
1	



Show that $h_\theta(x)$ models the « AND » Logical Function

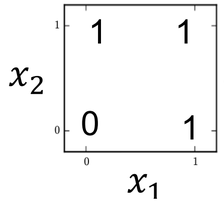
x_1	x_2	$h_\Theta(x)$
0	0	
0	1	
1	0	
1	1	





Show that $h_\theta(x)$ models the « OR » Logical Functior

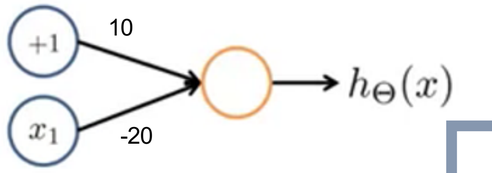
x_1	x_2	$h_\Theta(x)$
0	0	
0	1	
1	0	
1	1	



using a sigmoid activation

EXAMPLE: LOGICAL FUNCTIONS

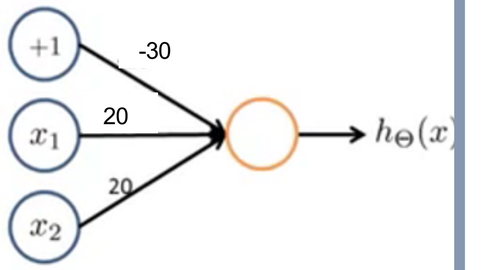
Single neurons can describe some logical functions:



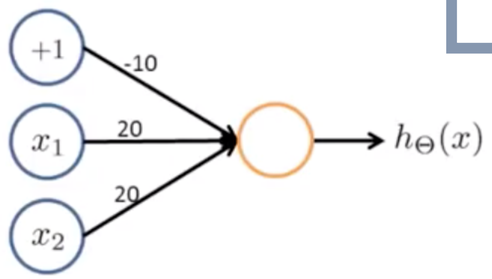
Show that $h_\theta(x)$ models the « NOT » Logical Function

x_1	$h_\Theta(x)$
0	

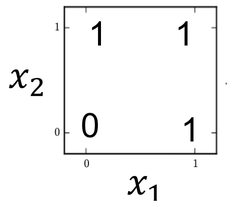
What do I mean by **single neuron**?



x_2	$h_\Theta(x)$
0	
1	
0	
1	



Show that $h_\theta(x)$ models the « OR » Logical Functior

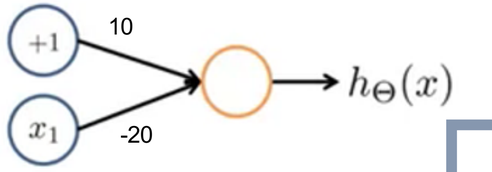


x_1	x_2	$h_\Theta(x)$
0	0	
0	1	
1	0	
1	1	

using a sigmoid activation

EXAMPLE: LOGICAL FUNCTIONS

Single neurons can describe some logical functions:

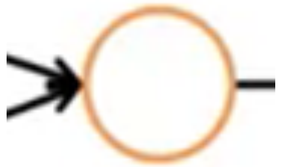


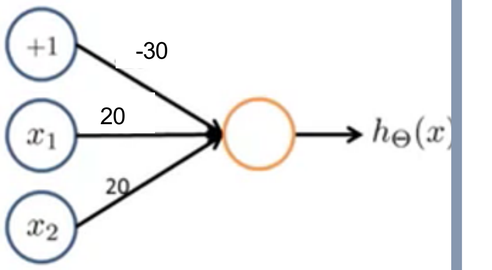
Show that $h_\theta(x)$ models the « NOT » Logical Function

x_1	$h_\Theta(x)$
0	

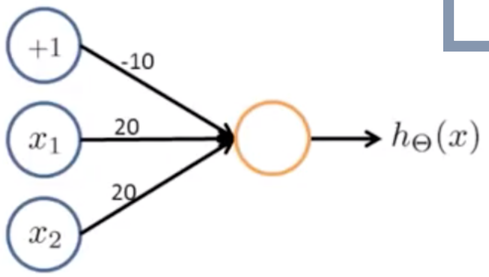
What do I mean by **single neuron**?

this:

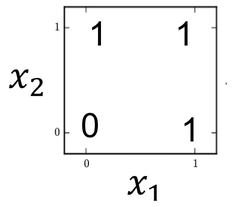




x_2	$h_\Theta(x)$
0	
1	
0	
1	



Show that $h_\theta(x)$ models the « OR » Logical Functior

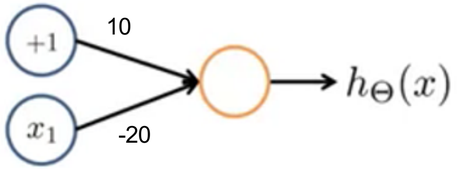


x_1	x_2	$h_\Theta(x)$
0	0	
0	1	
1	0	
1	1	

using a sigmoid activation

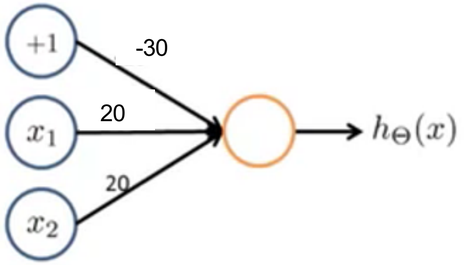
EXAMPLE: LOGICAL FUNCTIONS

Single neurons can describe some logical functions:

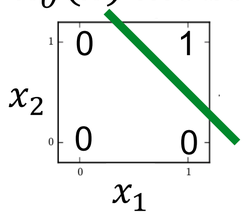


Show that $h_\theta(x)$ models the « NOT » Logical Function

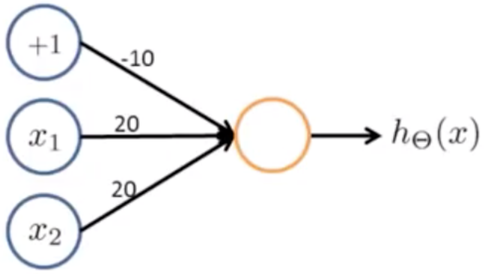
x_1	$h_\Theta(x)$	
0	0.9999546	→ 1
1	4.539787e-05	→ 0



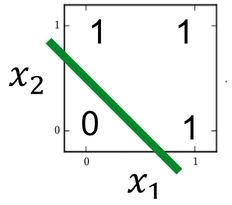
Show that $h_\theta(x)$ models the « AND » Logical Function



x_1	x_2	$h_\Theta(x)$	
0	0	9.3576e-14	→ 0
0	1	4.539787e-05	→ 0
1	0	4.539787e-05	→ 0
1	1	0.9999546	→ 1



Show that $h_\theta(x)$ models the « OR » Logical Functior

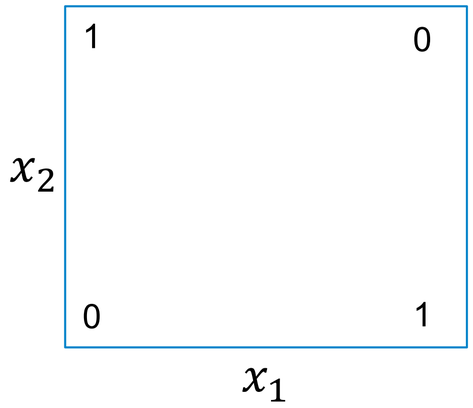


x_1	x_2	$h_\Theta(x)$
0	0	
0	1	
1	0	
1	1	

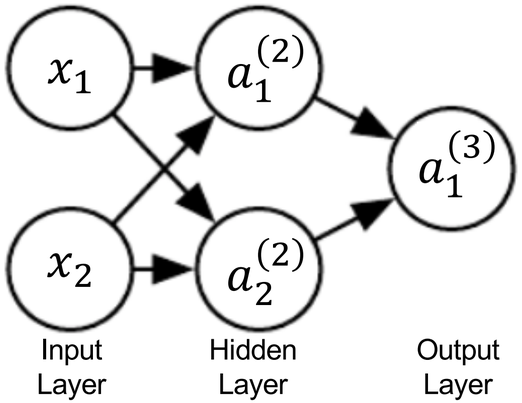
using a sigmoid activation

Modelling the XOR function with a neural network:

The ”Exclusive OR” or XOR function: when one of the input x_1, x_2 values is equal to 1 and the other to 0, XOR returns the value 1, otherwise 0.



This cannot be approximated by a single neuron, as we saw that the Logistic Regression Decision Boundary was a line. We need a hidden layer! Let us try the simplest possible neural network:



example from Goodfellow et al, 2016

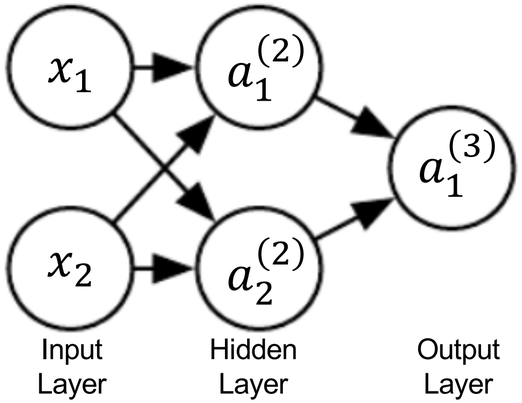
1EDSML
ASD1E2
CE1MGD
SEMS12
EGDCD1

Modelling the XOR function with a neural network:

Assume that we have a Bias term and the parameters of the hidden layer are:

$$\theta^{(1)} = \begin{pmatrix} 0 & 1 & 1 \\ -1 & 1 & 1 \end{pmatrix}$$

The activation function is the ReLU Function.



And that the parameters of the output layer are:

$$\theta^{(2)} = \begin{pmatrix} -\frac{1}{2} & \frac{5}{8} & -1 \end{pmatrix}$$

The activation function is the Logistic Function.

Exercise: Calculate the network output for each of the four input points $\begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$***:***

$$\begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \end{pmatrix}, \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

example from Goodfellow et al, 2016

EXAMPLE: LOGICAL FUNCTIONS

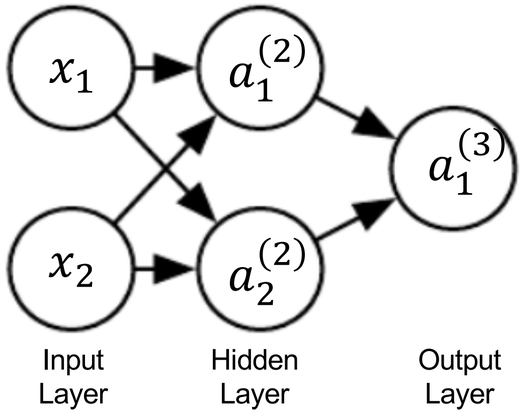
Modelling the **XOR** function with a neural network:

$$\text{We have } \begin{pmatrix} a_1^{(2)} \\ a_2^{(2)} \end{pmatrix} = ReLU\left(\theta^{(1)} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \end{pmatrix} \right)$$

$$\text{with } \quad \theta^{(1)} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 0 & 1 & 1 \\ -1 & 1 & 1 \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 0 + x_1 + x_2 \\ -1 + x_1 + x_2 \end{pmatrix}$$

$$\text{And we have } a_1^{(3)} = \sigma\left(\theta^{(2)} \begin{pmatrix} a_0^{(2)} \\ a_1^{(2)} \\ a_2^{(2)} \end{pmatrix} \right)$$

$$\text{with } \theta^{(2)} \begin{pmatrix} a_0^{(2)} \\ a_1^{(2)} \\ a_2^{(2)} \end{pmatrix} = \begin{pmatrix} -\frac{1}{2} & \frac{5}{8} & -1 \end{pmatrix} \begin{pmatrix} a_0^{(2)} \\ a_1^{(2)} \\ a_2^{(2)} \end{pmatrix} = \begin{pmatrix} -\frac{1}{2} + \frac{5}{8} a_1^{(2)} - a_2^{(2)} \end{pmatrix}$$



example from Goodfellow et al, 2016

EXAMPLE: LOGICAL FUNCTIONS

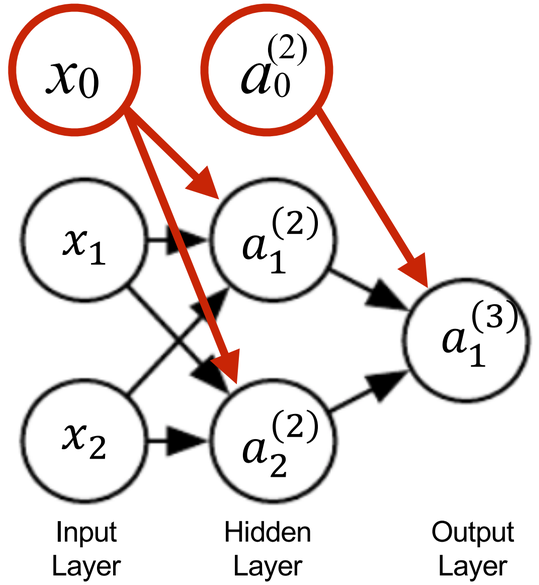
Modelling the XOR function with a neural network:

