Fundamentals of artificial neural networks

Franco Scarselli

▶ DEPARTMENT OF INFORMATION ENGINEERING AND MATHEMATICS

UNIVERSITÀ DI SIENA 1240

OUTLINE

- (Very) short introduction to perceptron neural networks
- Aproximation capability
- Learning capability
- Generalization capability
- Autoencoders
- Recurrent neural networks
- Deep networks

UNIVERSITÀ DI SIENA 1240

Machine learning

Difficult problems in computer science

- Machine vision, automatic drug design, speech understanding, machine translation, …

- Nobody can write a program that solve them
  - humans cannot solve them or
  - humans are used to solve them, but …. they do not know how they do!

UNIVERSITÀ DI SIENA 1240

Machine learning

Can you describe how you recognize an apple in images?

- It just looks like a red circle

Red?

UNIVERSITÀ DI SIENA
1240

Machine learning

Red circle?

UNIVERSITÀ DI SIENA 1240

Machine learning

If you cannot write a program that solves a problem

…. let computers learn the solution!!

- By examples
  - e.g. examples of images containing or not containing apples
- In most of the cases, humans and animals learn to solve problems by examples

UNIVERSITÀ DI SIENA 1240

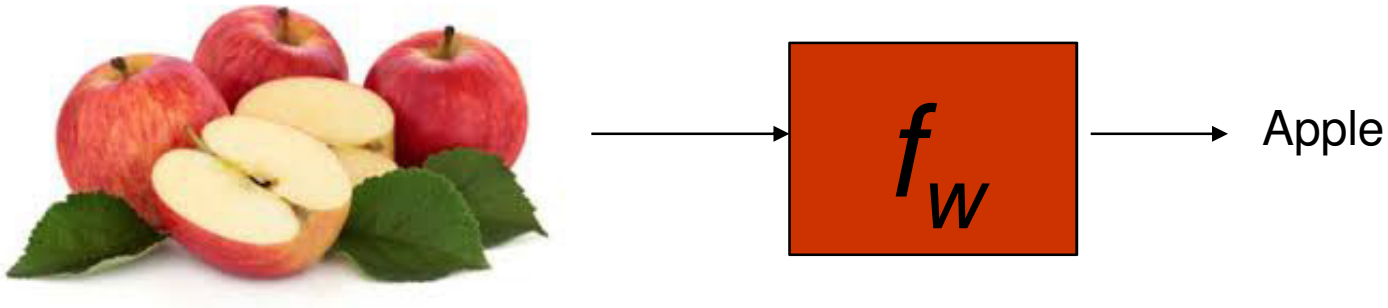
Machine learning: models

Consider a parametric model f_w

- f_w takes in input a pattern represented by vector $z=[z_1,..z_n]$
- f_w returns an output vector $o=[o_1,..,o_m]$

Example

- Input images: $[z_1,..z_n]$ are the pixels
- Output: o=[0,..0,1,0,…0,0]
  one hot coding of a set of objects
  a one in i-th positions represents i-th object



UNIVERSITÀ DI SIENA 1240

Machine learning: problems

Classification problems

- the pattern has to be assigned a class in a finite set
- the output o
  - two classes: o=1 or o=0 according to the class
  - several classes: o=[0,…,1,…0], (one hot coding)
- Example: recognized the object represented by an image

Regression problems

- the pattern has to be assigned a set of (real) numbers
- Example: returns the probability that object represented by an image is a cat

UNIVERSITÀ DI SIENA 1240

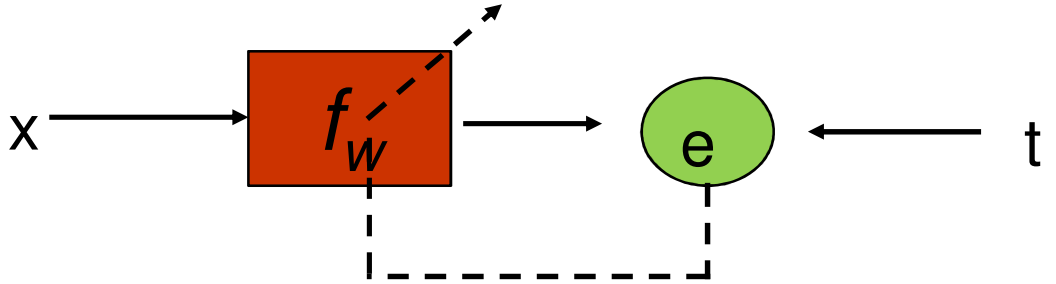
Machine learning: supervised training

Supervised dataset

- A set of pairs $D=\{(x_1,t_1), (x_k,t_k)\}$ is a set of pairs pattern-target
- Usually split in
  - Train set L: for training the parameters
  - A validation set V: to adjust other parameters…..
  - A test set T: to measure the expected performance of the trained model

Training

- Define an error function e_w based on train set
- Optimize e_w by some optimization algorithm



UNIVERSITÀ DI SIENA 1240

Machine learning: error functions

Mean square error

$$e_w = \frac{1}{k}\sum_i \|t_i - fw(xi)\|^2$$

- the most one
- both for classification an regression problems

Cross entropy

$$e_w = \sum_i \sum_j t_{ij} \log(f_w(xij))$$

- often used in deep learning
- only for classification problems

UNIVERSITÀ DI SIENA 1240

Machine learning: measuring performance

- The performance on test set: it depends on the problem
- Mean square error and cross entropy
  - but usually not what we want
  - Training error is often different from test error!!
- Classification problems
  - Accuracy, F1, ROC AUC,…
- Regression
  - Relative error, …
- Ranking problems
  - profit, MAE, …
- ….

UNIVERSITÀ DI SIENA 1240

Artificial neural networks (ANNs)

A class of machine learning models inspired by biological neural networks

- A set of simple computational units (neurons)
- Neurons are connected by a network
- The behavior of the network depends on the interactions among neurons
- The connectivity is learned

UNIVERSITÀ DI SIENA 1240

Artificial neural networks (NNs)

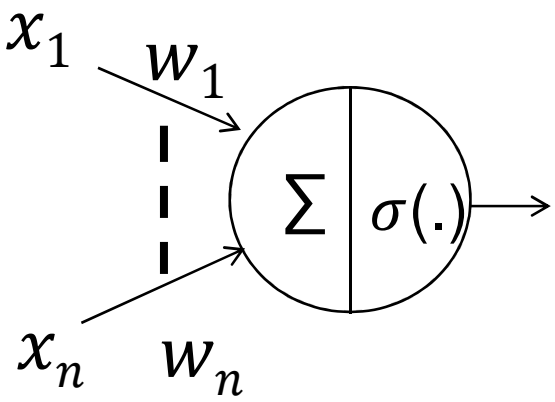
Ridge neurons

- In the most common case, each neuron has
  - a set inputs $x_1, \ldots, x_n$
  - a set weights $w_1, \ldots, w_n$

- The neuron computes
  - an activation level a $= \sum_i w_i x_i$
  - an output level o$= \sigma(a)$
  - σ is called activation function

$$y = \sigma\left(\sum_{i=1}^{d} v_i x_i + b\right) = \sigma(vx + b)$$

Vectorial formulation

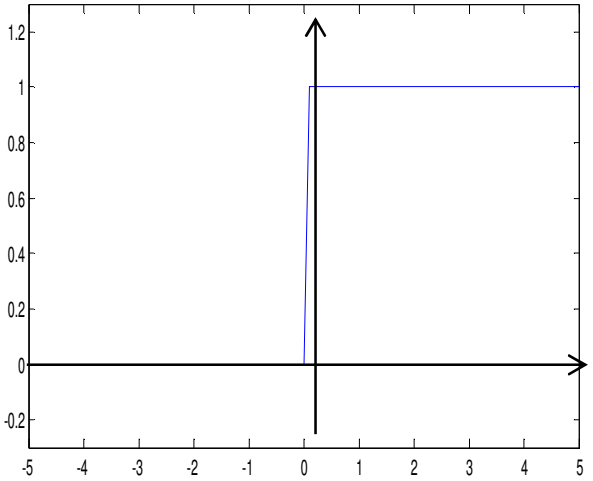


UNIVERSITÀ DI SIENA 1240

Types of neurons

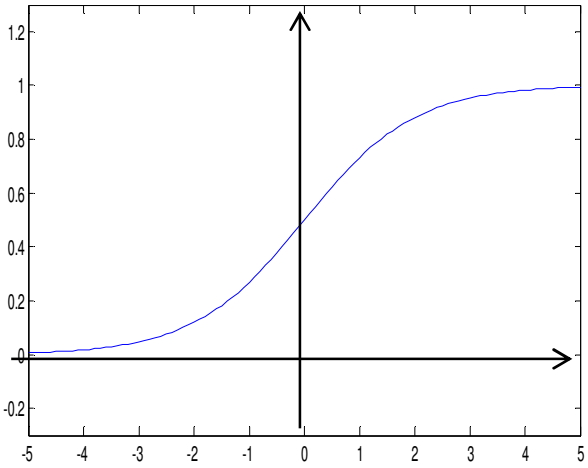
Step

- Also called heavy-side
- It takes a “hard decision”
- rarely used in practice, since
  - bad: It is not continuous
  - bad: Its derivative is 0 everywhere



Sigmoidal

- e.g. tanh, arctan, logsig
- they take a “soft decision”
- the most used in (old) neural networks
  - Good: continuous and non-zero derivative
  - Bad: derivative is zero in practice for very large and small inputs

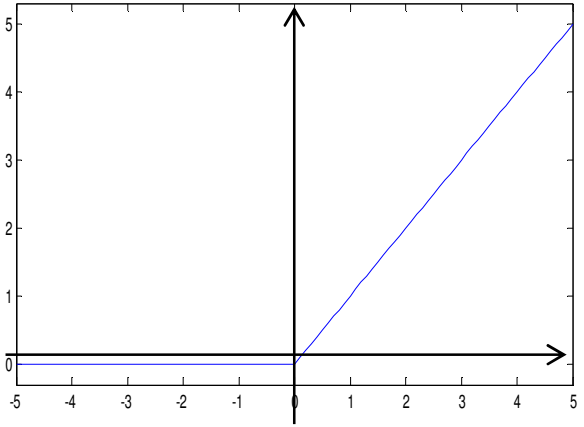


UNIVERSITÀ DI SIENA 1240

Types of neurons

Piecewise linear

- Rectifier Linear unit (ReLu), leaky ReLU
- It transmits the signal for positive values
- used in modern deep neural network
  - Bad/good: its derivatives is 0 for negative inputs
  - Bad/good: no upper bound

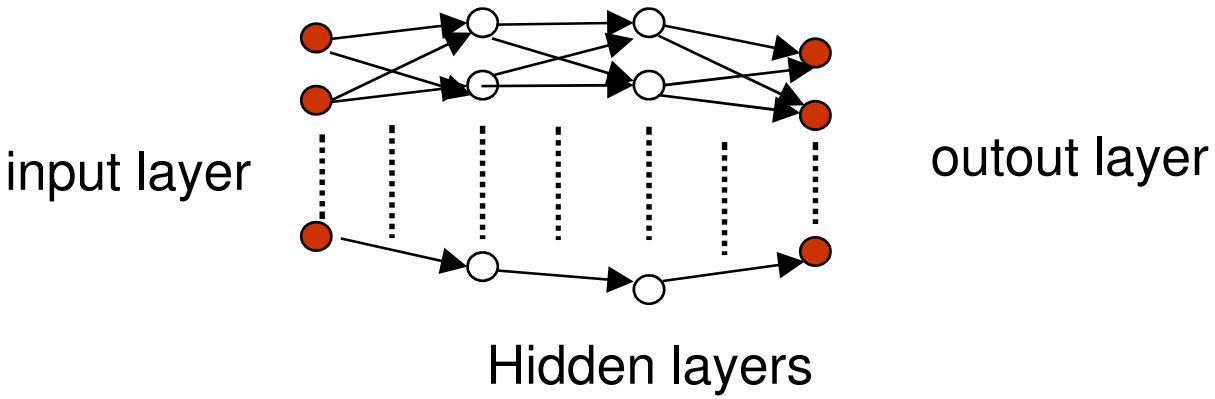


UNIVERSITÀ DI SIENA 1240

Multilayer feedforward neural networks (FNNs)

Multilayer perceptrons… also called back propagation networks… also called feedforward neural networks

- it is one of the oldest network models
- Neurons are disposed in layers: inputs, hiddens, outputs
  - The neurons of each layer take in input the outputs of the neurons of the previous layer
  - No connection is allowed intra-layer and between non consecutive layers

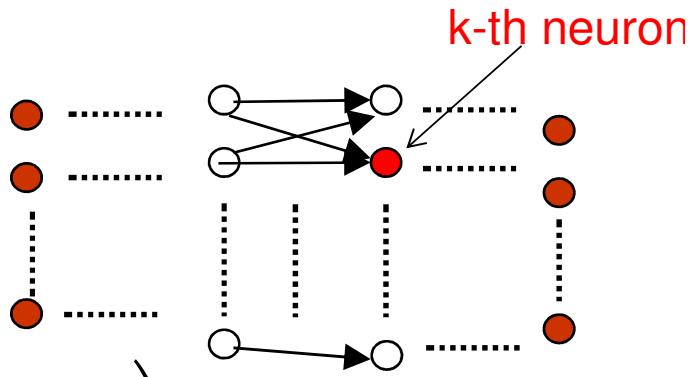


Multilayer feedforwa neural networks (FNNs)

Mulilayer networks

- each neuron takes in input the output of other neurons
- **a complex behaviour emerges from the simple activity of each neuron**
- The k-th neuron in the l-th layer has a bias b_l^k and weights $w^l_{(1,k)}, .., w^l_{(dl-1,k)}$
- its output y^l_k is

$$y_k^l = \sigma\left(\sum_{i=1}^{d_{l-1}} w_{(i,k)}^l y_i^{l-1} + b_k^l\right)$$



- The output y^l of the l-th layer is

$$y^l = \sigma(W^l y^{l-1} + bl) = \sigma\left(\sum_{i=1}^{d_{l-1}} w_i^l y_i^{l-1} + b_k^l\right)$$

Matrix form

Vector form

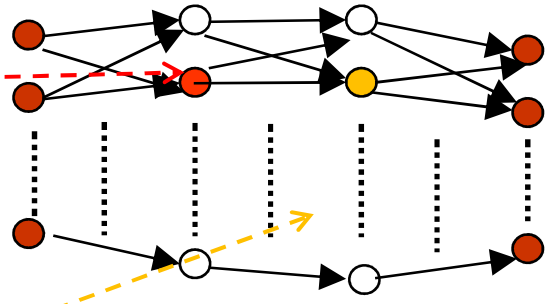
UNIVERSITÀ DI SIENA 1240

Multilayer feedforwa neural networks (FNNs)

Multilayer networks

- each neuron takes in input the output of other neurons
- **a complex behaviour emerges from the simple activity of each neuron**
- e.g., the output of the first layer is

$$y^1 = \sigma(W^1 x + b^1)$$



- e.g., the output of the second layer is

$$y^2 = \sigma(W^2 \sigma(W^1 x + b^1) + b^2)$$

- e.g., the output of the third layer is

$$y^3 = \sigma(W^3 \sigma(W^2 \sigma(W^1 x + b^1) + b^2) + b^3)$$

UNIVERSITÀ DI SIENA 1240

Interesting theoretical properties of NN

- Approximation capability
  The capability of NN model of approximating a target function
- Generalization capability
  The capability of a trained NN to generalize to novel unseen patterns
- Optimal learning
  The capability of training algorithm to produce the optimal patterns avoiding local minima

UNIVERSITÀ
DI SIENA
1240

Approximation capability

UNIVERSITÀ DI SIENA 1240

Practical question: approximation capability

- What type of applications can be implemented by a FNN?
- Are FNNs limited in some sense?

Answer (ver 1.0)

- FNNs are universal approximators, so that they can implement any application!

- Let us understand better the answer
- Let us understand the limits of such answer

UNIVERSITÀ DI SIENA 1240

Approximation capability

- Given a target function t, a precision ε, a norm ||.||, is there a NN such that implements a function for which

$$||t - f|| \leq \epsilon$$

holds?

- The intuitive answer is almost always yes (By Cybenko; Hornik et alt.; Funahashi)
  - True for most target functions t: continuous function, discontinues but measurable, ….
  - True for most of the norms: sup norm, Euclidean norm, integral norms, …
  - True for most of the feedforward NN with ridge and gaussian activation functions, sigmoidal, Relu, ….

UNIVERSITÀ DI SIENA 1240

Approximation capability for continuous functions

Let

- $\Sigma^3{}_\sigma$ be the set of functions be implementable by a FNNs with activation function σ and 3 layers (one hidden layer)

$$\Sigma^3{}_\sigma = \{f(x)|v\ \sigma(Wx + b_1) + b_2\}$$

- σ be a sigmoidal activation function
- C be the set of continuous function
- $||.||_\infty$ be the sup norm, namely for two functions

$$\left|\left|f\right|\right|_\infty = \max_x |f(x)|$$

Theorem $\Sigma^3{}_\sigma$ is dense in C i,.e.,for any ε, any $t \in C$, there is a $f \in \Sigma^3{}_\sigma$ such that

$$\left|\left|t - f\right|\right|_\infty \leq \epsilon$$

UNIVERSITÀ DI SIENA 1240

Approximation capability for continuous functions

A sketch of the proof will help to better understand NNs

Focus on

- FNNs with one hidden layer
- a single outpiut
- linear activation functions in outputs
- sigmoidal activation function inn hidden layer

$$y = W^2\sigma(W^1x + b^1) + b^2 = \sum_{i=1}^{k} w_i^2\, \sigma\left(\sum_{j=1}^{n} w_{i,}^1 x + b^l\right) + b^2$$

Two step proof

- Approximation of functions on R (single input)
- Extension to function on R^n

UNIVERSITÀ DI SIENA 1240

Sketch of the proof: single input functions

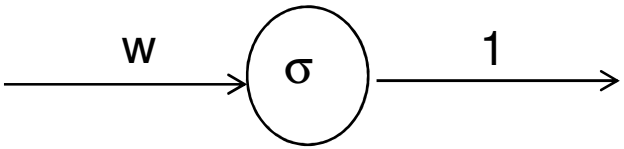
The main idea

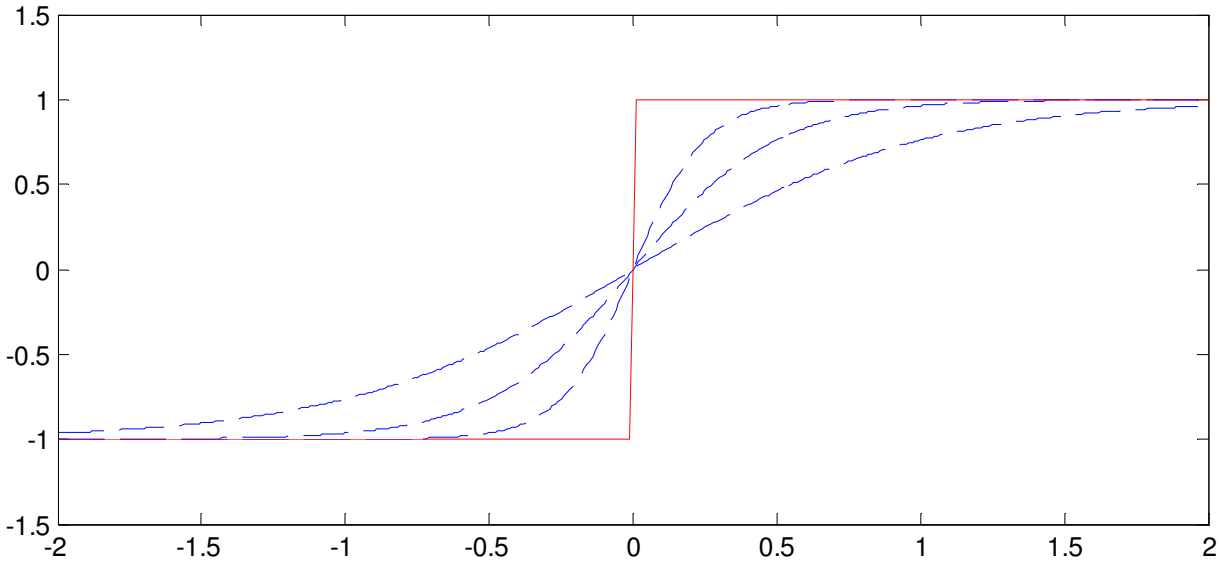
1. A single neuron implements sigmoid function
2. A sigmoid can approximate a step function
3. Many step functions can approximate a staircase function
4. Staircase function can approximate any continuous function

UNIVERSITÀ DI SIENA 1240

Sketch of the proof: single input functions

A neural network with one hidden neuron and increasing input-to-hidden weight

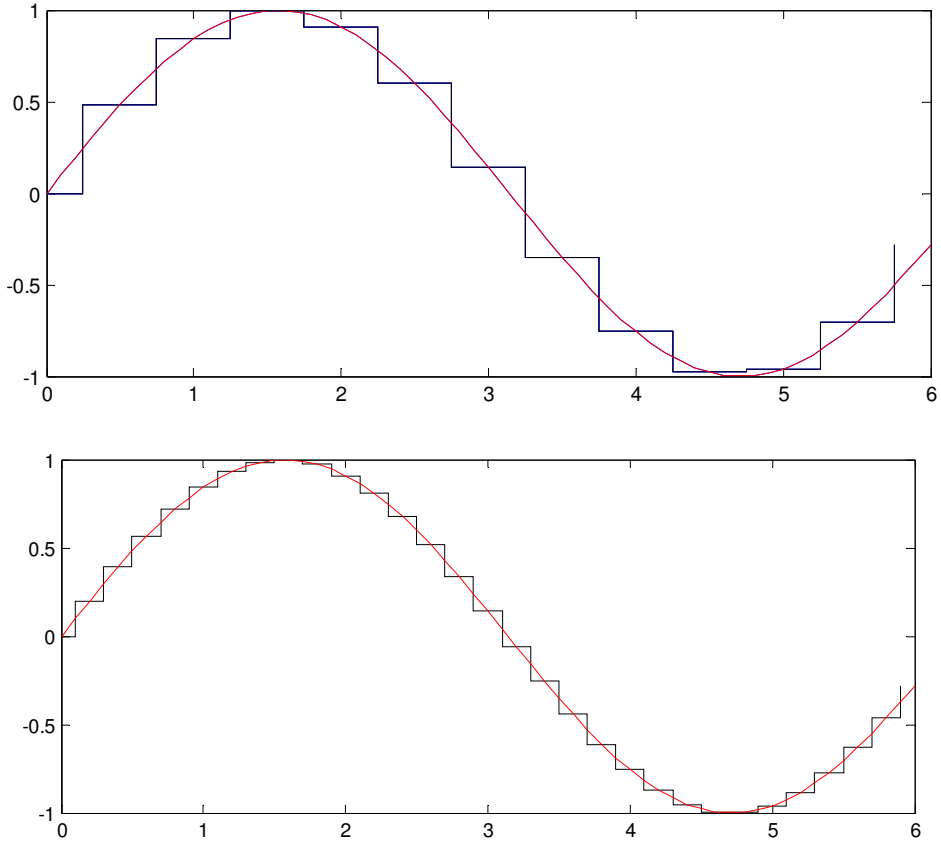




UNIVERSITÀ DI SIENA 1240

Sketch of the proof: single input functions

- A staircase function is made by many step functions
- Staircases functions can approximate any continuous function.
- The precision of approximation improves increasing the number of steps



UNIVERSITÀ DI SIENA 1240

Sketch of the proof: another approach

The main idea

1. For any polynomial p, we can construct FNNs with analytic activation function that approximates p

2. Polynomials can approximate any continuous function

UNIVERSITÀ DI SIENA 1240

Sketch of the proof: approximating polynomials

Analytic functions … let us remember what is

- A function is analytic if for each x0 its Taylor series converges to f in a neighborhood of x0

$$A \lim_{x \to \infty} {S_{x0}}^{T}(x) = f(x)$$

- Taylor series of a function f developed up to T term computed in x0 with rest

$${S_{x0}}^{T}(x) = \sum_{k=0}^{T} \frac{f^{(k)}(x0)}{k!}(x - x0)^{k} \; [+RT(x - x0)]$$

