Linear models
[IIA – Lect. ___and ___]

Alessio Micheli

micheli@di.unipi.it

Dipartimento di Informatica
Università di Pisa - Italy

**Computational Intelligence &
Machine Learning Group**

DRAFT, please do not circulate! *2023*

Sintesi (ingredienti)

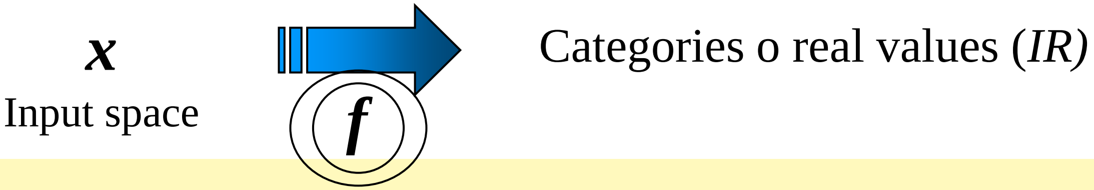
- **Dati** di Allenamento
- **Spazio delle Ipotesi *H***,
  - costituisce l’insieme delle funzioni che possono essere realizzate dal sistema di apprendimento;
  - si assume che la funzione da apprendere ***f*** possa essere rappresentata da una ipotesi ***h*** in ***H*** (selezione di h attraverso i dati di apprendimento)
  - o che almeno una ipotesi ***h*** in ***H*** sia simile a ***f*** (approssimazione);
- Algoritmo di Ricerca nello Spazio delle Ipotesi, **alg. di apprendimento**
  - E.g. Adattamento dei parametri liberi del modello al task
- **NOTA:** ***H*** non puo` coincidere con l’insieme di tutte le funzioni possibili e la ricerca essere esaustiva → *Bias Induttivo*

Tasks: Supervised Learning

- Given: Training examples as <*input,output*>=(*x*,*d*) (***labeled*** **examples**) for an unknown function *f* (known only at the given points of example)

  Def

  - Target value: desiderate value *d* or *t* or $\boxed{y}$... is given by the teacher according to *f*(*x*) to label the data
- Find: A *good* approximation to *f* (i.e. a hypothesis *h* that can used for prediction on unseen data *x'*)



x
Input space
f
Categories o real values (*IR*)

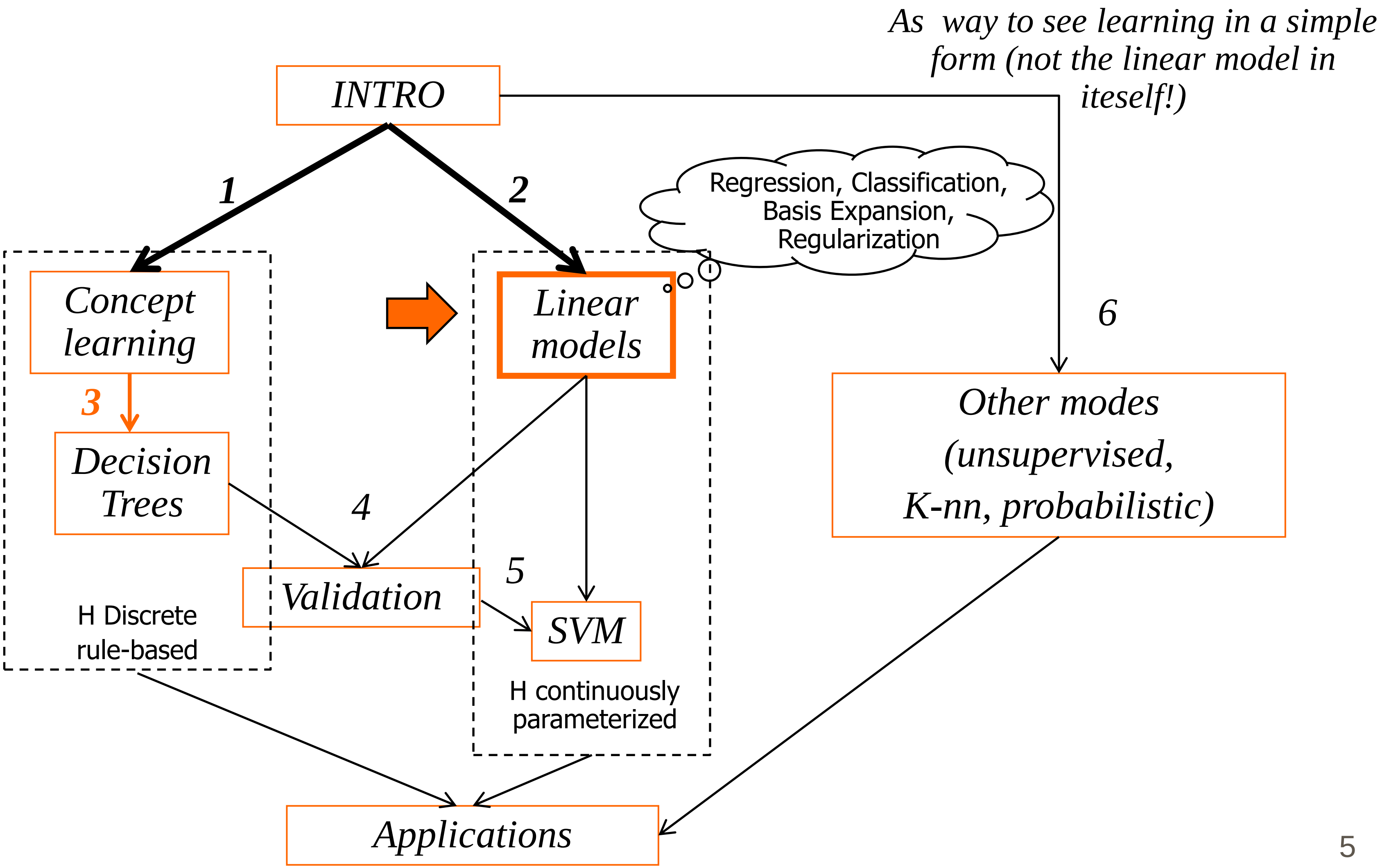
- Target *d* (or *t or y*): a numerical/ categorical *label*
  - *Classification:* discrete value outputs:

    $f(x) \in \{1,2,\dots,K\}$ *classes* (*discrete-valued function*)
  - *Regression:* real continuous output values (approximate a real-valued target function, in R or R^K)

Both as a *task of function approximation*

In the course flow

In the course: example for *numerical eq*. language (H continuous space)


As way to see learning in a simple form (not the linear model in iteself!)
INTRO
1
2
Regression, Classification, Basis Expansion, Regularization
Concept learning
3
Decision Trees
Linear models
6
Other modes (unsupervised, K-nn, probabilistic)
4
Validation
5
SVM
H Discrete rule-based
H continuously parameterized
Applications

Linear models

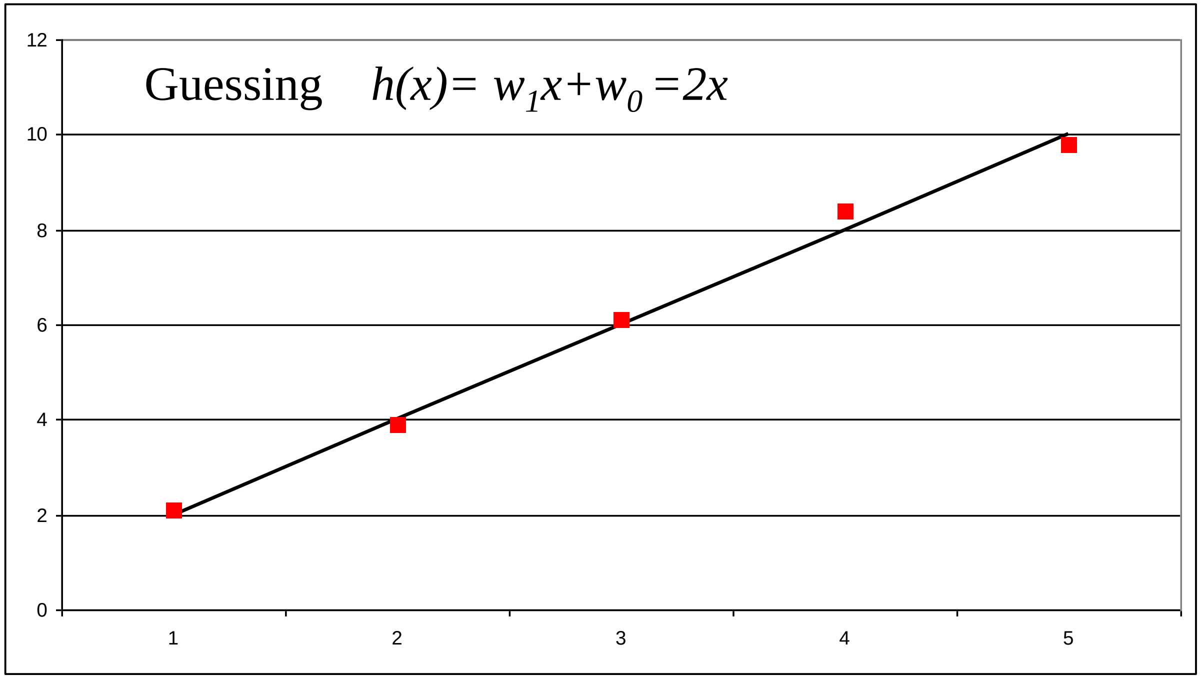
Regression

Tasks: regression: example

- Process of estimating of a real-value function on the basis of finite set of noisy samples
  - known pairs$(x\ ,f(x)+random\ noise)$

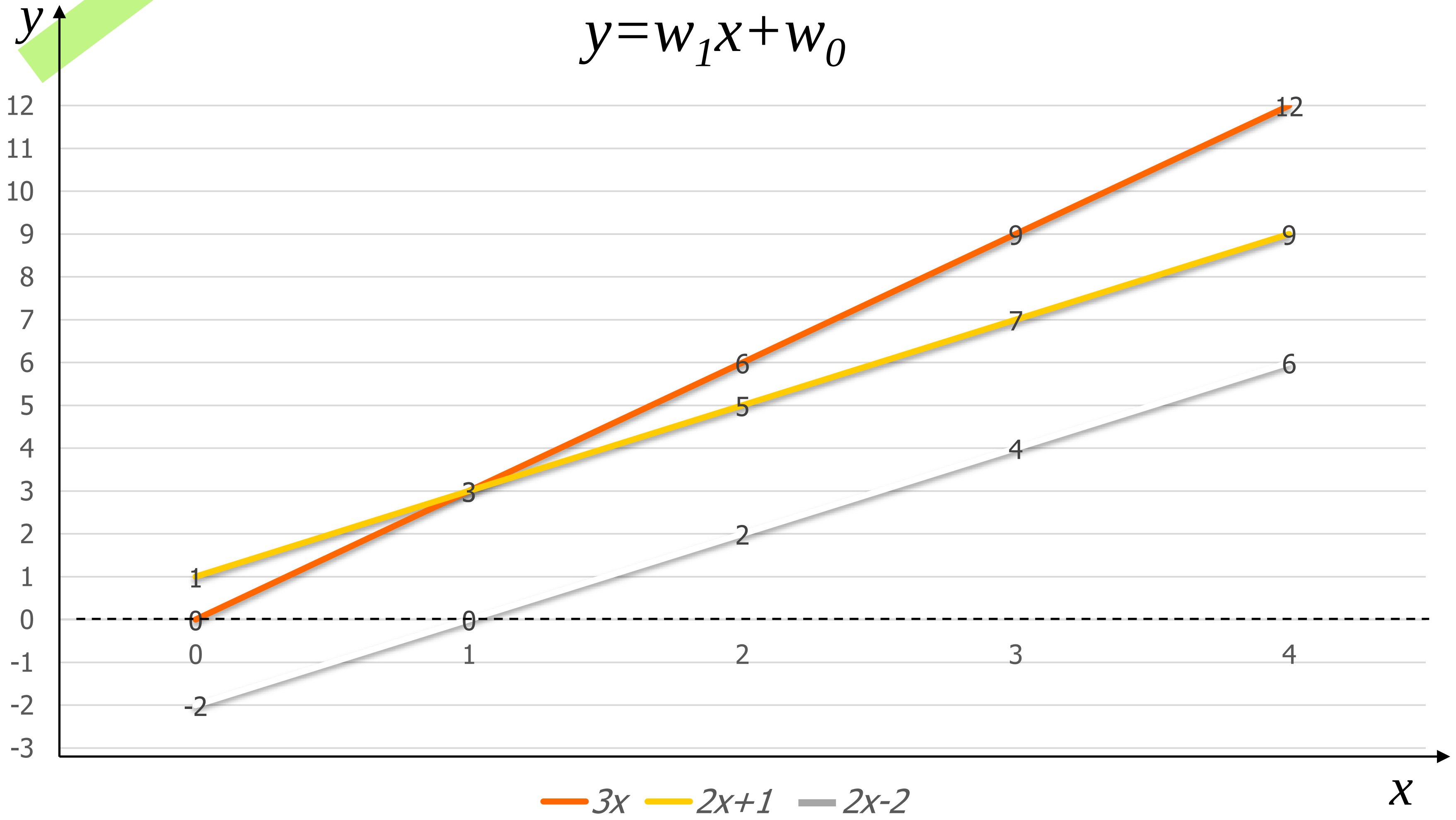
Task (exercise): find f for the data in the following table:

x	$target$
1	2.1
2	3.9
3	6.1
4	8.4
5	9.8
…	…



We want to solve it (how to find w_1 and w_0) in a «systematic» way

Examples of linear models



$$y=w_1x+w_0$$

Linear models

- **How to use the model**
- **Build (learn) the model (a simple regression model)**
  - i.e how to find the ***w*** values of the linear model in a «systematic» way

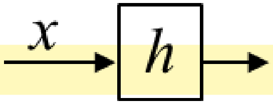
Univariate Linear Regression

- Univariate case, simple linear regression :
- We start with 1 input variable x, 1 output variable y

Def

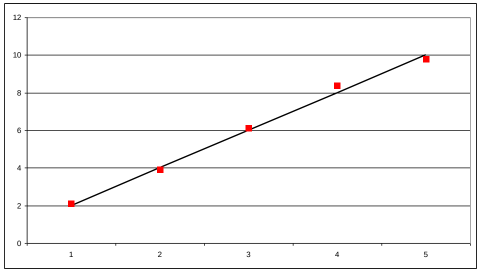
- We assume a model $h_{\mathbf{w}}(x)$ expressed as

$$\boxed{out=h(x)=w_1x+w_0}$$



- *where w are real-valued coefficients/free parameters (weights)*

- ***Fitting*** the data by a "straight line"

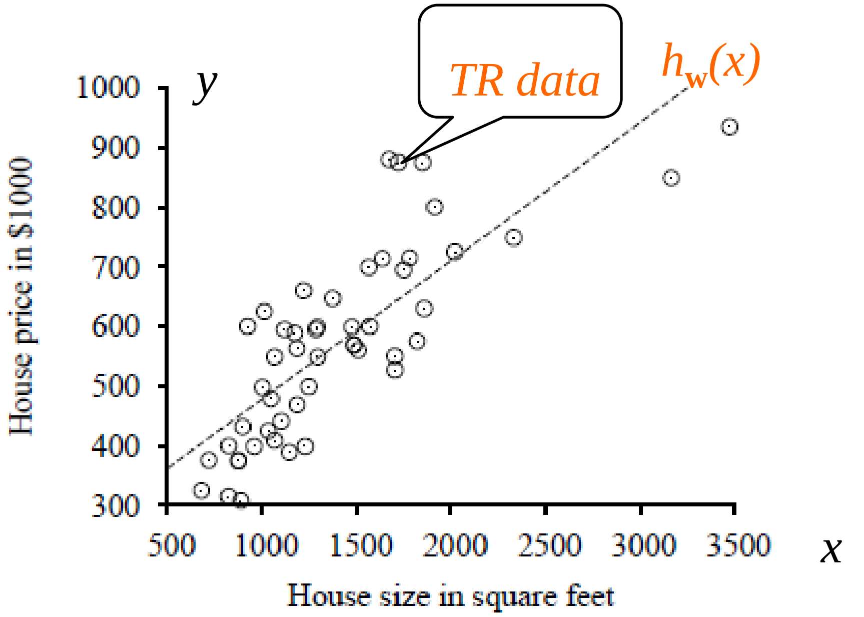


Task and Model (an example)

- Find *h* (linear model) that best *fit* data (from observed data set of x and y values)
- Assuming that the given variable (y) is (linearly) related to another variable (x) (or variables) by $y=w_1x+w_0+$ *noise*, where w are the free par. and the *noise* is the error in measuring the targets (with normal distribution)
- we build a model (finding w values) to predict/estimate the price (y) of points for other (unobserved) x values (prediction)

***AIMA example**: data points of price versus floor space of houses for sale (CA,2009)*

$h_w(x)=0.232x+246$



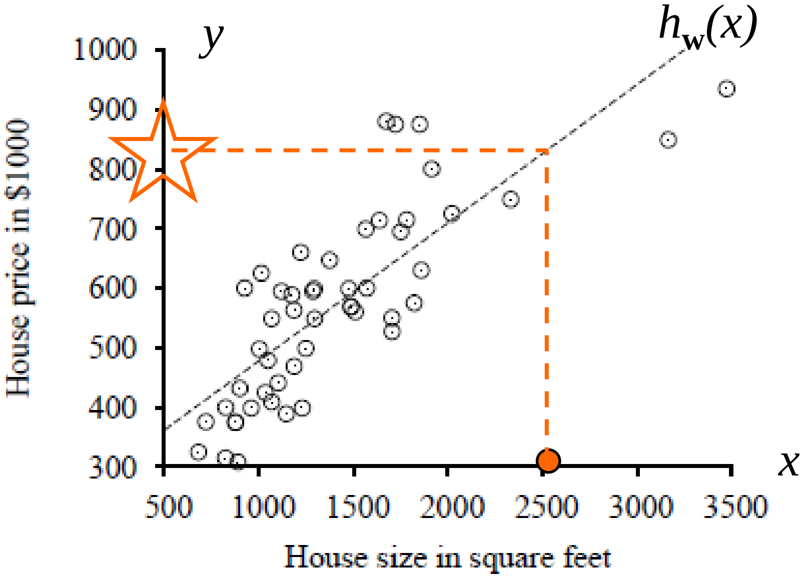
Use it (as ML predictive tool)!

