# VC-DIMENSION

- measure of complexity of the model (degree of freedom of the model) (flexibility to fit data)
- by VC-dimension es. we can measure # of parameter of polynomial
- We can use $R_{emp}$ to approximate the R (general rule) => yes, considering $R_{emp}$ and VC-confidence

=> VC-BOUND: holds with probability $1-\delta$ that:

$$R \leq R_{emp} + \varepsilon\left(\frac{1}{l}, VC, \frac{1}{\delta}\right)$$

(VC confidence)

=> it's a guaranteed risk: measure this quantity can measure the upperbound

=> we never measure R ~> because you need to know the entire distribution of the data

- $\varepsilon$ is a function that grows with VC-dim
  ($VC \uparrow \Rightarrow \varepsilon \uparrow$)
  ($l \uparrow ; \delta \uparrow \Rightarrow \varepsilon \downarrow$)

$\delta$ is the confidence
=> rules the bound holds (PROB)
↳ rule the confidence

- $R_{emp}$ decreases using complex models => high VC-dimension

Q: if $\delta = 0{,}01$ => bound hold with Prob 0,99 => prob the bound is TRUE

=> NB: we can upbound the generalization error R with prob $1-\delta$

where ingredients are: $R_{emp}$ (can measure)
VC-dim (you can control, es. # data)
=> correlation btw R and $R_{emp}$

- if ↑ # data can still have an high VC-dim but still have a good control

because ↑ l, ↑ VC => with $\frac{1}{l} \cdot VC$

=> increase l => increase VC
=> ↑ l
=> higher VC

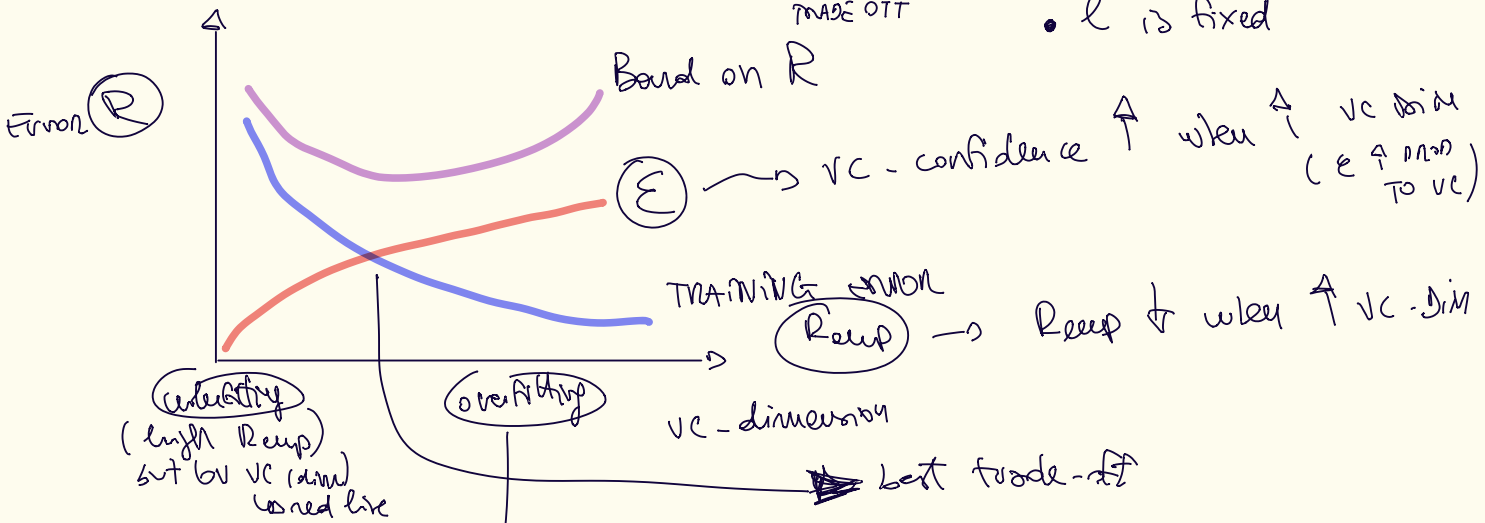
- if for 1 point, take it 100 times
  => the bound => P of taking 100 times (randomly) the same point is very low and belongs to $1-\delta$ zone
  (to cut-off strange cases)

(NB): fixing $\delta$ we can explain underfitting/overfitting and how control them (when they happens on theoretical POV)

=>. higher $l$ (# data) => lower VC confidence and BOUND close to R

- too simple model (low VC-dimension) can be not sufficient
  => due to high $R_{emp}$ => UNDERFITTING
- higher VC-dimension (fix $l$) => lower $R_{emp}$
  => but VC-confidence & R may increase
  => OVERFITTING

- STRUCTURAL RISK MINIMIZATION: way to minimize the bound
  => find the TRADE OFF
- $l$ is fixed




ε -> VC-confidence ↑ when ↑ VC dim (ε proportional to VC)

$R_{emp}$ -> $R_{emp}$ ↓ when ↑ VC-dim

Blue ⊕ Red : take the sum => purple => U-shape

low $R_{emp}$
VC-dim ↑, VC-confidence ↑, red line ↑ => overfitting

$$R_{emp} = \frac{1}{l} \sum_{i=1}^{l} L(h(x_i), y_i)$$

# LEARNING ALGO 4 Reg./Class using linear model (based on LMS) (CLASSIF.

learning problem: Find $w$ s.t. minimize the loss on Training data

$$R_{emp} = \frac{1}{\ell} \sum_{p=1}^{\ell} L\left(h_w(x_p), y_p\right)$$

→ Assume use least square (as in regression

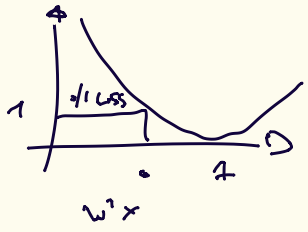
a) Replace ORIGINAL OBJECTIVE function $L(0/1 \text{ loss})$ by a smooth differentiable one => MSE (LOSS)

- $h(x) = 1$ if $w^T x > 0$

⇓

minimum error is for $w^T x > 0$

=> hence no classification error




Same as regression $E(w) = \sum_{p=1}^{\ell} \left(y_p - x_p^T w\right)^2 = \|y - Xw\|^2$

MATRIX of DATA

(find $w$ to minimize sum of square)

here we CANNOT use $h(x)$ inside because $h(x) = \text{sign}(w^T x + w_0)$

=> discrete function with a threshold (not differentiable

if target is positive, we are going to $w^T x$ close to 1

=>

we do the derivative for $w = 0, \dots, n$ (no just $w_0$ and $w_1$)

imposing gradient = 0

$$\frac{\partial E(w)}{\partial w_j} = \frac{\partial \sum_{p=1}^{\ell} \left(y_p - x_p^T w\right)^2}{\partial w_j} = \sum_{p=1}^{\ell} 2\left(y_p - x_p^T w\right) \cdot \frac{\partial \left(y_p - x_p^T w\right)}{\partial w_j} =$$

$$= \sum_{p=1}^{\ell} 2\left(y_p - x_p^T w\right) \left( \frac{\partial y_p}{\partial w_j}^{0} - \frac{\partial (x_p)_1 w_1}{\partial w_j}^{0} - \frac{\partial (x_p)_2 w_2}{\partial w_j}^{0} - \dots - \frac{\partial (x_p)_j w_j}{\partial w_j}^{1} - \dots \right) =$$

$$= -2 \sum_{p=1}^{\ell} \left(y_p - x_p^T w\right) (x_p)_j = -\sum_{p=1}^{\ell} 2\, \delta_p\, x_{p,j}$$

Linear ALGO FOR CLASSIFICATION: $E(w) = \sum_{p=1}^{l} (y_p - x_p^T w)^2$

$$\frac{\partial E(w)}{\partial w_j} = 2 \sum_{p=1}^{l} (y_p - x_p^T w) \cdot \frac{\partial (y_p - x_p^T w)}{\partial w_j} =$$

$$= 2 \sum_{p=1}^{l} (y_p - x_p^T w) \cdot \left( \frac{\partial y_p}{\partial w_j} - \frac{\partial (x_p)_1}{\partial w_j} w_1 - \frac{\partial (x_p)_2}{\partial w_j} w_2 - \dots - \frac{\partial (x_p)_j}{\partial w_j} w_j \dots \right)$$

$$= -2 \sum_{p=1}^{l} (y_p - x_p^T w)(x_p)_j = \boxed{- \sum_{p=1}^{l} 2 \delta_p x_{p,j}}$$

$\Rightarrow$ USE NOW **GRADIENT DESCENT**

- la derivazione di prima consente un ALGORITMO ITERATIVO

$$\frac{\partial E(w)}{\partial w_j} = -2 \sum_{p=1}^{l} y_p x_{p,j} = -2 \sum_{p=1}^{l} (y_p - x_p^T w)(x_p)_j$$

- GRADIENT = ASCENT DIRECTION: ne con move verso il minimo con il

GRADIENT DESCENT $= -\frac{\partial E(w)}{\partial w} = \Delta_w$

$\Rightarrow$ LOCAL SEARCH: - inizializzo il vettore dei pesi
- itero modificandolo fino a decrescere per minimizzare la error function

- GRADIENT vector (for each component)

$$\Delta_w = -\frac{\partial E(w)}{\partial w} = \begin{bmatrix} -\frac{\partial E(w)}{\partial w_1} \\ -\frac{\partial E(w)}{\partial w_2} \\ \dots \\ -\frac{\partial E(w)}{\partial w_j} \end{bmatrix} = \begin{bmatrix} \Delta w_1 \\ \Delta w_2 \\ \dots \\ \Delta w_m \end{bmatrix}$$

- in multi-dim space $w_0$ is omitted (we not visualize it)

- AS iterative approach: we LEARNING rule on DELTA changing w proportionally to the (opposite) of GRAD current

$$w_{new} = w + eta \cdot \Delta w$$

the learning rate

A simple algorithm:

1) Start with weight vector $w_{initial}$ (small), fix *eta* ($0<eta<1$).
2) Compute $\Delta w = -$"gradient of $E(w)$" $= -\frac{\partial E(w)}{\partial w}$ (or for each $w_j$)
3) Compute $w_{new} = w + eta*\Delta w$ (or for each $w_j$)

Repeat (2) until convergence or $E(w)$ is "sufficiently small"

- $\Delta w/l$ : least mean squares (dividing by $l$, that will be the standard case)
- Batch versions ($\Delta w$ after each "epoch" of $l$ training patterns)
- *eta* ($\eta$): step size = *learning rate*: speed/stability trade-off: can be (gradually) decreased to zero (guarantee convergence, avoiding oscillation around the min.): many variants will be introduced later



- BATCH: gradient is the sum over all the $l$ patterns

$$\frac{\partial E_w}{\partial w_j} = -2\sum_{p:1}^{l}(y_p - x_p^T w)\, x_{pj}$$

Ultimate weight after the sum

- ONLINE/STOCHASTIC
  - upgrade weights with the error computed for each pattern

$$\frac{\partial E_p(w)}{\partial w_j} = -2(y_p - x_p^T w)\, x_{pj} = -\Delta_p w_j$$

- GRADIENT DESCEND CORRESPOND TO ERROR CORRECTION RULE (DELTA RULE)
  - change $w_j$ proportionally to the error (as (target − output))
- (target $y$ − output) = 0 => NO ERROR => NO CHANGES
- if input is positive (input$_j$ > 0) and ERROR > 0 => OUTPUT IS TOO LOW ($\Delta$ positive) => increase $w_j$ => ↑ output => less error
- if input is > 0 and error < 0 => output too high (negative $\Delta$) => decrease $w_j$ => reduce output => less error

$$\Delta w_j = 2\sum_{p=1}^{l}(x_p)_j\,(y_p - x_p^T w)$$

$$w_{new} = w + eta \cdot \Delta w$$

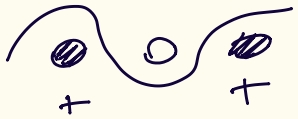
ACCURACY = media of correct classified $\frac{(l - NUM\_ERRORS)}{l}$

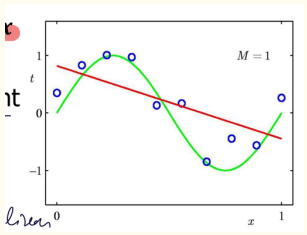
$$L(h(x_p), d_p) = \begin{cases} 0 & \text{if } h(x_p) = d \\ 1 & \text{else} \end{cases}$$

$$mean\text{-}error = \frac{1}{l}\sum_{p=1}^{l} L(h(x_p), d_p)$$

# INDUCTIVE BIAS

- **Language bias**: the $H$ is a set of linear functions (too rigid and restrictive) → vary the
- **SEARCH BIAS**: ordered the search attraverso least square → minimization goal

es.  => need a non-linear model




=> COME ESTENDERE UN MODELLO LINEARE

- in $h_w(x) = w_1 x + w_0$ or $h_w(x) = w^T x$

**linear** is the **way** in which the **coefficient** $w$ occurs in the **regression equation** (not red line)

=> we can USE TRANSFORMED INPUTS, $x, x^2, x^3, x^4$ and alg doesn't change

CON RELAZIONI NON LINEARI TRA input e output

with LMS:

$$h_w(x) = w_0 + w_1 x_1 + w_2 x_2 + \cdots + w_M x^M = \sum_{j=0}^{M} w_j x^j$$

=> POLYNOMIAL REGRESSOR

=> Call this **LINEAR BASIS EXPANSION**

$$h_w(x) = \sum_{k=0}^{K} w_k \phi_k(x)$$

- Riscrivo il modello lineare come somma di trasformazioni del vettore originale $x$ tramite la funzione $\phi$ di ogni componente pesata $w_k$