We have $\begin{pmatrix} a_1^{(2)} \\ a_2^{(2)} \end{pmatrix} = ReLU\left(\theta^{(1)} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \end{pmatrix}\right)$

with $\theta^{(1)} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 0 & 1 & 1 \\ -1 & 1 & 1 \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 0 + x_1 + x_2 \\ -1 + x_1 + x_2 \end{pmatrix}$

And we have $a_1^{(3)} = \sigma\left(\theta^{(2)} \begin{pmatrix} a_0^{(2)} \\ a_1^{(2)} \\ a_2^{(2)} \end{pmatrix}\right)$

with $\theta^{(2)} \begin{pmatrix} a_0^{(2)} \\ a_1^{(2)} \\ a_2^{(2)} \end{pmatrix} = \begin{pmatrix} -\frac{1}{2} & \frac{5}{8} & -1 \end{pmatrix} \begin{pmatrix} a_0^{(2)} \\ a_1^{(2)} \\ a_2^{(2)} \end{pmatrix} = \begin{pmatrix} -\frac{1}{2} + \frac{5}{8} a_1^{(2)} - a_2^{(2)} \end{pmatrix}$



example from Goodfellow et al, 2016

Modelling the XOR function with a neural network:

It is easy to see that :

$$\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix} \Rightarrow \begin{pmatrix} a_1^{(2)} \\ a_2^{(2)} \end{pmatrix} = ReLU \begin{pmatrix} 0 \\ -1 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

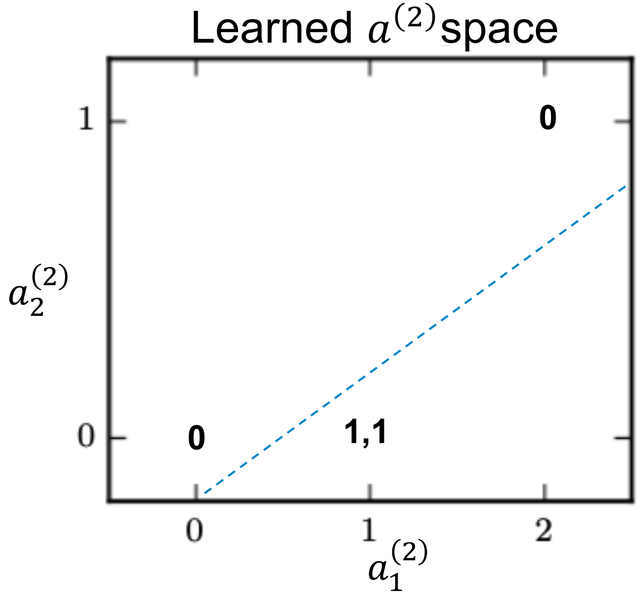
$$\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \end{pmatrix} \Rightarrow \begin{pmatrix} a_1^{(2)} \\ a_2^{(2)} \end{pmatrix} = ReLU \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

$$\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \Rightarrow \begin{pmatrix} a_1^{(2)} \\ a_2^{(2)} \end{pmatrix} = ReLU \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

$$\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \end{pmatrix} \Rightarrow \begin{pmatrix} a_1^{(2)} \\ a_2^{(2)} \end{pmatrix} = ReLU \begin{pmatrix} 2 \\ 1 \end{pmatrix} = \begin{pmatrix} 2 \\ 1 \end{pmatrix}$$

$\begin{pmatrix} 0 + x_1 + x_2 \\ -1 + x_1 + x_2 \end{pmatrix}$

The hidden layer changes the position of the four points such that they can now be linearly separated by the output layer!



..and the output layer $a_1^{(3)} = \sigma\left(\frac{5}{8}a_1^{(2)} - a_2^{(2)} - \frac{1}{2}\right)$ *gives XOR function for the four points!*

example from Goodfellow et al, 2016

Example: Logical Functions

Modelling the **XOR** function with a neural network:

Did we train a network here? Why?

EXAMPLE: LOGICAL FUNCTIONS

Modelling the **XOR** function with a neural network:

Did we train a network here? Why?

No, we just demonstrated that this trained network works as an XOR function

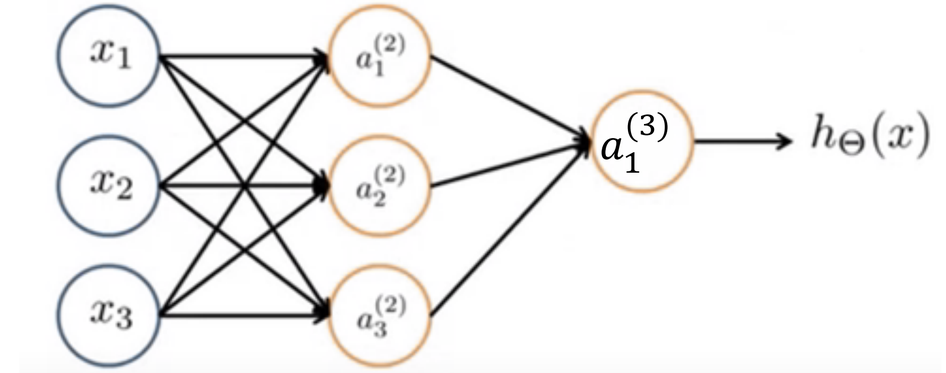
Neural Networks and Logistic Regression

- There are some complex mathematical functions (such as the “Exclusive OR”) that **Logistic Regression cannot represent**.

- Linear Logistic Regression represents a single neuron and is the shallowest form of neural network. The input features are given and the **Decision Boundary is linear**. It is like the output layer of a neural network. But you may need to **“massage”** the input features to help this linear Decision Boundary separate the data properly in the output layer. The **Neural Network**, by transforming the initial variable at each layer, does this “massaging”.

- The “**Universal Approximation Theorem**” (Hornik et al, 1989) states that a neural network with at least one hidden layer can approximate arbitrarily well any function from any finite dimensional space to another, provided that the network is given enough neurons.

Notations and definitions:



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

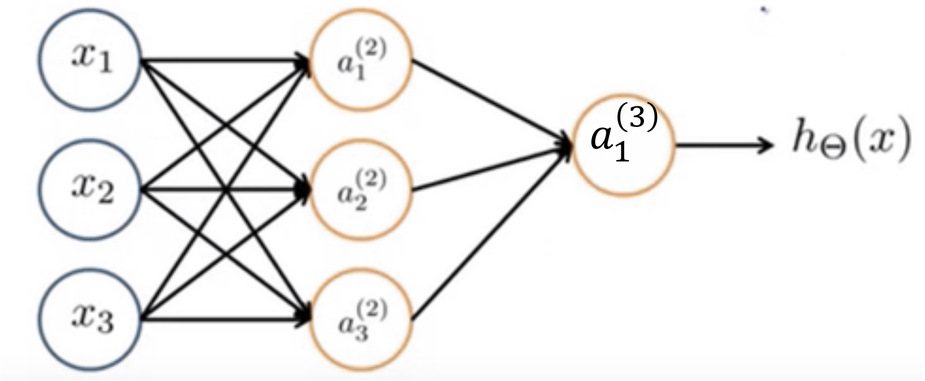
$\theta^{(l)}$ = matrix of weights controlling mapping from layer l to layer $l+1$

$\theta_{jk}^{(l)}$ = weight from neuron k in layer (l) to neuron j in layer $(l+1)$

$$a_1^{(2)} = g\left(\theta_{10}^{(1)}x_0 + \theta_{11}^{(1)}x_1 + \theta_{12}^{(1)}x_2 + \theta_{13}^{(1)}x_3\right)$$
$$a_2^{(2)} = g\left(\theta_{20}^{(1)}x_0 + \theta_{21}^{(1)}x_1 + \theta_{22}^{(1)}x_2 + \theta_{23}^{(1)}x_3\right)$$
$$a_3^{(2)} = g\left(\theta_{30}^{(1)}x_0 + \theta_{31}^{(1)}x_1 + \theta_{32}^{(1)}x_2 + \theta_{33}^{(1)}x_3\right)$$

$$h_\theta(x) = a_1^{(3)} = g\left(\theta_{10}^{(2)}a_0^{(2)} + \theta_{11}^{(2)}a_1^{(2)} + \theta_{12}^{(2)}a_2^{(2)} + \theta_{13}^{(2)}a_3^{(2)}\right)$$

Number of parameters in a network:



Number of parameters (or weights) $\theta_{jk}^{(l)}$, assuming bias term for each layer?

On this example:
(Number of nodes in input layer + 1) × Number of nodes in hidden layer +
(Number of nodes in hidden layer + 1) × Number of nodes in output layer

= (3+1) x 3 + (3 +1) x 1 = **16**

Example with two hidden layers

Forward propagation

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

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

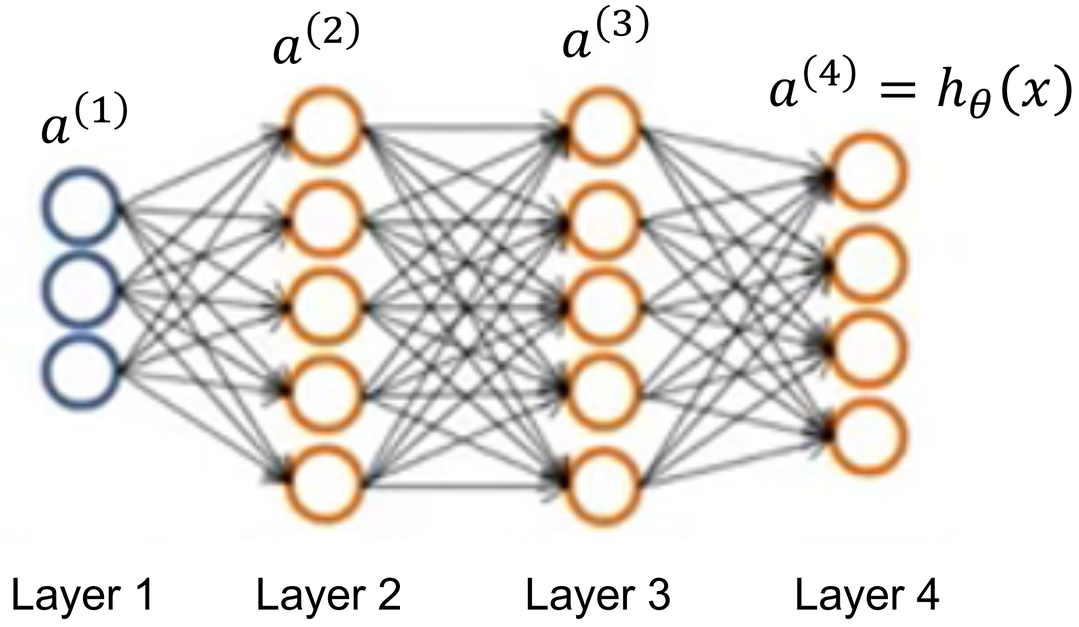
$$a^{(2)} = g\left(z^{(2)}\right)$$

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

$$a^{(3)} = g\left(z^{(3)}\right)$$

$$z^{(4)} = \theta^{(3)} a^{(3)}$$

$$a^{(4)} = g\left(z^{(4)}\right) = h_\theta(x)$$



Number of parameters (assuming bias terms)?