UNIVERSITÀ DI SIENA 1240

Sketch of the proof: approximating polynomials

Analytic functions … useful intuitive facts

- Analytic functions looks like polynomials (their Taylor series)
- Actually, they looks like polynomials except for the error RT(x-x0), which is smaller than $O((x-x0)^T)$
- Thus, a neuron with an analytic activation function looks like a polynomial…
- … then, an FNN looks like a combination of polynomials

Sketch of the proof: approximating polynomials

Go back to the original goal …

- for any polynomial $p(x)=c_0+c_1x+c_2x^{2+}+...+c_rx^r$, construct FNNs with analytic activation function that approximates p
- an FNN looks like a combination of polynomials…
- the goal is easily reached, just find the right combination …

- **Theorem. Suppose that σ is analytic in a neighborhood of x0, then $\lim_{\alpha\to 0} \mathbf{p}_\alpha(\mathbf{x}) = \mathbf{p}(\mathbf{x})$, where**

r neurons

$$p(x) = \lim_{\alpha\to 0} p_\alpha(x) = \lim_{\alpha\to 0} \sum_{l=0}^{r} \left(\sum_{k=l}^{r} (-1)^{k+l} \frac{c_k}{\alpha\sigma^{(k)}x0)} \right) \sigma(l\alpha x + x0)$$

This is a FNN

Second layer weight

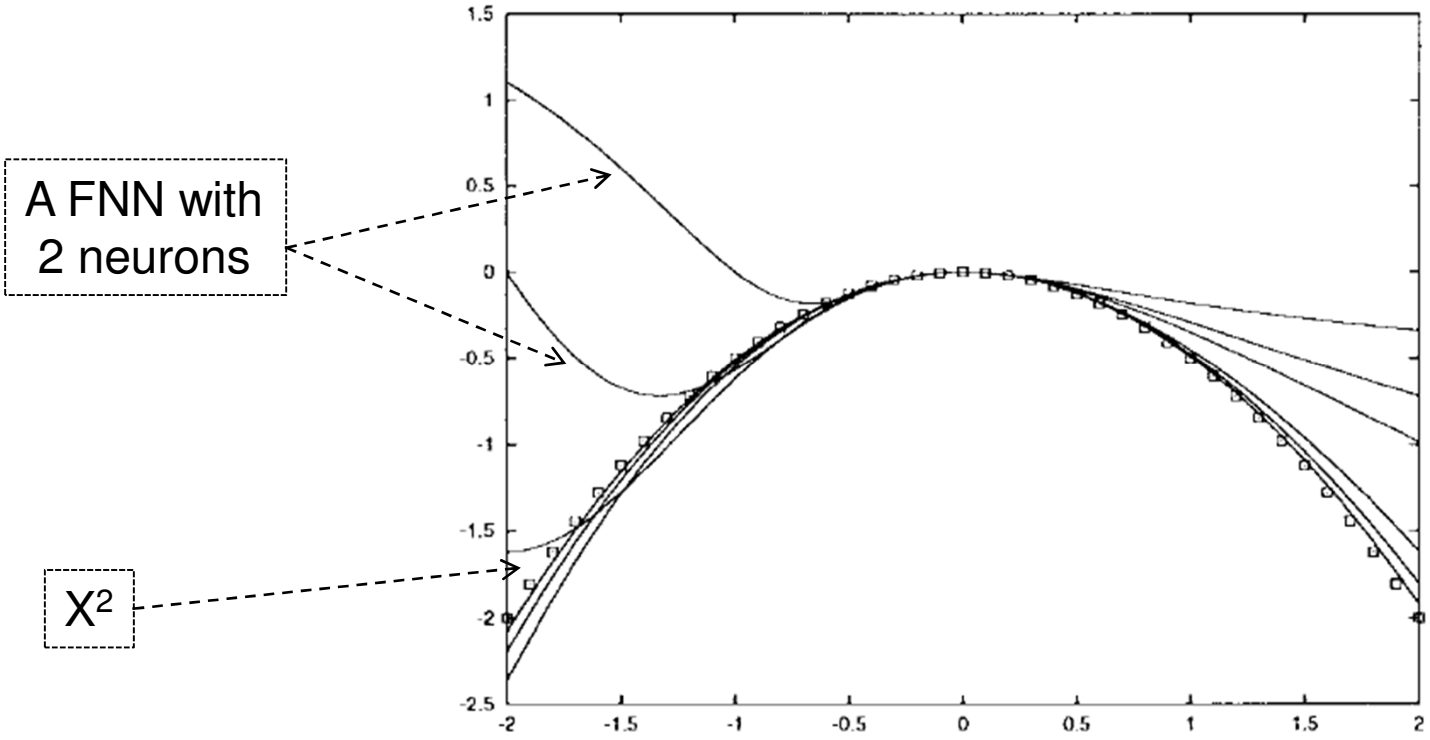
First layer weight

Hidden bias

UNIVERSITÀ DI SIENA 1240

Sketch of the proof: approximating polynomials

- **Theorem. A polynomial $p(x)=c_0+c_1x+c_2x^{2+}+...+c_rx^r$, can be approximated (up to any precision) by a FNN with r neurons!**



UNIVERSITÀ DI SIENA 1240

Sketch of the proof: functions with several inputs

How can we extend our results to functions with many inputs?

- Let us start with the case when the target function is a ridge

Ridge functions

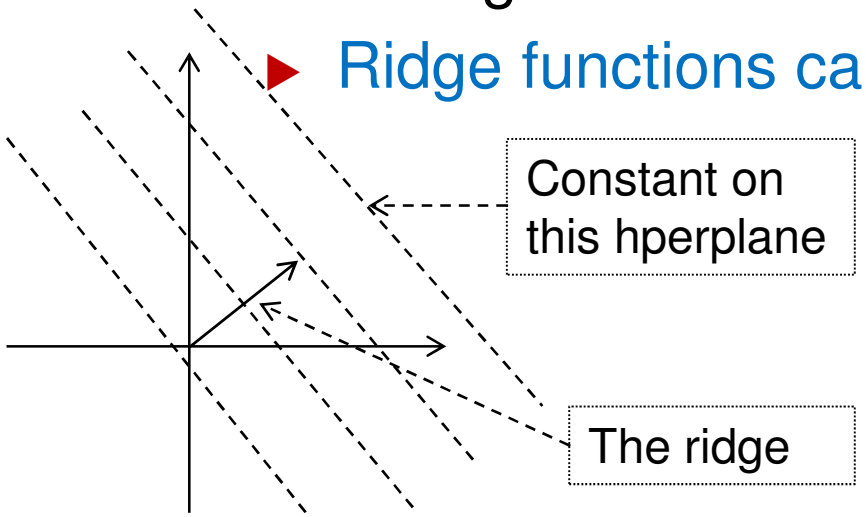
- A ridge function g:R^n->R, can be written as

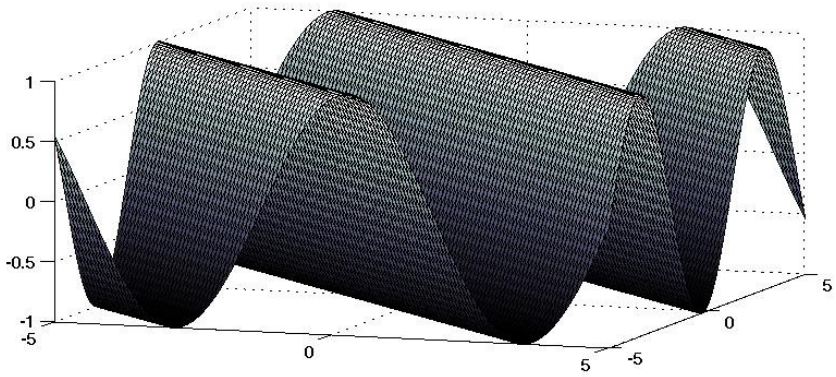
$$g(x) = h(wx)$$

where h is a function of single input

The direction of the ridge

- Ridge functions are constant on hyperplanes orthogonal to the ridge
- Ridge functions can be approximated by FNNs





UNIVERSITÀ DI SIENA 1240

Sketch of the proof: functions with several inputs

How can we extend our results to functions with many inputs?

- Just prove that

Theorem For any target function $t:R^n \to R$ and any ε, there exist ridge functions $g_1,..,g_k$, such that $\|t - f\| \leq \varepsilon$ where

$$f(x) = \sum_{i=1}^{k} g_i(x)$$

UNIVERSITÀ DI SIENA 1240

Solution 1: the Radon transform

- The Radon transform *Rf* of a function f allows to specify f in terms of their integrals over hyperplanes
- It is used in computed axial tomography (CAT scan), electron microscopy, …
- The inverse Radon transform is

$$\mathrm{f}(x) = \int_{[\![v]\!]=1} Rf(wx, w)\mathrm{dw}$$

Sum over all the hyperplanes to compute f(x)

Intuitively, it is the integral of f over the hyperlane orthogonal to w and passing through x

UNIVERSITÀ DI SIENA 1240

Solution 1: the Radon transform

- The inverse Radon transform contains an integral, which can ne approximated by finite sum

$$\mathrm{f}(x) = \int_{[\![v]\!]=1} Rf(wx, w)dw \approx \sum_{i=1}^{k} Rf(w_i x, w_i)dwi$$

- The result is a sum of ridge functions!!

This is a ridge function

UNIVERSITÀ DI SIENA 1240

Solution 2: combination of polynomials

How can we extend our results to functions with many inputs?

- Restrict you attention to polynomials and prove that

Theorem For any target polynomial $t:R^n \to R$, there exist ridge $g_1,..,g_k$, such that

$$t(x) = \sum_{i=1}^{k} g_i(x)$$

UNIVERSITÀ DI SIENA 1240

Solution 2: combination of polynomials

- Notice that the space of polynomials is related to the linear space of its parameters

- A generic polynomial with 3 variables and degree 3

$$p(x_1, x_2, x_3) = \alpha_1 x_1^3 + \alpha_2 x_1^2\, x_2^1 + \alpha_3 x_1^2 x_3^1 + \alpha_4 x_1^1\, x_2^1\, x_3^1 + \cdots$$

- Its representation as a vector in linear space

$$[\alpha_1, \alpha_2, \alpha_3, \alpha_4, \ldots]$$

- The dimension of the space of polynomials with n variables and degree r is

$$N = \binom{r + n - 1}{n - 1}$$

UNIVERSITÀ DI SIENA 1240

Solution 2: combination of polynomials

It has been proved that

- a set of ridge polynomials in the form of

$$(w_1 x + b_1)^{r'} (w_2 x + b_2)^r, (w_2 x + b_2)^r, \ldots.$$

for random v_i, b_i are, in most of the cases, linearly independent!!

- With $\binom{r+n-1}{n-1}$ random ridge polynomials you can, in most of the cases, generate the full space of polynomials n variable and degree r!!

UNIVERSITÀ DI SIENA 1240

Other solutions

- Other solutions exist, e.g. based on Fourier transform
- Not matter of this course

UNIVERSITÀ DI SIENA 1240

Possible extensions

Universal approximation

- Activation functions
  - tanh, arctan, ReLU, step, any analytic function, any step function
  - Non admitted: polynomials
- Error norm
  - Sup, L1, L2, … sobolev,…
- Arcitecture
  - Any number of inputs and outputs
  - At leas one hidden layer (included many)

UNIVERSITÀ DI SIENA 1240

Back to practice

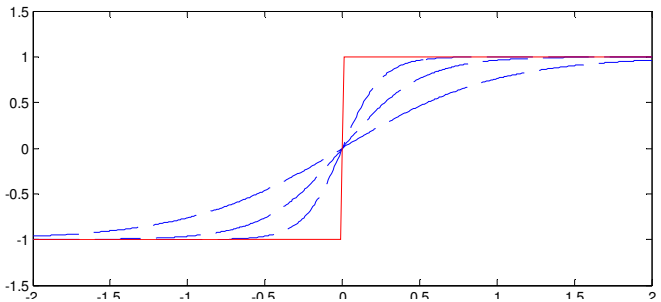
- In general, the FNNs described by results on approximation theory are not encountered in practical experiments
  - learning algorithm often produce FNNs without an intuitive explanation
- But in particular cases, the consequences of approximation theory are evident also in practice.

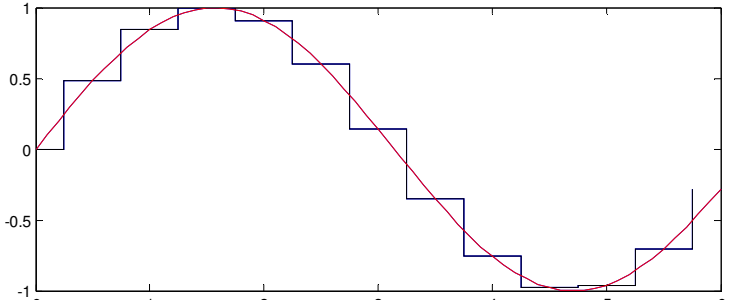
UNIVERSITÀ DI SIENA 1240

Back to practice: weight dimension

About weight dimension, from approximation by staircases

- approximation by sigmoids is reached by large weights in first layer and small weights in the second layer
- large weights produce saturation in sigmoids and make learning difficult
- ReLUs do not have saturations, but very large weights may produce large gradients
- In practical experiments,
  - such a difficulty is encountered, when the target function looks likes a staircase, e.g. it is discontinuous somewhere
  - In this case learning is difficult





UNIVERSITÀ DI SIENA 1240

Back to practice: weight dimension

About weight dimension, from approximation by polynomials

- With sgmoids approximation is reached by small weights in first layer and large weights in the second layer
- such a configuration makes the network sensitive to weight noise and makes learning difficult
- In practical experiments,
  - such a difficulty is encountered, when the target function is a polynomial and the activation function is a polynomial!!

$$p(x) = \lim_{\alpha \to 0} p_\alpha(x) = \lim_{\alpha \to 0} \sum_{l=0}^{r} \left(\sum_{k=l}^{r} (-1)^{k+l} \frac{c_k}{\alpha_k \sigma^{(k)} x0)} \right) \sigma(l\alpha x + x0)$$

UNIVERSITÀ DI SIENA 1240

Back to practice: weight dimension

Notice that

- in several applications the target function is almost polynomial (e.g., in dynamic system identification)
- common tricks to alleviate the problem
  - add input to output weights
  - use neural network in parallel with a polynomial approximation
  - Use ReLU in place of sigmoids mayalso work

UNIVERSITÀ DI SIENA 1240

Back to practice: ridge direction

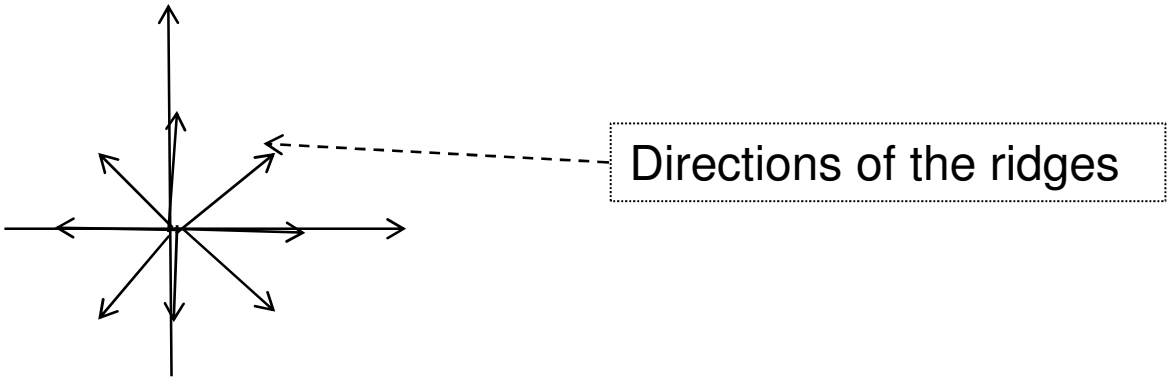
What about the directions (not dimension) of ridges $w_1^1, \ldots, w_n^1$?

$$y = \sum_{i=1}^{k} w_i^2 \, \sigma\left(\sum_{j=1}^{n} w_i^1 x + b^l \right) + b^2$$

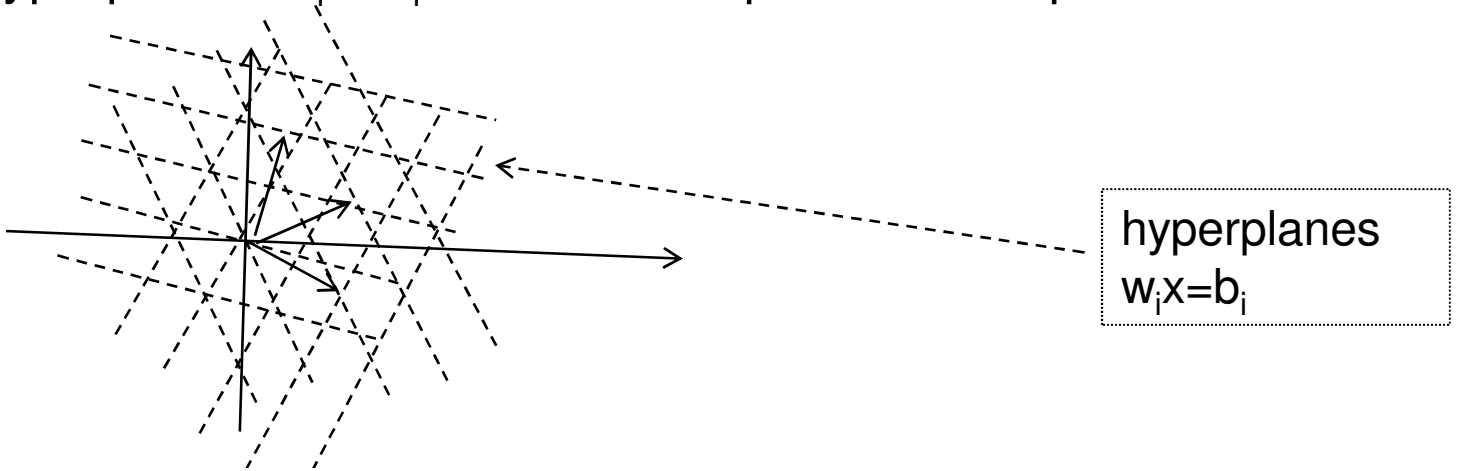
UNIVERSITÀ DI SIENA 1240

Back to practice: ridge direction

- The results from Radon transform tell us that the directions and the biases can be chosen by a grid on the ball $|w|=1$



- When combined with a selection of the biases from a grid, the hyperplanes $w_i x=b_i$ constitutes a partition of input domain !



UNIVERSITÀ DI SIENA 1240

Back to practice: ridge direction

- The results from polynomial combination suggest that ridges and the biases (the partition) can be even random!

UNIVERSITÀ DI SIENA 1240

A constructive algorithm from theory

A simple algorithm (me and Ah Chung)

1. Chose the first hidden layer weights and biases from a in a random way (or uniformly from a grid)
2. Make the first hidden layer weights very small (or very large)
3. Adapt only the send layer weights and bias to minimize the error function

Notice that

- this algorithm does not suffer from local minima!!!!

(the error is quadratic w.r.t the last layer parameters)

- It works also for many outputs

UNIVERSITÀ DI SIENA 1240

The error is quadratic w.r.t the last layer parameters

Let us look at a FNN formulas using matrices

The dataset

- $\{(x_1,t_1), \ldots (x_k,t_k)\}$ a set of patterns
- $X = [x_1, .. xk\,]$ Set of Inputs in matricial form
- $T = [t_1, .. t_k]$ Set of targets in matricial form

The network output Y

- $H = \sigma(W^1 X + b^1\,\mathbf{1})$ Matrix of hidden
- $Y = W^2 H + b^2 \mathbf{1}$ Matrix of outputs

where

$$\mathbf{1} = [1 \quad \ldots \quad 1]$$

W^1,W^2, are the input to hidden weight matric and the hidden to output weight matrix, respectively

The error is quadratic w.r.t the last layer parameters

Let us look at a FNN using using matrixes

The error

- $e = \|vec(T - Y)\|^2$

Vec transforms a matrix into a vector

It is quadratic

$$e = \|vec(T - Y)\|^2$$
$$= \|vec(T - W^2H + b^2\mathbf{1})\|^2$$
$$= \|vec(T) - vec(W^2H + b^2\mathbf{1})\|^2$$
$$= \|vec(T) - \boldsymbol{H'}\otimes\boldsymbol{I_r}\,\boldsymbol{vec}(W^2) - \mathbf{1'}\otimes\boldsymbol{I_r}\,b^2\|^2$$
$$= \|vec(T) - [\boldsymbol{H'}\otimes\boldsymbol{I_r}, \mathbf{1'}\otimes\boldsymbol{I_r}]\,[\boldsymbol{vec}(W^2)', b^2]\|^2$$

Compatibility between Kronecker product and vec

$$vec(ABC) = C'\otimes A\ vec(B)$$
$$vec(BC) = C'\otimes I\ vec(B)$$

Vector of the paramters

Kroneker product

Identity matrix

Vector-matrix product

UNIVERSITÀ DI SIENA 1240

Constructing the network: ELM and fixed basis functions

Extreme learning machines (ELM) (Huang)

1. In ELM only the last layer weights and bias are trained
2. the second layer weights are computed by the pseudoinverse

Claimed advantages

- very fast to train
- approximation property is conserved
- good generalization ? (we will discuss this later)

In general, this is called approximation by fixed basis functions

- Polynomial
- Gaussian functions
- Support vector machine
- …

UNIVERSITÀ DI SIENA 1240

Constructing the network: ELM and fixed basis functions

So, why should we use FNN instead of ELM?

- Approximation by FNNs require a smaller number of neurons!
- Need to discuss about resource usage, not only about universal approximation!

UNIVERSITÀ DI SIENA 1240

Back to the initial practical question

- What type of applications can be implemented by a FNN?

Answer (ver 1.0)

- Almost all common FNNs are universal approximators: they can implement any application!

Advanced question

- Does this mean that all the FNNs are equivalent?

Answer (ver 2.0)

- No, the precision of the approximation depends on the FNN architecture, the number of neurons/paramters ….