- aggiungo all'input vector con l'aggiunta di variabili che sono la trasformazione del vettore $x$ in accordo alla funzione $\phi$ ($\phi_k : R^M \to R$)

es. polinomial representation: $x: \phi(x) = x_j^2$

or

$\phi(x) = x_j x_i$

non-linear transformation of simple inputs: $\phi(x) = \log(x_j)$ $\phi(x) = \sqrt{x_j}$

- NUMERO di PARAMETRI $K > M$ (before was $M$)
- $\phi$ are fixed before seeing the TR
- CURSE OF DIM
- Il MODELLO è lineare nei parametri (anche in $\phi$, non in $x$)

(+): Posso MODELLARE RELAZIONI più COMPLICATE (NON LINEARI)

(-) with LARGE BASIS of FUNCTION RISK OVERFITTING => NEED CONTROL COMPLEXITY

=> CONTROL COMPLEXITY

Lo polinomio con GRADO ALTO

1) Ridge regression / Tikonov REGULARIZATION:

- smoothed model
- aggiungo dei vincoli alla somma dei valori $|w_j|$ penalizzando modelli con alti valori di $|w|$ => less COMPLEX model

↳ norm of W

over data term [(M)SE]

$$loss(w) = \sum_{p=1}^{l} (y_p - x_p^T w)^2 + \lambda \|w\|^2$$

- $\sum_{p=1}^{l} (y_p - x_p^T w)^2$ → E data term
- $\lambda \|w\|^2$ → regularization / penalty term
- $\|w\|^2 \rightarrow \sum w_j^2$
- Control the size of the weight
- $\lambda$ chosen in model selection (regularization hyperparameter)

↳ più la OBJECTIVE function si chiama LOSS (USATA per stimare LA funzione COSTO per allenare il modello)

- mentre E (error) è utile per CALCOLARE IL MODEL error e usata nel DATA TERM DELLA LOSS

GRADIENT APPROACH:

=> new rule $w_{new} = w + eta \cdot \Delta w \boxed{-2\lambda w}$

=> WEIGHT DECAY: add $2\lambda w$ al gradiente

$\lambda = 0$ Good FITTING

=> Min Remp (overfitting

=> increase VC-dim

$$loss(w) = \sum_{p=1}^{l} (y_p - x_p^T w) + \lambda \|w\|^2$$

=> gradient => $w_{new} = w + eta \cdot \Delta_w - 2\lambda w$

## Tikhonov regularization: trade-off

Dip. Informatica University of Pisa

$$Loss(w) = \sum_{p=1}^{l} (y_p - x_p^T w)^2 + \lambda \|w\|^2$$ (error E(w))

- Note the balancing (trade-off) between the two terms:
  - Small ***lambda*** ($\lambda$) value → minimizing the loss the focus is on obtaining a small error data term (first term, minimize just the training error) with a too complex model (high norm of the weights), the risk is of ***overfitting***,
  - High ***lambda*** ($\lambda$) → minimizing the loss the focus is on the second term, 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*** ($\lambda$)
  - We will see soon some examples

& controllare la complessità ha è un LANGUAGE BIAS (control Hypothesis space) BUT SEARCH BIAS perché è nella LOSS

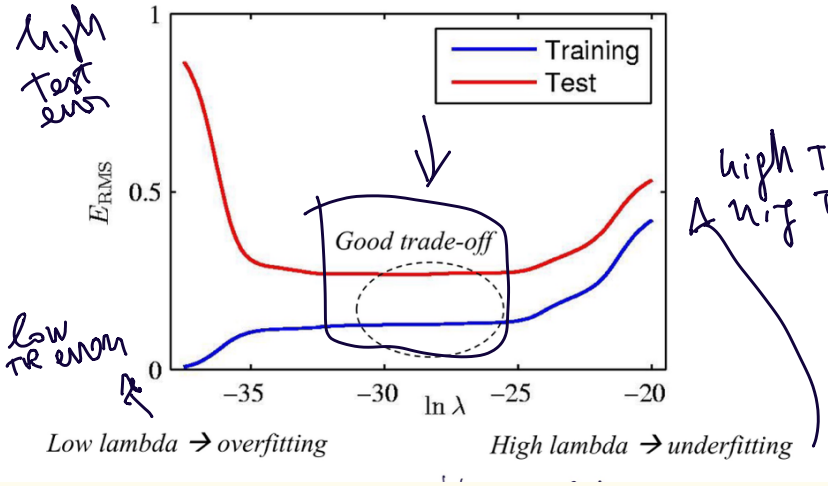
## Tikhonov and SLT

Dip. Informatica University of Pisa

- The penalty term penalizes high value of the weights and tends to drive all the weights to smaller values
  - E.g. Some weights values can go even to zero
- It implements a control of the model complexity
- This leads to a model with less (or proper) *VC-Dim* (with a trade-off obtained through **just a (1) parameter** that you can control: the $\lambda$)

↳ mi serve lambda e non decido il grado del polinomio in VALIDATION perché è più generico e posso applicarlo non solo al polinomio

=> con 1 PARAMETRO CONTROLLO LA COMPLESSITÀ




high test error
low TR error
high TR error & high test error } underfitting

$L_2$: ridge regression
$L_1$: lasso

The *L2 norm* penalizes the square value of the weight and tends to drive all the weights to smaller values.

On the other hand, the *L1 norm* penalizes the absolute value of the weights and tends to drive some weights to exactly zero (while allowing some weights to be large) → toward feature selection!

- Unfortunately $\| \ \|_1$ (absolute value) introduce a non differentiable loss → needs other approaches

---

**K-NN**: k-nearest neighbor

LAZY (before EAGER)

- STORE TR data
- wait until a data point is presented
- construct ad-hoc hypothesis to classify that data point

ALGORITHM:

1) Store TR data $\langle x_p, y_p \rangle \quad p = 1, \dots, l$
2) given an input x with dimension n
3) Find the nearest training example $x_i$
   => find i s.t. have minimum $d(x, x_p)$
   => $i(x) = \arg\min_p d(x, x_p)$
   => euclidean distance

$$d(x, x_p) = \sqrt{\sum_{t=0}^{n} (x_t - x_{pt})^2} = \| x - x_p \|$$

4) then output x

with $k = 1$ NN
can solve limitation of linear model

- 1-nn return + for $x_q$
- 5-nn return – for $x_q$

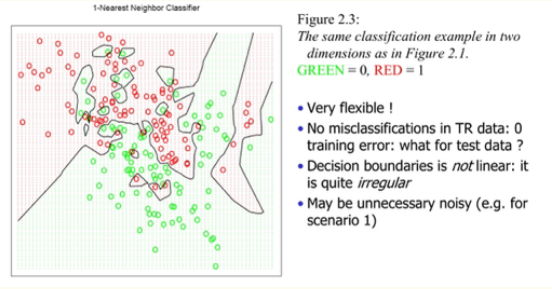



Figure 2.3:
*The same classification example in two dimensions as in Figure 2.1.*
GREEN = 0, RED = 1

- Very flexible !
- No misclassifications in TR data: 0 training error: what for test data ?
- Decision boundaries is *not* linear: it is quite *irregular*
- May be unnecessary noisy (e.g. for scenario 1)

# K-Nearest Neighbors

Dip. Informatica
University of Pisa

- A natural way to classify a new point is to have a look at its neighbors,
- and take an average:

$$avg_k(\boldsymbol{x}) = 1/k \sum_{\boldsymbol{x}_i \in N_k(\boldsymbol{x})} y_i$$

- where $N_k(\boldsymbol{x})$ is a neighborhood of $\boldsymbol{x}$ that contains exactly $k$ neighbors (closest patterns according to *distance d*):
  - k-nearest neighborhood: **K-nn**.
- If there is a clear dominance of one of the classes in the neighborhood of an observation $\boldsymbol{x}$, then it is likely that the observation itself would belong to that class, too. Thus the classification rule is the **majority voting** among the members of $N_k(\boldsymbol{x})$. As before,

$$h(\boldsymbol{x}) = \begin{cases} 1 & \text{if } avg_k(\boldsymbol{x}) > 0.5 \\ 0 & \text{otherwise} \end{cases} \quad \text{for targets } y \text{ in } \{0,1\}$$

- For regression task: use directly the *avg*: mean over K-nn

$$avg_k(x) = \frac{1}{k} \sum_{x_i \in N_k(x)} y_i$$

↑ neighborhood

KNN FOR MULTI-CLASS (NOT INFORMATIVE)

$$h(x) = \arg\max_v \sum_{x_i \in N_k(x)} 1_{v, y_i}$$

$$1_{v, y_i} = \begin{cases} 1 & \text{if } v = y_i \\ 0 & \text{else} \end{cases}$$

↓ - count cbns of neighborhood

look curse of dim.

---

NN

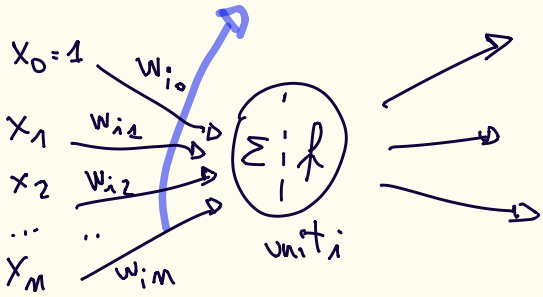
ARTIFICIAL NEURON: PROCESS UNIT

- node, neuron, unit
- Input: external source or other units
- Input connections: weights w: free-parameters

$$\begin{cases} net_i(x) = \sum_j w_{ij} x_j \\ o_i(x) = f(net_i(x)) \end{cases}$$

~~weighted sum~~ is called net input to unit i

net for the unit i (Σ of the input signals)

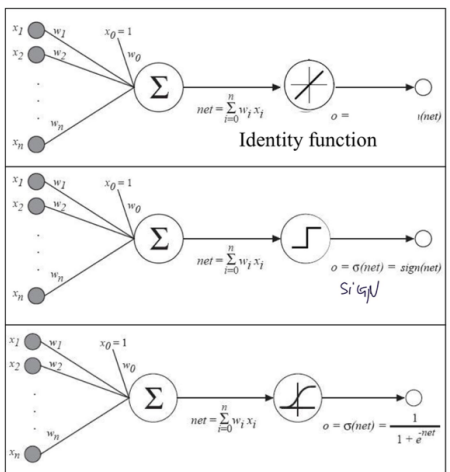


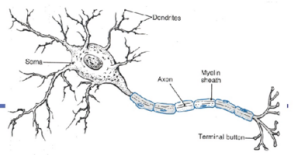

$w_{ij}$ is the weight of the unit i coming from j unit to i

(f) is the activation function: linear, LTU, ...

PERCEPTRON: • biologically inspired model
• with a simple computational unit
• can be composed and connected to build network: FROM LTU to MLP
• a neuron i get active when Σ connections $w_{ij}$ coming from unit j connected to it are active, PLUS a BIAS, is > 0
• BINARY CLASSIFICATION MODEL

## Neuron: Three activation functions




$$h(\boldsymbol{x}) = \sum_i w_i x_i$$

Linear activation function

⇒ classification

**Perceptron** (see LTU)
Threshold activation function

1 perceptron cannot solve the XOR => MLP yes: with 1 hidden layer

problem becomes linear separable with 1 hidden layer

Re-represent the input space in a way that make easier the task

- Useful concept of internal **(re)representation** of input variables via internal (hidden) units
- Developing *high level (hidden) features* is a key factor in NN
- The representation in the hidden layer makes easier the task to the output layer (last unit).
  - E.g. See the figure on the right: point *(1,0)* (in the sud-est corner) became $h_1=0\ h_2=1$ (north-west corner) → now it is a linear separable problem

A look ahead:

- Such ***composition*** of sub-/intermediate operations to perform a complex task can be extended through *many layers* of abstraction
- In NN such internal representation/intermediate features can be ***learned***
- Learning internal distributed representation → *representation learning* and *deep learning* concepts (link to next parts of the ML course)

A. Micheli 23

• 1 hidden layer to implement all possible function (1 level just not)

↓ due to single perceptron linear sep. problem → solved

## LEARNING FOR 1 UNIT MODEL:

1) Adaline: adaptive linear neuron (what I know): correspond to LMS direct solution (regression)
   and LTU and LMS for classification
   - gradient descend solution
   - AN APPROACH THAT WE will GENERALIZE TO MLP
   - regression
   - classification

2) PERCEPTRON:
   - non-linear model: but includes THRESHOLD ACTIVATION FUNCTION
   - ONLY FOR CLASSIFICATION
   - CONVERGENCE THEOREM

- perceptron learning algorithm: minimize # misclassified patterns:
  find w s.t. $sign(w^T x) = d$ (or TK)
- online algorithm: a step made for each input pattern

1. Initialize the weights ($w_{0,1}$) (either to zero or to a small random value)
2. pick a learning rate $\eta$ (leo) (this is a number between 0 and 1) => $\eta \in [0,1]$
3. until stopping condition is satisfied (e.g. weights don't change):

For each training pattern $(x, d)$ $(d = +1 \text{ or } -1)$: x = input pattern vector, d = desired output