1EDSML
ASD1E2
CE1MGD
SEMS12
EGDCD1

Example with two hidden layers:

Forward propagation

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

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

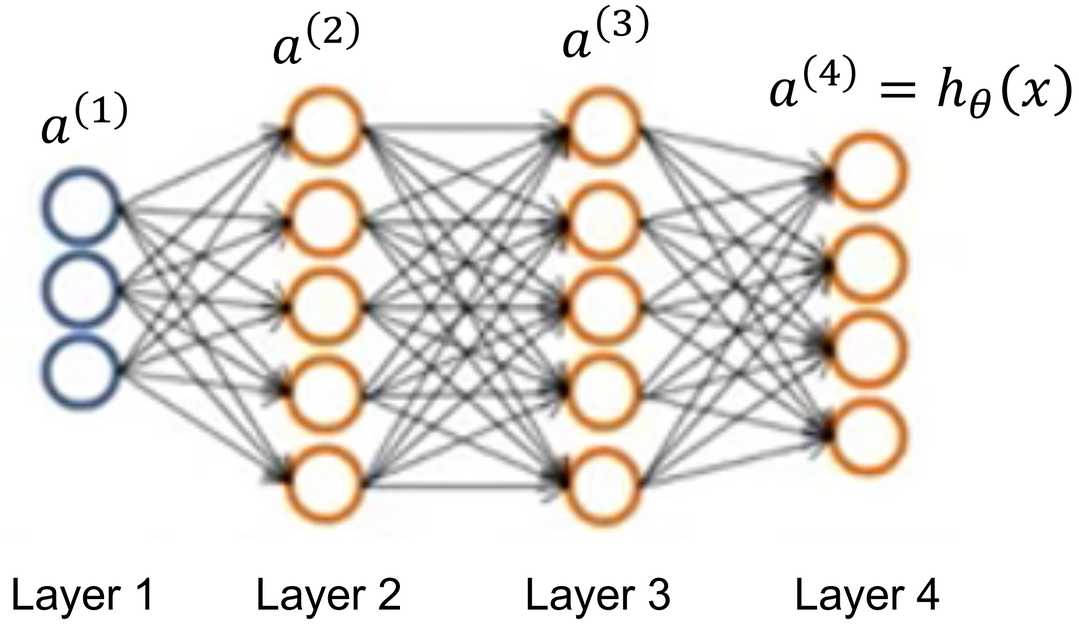
$$a^{(2)} = g\left(z^{(2)}\right)$$

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

$$a^{(3)} = g\left(z^{(3)}\right)$$

$$z^{(4)} = \theta^{(3)} a^{(3)}$$

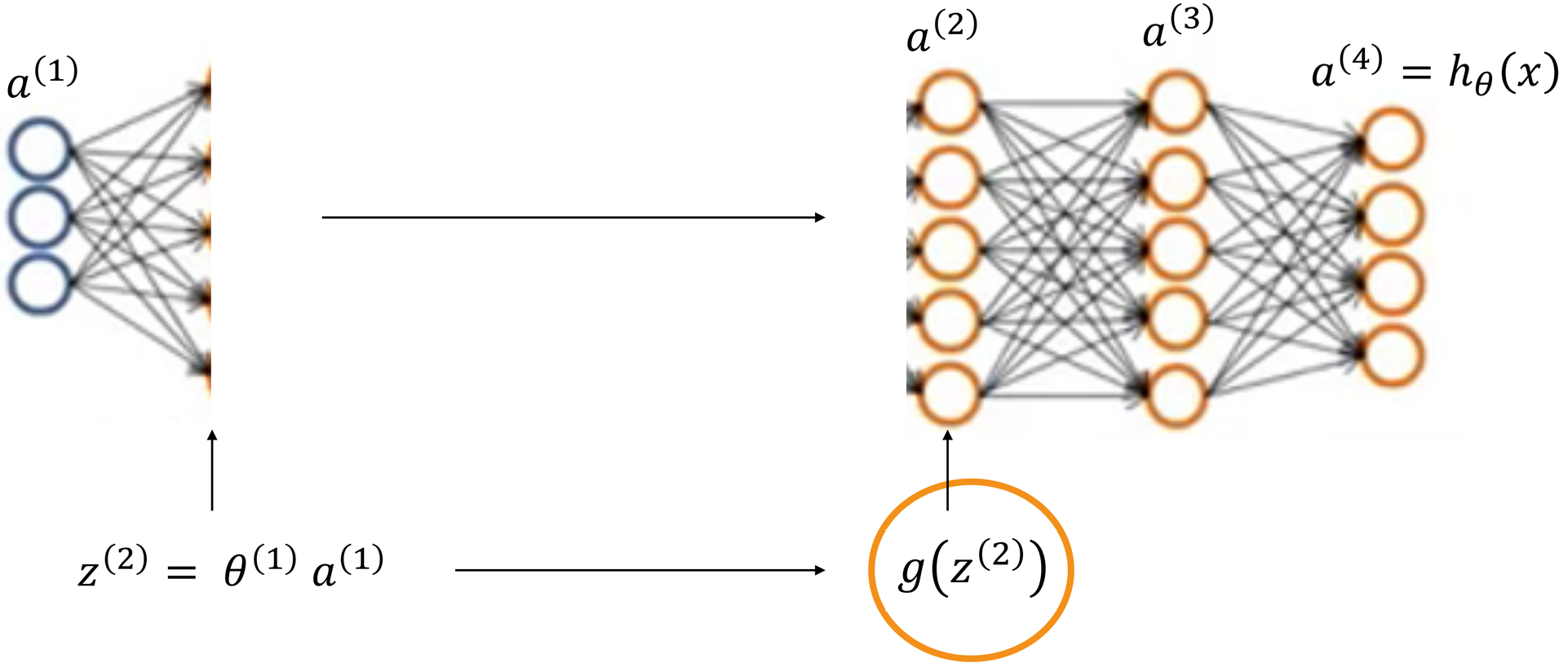
$$a^{(4)} = g\left(z^{(4)}\right) = h_\theta(x)$$



Number of parameters (assuming bias terms)?

(3+1)x5 + (5+1)x5 + (5+1)x4 = **74**

Example with two hidden layers:

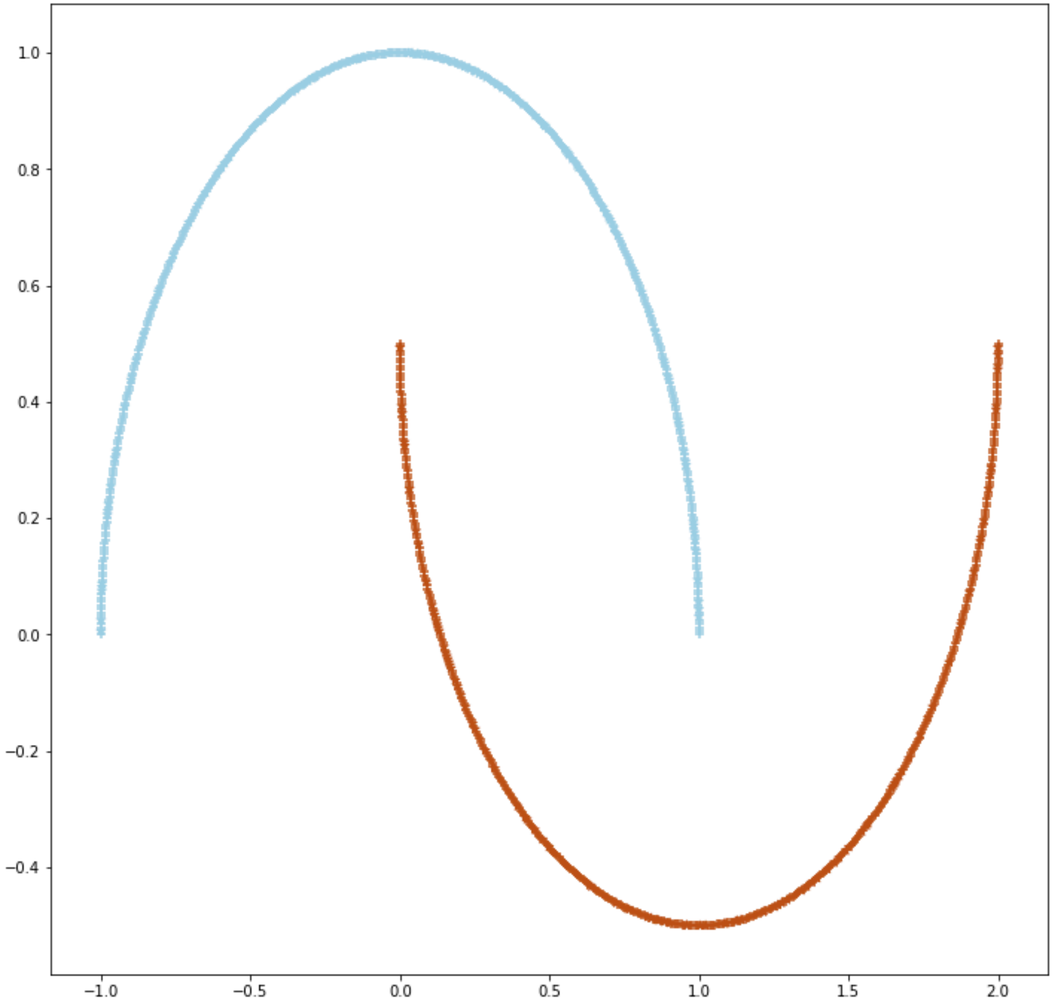


$$z_1^{(2)} = \theta_{10}^{(1)} x_0 + \theta_{11}^{(1)} x_1 + \theta_{12}^{(1)} x_2 + \theta_{13}^{(1)} x_3$$
$$z_2^{(2)} = \theta_{20}^{(1)} x_0 + \theta_{21}^{(1)} x_1 + \theta_{22}^{(1)} x_2 + \theta_{23}^{(1)} x_3$$
$$z_3^{(2)} = \theta_{30}^{(1)} x_0 + \theta_{31}^{(1)} x_1 + \theta_{32}^{(1)} x_2 + \theta_{33}^{(1)} x_3$$

$$a_1^{(2)} = g\left(\theta_{10}^{(1)} x_0 + \theta_{11}^{(1)} x_1 + \theta_{12}^{(1)} x_2 + \theta_{13}^{(1)} x_3\right)$$
$$a_2^{(2)} = g\left(\theta_{20}^{(1)} x_0 + \theta_{21}^{(1)} x_1 + \theta_{22}^{(1)} x_2 + \theta_{23}^{(1)} x_3\right)$$
$$a_3^{(2)} = g\left(\theta_{30}^{(1)} x_0 + \theta_{31}^{(1)} x_1 + \theta_{32}^{(1)} x_2 + \theta_{33}^{(1)} x_3\right)$$

Previous Non-Linear Example

Neural networks on non-linear logistic regression example:



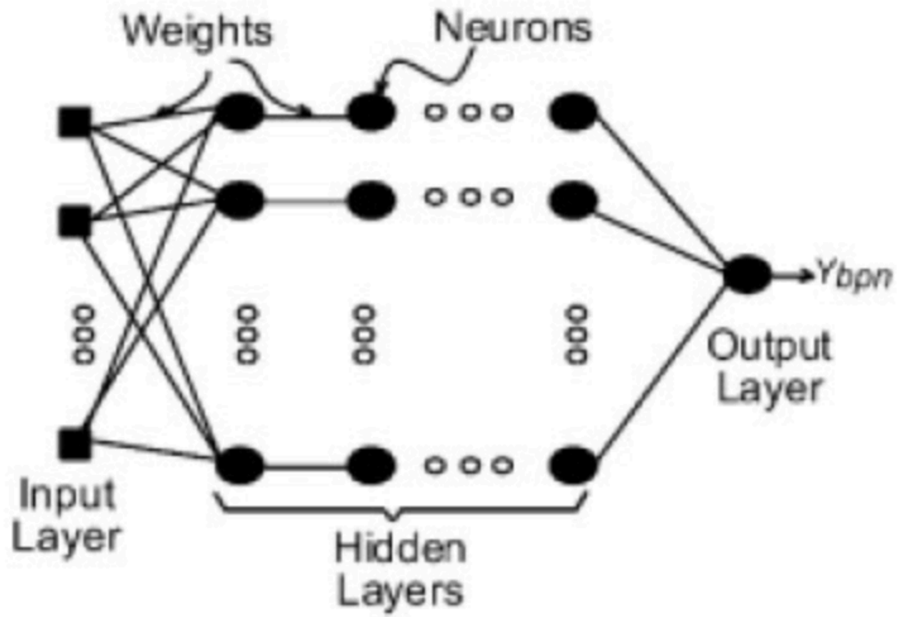
The m =1000 data points $\left(x_1^i, x_2^i\right)_{i=1,1000}$ are in the plane.