- Predict/estimate the price (y) of points for other (unobserved) x values

***AIMA example**: data points of price versus floor space of houses for sale (CA,2009)*

$h_w(x)=0.232x+246$

Use: $x'=2500 \rightarrow y'=826$



Build it: Learning via LMS (premise)

- We have to find the values of the parameters $\boldsymbol{w}$ (w_1 and w_0 in the univariate case) in order to minimize the model output error (to make a good *fitting*)

- Infinite hypothesis space (continuous w values) but we have nice solution from classical math (going back to Gauss/Legendre ~1795!)
  - Surprisingly we can "learn" by this basic tool
  - Although simple it includes many relevant concept of modern ML and it is a basis of evolute methods in the field

- Let us define a *loss / error function* and use the Least Mean Square (LMS) approach

Build it: Learning via LMS (I)

- Learn → find ***w*** such that minimize **error**/empirical **loss** (best data fitting – on the training set with *l* examples)

Def

- **Given** a set of *l* training examples $(\mathbf{x}_{p,}y_p)$ $p=1..l$
- **Find**: $h_w(x)$ in the form w_1x+w_0 (hence the values of ***w***) that minimizes the expected loss on the training data.

Def

- For the loss we use the square of errors:
- Least (Mean) Square: Find ***w*** to *minimize* the residual sum of squares $[argmin_\mathbf{w}\ Error(\mathbf{w})\ in\ L_2]$:

Def

$$Loss(h_\mathbf{w}) = E(\mathbf{w}) = \sum_{p=1}^{l}(y_p - h_\mathbf{w}(x_p))^2 =$$

p runs over patterns/examples

$$\sum_{p=1}^{l}(y_p - (w_1 x_p + w_0))^2$$

To have the mean divide by l

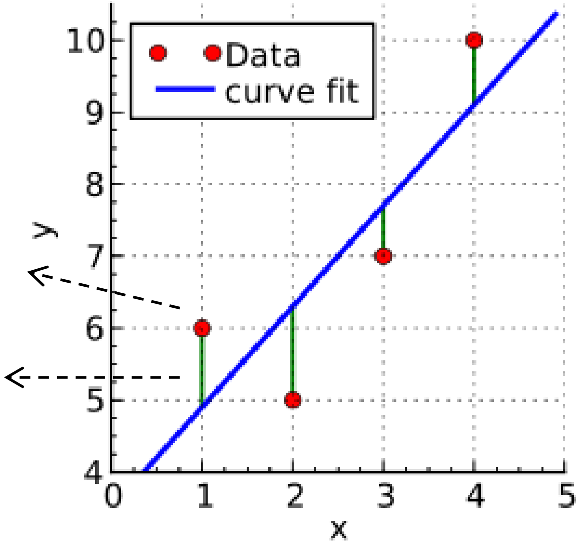
Where x_p is p-th input, y_p the output for p, w free parameters, l num. of examples

Build it: Learning via LMS (II)

Why LMS to find the best *h*?

- Least (Mean) Square: Find **w** to *minimize* the residual sum of squares [*argmin*$_{\mathbf{w}}$ *Error*(**w**) *in* L_2]:

$$y = \overset{h_{\mathbf{w}}(x)}{w_1x+w_0} + noise$$



$(\mathbf{x}_p, y_p)$

Errors

Different blue lines will have different green bars. Minimizing the green bars (residuals /errors) is a way to find the best approximation/fitting of the data (i.e. our $h_{\mathbf{w}}(x)$ *or blue line).*
*The squares of errors E(**w**) quantify such green bars:*

$$E(\mathbf{w}) = \sum_{p=1}^{l} \left(y_p - h_{\mathbf{w}}(x_p)\right)^2$$

- The method of **least squares** is a standard approach to the approximate solution of over-determined systems, i.e., sets of equations in which there are more equations than unknowns*.

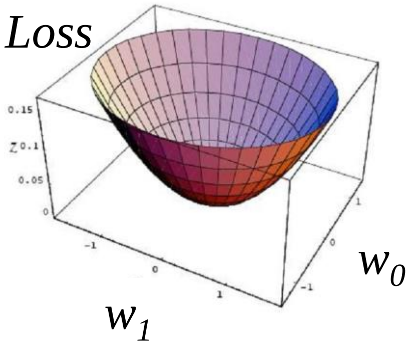
How to solve?

- Remember: local minimum as stationary point: the *gradient* is null

$$\frac{\partial E(\mathbf{w})}{\partial w_i} = 0, \qquad i = 1,\ldots,\dim\_input + 1$$

- For the simple Lin. Regr. (2 free parameters)

$$\frac{\partial E(\mathbf{w})}{\partial w_0} = 0 \qquad \frac{\partial E(\mathbf{w})}{\partial w_1} = 0$$



Convex loss function → we have the following solution (no local minima)

(just to know that it exist!)

$$\boxed{w_1 = \frac{\sum x_p y_p - \frac{1}{l}\sum x_p \sum y_p}{\sum x_p^2 - \frac{1}{l}(\sum x_p)^2} = \frac{\mathrm{Cov}[x,y]}{\mathrm{Var}[x]}}, \qquad \boxed{w_0 = \overline{y} - w_1 \overline{x}}$$

p: 1→l

$$\frac{1}{l}\sum_{p->l} y_p \qquad \frac{1}{l}\sum_{p->l} x_p$$

Where l is num. of examples

***Exercise**: compute w_0 and w_1 according to the next slide results (extended to l patterns)*

Compute the gradient for 1 (each) pattern *p* *[#tech]*

Hence we omit p for x

Basic rules:

$$\frac{\partial}{\partial w} k = 0, \frac{\partial}{\partial w} w = 1, \frac{\partial}{\partial w} w^2 = 2w$$

$$\frac{\partial (f(w))^2}{\partial w} = 2 f(w) \frac{\partial (f(w))}{\partial w}$$

Der. sum = sum of der.

We will call (y-$h(x)$) "delta"

$$\frac{\partial E(\mathbf{w})}{\partial w_i} = \frac{\partial (y - h_{\mathbf{w}}(x))^2}{\partial w_i} =$$

$$= 2(y - h_{\mathbf{w}}(x)) \frac{\partial (y - h_{\mathbf{w}}(x))}{\partial w_i} = 2(y - h_{\mathbf{w}}(x)) \frac{\partial (y - (w_1 x + w_0))}{\partial w_i}$$

$$\boxed{\frac{\partial E(\mathbf{w})}{\partial w_0} = -2(y - h_{\mathbf{w}}(x))}$$

$$\boxed{\frac{\partial E(\mathbf{w})}{\partial w_1} = -2(y - h_{\mathbf{w}}(x)) \cdot x}$$

Redo this by yourself as an exercise

Summarizing

- **Given** a set of l training examples (x_p, y_p) and a loss function
- **Find**: The weight w values to build the $h_w(x)$ expressed as $y/out=w_1x+w_0$ (minimizing the expected loss on the training data)

Direct solution

$$w_1 = \frac{\sum x_p y_p - \frac{1}{l}\sum x_p \sum y_p}{\sum x_p^2 - \frac{1}{l}(\sum x_p)^2}$$

$$w_0 = \bar{y} - w_1 \bar{x} \ .$$

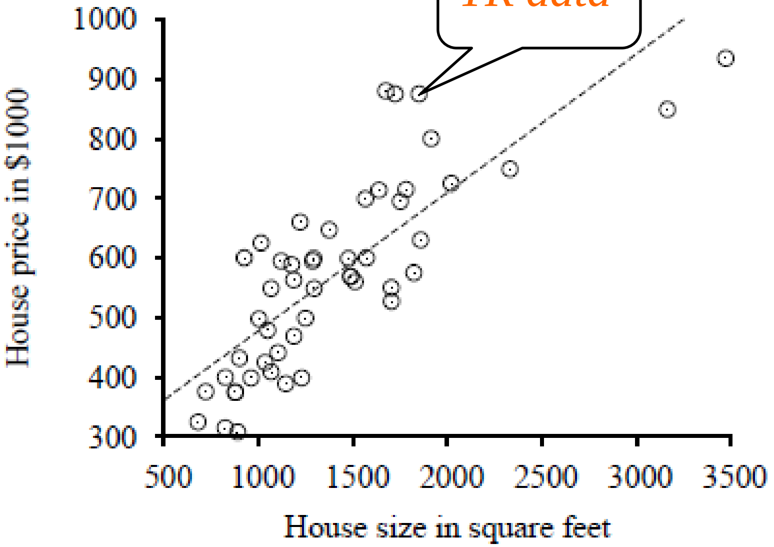
Over p (see later)

E.g.

$$h_w(x) = w_1 x + w_0 = \\ = 0.232x + 246$$

In the AIMA (house) example

- Now you can use it for *new* x' to predict y'



- **Let us see a different approach to use the gradient** → next slide

REPETITA da lezione *local search*

Spazi continui - gradient

- Se la f è continua e differenziabile, e.g. quadratica rispetto ad x
- Il minimo o massimo si può cercare utilizzando il **gradiente**, che *restituisce la direzione di massima pendenza nel punto*
- E.g. data f obiettivo (in 3D)
- $$\nabla f = \left(\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \frac{\partial f}{\partial x_3} \right)$$ *Posizioni 1, 2,3 nel vettore **x***
- Hill climbing iterativo: $\mathbf{x}_{new} = \mathbf{x} + \eta \nabla f(\mathbf{x})$

Uso + per salire (maximization)

Uso – per scendere (minimization)

Eta: step size (e.g. costante positiva<1)

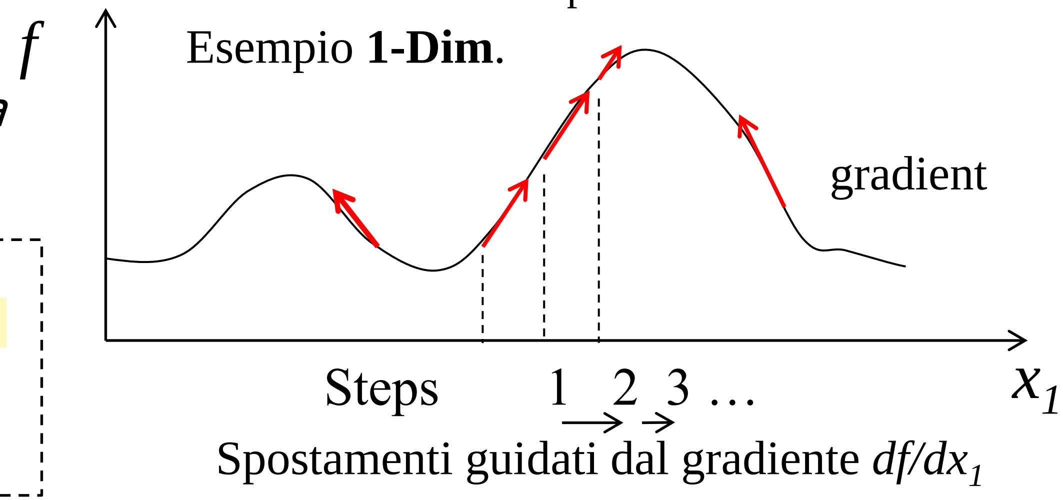
Quantifica lo spostamento, senza cercarlo tra gli infiniti possibili successori!

Nota generale: nel ML non sempre è necessario il min/max assoluto.

N.B: *Questo slide è quello della <u>prima parte del corso</u> (vedere anche l'esempio).*

*Qui usiamo **w** (e non **x**) come variabile della loss E(**w**)*

E il gradiente è $\dfrac{\partial E(\mathbf{w})}{\partial w_i}$



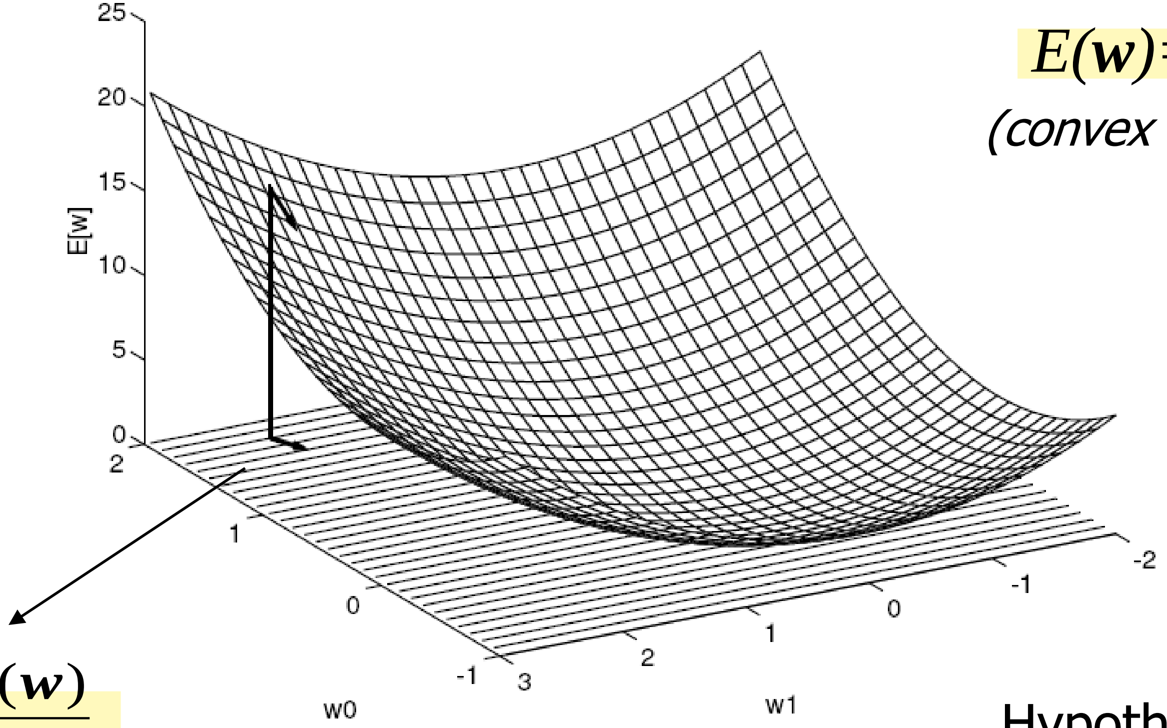
Spostamenti guidati dal gradiente df/dx_1