UNIVERSITÀ DI SIENA 1240

Going beyond universal approximation

The approximation precision depends on

- the complexity of the model
- the measure of the approximation
- the complexity of the function to be approximated

The idea

- fix a set of functions having a given complexity
- fix a measure of approximation
- study how the approximation improves with a larger and lager number of neurons

UNIVERSITÀ DI SIENA 1240

Going beyond universal approximation

Barron proved that

- Let *t* the target function, T is its Fourier transform and

How much t is complex ⟶ $C_t = \int_{R^n} |v| T(v) dv$

- There is a FNN f_k with sigmoidal activation function and k hiddens

$$\int_{B_r} (t - fk)^2 \leq \frac{(2rCt)^2}{k}$$

Linear convergence of error with the number of neurons

Thus,

- The square error decreases linearly with the number of hiddens

UNIVERSITÀ DI SIENA 1240

Constructing the network

- There is a FNN f_k with sigmoidal activation function and k hiddens

$$\int_{B_r} (t - fk)^2 \leq \frac{(2rCt)^2}{k}$$

Such a bound can be achieved by this procedure which iteratively add neuron

1. Set f_0 (x) equal to the constant 0 function
2. Set f_i (x)=α f_{i-1}(x) + β σ(wx+b)
   - Optimize α ,β,w,b
     (the error must be decrease for a given amount O(1/i))
3. Repeat 2 until the desired error is achieved

UNIVERSITÀ DI SIENA 1240

FNNs vs basis functions

Barron 1993 proved also that

- For every choice of basis function $h_1,..,h_k$, S being the set of functions spanned by $h_1,..,h_k$ and T_c being the set of functions whose complexity is smaller than C, we have

Sublinear convegence

$$\sup_{t \in T_c} \min_{f_k \in S} \int_{[0,1]^n} (t - fk) \geq \kappa \frac{C}{n \sqrt[n]{k}}$$

where κ is an universal consta

Thus,

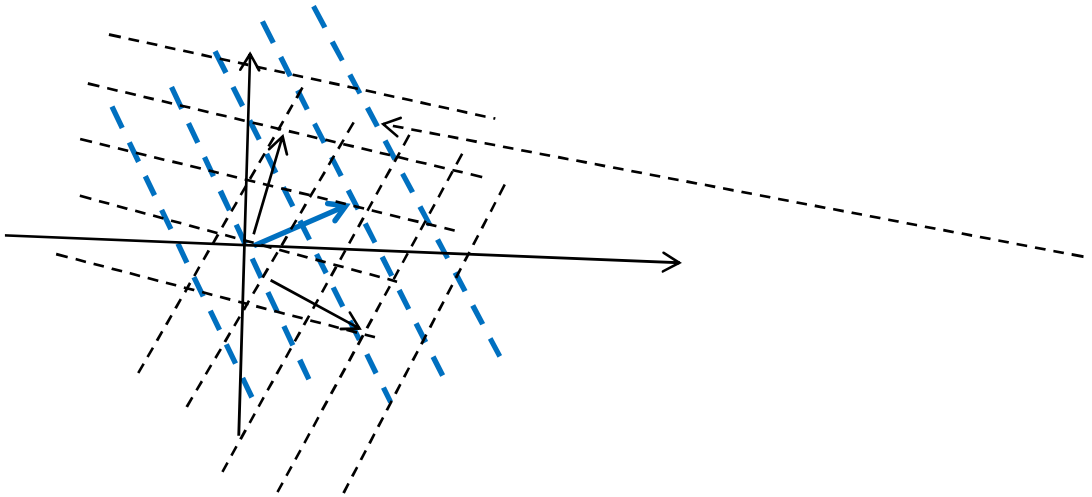
- There are target functions for which approximation by ELM, polynomials, … is much worse than approximation by FNN!
  - It is O(1/k) in FNN wrt $O(1/\sqrt[n]{k})$
  - When the input space is large, the difference is huge

UNIVERSITÀ DI SIENA 1240

FNNs vs ELMs: an intuitive explanation

Back to the ridge grid concept

- When the first layer parameters are random, the hyperplanes $w_i x = b_i$ forms a sort of random grid
- But, when those neurons are really useful?
  - If the target function t is a ridge, only neurons having the direction of the ridge are useful
  - If the target function t is constant in a region, the neurons changing in such a region are not useful



Neurons useufl to approximate a ridge function

UNIVERSITÀ DI SIENA 1240

The complexity of different classes of functions

Back to Barron result

- There is a FNN f_k with sigmoidal activation function and k hiddens

$$\int_{B_r} (t - fk)^2 \leq \frac{(2rCt)^2}{k}$$

How much t is complex

- How large is C_t in practice?

UNIVERSITÀ DI SIENA 1240

The complexity of different classes of functions

Barron proved that

- Ridge functions, t(x)=h(wx),

$$C_t \leq |w| h'(0)$$

The complexity does not depend on number of inputs!

- Radial basis functions, t(x)=h(|x|),

$$C_t \leq n^{1/2}$$

It depends on input dimension

- For polynomial, Barron proved that C_t depends only on the coefficients.

But it is even better, a finite number of sigmoidal neurons are required for any degree of approximation

UNIVERSITÀ DI SIENA 1240

The complexity of combined functions

Barron proved that

- Translation, t(x)=h(x+b)

$$C_t = C_h$$

Translation does not affect complexity

- Linear combination, $t(x) = \sum_{i=1}^{k} \alpha_i h_i(x)$

$$C_t = \sum_{i=1}^{k} \alpha_i C_i$$

UNIVERSITÀ DI SIENA 1240

The complexity of combined functions

Barron proved that

- Product, t(x)=h(x)*g(x)

$$D_t = D_h\, D_g$$
$$C_t = Dh\; C_k + D_k\, C_h$$

- where

$$C_t = \int_{R^n} |v| T(v) dv \qquad D_t = \int_{R^n} T(v) dv$$

(T is the Fourier transform of t)

UNIVERSITÀ DI SIENA 1240

Which are the complex functions?

A doubt

- For all the above classes functions (except for gaussian), the error decreases linearly with number of hidden units
- Even if the combined function are simple
- Which are the functions which requires a lot of hiddens?

Intuitive answer

- Complex functions cannot be defined from simple functions in few steps!!!
- Several products, sums … are required
- Or several compositions t(x)=F(h(x)) are required

UNIVERSITÀ DI SIENA 1240

Final pratical remarks about approximation capability

Properties

- most of the common models are universal approximators
- but different architectures have different properties!!

In practice

- the best architecture depends on the problem

UNIVERSITÀ
DI SIENA
1240

Learning capability

UNIVERSITÀ DI SIENA 1240

Practical question: learning capability

Now, we know what a FNN can approximate any function, but what about what a FNN can learn?

UNIVERSITÀ DI SIENA 1240

Learning in neural networks

Optimization of error function ver 1.0

- by gradient descent
- update the parameters according to

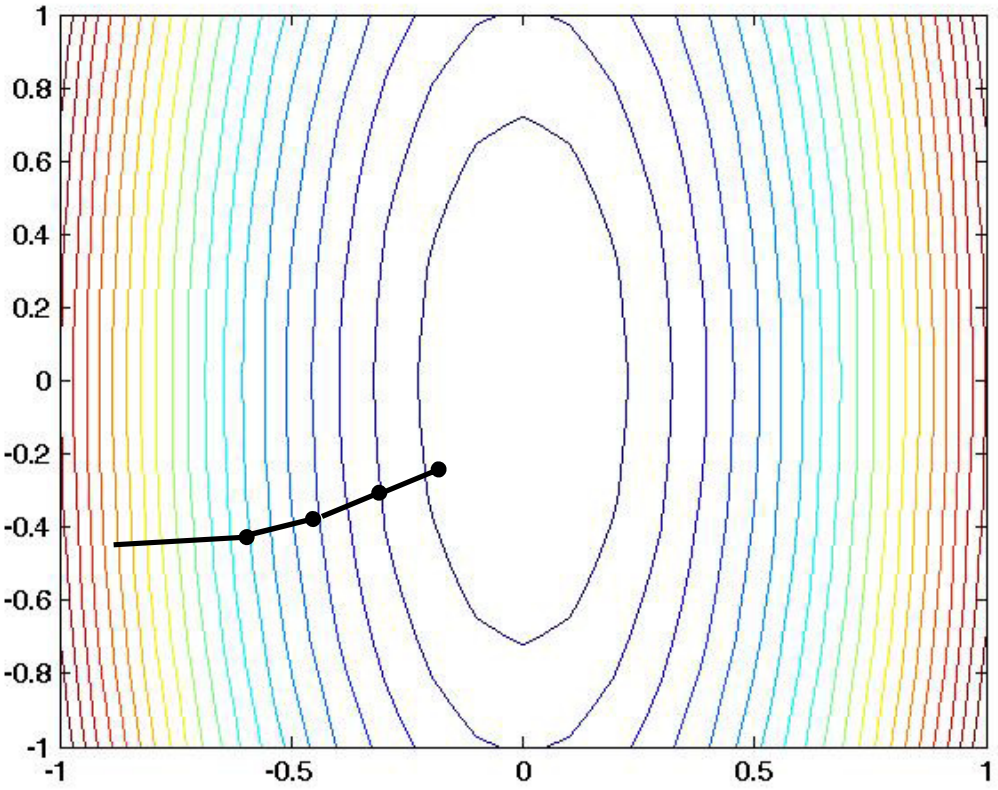
$$w(t+1) = w(t) - \lambda \frac{\partial ew}{\partial w}$$

until a desired minimal error is obtained

UNIVERSITÀ DI SIENA 1240

Learning in neural networks

Gradient descent



UNIVERSITÀ DI SIENA 1240

Learning in neural networks

Gradient computation

- by an algorithm called backpropagation
- linear time w.r.t. the number of neurons
- not a matter of this course

Optimization of error function

- Several variants of gradient descent exist: momentun, batch, …
- Several other optimization algorithms exist: resilient backpropagation, conjugate gradient, Newton, …
- Not matter of this course

UNIVERSITÀ DI SIENA 1240

An old experiment

The idea (Lawrence et al.)

- construct a random network N1 with k1 hiddens
- generate a random domain and use N1 to generate the targets
- train another network N2 with k2 hiddens

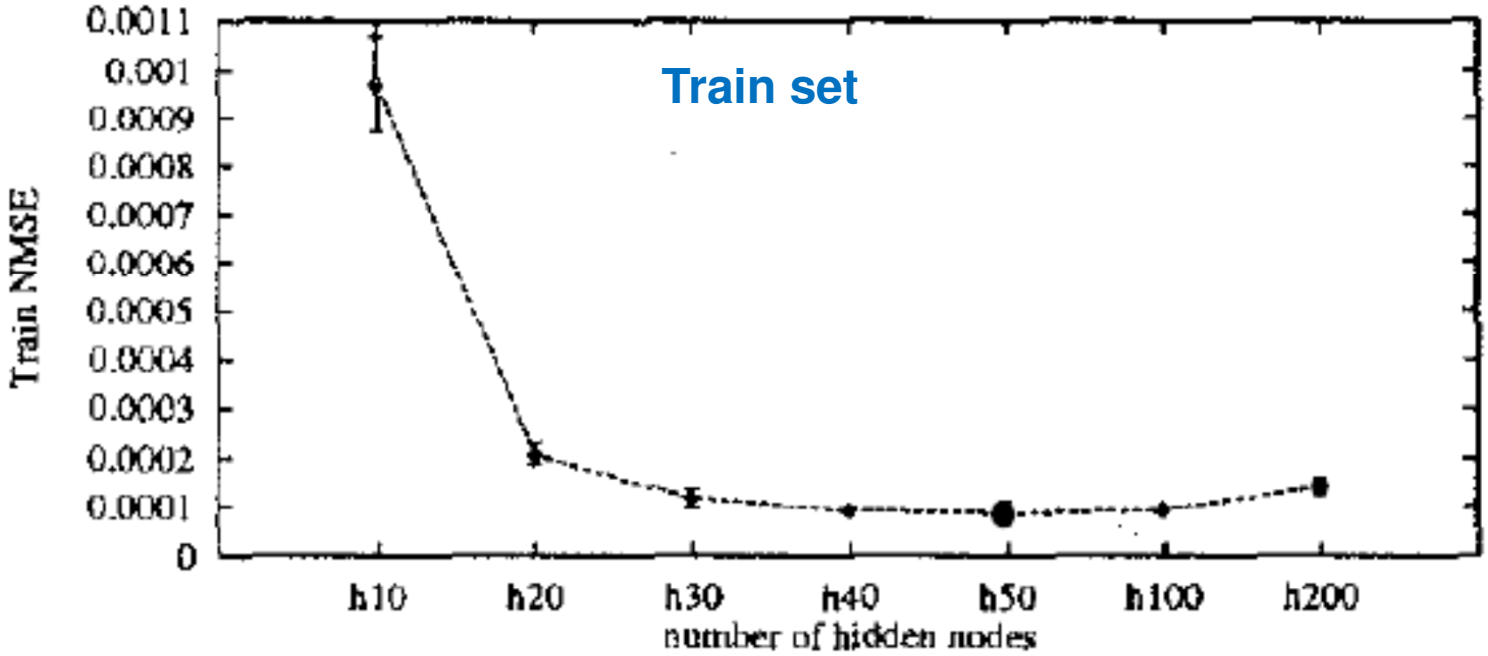
- When k2>=k1, the best minina has cost 0!

UNIVERSITÀ DI SIENA 1240

An old experiment

Results

- target network k1=10
- 5 inputs,5 output
- 100 patterns train set
- Experiments repeated several times

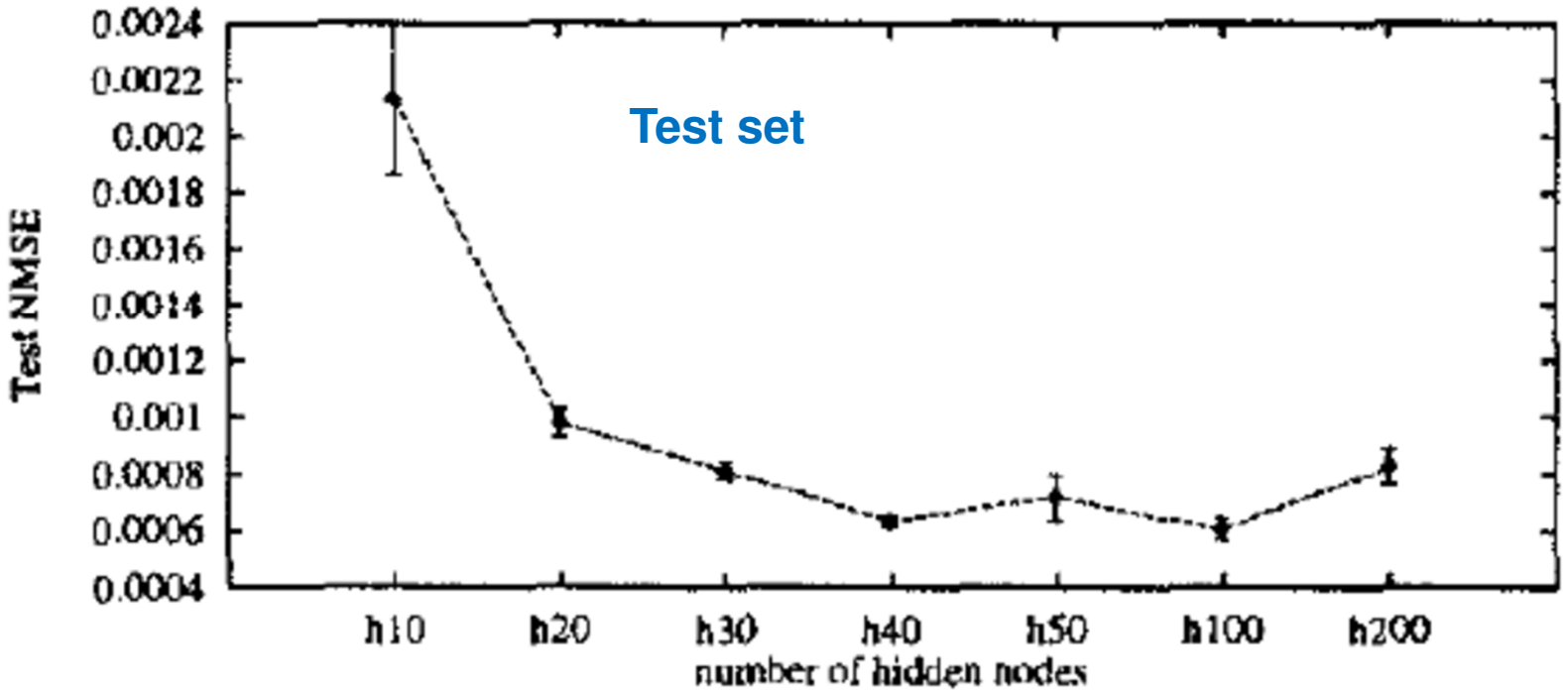


UNIVERSITÀ DI SIENA 1240

An old experiment

Results

- target network k1=10
- 5 inputs,5 output
- 100 patterns train set
- Experiments repeated several times



UNIVERSITÀ DI SIENA 1240

An old experiment

Conclusions

- Training often does not produce the optimum
  - Training is a challenge
- The error improves increasing the number of hidden units
  - The more the parameters, the better the approximation
- The error on test set may increase even it increases on train set
  - Generalization is a challenge

UNIVERSITÀ DI SIENA 1240

Practical question: learning capability

Why does training fail

Answer (ver 1.0): local minima

- learning capability depends on the presence/absence of local minima in error function

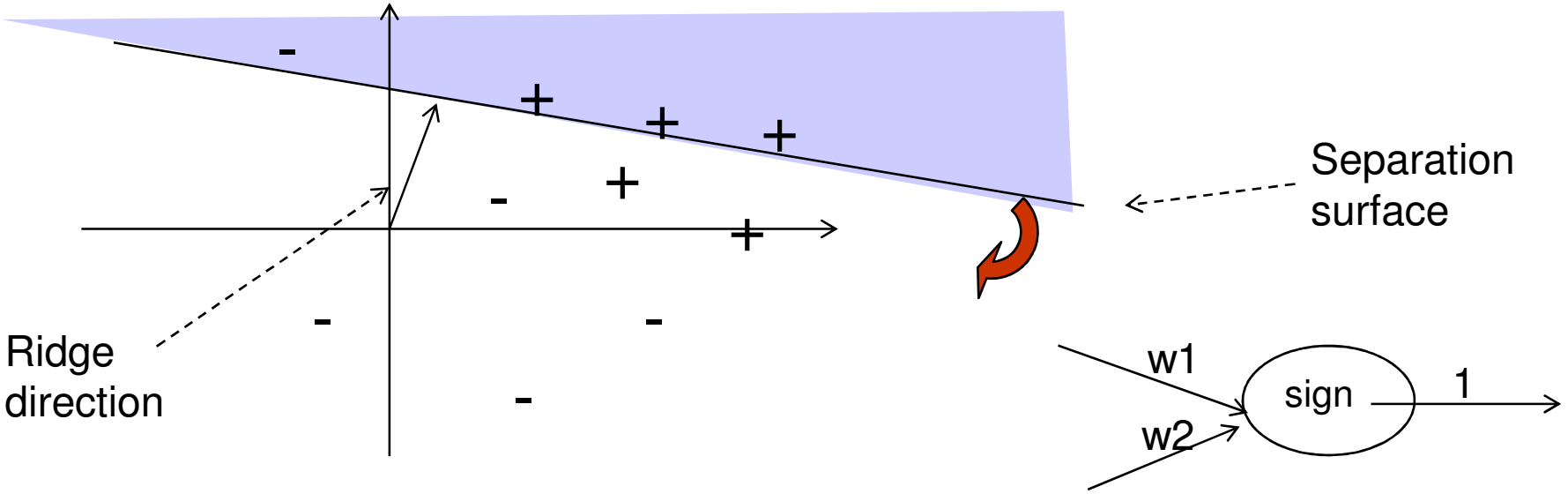
- Theoretical results on this are few and incomplete … let us review them

UNIVERSITÀ DI SIENA 1240

Absence of local minina

There is no local minimum if

- Network
  - a single neuron (with sign activation)
- Patterns
  - Linearly separable patterns
- Proof: look at the separation surface and how it can be moved

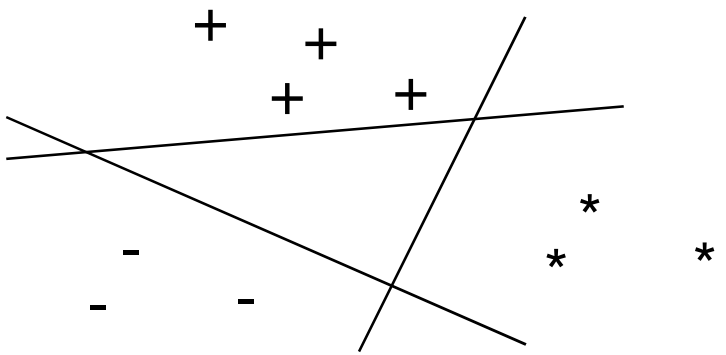


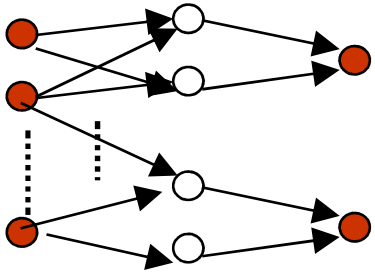
UNIVERSITÀ DI SIENA 1240

Absence of local minina

Extension to FNNs (Gori et al.)

- Network
  - one hidden layer
  - sigmoidal outputs, one hot coding of C classes
  - hidden to output weights are separated for each class
- Patterns
  - Linearly separable





input layer output layer

UNIVERSITÀ DI SIENA 1240

Absence of local minina

(At least) so many neurons as patterns (Yu et alt.)

- Network
  - One hidden layer with n neurons
  - Linear outputs
- Patterns
  - n-1 distinct pattern

So many neurons as patterns

Proof… the idea

Remember that

$$e = \|vec(T - Y)\|^2 = \|vec(T) - [\boldsymbol{H'} \otimes \boldsymbol{I_r}, \boldsymbol{1'} \otimes \boldsymbol{I_r}]\,[\boldsymbol{vec}(W^2)', b^2]\|^2$$
$$= \|vec(T) - Rp_2\|^2$$

$p_2 = [\boldsymbol{vec}(W^2)', b^2]$

$R = [\boldsymbol{H'} \otimes \boldsymbol{I_r}, \boldsymbol{1'} \otimes \boldsymbol{I_r}]$,

- with neurons and n–1 patterns R is a square matrix
- if R is full rank, than the linear system has a solution!

  0=vect(T)-Rp$_2$
  - The trainset can be perfectly learned (error =0)!

UNIVERSITÀ DI SIENA 1240

So many neurons as patterns

Proof… the idea

Remember that

$$e = \|vec(T) - Rp_2\|^2$$

$p_2 = [\boldsymbol{vec}(W^2)', b^2]$

$R = [\boldsymbol{H'} \otimes \boldsymbol{I_r}, \boldsymbol{1'} \otimes \boldsymbol{I_r}],$

- The gradient is

$$\frac{\partial e}{\partial p_2} = 2(vec(T) - Rp_2)'R$$

  - If R is full rank, than the gradient is 0 only when the error is 0!