A group of data are one color, the other group another color.

Question: predict the color at each location of the plane.

Previous Non-Linear Example

Define a simple neural network with:



n=2 n=25 n=25 n=25 n=1

Neural Network Architecture

Input Layer: Two Neurons (x_1 and x_2)

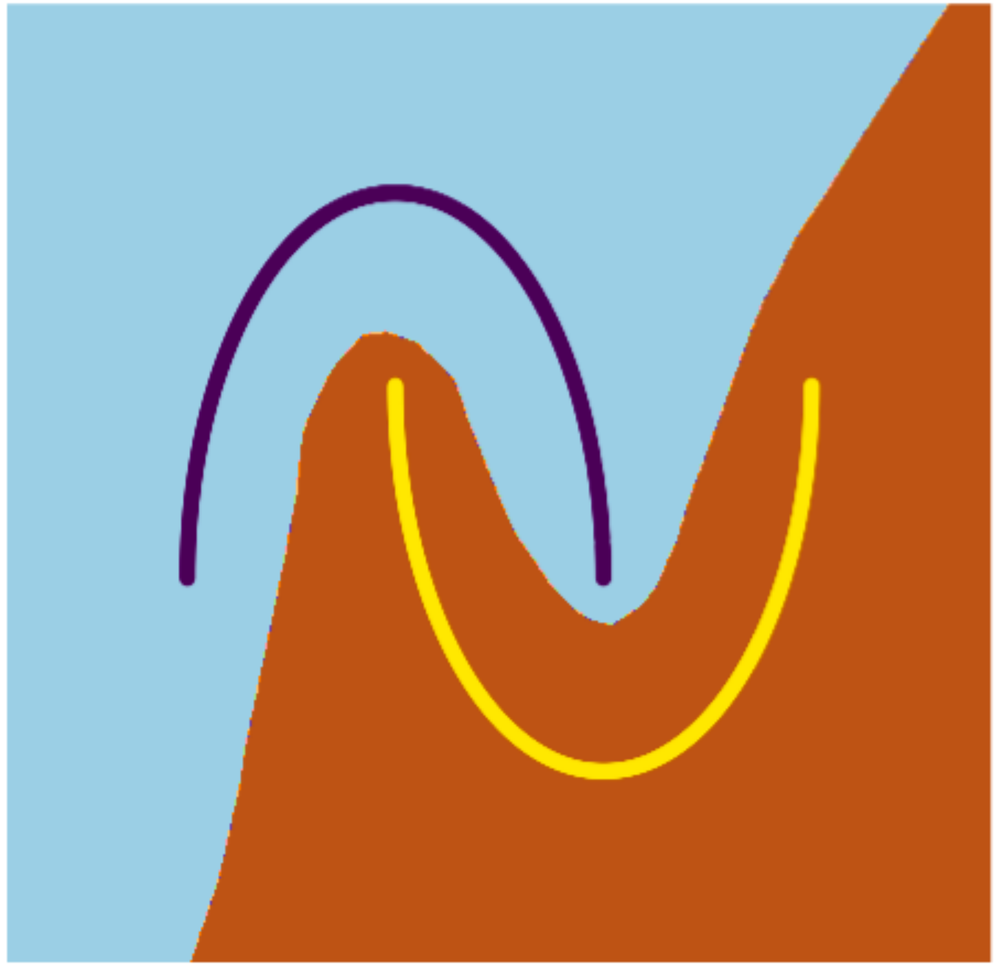
Three Hidden Layers Each 25 Neurons

Output Layer: One neuron: Probability of Being Blue

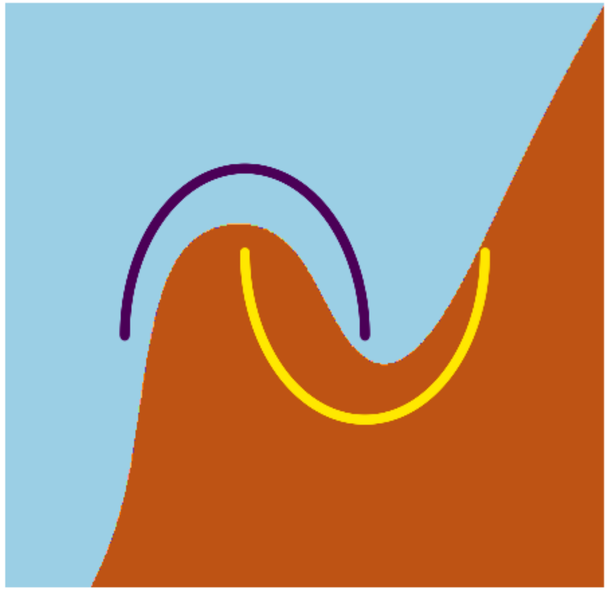
Total Number of Weights (if bias terms):
(2+1)x25 + (25+1)x25 + (25+1)x25 + (25+1)x1 = 1401

Previous Non-Linear Example

This network results:



Previous non-linear logistic regression result:



TOY EXAMPLES

https://playground.tensorflow.org/

https://playground.tensorflow.org/

Four classification examples:

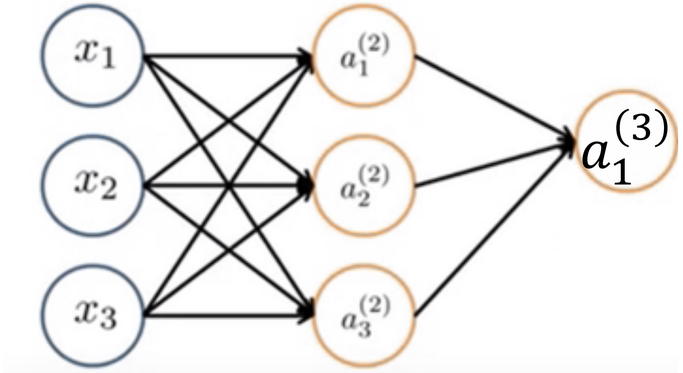
- The third one can be addressed by logistic regression, that is with no hidden layer at all and just one output neuron.

- The second one cannot be addressed by logistic regression, because it is clearly non linear and it is similar to the XOR function. We need at least one hidden layer.

- The first one is the circle. With just one hidden layer and 4 neurons in it we reproduce the circle. But of course if we use x_1^2 and x_2^2 as input we do not even need a hidden layer, meaning that logistic regression does the job!!

Cost Function for Binary Classification

The class of each data point i is: $y_i = 0 \text{ or } 1$



$h_\theta(x) = a_1^{(3)}$ (or $a_{1i}^{(3)}$ for each data point i)

The cost function is the cross-entropy already used in the Logistic Regression case:

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}\left[y_i \log a_{1i}^{(3)} + (1-y_i)\log\left(1-a_{1i}^{(3)}\right)\right]$$

We will see in the third week that cross-entropy is equal to (minus) the log-likelihood of a Bernouilli distribution.

Multi-Class Classification

But what do we do when we have more than 1 class?

One VS rest approach with 3 classes:

- Train the 3 Neural Networks associated with the three following binary problems:
  - Merge classes 2 and 3 as class “zero” and calculate parameters θ_1 of function:
$$h^1_{\theta_1}(x) = P(y = 1|x, \theta_1)$$
  - Merge classes 1 and 3 as class “zero” and calculate parameters θ_2 of function:
$$h^2_{\theta_2}(x) = P(y = 2|x, \theta_2)$$
  - Merge classes 1 and 2 as class “zero” and calculate parameters θ_3 of function:
$$h^3_{\theta_3}(x) = P(y = 3|x, \theta_3)$$

- For each new unlabeled point x, calculate the three probabilities above and pick as the predicted class the one corresponding to the highest of the three probabilities.

! θ_i represent independent parameter of different networks (not layer indices)

1EDSML
AS01E2
CE1MG0
SEMS12
EG0C01

Multi-class classification using **Softmax** function:

Forward propagation

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

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

$$a^{(2)} = g\left(z^{(2)}\right)$$

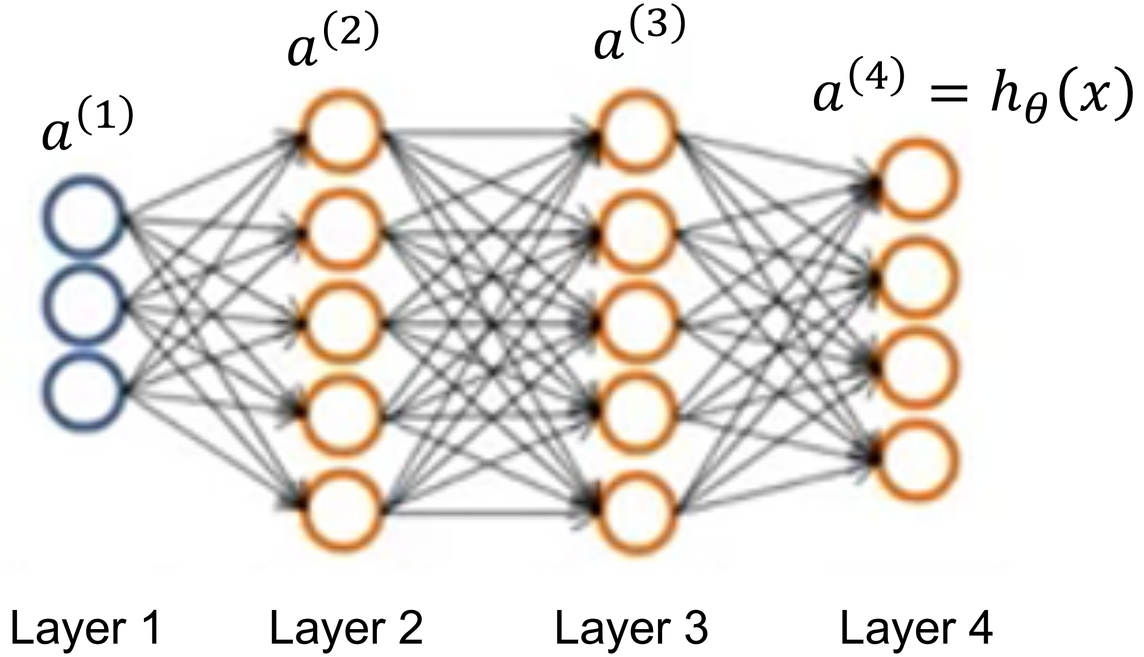
$$z^{(3)} = \theta^{(2)} a^{(2)}$$

$$a^{(3)} = g\left(z^{(3)}\right)$$

$$z^{(4)} = \theta^{(3)} a^{(3)}$$

$$a^{(4)} = g\left(z^{(4)}\right) = h_\theta(x)$$

Softmax



Multi-class classification using Softmax function:

Suppose the output of the last layer of the neural network is $\begin{pmatrix} z_1 \\ z_2 \\ \cdot \\ z_i \\ \cdot \\ z_{n-1} \\ z_n \end{pmatrix}$ of size n

The **Softmax** function transforms it into an output probability vector:

$$\begin{pmatrix} p_1 \\ p_2 \\ p_3 \\ \cdot \\ p_i \\ \cdot \\ p_{n-1} \\ p_n \end{pmatrix} = \begin{pmatrix} \frac{e^{z_1}}{\sum_{j=1}^{n} e^{z_j}} \\ \frac{e^{z_2}}{\sum_{j=1}^{n} e^{z_j}} \\ \cdot \\ \frac{e^{z_i}}{\sum_{j=1}^{n} e^{z_j}} \\ \cdot \\ \frac{e^{z_{n-1}}}{\sum_{j=1}^{n} e^{z_j}} \\ \frac{e^{z_n}}{\sum_{j=1}^{n} e^{z_j}} \end{pmatrix}$$

Example using the **Softmax** function:

$$\begin{pmatrix} z_1 \\ z_2 \\ z_3 \\ z_4 \end{pmatrix} = \begin{pmatrix} 5 \\ -1 \\ 2 \\ -4 \end{pmatrix}$$

$$\begin{pmatrix} p_1 \\ p_2 \\ p_3 \\ p_4 \end{pmatrix} = \begin{pmatrix} \frac{148.41}{156.19} \\ \frac{0.37}{156.19} \\ \frac{7.39}{156.19} \\ \frac{0.02}{156.19} \end{pmatrix} = \begin{pmatrix} 0.950 \\ 0.003 \\ 0.047 \\ 0.000 \end{pmatrix}$$

SoftMax Function does two things:

- Transform the vector into probabilities that are between 0 and 1 and add up to 1
- Transform the largest value into a value close to 1 and the lowest into one close to 0

Which cross-entropy function for training with Softmax?