- Compute output activation $out = sign(w^T x)$ [+1,-1] (omitting $^T$)
- If $out = d$, don't change weights (I.e. "minimize (only) misclassifications")
- If $out \neq d$, update the weights:
  - $w_{new} = w + \eta\ d\ x$

add $+\eta x$ if $wx \leq 0$ and $d=+1$
$-\eta x$ if $wx > 0$ and $d=-1$

else $sign(w^T x) \Leftrightarrow w_{new} = w + \eta x$ (d=

$$w_{new} = w + \eta\ d\ x$$

Hebbian learning on

$$w_{new} = w + \eta\ (d - out)\ x = w + \delta x$$

$\delta$ = error signal

for more convenient rule

↓

CHANGE w TO PROPORTIONALLY TO the ERROR

- a PERCEPTRON represent a LINEAR DECISION BOUNDARY (LIV)
- PERCEPTRON can only SOLVE LINEAR SEPARATION PROBLEM
- a PERCEPTRON always LEARN a SOLUTION (on what is able to represent) => PERCEPTRON CONVERGENCE THEOREM

PERCEPTRON CONVERGENCE THEOREM:

- IL PERCEPTRON GARANTISCE LA CONVERGENZA (classifica correttamente tutti gli INPUT PATTERNS) IN UN NUMERO FINITO DI STEPS IF THE PROBLEM IS LINEAR SEPARABLE (posso ottenere diverse soluzioni (first solution is not unique)

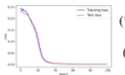
perceptron learning ALGO vs LMS

$$w_{new} = w + \eta\ (d\text{-}out)\ x$$

- Apparently similar but note that:
  - LMS rule was derived (by the gradient) without threshold activation functions: minimization of the error of the linear unit (using directly $wx$)
  - The perceptron use $out = sign(w^T x)$
- Hence, for training:
  - $\delta = (d - w^T x)$ for the LMS approach
    versus
  - $\delta = (d - \text{sign}(w^T x))$ for the perceptron learning alg.
- Of course the model trained with LMS can still be used for classification applying the threshold function $h(x) = sign(w^T x)$ (LTU)

**Differences Summary**

| Perceptron Learn. Alg. | LMS Alg. |
|---|---|
| • Min. misclassifications ($out = sign(w^T x)$) | • Min. $E(w)$ with $out = w^T x$ |
| • it always converges for linear separable problems (in a finite number of steps) to a perfect classifier | • asymptotic convergence (*), also for not linear separable problems |
| • (else no convergence) | • not always zero classification errors, linear separable problems |
| • difficult to be extend to networks of units (NN) | • it can be extended to networks of units (NN), using the gradient based approche |



(*) example of *asymptotic* convergence (training with MSE loss)

41

Proof: perceptron convergence theorem

- focus on: POSITIVE PATTERN TASK (classification)

- assume: $(x_i, d_i)$ in the TR with $d_i = +1$ || $-1$ (input pattern)

oss: LINEAR SEPARABLE means => $\exists\ w^*$ solution s.t. (desired output of pattern $i$)

$$d_i (w^* x_i) \geq \alpha = \min_i d_i (w^* x_i) > 0$$

if the product is positive => $\exists$ min of pos value in the TR that is positive

if PATTERN is positive, hyperplane is positive the product is positive

is always positive

=> hence $w^* (d_i x_i) \geq \alpha$

- defining $x_i' = (d_i x_i)$ then $w^*$ is a solution iff $w^*$ is a solution of $(x_i', +1)$ (all positive pattern TASK) of original problem

(if) $w^*$ solve ALL TASK $\to d_i (w^* x_i) \geq \alpha \Rightarrow (w^* d_i x_i) \geq \alpha$

$\Rightarrow (w^* x_i') \geq \alpha \Rightarrow w^*$ is a solution of $(x_i', +1)$

only if: $w^*$ is a solution of $(x_i', +1) \Rightarrow (w^* d_i x_i \geq \alpha) \Rightarrow d_i (w^* x_i) \geq \alpha$ rewrite

$\Rightarrow w^*$ solve for $x_i$

transformed problem

assuming: • $w(0) = 0$ (at step 0) => WAT step 0 (updated weight is zero)

• $\eta = 1$

• $\beta = \max |x_i|^2$ → euclidean norm of the vector patterns in the TR $\|\|_2$

– after $\varphi$ errors (all FALSE NEGATIVES)

$$w(\varphi) = \sum_{j=1 \to \varphi} x_{(i_j)}$$

$i_j$ patterns ∈ misclassified patterns

→ sum up x thru the updating of weights (only missclass.)

because $w(j) = w(j-1) + x_{i_j}$ ↑

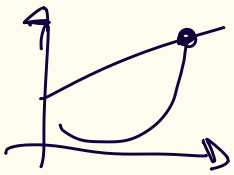
• $i_j$ is sub-index to misclass PATT.

(Rule: $w_{new} = w + \eta\, d\, x$) but all d are +1, only positive patterns, and $\eta = 1$

↳ $d = 1$ (all pos.); $\eta = 1$

BASIC IDEA: can find lower and upper bound to $|w|^2$ as a function of $\varphi^2$ steps (lower b) and $\varphi$ steps (upper)

=> can find $\varphi$ (# steps) s.t. algo converges →



(PT 1)

proof: lower bound on $|w(\varphi)|^2 = w \cdot w(\varphi) = w \sum_{j=1 \to \varphi} x_{i_j} \geq \varphi\alpha$

Recall that $\alpha = \min_i (w^* x_i)$

Cauchy Swartz: $(wv)^2 \leq |w|^2 |v|^2$ where $|w|^2 = \|w\|_2^2$

$$|w^*|^2 |w(\varphi)|^2 \geq (w \cdot w(\varphi))^2 \geq (\varphi\alpha)^2$$

$$\Rightarrow |w(\varphi)| \geq \frac{(\varphi\alpha)^2}{|w^*|^2}$$

(PT 2): upper bound on $|w(\varphi)|^2$

$|a+b|^2 = \sum_i (a_i + b_i)^2 = \sum_i a_i^2 + \sum_i 2a_i b_i + \sum_i b_i^2 = |a|^2 + 2ab + |b|^2$

$$|w(\varphi)|^2 = |w(\varphi-1) + x_{i_\varphi}|^2 = |w(\varphi-1)|^2 + 2w(\varphi-1)x_{i_\varphi} + |x_{i_\varphi}|^2$$

For $\varphi$-th error this is $< 0$

$$\hookrightarrow |w(\varphi)|^2 \leq |w(\varphi-1)|^2 + |x_{i_\varphi}|^2$$

and by iteration

$$(w(0) = 0) \Rightarrow |w(\varphi)|^2 \leq \sum_{j=1 \to \varphi} |x_{i_j}|^2 \leq \alpha\beta$$

$\beta = \max_i |x_i|^2$

(PT 3): upper & lower

$$\alpha\beta \geq |w(\varphi)|^2 \geq \frac{(\varphi\alpha)^2}{|w^*|^2} \qquad \varphi\beta \geq \varphi^2\alpha^2 \qquad \beta \geq \alpha^2 \Rightarrow$$

the upper Bound

$$\boxed{\varphi \leq \frac{\beta}{\alpha^2}}$$

convergence theorem: th. P. converge (class correctly with 'i' pattern) in a finite # steps if prob. lin. separable

- given $(x_i, d_i)$ in the TR with $d_i = 1 \,||\, -1$

linear separable $\Rightarrow \exists\, w^*$ solution s.t. $d_i(w^* x_i) \geq \alpha = \min_i d_i(w^* x_i) > 0$

hence $w^*(d_i x_i) \geq \alpha$

defining $x_i' = (d_i x_i)$ then $w^*$ is a solution iff $w^*$ is a solution of $(x_i', 1)$

$\Rightarrow$ if $w^*$ solve the task $\Rightarrow d_i(w^* x_i) \geq \alpha \Rightarrow (w^* d_i x_i) \geq \alpha \Rightarrow$

$\rightarrow (w^* x_i') \geq \alpha \Rightarrow w^*$ is a solution of $(x_i', +1)$

only if $w^*$ is a solution of $(x_i', +1) \Rightarrow (w^* d_i x_i) \geq \alpha \Rightarrow d_i(w^* x_i) \geq \alpha$

$\Rightarrow w^*$ solve for $x_i$

- assume: $w(0) = 0$

$\eta = 1$

$\beta = \max_i |x_i|^2$

AFTER $q$ errors $w(q) = \sum_{j=1 \to q} x_{i_j}$ ← index of misclass pnt.

update rule: $w_{new} = w_{old} + \eta\, d\, x$ (with $\eta = 1$, $d = 1$)

$\Rightarrow w(j) = w(j-1) + x_{i_j}$

- can find lower & upper bound to $|w|^2$ as a function of $q^2$ steps and $q$ steps (upper bound)

(Cauchy-Schwartz)

$\Rightarrow$ can find $q$ (# steps) s.t. elp converges

$\alpha = \min_i (w^* x_i)$

- lower bound $|w(q)|^2$: $w^* w(q) = w^* \sum_{j=1 \to q} x_{i_j} \geq q\alpha$

Cauchy-Schwartz

$|w^*|^2 |w(q)|^2 \geq$

$\geq (w^* w(q))^2 \geq (q\alpha)^2$

lower bound

$$|w(q)|^2 \geq \frac{(q\alpha)^2}{|w^*|^2}$$

UPPER BOUND

$$|w(q)|^2 = |w(q-1) + x_{iq}|^2 = |w(q-1)|^2 + 2w(q-1)x_{iq} + |x_{iq}|^2$$

for the $q^{th}$ error is $< 0$

$$|w(q)|^2 \le |w(q-1)|^2 + |x_{iq}|^2$$

by iteration $w(0) = 0$

$\beta = \max_i |x_i|^2$

$$|w(q)|^2 \le \sum_{j=1-Dq} |x_{ij}|^2 \le q\beta$$

Upper Bound

UPPER B

$$q\beta \ge |w(q)|^2 \ge \frac{(q\alpha)^2}{|w^*|^2}$$ Lower

$$\alpha' = \frac{\alpha^2}{|w^*|^2}$$

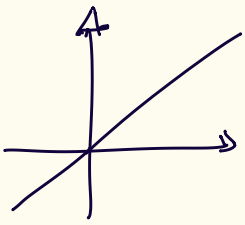
$$q\beta \ge q^2\alpha'$$

$$\beta \ge q\alpha'$$

$$q \ge \frac{\beta}{\alpha'}$$

# ACTIVATION FUNCTIONS for neurons

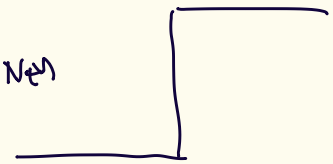
1) **Linear**: $\begin{cases} net_i = \sum_j w_{ij} x_j \\ o_i = f(net_i(x)) \end{cases}$ $\quad f(x) = \sum w_i x_i$



⇒ DERIVATIVE IS 1

2) **THRESHOLD** (perceptron/LTU):

$\begin{cases} net_i = \sum w_{ij} x_j \\ o_i = SIGN(net_i(x)) \end{cases}$ ⇒ DERIVATIVE ? UNDEFINED (only is not used)

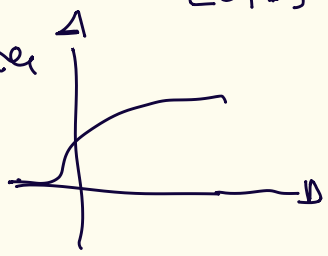


3) **LOGISTIC / SIGMOID**: NON-LINEAR SQUASHING FUNCTION: assumes values continuously in $[0,1]$

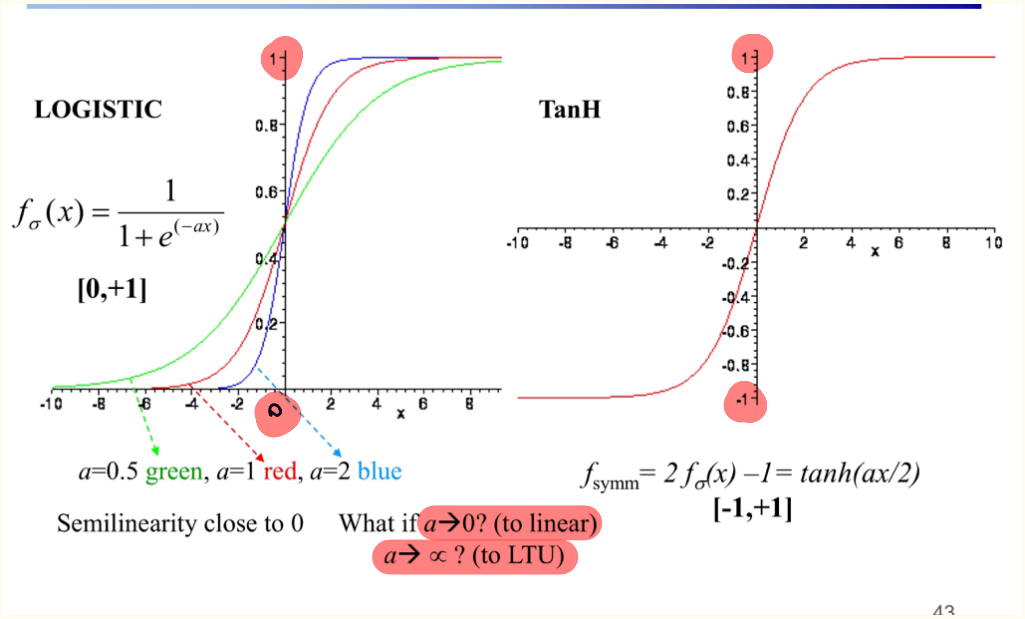
a) Logistic

$$f_\sigma(x) = \frac{1}{1+e^{(-ax)}}$$

a - slope parameter



- PROPERTY: SMOOTHED DIFFERENTIABLE THRESHOLD FUNCTION




Derivative logistic

a = 1

- $\frac{\partial f_\sigma(x)}{\partial x} = f_\sigma(x)(1 - f_\sigma(x))$

derivative tanh

- $\frac{\partial f_{TANH}(x)}{\partial x} = 1 - f_{TANH}(x)^2$

- Rectifier →**ReLU** (Rectified Linear Unit): see *Deep Learning* section

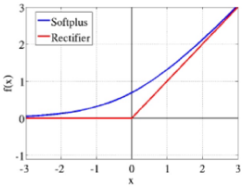
It becames a default choice for Deep models, so it (and its variants) deserves more attention discussing Deep Learning

$$f(x) = \begin{cases} 0 & for \quad x < 0 \\ x & for \quad x \geq 0 \end{cases}$$

$$f(x) = \max(0, x)$$

- **Softplus** (smooth approximation)

$$f(x) = \ln(1 + e^x)$$




- An updated list (Note: many perform comparably!):
https://en.wikipedia.org/wiki/Activation_function