Error surface for linear model with 2 weights (*w*)

Parabolic for the

$E(\mathbf{w}) = E([w_0, w_1]^T)$

(convex quadratic function)



$$-\frac{\partial E(\mathbf{w})}{\partial \mathbf{w}}$$

Gradient of $E(\mathbf{w})$

Our "compass" to find the minimum

Hypothesis space with 2 parameters w_0, w_1

Gradient descent (local search)

- Previous derivation suggest the line to construct an **iterative algorithm** based on : $\frac{\partial E(\mathbf{w})}{\partial w_i}$

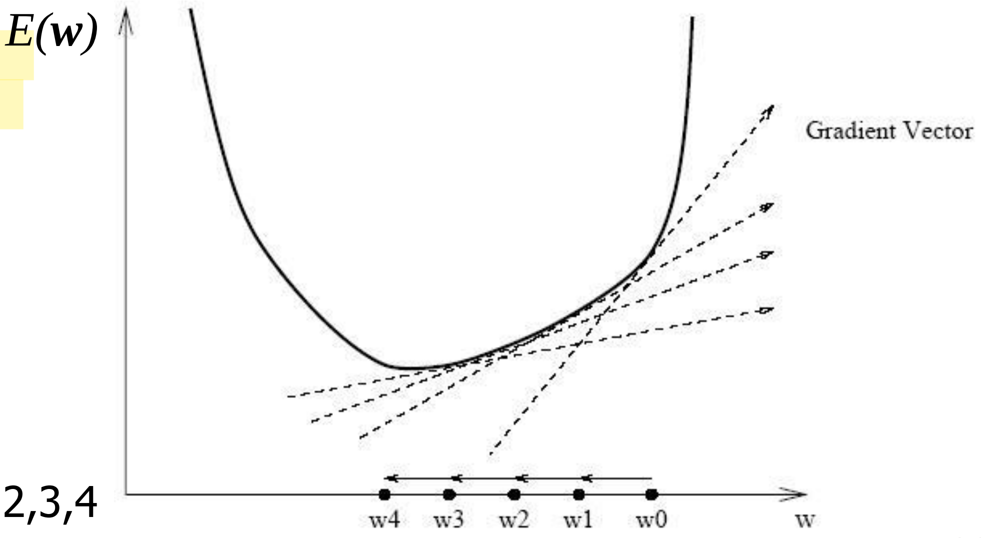
Def
- **Gradient** = *ascent direction*: we can move toward the minimum with a gradient *descent* (Δw= -gradient of $E(\mathbf{w})$)

→ *Local search*: begins with initial weight vector. Modifies it iteratively to decrease up to minimize the error function (steepest descent).

Def

$$\mathbf{w}_{\text{new}} = \mathbf{w} + eta*\Delta\mathbf{w}$$

Where eta (η) is a constant (step size), called ***learning rate***



A single *w* at step 0,1,2,3,4

Gradient descent: intuitive motivations

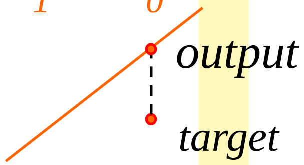
$\mathbf{w}_{\text{new}} = \mathbf{w} + eta*\Delta\mathbf{w}$, where we obtained:

$$\Delta w_0 = -\frac{\partial E(\mathbf{w})}{\partial w_0} = 2(y - h_{\mathbf{w}}(x)) \qquad \Delta w_1 = -\frac{\partial E(\mathbf{w})}{\partial w_1} = 2(y - h_{\mathbf{w}}(x)) \cdot x$$

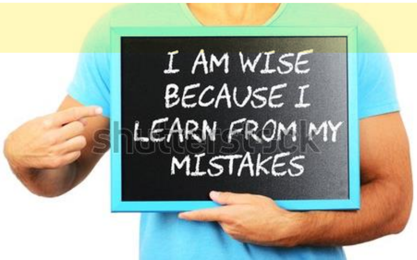
Def

- This is an "error correction" rule (call it ***delta rule***) that changes the $\mathbf{w}$ proportional to the error (*target-output*):
  - (*target* y – *output*) = err=0 → no correction
  - *output*>*target* → (y-h)<0 (output is too high)
    - → Δw_0 negative → reduce w_0 and
    - if (input x >0) Δw_1 negative → reduce w_1 (else increase w_1)
  - *output*<*target* → (y-h) >0 (output is too low) … [exercise]

$h_{\mathbf{w}}(x)=w_1x+w_0$



- *We improve learning form previous errors*



Gradient descent: final discussion

Gradient descent approach it is simple and effective local search approach to LMS solution and

- It allows us to search through an *infinite hypothesis* space!
- It can be easily always applied for *continues* H and differentiable loss: NOT ONLY to linear models !!!! (e.g. Neural Networks and deep learning models)
- *Efficient*? Many improvement are possible, e.g. Newton's methods; Conjugate Gradient, …

Linear models

- **Extension to l data**
- **Extension to input vectors**
- **Summary of the LMS**
- **The gradient descent training algorithm**

For *l* patterns

We sum up for *l* patterns (x_p, y_p) …

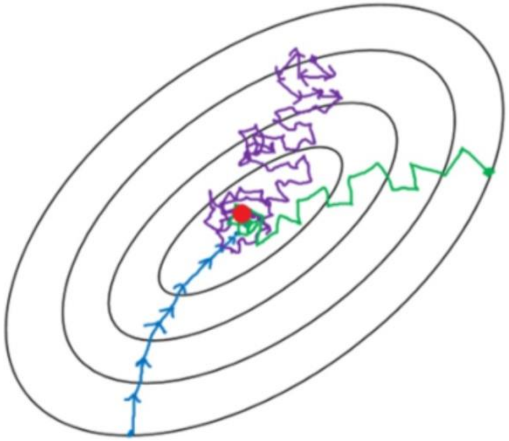
Def

$$\Delta w_0 = -\frac{\partial E(\mathbf{w})}{\partial w_0} = 2\sum_{p=1}^{l}(y_p - h_{\mathbf{w}}(x_p)) \qquad \Delta w_1 = -\frac{\partial E(\mathbf{w})}{\partial w_1} = 2\sum_{p=1}^{l}(y_p - h_{\mathbf{w}}(x_p)) \cdot x_p$$

Where x_p is p-th input, y_p the output for p, w free par., l num. of examples

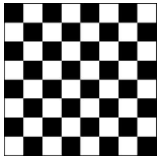
Exercise: Compute these Δ*w*, *hint: der. of sum = sum of der.*

- We can update ***w*** after (repeating) an "epoch" *l* training data → *Batch algorithm* (blu) (as used above)
- Or we can update ***w*** after each pattern *p* → *On-line algorithm* (stochastic gradient descent) (purple and green):
  - it can be the faster, but need smaller step



Multidimensional input (input vectors x): examples

- This is the standard case, using 2 up to hundreds of input variables/features $\boldsymbol{x} = [x_1, x_2, .., x_n]$ *The input **pattern** is a vector*



  - E.g. Mitchell chap.1: Evaluate (score) the **board state** in a checkers game by num. of black/white pieces/kings/captured pieces in the next turn : 6 variables. $\boldsymbol{w}$ weights such "board" features.
  - E.g. AIMA example: **house price** considering also the num. of rooms, age of the house, district rank, …

- … many many possibilities in a wide range of applicative fields…

Resume on DATA notation

Pattern	x_1	x_2	x_i	x_n
Pat 1	$x_{1,1}$	$x_{1,2}$		$x_{1,n}$
…				
Pat *p*	$x_{p,1}$	$x_{p,2}$	$\boxed{x_{p,i}}$	$x_{p,n}$
…				

$\mathbf{X}$ is a matrix $\mathbf{l \text{ x } n}$

$\mathbf{l}$ rows, $\mathbf{n}$ columns (features, variables, attributes)

$p=1..l, \quad i=1..n$

We often need to omit some indices when the context is clear, e.g.:

- Each row, generic $\mathbf{x}$ (vector - bold), a raw in the table: (input) example, pattern, instance, sample, …, input vector, ...
- x_i or x_j (scalar): component *i* or *j* (given a pattern, i.e. omitting *p*)
- $\mathbf{x}_p$ (or $\mathbf{x}_i$) (vector – bold) *p-th* (or *i-th*) raw in the table = pattern *p* (or *i)*
- $\boxed{x_{p,i}}$ (scalar) also as $(\mathbf{x}_p)_i$: component *i* of the pattern *p*

  (or we use $x_{p,j}$ for the component *j, etc.*)
- For the target we will typically use just y_p with $p=1..l$ (the same for *d* or *t*)
- *Note: in the slide <Learning via LMS>,* univariate case with 1 variable: $x_p=x_{p,1}= (\mathbf{x}_p)_1$

Multidimensional input: Notation

Previously also
d_p or t_p

- Assuming column vector for $\boldsymbol{x}$ and $\boldsymbol{w}$ *(in bold)*
- Number of data l, dimension of input vector n, y_p *(targets)* $p=1..l$

$$\boldsymbol{w}^T\boldsymbol{x} + w_0 = w_0 + w_1x_1 + w_2x_2 + \cdots + w_nx_n = w_0 + \sum_{i=1}^{n} w_ix_i$$

- Note that often, as before, the transpose notation T in $\boldsymbol{w}^T$ is omitted

- w_0 is the *intercept, threshold, bias, offset*…..

Often it is convenient to include the constant $x_0 = 1$ so that we can write it as :

$$\boldsymbol{w}^T\boldsymbol{x} = \boldsymbol{x}^T\boldsymbol{w}$$

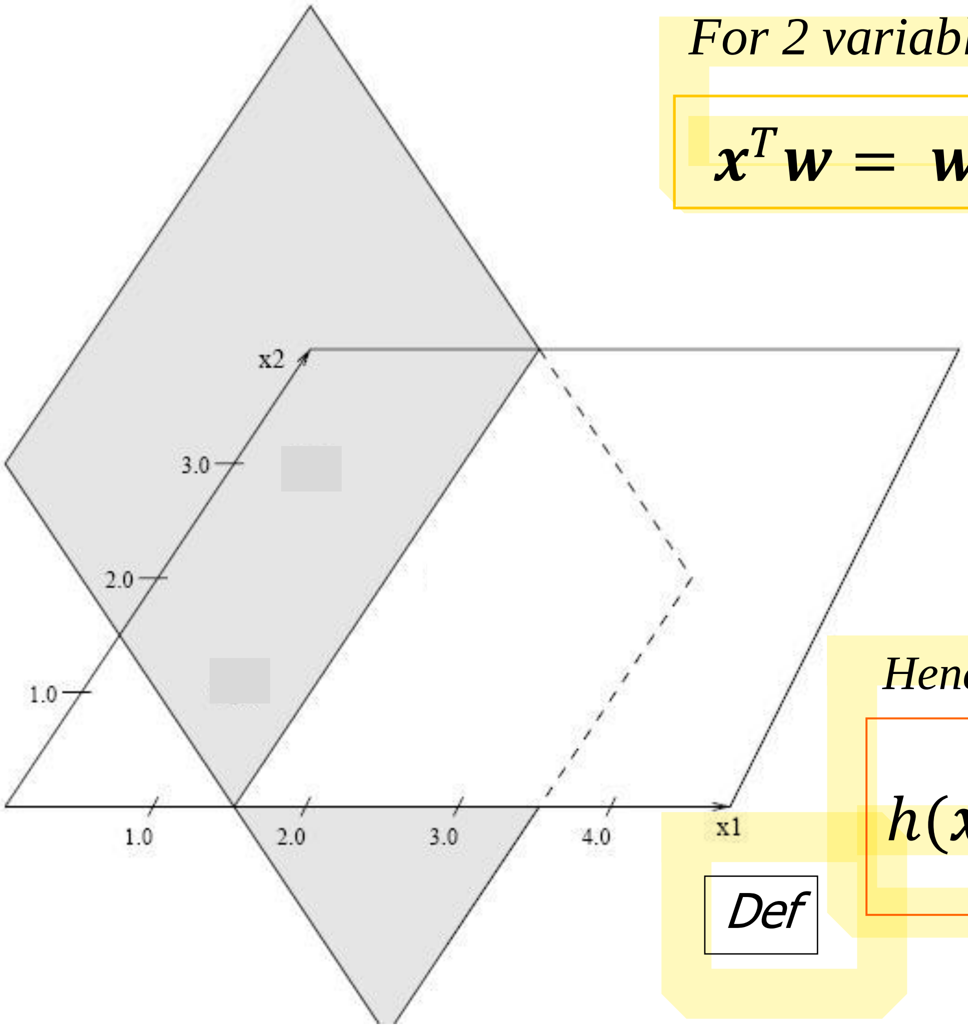
Inner product

$$\boldsymbol{x}^T = [1, x_1, x_2, .., x_n]$$

$$\boldsymbol{w}^T = [w_0, w_1, w_2, .., w_n]$$

Note: ***w*** **continuous (free) parameters "weights"**

Geometrical view: hyperplane



For 2 variables

$$\boldsymbol{x}^T\boldsymbol{w} = \boldsymbol{w}^T\boldsymbol{x} = w_0 + w_1x_1 + w_2x_2$$

n dim:

$$\boldsymbol{w}^T = [w_0, w_1, w_2, .., w_n]$$

Pattern p

Feature i

Hence:

$$h(\boldsymbol{x}_p) = \boldsymbol{x}_p{}^T\boldsymbol{w} = \sum_{i=0}^{n} x_{p,i} w_i$$

Def

Summary:
Input dimension *n*, *l* patterns

- **Given** a set of l training examples $(\boldsymbol{x}_p, y_p)$
- **Find**: The weight vector $\boldsymbol{w}$ that minimizes the expected loss on the training data

Def

$$E(\mathbf{w}) = \sum_{p=1}^{l} (y_p - \boldsymbol{x}_p{}^T\boldsymbol{w})^2 = \quad ||\boldsymbol{y} - \boldsymbol{X}\boldsymbol{w}||^2$$

Where $\mathbf{x}_p$ is p-th input vector, y_p the output for p, $\mathbf{w}$ free par., l num. of examples, n input dim.