The rest of the proof (skipped)

- When R is not full rank (very rare case in practice), then equilibrium points correspond unstable points

UNIVERSITÀ DI SIENA 1240

Absence of local minina

Linear networks (Baldi)

- Network
  - One hidden layer
  - Linear outputs
- Patterns
  - Any set of patterns

- The result
  - The error surface show a global minima and several saddle points but no local minima

UNIVERSITÀ DI SIENA 1240

Presence of local minima

Many minima for a perceptron (Auer 1996)

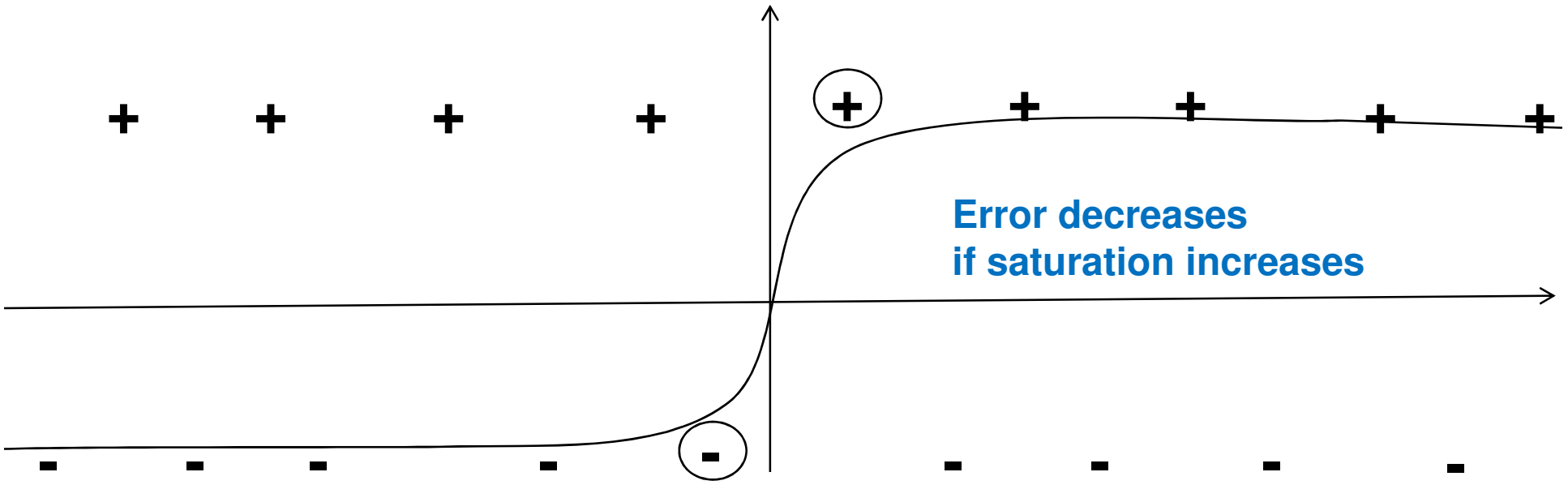
- Network
  - A single perceptron
  - Sigmonidal activation function
- Patterns
  - K patterns, ad hoc displacing
- Number of local minima
  - At least

UNIVERSITÀ DI SIENA 1240

Presence of local minima

A sketch of the proof for a single input perceptron

- Only the patterns where the neurons are not saturated affects the derivative of the error
- The following figure shows a local minima, as we can only improve the errors on the circled patterns

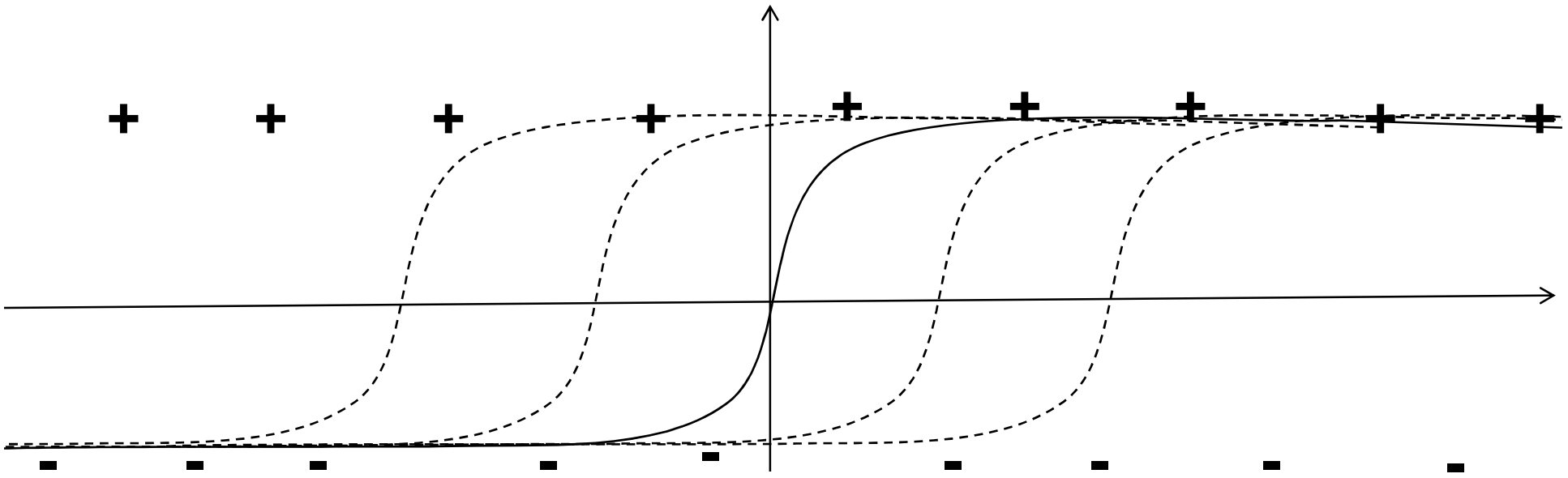


UNIVERSITÀ DI SIENA 1240

Presence of local minima

A sketch of the proof for a single input perceptron

- There is a lot of minima



UNIVERSITÀ DI SIENA 1240

Presence of local minina

Extended XOR (Bianchini)

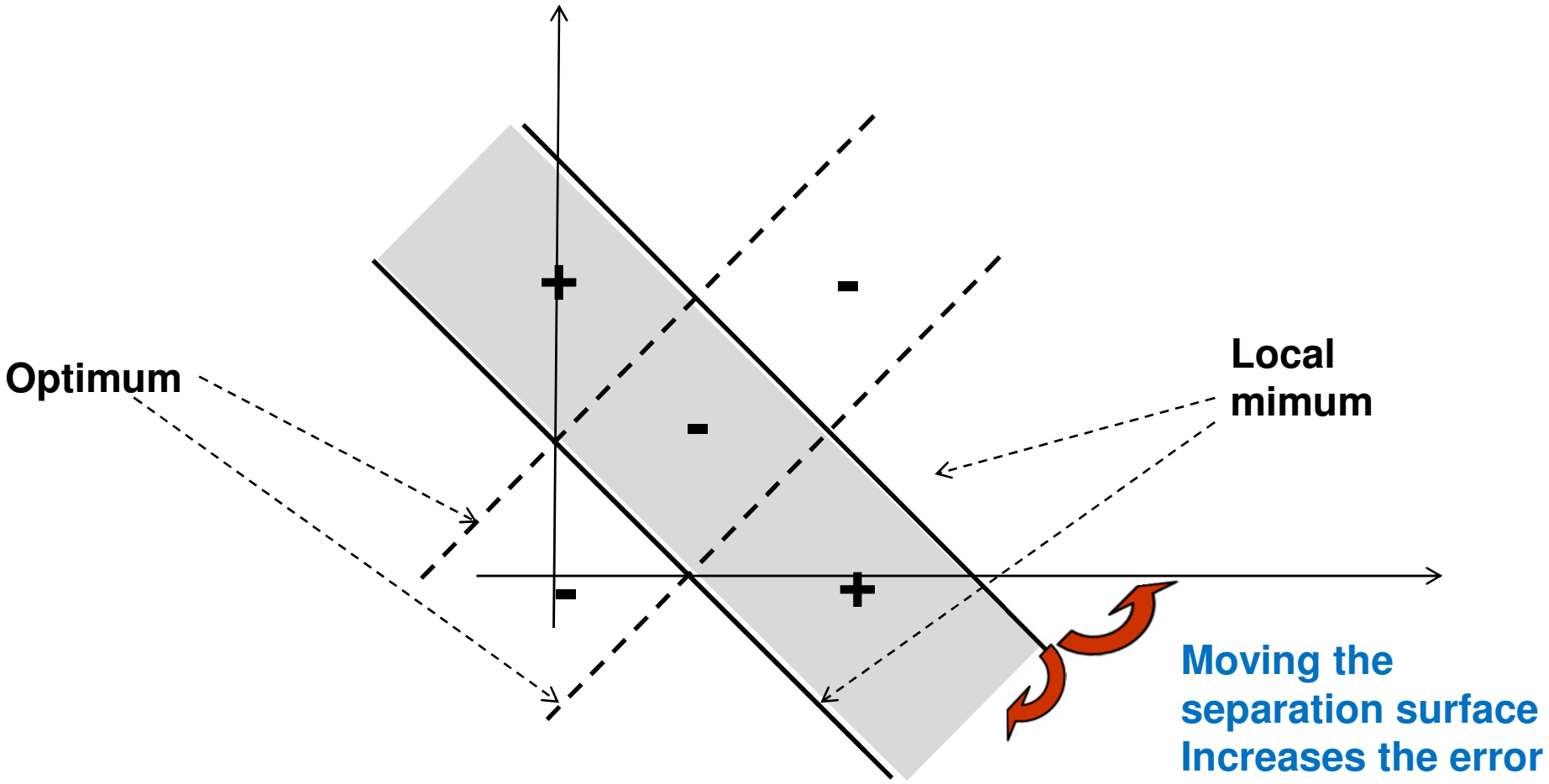
- Network
  - One hidden layer with 2 neurons
  - Sigmonidal output activation function
- Patterns
  - 5 patterns

x1	x2	target
0	0	0
0	1	1
1	0	1
1	1	0
0.5	0.5	0

UNIVERSITÀ DI SIENA 1240

Presence of local minina

Extended XOR (Bianchini 96)



UNIVERSITÀ DI SIENA 1240

Practical question: learning capability

Why does training fail?

- Are really local minima the answer?

We may have disregarded the role of

- attracting regions
- flat regions (and saddle points)
- regions with deep valley

UNIVERSITÀ DI SIENA 1240

Attracting regions

Attracting regions (local minima to infinity)

- The error decrease on a path which makes weights larger and larger
- The minima, if it exists, it is at infinity: it can be both a global minimum or a sub-minimum

Why learning is difficult

- Lager weights produce saturations in networks with sigmoids (and flat error surface)
- Large weights may produce numerical problems
- Large weights may produce large oscillations during learning

UNIVERSITÀ DI SIENA 1240

Attracting regions

Networks with sigmoidal activation

- They may have a large number of attracting regions, due to the saturation of the sigmoid when the weights are large
- If output neurons use sigmoids and the error function is MSE
  - a perfect learning of the train set requires for infinitive weights!

Networks with only ReLU activations

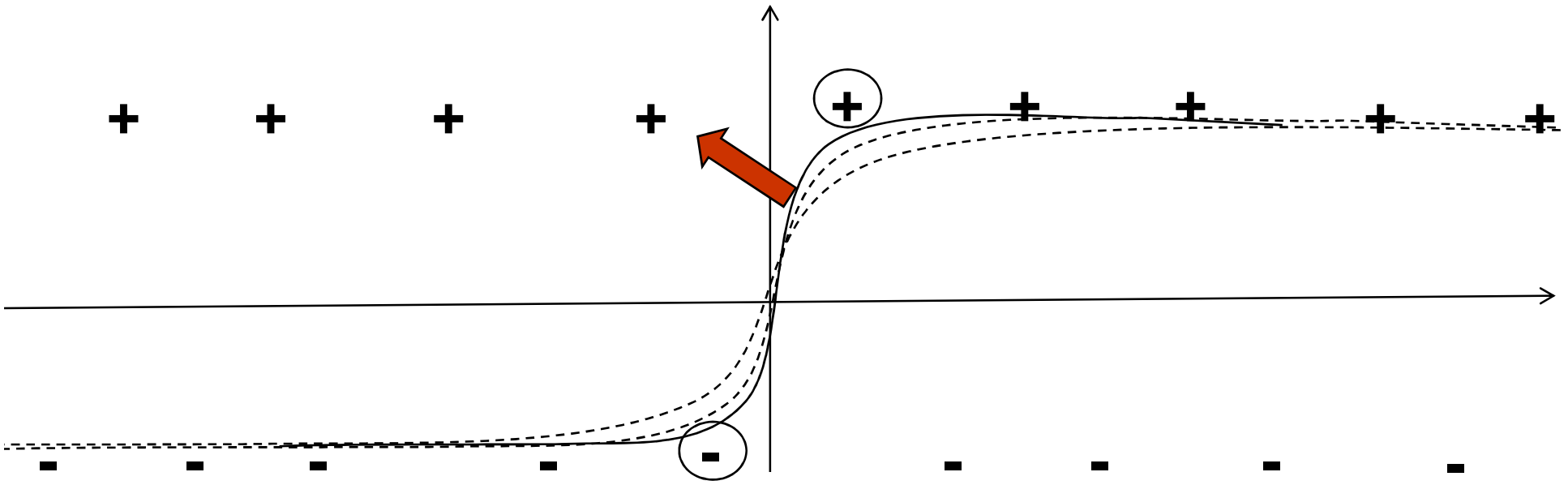
- They require large weights to implement discontinuous or unbounded functions (recall approximation by step functions)
- Otherwise,
  - With MSE, attracting regions do not exist
  - With cross entropy and soft max attracting regions do exists, but they are equivalent finite minima

UNIVERSITÀ DI SIENA 1240

Attracting regions

With sigmoids

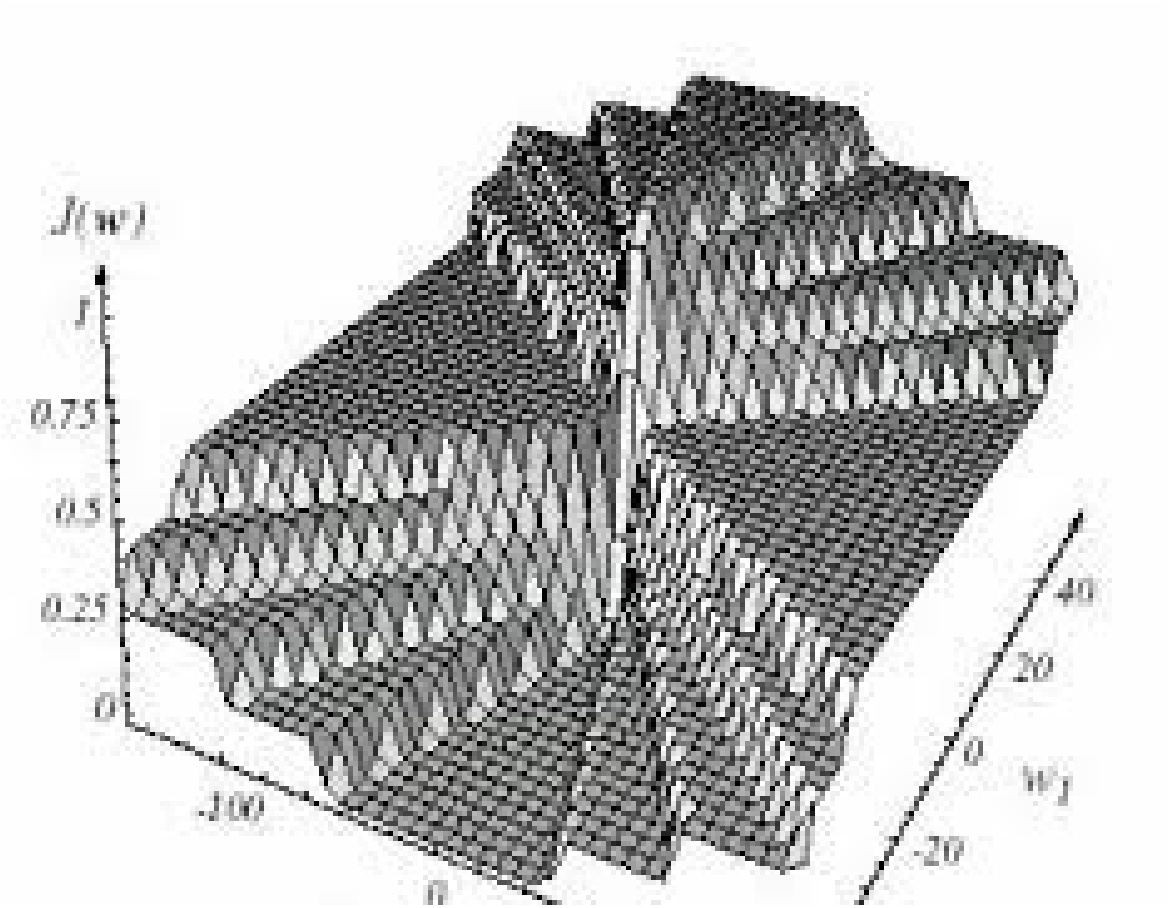
- if input to hidden weight increases, saturation increases
- If saturation increases, the error decreases



UNIVERSITÀ DI SIENA 1240

Flat regions (and saddle points)

An example of an error surface



UNIVERSITÀ DI SIENA 1240

Flat regions (and saddle points)

Why learning is difficult

- No ways to distinguish between saddle points, flat surface and minima
- Very slow convergence rate
- Numerical errors

In both networks with sigmoids and ReLU

- In sigmoids flat surfaces are due to saturations
- In ReLU, flat surfaces are due to the constant part ot ReLU
  - Some approaches use
    - Leaky ReLU $\sigma(x)=x$ if $x>0$, $\sigma(x)=ax$ otherwise, with $0<a<1$
    - Exponential linear unit (ELU)
      $\sigma(x)=x$ if $x>0$, $\sigma(x)=a(e^{-x}-1)$ otherwise, with $a>0$
    - ….

UNIVERSITÀ DI SIENA 1240

Regions with deep valleys

Condition number for a matrix A

- The ratio between the largest and the minimum eigenvector
  - $k(A) = \lambda_{max}(A) / \lambda_{min}(A)$

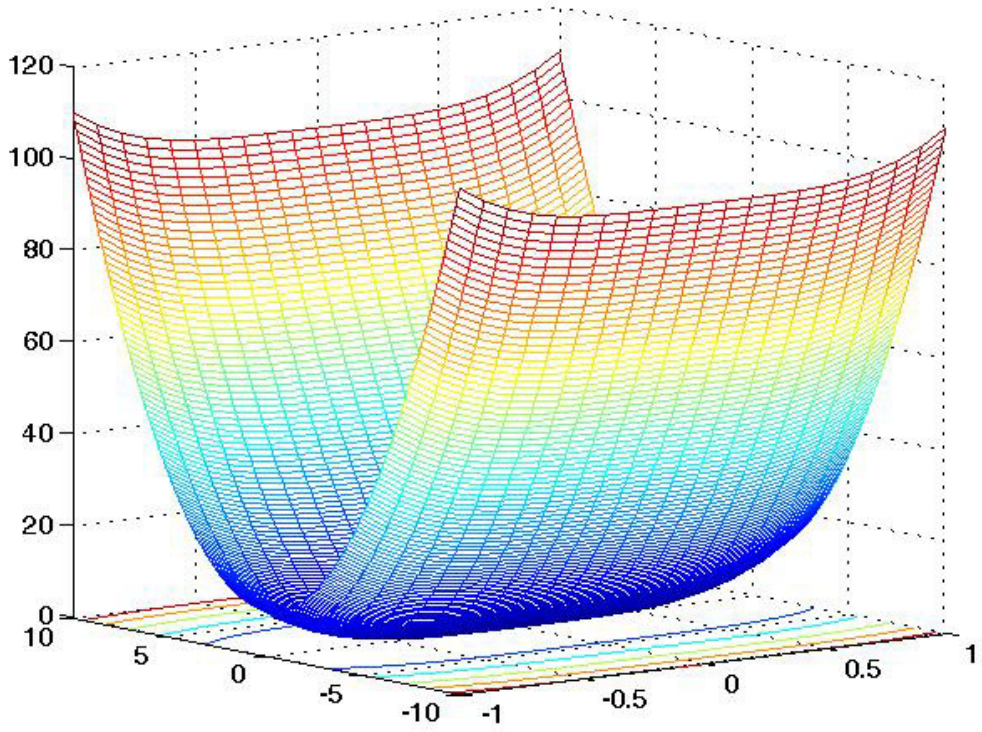
An error function e_w having Hessian matrix $\nabla^2 e_w$ with a large condition number

- the error function has a deep valley
- optimization is difficult when $k(\nabla^2 e_w)$ is large
  - Explanation 1.0

  Gradient descent follows a zig-zag path!!

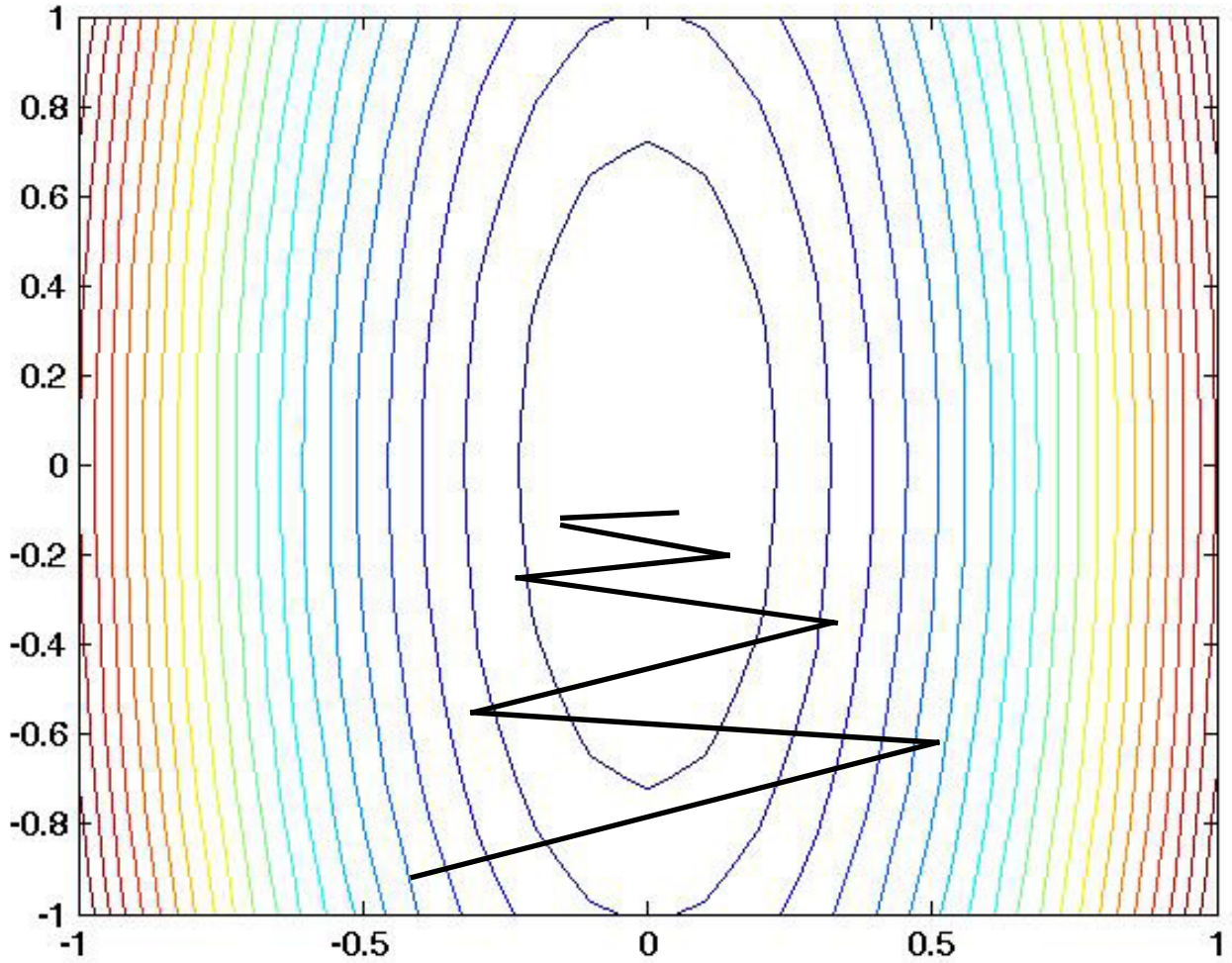
UNIVERSITÀ DI SIENA 1240

Regions with deep valleys



UNIVERSITÀ DI SIENA 1240

Regions with deep valleys



UNIVERSITÀ DI SIENA 1240

Regions with deep valleys: a theoretical viewpoint

Remember gradient descent

- $e_{w\,(t)}$ error function at iteration t w.r.t. weights w(t)
- α learning rate $\nabla ew_{(t)}$ gradient

$$w(t+1) = w(t) - \alpha \nabla ew_{(t)}$$

- Gradient descent converges if e_w is convex (or if it starts in an attraction basin)

- Let us assume an optimal α is chosen

UNIVERSITÀ DI SIENA 1240

Regions with deep valleys: a theoretical viewpoint

It can be proved

- Let w* be the optimal weights and $\nabla^2 e$ the Hessian matrix in w*

$$\|w^* - w(t)\| \leq \frac{(k(\nabla^2 e) - 1)}{(k(\nabla^2 e) + 1)} \|w^* - w(t-1)\|$$

- Thus, in order to obtain an error ε, we need k iterations

$$k = O\left(\frac{(k(\nabla^2 e) - 1)}{(k(\nabla^2 e) + 1)} / \log(\varepsilon)\right)$$

UNIVERSITÀ DI SIENA 1240

Regions with deep valleys: a theoretical viewpoint

- In a general case, the function may not strongly convex
  - In this case, condition number of Hessian may be infinitive
- Let e_w .satisfies the following (L - Lipschitz continuous gradient function, L-lcg function)

$$\|\nabla e_{w1} \_ \nabla e_{w2}\| \leq L\|w1 - w2\|$$

It can be proved

- Basic gradient descent is $O(L/\varepsilon)$
- Newton is $O\left(\sqrt{L/\varepsilon}\right)$

UNIVERSITÀ DI SIENA 1240

Regions with deep valleys: a theoretical viewpoint

A lower bound

- For any ε, there exists an *L-lcg.* function *f* , such that any first-order method takes at least