# LMS with sigmoid $f_\sigma$

- Sigmoid è smoothed differentiable threshold function
- => can derive LMS algorithm => computing GRADIENT OF mean square LOSS FUNCTION
  - from $o(x) = x^T w$ to $o(x) = f_\sigma(x^T w)$

=> FIND w TO MINIMIZE the RESIDUAL SUM of SQUARES

$$E(w) = \sum_{p=1}^{l} \left(d_p - o_n(x_p)\right)^2 = \sum_{p=1}^{l} \left(d_p - f_\sigma(x_p^T w)\right)^2$$

COMPUTE THE GRADIENT

- start derivative of $E_p$ WRT $w_j$

$$\frac{\partial E_p}{\partial w_j} = \frac{\partial E_p}{\partial net} \cdot \frac{\partial net}{\partial w_j} =$$

$$\frac{\partial net(x_p)}{\partial w_j} = \frac{\partial \sigma(x_{p,i} w_i)}{\partial w_j} \Rightarrow \frac{\partial x_{p,j} w_j}{\partial w_j} = (x_p)_j$$

$$\frac{\partial E_p}{\partial net} = \frac{\partial E_p}{\partial out} \cdot \frac{\partial out}{\partial net} \rightarrow \frac{\partial f(net)}{\partial net} = f'_{net}$$

$$\frac{\partial \left(d_p - o_{out_p}\right)^2}{\partial out_p} = 2\left(d_p - out_p\right) \cdot \frac{\partial \left(d_p - out_p\right)}{\partial out_p} = -1$$

$$= -2\left(d_p - out_p\right)$$

$$= -2\left(d_p - out_p\right) \cdot f'_{net}(x_p)_j$$

---

$$E(w) = \sum_{p=1}^{l} \left(d_p - o(x_p)\right)^2$$

$$\begin{cases} net_i = \sum_j w_{ij} x_j \\ o_i = f_\sigma(net_i(x)) \end{cases}$$

$$\frac{\partial E_p}{\partial w_j} = \frac{\partial E_p}{\partial net} \cdot \frac{\partial net_{(xp)}}{\partial w_j} \rightsquigarrow \frac{\partial \sum_i x_{pi} w_i}{\partial w_j} = x_{pj}$$

$$\downarrow$$

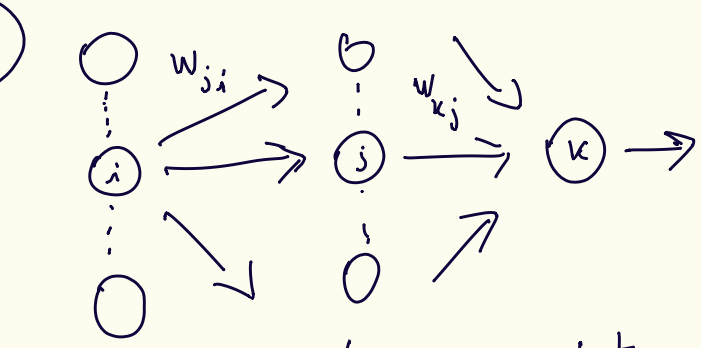
$$\frac{\partial (d_p - o_p)^2}{\partial o_p} \cdot \frac{\partial o_p}{\partial net} \Rightarrow \frac{\partial f_\sigma(net)}{\partial net} = f'_\sigma(net)$$

$$\downarrow$$

$$2(d_p - o_p) \cdot \frac{\partial (d_p - o_p)}{\partial o_p} = -2(d_p - o_p)$$

$$\Rightarrow -2(d_p - o_p) \cdot f'_\sigma(net)$$

MLP




INPUT layer — Hidden layer — output layer

- UNITÀ CONNESSE DA UN DEI LINK PESATI
- ORGANIZZATE IN LAYERS
- INPUT layer proietta nella hidden unit
- feedforward architecture

- TWO LAYER FEEDFORWARD NN

$$h(x) = f_k\left(\sum_j w_{kj} f_j\left(\sum_i w_{ji} x_i\right)\right)$$

$$h(x) = f_k\left(\sum_j w_{kj} f_j\left(\sum_i w_{ji} x_i\right)\right)$$

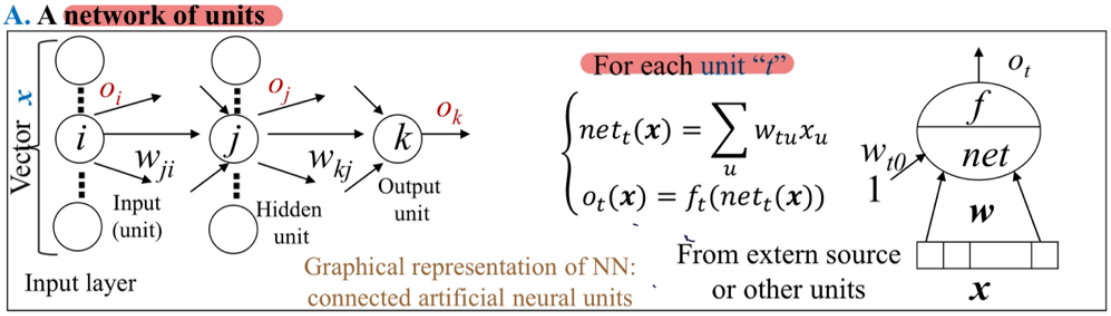
feed forward — non-linear function (sigmoid) — input

# The NN and their Units

## Standard feedforward NN (with one hidden layer)

Dip. Informatica
University of Pisa

**A. A network of units**




Graphical representation of NN: connected artificial neural units

Notation:

- The index *i* denotes a generic unit, it can be either *k* or *j* (*)
- The index *u* denotes a generic input component, it can be either *i* or *j* (*)
- In the unit, ***x*** is a generic input from an external source (input vector) or from other units (*) *according to the position of the unit in the network*
- If we load the pattern ***x*** in the input layer, we can use the notation with ***o*** for both the inputs and the hidden units outputs. Hence, *inside the network*, the input to each unit *t* from *any source u* (through the connection $w_{tu}$) is typically denoted as $o_u$ (we will use this style in the back-prop derivation).

57

A. Micheli

- MLP need A NON-LINEAR activation function IN THE hidden layer
  ⇒ otherwise becomes A LINEAR MODEL

$$\begin{cases} net_t(x) = \sum_u w_{tu} x_u \\ out_t(x) = f_t(net_t(x)) \end{cases}$$

$$\begin{cases} net_t(x) = \sum_u w_{Tu} x_u \\ out_T(x) = f_t(net_T(x)) \end{cases}$$

- change w can have ≠ functions ⇒ different H space
- output: discrete || continuous values ⇒ classification / regression task
  (just change ACTIVATION FUNCTION OF THE OUTPUT UNITS)

CHANGE THIS → $h(x) = f_k\left(\sum_j w_{kj} f_j\left(\sum_i w_{ji} x_i\right)\right)$ → SIGMOID / linear activation function

# NN and LBE

- NN is a function NON-LINEAR in the parameters $W$
  - $W$ param. is inside a non-linear activation function

$$h_k(x) = f_k\left(\sum_j w_{kj} f_j\left(\sum_i w_{ji} x_i\right)\right)$$

STANDARD: fixed LBE: $\sum_j w_j \phi_j(x)$ • linear wrt parameters • phi predefined

NN: flexible/adaptive LBE: — consider this as phi function

- $h(x) = f_k\left(\sum_j w_{kj} \boxed{f_j\left(\sum_i w_{ji} x_i\right)}\right)$
- $\phi_j(x, w) = f_j\left(\sum_i w_{ji} x_i\right)$ phi depends on w

↓

is a LBE where the phi IS NOT FIXED (NOT), chosen a PRIORY BY you

we have weights inside, changed by TR ALGORITHM

=> change automatically according to your task

- each BASIS FUNCTION (hidden unit) computes a new NON-LINEAR derived FEATURES, ADAPTIVELY TO THE TR DATA
- ERROR FUNCTION NO MORE CONVEX => non-linear depending wrt to w

=> non-linear optimization problem

## 2. Universal approximation (important!)

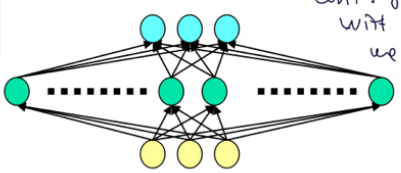
Dip. Informatica University of Pisa

- Many early results (Cybenko 1989, Hornik et al. 1993, etc…)
- Shortly:
  - A single hidden-layer network (with logistic activation functions) can approximate (arbitrarily well) every continuous (on hyper cubes) function (provided enough units in the hidden layer)
    [e.g. image it generalizes approximation by finite Fourier series]
  - A MLP network can approximate (arbitrarily well) every input-output mapping (provided enough units in the hidden layers)

Given ε, ∃ $h(x)$ s.t. $|f(x)-h(x)|<\varepsilon$ ∀ $x$ in the hypercube

(hypothesis: prompt find type)

$$h(x) = \underset{\text{linear}}{f_k}\left(\sum_j w_{kj} f_j\left(\sum_i w_{ji} x_i\right)\right)$$



(so: can approx any cont. $f(x)$ with precision we want)

*Existence theorem*! This not provide the algorithm nor the number of units

71

- NN are universal approximators

ANY degree of accuracy you like

- given $\varepsilon$, $\exists\, h(x)$ such that $|f(x) - h(x)| < \varepsilon$ $\forall\, x$ in hypercube

# NN - learning algorithm

: adapt free parameters $w$ to obtain best approximation of the target function
= minimize error via LMS

$$E(w) = R(emp) = \frac{1}{\ell} \sum_{p=1}^{\ell} \left( L(h(x_p)), d_p \right)$$

$$L(h(x_p), d_p) = MSE = (d_p - h(x_p))^2$$

FIND $w$ TO min. residual sum of squares

BACKPROP.

- STIMARE IL CONTRIBUTO DELLE HIDDEN UNITS PER L'ERRORE ALL'OUTPUT LEVEL
  => Computing GENERALIZED $\Delta$ RULE

$$E_{TOT} = \sum_p E_p = \frac{1}{2} \sum_{k=1}^{K} (d_k - out_k)^2$$

FIND $w$ TO MINIMIZE $E_{TOT}$

**Algorithm 1:** Back-propagation (the same presented in general for the LMS)

1. Initialize all the weights $\mathbf{w}$ in the network and $\eta$;
   Compute units outputs and $E_{tot}$;
   **while** $E_{tot} > \epsilon$ *(desired value or other criteria)* **do**
   $\forall w \in \mathbf{w}$: $\Delta w = -\frac{\partial E_{tot}}{\partial w}$ (Step 1);
   $w^{new} \leftarrow w + \eta \Delta w + \dots$ (Step 2) ;
   Compute units outputs and $E_{tot}$;
   **end**

1) compute partial derivatives (gradient)
2) update the weights

$$\Delta_w = -\frac{\partial E_{TOT}}{\partial w} = -\sum_p \frac{\partial E_p}{\partial w} = \Delta_p w$$

$$\Delta_p w_{tM} = -\frac{\partial E_p}{\partial w_{tM}} = \left[ -\frac{\partial E_p}{\partial net_t} \right] \cdot \left[ \frac{\partial net_T}{\partial w_{TM}} \right] = \left[ \delta_t \right] \left[ O_M \right]$$

$$\frac{\partial net_t}{\partial w_{tM}} = \frac{\partial \sum_j w_{tj} O_j}{\partial w_{tM}} = O_M \quad \text{for } j = M$$

$$\delta_t = -\frac{\partial E_p}{\partial net_T} = \left[ -\frac{\partial E_p}{\partial O_t} \right] \cdot \frac{\partial O_t}{\partial net_t} =$$

$$\frac{\partial f_t(net_t)}{\partial net_t} = f_t'(net_t)$$

(1) OUTPUT LAYER $= -\frac{\partial E_p}{\partial O_t} \cdot f_t'(net_t)$

$$-\frac{\partial E_p}{\partial O_t} = \frac{\partial \frac{1}{2} \sum_{R=1}^{K} (d_R - O_R)^2}{\partial O_t} = d$$

## GRADIENT COMPUTATION

$$net_T = \sum_{i=1}^{n} w_{Ti} o_i$$

$$o_t = f_t(net_t)$$

$$\Delta w = -\frac{\partial E_w}{\partial w} = -\sum_{p=1}^{\ell} \frac{\partial E_p}{\partial w} = \sum_{p=1}^{\ell} \Delta_p w$$

- compute $\forall$ component $u$ of the vector $w$ the $\Delta_p w_{t,u}$

$$\Delta_p w_{t,u} = -\frac{\partial E_p}{\partial w_{tu}} = \underbrace{-\frac{\partial E_p}{\partial net_t}}_{\delta_t} \cdot \underbrace{\frac{\partial net_t}{\partial w_{tu}}}_{o_u} = \delta_t \, o_u$$

$$o_j = \frac{\partial net_t}{\partial w_{tu}} = \frac{\partial \sum_j w_{tj} o_j}{\partial w_{tu}} \cdot * = o_u$$