- You will learn in Numerical Analysis/ ML courses how to solve it in general (direct solution) by the **normal equation**
- We still have the **iterative gradient descent algorithm**:

Def

$$\Delta w_i = -\frac{\partial E(\boldsymbol{w})}{\partial w_i} = 2\sum_{p=1}^{l}(y_p - h_{\boldsymbol{w}}(\boldsymbol{x}_p)) \cdot x_{p,i} \quad = 2\sum_{p=1}^{l}(y_p - \boldsymbol{x}_p{}^T\boldsymbol{w})\, x_{p,i}$$

To have the mean divide by l

2 is constant that can be ignored to develop the algorithm

Component i of pattern p

This is a VERY EASY vectorial extension to the gradient:

Each x_i has its w_i and its Δw_i (as the single x before)

The gradient vector for dimension *n*

$$\Delta\boldsymbol{w} = -\frac{\partial E(\boldsymbol{w})}{\partial \boldsymbol{w}} = \begin{bmatrix} -\dfrac{\partial E(\boldsymbol{w})}{\partial w_1} \\ -\dfrac{\partial E(\boldsymbol{w})}{\partial w_2} \\ -\dfrac{\partial E(\boldsymbol{w})}{\partial w_i} \\ \dots \\ -\dfrac{\partial E(\boldsymbol{w})}{\partial w_n} \end{bmatrix} = \begin{bmatrix} \Delta w_1 \\ \Delta w_2 \\ \Delta w_i \\ \dots \\ \Delta w_n \end{bmatrix}$$

We can work in a multi-dim space without the need to visualize it

P.s. w_0 is omitted above

Summary: Gradient descent algorithm

Def A simple algorithm:

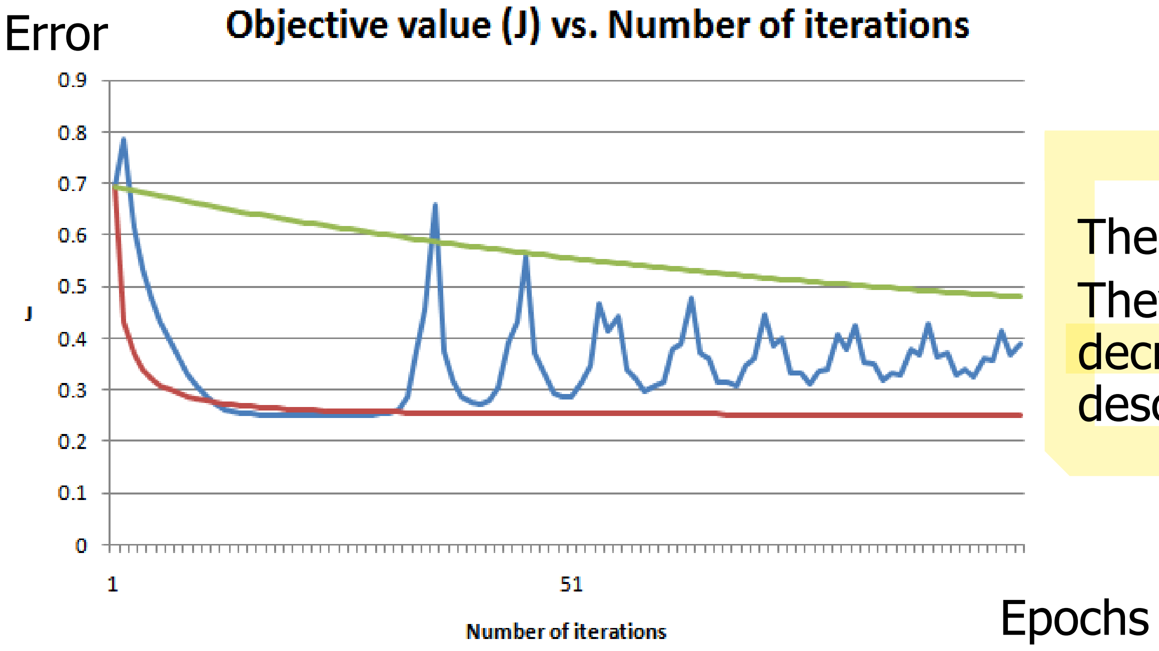
1) Start with weight vector $\boldsymbol{w}_{\text{initial}}$ (small), fix *eta* $(0<eta<1)$.
2) Compute $\Delta \boldsymbol{w}$= – "gradient of $E(\boldsymbol{w})$" = - $\frac{\partial E(\boldsymbol{w})}{\partial \boldsymbol{w}}$ (i.e. for each w_i)
3) Compute $\boxed{\boldsymbol{w}_{\text{new}} = \boldsymbol{w} + eta * \Delta \boldsymbol{w}}$ (i.e. for each w_i) - - - -> Learning rule

   where *eta* is the "step size" (learning rate) parameter
4) Repeat (2) until convergence or $E(\boldsymbol{w})$ is "sufficiently small"

You can see the Python alg. for details
Estratto-LinearLearner-per-lezione.pdf

- ***$\Delta w / l$: least <u>mean</u> squares***
- Batch versions ($\Delta \boldsymbol{w}$ after each "epoch" of l data): as previous 2 slides
- Stochastic/on-line version: upgrade $\boldsymbol{w}$ for each pattern p (by $\Delta_p \boldsymbol{w} = \boldsymbol{x}_p(y_p - \boldsymbol{x}_p^T \boldsymbol{w})$, without wait the total sum over l)
- eta = learning rate: speed/stability trade-off: can be (gradually) decreased to zero (guarantee convergence, avoiding oscillation around the min.)

Learning curve examples



Def

These are **learning curves**:
They show how the error decreases through gradient descent iterations

Exercise: *1 is slow, 1 is unstable, 1 is good: which one?*

Advantages of linear models

- If it works well it is a “wonderful” model
  - Very simple
  - All the information on data are in w
  - Easy to be interpreted: everyday practice in medicine, biology, chemistry, economy, …
  - Noisy data are allowed
  - Statisticians are happy (nice properties)
  - Linear phenomena: a dream for science: ideal to make a “natural law”

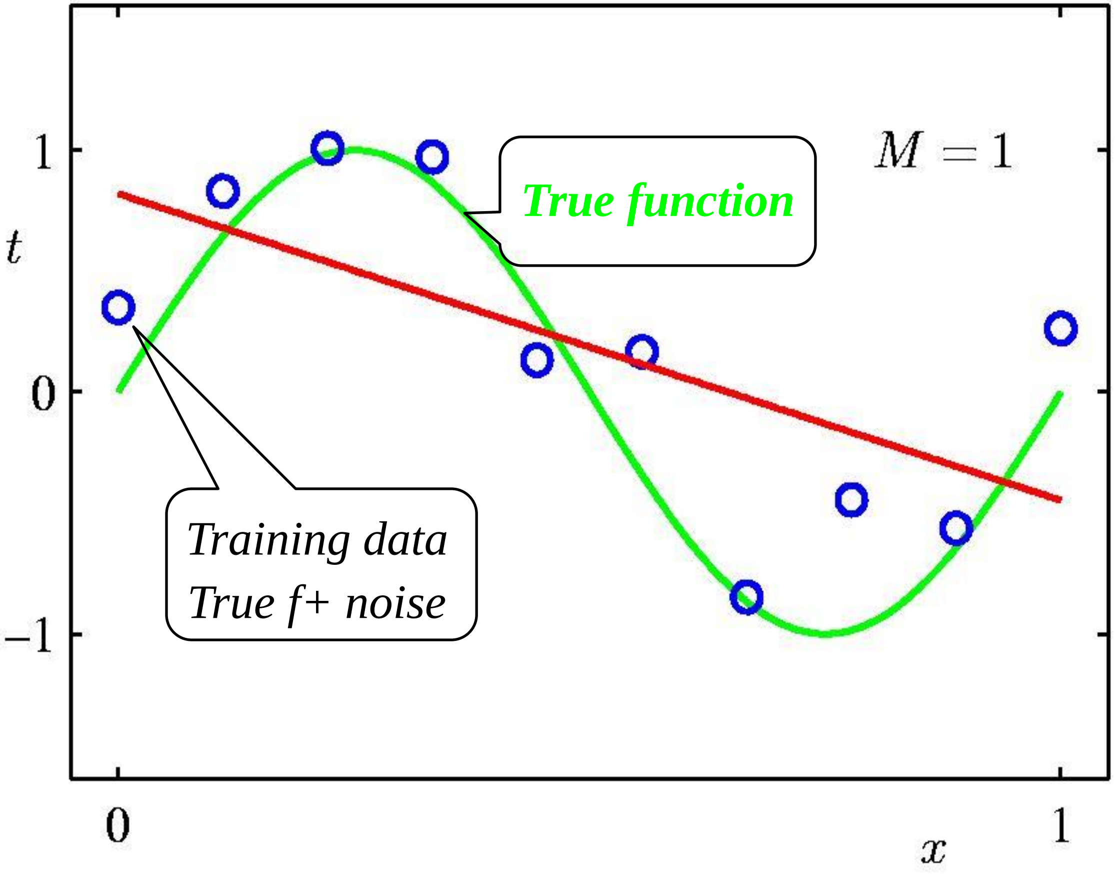
- A baseline for learning (first: is it a linear problem?)
- It is used/included in more complex models

PP*

Linear models

- Limitations
- Linear Basis Expansion
- Regularization

Limitations: regression tasks for non linear problems



True function

Training data
True *f*+ noise

Too poor solution

Moving toward non-linear relationships

- Note that in $h_{\mathbf{w}}(x)=w_1x+w_0$ or $h_{\boldsymbol{w}}(\boldsymbol{x}) = \boldsymbol{w}^{\mathrm{T}} \cdot \boldsymbol{x}$
- As Statistical Parametric models:

  "linear" does not refer to this straight line, but rather to the way in which the regression coefficients $\boldsymbol{w}$ occur in the regression equation

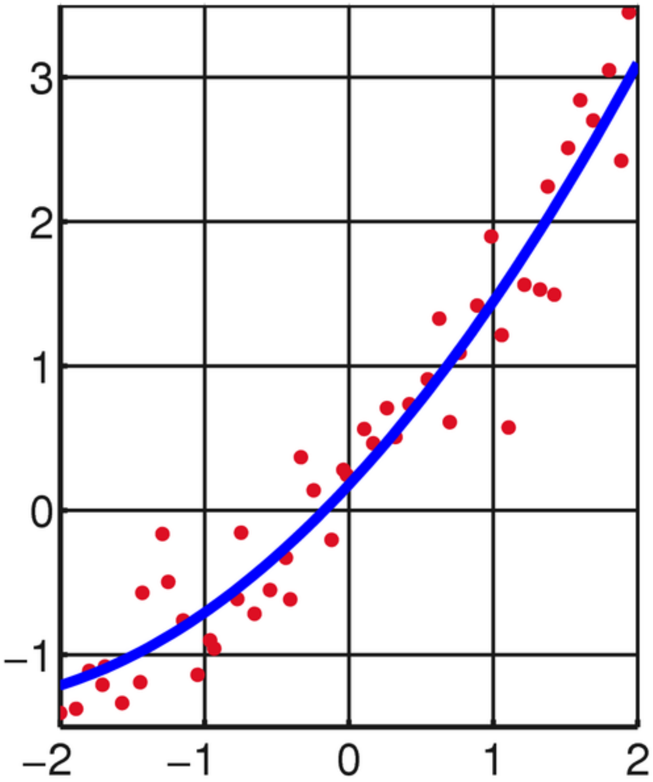
- Hence, we can use also transformed inputs, such are x, x^2, x^3, x^4, …. with *non-linear* relationship inputs and output, holding the learning machinery (Least Square solution) developed so far…

$$h_{\mathbf{w}}(x) = w_0 + w_1 x + w_2 x^2 + \ldots + w_M x^M = \sum_{j=0}^{M} w_j x^j$$

polynomial regression

Example

- The result of fitting a quadratic function (in blue), **M=2**, through a set of data points (in red).
- In linear least squares the function need *not* be **linear** in the *argument* (input variables) but *only in the parameters (**w**)* that are determined to give the best fit.



A generalization - LBE

(shown for regression)

- Basis transformation: **linear *basis expansion* :**

Def

$$h_{\boldsymbol{w}}(\boldsymbol{x}) = \sum_{k=0}^{K} w_k \phi_k(\boldsymbol{x})$$

- Augment the input vector with additional variables which are transformations of x according to a function phi (ϕ_k: $R^n \rightarrow R$)
- E.g
  - Polynomial representation of $\mathbf{x}$: $\phi(\mathbf{x})=x_j^2$ *or* $\phi(\mathbf{x})= x_j\, x_i,$ *or* ….
  - Non-linear transformation of single inputs: $\phi(\mathbf{x})=log(x_j)$, $\phi(\mathbf{x})= root(x_j)$, ….
  - Non-linear transformation of multiple input: $\phi(\mathbf{x})= ||x||$
  - ***Splines, …, ….***
- **Number parameters $K > n$** (before it was n)
- The model is *linear in the parameters (also in phi, not in **x**): we can use the same learning alg. as before!*

Basis Expansion: examples

- Basis transformation: **linear *basis expansion* :**

$$h_{\boldsymbol{w}}(\boldsymbol{x}) = \sum_{k=0}^{K} w_k \phi_k(\boldsymbol{x})$$

EXAMPLES:

- *[1-dim **x**]* $\phi_j(x) = x^j.$

  $h(\mathbf{x}) = w_0 + w_1 x + w_2 x^2 + \ldots + w_M x^M = \sum_{j=0}^{M} w_j x^j$

  1-dim polynomial regression (K=M)

[For the number of terms in polynomial with D dim. input see a slide later]

- Or *"any other", e.g.* $\phi(\mathbf{x})=\phi([x_1, x_2, x_3])$

$h(\mathbf{x}) = w_1x_1 + w_2x_2 + w_3\log(x_2) + w_4\log(x_3) + w_5(x_2x_3) + w_0$

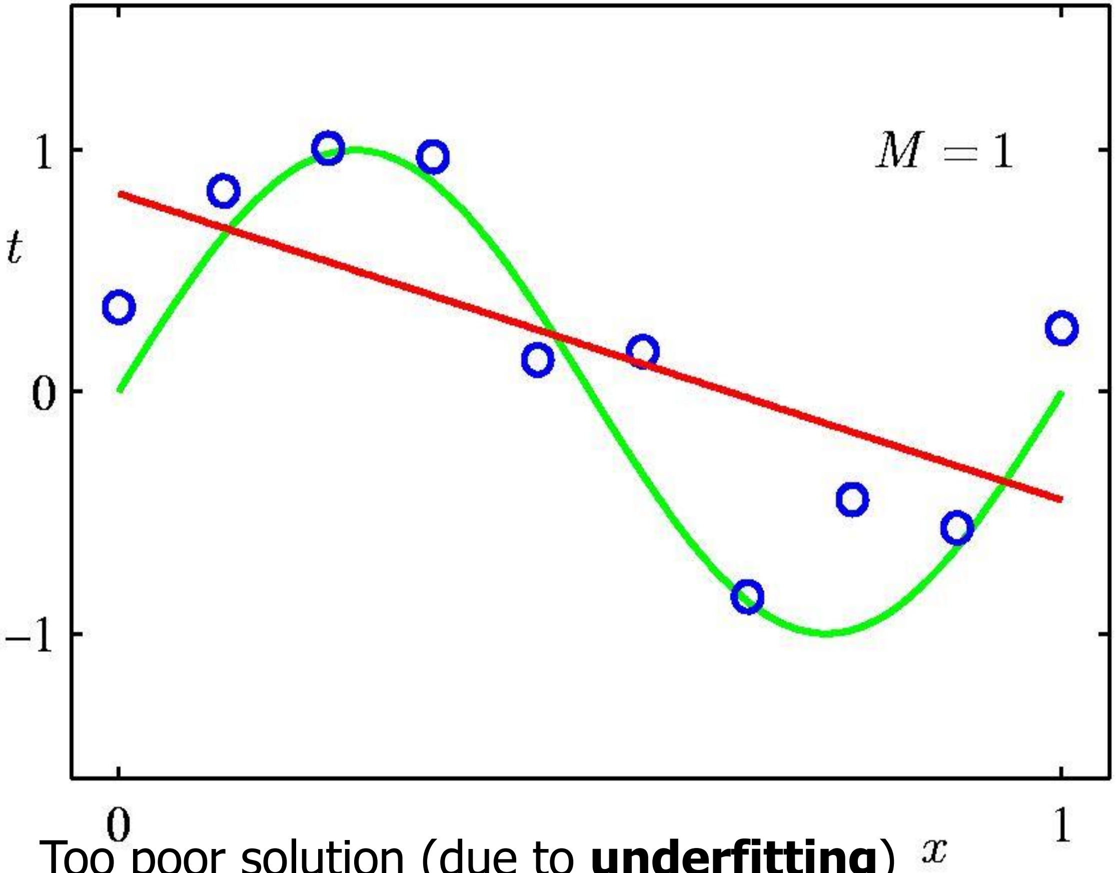
Exercize*: try to compute the gradient for such cases*

Basis Expansion criticism

- Which phi (ϕ) ? Toward the so called "**dictionary**" approaches
- PROS: It is more expressive: it can model more complicated relationships (than linear).
- CONS: With large basis of functions, we require methods for controlling the *complexity* of the model:
  - see the motivation in the next slides….