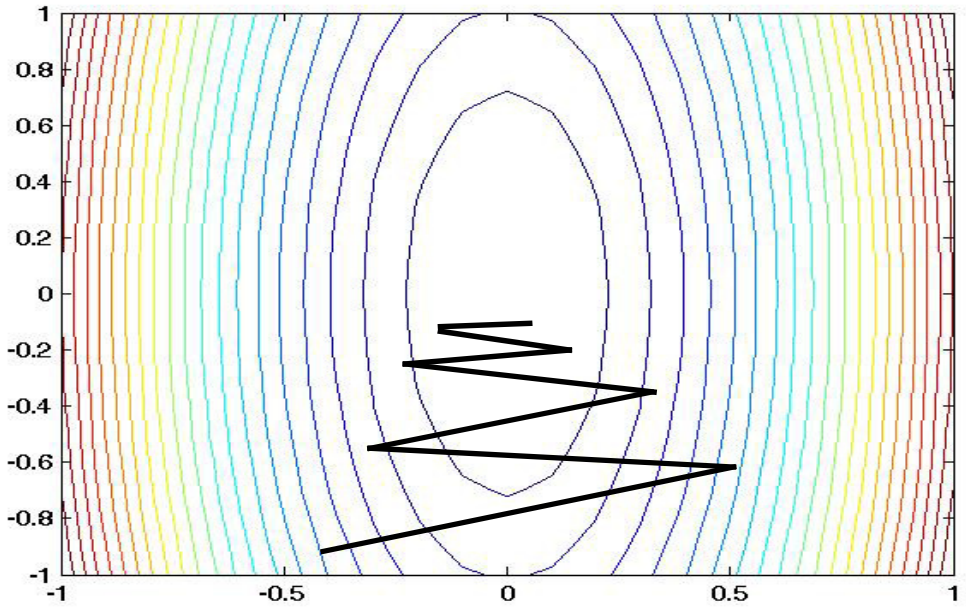
$$O\left(\sqrt{L/\varepsilon}\right)$$

steps

UNIVERSITÀ DI SIENA 1240

Intuitive message 1

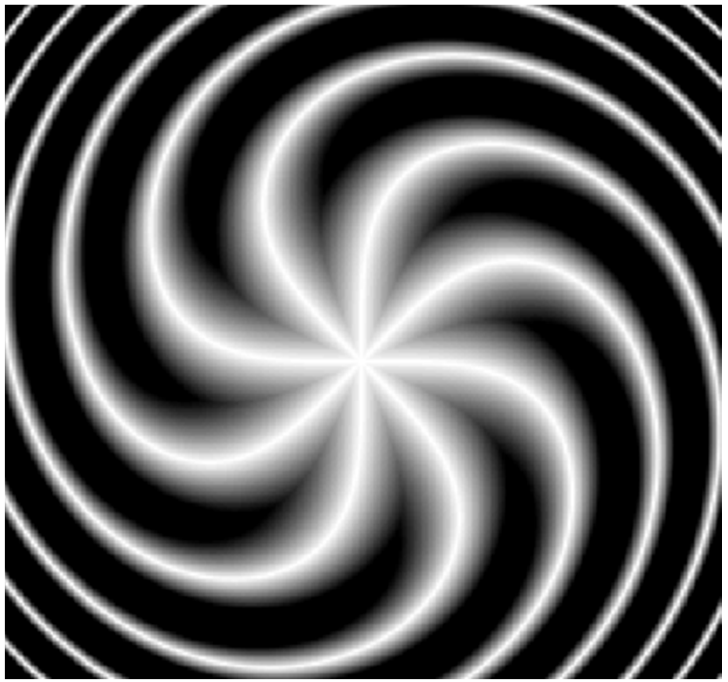
- Learning may be difficult even if the error function has only a minimum: the minimum may be at the bottom of a (badly conditioned) deep valley!!



UNIVERSITÀ DI SIENA 1240

Intuitive message 2

- Learning may bed difficult even if the error function has only a minimum: when the L-lcg function is large the gradient may be and widely vary giving rise to very weird functions!!



UNIVERSITÀ DI SIENA 1240

Learning capability: a negative result

Loading problem (decision version)

- Given a neural network architecture and a set of training examples, does there exist a set of edge weights for the network so that the network produces the correct output for all the examples

It has bee proven that the following Judd

- Loading problem is NP-complete!!

The theorem holds for

- binary functions
- network with threshold activation functions

UNIVERSITÀ DI SIENA 1240

Learning capability: a negative result

Extensions proven by Judd

- only two layers
- fan-in smaller or equal 3
- Only 67% are required to be correct
- …

UNIVERSITÀ DI SIENA 1240

Final remarks about learning capabibility

In practice

- even if a model can approximate a target function, such a model may not easy to learn
- learning capability depends
  - on the problem (train set)
  - the adopted model

- In general
  - the smallest the data, the simplest the learning
  - the larger the model, the simplest the learning

UNIVERSITÀ DI SIENA 1240

Other aspects we have not considered

Information contained in features

- Noise and lack of information may prevent perfect loading
  - It is difficult to know whether your learning algorithm works
- When the information in feature is very few, you may want to adopt transductive learning methods

Error function adopted for learning

- Training error function is often different from test error function
  - You may want to try different train error function

UNIVERSITÀ DI SIENA 1240

Other aspects we have not considered

Patterns are trained independently

- Good precision does not ensure that derivatives are approximated
- Relationships between patterns are not ensured
  - There are machine learning methods suitable for this case

UNIVERSITÀ
DI SIENA
1240

Generalization capability

UNIVERSITÀ DI SIENA 1240

Pratical question: generalization capability

After training, how does the FNN will perform on new patterns from a test set?

Generalization

- let us measure this with the performance of a model f on a test set T

f= Ψ(L) is produced by a learning algorithm Ψ using a train set L

UNIVERSITÀ DI SIENA 1240

Pratical question: generalization capability

Interesting questions

- what is the predicted performance of f on test set T?
- This question is related to
  - which model should we choose?
  - which learning algorithm should we choose?
  - When learning should be stop?

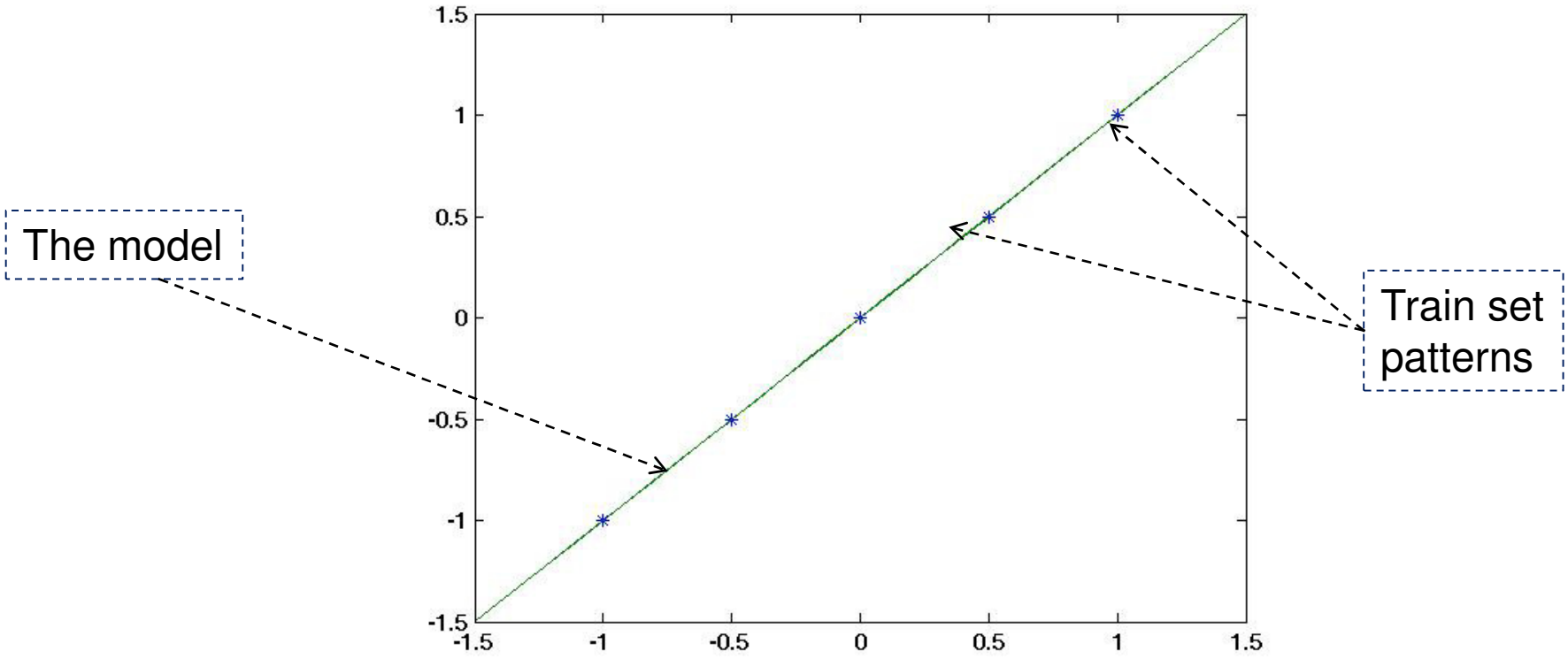
Answer 1.0

- no answer is possible without assumptions on training algorithm and network architecture

UNIVERSITÀ DI SIENA 1240

Measuring generalization capabilty requires assumptions

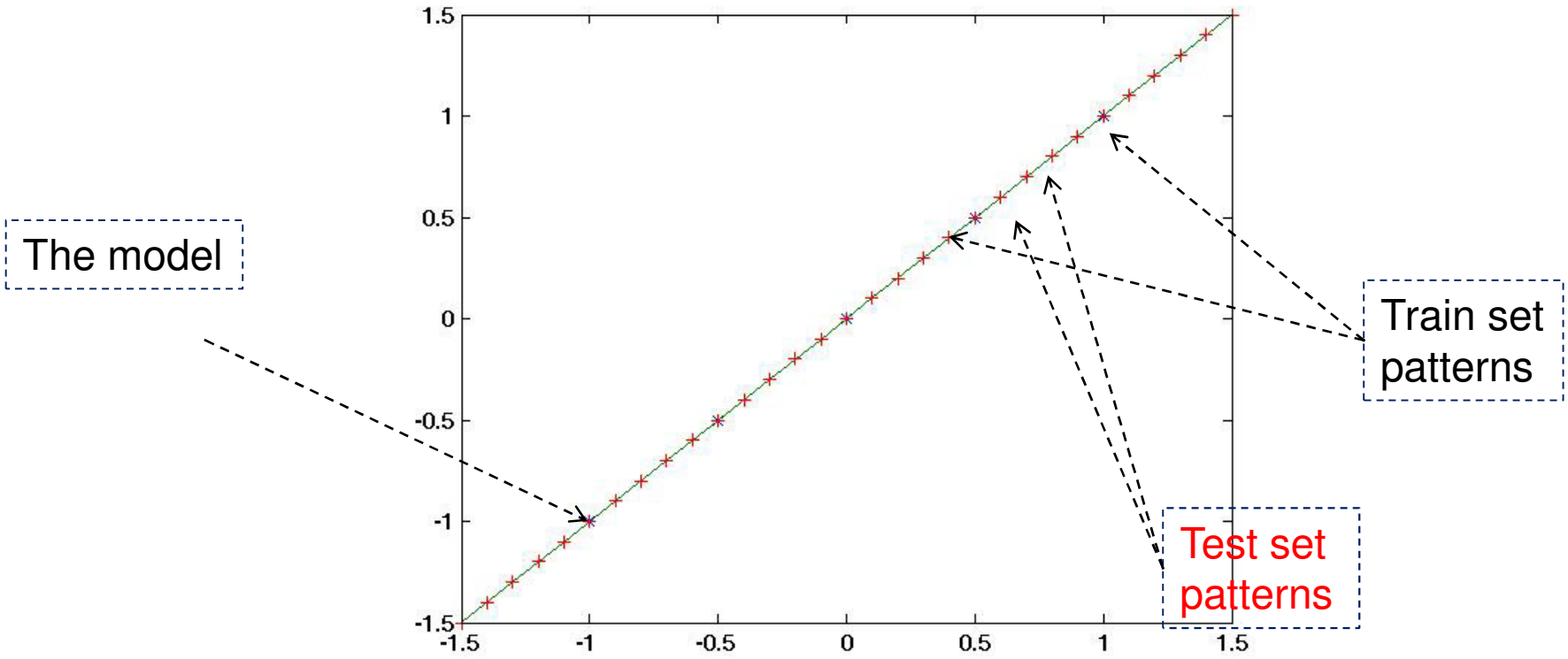
How well will this model generalize on novel patterns?



UNIVERSITÀ DI SIENA 1240

Measuring generalization capabilty requires assumptions

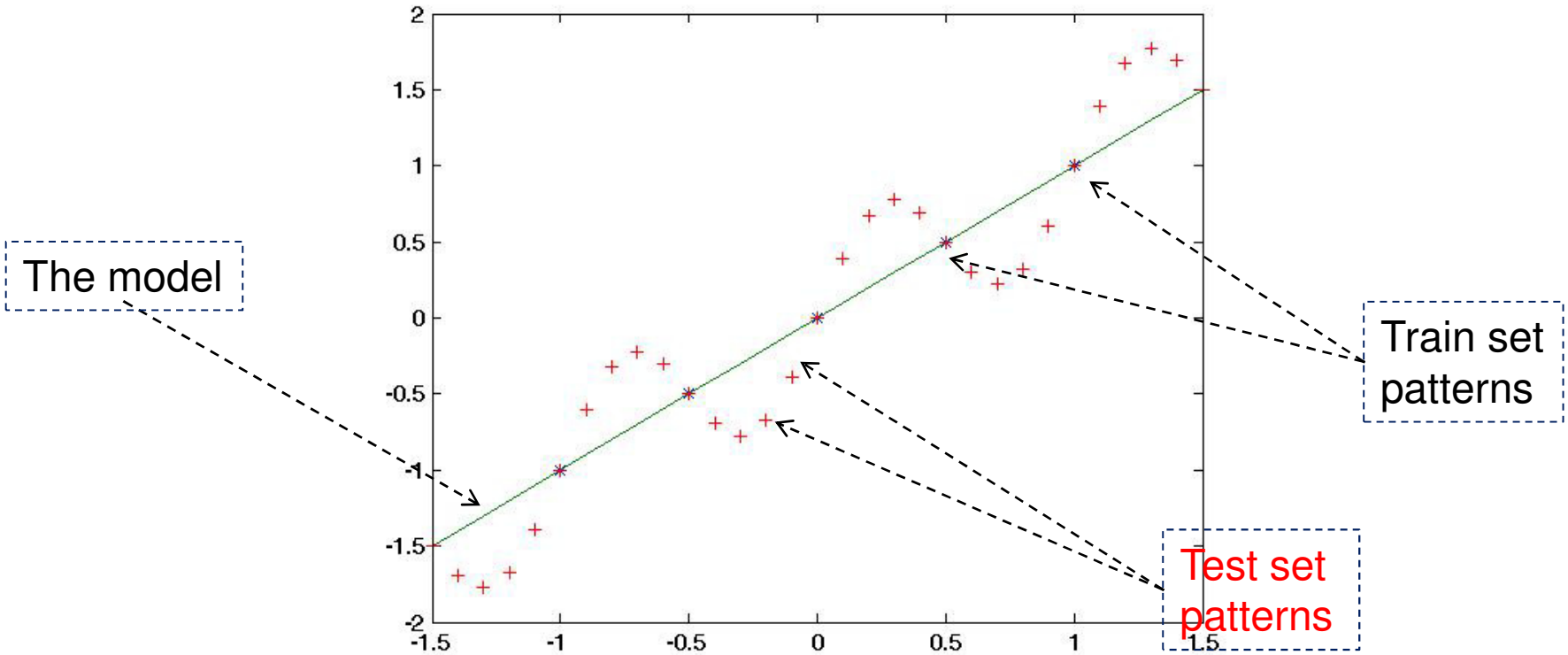
Very well!!!



UNIVERSITÀ DI SIENA 1240

Measuring generalization capabilty requires assumptions

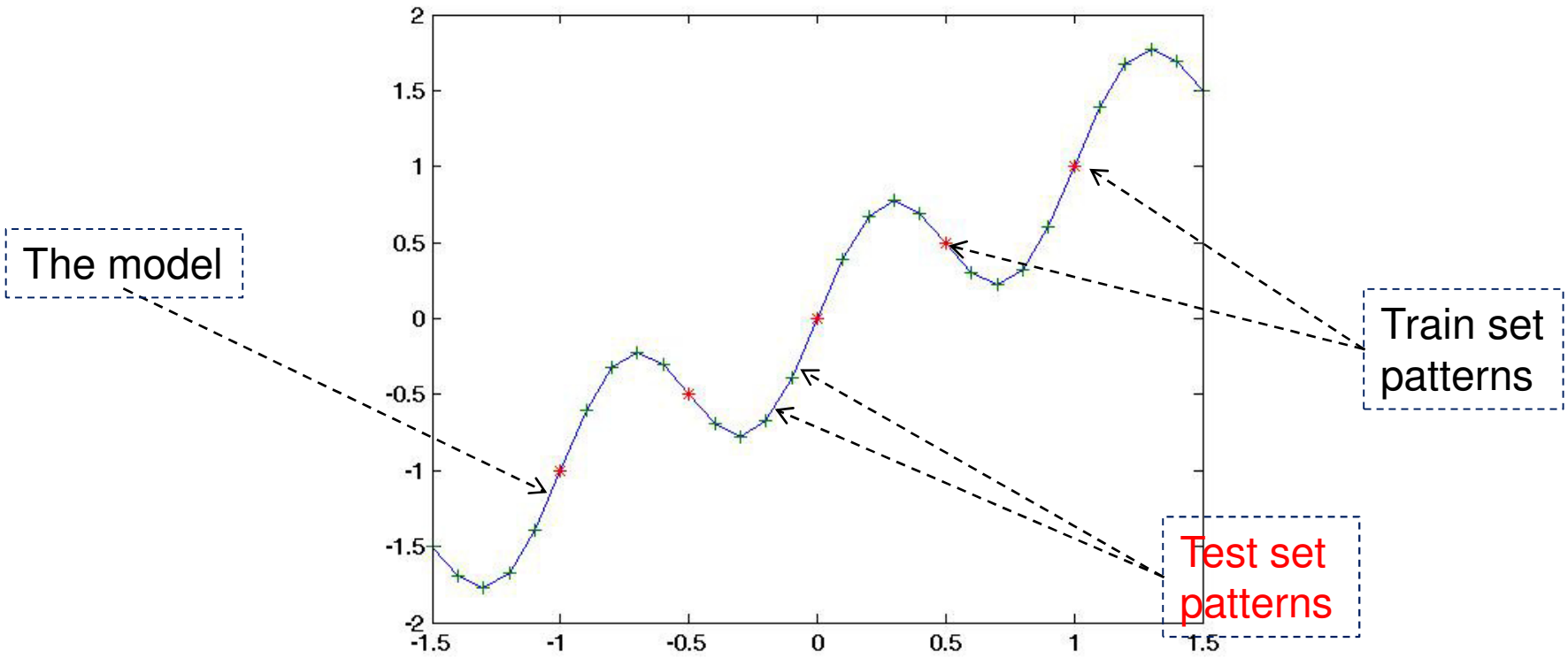
Very bad!!!



UNIVERSITÀ DI SIENA 1240

Measuring generalization capabilty requires assumptions

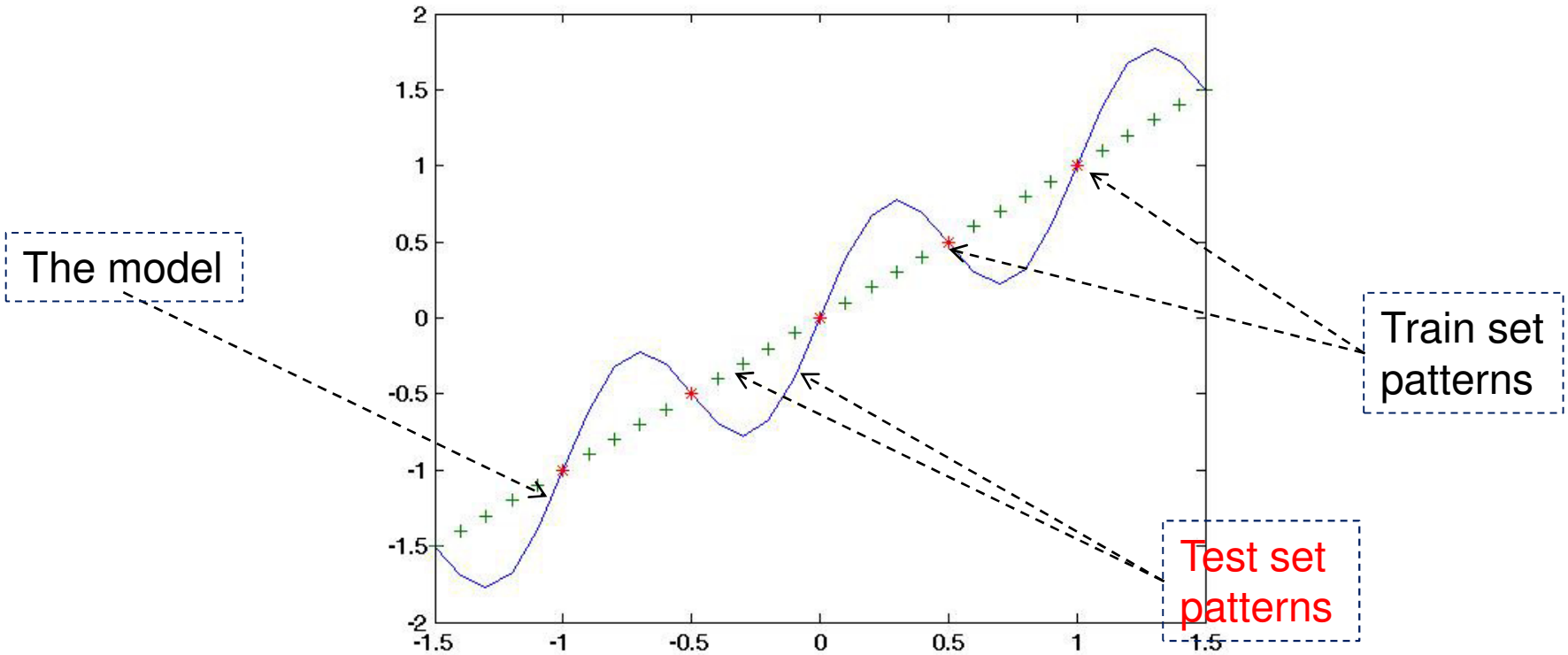
Another model?