* cuz $w_{th} \cdot o_h$ is constant WRT $w_{tu}$ but for $j = u$

$$\Rightarrow \Delta_p w_{tu} = \delta_t \cdot o_u$$

$$\delta_t = -\frac{\partial E_p}{\partial net_t} = \underbrace{-\frac{\partial E_p}{\partial o_t}}_{\text{2 CASES}} \cdot \frac{\partial o_t}{\partial net_t} = \frac{\partial f_t(net_t)}{\partial net_t} = f_t'(net_t)$$

### 1) OUTPUT UNIT (been) : assume $t = k$ (consider an output unit of the network)

$$-\frac{\partial E_p}{\partial o_k} = -\frac{\partial \frac{1}{2} \cdot \sum_{R=1}^{K} (d_R - o_R)^2}{\partial o_k} = 2 \cdot \frac{1}{2} (d_k - o_k) = (d_k - o_k)$$

$$\Rightarrow \boxed{\delta_k = (d_k - o_k) \cdot f_k'(net_k)}$$

### 2) HIDDEN LAYER (# output units $K$) : $t = j$ => chain rule

$$-\frac{\partial E_p}{\partial o_j} \overset{\text{(1)}}{=} \sum_{k=1}^{K} -\frac{\partial E_p}{\partial net_k} \cdot \frac{\partial net_k}{\partial o_j} = \sum_{k=1}^{K} \delta_k \cdot \frac{\partial \sum_s w_{ks} o_s}{\partial o_j} = \sum_{k=1}^{K} \delta_k \cdot w_{kj}$$

SUBSTITUTING IN (*)

$$\Rightarrow \boxed{\delta_j = \left(\sum_{k=1}^{K} \delta_k \cdot w_{kj}\right) \cdot f_j'(net_j)} \quad \checkmark$$

**Summary**

We derived, for two layers,

$$\Delta_p w_{tu} = \delta_t \cdot o_u$$

where $\delta_t$ is the error signal available to the unit $t$, and $o_u$ is the input to $t$ from a generic unit $u$, i.e. the input through the connection weighted by $w_{tu}$. In particular:

- $\Delta_p w_{kj} = \delta_k \cdot o_j$ if $t = k$ (output unit) and the input is from unit $j$
- $\Delta_p w_{ji} = \delta_j \cdot o_i$ if $t = j$ (hidden unit), and input $i$

The other two equations specify the error signals:

$\delta_k = (d_k - o_k) \cdot f_k'(net_k)$ if $t = k$ (output unit)

$\delta_j = (\sum_{k=1}^{K} \delta_k w_{kj}) \cdot f_j'(net_j)$ if $t = j$ (hidden unit)

backprop.

$$\Delta w = -\frac{\partial E_{tot}}{\partial w} = -\sum_{p=1}^{l} \frac{\partial E_p}{\partial w} = \sum_{p=1}^{e} \Delta_p w$$

$$\cdot \Delta_p w_{T,U} = -\frac{\partial E_p}{w_{T,U}} = -\frac{\partial E_p}{\partial net_T} \cdot \left(\frac{\partial net_t}{\partial w_{T,U}}\right) = \delta_t \, o_M$$

$$= \frac{\partial \sum_j^2 w_{tj} o_j}{\partial w_{T,U}} = o_M \quad \text{when } j = U$$

$$= -\frac{\partial E_p}{\partial net_T} = -\frac{\partial E_p}{\partial o_T} \cdot \left(\frac{\partial o_T}{\partial net_T}\right) = \delta_t$$

$$\rightarrow \frac{\partial f_t(net_t)}{\partial net_T} = f_t' \, net_t$$

2 cases

$j$ = output

$t$ = hidden unit

$T = K$ outputs $\Rightarrow$ $\delta_K = -\frac{\partial E_p}{\partial o_K} \cdot \frac{o_K}{\partial net_K} = (d_K - o_K) \cdot f_K' \, net_K$

$$\frac{-\partial \frac{1}{2} \sum_{R=1}^{K} (d_R - o_R)^2}{\partial o_K} = (d_K - o_K)$$

$t = j$

$$\delta_j = \left(-\frac{\partial E_p}{\partial o_j}\right) \cdot \frac{\partial o_j}{\partial net_j} = \sum_{K=1}^{K} \delta_K w_{Kj} \cdot f_j'(net_j)$$

$$\frac{\partial f_j(net_j)}{\partial net_j} = f_j'(net_j)$$

CHAIN RULE

$$= \sum_{K=1}^{K} \left(\frac{-\partial E_p}{\partial net_K}\right) \cdot \frac{\partial net_K}{\partial o_j} = \sum_{K=1}^{K} \delta_K \cdot w_{Kj}$$

$\delta_K$

$$\frac{\partial \sum_s w_{K,s} \cdot o_s}{\partial o_j} = (w_{Kj})$$

$$\Rightarrow \boxed{\delta_j = \sum_{K=1}^{K} \delta_K w_{Kj}}$$

Hyperparameters: valori da scegliere x inizializzare e fare il TRAIN

# LEARNING CURVE ANALYSIS

① EARLY STOPPING : use a VALIDATION SET to decide when to STOP
(look over a set of epoch.
if validation error ↑
⇒ STOP)

- since training means
make grows # params
⇒ go back limits complexity

⇒ go back to best solution before see increase in VL set

② REGULARIZATION

- training ⇒ weights increase : optimize the loss considering the weights values

⇒ TIKONOV : add penalty term to the error function

$$loss(w) = \sum_p (d_p - o(x_p))^2 + \lambda \|w\|^2$$

effect: weight decay : $w_{new} = w + eta \cdot \Delta_w - \lambda w$

(if applied to linear model: RIDGE regression)

- Loss : is the objective function : MSE + PENALTY TERM
- Error : risk for the data term (MSE, depends on data): $\sum_p (d_p - O(x_p))^2$
TO EVALUATE MODEL ERROR

Regularization NOT CONTROL the STABILITY
BUT CONTROL THE COMPLEXITY

#UNITS : control the complexity, MODEL SELECTION issues
too few hidden units → underfitting
too many → overfitting

⇒ 2 METHODS : CONSTRUCTIVE APPROACH
PRUNING METHODS

CONSTRUCTIVE APPROACH: incremental, build network starting with a minimal configuration, add new units and connections during training

example: CASCADE CORRELATION learning algorithm

- START WITH a minimal network
- add units until the error is slow
- cc off learn about network weights and network topology (#units)
- H space of flexible size since ~~#~~units decided by the algorithm
- TRAIN a SINGLE UNIT for each STEP
  (then decide to add or not the units)

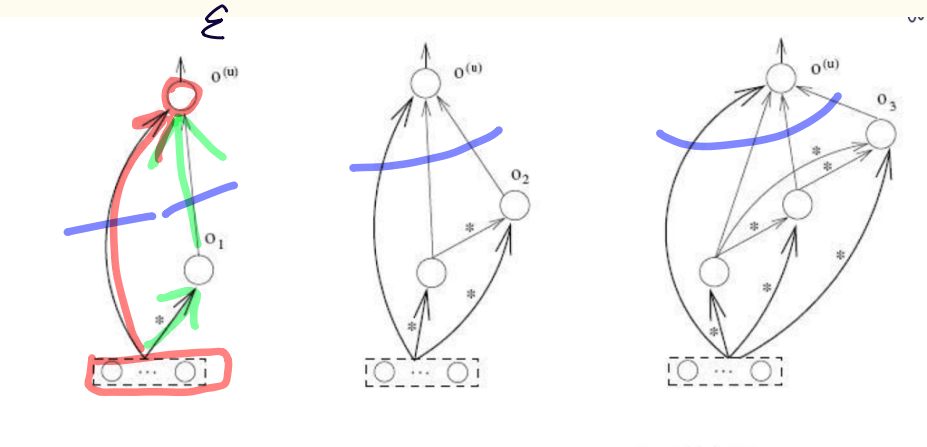
Inizio solo con input e allora l'output unit: se errore basso => stop

altrimenti aggiungo un hidden unit e la connetto, e la alleno per errore CORRELATA ALL'errore che ho sullo output layer

- RIALLENO LA RETE (non la hidden unit non più nuova)

E così

- add $O_2$ frozing weights and correlate the output of this hidden unit to the residual error




## Learning for CC

Dip. Informatica
University of Pisa

- Specifically, the algorithm works interleaving the minimization of the total error function (LMS), e.g. by a simple backpropagation training of the output layer, and the **maximization** of the (non-normalized) correlation, i.e. the covariance, of the new inserted hidden (candidate) unit with the residual error:

$$S = \sum_k \left| \sum_p (o_p - \underbrace{mean_p(o_p)}_{\bar{o}})(E_{p,k} - \underbrace{mean_p(E_{p,k})}_{\bar{E}}) \right|$$

$E_{p,k} = (o_{p,k} - d_{p,k})$

$E$: residual error with $o_{p,k}$ at the output layer

$p$: pattern, $k$: output unit

$o$ (in eq. for S): candidate outpu

- **Exercise:** derivation (sketch on blackboard)
  hints: $d|f|/dx = sign(f)\ df/dx$; assume no effect of variations of $o_p$ over $\bar{o} = mean_p(o_p)$
- Result:

$$\frac{\partial S}{\partial w_j} = \sum_k \text{sgn}\, S_k \sum_p (E_{p,k} - mean_p(E_{p,k}))\, f'(net_{p,h}) I_{p,j}$$

Input

$h$ : candidate index (not needed since we have just one candidate at time)

- Weight update: why + dS/dw$_j$ ?
  => + GRAD we are MAXIMIZE (gradient ascent instead of GRAD desc.

71

A. Micheli

$$\frac{\partial S}{\partial w_j} = \frac{\partial \left( \sum_{k=1}^{K} \left| \sum_{p=1}^{P} (o_p - avg_p(o)) \cdot (E_{k,p} - avg_p(E_k)) \right| \right)}{\partial w_j} =$$

$$= \sum_{k=1}^{K} \left[ sign(S_k) \cdot \sum_{p=1}^{P} \frac{\partial (o_p - avg_p(o))(E_{k,p} - avg_p(E_k))}{\partial w_j} \right] =$$

CHAIN RULE

$$= \sum_{k=1}^{K} \left[ sign(S_k) \cdot \sum_{p=1}^{P} \left( E_{p,k} - avg_p(E_k) \right) \cdot \frac{\partial (o_p - avg_p(o))}{\partial net_p} \cdot \frac{\partial net_p}{\partial w_j} \right)$$

$$= \sum_{k=1}^{K} \left[ sign(S_k) \cdot \sum_{p=1}^{P} \left[ E_{p,k} - avg_p(E_k) \right) \cdot 1 \cdot \frac{\partial net_p}{\partial w_j} \cdot \frac{\partial o_p}{\partial net_p} \right]$$

$$= \sum_{k=1}^{K} (sign(S_k) \cdot \sum_{p=1}^{P} (E_{p,k} - AVG(E_k)) \cdot f'(net_p) (I_{j,p})$$

$I_{j,p}$

$f'(net_p)$

SLT

- can approximate estimation of VL set error (for model selection) and TS error (model assessment)

  => SRM: Structural Risk Minimization and VC-dimension

VC-DIMENSION: measure of Flexibility (complexity of a CLASS of HYPOTESIS (also for ∞ hypotesis space H)

- N data = $2^N$ possibili dicotomie
- H frammenta X iff H può rappresentare tutte le possibili dicotomie su X (o errore)
- VC-DIMENSION DI UNA CLASSE di funzioni H È LA CARDINALITÀ MASSIMA DI UN INSIEME (configurazione) DI PUNTI IN X (INSTANCE SPACE) CHE PUÒ ESSERE FRAMMENTATO DA H

  ↳ instance space

  X = 3

VC(H) = p

- H frammenta almeno un insieme (una configurazione) di p punti
- H non frammenta nessun insieme (configurazione) di p+1 punti

(quindi basta trovare una configurazione di punti che può essere frammentata)

VC(H) ≥ 3  VC(H) < 4  => VC(H) = 3

• N° di punti che possono essere classificati senza errore

- In general: The VC dimension of a class of linear (separator/decision) hyperplanes (LTU) in a **n**-dimensional space is **n + 1**

*Note: upper-bound, it can be less constraining the model, we'll see how...*

• our aim is to reduce VC-dim

es |Dataset| must be linear in the free parameters of a linear model

STRUCTURAL RISK MINIMIZATION

- SRM USA la VC-dimension come CONTROLLING PARAMETER per minimizzare il GENERALIZATION BOUND SU R

  => must minimize the risk

- Assumig FINITE VC-DIMENSION posso definire modelli ANNIDATI dello spazio delle ipotesi in base alla VC-dimension

$H_1 \subseteq H_2 \subseteq \dots \subseteq H_n$

$VC(H_1) \leq VC(H_2) \leq \dots \leq VC(H_n)$

Some example?