Take the example of predicting whether the colour at one pixel of an image is brown, yellow or blue. We have three classes.

First approach
Code each colour as a number: 1 for brown, 2 for yellow, 3 for blue.

Softmax Function

Which cross-entropy function for training with **Softmax**?
We use one-hot encoding for defining each class membership:

Take the example of predicting whether the colour at one pixel of an image is brown, yellow or blue. We have three classes.

First approach
Code each colour as a number: 1 for brown, 2 for yellow, 3 for blue.
But this may create an artificial distance between brown and blue larger than between brown and yellow or yellow and blue!

Second approach: one-hot encoding
Represent the class of each pixel by a vector c of dimension equal to the number of classes:

If pixel is brown: $c = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}$, *If pixel is yellow:* $c = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}$, *If pixel is blue:* $c = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$.

1EDSML
ASO1E2
CE1MGO
SEMS12
EGOCO1

Cost function in Softmax multi-class classification:

For one hot-encoded yellow data point i of the Training Set: $y^{(i)} = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}$

If $\begin{pmatrix} p_1 \\ p_2 \\ p_3 \end{pmatrix}$ is the probability vector calculated by Softmax for this point , the cross-entropy is defined as:

$$J(\theta) = -0 \times log p_1 - 1 \times log p_2 - 0 \times log p_3 = -log p_2$$

So, in the multi-class Softmax case, the *cross-entropy is minus the log of the output probability associated with the actual class of the data point*.
Logically, if $p_2 = 1, J(\theta) = 0,$ and if $p_2 = 0, J(\theta) = +\infty$

For the binary case, it is easy to check that we get the formula already seen on Monday.

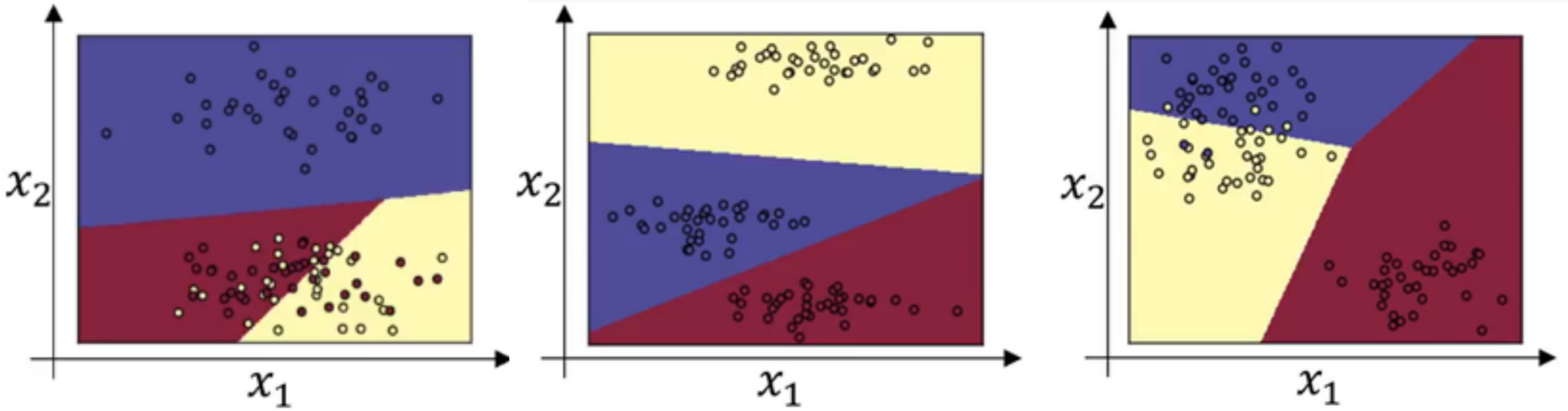
For m data points, the above is averaged for all the data points.

1EDSML
ASO1E2
CE1MGO
SEMS12
EGOCO1

Example: classify 2D data in 3 classes

$$(x_1, x_2) \Rightarrow \begin{matrix} z \\ \begin{pmatrix} z_1 \\ z_2 \\ z_3 \end{pmatrix} \end{matrix} \Rightarrow \begin{matrix} p \\ \begin{pmatrix} p_1 \\ p_2 \\ p_3 \end{pmatrix} \end{matrix}$$

$$\begin{pmatrix} p_1 \\ p_2 \\ p_3 \end{pmatrix} = Softmax \begin{pmatrix} z_1 \\ z_2 \\ z_3 \end{pmatrix} = Softmax \begin{pmatrix} \theta_{10}^{(1)} + \theta_{11}^{(1)} x_1 + \theta_{12}^{(1)} x_2 \\ \theta_{20}^{(1)} + \theta_{21}^{(1)} x_1 + \theta_{22}^{(1)} x_2 \\ \theta_{30}^{(1)} + \theta_{31}^{(1)} x_1 + \theta_{32}^{(1)} x_2 \end{pmatrix}$$



This neural network is quite simple and equivalent to a Logistic regression, hence the straight Decision Boundaries.

1EDSML
AS01E2
CE1MG0
SEMS12
EG0C01

Predicting a test sample class after training:

Suppose the output of the last layer of the neural network is $\begin{pmatrix} z_1 \\ z_2 \\ \cdot \\ z_i \\ \cdot \\ z_{n-1} \\ z_n \end{pmatrix}$ of size n

The **Softmax** function transforms it into an output probability vector:

$$\begin{pmatrix} p_1 \\ p_2 \\ p_3 \\ \cdot \\ p_i \\ \cdot \\ p_{n-1} \\ p_n \end{pmatrix} = \begin{pmatrix} \frac{e^{z_1}}{\sum_{j=1}^{n} e^{z_j}} \\ \frac{e^{z_2}}{\sum_{j=1}^{n} e^{z_j}} \\ \cdot \\ \frac{e^{z_i}}{\sum_{j=1}^{n} e^{z_j}} \\ \cdot \\ \frac{e^{z_{n-1}}}{\sum_{j=1}^{n} e^{z_j}} \\ \frac{e^{z_n}}{\sum_{j=1}^{n} e^{z_j}} \end{pmatrix}$$

The class with the highest Softmax probability is selected

Softmax Function

A good resource with an explanation by Andrew Ng:

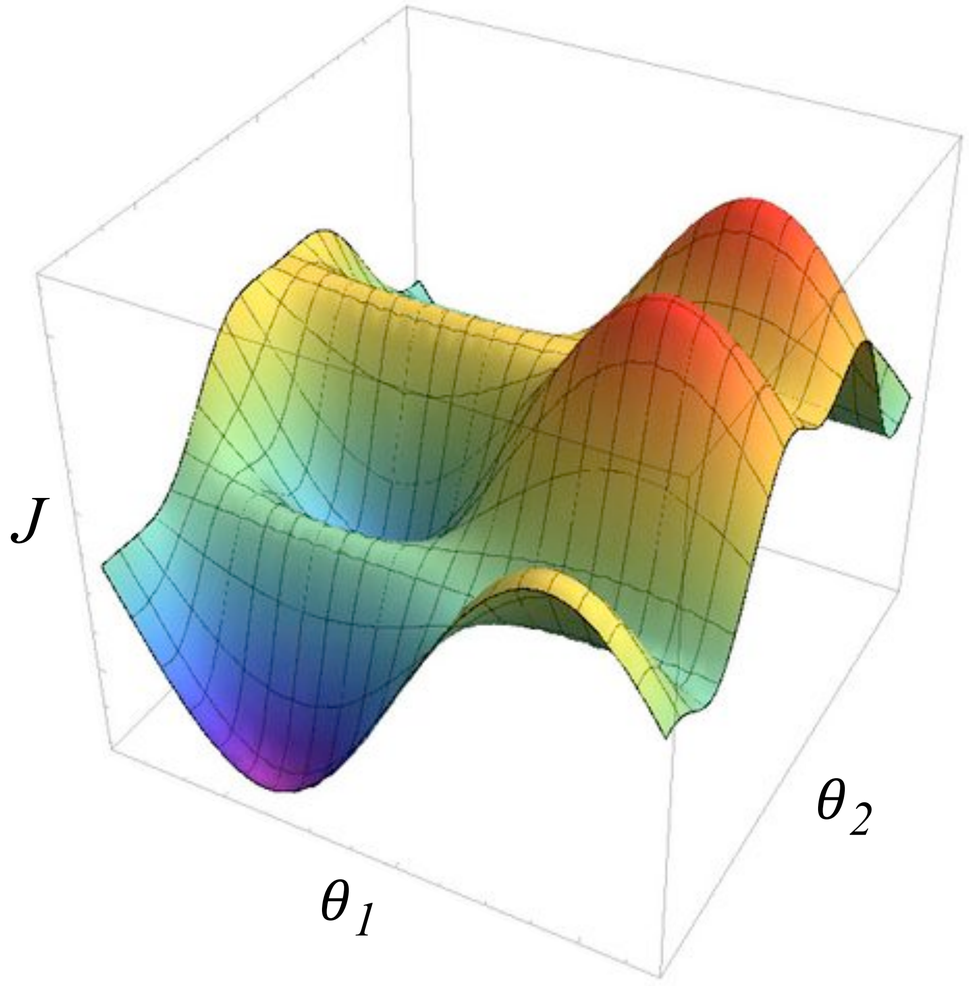
https://www.youtube.com/watch?v=LLux1SW--oM

Feed-Forward Neural Networks

1. From Logistic Regression to Single Neuron Representation
2. Feed-Forward Neural Networks
3. **Back-Propagation**
4. Batch, Stochastic and Mini-Batch Gradient Descent
5. Some basic architectures

Cost Function Minimisation

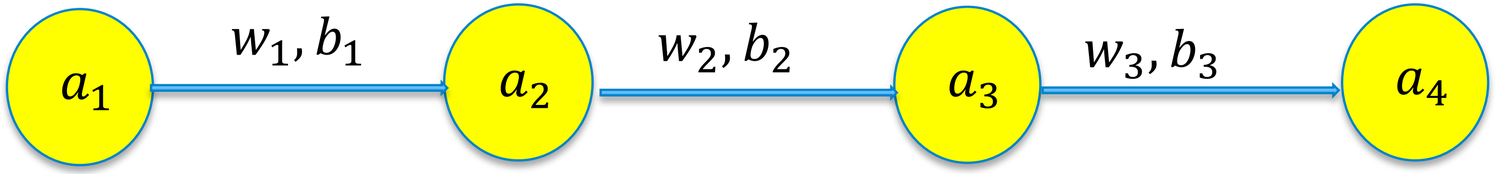
How do we minimise cost functions with neural networks?