UNIVERSITÀ DI SIENA 1240

Measuring generalization capabilty requires assumptions

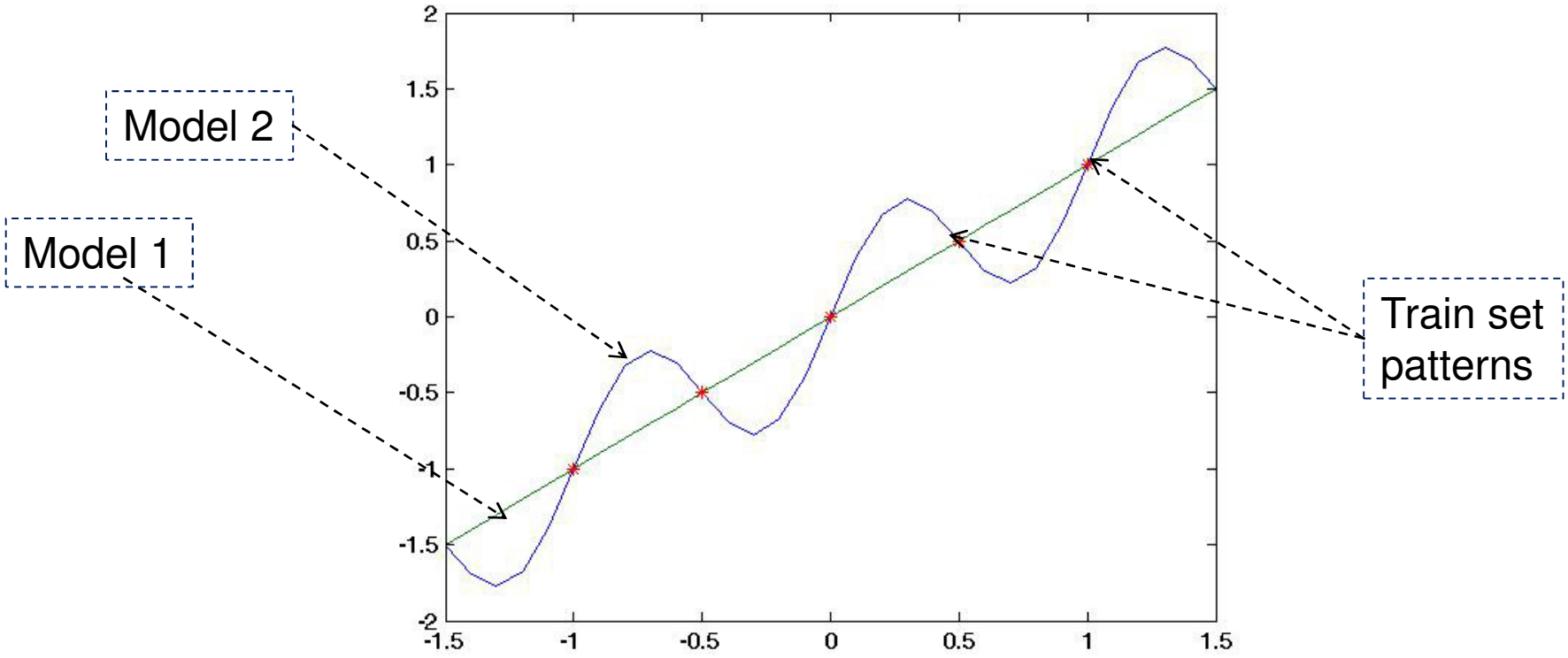
Another model? …. It does not generalize well in this case!



UNIVERSITÀ DI SIENA 1240

Measuring generalization capabilty requires assumptions

Which model is best to have a good generalization?



UNIVERSITÀ DI SIENA 1240

No free lunch theorem

Intuitive version

- Without constraints on the considered problem, all the learning algorithms may show the same generalization error!!

Formally

- *A training algorithm which produces a model f:X->Y , using a train set*
- X= a fine set of inputs, Y= a finite set of outputs
- *Train set* $L=\{(x_i,t_i) \mid x_i \in X,\ t_i \in Y\}$
- *Test set* $T=\{x_i \mid x_i \in X\}$
- *The target function t:X->Y*
- *Error on test set* $e_{test} = \sum_{x_i \in T} L(t(x_i), f(x_i))$ *for some error function L*

If the target function t is uniformly sampled, for any learning algorithm *mean(e_{test})* is constant

UNIVERSITÀ DI SIENA 1240

No free lunch theorem

It means that without assumptions

- The learned model which is better on a problem is worse on another:
  - If A1 is better than A2 on certain kind of problems, there must be another kind of problems where A2 is better than A1
  - averaged over all the problems, both algorithms are equally good.
- Even a random learning algorithm performs as well as the other algorithms
- Generalization is not possible without a (often implicit) bias of the algorithm

UNIVERSITÀ DI SIENA 1240

Measuring generalization capabilty requires assumptions

Assumption 1: data distribution

- The data on test set (working environment) are drawn from the same distribution as the data in train set
  - Not obvious in real applications
  - It means that the pair pattern-targets must be drawn from the same distributions

UNIVERSITÀ DI SIENA 1240

Measuring generalization capabilty requires assumptions

Occam's razor (law of parsimony):

- the simplest explanation is usually the correct one

Assumption 2:

- the model that produces the best generalization is the simplest (among those that classifies correctly the train set)

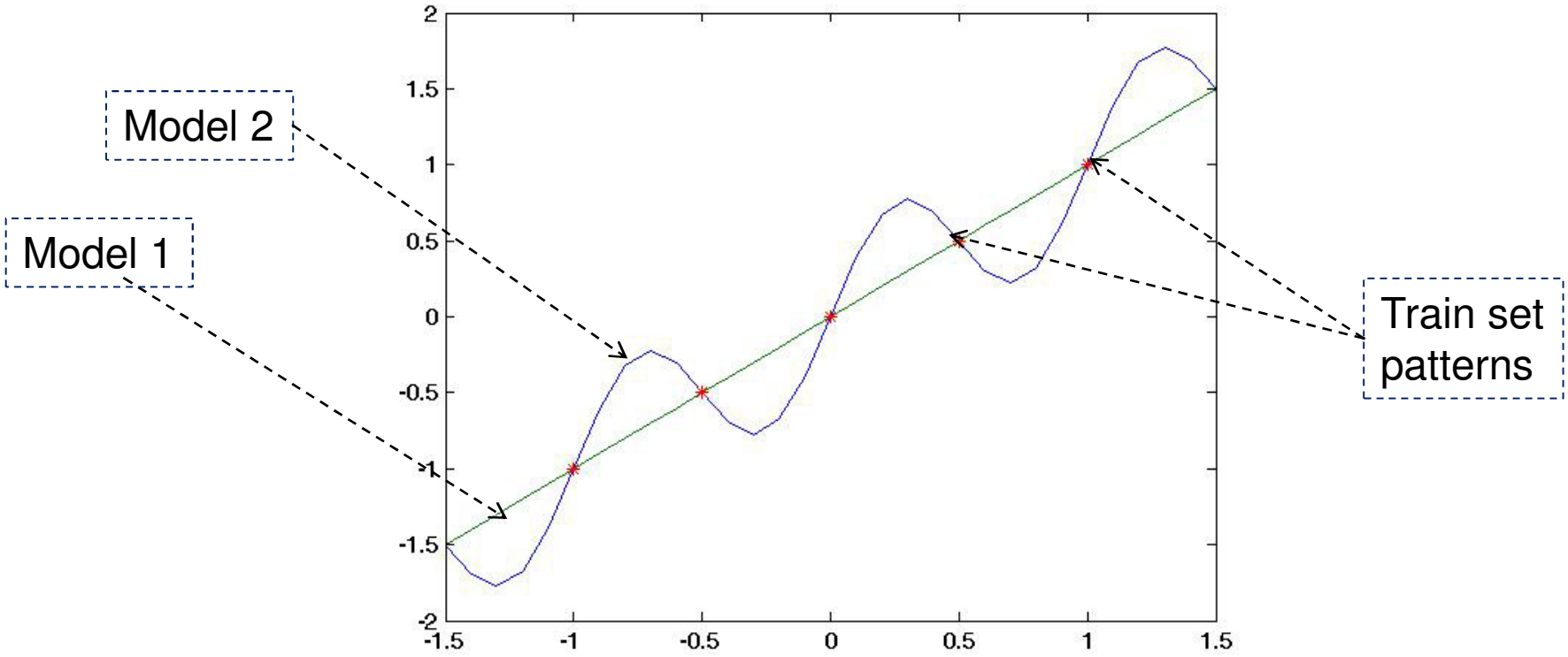
Such an assumption is about real life applications!

Next step … how do we measure simplicity/complexity?

UNIVERSITÀ DI SIENA 1240

Measuring simplicity

Better model 1, which is the simplest



UNIVERSITÀ DI SIENA 1240

Measuring simplicity

Intuitive ideas: the complexity of a parametric model depends on

- the number of roots the model can have
- the number of maxima/minima the model can have
- the number of patterns the model can interpolate
- the number of ways a set of patterns can be classified
- …

UNIVERSITÀ DI SIENA 1240

Measuring simplicity: Vapnik-Chervonenkis dimension (VCD)

Intuitive definition of "shatter"

- A classifier f_w is said to shatter a set of patterns $x_1, .. xk,$ if by changing the parameters we can classify the patterns in any possible way

Formal definition of "shatter"

- for any possible assigmament $t_1, .. t_k$, $t_i \in \{0,1\}$, there is a set of parameters w such that

$$f_w(x_i) > 0 \text{ if } t_i > 0$$
$$f_w(x_i) < 0 \text{ if } t_i < 0$$

UNIVERSITÀ DI SIENA 1240

Examples of shattering

Example 1

- A linear function $f_w(x) = w_1 x + w_0$ can shatter the set {1,2}... (and any set of 2 reals)

Example 2

- A polynomial $f_w(x) = w_3 x^3 + w_2 x^2 + w_1 x + w_0$ can shatter the set {0,1,2,3}... (and any set of 4 reals)

Example 3

- The function $f_w(x) = tanh(w_1 x + w_0)$ can shatter the set {1,2}... (and any set of 2 reals)

Example 4

- The function $f_w(x) = sign(sin(w_1 x + w_0))$ can shatter any set in R!!!

UNIVERSITÀ DI SIENA 1240

Vapnick-Chervonenkis dimension (VCD)

Intuitive definition

- The VCD dimension of classifier f_w is the dimension of the largest pattern set on which we can implement any classifier

Formal definition

$$VCD(f_w) = \max_X(|X|), \qquad \text{X is a set shattered by fw}$$

UNIVERSITÀ DI SIENA 1240

Intuitive ideas about VCD

VCD provides lower bounds on

- the maximum number of roots
  (models with a single input)
- the maximum number of minima/maxima
  (models with a single input)
- the maximum number of regions partitioned by the model
  (models with many inputs)


UNIVERSITÀ
DI SIENA
1240

UNIVERSITÀ DI SIENA 1240

Examples of VCD

Examples

- linear functions $f_w(x) = w_1 x + w_0$

  $VCD(f_w) = 2$

- Polynomials of order k, $f_w(x) = w_k x^k + \ldots + w_1 x + w_0$

  $VCD(f_w) = k+1$

- Neural networks with k neurons, single hidden layer, step activation function

  $VCD(f_w) = k+1$

UNIVERSITÀ DI SIENA 1240

VCD of neural networks

Neural networks with *p* parameters, bounds for the order of growth of VCD(f_w)

- FNNs with step activation function
  Upper bound: O(p log p),
  Lower bound o(p log p)
- FNNs with piecewise polynomial activation function
  Upper bound: $O(p^2)$,
  Lower bound $o(p^2)$
- FNNs with, tanh, logsig, atan activation function
  Upper bound: $O(p^4)$ … this may be overestimated
  Lower bound $o(p^2)$

UNIVERSITÀ DI SIENA 1240

VCD and test error

Vapnik proved that

- the problem is to learn a binary classifier f_w
- V is VC dimension of f_w
- Mean train error $e_{train} = \frac{1}{N}\sum_{i=1}^{N}|t_i - fw(xi)|$
- Mean test error $e_{test} = \frac{1}{N}\sum_{i=1}^{N2}|\overline{t_i} - fw(\overline{x_i})|$
- $0<\varepsilon<1$
- Train and test patterns (and targets) are drawn by the same distribution

Then

$$\mathrm{P}\left(e_{test} \leq e_{train} + \sqrt{\frac{V(\log(\frac{2N}{V}) + 1) - \log(\frac{1-\varepsilon}{4})}{N}} \right) \geq \varepsilon$$

UNIVERSITÀ DI SIENA 1240

VCD and test error

Vapnik proved that (Vapnik 1989)

$$\mathrm{P}\left(e_{test} \leq e_{train} + \sqrt{\frac{V(\log(\frac{2N}{V}) + 1) - \log(\frac{1-\varepsilon}{4})}{N}} \right) \geq \varepsilon$$

- The larger the number of train samples N, the smaller the generalization error
  - **Overfitting behaviour when training with few examples**

UNIVERSITÀ DI SIENA 1240

VCD and test error

Vapnik proved that (Vapnik 1989)

$$\mathrm{P}\left(e_{test} \leq e_{train} + \sqrt{\frac{V(\log(\frac{2N}{V}) + 1) - \log(\frac{1-\varepsilon}{4})}{N}} \right) \geq \varepsilon$$

- The larger VD dimension V, the larger generalization error
  - **Complex models produce worse generalization**

UNIVERSITÀ DI SIENA 1240

VCD and test error

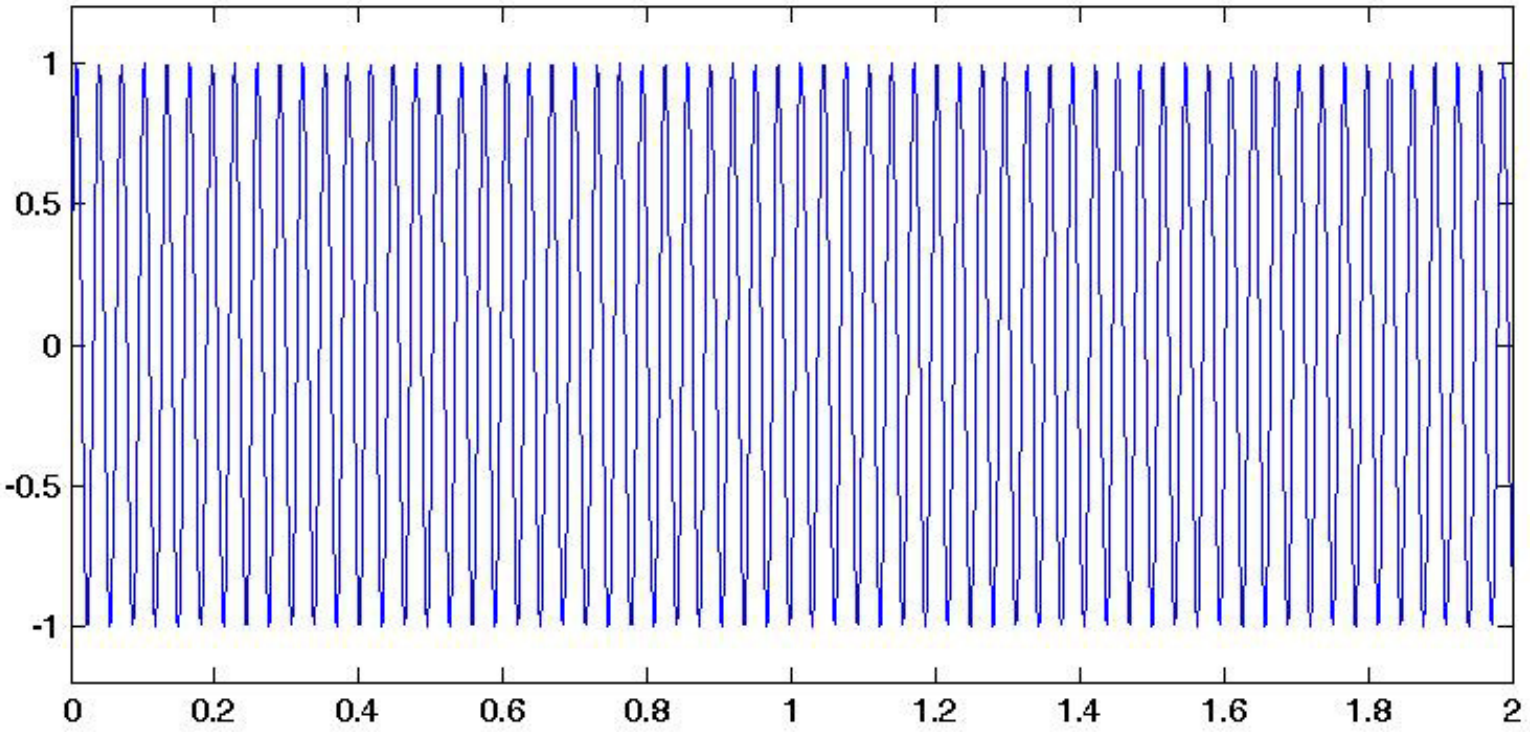
It has been proved also the converse

- When VCD is large then the generalization probability is large!!
- If VCD is infinitive, then it may be impossible to learn a model with bounded generalization error !!!
  - Can you guess why?

UNIVERSITÀ DI SIENA 1240

VCD and test error

The function *sin(w_1x), large w_1*



UNIVERSITÀ DI SIENA 1240

Generalization capablity in practice

In practice

- The VCD allows to estimate the error on test, but the VCD cannot be used in practice, since the bounds are too raw

Alternatives for choosing architectures, algorithms, ….

- Predict the perfomance on test set (by validation)
- Keeping weight small
  - This includes Support Vector Machine …. Not a matter of this course
- Pooling

UNIVERSITÀ DI SIENA 1240

Validation by random subsampling

Let *D* be the available dataset

1. Divide *D* randomly into a train T and a validation subset V
2. Train the model on T
3. Evaluate the mode on the validation set N
4. Repeat *k* times
5. Calculate the average error rate

UNIVERSITÀ DI SIENA 1240

Validation k-fold cross validation

Let *D* be the available dataset

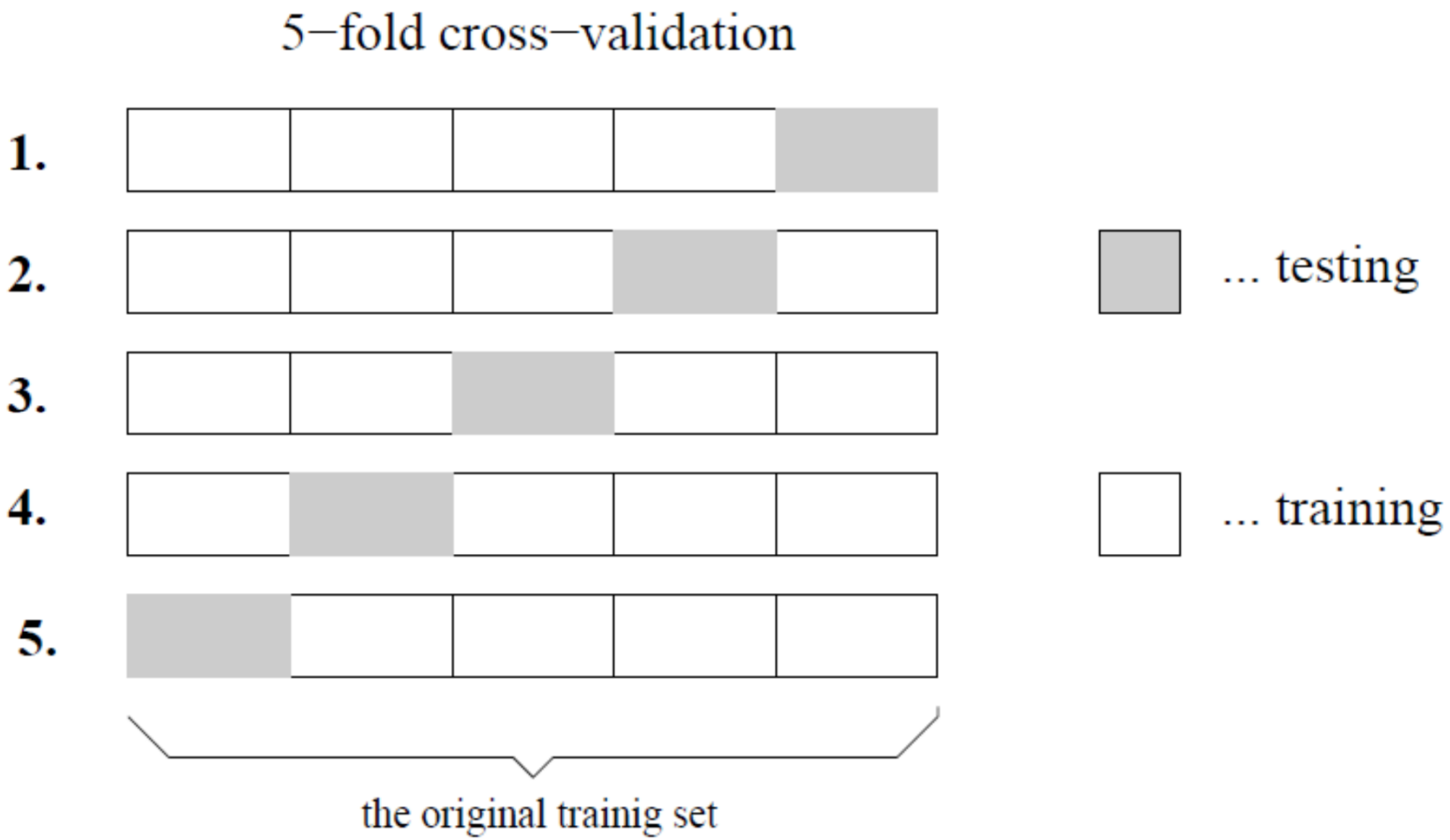
1. Divide *D* randomly into a train T1,..,Tk subsets
2. For i=1 to k

   Train the model on all the set except T_i

   Evaluate the model on T_i
3. Calculate the average error rate

UNIVERSITÀ DI SIENA 1240

Validation k-fold cross validation



UNIVERSITÀ DI SIENA 1240

Validation: why does it work?

Why does it work?

- Validation just allows to experimentally predict the error on test set
- We must assume that validation set is drawn from the same distribution of test and train set

UNIVERSITÀ DI SIENA 1240

Validation: what is it useful for ?

- To compare different models (neural networks, Bayesian models, …)
- To compare different architectures (number of layers, number of neurons, …)
- Decide when to stop learning
- ….

UNIVERSITÀ DI SIENA 1240

The role of weight sizes in neural networks

Does weight size affect generalization?

- network with small weights produce smooth function
- smooth functions look simpler than non-smooth ones

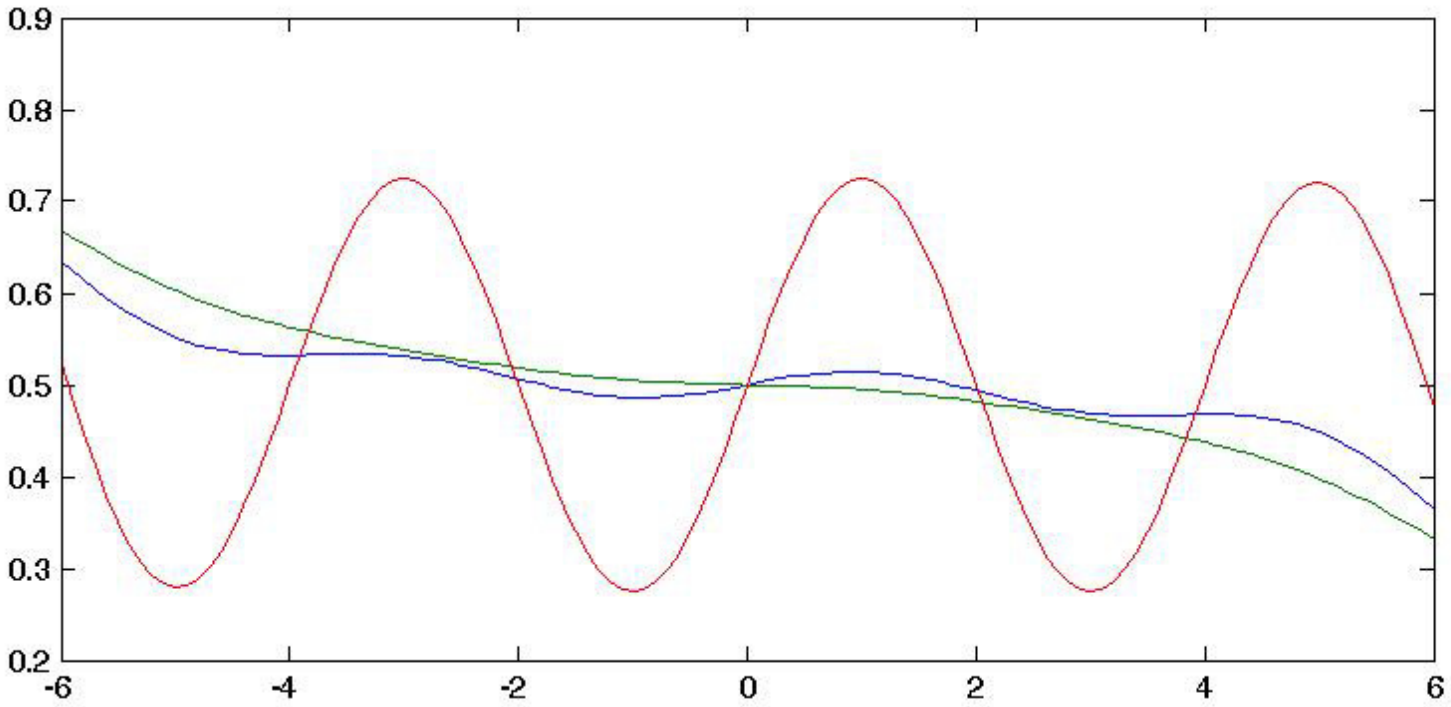
Notice

- Networks with small weights are universal approximators ….
  - Can you prove this?