- NN with increasing number of Hidden units (roughly), but also the #epochs (we know that also the size of weights care and a NN is a variable-size H)
- Polynomial of increasing degree *M*
- Increasing values for c, where ||w|| < c for regularization (or controlled by λ)
- Increasing number of nodes in a decision tree (roughly)
- *Note: important* hyper-parameters → related to VC-dim

A. Micheli

# SRM for model selection

Dip. Informatica
University of Pisa

Growing VC-dim: empirical (TR) error decreases, VC-confidence increases

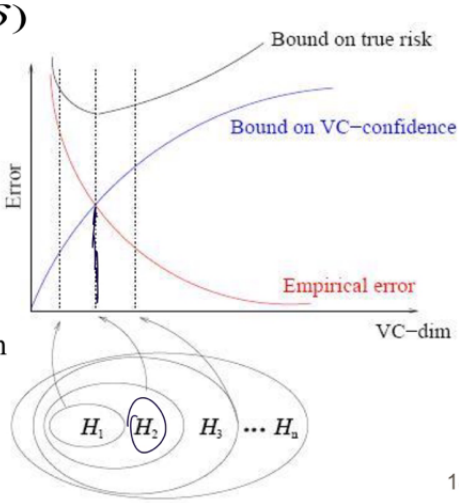
**S**tructural **R**isk **M**inimization: find a trade-off on the bound (as seen in the introduction using $l$):

$$R \leq R_{emp} + \varepsilon(1/l, VC, 1/\delta)$$

- $\mathcal{H}_1 \subseteq \mathcal{H}_2 \subseteq \cdots \subseteq \mathcal{H}_n$
- $VC(\mathcal{H}_1) \leq \cdots \leq VC(\mathcal{H}_n)$

e.g. NN with increasing number of Hidden units

**Model selection**: choose the model ($h$) with the better bound on the true risk



SVM

N: # examples (instead of l)

m: dimension of input vector (instead of n)

b: bias (instead of $w_0$)

SVM: - linear machine (LTU, perceptron)
- maximization on separation margin
- structural risk minimization

- initially (hard margin SVM)
- assume:
  - linear separable problems
  - no errors in data

- given TR set $T = \{(x_i, d_i)\}_{i=1}^{N}$
- find hyperplane of equation $w^T x + b = 0$ to separate the examples

$w^T x_i + b \geq 0$ for $d_i = +1$

$w^T x_i + b < 0$ $d_i = -1$

- $g(x) = w^T x + b$ è CHIAMATA DISCRIMINATIVE function
- $h(x) = \text{sign}(g(x))$ è L'IPOTESI

- MARGIN = $\rho$ (SEPARATION, RHO) = il doppio della distanza tra l'iperpiano e il punto più vicino

- posso avere diversi margini = non con UGUAGLIANZA stretta => soluzioni ≠ in termini di margin
  ≠ dal MLP che le soluzioni sono equi

- voglio L'OTTIMO iperpiano ⟶ quello CHE MASSIMIZZA IL MARGINE $\rho$

→ $w_0^T x + b_0 = 0$ dove in $w_0^T$ 0 is OPTIMAL

→ (L'OTTIMO È trovare) $\rho = \frac{2}{\|w\|}$ MASSIMIZZARE $\rho$ ⟺ MASSIMIZZARE $\|w\|$

FORMA CANONICA DELL'IPERPIANO ⟶ $d_i(w^T x_i + b) \geq 1 \quad \forall i = 1..N$

support vector $x^{(s)}$ soddisfa l'eq. $d^{(s)}(w^T x^{(s)} + b) = 1$

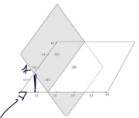
$$\mathbf{w}^T\mathbf{x}_i + b \geq 0 \quad \text{if } d_i = +1$$
$$\mathbf{w}^T\mathbf{x}_i + b < 0 \quad \text{if } d_i = -1$$

Re-scaling $\mathbf{w}$ and $b$ (see scaling-freedom property) so that the closest points to the separating hyperplane satisfy $|g(\mathbf{x}_i)| = |\mathbf{w}^T\mathbf{x}_i + b| = 1$, we can write:

$$\mathbf{w}^T\mathbf{x}_i + b \geq 1 \quad \text{if } d_i = +1$$
$$\mathbf{w}^T\mathbf{x}_i + b \leq -1 \quad \text{if } d_i = -1$$

In a compact form:

$$d_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 \; \forall i = 1, \cdots, N$$



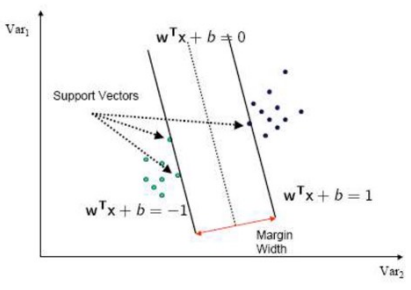
→ if satisfy this, all points are exactly classified

**Support Vector**

A support vector $\mathbf{x}^{(s)}$ satisfies the previous equation exactly:

$$d^{(s)}(\mathbf{w}^T\mathbf{x}^{(s)} + b) = 1$$

→ POINTS AT THE edge of the MARGIN

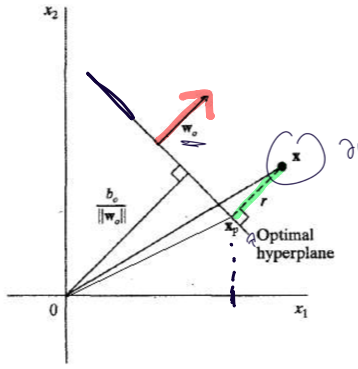


VOGLIO TROVARE LA LARGHEZZA DEL MARGINE (SIZE)

## Distance to the hyperplane

Let's denote $g(\mathbf{x})$ as the discriminant function: $g(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + b$.
Recall that $\mathbf{w_o}$ is a vector orthogonal to the hyperplane
Let's denote the distance between $\mathbf{x}$ and the optimal hyperplane with $r$ (according to the orthogonal vector to the hyperplane) (exactly $W_o$)

AIM: measure the margin




any point x expressed w.r.t. $x_p$ and r

adapt

Figure: Distance between a point and the optimum hyperplane.

point on plane

$$\text{pour} \longrightarrow \mathbf{x} = \mathbf{x_p} + r\frac{\mathbf{w_o}}{\|\mathbf{w_o}\|} \longrightarrow$$

real movement of r and its direction



$\Rightarrow$ posso esprimere x come

$$x = x_p + r\frac{w_o}{\|w_o\|}$$

- $x_p$: proiezione di x sull'ottimo ipersiano
- $r$: distanza (will be the margin)
- $\frac{w_o}{\|w_o\|}$: orthogonal vector with size 1 (cioè is normalized w.r.t. to norm $\|w\|$)

$$x = x_p + r\frac{w_o}{\|w_o\|}$$

$$x = \left(x_p + r\frac{w_o}{\|w_o\|}\right)$$

$$g(x) = w_o^T x + b_o$$

$$g(x) = g\left(x_p + r\frac{w_o}{\|w_o\|}\right) = \underbrace{w_o^T x_p + b_o}_{g(x_p)} + w_o^T r\frac{w_o}{\|w_o\|} = g(x_p) + w_o^T r\frac{w_o}{\|w_o\|} =$$

$$= g(x_p) + r\, w_o^T\frac{w_o}{\|w_o\|} = r\frac{\|w_o\|^2}{\|w_o\|} = r\|w_o\|$$

$x_p$ è un punto sull'iperpiano quindi è zero

$$g(x) = r\|w_o\|$$

$$r = \frac{g(x)}{\|w_o\|}$$

- Considero la distanza tra l'ipepiano e il support vector positivo $x^{(s)}$

$$\text{i.e. } r \text{ per } x^{(s)} = \frac{g(x^{(s)})}{\|w_o\|} = \frac{1}{\|w_o\|} = \frac{\rho}{2}$$

$\rho$ = MARGIN

this distance is half of the margin

$$\rho = \frac{2}{\|w_o\|}$$

adesso trovo $w_0$ e $b$ che mi da la soluzione con il MARGINE MASSIMO for class. with no errors

HARD MARGIN SVM

given training examples $T = \{(x_i, d_i)\}_{i=1}^{N}$

find optimal $w_0$ e $b$ che minimizzano

$$\Psi = \frac{1}{2} w^T w \quad (\min \|w\|)$$

SATISFYING zero-errors condition (constraint)

$$d_i (w^T x_i + b) \geq 1 \quad \forall i = 1, \dots, N$$

- objective function $\varphi(w)$ is quadratic and convex in $w$
- constraints are linear in $w$
- Solving this problem scales with size of input space m (later change on N: # data)

WHY INTRODUCING SVM? GOOD IN TERM OF GENERALIZATION

## Why does this improve the generalization?

We fixed the training error on linearly separable problems. Than minimizing the norm of **w** is equivalent to minimizing the VC dimension and thus to minimizing the capacity term (VC-confidence) $\epsilon$ in

$$R[h] \leq R_{emp}[h] + \epsilon(VC, N, \delta)$$

" VC-confidence — thru objective f(x) of primal form

Relevant!!!

**Theorem - Vapnik**

Let $D$ be the diameter of the smallest ball around the data points $\mathbf{x}_1, \cdots, \mathbf{x}_N$. For the class of separating hyperplanes described by the equation $\mathbf{w}^T\mathbf{x} + b = 0$ the upper bound to the VC dimension is

$$VC \leq min(\lceil \frac{D^2}{\rho^2} \rceil, m_0) + 1 \qquad \rho$$

since $\frac{D^2}{\rho^2} = Radios^2 \|\mathbf{w}\|^2$, VC can be less than $m_0 + 1$ computed for general hyperplanes by constraining the search to the "regularized" one with maximum margin : regularization => reduce VC <=> control size of weights $\|w\|$ ?



- Fix the $R_{emp}$ (even posed to zero) (no error cuz is linear. sep.)
  => just minimize $\|w\|$ is equivalent to minimize the VC-dimension
  => minimize the VC-confidence

implementing directly STRUCTURAL RISK MIN.

- SVM minimize VC-confidence thru maximization of the margin thru VAPNIK THEOREM

$$VC \leq \min\left(\left\lceil \frac{D^2}{\rho^2} \right\rceil, m_0\right) + 1$$

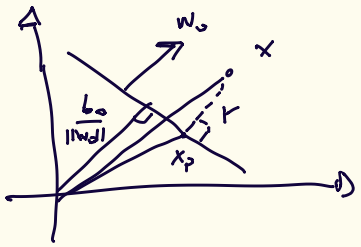
$D$ = diameter of the ball
$m_0$ = dimension of input space

=> CAN find an optimal solution in term of Generaliz. with no error and no hyperparameters

Forma canonica $d_i (w^T x_i + b) \geq 1 \quad \forall i = 1, \dots, N$

support vector $x^{(s)}$

$$d^{(s)} (w^T x^{(s)} + b) = 1$$




distanza dell iperpiano di un punto x

$$x = x_p + r \frac{w_0}{\|w_0\|}$$

LAGRANGIANA iperpiano OTTIMO

$$g(x) = w^T x_0 + b_0$$

$$g\left(x_p + r \frac{w_0}{\|w_0\|}\right) = \underbrace{w^T x_p + b_0}_{g(x)} + w^T r \frac{w_0}{\|w_0\|} = \underbrace{g(x)}_{0} + r \frac{w^T w_0}{\|w_0\|} = r \frac{\|w_0\|^2}{\|w_0\|} =$$

$$r \|w_0\|$$

$$g(x) = r \|w_0\|$$

$$r = \frac{g(x)}{\|w_0\|} \quad \text{per } x^{(s)}$$

$$\Rightarrow r = \frac{g(x^{(s)})}{\|w_0\|} = \frac{1}{\|w_0\|} = \frac{\rho}{2} \qquad \rho = \frac{2}{\|w_0\|}$$

TROVO w e b OTTIMI per MASSIMIZZARE IL MARGINE: HARD MARGIN

- given Tr samples $T = \{(x_i, d_i)\}_{i=1}^{N}$ find w & b s.t. minimize (CANONICAL FORM) $\psi(w) = \frac{1}{2} w^T w$ satisfying the inequality condition:

$$d_i (w^T x_i + b) \geq 1$$

THEOREM: maximum margin solution correspond the MINIMUM POSSIBLE VC-dimension $\Rightarrow$ best possible generalization capability

$$R \leq R_{emp} + \varepsilon(VC, N, \delta) \qquad VC \leq \min\left(\left\lceil \frac{D^2}{\rho^2} \right\rceil, m_0\right) + 1$$

OPTIMAL VALUES of w and b

$$w_0 = \sum_{i=1}^{N} \alpha_{0,i} d_i x_i \quad \Rightarrow \text{so the optimal hyperplane}$$

$$w_0^T x + b_0 = 0 \Rightarrow \sum_{i=1}^{N} \alpha_{0,i} d_i x_i^T x + b_0 = 0$$

KUN-TACKER CONDITION

$$\alpha_i (d_i (w^T x_i + b) - 1) = 0$$

if $\alpha_i > 0 \Rightarrow d_i (w^T x_i + b) = 1$ and $x_i$ is a support vector

if $x_i$ is NOT a support vector $\Rightarrow \alpha_i = 0$

$\Rightarrow$ CAN RESTRICT THE COMPUTATION to $N_s$: $w_0 = \sum_{i=1}^{N_s} \alpha_{0,i} d_i x_i$

$\rightarrow$ HYPERPLANE DEPENDS ONLY FROM SUPPORT VECTOR

- solving DUAL problem obtaining $\alpha \quad \{\alpha_{0,i}\}_{i=1}^{N} \qquad f(x) = w_0^T x + b_0 = 0$

- compute $w_0 = \sum_{i=1}^{N} \alpha_{0,i}\, d_i\, x_i$

- compute $b_0 = 1 - w_0^T x^{(s)}$ corresponding to positive support vector

$$b_0 = 1 - \sum_{i=1}^{N} \alpha_{0,i}\, d_i\, x_i^T x^{(s)}$$

so $w_0 = \sum_{i=1}^{N} \alpha_{0,i}\, d_i\, x_i$

decision surface is given by $w_0^T x + b_0 = 0 \iff \sum_{i=1}^{N} \alpha_{0,i}\, d_i\, x_i^T x + b_0 = 0$ (DOT PRODUCT)

=> classify point $x$ with the sign of $f(x)$

## Why does this provide an elegant approach?

For **linearly separable** data there are many solutions. Vapnik propose an "optimal separating hyperplane" maximizing the margin providing (CUT IS CONVEX PROBLEM)

- an unique solution (not for iterative Perceptron Learn. Alg.) with zero errors (not for LMS) for the binary classifier
- an automatized approach to SRM that minimizes VC-confidence (by maximizin the margin) as part of the training process (directly into the optimization process), without hyper-parameters in the linear-separable case
- the use of solver in the class of constrained Quadratic Programming (instead of gradient descent) with a nice dual form (showing support vectors and the dot product among patterns)
- a solution focused on "selected" training data, the support vectors
  - (nice) not depending directly on the point far away from the decision boundary
  - (less nice) hoping that the points at the boundaries are not the noisier ones.