1st Order Polynomial (linear model)

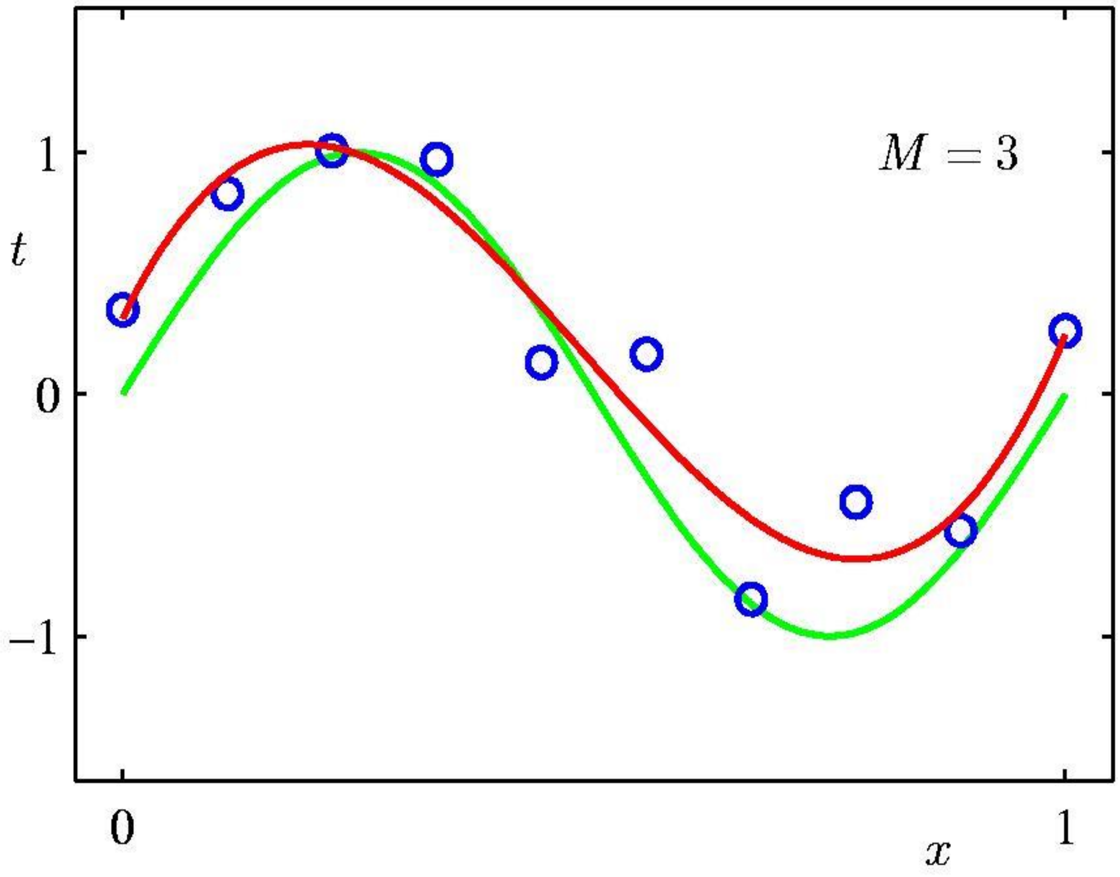
$$h_{\mathbf{w}}(x) = w_0 + w_1 x + w_2 x^2 + \ldots + w_M x^M = \sum_{j=0}^{M} w_j x^j$$



© Bishop PRML

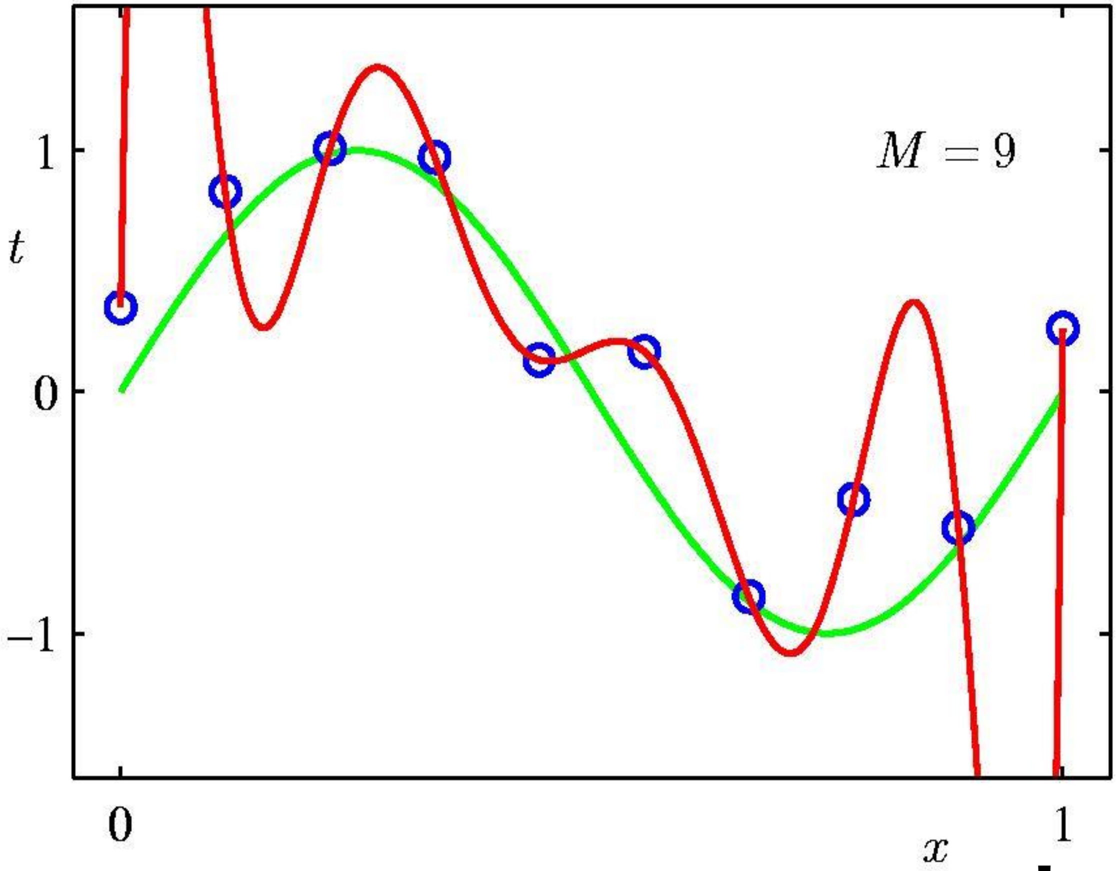
Too poor solution (due to **underfitting**)

3rd Order Polynomial



More flexibility is useful !!!

9th Order Polynomial



Much more flexibility, **but may be excessive!**

E($\boldsymbol{w}$)= 0 on training data!!! But error on test set ?

Too complex model: fit the noise!

Poor representation of the (green) true function (due to **overfitting**)

Complexity tradeoff

- Too simple model:
  - Does not even fit the data well

Underfitting

- Too complex model:
  - Highly sensitive to slight perturbations of the data

Overfitting

- Want to choose regularization to balance out the two cases **trough control of model complexity** (seen as flexibility)

- Whereas **complexity** is **not** for the computational cost but a measure of the flexibility of the model to fit the data

Toward Regularization

- **Regularization** can *control overfitting* by penalizing "complex" functions with large weights (*w*) values (i.e. toward coefficient shrinkage) or the number of free parameters (*w* not = zero)
- While maintain the flexibility of the hypothesis space
- **Occam (Ockham) razor** (1300!)…
  "The simplest explanation is more likely the correct one" Or *"prefer the simplest hypothesis that fits the data"*
- This fundamental concept in ML will be found again and we will "rationalize" it at the end (to quantify it see #ML)
- Now let us introduce an approach for linear models: regularization by ridge regression

Regularization by Ridge Regression

Def

- Ridge regression / **Tikhonov regularization:** smoothed models
  – possible to add constraints to the sum of value of $|w_j|$ favouring "sparse" models e.g. with less terms due to weights w_j =0 (or close tro 0) (it means a less complex model)

Error data term + Regularization/penalty term

Def

$$Loss(h_{\mathbf{w}}) = \sum_{p=1}^{l} (y_p - h_{\mathbf{w}}(\mathbf{x}_p))^2 + \lambda \|\mathbf{w}\|^2 \longrightarrow \sum_i w_i^2$$

Lambda λ is called the regularization coefficient (a constant parameter or hyperparameter)

– Effect : **weight decay** (basically add $2\lambda w$ to the gradient of the Loss)

$$\mathbf{w}_{\text{new}} = \mathbf{w} + eta * \Delta\mathbf{w} - 2\lambda\mathbf{w}$$

E.g. with zero gradient, it decreases the value of each w with a fraction of the old w.

EXERCIZE: derive it separating data term (• eta) and penalty term (• λ)

Andrej Nikolaevič Tikhonov

Russian mathematician
1906 –1993

Regularization by Ridge Regression: discussion

- General applicability (not only for polynomials)
  - e.g. we can control model complexity just using lambda, without not knowing M as for polynomials, or even when the model complexity is not known
- Note the balancing (trade-off) between the two terms:
  - Small error data term (minimize just the training error) is not enough for us*: we want also to control the complexity of the model to control the ***overfitting***, hence we introduce the second term in the minimization.
  - On the other side, we can exceed because to much weight to the second term (high lambda value) leads to focus the minimization only (or mostly) on the regularization, hence the data error (first term) could grow too much, i.e. moving to ***underfitting***
  - The trade-off is ruled by the value of ***lambda*** (λ)

Error data term + Regularization/penalty term

$$Loss(h_{\mathbf{w}}) = \sum_{p} (y_p - h_{\mathbf{w}}(\mathbf{x}_p))^2 + \lambda \, \|\mathbf{w}\|^2 \searrow \sum w_i^{\,2}$$

Lambda is called the regularization coefficient

Summuary of the exercises

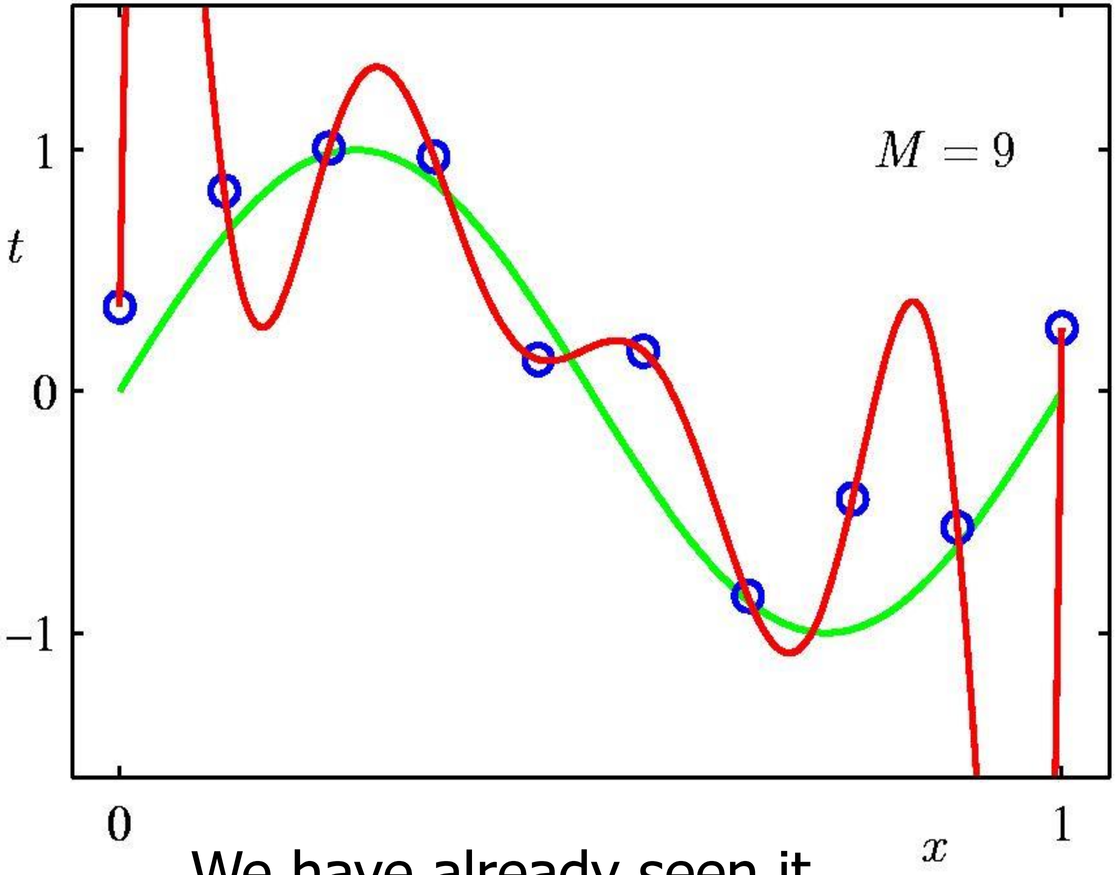
- Riflettere sui valori lambda alti o bassi
  - (in termini di complessità dei modelli, effetti su underfitting e overfitting, collegarlo poi a quando parleremo di SLT in una prossima lezione)
- Ricavare la regola di aggiornamento dei pesi con *loss di Tiknhonov*
  - calcolando di nuovo la derivate parziale di *L* rispetto ai w, ma separando per semplicità data term (da moltiplicare poi per eta) e penalty term (da moltiplicare poi per λ))

Facoltativo:

- Aggiungere il weight decay all'alg. in python

Repetita: 9th Order Polynomial

As using $\lambda = 0$



We have already seen it

$E(\boldsymbol{w}) = 0$ on training data!!! And **overfitting**

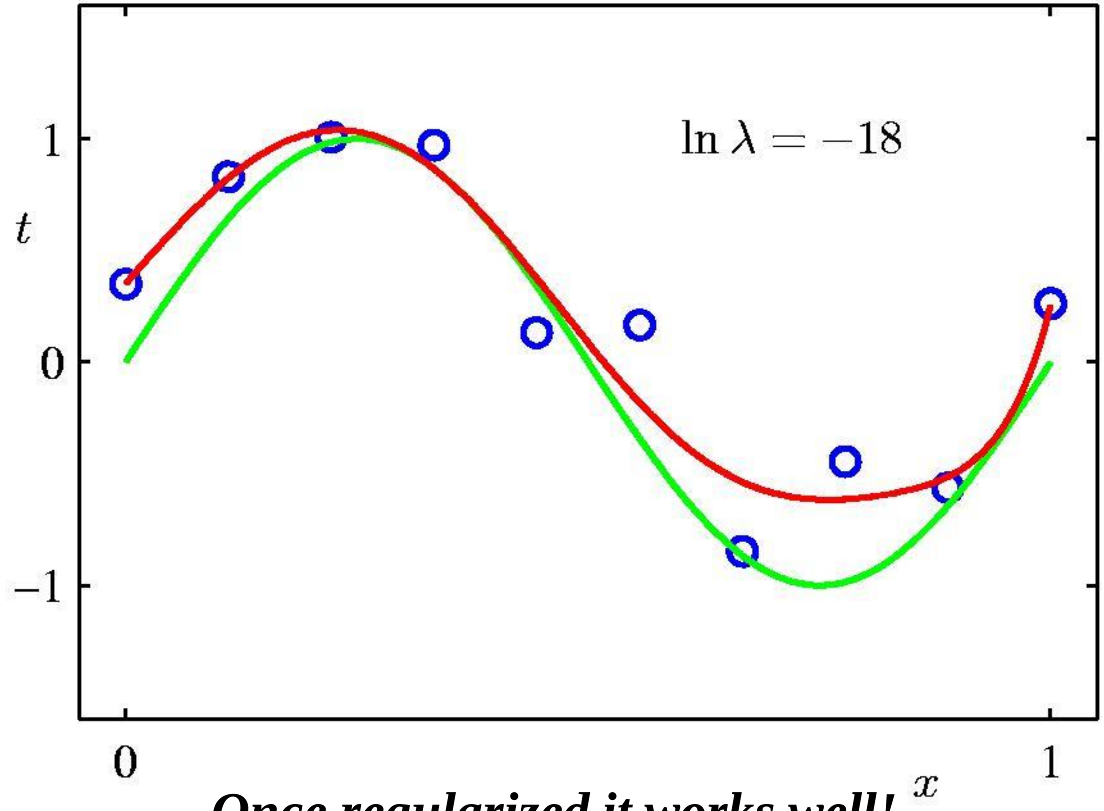
Regularization effect

$$\ln \lambda = -18$$

- 9th Order Polynomial

λ=0.0000000152

Suitable value of lambda



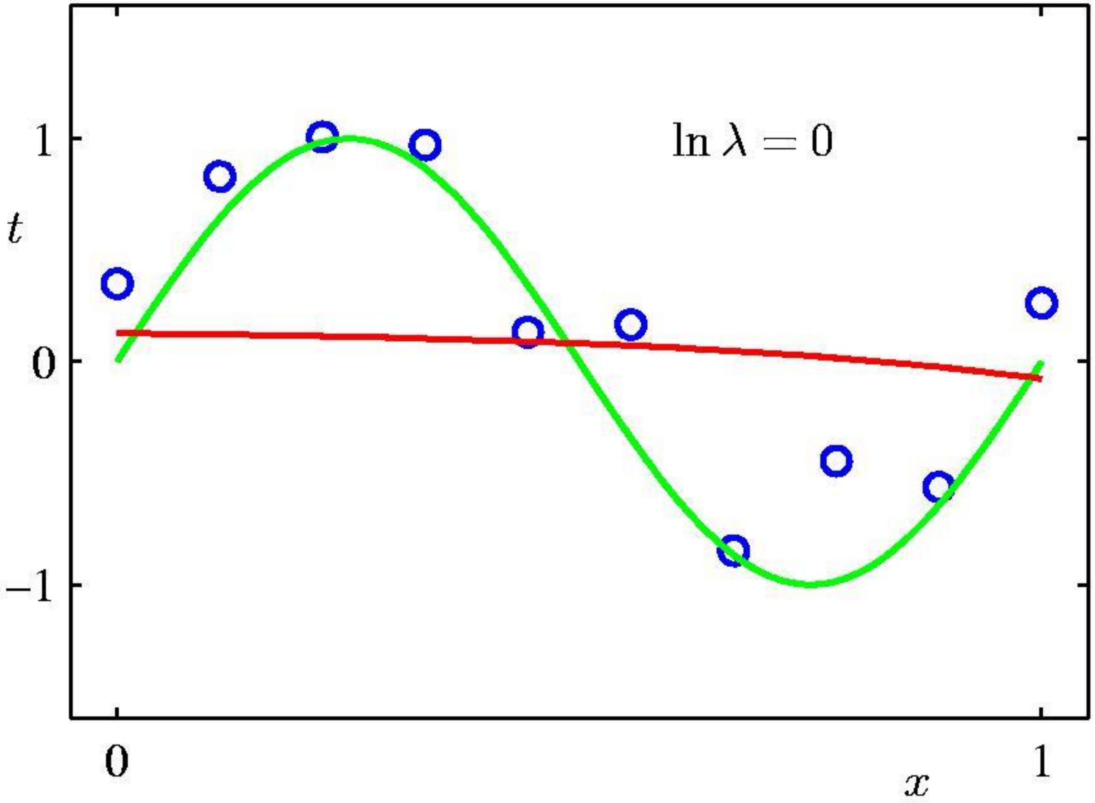
Once regularized it works well!