UNIVERSITÀ DI SIENA 1240

The role of weight sizes in neural networks

A neural network a single layer and different weights sizes (w=0.3, 0.4, 0.5)



UNIVERSITÀ DI SIENA 1240

VCD and weight size

An extended version of VC dimension (Bartlet)

- Usual mean test error $e_{test} = \frac{1}{N}\sum_{i=1}^{N2} |\overline{t_i} - fw(\overline{x_i})|$
- Error on train set
  (those patterns for which the output is small are errors)
  $e_{train}^{\gamma} = \frac{1}{N}\sum_{i=1}^{N} d_{\gamma}(t_i, f_w(xi))$,
  $d_{\gamma}(ti, f_w(xi)) = 1$ if $t_i \bullet f_w(xi) \leq \gamma$, $d_{\gamma}(ti, f_w(xi)) = 0$ otherwise
- $V(\gamma)$ fat-shattering dimension
  - $V(\gamma)$ is the maximum dimension of a set shattered with error smaller than γ

Then

$$P\left(e_{test} \leq e_{train}^{\gamma} + \sqrt{2\frac{V(\gamma/16)\ \ln(\frac{34eN}{V(\gamma/16)})\log(578N)\ +\ln(\frac{4}{1-\varepsilon})}{N}} \right) \geq \varepsilon$$

UNIVERSITÀ DI SIENA 1240

VCD and weight size

An approximated version of VC dimension (Bartlet)

$$\mathrm{P}\left(e_{test} \leq e_{train}{}^{\gamma} + \sqrt{2\frac{V(\gamma/16)\,\ln(\frac{34eN}{V(\gamma/16)})\log(578N) + \ln(\frac{4}{1-\varepsilon})}{N}} \right) \leq \varepsilon$$

- The bound has similar properties w.r.t those of VC,
  - The larger $V(\gamma)$ the worse the generalization
- even if
  - $e_{train}{}^{\gamma}$ is larger than e_{train}
  - $V(\gamma)$ is larger than V

VCD and weight size

Bartlet proved that

- Neural network with sigmoidal activation function, having output in [-M/2,M/2] and module of derivative smaller than β
- Module of input smaller than B
- L layers
- **Weights bounded by α**

Weight dimension

$$V(\ ) \leq \frac{4B^2}{\gamma^2}\left(\frac{M}{\gamma}\right)^{2(L-1)} (2\alpha\beta)^{L(L+1)} log(3^{L-1}(L-1)!\,(2n-1))r^2$$

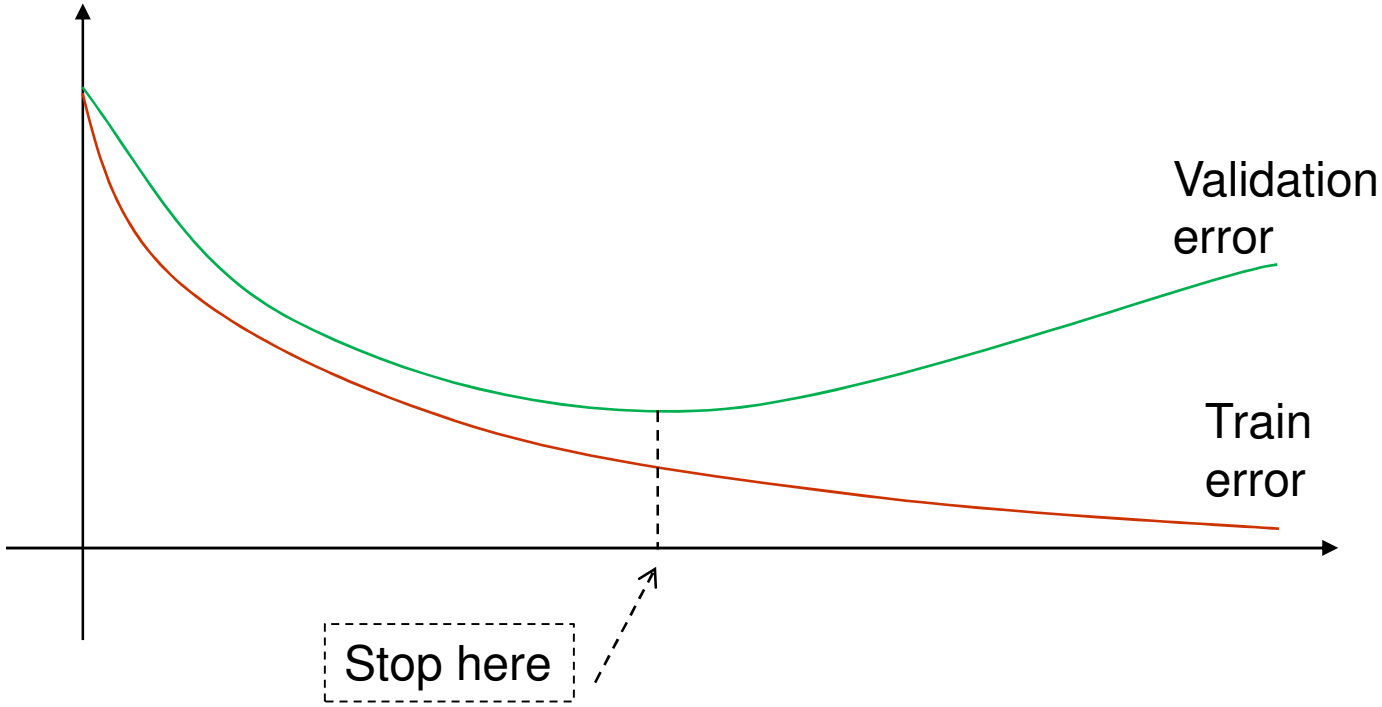
- **The smaller the weights, the smaller the fat-shattering dimension $V(\gamma)$!!**

UNIVERSITÀ DI SIENA 1240

Keeping weight small

Early stopping

- Initialize the weights to small values
- Train the network
  - stop when error on validation increases



UNIVERSITÀ DI SIENA 1240

Keeping weight small

Penalty (weight decay)

- Add penalty on a weight to error

$$e_{train} = \frac{1}{N}\sum_{i=1}^{N}(t_i - fw(xi)) + \lambda p(w)$$

where

$$p(w) = \sum_i {w_i}^2$$

Notice that

$$\frac{\partial p(w)}{\partial w_i} = -2wi$$

Weight decay

UNIVERSITÀ DI SIENA 1240

Keeping weight small

Constraint on neuron weights

- For each neuron k, activate a constraint when the input weight is larger than a given maximum

$$p_k(w) = \begin{cases} \sum_i {w_{ik}}^2 - M & if \sum_i {w_{ik}}^2 > M \\ 0 & otherwise \end{cases}$$

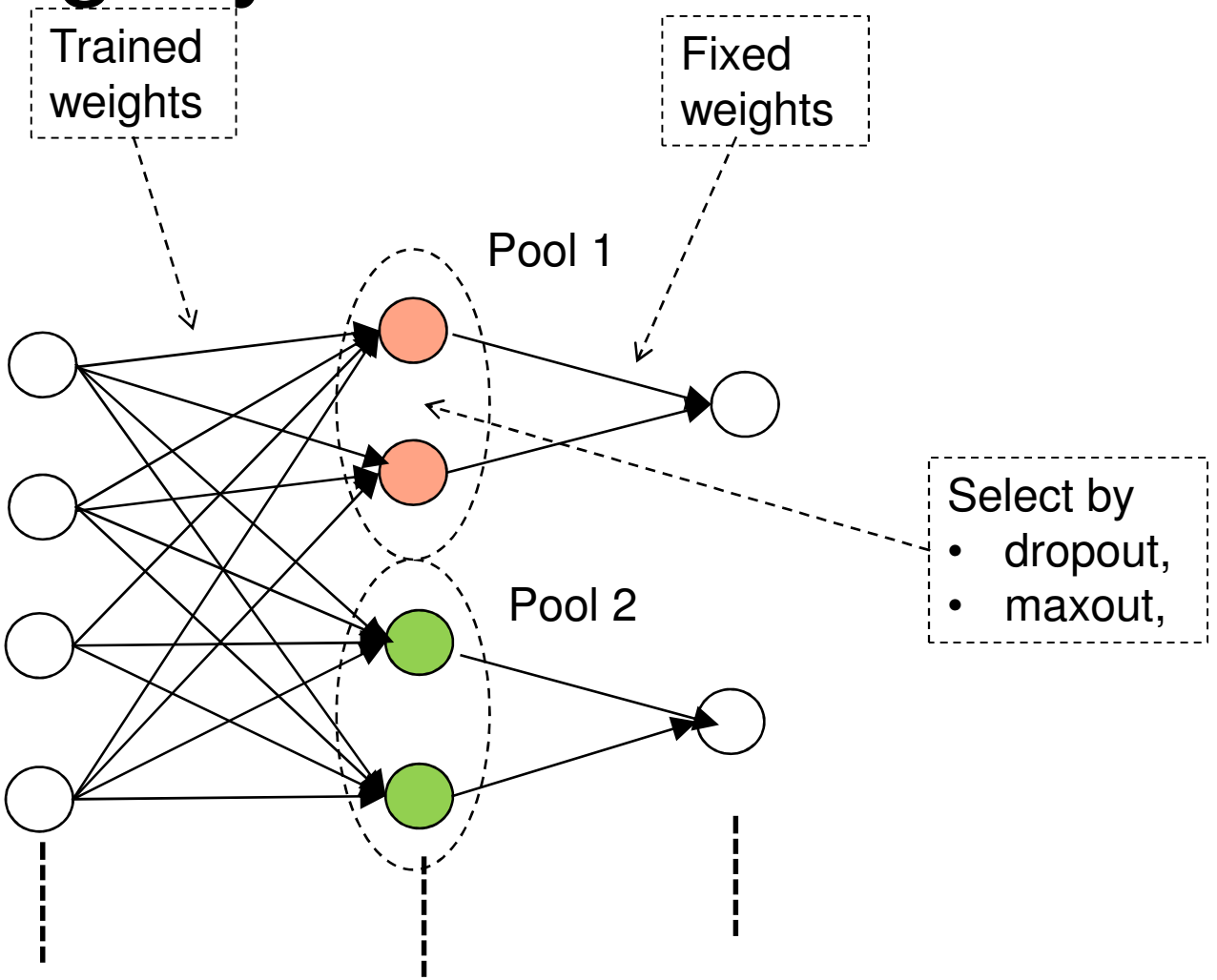
UNIVERSITÀ DI SIENA 1240

Pooling layers

- Neurons of a layer are grouped in subset
- For each pattern, only a fraction of the neurons in a group are activated
  - The output of the other neurons is not considered
- The active neurons an a group are selected by
  - Taking the max (maxout)
  - Taking a random set (dropout)

UNIVERSITÀ DI SIENA 1240

Pooling layers



UNIVERSITÀ DI SIENA 1240

Pooling layers

- Pooling layers reduce parameters and neurons
- Pooling layers reduce VC dimension

UNIVERSITÀ DI SIENA 1240

Other explanations

- Early stopping helps because it predicts the test set performance
- Weight decay disactivate some of the weights
- Pooling removes similar features
- …

UNIVERSITÀ DI SIENA 1240

Final practical remarks about generalization capability

In pratice

- In general
  - the largest the model architecture, the smallest the generalization
  - the largest the weights, the smallest the generalization
  - The smallest the train set, the smallest generalization
- But remember that the generalization capability depends in complex way on
  - the model architecture
    (the type, the number of weights/neurons, the size of the weights),
  - the problem
    (the type, the number of examples in train set)

UNIVERSITÀ DI SIENA 1240

Approximation, learning, generalization: the global picture

Antagonist goals

- Larger models improve approximation and learning, but they decrease generalization
- Larger weights improve approximation, but they decrease generalization
- Larger trainset improve generalization, but makes learning more difficult

UNIVERSITÀ DI SIENA 1240

Approximation, learning, generalization: the global picture

	approximation	**learning**	**generalization**
Large train set		worse	better
Large weights	better	?	worse
Large model	better	better	worse

UNIVERSITÀ DI SIENA 1240

Other constraints to considered to select the architecture

Constraints from the considered problem

- There is a minimum dimension for the model
  - In a practical application, the approximation (error) cannot be smaller than a minimum
  - Too small models cannot reach such a approximation minimum
- There is a maximum amount of data available for training and validation
  - Collecting/labelling data is expensive, …
- The computation resources are bounded
  - Dimension of train set and model dimension affect the required computational resources

UNIVERSITÀ DI SIENA 1240

Why generalization may be different from what expected

It is assumed that the patterns of train, validation, and test sets are drawn from the same distribution

- The distribution is different in most of real life applications (performance on test much worse than on validation/train)
- Patterns in test may appear also in training (performance on test better than expected, when a lookup table is used)
- Often there are relationships between patterns
  - Patterns are not independent
  - Using relationships improve performance on test

UNIVERSITÀ DI SIENA 1240

Our analysis of generalization does not include reliability

Reliability

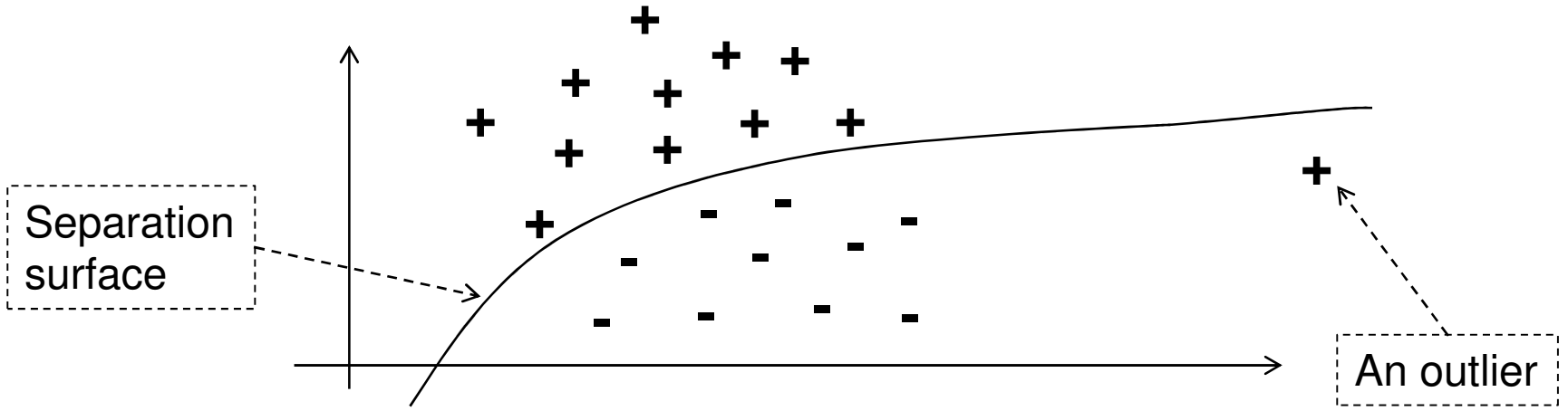
- A measure of how much you can trust the prediction
- Very important in verification problems
- Generalization theory is not of help
  - It tells you how many errors your model will do on the whole test set
  - It does tell you anything about the reliability of the prediction on a single pattern

+

UNIVERSITÀ DI SIENA 1240

Reliability

- A measure of how much you can trust the prediction
- Very important in verification problems
- Common neural networks do not provide a reliability measure
- The prediction may be very unreliable for outliers!
  - Separation surface is unreliable where there are no train patterns



UNIVERSITÀ DI SIENA 1240

Reliability in machine learning

Predictive models (bayesian models, autoencoder)

- Model trainset distribution
- Predict the probability that a pattern is generated from the same distribution as that of training set
- Good to recognize outliers
- Good for verification problems

Discriminative models (common neural networks, SVMs,..)

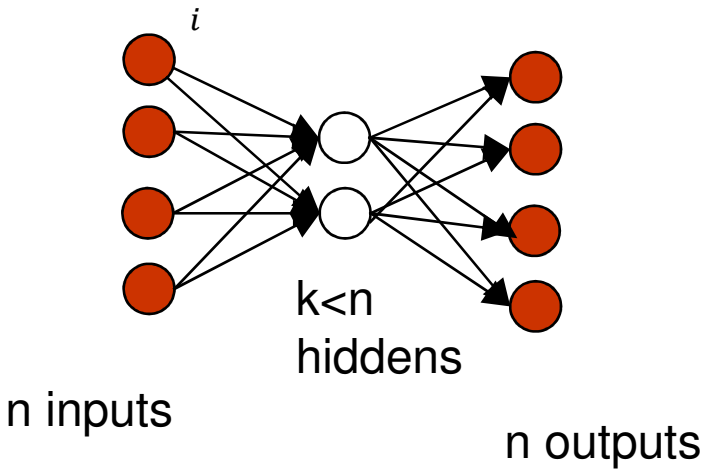
- Do not model trainset distribution
- Predict the most probable class (or targets)
- Good for classification problems

UNIVERSITÀ DI SIENA 1240

Autoencoders

- Input and output layer have the same number of neurons *n*
- One (or many) hidden layer with k<n neurons
- The network is trained to copy input x to output $f_w(x_i$

$$e_{train} = \sum_i (x_i - fw(xi))^2$$



UNIVERSITÀ DI SIENA 1240

Autoencoders: how they are used

For verificaton

- During trainining, the autoencoder is trained to copy input to ourput
- During test, we use $(x-f_w(x))^2$ as a measure of the probability that x belong to the train set distribution

For classificaton into k classes

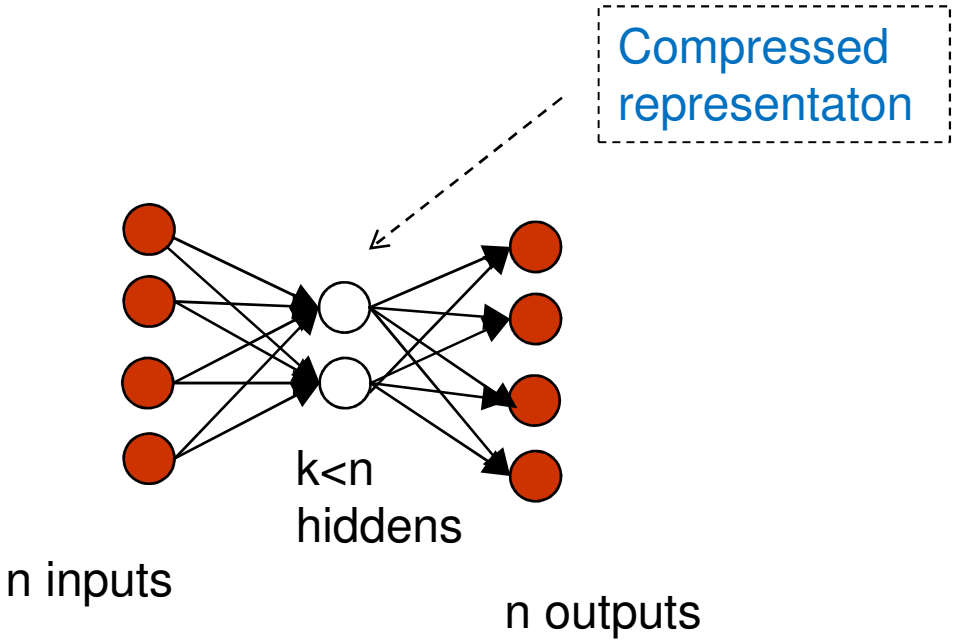
- During trainining, the k-th autoencoder is trained to copy input to output using only positive patterns of each class
  - Eventually error is changed to accommodate negative examples
- During test, it is returned the class of the autoencoder that obtains the smallest errror $(x-f_w(x))^2$

UNIVERSITÀ DI SIENA 1240

Autoencoders: how they are used

For feature dimension reduction

- During trainining, the autoencoder is trained to copy input to ourput
- During test, the hidden node output is used as compressed representation of the features

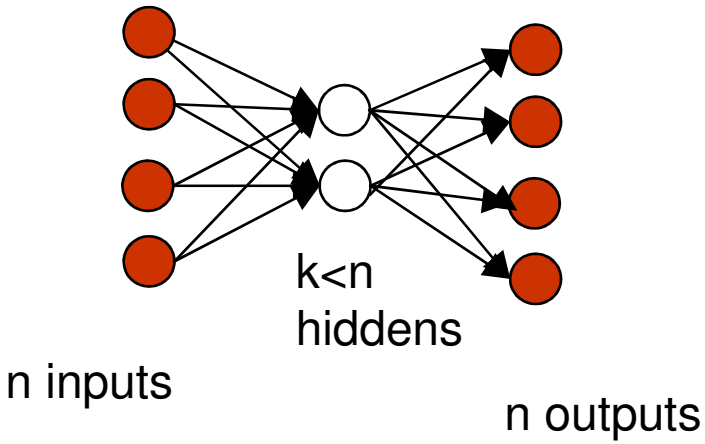


UNIVERSITÀ DI SIENA 1240

Autoencoders: why they work

Why autoencoders work

- The autoencoder cannot approximate the identity function
- The autoencoder compresses the input information into a smaller space
  - The smaller the hidden layers, the larger the compression



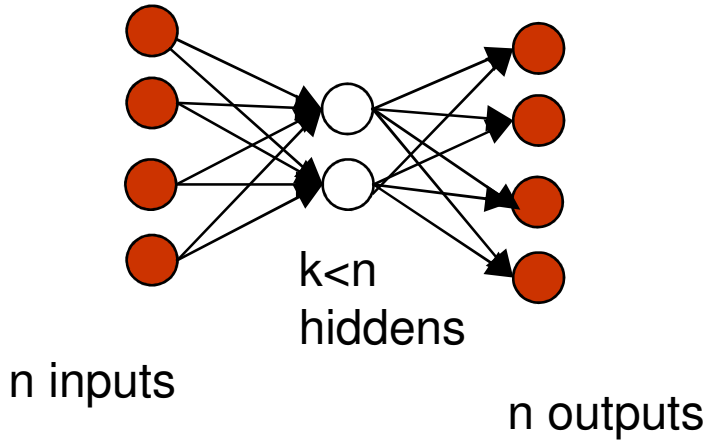
UNIVERSITÀ DI SIENA 1240

Autoencoders: why they work

- The function implemented by an autoencoder is

$$f(x) = W_2\sigma(W_1x + b_1) + b_2$$

- An autoencoder implements a sort of non linear version of PCA (Principal component analysis)