1EDSML
AS01E2
CE1MG0
SEMS12
EG0C01

Simple example that illustrates the back-propagation principles:

Let us take a very simple network of $L = 4$ layers with one neuron in each.
We have six parameters, three weights w_i and three biases b_i(we could have used the θ notation as before but the (w_i, b_i)notation is quite common too).



Suppose we have just one sample (x, y) (hence $a_1 = x$ and y is the real number target), and a regression neural network.

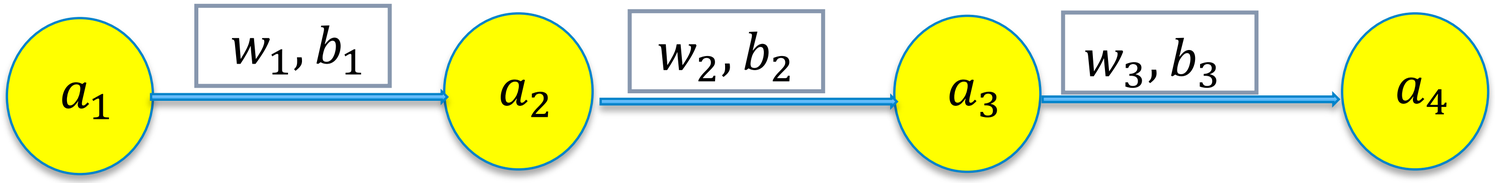
The L_2 cost function is $C = \frac{1}{2}(a_4 - y)^2$.

We have: $a_2 = g(z_2)$ with $z_2 = w_1 a_1 + b_1$
$a_3 = g(z_3)$ with $z_3 = w_2 a_2 + b_2$
$a_4 = g(z_4)$ with $z_4 = w_3 a_3 + b_3$

1EDSML
ASO1E2
CE1MGO
SEMS12
EGOCO1

Simple example that illustrates the back-propagation principles:

Let us take a very simple network of $L = 4$ layers with one neuron in each.
We have six parameters, three weights w_i and three biases b_i(we could have used the θ notation as before but the (w_i, b_i)notation is quite common too!).



Suppose we have just one sample (x, y) (hence $a_1 = x$ and y is the real number target), and a regression neural network.

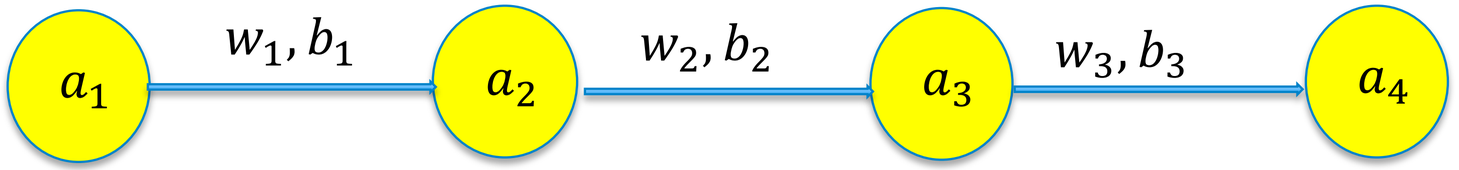
The L_2 cost function is $C = \frac{1}{2}(a_4 - y)^2$.

We have: $a_2 = g(z_2)$ with $z_2 = w_1 a_1 + b_1$
$a_3 = g(z_3)$ with $z_3 = w_2 a_2 + b_2$
$a_4 = g(z_4)$ with $z_4 = w_3 a_3 + b_3$

How to calculate the derivative of the Cost Function C according to each of the six parameters?

1EDSML
ASO1E2
CE1MGO
SEMS12
EGOCO1

Chain rule derivation in back-propagation:



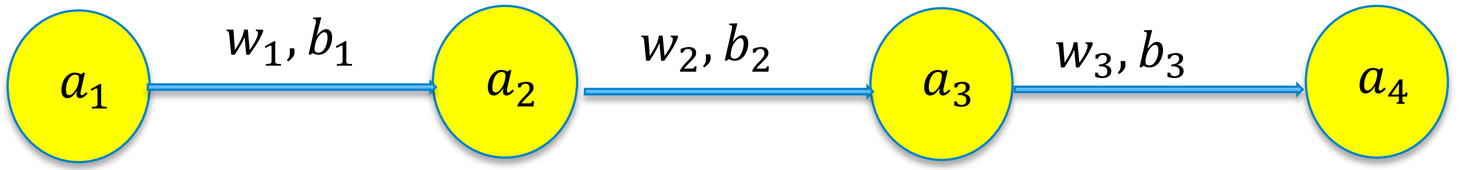
The L_2 cost function is $C = \frac{1}{2}(a_4 - y)^2$.

We have: $a_2 = g(z_2)$ with $z_2 = w_1 a_1 + b_1$
$a_3 = g(z_3)$ with $z_3 = w_2 a_2 + b_2$
$a_4 = g(z_4)$ with $z_4 = w_3 a_3 + b_3$

Using the chain rule, the derivatives according to w_3 and b_3 are:

$\frac{\partial C}{\partial w_3} = \frac{\partial C}{\partial a_4}\frac{\partial a_4}{\partial z_4}\frac{\partial z_4}{\partial w_3} = (a_4 - y)g'(z_4)a_3$	$\frac{\partial C}{\partial b_3} = \frac{\partial C}{\partial a_4}\frac{\partial a_4}{\partial z_4}\frac{\partial z_4}{\partial b_3} = (a_4 - y)g'(z_4)$

Chain rule derivation in back-propagation:



Thanks to the formula: : $$\frac{\partial C}{\partial a_3} = \frac{\partial C}{\partial a_4}\frac{\partial a_4}{\partial z_4}\frac{\partial z_4}{\partial a_3} = (a_4 - y)g'(z_4)w_3$$

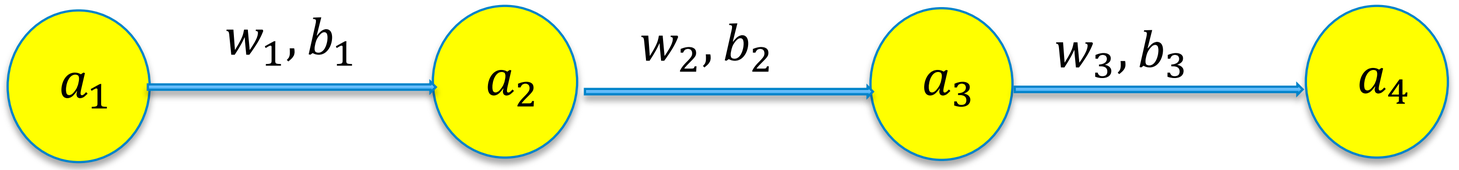
We can keep moving backwards and now obtain the derivatives of C in w_2 and b_2

$$\frac{\partial C}{\partial w_2} = \frac{\partial C}{\partial a_3}\frac{\partial a_3}{\partial z_3}\frac{\partial z_3}{\partial w_2} = (a_4 - y)g'(z_4)w_3 g'(z_3)a_2$$

$$\frac{\partial C}{\partial b_2} = \frac{\partial C}{\partial a_3}\frac{\partial a_3}{\partial z_3}\frac{\partial z_3}{\partial b_2} = (a_4 - y)g'(z_4)w_3\ g'(z_3)$$

1EDSML
AS01E2
CE1MG0
SEMS12
EG0C01

Chain rule derivation in back-propagation:



Thanks to the formula: $\frac{\partial C}{\partial a_2} = \frac{\partial C}{\partial a_3}\frac{\partial a_3}{\partial z_3}\frac{\partial z_3}{\partial a_2} = (a_4 - y)g'(z_4)w_3\ g'(z_3)w_1$

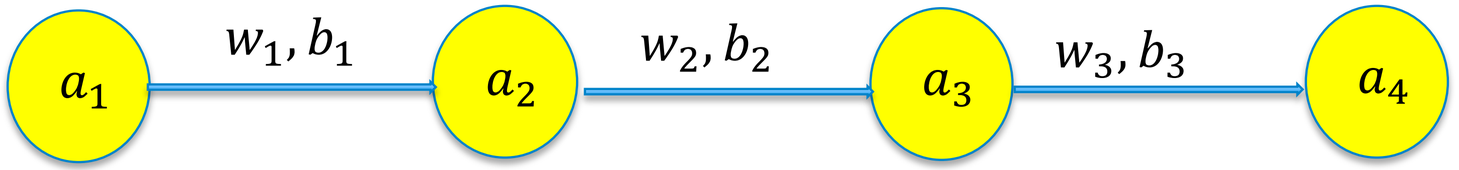
We can keep moving backwards and obtain the derivatives of C in w_1 and b_1

$$\frac{\partial C}{\partial w_1} = \frac{\partial C}{\partial a_2}\frac{\partial a_2}{\partial z_2}\frac{\partial z_2}{\partial w_1} = (a_4 - y)g'(z_4)w_3\ g'(z_3)w_1 g'(z_2)x$$

$$\frac{\partial C}{\partial b_1} = \frac{\partial C}{\partial a_2}\frac{\partial a_2}{\partial z_2}\frac{\partial z_2}{\partial b_1} = (a_4 - y)g'(z_4)w_3\ g'(z_3)w_1\ g'(z_2)$$

1EDSML
ASO1E2
CE1MGO
SEMS12
EGOCO1

Chain rule derivation in back-propagation:



Thanks to the formula: $\frac{\partial C}{\partial a_2} = \frac{\partial C}{\partial a_3}\frac{\partial a_3}{\partial z_3}\frac{\partial z_3}{\partial a_2} = (a_4 - y)g'(z_4)w_3\ g'(z_3)w_1$ X w_2

We can keep moving backwards and obtain the derivatives of C in w_1 and b_1

$$\frac{\partial C}{\partial w_1} = \frac{\partial C}{\partial a_2}\frac{\partial a_2}{\partial z_2}\frac{\partial z_2}{\partial w_1} = (a_4 - y)g'(z_4)w_3\ g'(z_3)w_1 g'(z_2)x$$

$$\frac{\partial C}{\partial b_1} = \frac{\partial C}{\partial a_2}\frac{\partial a_2}{\partial z_2}\frac{\partial z_2}{\partial b_1} = (a_4 - y)g'(z_4)w_3\ g'(z_3)w_1\ g'(z_2)$$

Comments about back-propagation:

- What do the expressions we derived for each of the network parameters do? and how will we use them?

Comments about back-propagation:

- What do the expressions we derived for each of the network parameters do? and how will we use them?