But what for noisy or non linearly separable data?

=> SOFT MARGIN: admit errors (not every separable, data with error)

=> data point inside the margin (also classification errors)

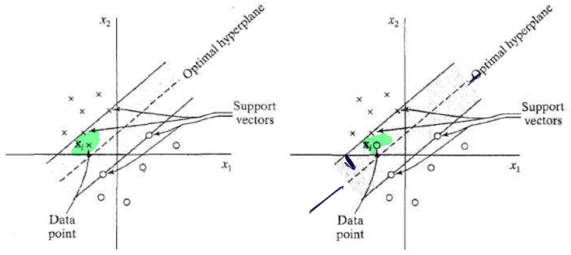



Figure: Patterns violating the margin (with and without classification error).

At least one point violate the exact-fit condition:

$$d_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1$$

Exercise: find the margin for the figures above without admitting points inside the margin. This explain the reason for soft margin: noise tolerance and larger margin

Soft Margin.

Note: admitting points inside the margin allows us to have a larger margin.

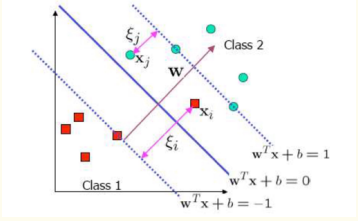


- admitting ERRORS:
  - admit POINTS IN THE MARGIN ALLOW US TO MAKE A LARGE MARGIN
    => BETTER GENERALIZATION

- introduce Slack-variables

$$\xi_i \geq 0 \quad \forall i = 1,\dots,N \qquad d_i(w^T x_i + b) \geq 1 - \xi_i \quad \forall i = 1,\dots,N$$

A SUPPORT VECTOR $x_i$ SATISFY: $d_i(w^T x_i + b) \geq 1 - \xi_i \quad \forall i = 1,\dots,N$




VAPNIK THEOREM NOT HOLD

PRIMAL FORM $\psi(w, \xi) = \frac{1}{2} w^T w + C \sum_{i=1}^{N} \xi_i$

CONSTRAINT $d_i (w^T x + b) \geq 1 - \xi_i \quad \forall i = 1, \dots, N$

$\xi_i \geq 0 \quad \forall i = 1, \dots, N$

C is REGULARIZATION hyper-parameter

low C => TR error admitted => possible underfitting

high C => no TR allowed => possible overfitting

We LOST SRM

⇓

A trade off between Keep minimized $\|w\|$ and VC-dimension

HARD MARGIN WITH C

=> High C

KAN-TACKER CONDITION

$\alpha_i \left( d_i (w^T x_i + b) + \xi_i - 1 \right) = 0 \quad \forall i = 1, N$

$\mu_i \xi_i = 0 \quad \forall i = 1, \dots, N$

↓

Lagrange multiplier

1 unknown

$0 < \alpha_i < C \Rightarrow \xi_i = 0$ (on the margin)

$\alpha_i = C \Rightarrow \xi_i \geq 0$ (inside the margin)

- Solve the dual problem with respect to $\{\alpha_i\}_{i=1}^N$
- Find $\mathbf{w_o}$ and $b_o$ from $\{\alpha_{o,i}\}_{i=1}^N$

$$\mathbf{w_o} = \sum_{i=1}^{N} \alpha_{o,i} d_i \mathbf{x_i}$$

$$b_o = d_j - \sum_{i=1}^{N} \alpha_{o,i} d_i \mathbf{x_i}^T \mathbf{x}_j \text{ for a pattern j such that } 0 < \alpha_j < C$$

(or an average of all the solutions for numerical stability)
The non zero Lagrangian Multipliers correspond to support vectors.

How do we use it?

As before:

$$g(\mathbf{x}) = \sum_{i=1}^{N} \alpha_{o,i} d_i \mathbf{x_i}^T \mathbf{x} + b_o$$

$$h(\mathbf{x}) = sign(g(\mathbf{x}))$$

PER i PROBLEMI NON LINEARI => need LBE moving in the feature SPACE

=> concept of (kernel)

non linear function map $\phi : R^{m_0} \to R^{m_1}$ $\quad x \longrightarrow \phi(x)$

, TR set becomes $T = \{(\phi(x), d_i)\}_{i=1}^{N} \quad \forall i = 1, N$

, hyperplane becomes $w^T \phi(x) + b = 0$

CHOICE $\phi$ is CHALLENGING in SVM

WITH LBE : NON-linear mapping of input patterns to high-dimensional FEATURE SPACE

(COVER'S THEOREM: the patterns are linear seperable with high probability in the feature space under such condition (increase the dimension from original space to high dim. space (where we project the data)

=> However, using high-dim. feature space (large LBE) can LEAD TO OVERFITTING (cost complexity of classifier) => KERNEL APPROACH

⊗ TO MANAGE THE FEATURE SPACE in the context of REGULARIZED MODEL

⇓

Complexity of LBE is related to the # free-parameters (# dimension of the new phi expansion) ed INPUT dimensionality

⊗ THE MARGIN RULE THE COMPLEXITY AND NOT THE DIMENSION of input spa

## Solving the problem in the Feature-Space

Non linear function map

$$\Phi : \mathbb{R}^{m_0} \to \mathbb{R}^{m_1}$$

$$\mathbf{x} \mapsto \Phi(\mathbf{x})$$

- The problem is formulated as before, but the training set is now $T = \{(\Phi(\mathbf{x}_i), d_i)\}_{i=1}^{N}$
- The hyperplane is now $\mathbf{w}^T\Phi(\mathbf{x}) + b = 0$

Incorporating the bias in the weight vector:

- $w(0) = b$
- $\Phi_0(\mathbf{x}) = 1$

$$\Phi(\mathbf{x}) = (\phi_0(\mathbf{x}) = 1, \phi_1(\mathbf{x}), \cdots, \phi_{m_1}(\mathbf{x}))^T$$

Decision surface equation: $\mathbf{w}^T\Phi(\mathbf{x}) = 0$ i.e. a linear **expansion of the basis functions** (where (i) the dimension of the enlarged space is allowed to get very large and (ii) the complexity dependes from the margin, not directly on the space dimension)

⇓

VC-DIMENSION ↗ INCREASE ACCORDING TO THE Feature space

## Linear hyperplane in feature space

The weight vector is a linear combination of the feature vectors:

$$\mathbf{w} = \sum_{i=1}^{N} \alpha_i d_i \Phi(\mathbf{x}_i)$$

Thus the hyperplane equation is:

$$\sum_{i=1}^{N} \alpha_i d_i \underbrace{\Phi^T(\mathbf{x}_i)\Phi(\mathbf{x})}_{\text{dot product}} = 0$$

Note the dot product is now in the feature space

Evaluating $\Phi(\mathbf{x})$ could be intractable

## Kernel trick

Fortunately under certain conditions we do NOT need to evaluate directly $\Phi(\mathbf{x})$.

We do NOT even need to know the feature space itself!

This is possible using a function to compute directly the dot products in the feature space

Kernel: $k : \mathbb{R}^{m_0} \times \mathbb{R}^{m_0} \to \mathbb{R}$

k is called *inner product kernel function*

: $k(\mathbf{x}_i, \mathbf{x}) = \Phi^T(\mathbf{x}_i)\Phi(\mathbf{x})$

argument K( DIRECTIONS) ⇒ compute the . product btw $\Phi^T(\mathbf{x}_i)\Phi(\mathbf{x})$

Property: a kernel function is a symmetric function $k(\mathbf{x}_i, \mathbf{x}) = k(\mathbf{x}, \mathbf{x}_i)$

The kernel function provides us a powerful abstraction tool.

## Primal problem in feature space

SOFT MARGIN AND KERNEL

### Primal Problem

Given the training set $T = \{(\Phi(\mathbf{x}_i), d_i)\}_{i=1}^N$ find the optimal value of $\mathbf{w}$ which minimizes the objective function

$$\Psi(\mathbf{w}, \xi) = \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_{i=1}^N \xi_i$$

under the constraints

$$d_i(\mathbf{w}^T\Phi(\mathbf{x}_i)) \geq 1 - \xi_i \; \forall i = 1, \cdots, N$$

$$\xi_i \geq 0 \; \forall i = 1, \cdots, N$$

Please note that **w is now in the feature space and thus its dimensionality** can lead to **an intractable problem**

w depends on dimensional $\Phi$
$\Phi$ can very high dimensional space

## Dual Problem in feature space

### Dual Problem

Given the training set $T = \{(\Phi(\mathbf{x}_i), d_i)\}_{i=1}^N$ find the optimal values of $\{\alpha_i\}_{i=1}^N$ which maximize the objective function

$$Q(\alpha) = \sum_{i=1}^N \alpha_i - \frac{1}{2}\sum_{i,j=1}^N \alpha_i\alpha_j d_i d_j \overbrace{k(\mathbf{x}_i, \mathbf{x}_j)}^{K_{i,j}}$$

NOT SURE

satisfying the constraints

$$\sum_{i=1}^N \alpha_i d_i = 0$$

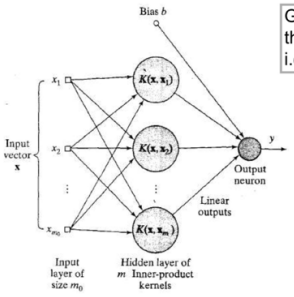
$$0 \leq \alpha_i \leq C \; \forall i = 1, \cdots, N$$

Remember C is a user specified non-negative parameter (regularization parameter) The kernel includes the mapping to the feature space. The dependences on $N$ becames now an advantage (due to the possible high dim of the feature space)



$\frac{1}{2\sigma^2} = \gamma$

## Architecture view of a SVM




Go back to see the slides on the NN as (a special kind, i.e. adaptive, of) **LBE**

Figure: "Architecture" of a SVM.

- Number of hidden units = Number of support vectors
  (... but not exactly the same unit, e.g. hidden units in NN have adpative free parameters)!

This is just a "picture" view of the SVM. Think by yourself on the comparison in terms of complexity control, efficiency, and other characteristics.



## Some examples of kernels

- Polynomial Learning Machine: $k(\mathbf{x}, \mathbf{x}_i) = (\mathbf{x}^T\mathbf{x}_i + 1)^p$ where $p$ is a user specified parameter
- Radial Basis Function Net: $k(\mathbf{x}, \mathbf{x}_i) = e^{-\frac{1}{2\sigma^2}\|\mathbf{x}-\mathbf{x}_i\|^2}$ where $\sigma^2$ is a user specified parameter A.k.a. Gaussian kernel
  [narrow peaked Kernels (small sigma) imply that reply for $\mathbf{x}_i$ is only $d_i$ (what does it resemble?)]
- Two-layer perceptron $k(\mathbf{x}, \mathbf{x}_i) = tanh(\beta_0\mathbf{x}^T\mathbf{x}_i + \beta_1)$ where $\beta_0 (> 0)$ and $\beta_1 (< 0)$ are user specified parameters

### Important

For polynomial learn. machines and RBN you always get an inner product kernel, while for a two-layer perceptron the Mercer's th. only holds for some choices of $\beta_0$ and $\beta_1$

### Note

Also note that using a kernel function of RBF type always leads to a feature space with an infinite number of dimensions

SVM: non-linear regression

regression: function that describes the dependency of A SCALAR from a VECTOR

$$d = f(x) + v$$

unknown function ↑ noise

- Training set $\{(x_i, d_i)\}_{i=1}^N$

- estimate the target $d$ using a LINEAR EXPANSION of non-linear functions

$\{\phi_j(x)\}_{j=0}^{m_1}$

$$y = h(x) = w^T\phi(x)$$ where $w = (w(0)=b, w(1), \ldots, w(m_1))^T$

TRANSFORMATION ← $\phi(x) = (\phi_0(x)=1, \phi(1)=x_1, \ldots, \phi_{m_1}(x))^T$

We have a function that describes the dependency of a scalar from a vector:

$$d = f(\mathbf{x}) + v$$

where:

- the function f is unknown
- the noisy term $v$ is statistically independent of the input vector $\mathbf{x}$

We only have a training set $T = \{(\mathbf{x}_i, d_i)\}_{i=1}^{N}$

We estimate $d$ using a linear expansion of non-linear functions $\{\phi_j(\mathbf{x})\}_{j=0}^{m_1}$

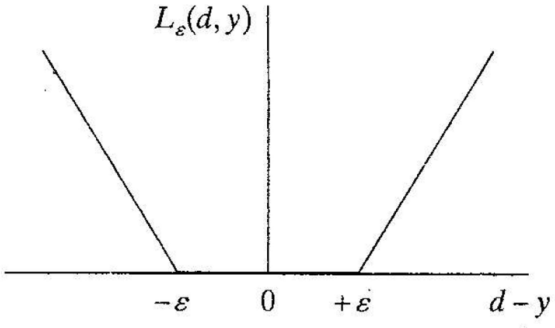
$$y = h(\mathbf{x}) = \mathbf{w}^T \Phi(\mathbf{x})$$

Where: $\mathbf{w} = (w(0) = b, w(1), \cdots, w(m_1))^T$

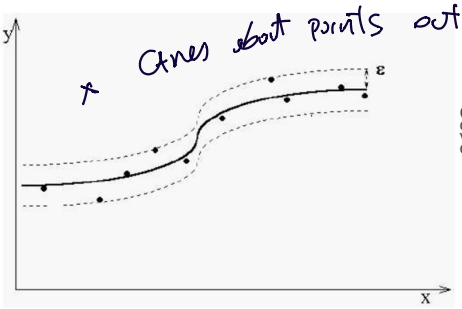
$\Phi(\mathbf{x}) = (\phi_0(\mathbf{x}) = 1, \phi_1(\mathbf{x}), \cdots, \phi_{m_1}(\mathbf{x}))^T$

## $\epsilon$-insensitive loss function

$$L_\epsilon(d, y) = \begin{cases} |d - y| - \epsilon & \text{if } |d - y| \geq \epsilon \\ 0 & \text{otherwise} \end{cases}$$




## $\epsilon$-tube




Close points do not care, only support vectors care (as for classification)

Figure: $\epsilon$-tube around d.