We avoid the overfitting

Too High lambda ($\ln \lambda = 0$)

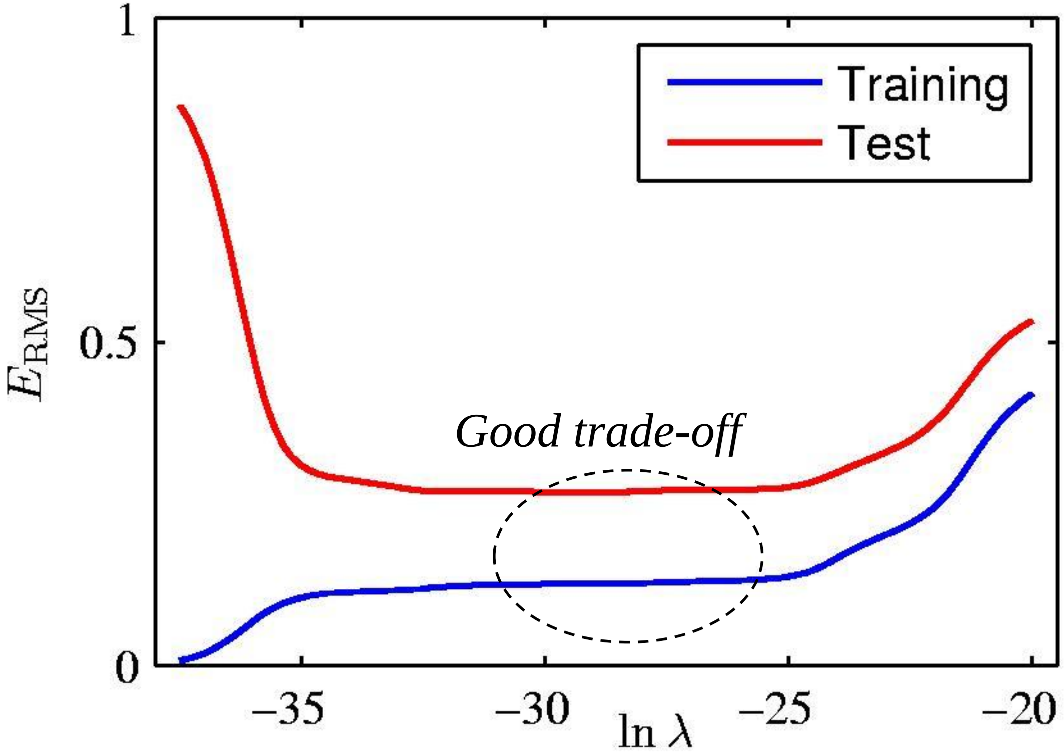
λ=1

Too high value of lambda



*Too rigid (**underfitting**)…**We need a trade-off** ….*

Regularization: E_{RMS} vs $\ln \lambda$



Low lambda → *overfitting*

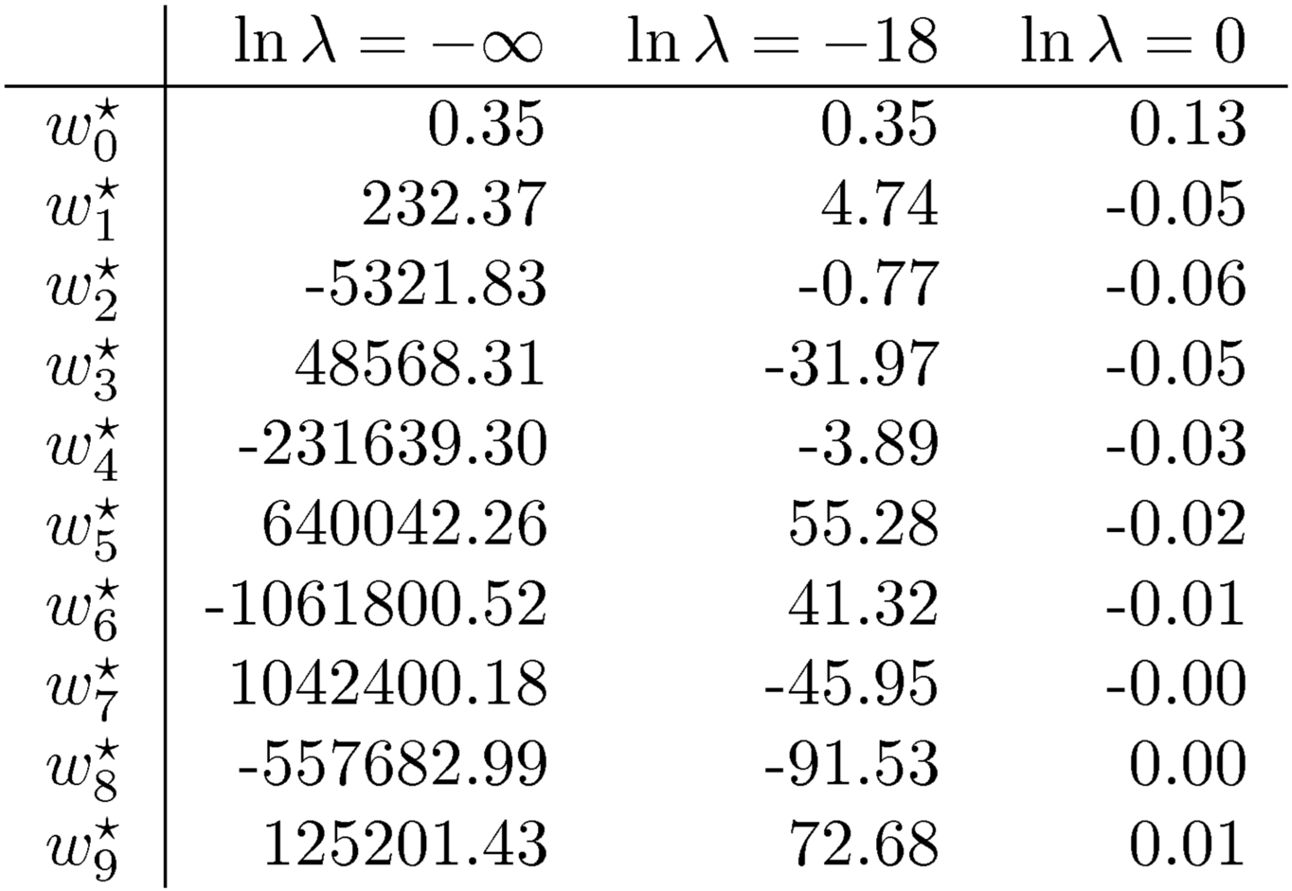
High lambda → *underfitting*

Polynomial Coefficients

0 lambda

λ=1

Very high lambda



	$\ln \lambda = -\infty$	$\ln \lambda = -18$	$\ln \lambda = 0$
$w_0^\star$	0.35	0.35	0.13
$w_1^\star$	232.37	4.74	-0.05
$w_2^\star$	-5321.83	-0.77	-0.06
$w_3^\star$	48568.31	-31.97	-0.05
$w_4^\star$	-231639.30	-3.89	-0.03
$w_5^\star$	640042.26	55.28	-0.02
$w_6^\star$	-1061800.52	41.32	-0.01
$w_7^\star$	1042400.18	-45.95	-0.00
$w_8^\star$	-557682.99	-91.53	0.00
$w_9^\star$	125201.43	72.68	0.01

Limitations of Fixed Basis Functions

BASE FISSATA

Dip. Informatica
University of Pisa

- Having basis function along each dimension of a D-dim. (K for LBE) input space requires combinatorial number of functions

  the curse of dimensionality.*

  - e.g. general polynomials of order 3 use all combination of input variables due to products $x_1x_2, x_2x_3,\dots,x_1x_2x_3$ etc. → D^3 (approximation as D increases)
- Data become sparse in this high volume → The amount of data needed to support the result often grows exponentially with the dimensionality.
- Phi are *fixed before* observing training data

❖ In other models, we shall see how we can get away with fewer basis functions, by choosing these using the training data: phi (in a hidden computational layer) depends on *w* and the model is not-linear in the parameters. E.g. ***Neural Networks*** #ML

❖ In other models the computation of the new embedding space (input transformation) is made implicitly through kernel functions and controlling the complexity of the model. E.g. ***SVM*** #ML and intro to SVM in the next lectures.

Linear model

Classification

We reuse the linear model

- The same models (used for regression) can be used for **classification**: categorical targets, **e.g. 0/1 or -1/+1.**

  - In this case we use a hyperplane (***wx***) assuming negative or positive values
  - We exploit such models to decide if a point ***x*** belong to positive or negative zone of the hyperplane (to classify it)
  - So we want to set ***w*** (by learning) s.t. we get good classification accuracy

Multidimensional input: Notation (almost repetita)

- Assuming column vector for $\boldsymbol{x}$ and $\boldsymbol{w}$ *(in bold)*

d_p or t_p $\quad p=1..l$

- Number of data l, dimension of input vector n, y_p ***(targets 0/1 or -1/+1)***

$$\mathbf{w}^T\mathbf{x} + w_0 = w_0 + w_1x_1 + w_2x_2 + \ldots + w_nx_n$$

NEW

- w_0 is the *intercept, threshold, bias, offset*.....

Often it is convenient to include the constant $x_0 = 1$ so that we can write it as :

$$\boldsymbol{w}^T\boldsymbol{x} = \boldsymbol{x}^T\boldsymbol{w}$$

Inner product

$$\mathbf{x}^T = [1, x_1, x_2, .., x_n]$$

$$\mathbf{w}^T = [w_0, w_1, w_2, .., w_n]$$

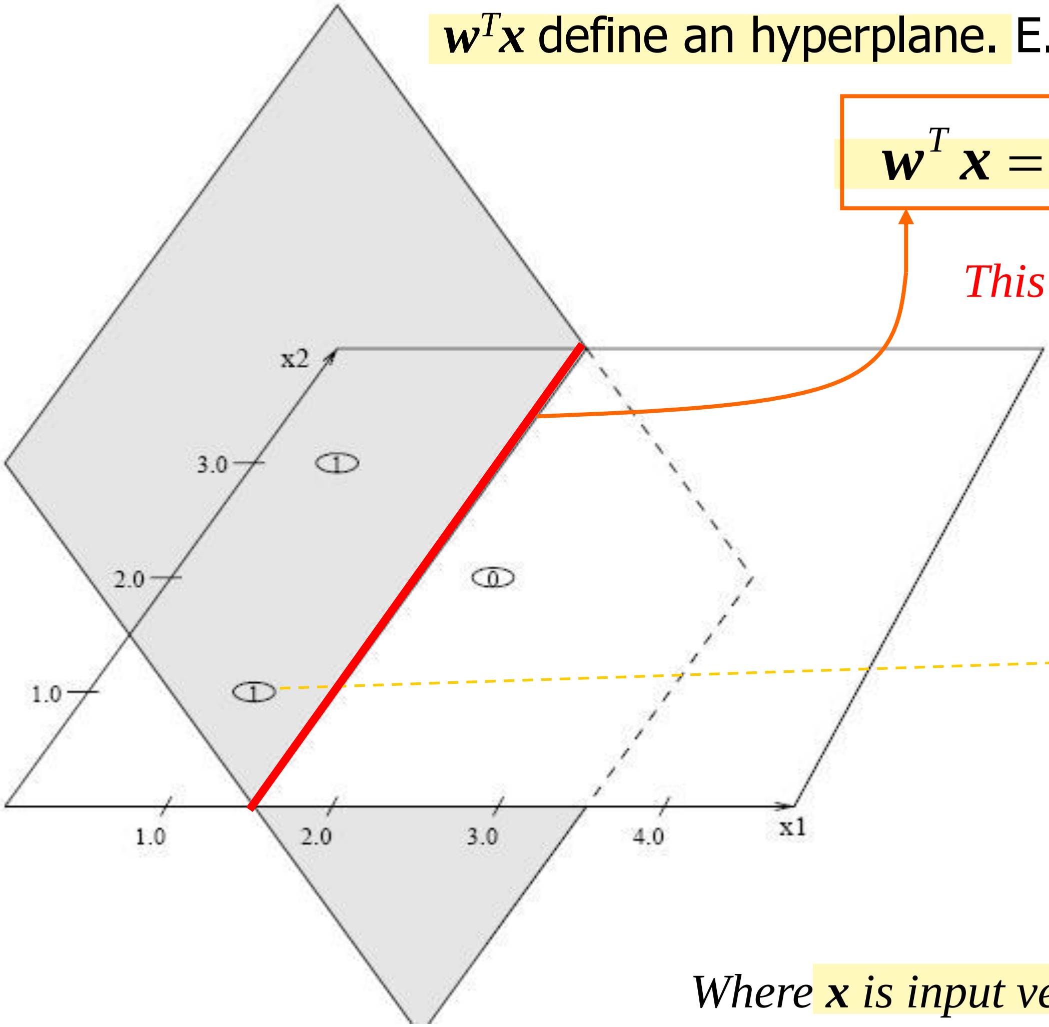
Note: ***w*** **continuous (free) parameters "weights"**

Geometrical view: hyperplane

$\mathbf{w}^T\mathbf{x}$ define an hyperplane. E.g. see the picture for 2 var.

$$\mathbf{w}^T\mathbf{x} = w_1x_1 + w_2x_2 + w_0 = 0$$

This is the decision boundary *Def*



Can be used to classify:

Examples <(x_1,x_2), y>:

<(1.0, 1.0), 1>

<(0.5, 3.0), 1>

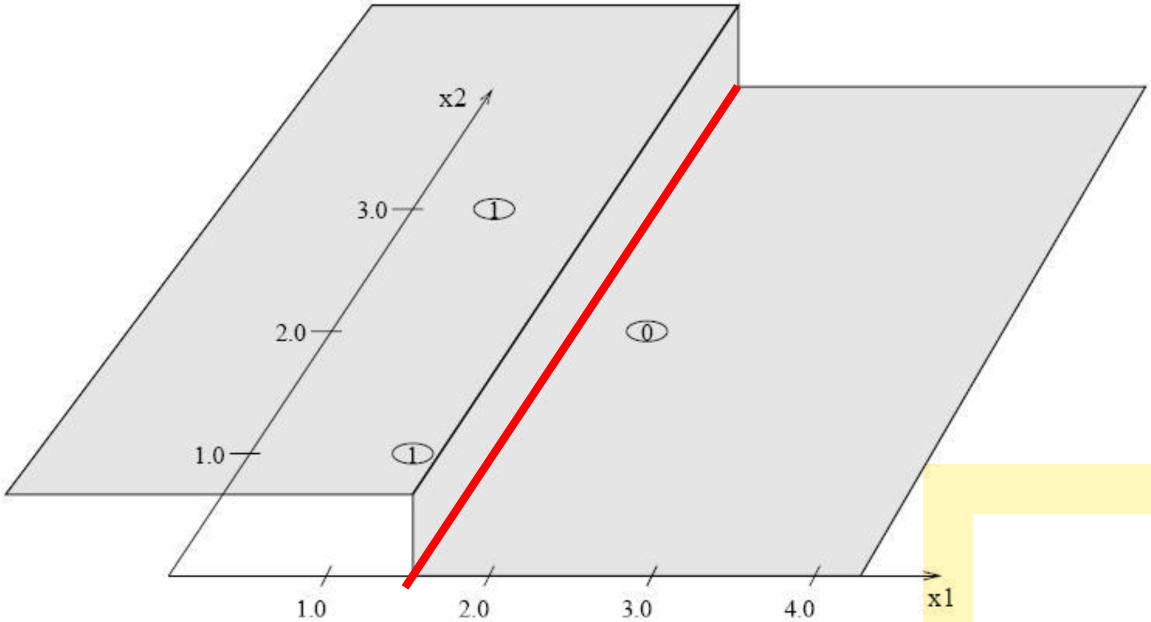
<(2.0, 2.0), 0>

Where ***x*** *is input vector,* ***w*** *free par. vector*

Geometrical view: classifier

Dip. Informatica
University of Pisa

Introducing a threshold function



Examples:

<(1.0, 1.0), 1>

<(0.5, 3.0), 1>

<(2.0, 2.0), 0>

[0,1] output range

$$h(\mathbf{x}) = \begin{cases} 1 & \_if \quad \mathbf{w}^T\mathbf{x} + w_0 \geq 0 \\ 0 & \_\_\_\_\_\_otherwise \end{cases}$$

or

[-1,+1] output range

$$h(\mathbf{x}) = sign(\mathbf{w}^T\mathbf{x} + w_0)$$

Using $\mathbf{x}_p$ *and including* w_0 *in* $\mathbf{w}$ ⇨

$$h(\mathbf{x}_p) = sign(\mathbf{x}_p{}^T\mathbf{w}) = sign\left(\sum_{i=0}^{n} x_{p,i} w_i\right)$$

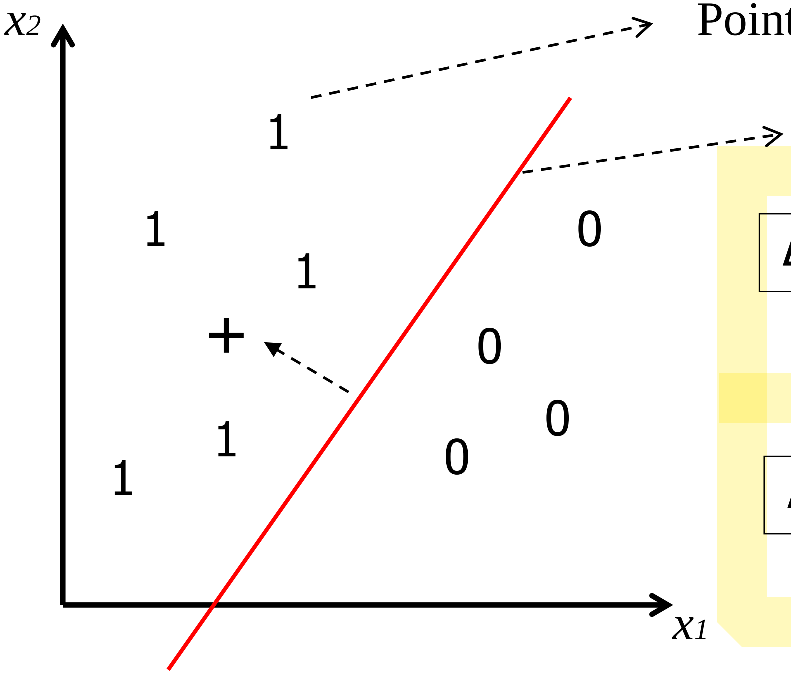
In this slide ***w*** *is omitted from* $h_{\mathbf{w}}$ *(we use just h)*

Classification by linear decision boundary

The classification may be viewed as the allocation of the input space in decision regions (e.g. **0/1**)

Example: linear separator on

2-dim instance space $\mathbf{x}=(x_1,x_2)$ in R^2 , $f(\mathbf{x})=0/1$ (or $-1/+1$)



Point belonging to class 1

Separating (hyper)plane : $\boldsymbol{x}$ s.t.

$$\boldsymbol{w}^T\boldsymbol{x} + w_0 = w_1x_1 + w_2x_2 + w_0 = 0$$

Def

$$h(\mathbf{x}) = \begin{cases} 1 & \_if \quad \mathbf{w}^T\mathbf{x} + w_0 \geq 0 \\ 0 & \_\_\_\_\_\_otherwise \end{cases}$$

[0,1] output range

or

Def

$$h(\mathbf{x}) = sign(\mathbf{w}^T\mathbf{x} + w_0)$$

[-1,+1] output range

Linear threshold unit (LTU)

How many? (H): set of dichotomies induced by hyperplanes

Threshold (bias w_0)

Note that, given the bias w_0, in the LTU

saying $h(\boldsymbol{x}) = \boldsymbol{w}^T\boldsymbol{x} + w_0 \geq 0$

is equivalent to say $h(\mathbf{x}) = \boldsymbol{w}^T\boldsymbol{x} \geq -w_0$

with $-w_0$ as the «threshold» value

- The two forms identify the same positive zone of the classifier
- The second one emphasizes the role of the bias as a threshold value to "activate" the +1 output of the classifier.

Use it: Example (AIMA)

Classify a new data (x_1,x_2)

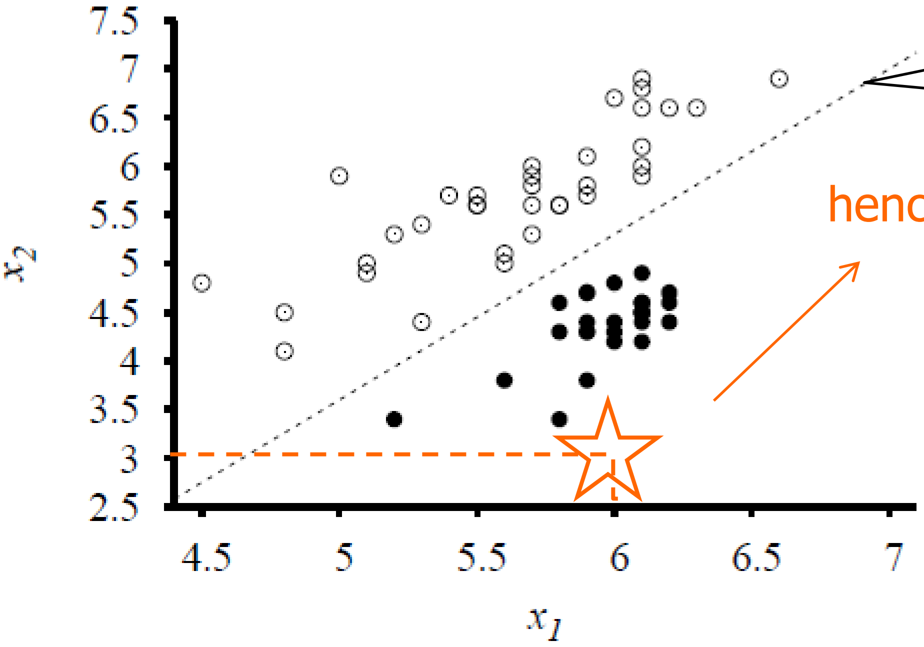
Find h s.t. given (x_1,x_2) return 0/-1 for *Earthquakes* and +1 for *Nuclear Explosion*

The learning alg. finds this decision boundary:

$$w_0 + w_1 x_1 + w_2 x_2 = 0$$

⇕ ⇕ ⇕

$$-4.9 + 1.7 x_1 - 1 x_2 = 0$$



hence **$h(6,3)$=*sign*$(w_0 + w_1*6 + w_2*3)$=**

=*sign*(- 4.9 + 1.7*6 -1*3)

=*sign*(2.3) =+1 ➔ *nuclear expl.*

Seismic data.

x_1 body wave magnitude, x_2 surface wave magnitude

Earthquakes (white) Nuclear Explosion (black)

1982-1990 Asia

Getting new examples is expensive;-)

Use it: Example (Spam)

- Find *h*(*mail*) +1 for *spam*, -1 *not-spam*
- Features ϕ(*mail*) = words [0/1] or phrases ("free money") [0/1] or length [integer]
- e.g $\phi_k(\mathbf{x}) = contain(word_k)$

"Bag of words" representation

- ***w*** → *weight* contribution of the input features to prediction
  - e.g. positive weight for "free money", negative for ".edu"
- $\mathbf{x}^T\mathbf{w}$ is the weighted combination
- $h_{\mathbf{w}}(x)$ provide the threshold to decide spam/not spam

$$h_{\mathbf{w}}(\mathbf{x}) = sign\left(\sum_k w_k \phi_k(\mathbf{x})\right)$$

>0 → + 1 = Spam !

The learning problem (for linear classifiers)

Def

- Assume we still use the *least squares*
- **Given** a set of *l* training examples
- **Find** $\mathbf{w}$ to *minimize* the residual sum of squares (LS):

To have the mean (LMS) divide by *l*

$$E(\mathbf{w}) = \sum_{p=1}^{l} (y_p - \underline{\mathbf{x}_p{}^T\mathbf{w}})^2 = \quad ||\mathbf{y} - \mathbf{X}\mathbf{w}||^2$$

Where $\mathbf{x}_p$ *is p-th input vector,* y_p *the output for p,* $\mathbf{w}$ *free par., l num. of examples, n input dim*

- Min error: if $y_p=1$ then $\mathbf{x}_p{}^T\mathbf{w} \rightarrow \mathbf{1}$; if $y_p=0/-1$ then $\mathbf{x}_p{}^T\mathbf{w} \rightarrow \mathbf{0/-1}$

- ***Note:*** *in E(***w***) we do* ***not*** *use the form h(***x***), as for regression, to hold a continuous differentiable loss (because h(***x***)=sign(***w***ᵀ***x***) for classification)!*

- We still have the **iterative gradient descent algorithm**:

Def

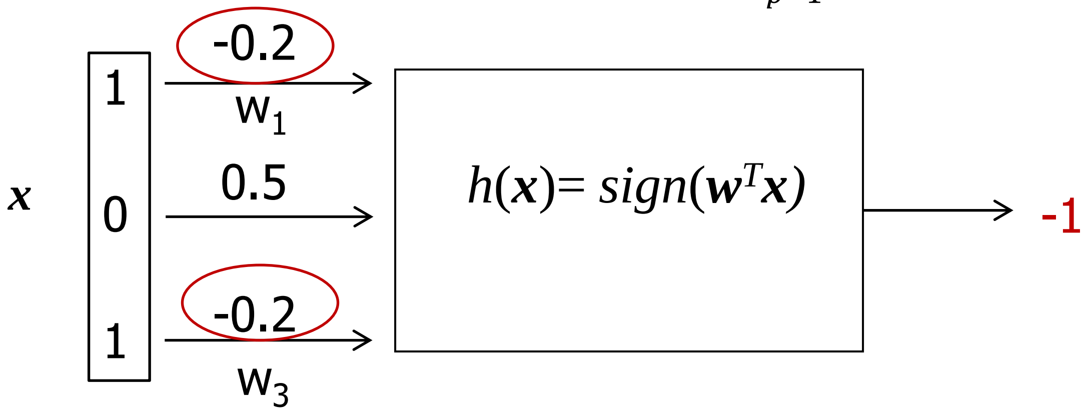
$$\Delta w_i = -\frac{\partial E(\mathbf{w})}{\partial w_i} = \sum_{p=1}^{l} (y_p - \mathbf{x}_p{}^T\mathbf{w}) \cdot x_{p,i} \quad i=0,..n$$

Exercise: Compute these Δw_i *, (2 is omitted)*

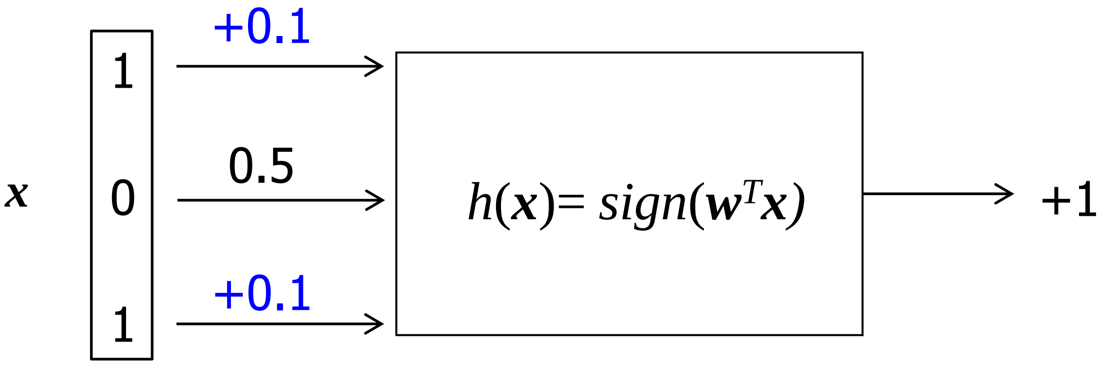
DeltaW as Error Correction Learning rule

Inputs Weights $(w_0=0)$

$$\Delta w_i = \sum_{p=1}^{l} (y_p - \boldsymbol{x}_p{}^T \boldsymbol{w}) \cdot x_{p,i}$$



If misclassified (because target is +1) → (1-(-???)) → High positive delta for w_1 and w_3 → increase them proportionally (with eta) to the *delta*, hence a positive value in this case [*error correction rule*]



e.g. (see figure):

Now is correct !!!

Exercise: compute the values ??? and how obtain the new weights (hence, also eta)!