each expression tell us how to update **one network parameter** in order to **improve the model** (reduce the cost function)

remember from first lecture:

$$\theta_b = \theta_a - \alpha \frac{\partial \boldsymbol{J}(\theta_a)}{\partial \boldsymbol{\theta}}$$

w_b, b_b

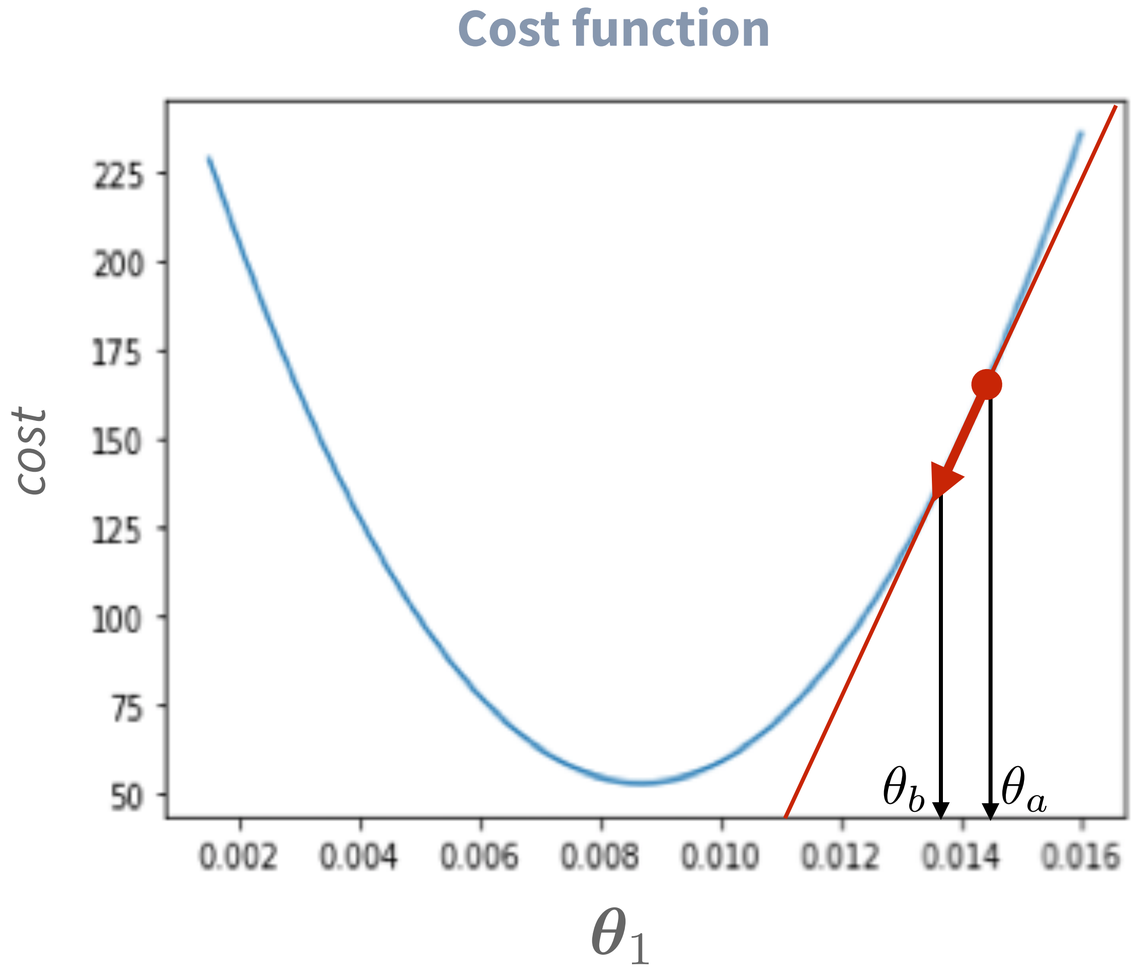
w_a, b_a

Feed-Forward Neural Networks

1. From Logistic Regression to Single Neuron Representation
2. Feed-Forward Neural Networks
3. Back-Propagation
4. **Batch, Stochastic and Mini-Batch Gradient Descent**
5. Some basic architectures

Gradient Descent

Recap of gradient descent:



$$\theta_b = \theta_a \overset{!}{-} \alpha \frac{\partial J(\theta_a)}{\partial \boldsymbol{\theta}}$$

Learning rate

1EDSML
ASO1E2
CE1MGO
SEMS12
EGOCO1

Batch VS stochastic gradient descent:

"Batch" Gradient Descent uses all m training set data $(x_i, y_i)_{i=1,..,m}$ at each gradient descent iteration. When m is very large (say m is in the 100,000's), the number of calculations is thus very large, and this is just to calculate one gradient on all the parameters $\theta_{ij}^{(l)}$.

Instead of summing over all the training set, then calculating the gradient and optimizing, we can iterate only using gradients at individual data points. With *Stochastic Gradient Descent*: every iteration works on one data point at a time.

Stochastic gradient descent

1. Randomly shuffle (reorder) training examples

2. Repeat {
   for $i := 1, \ldots, m$ {
   $\theta_j := \theta_j - \alpha(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)}$
   (for every $j = 0, \ldots, n$)
   }
   }

Gradient Descent Strategies

Mini-batch gradient descent (commonly used):

Rather than the two extremes of *Batch* and *Stochastic Gradient Descent*, one often chooses the intermediate *Mini-Batch Gradient Descent*, where the gradient is calculated on a small subset of data.

Mini-batch gradient descent

Say $b = 10, m = 1000$.

Repeat {

for $i = 1, 11, 21, 31, \ldots, 991$ {

$$\theta_j := \theta_j - \alpha \frac{1}{10} \sum_{k=i}^{i+9} (h_\theta(x^{(k)}) - y^{(k)}) x_j^{(k)}$$

(for every $j = 0, \ldots, n$)

}

}

Mini-batch size often is referred to as **batch size**

Gradient Descent Strategies

Mini-batch gradient descent (commonly used):

Rather than the two extremes of *Batch* and *Stochastic Gradient Descent*, one often chooses the intermediate *Mini-Batch Gradient Descent*, where the gradient is calculated on a small subset of data.

Mini-batch gradient descent

Say $b = 10, m = 1000$.

Repeat {

for $i = 1, 11, 21, 31, \ldots, 991$ {

$$\theta_j := \theta_j - \alpha \frac{1}{10} \sum_{k=i}^{i+9} (h_\theta(x^{(k)}) - y^{(k)}) x_j^{(k)}$$

(for every $j = 0, \ldots, n$)

}

}

Mini-batch size often is referred to as **batch size**

Mini-batch gradient descent is sometimes referred to as stochastic gradient descent, but pure stochastic gradient descent only uses one data sample per iteration.

Gradient Descent Strategies

Three gradient-descent possible strategies:

Batch (also called Full-Batch) Gradient Descent:
Using all m training set data $(x_i, y_i)_{i=1,..,m}$ at each gradient descent Iteration

Stochastic Gradient Descent:
Use one single data (x_i, y_i) at each gradient descent iteration

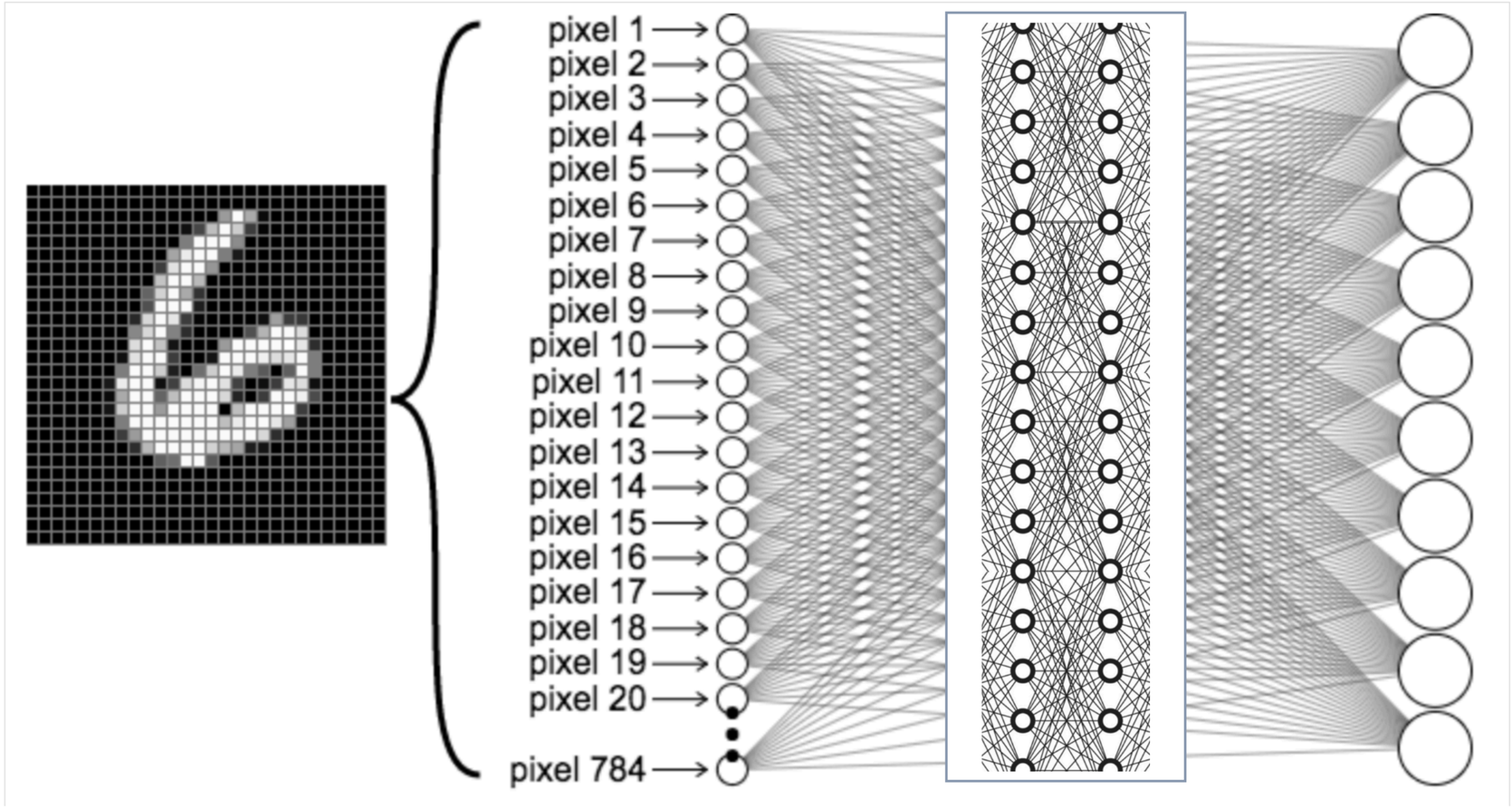
Mini-Batch Gradient Descent:
Use small number (say a few tens or hundreds) of data (x_i, y_i) at each gradient descent iteration

*An **epoch** is a training iteration over the whole training set. It is thus composed of one single gradient descent iteration in the Batch case, and as many gradient descent iterations as there are training data in the Stochastic Gradient Descent case.*

Training for one epoch consists of the following steps:

1. Initialize the training with random parameters $\theta_{jk}^{(l)}$
2. Calculate $h_\theta\left(x^{(i)}\right)$ for new mini-batch of training set data $x^{(i)}$
3. Calculate cost/loss function $J(\theta)$
4. Calculate gradients by back-propagation
5. Modify parameters $\theta_{jk}^{(l)}$ by gradient descent

MNIST example:



1EDSML
ASO1E2
CE1MGO
SEMS12
EGOCO1

Results of using this FFN on MNIST:

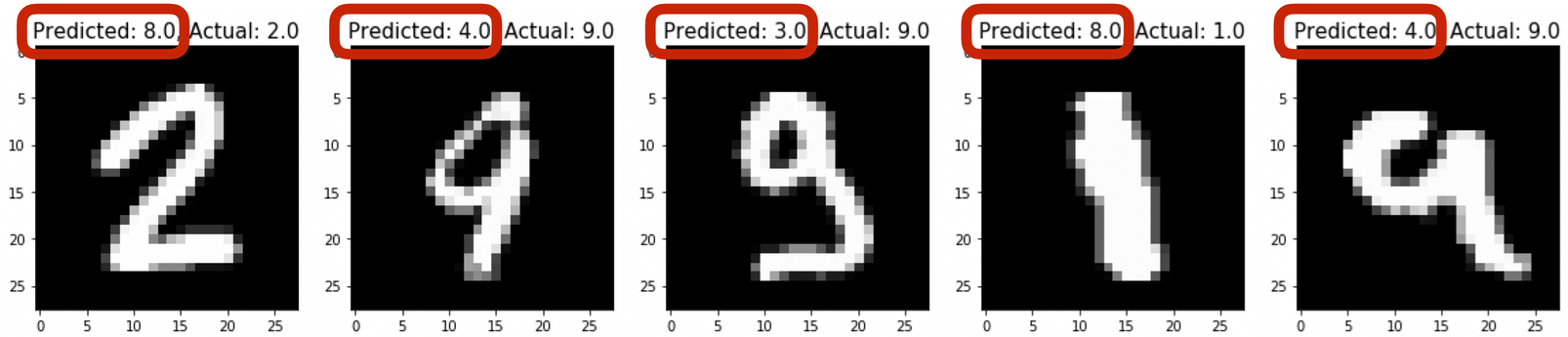
<u>60000 Training Images:</u>

Mean Accuracy: 0.95
Misclassified Images: 2859 (4.8%)