Note: again only the support vectors guide the solution, as the cost function ignores any training data close to the model prediction, i.e. within a threshold $\epsilon$ (for class. were for points that lie beyond the margin)

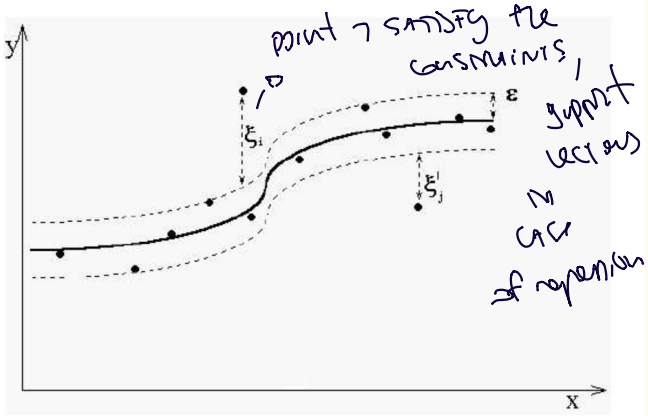



Figure: $\epsilon$-tube and slack variables.

## Formulating the optimization problem

Introduce the non-negative slack variables $\xi_i'$ and $\xi_i$ for all i=1,...,N

$$-\xi_i' - \epsilon \leq d_i - \mathbf{w}^T \Phi(\mathbf{x}_i) \leq \epsilon + \xi_i \; \forall i = 1, \cdots, N$$

This leads to the following constraints:

$$\left.\begin{array}{l} d_i - \mathbf{w}^T \Phi(\mathbf{x}_i) \leq \epsilon + \xi_i \\ \mathbf{w}^T \Phi(\mathbf{x}_i) - d_i \leq \epsilon + \xi_i' \\ \xi_i \geq 0 \\ \xi_i' \geq 0 \end{array}\right\} \forall i = 1, \cdots, N$$

## Primal Problem

### Primal Problem

Given the training set $T = \{(\mathbf{x}_i, d_i)\}_{i=1}^{N}$ find the optimal values of $\mathbf{w}$ such that the following objective function is minimized

$$\Psi(\mathbf{w}, \xi, \xi') = \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_{i=1}^{N}(\xi_i + \xi_i')$$

under the constraints

$$\left.\begin{array}{l} d_i - \mathbf{w}^T \Phi(\mathbf{x}_i) \leq \epsilon + \xi_i \\ \mathbf{w}^T \Phi(\mathbf{x}_i) - d_i \leq \epsilon + \xi_i' \\ \xi_i \geq 0 \\ \xi_i' \geq 0 \end{array}\right\} \forall i = 1, \cdots, N$$

## Computing the weight vector

Solving the dual problem we obtain the optimal values of the Lagrangian multipliers $\{\alpha_i\}_{i=1}^{N}$ and $\{\alpha_i'\}_{i=1}^{N}$

Then we can compute the optimal value of vector $\mathbf{w}$:

$$\mathbf{w} = \sum_{i=1}^{N} \underbrace{(\alpha_i - \alpha_i')}_{\gamma_i} \Phi(\mathbf{x}_i)$$

$$\Updownarrow$$

$$\mathbf{w} = \sum_{i=1}^{N} \gamma_i \Phi(\mathbf{x}_i)$$

### Support vector

In this case support vectors correspond to non-zero values of $\gamma_i$

## Computing the estimate

The estimate function is defined as $h(\mathbf{x}) = y = \mathbf{w}^T \Phi(\mathbf{x})$

Using the linear expansion for $\mathbf{w}$ we get:

$$h(\mathbf{x}) = \sum_{i=1}^{N} \gamma_i \Phi^T(\mathbf{x}_i)\Phi(\mathbf{x}) = \sum_{i=1}^{N} \gamma_i k(\mathbf{x}_i, \mathbf{x})$$

BIAS-VARIANCE: investigo gli errori a un punto guardando + realizzazioni del TR set
- la decomposizione dell'errore (su + TR) in un punto x; 3 COMPONENTI ALCUNI

1) BIAS: quantifica la discrepanza tra TRUE FUNCTION e $h(x)$
(se H è piccolo, il BIAS è ALTO) (MEAN $h(x)$ over possible TR sets)

2) VARIANCE: QUANTIFICA la VARIABILITÀ della risposta del modello h per diverse realizzazioni del training ~~SET~~ (dato all'elevata flessibilità)
=> flexible model if change the set drastically ≠ hypothesis

3) NOISE: le label includono un RANDOM error (if given x there are more possible d, targets)

- ensure Regression case

→ — BIAS-VAR ANALYSIS

- given a data point x WHAT IS the EXPECTED PREDICTION ERROR? ERROR in that point averaged over possible realizations of TR set

$$E_p[y - h(x)]^2 \quad \Rightarrow \text{DECOMPOSE IT IN THE 3 COMPONENTS}$$

AVG when you change the TR set

- let be Z una random variable con valori $z_i = 1, \dots, l$ with prob. distr. $P(z)$
- la MEDIA (o expected value): $\bar{z} = E_p[z] = \sum_{i=1}^{l} z_i P(z_i)$
- la VARIANZA: $VAR[z] = E[z - \bar{z}]^2 = E[z^2] - \bar{z}^2$

lemma: $E[z^2] = \bar{z}^2 + VAR[z]$

$$E_p[y - h(x)]^2 = E_p[h(x)^2 - 2yh(x) + y^2] =$$
$$= E_p[h(x)^2] + E_p[y^2] - 2E_p[y]E_p[h(x)] =$$

- let $\bar{h}(x) = E_p[h(x)]$

lemma $E_p[h(x)^2] = E_p[(h(x) - \bar{h}(x))^2 + \bar{h}(x)^2$

$z = h(x)$

$E_p[y^2] = E_p[(y - f(x))^2] + f(x)^2$

cuz $E_p[y] = f(x) = E_p(f(x) + \underbrace{\epsilon}_{0})]$

$$= E_p[h(x) - \bar{h}(x))^2 + \bar{h}(x) - 2E_p[y]E_p[h(x)] +$$
$$+ E_p[(y - f(x))^2] + f(x)^2 =$$
$$= E_p[(h(x) - \bar{h}(x))^2] + \quad \text{VARIANCE}$$
$$+ (\bar{h}(x) - f(x))^2 + \quad \text{BIAS}^2$$
$$+ E_p[(y - f(x))^2] \quad \text{noise}^2$$

$= VAR[h(x)] + BIAS[h(x)]^2 + E_p[\varepsilon^2] =$

$= VAR[h(x)] + BIAS[h(x)]^2 + \sigma^2$

EXPECTED PREDICTION ERROR = VARIANCE + BIAS$^2$ + NOISE$^2$

ENSABLE LEARNING: bagging and boosting

IDEA: 25 ≠ TR set or ≠ RAND INITIALIZATION of network
=> ≠ models with high VARIANCE
BUT the OUTPUT MEAN is good!
(not the mean error)
=> ENSAMBLING

Recall Regularization:

$$\text{Loss}(\mathbf{w}) = \sum_p (d_p - o(\mathbf{x}_p))^2 + \lambda \|\mathbf{w}\|^2$$

$\lambda \uparrow$ => BIAS ↑, VARIANCE ↓

Left (data) term (TR data) Penalty term

Varying λ we can obtain
- less regularized (complex) solutions (low λ)
- more regularized (less complex) solutions (high λ)

•The following example (from Bishop) (should be) using RBF network with LMS

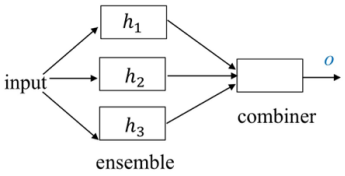
(By the way, note: that the intercept is typically left out of the penalty term)

20

## Ensemble Learning

Dip. Informatica
University of Pisa

- A first look to **ensemble learning**
- Take advantage of multiple models




Simplest case ("voting schema"):

- Regression
  - simple average committee $o(x) = 1/K \sum_{i=1}^{K} h_i(x)$
  - $loss(E_i(h_i)) \le E_i(loss(h_i))$ I.e.
  - ($mean_D$) square error of a committee ≤ mean of the ($mean_D$) square error
    e.g. $(t-o)^2$, t=5, 2 models → $(5-3)^2 = 4 < ((5-2)^2+(5-4)^2)/2 = 5$
    or even: $(5-5)^2 = 0$ while $((5-6)^2 + (5-4)^2)/2 = 1$
- Classification
  - Take a vote over many classifier (given they provide "diversified" responses): why ? (or classify after the mean of the continuous outputs)

Alternative ("stacking schema"): the combiner is again a ML model

A. Micheli

27

- SEND ALL the outputs to a combiner
- take the mean
- produce output

. the loss of the committee (mean of $h_i$) is always ≤ mean loss of each reg/class

$Loss(mean(h_i)) \le AVG_i(Loss(h_i))$

# SOM - K-MEANS

- UNSUPERVISED NN : no Teacher
- TR = <d>
- clustering → VECTOR QUANTIZATION for partitioning clustering
- dimensional reduction / visualization (PCA)

- SIMILARITY MEASURE : dipende dalla metrica (es. euclidian distance)

- VECTOR QUANTIZATION : Tecnica per l'ENCODING di DATA MANIFOLD (es 3D projected on 2D) (riduzione le dimensioni)
  - un submanifold $V \subseteq R^N$ using un finite set $w = (w_1, \dots, w_K)$ of reference vectors $w_i \in R^M$ (use just finite of this. prototype per rappresentare tutti i dati)
  - un data vector $x$ è il BEST MATCHING o WINNING reference vector $w_{i^*(x)}$ (*: I'm the winner for $x$ e $w$ è il reference vector (cluster center da rappresentare $x$)) of $w$ per cui il DISTORTION error $d(x, w_{i^*(x)})$ è MINIMA (il centroide più vicino)
  - questa PROCEDURA DIVIDE il MANIFOLD $V$ in sub-regioni $V_i$ → $V_i = \{ x \in V \text{ s.t. } \|x - w_i\| \leq \|x - w_j\| \ \forall j \}$

- QUANTIZATION: form continuous to discrete values
- CLUSTERING has interest / useful group of data (→ always reduce the ~~DEFINITION~~ QUANTIZATION error
- (GOAL): PARTIZIONE OTTIMA DI UNA DISTRIBUZIONE SCONOSCIUTA DI $x$-input space in regions (cluster) approssimate da cui cluster CENTROID (= prototype)
  - ASSEGNO ogni punto al suo cluster center
    $x \to c(x)$ or $w_{i^*(x)}$ (prototype)
  - considero SQUARED ERROR DISTORSION
    $d(x_i, c(x_i)) = \|x_i - c(x_i)\|^2$ (w: prototype vectors)
    ↓ prototype center

$$E = \int f(d(x, w_{i^*(x)})) \cdot p(x)\,dx = \int \|x - w_{i^*(x)}\|^2 p(x)\,dx$$

# Quantization error function

$$F = \sum_i^l \sum_j^K \|x_i - w_j\|^2 \delta_{winner}(i,j)$$

Dip. Informatica
University of Pisa

$$E = \int f(\mathrm{d}(\boldsymbol{x}, \boldsymbol{w}_{i^*(\boldsymbol{x})}))\, p(\boldsymbol{x})\, d\boldsymbol{x} = \int \left\| \boldsymbol{x} - \boldsymbol{w}_{i^*(\boldsymbol{x})} \right\|^2 p(\boldsymbol{x})\, d\boldsymbol{x}$$

from continuous to discrete

(squared error criterion)

$$E = \sum_i^l \sum_j^K \left\| \boldsymbol{x}_i - \boldsymbol{w}_j \right\|^2 \delta_{winner}(i,j)$$

Discrete version

$\delta_{winner}(i,j)$ characteristic function of the receptive field of $\boldsymbol{w}_j$
1 if $j$ is the winner for $\boldsymbol{x}_i$, 0 otherwise

$p(\boldsymbol{x})$ is the probability distribution of $\boldsymbol{x}$

$$E = \int f(d(x, w_{i^*(x)})) \cdot p(x)\,dx = \int \|x - w_{i^*(x)}\|^2 p(x)\,dx =$$
$$= \sum_i^l \sum_j^K \|x_i - x_j\| \cdot \delta_{winner}(i,j)$$

- need $\delta_i$ that is receptive field of $w_j$ -> 1 if $j$ is winner for $x_i$
  0 else
  (cut to an end so not cluster