Summarizing

- Model trained (on tr set) with LS (LMS) on ***wx***

  by the simple gradient descent algorithm used for linear regression

- Model used for classification applying the threshold function, obtaining: $h(\mathbf{x}) = \mathrm{sign}(\mathbf{w}^T\mathbf{x})$

- The error can be computed as *classification error* or number of misclassified patterns (not only by the Mean Square Error)

$$L(h(\mathbf{x}_p), d_p) = \begin{cases} 0 & \_if \quad h(\mathbf{x}_p) = d_p \\ 1 & \_\_\_\_\_\_\_\_ otherwise \end{cases}$$

$$mean\_err = \frac{1}{l} \underbrace{\sum_{p=1}^{l} L(h(\boldsymbol{x}_p), d_p)}_{num\_err}$$

Def

- ACCURACY = mean of correctly classified = $(l\text{-}num\_err)/l$

Learning Algorithm

- The same as for regression: see “Gradient descent algorithm”
- Also the linear basis expansion and Tikhonov can be applied as well

- Note that (learning algorithm):
  - It converges asymptotically to the MinLS
  - Not necessarily to the min number of classification errors

- [(#ML) If $\boldsymbol{w}$ are not changed for correct outputs ($h_{\mathbf{w}}(\mathbf{x}_p)=d_p$) we obtain the update rule for the *perceptron*]

Example: Conjunctions

- We can represent conjunctions by the linear models, e.g.:
- **Conjunctions:**

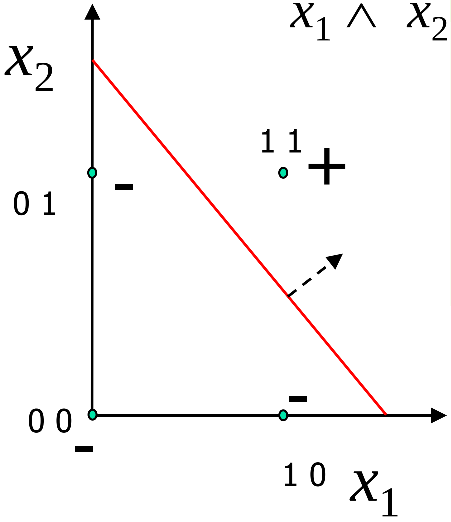
$$h(\mathbf{x}) = \begin{cases} 1 & \_if \quad \mathbf{w}^T\mathbf{x} + w_0 \geq 0 \\ 0 & \_\_\_\_\_\_otherwise \end{cases}$$

4 var: $x_1 \wedge x_2 \wedge x_4 \leftarrow\!\rightarrow y$

- $1\ x_1 + 1\ x_2 + 0\ x_3 + 1\ x_4 > 2$
- $1\ x_1 + 1\ x_2 + 0\ x_3 + 1\ x_4 \geq 2.5$

2 var: $x_1 \wedge x_2 \leftarrow\!\rightarrow y$

- $1\ x_1 + 1\ x_2 > 1$
- $1\ x_1 + 1\ x_2 \geq 1.5$



$x_1 \wedge x_2$	AND
00	0
01	0
10	0
11	1

Exercise:

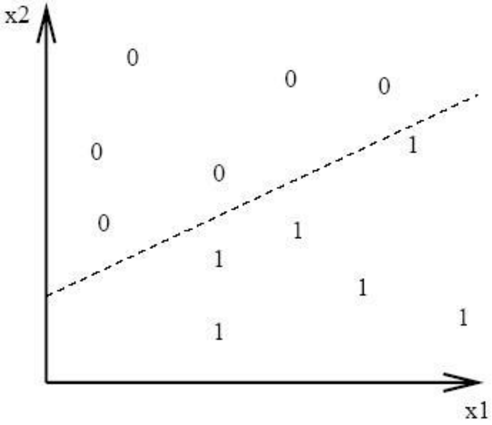
- Can you do ***or***, ***not*** ? How?
- Can you do all the possible logic function? [difficult]

In general, **w** can be learned to find this solution

Limitations

Def

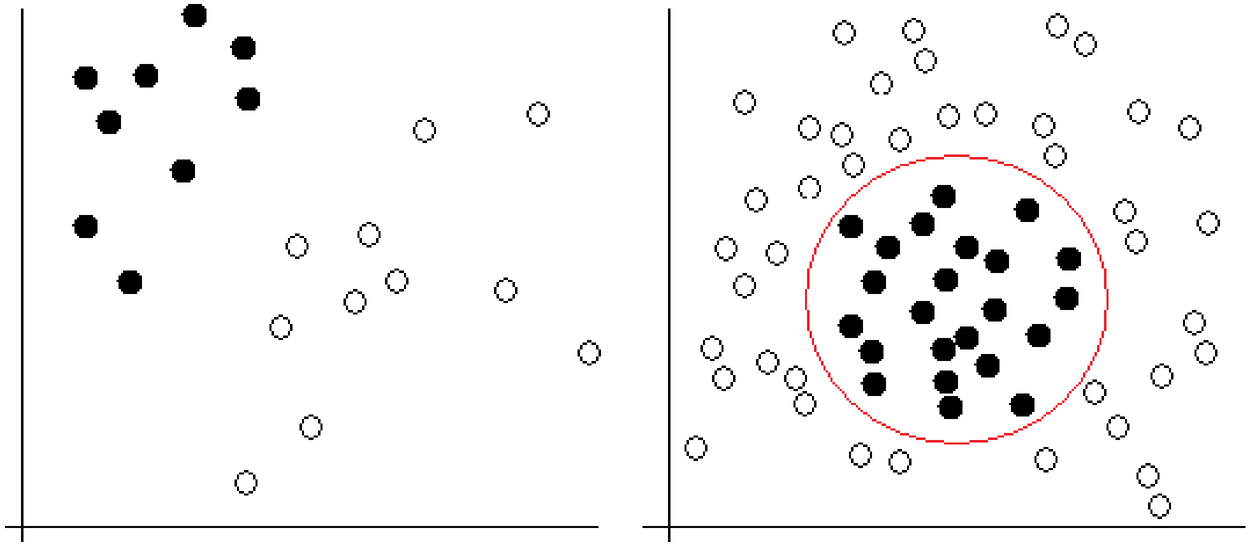
- In geometry, two set of points in a two-dimensional plot are ***linearly separable*** when the two sets of points can be completely separated by a single line
- In general, two groups are *linearly separable* in *n*-dimensional space if they can be separated by an (*n* -1)-dimensional hyperplane.



- The linear decision boundary can provide exact solutions only for linearly separable sets of points

Linear versus Non-Linearly Separable

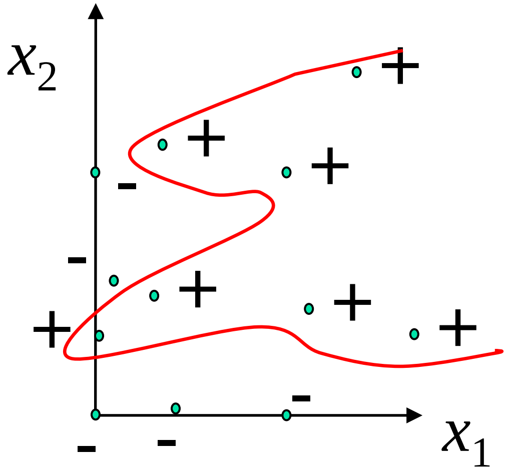
- Exercise: which one is the non-linearly separable case?



Non linear decision boundary: Examples

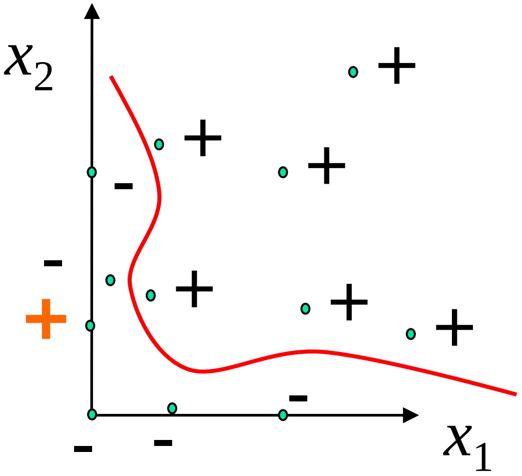
Repetita:

- Also the **linear basis expansion** and **Tikhonov regularization** can be applied as well



***Non linear** decison boundary*
*(by a **basis expansion**)*

*We obtain a **not-smooth** classiffier that may need regularization*



***Non linear** decison boundary*
*(by a **basis expansion**) **regularized**.*

*In this example 1 training error is admitted (**orange**)*

Multi-class tasks (facoltativo)

Two very simple approaches for multi-class:

- Class 1-of-K rep: {red, green, blue} → (0,0,1), (0,1,0), (1,0,0). → solve tree linear models
- **OVA:** "one-vs-all": a discriminat function for each class

built on top of real-valued binary classifiers

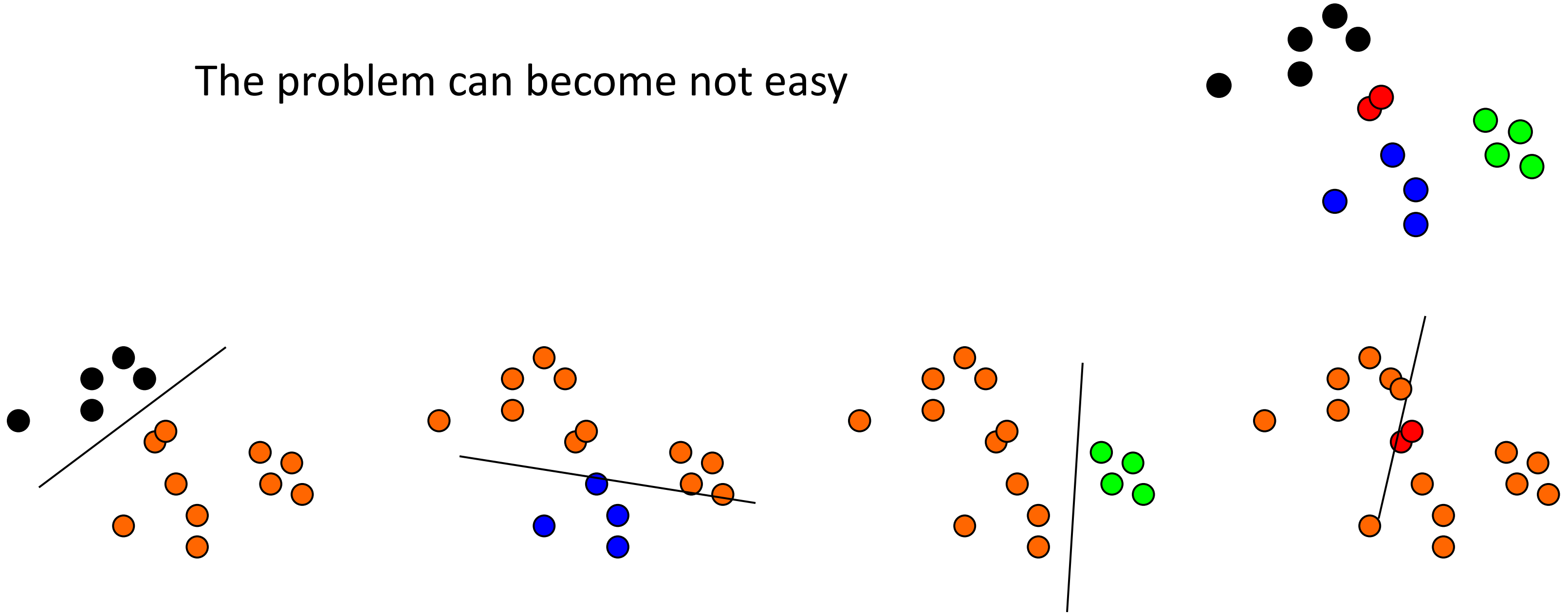
- train K different binary classifiers, each one trained to distinguish the examples in a single class from the examples in all remaining classes.
- to classify a new example, the K classifiers are run, and the classifier which outputs the largest (most positive) value is chosen.

- AVA: "all-vs-all" = "one-versus-one":
  - each classifier separates a pair of classes. Apply it to all the pairs: K(K-1)

    [How many training?*]
  - to classify a new example, all the classifiers are run, and the winner is the one with the max sum of outputs versus all the other classes **OR** the class with most votes
  - training data set for each classifier is much smaller

Criticism (facoltativo)

The problem can become not easy



- Masking: classes can be masked by others (for high K)
- A wide array of more sophisticated approaches for multiclass classification exists …
- Some models can deal directly with multi-output

Other learner models for classification

- Linear Discriminant Analysis (also multi-class)

- Logistic regression
  - *P(y|x)* starting from modeling the class density as a know density
  - Soft threshold (continuous, differentiable) with logistic function (instead of 0/1 hard threshold) [#ML]

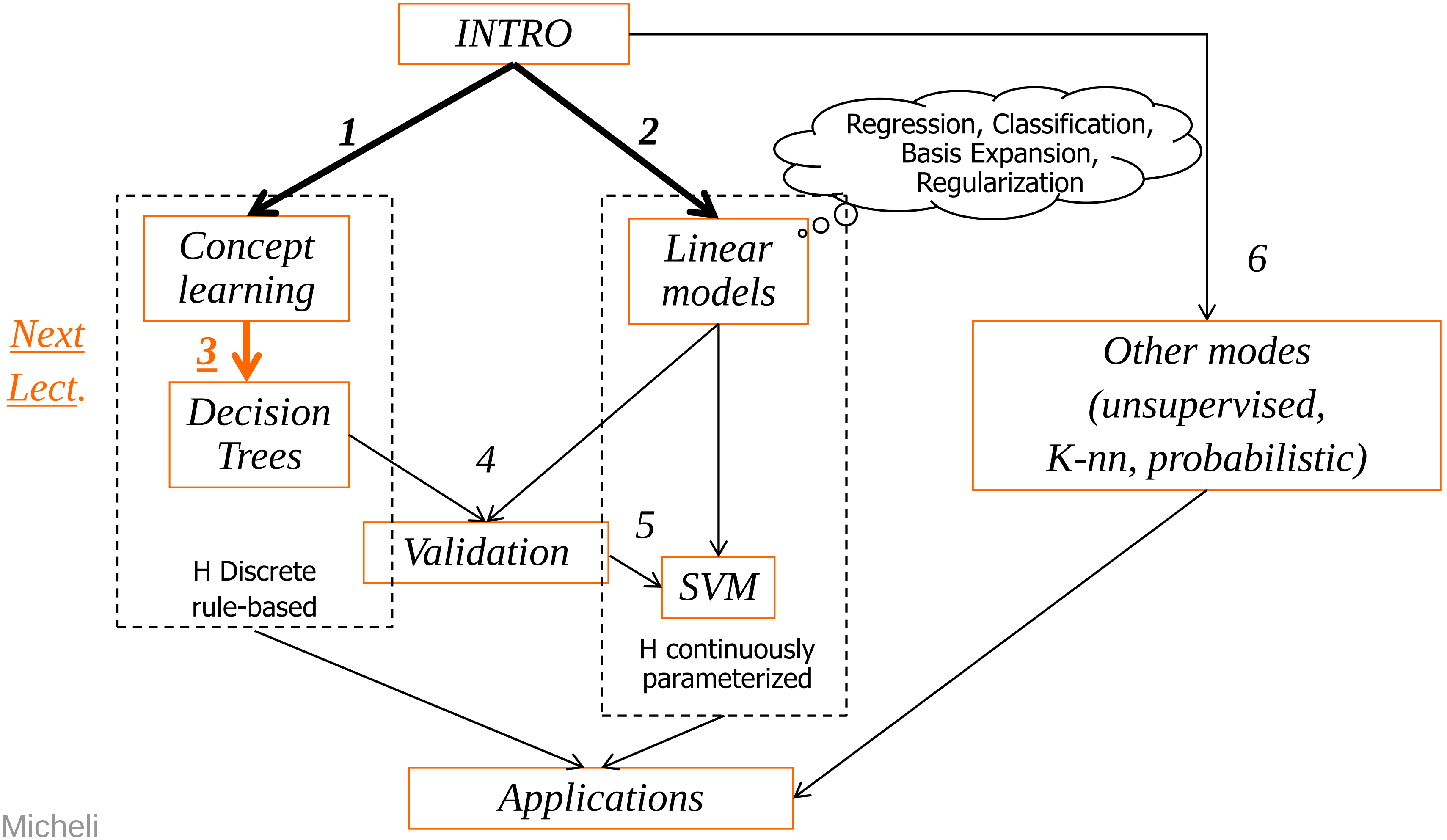
- Neural networks (NN) and SVM mentioned before: **flexible models** including non-linear approximation for both regression and classification. [#ML]:
  - NN use many units (similar to LTU) within different layers
  - Feature representation learning in each layer (deep learning concept)
  - Gradient descent approach for learning

Conclusions on Linear Models

- Basic well-founded approach for both **regression** and **classification**
  - Very compact way to represent knowledge
  - But with a strong assumption on the relationship among data
- An iterative error correction (**LMS**) algorithm that search continuous hypothesis spaces
  - (that is the basis for many other ML approaches)
- A view of limitations of linear approaches and of the needs for more **flexible** ML models and their issues:
  - An extension of linear model for non-linear tasks
  - An introduction to the **control of complexity (regularization)**

In the course flow

In the course: example for *numerical eq*. language (H continuous space), now we return to *symbolic rules* language (by *DT*) [see 2 parallel tracks]


INTRO
1
2
Regression, Classification, Basis Expansion, Regularization
Concept learning
3
Next Lect.
Decision Trees
Linear models
6
Other modes (unsupervised, K-nn, probabilistic)
4
Validation
5
SVM
H Discrete rule-based
H continuously parameterized
Applications

Bibliography (last 2 lectures)

- AIMA , ed .3: **chap 18.6** (18.6.1,18.6.2,18.6.3)

- Further readings (not mandatory!):
  - Mitchell: Linear model and LMS alg.: chap 4.4
  - To go in deep for **Linear least squares**

    You can start from www resource as (With nice examples):

    https://en.wikipedia.org/wiki/Linear_least_squares_%28mathematics%29
  - **LS in general**

    http://en.wikipedia.org/wiki/Least_squares
  - **Polynomial curve fitting** (phi) and discussion: Bishop book in chapter 1

For information

Alessio Micheli

micheli@di.unipi.it

www.di.unipi.it/groups/ciml

IN SUPREMÆ DIGNITATIS
1343

Dipartimento di Informatica
Università di Pisa - Italy

Computational Intelligence &
Machine Learning Group