<u>10000 Test Images:</u>

Mean Accuracy: 0.94
Misclassified Images: 618 (6.2%)

misclassified images



Feed-Forward Neural Networks

1. From Logistic Regression to Single Neuron Representation
2. Feed-Forward Neural Networks
3. Back-Propagation
4. Batch, Stochastic and Mini-Batch Gradient Descent
5. **Some basic architectures**

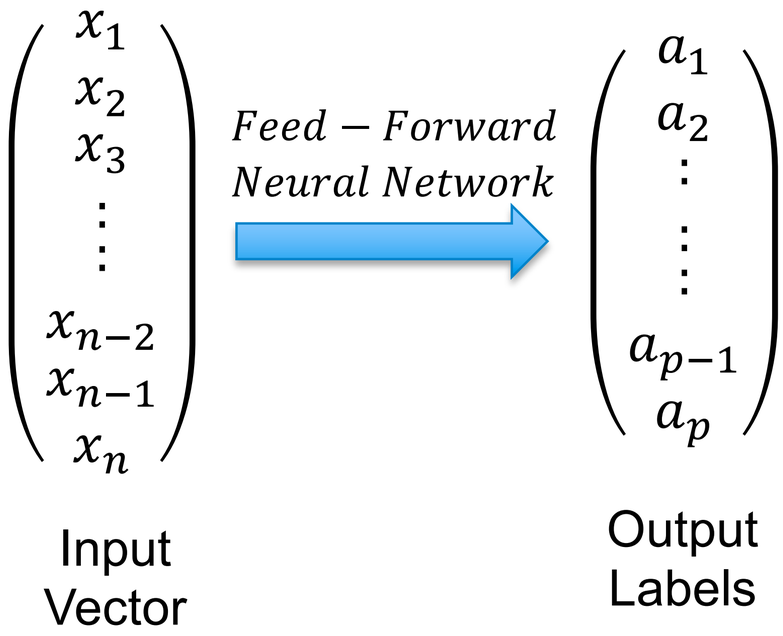
1EDSML
ASD1E2
CE1MGD
SEMS12
EGDCD1

Interpretation of feed-forward networks in supervised learning:

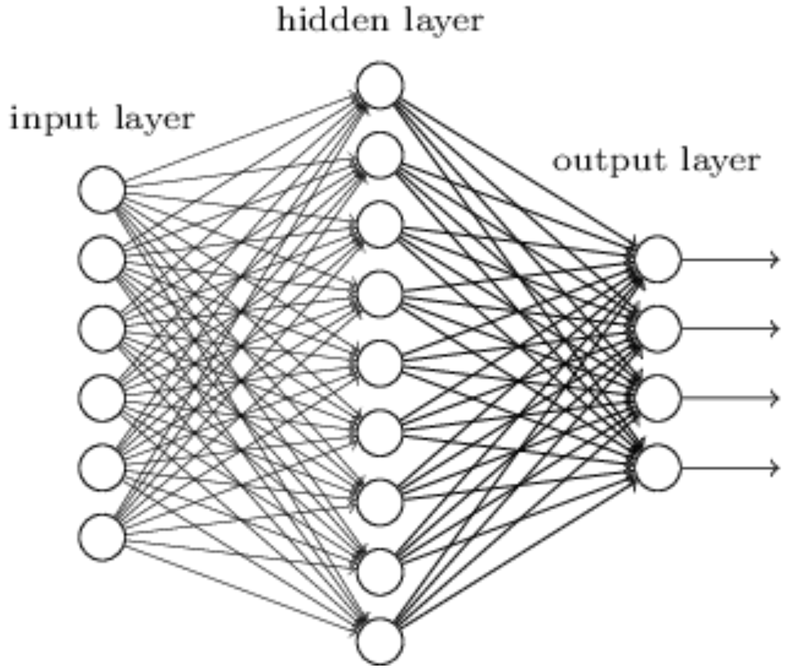
Suppose we have m pairs of data.
Each pair is composed of a vector of dimension n and a vector of dimension p (the labels).

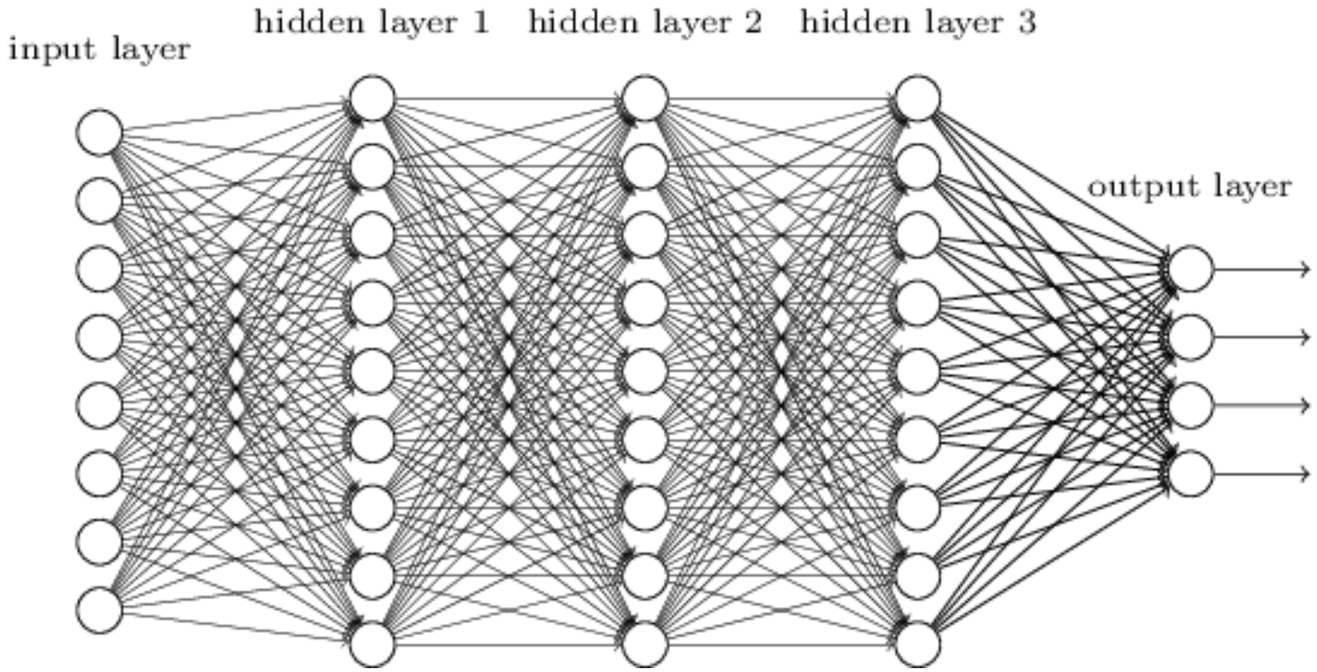
A neural network is simply a function that maps any vector of dimension n into a (discrete or continuous) vector of dimension p.

In order to calculate the parameters of this function, we train the parameters of the neural network by back-propagation using the m pairs of data as Training Set.



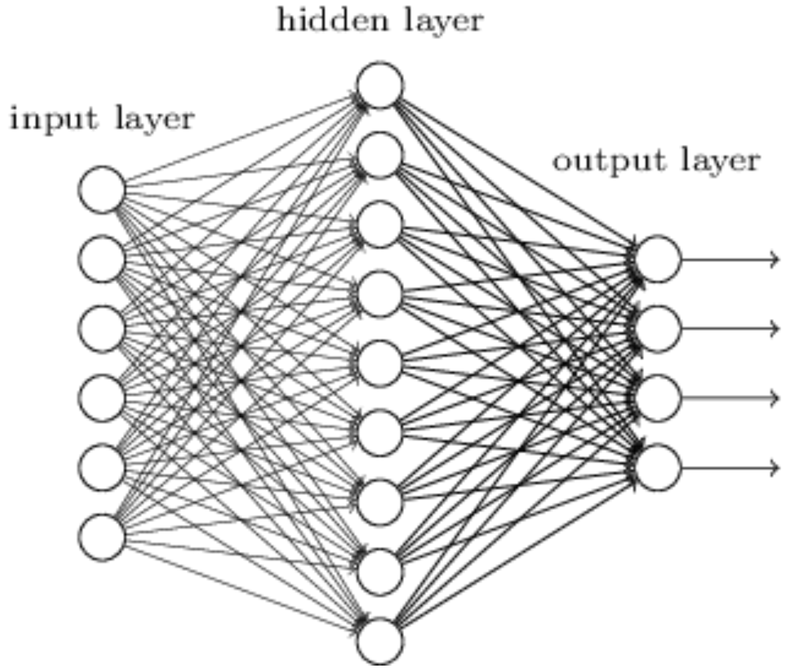
Some examples of simple FFNs:

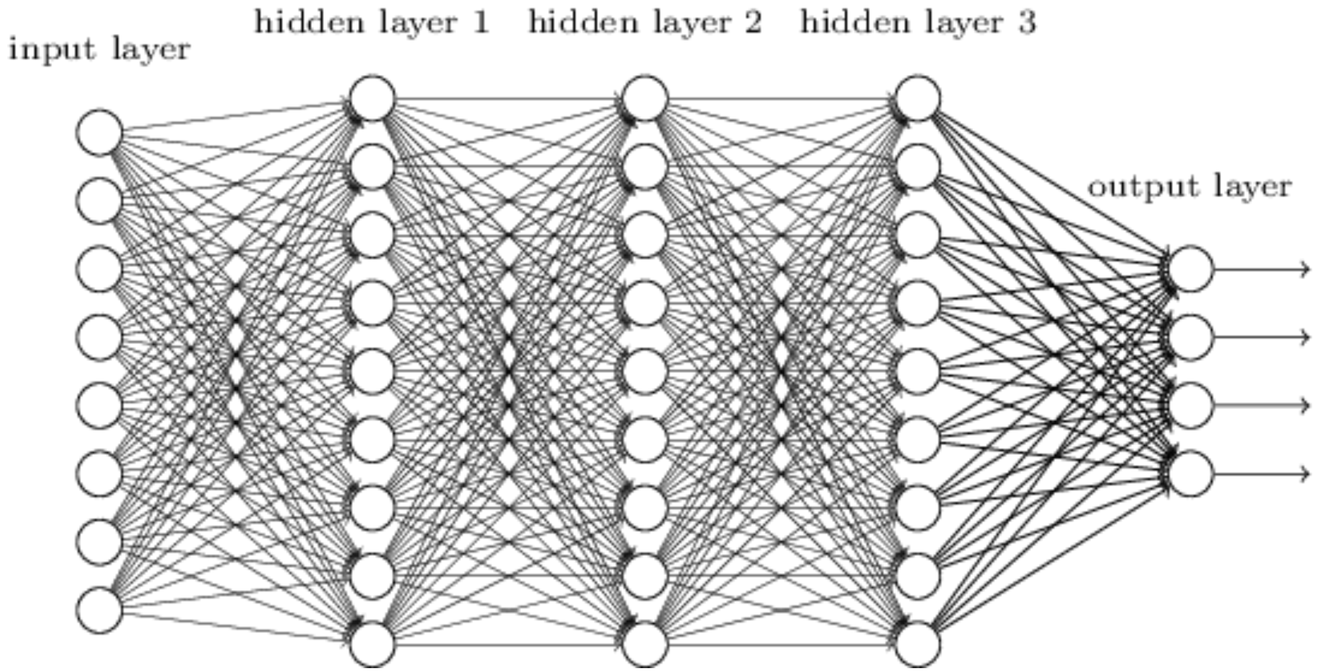




- What do you think is the benefit of having deeper networks?

Some examples of simple FFNs:





- What do you think is the benefit of having deeper networks?
- and the downsides? On Monday we will discuss it.

SUMMARY

- **Logistic regression** as a **single neuron** neural network.
- In feed-forward neural networks, hidden layers provide a «massaging» of the input features in order to make them **linearly separable** by the output layer.
- The **backpropagation** algorithm provides an efficient way to change the network parameters in the direction that reduce the value of the cost function.
- Batch, stochastic, and mini-batch gradient descent are strategies to train networks using different amounts of data at each iteration (**do not confuse iteration with epoch**).
- Supervised neural networks can be interpreted as operators that **map** input vectors into output vectors.