UNIVERSITÀ DI SIENA 1240

Autoencoders vs common networks

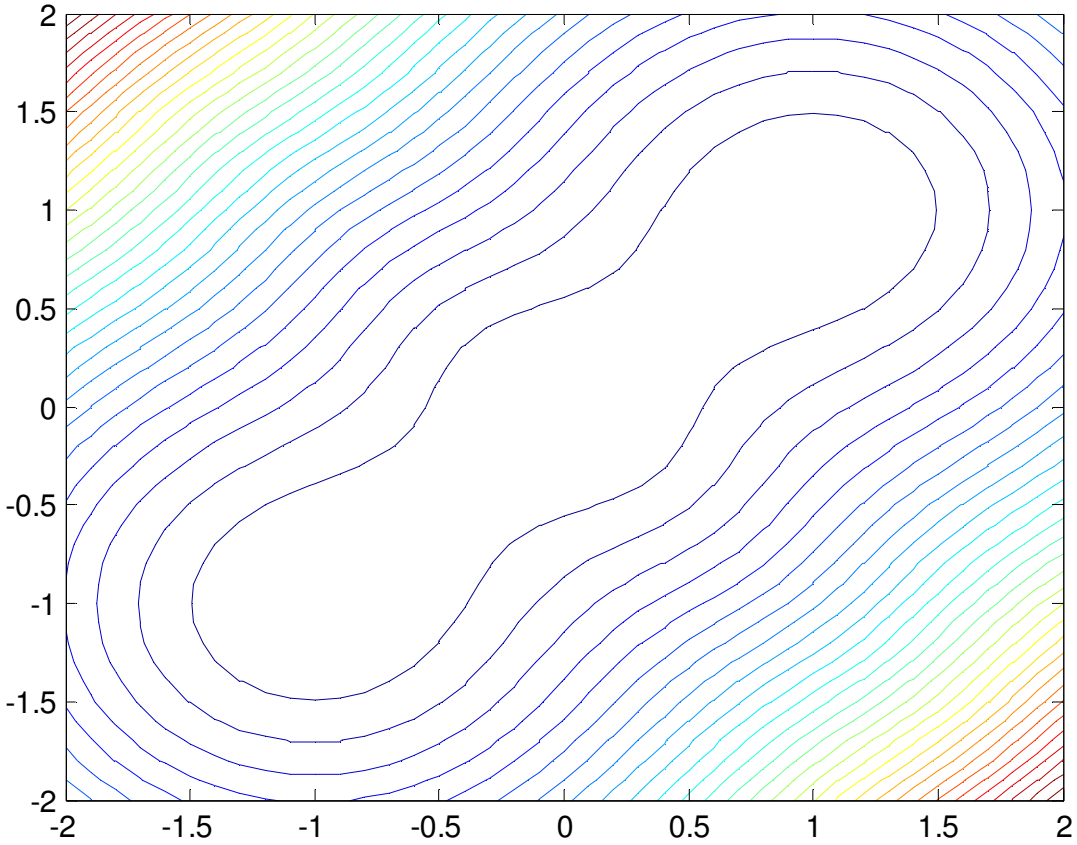
It has been proved that (Gori … and me)

- The separation surfaces for autoencoders are always closed
  (provided that the number of hidden neurons is smaller than the number of inputs)
- The separation surface of a FNN can be both open and closed

UNIVERSITÀ DI SIENA 1240

Autoencoders vs common networks

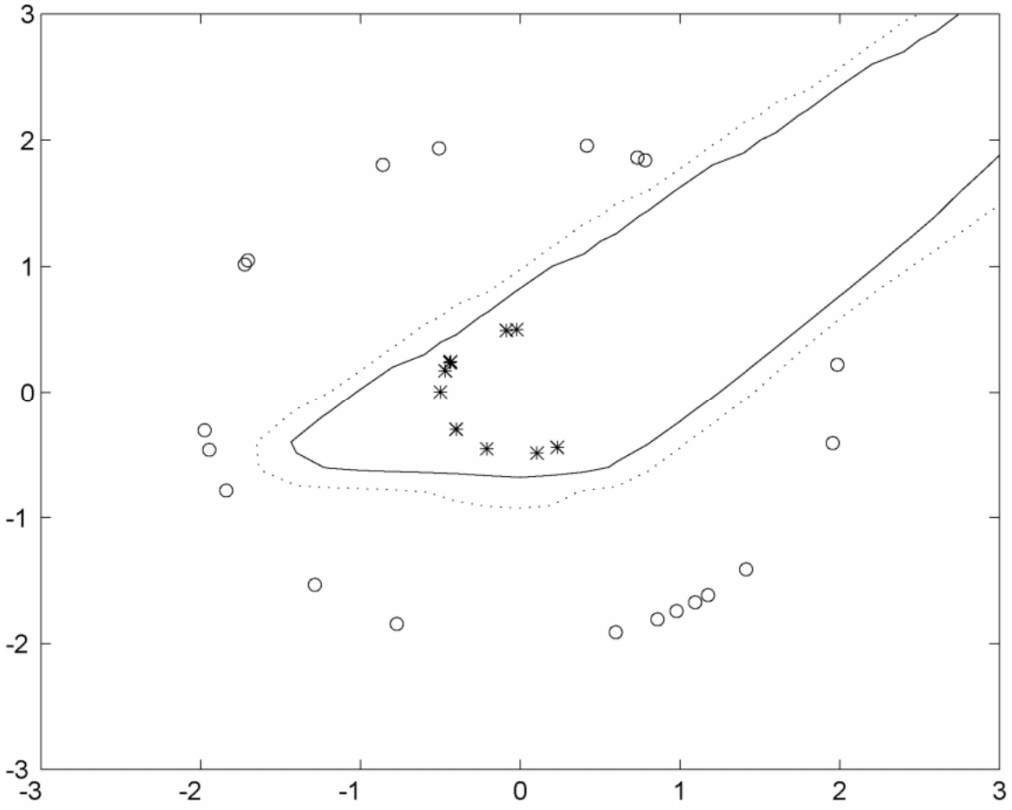
- A contour plot of an autoencoder



UNIVERSITÀ DI SIENA 1240

Autoencoders vs common networks

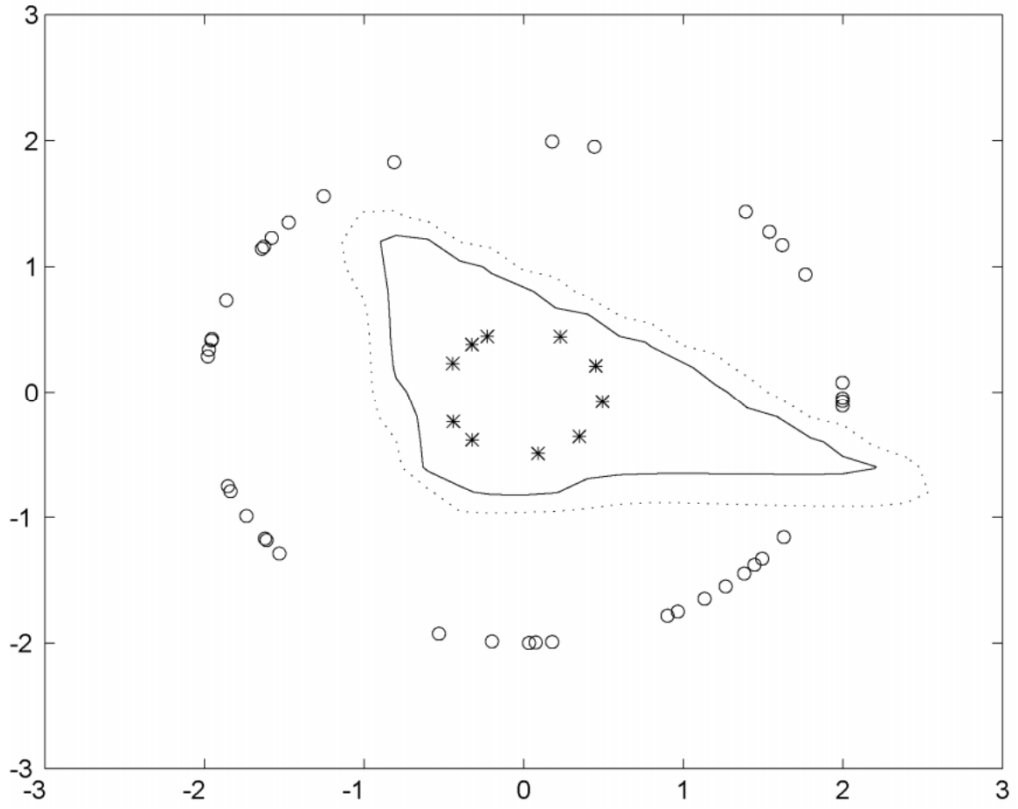
The separation surface of a feedforward network with 5 hiddens



UNIVERSITÀ DI SIENA 1240

Autoencoders vs common networks

The separation surface of a feedforward network with 5 hiddens



UNIVERSITÀ DI SIENA 1240

Autoencoders vs common networks

Intuitively

- Autoencoders are advantageous for verification
  - due to closed surface fact
  - due to the fact that it is a predictve model

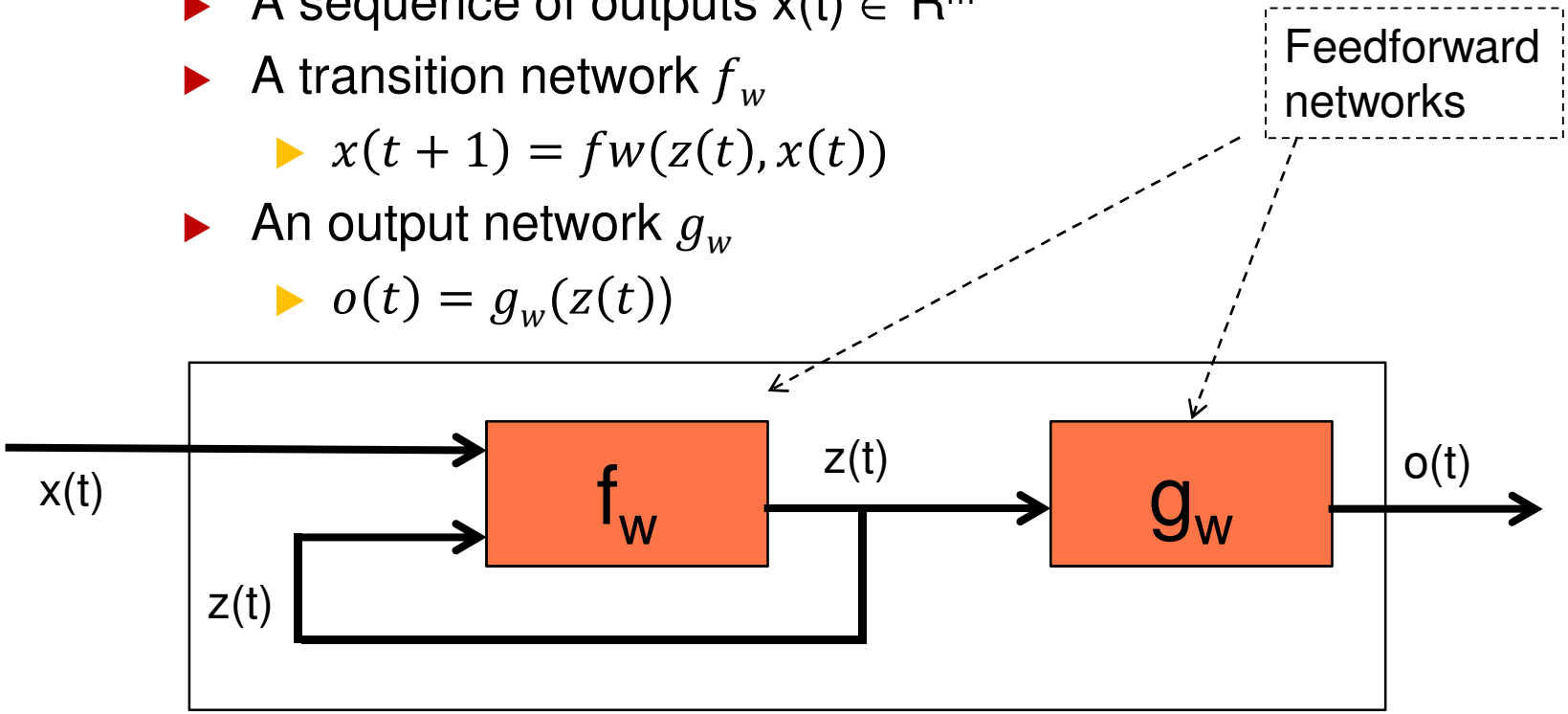
Recurrent neural networks

UNIVERSITÀ DI SIENA 1240

Recurrent neural networks (RNNs)

Dynamical neural networks

- An internal state $z(t) \in R^s$
- A sequence of inputs $x(t) \in R^n$
- A sequence of outputs $x(t) \in R^m$
- A transition network f_w
  - $x(t+1) = fw(z(t), x(t))$
- An output network g_w
  - $o(t) = g_w(z(t))$



UNIVERSITÀ DI SIENA 1240

Training recurrent neural networks

The train set

- A pattern is a pair of sequences of inputs and desired outputs
- $L = \{(\bar{x}, \bar{t}) | \bar{x} = x(0), x(1), \ldots x(T), \bar{t} = t(1), .. t(T)\}$

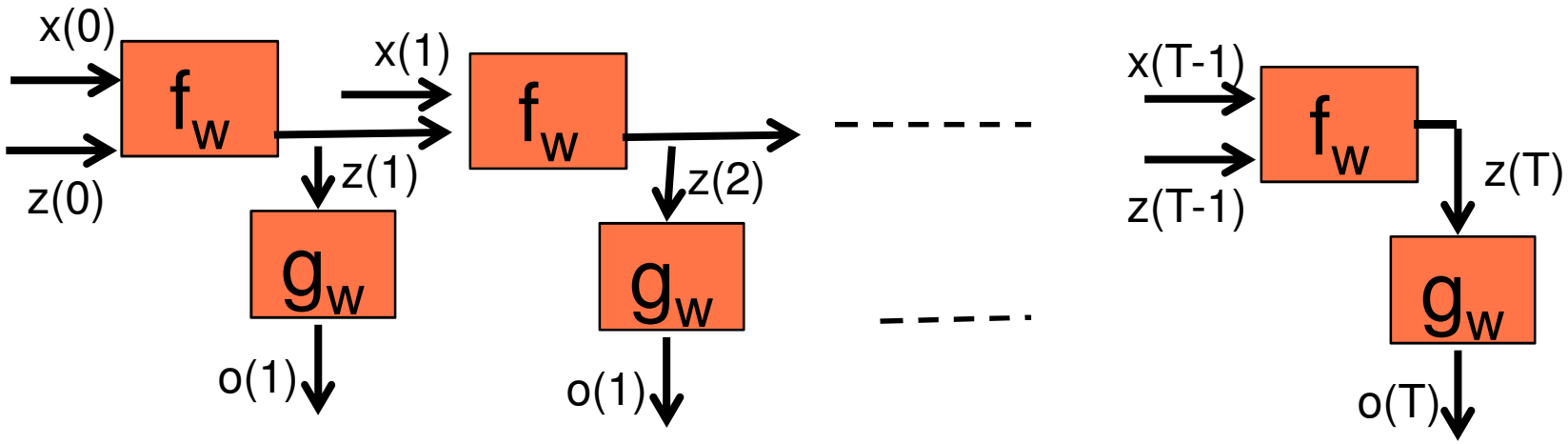
Training recurrent neural networks

Unfolding network

- A copy of f_w, g_w for each time instance, connected in sequence

The unfolding network is a feedforward network (with shared weights)

- The unfolding network can be trained by standard backpropagation
  (just remember to accumulate gradients)



UNIVERSITÀ DI SIENA 1240

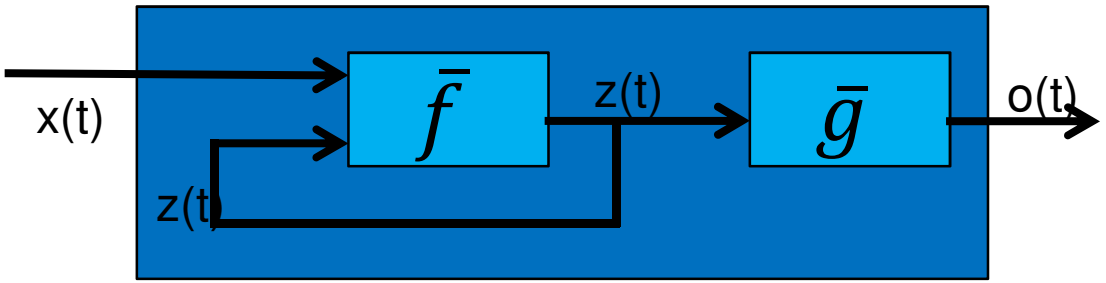
Approximation capability of RNNs

RNN approximation ver 1.0

- given dynamical system, can it be approximately simulated by a RNN?

Formally, is there a RNN that simulates the following?

- a transition function $\bar{f}$
- an output function $\bar{g}$

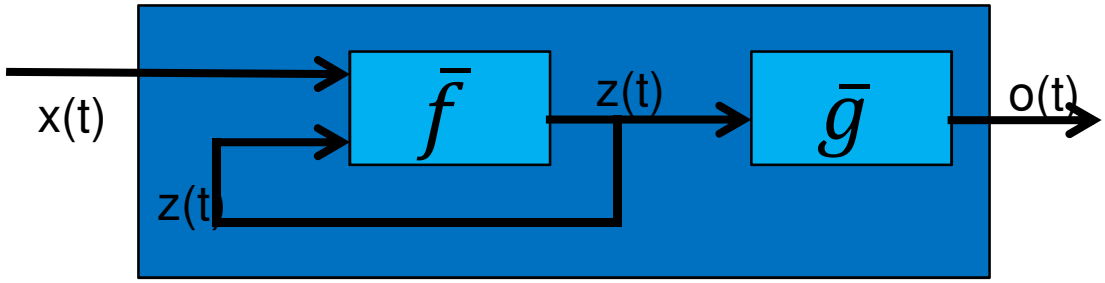


UNIVERSITÀ DI SIENA 1240

Approximation capability of RNNs

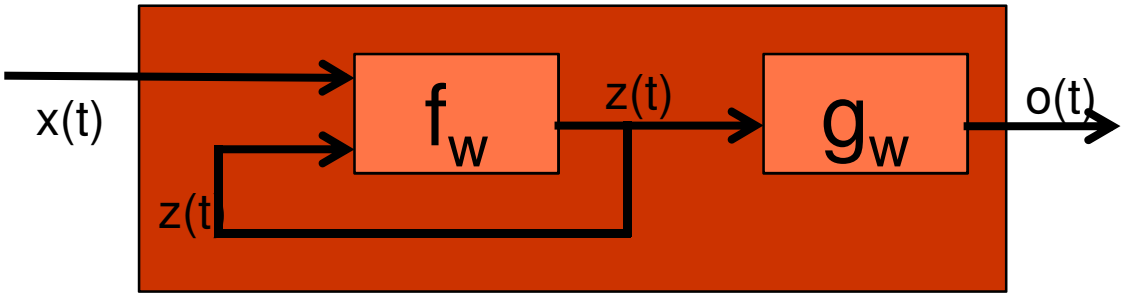
Obvious answer

- Yes
  Just take neural networks that approximate $\bar{f}$ and $\bar{g}$



Target dynamic system

RNN



UNIVERSITÀ DI SIENA 1240

Simulating a dynamical system

Obvious answer

- Yes
  Just take neural networks that approximate $\bar{f}$ and $\bar{g}$

But notice

- We implicitly assumed the state dimension is known (at least finite)
- The transition and output networks must have at least one hidden layer
- The sequences of outputs generated by the two dynamical system may diverges with time

UNIVERSITÀ DI SIENA 1240

Approximation capability

An advanced question

- Let us consider a function t that takes in input sequences of patterns $\overline{x(T)} = x(0), x(1), \ldots x(T)$ and returns an output $t(\overline{x(T)}) = o(T)$
- Can a RNN approximate t?

An intuitive answer

- Yes, provided that
  1. The transition network can code the input sequences (or all the relevant information) and store into the state z(t)
  2. The output network can decode such a representation and produce the output
- If the coding exists, the thesis is true by universal approximation

UNIVERSITÀ DI SIENA 1240

Approximating sequences

- Yes, provided that
  1. The transition network can code the input sequences (or all the relevant information) and store into the state z(t)

Simple case: the dimension of the state must be large enough

- It must hold s > T*n
- just copy inputs to the state

The general case, s<T*n

- a set of integers v_1, ..,v_k can be coded with the real number 0.v_1, ..v_k
  - E.g. 0,1234 is a coding of 1, 2, 3, 4
- a set of real number v_1, ..,v_k can be approximately coded with the real number 0.[v_1] ..[v_k], where [.] is the rounding
  - E.g. 0,1234 is a coding of 1.12, 2.15, 3.41, 4.32

UNIVERSITÀ DI SIENA 1240

Approximating sequences

It has been proved that (Hammer)

- Let us consider a function t that takes in input sequences of patterns $\overline{x(T)} = x(0), x(1), \ldots x(T)$ and returns an output $t(\overline{x(T)}) = o(T)$
- t can be approximated by a RNN which is feeded with x(i) and return o(i) for each i
  - If t is measurable and the $x(0), x(1), \ldots x(T)$, are reals, then the approximation is only in probability
  - If t is continuous and the $x(0), x(1), \ldots x(T)$, are integers, then the approximation is w.r.t. sup norm

UNIVERSITÀ DI SIENA 1240

RNN approximation in practice

- when s<<T*n, the coding function exists, but
  - it is complex
  - It is very sensible to noise
  - .. so it is difficult to learn
- In real life tasks
  - only a small part of the information in inputs is useful
  - Inputs are recursively processed
  - Inputs belong to a sub.mainfold
  - … information to be stored is much smaller and a much smaller state dimension is required

UNIVERSITÀ DI SIENA 1240

Learning capability of RNNs

Intuitively

- Known results recall that feedforward networks

It has been proved that (Bianchini et alt)

- a RNN with one layer network
- error has no local minima, if
  - the matrix of the inputs $x^1(0)$, …, $x^1(T)$, $x^2(1)$… $x^2(T)$, … is full-rank
  - Thus, the total number of time instances cannot larger than input dimension

UNIVERSITÀ DI SIENA 1240

Learning capability of RNNs

Intuitively

- Known results recall that feedforward networks

It has been proved that (Bianchini et alt)

- a RNN with one layer network
- error has no local minima, if
  - The sequences are linearly separable
  - The weights of links connecting states to states are positive

UNIVERSITÀ DI SIENA 1240

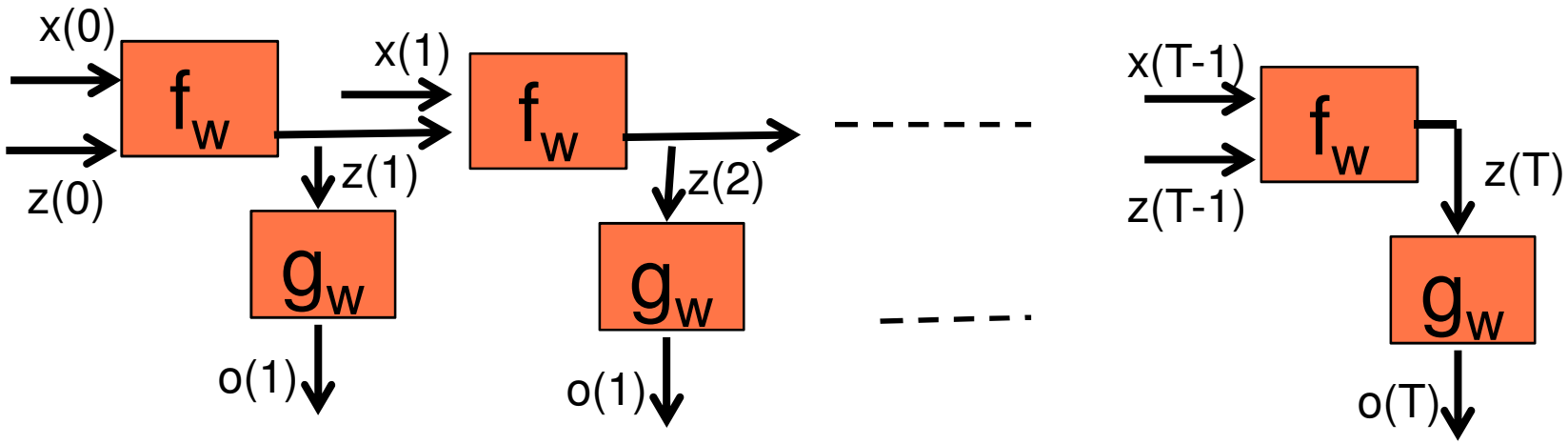
Long-term dependencies

Intuitively

- It is difficult to learn a RNN when the at time T depends on the input at time t and t<<T

Common explanation

- The gradient become smaller and smaller when propagated through the layers of the unfolding network



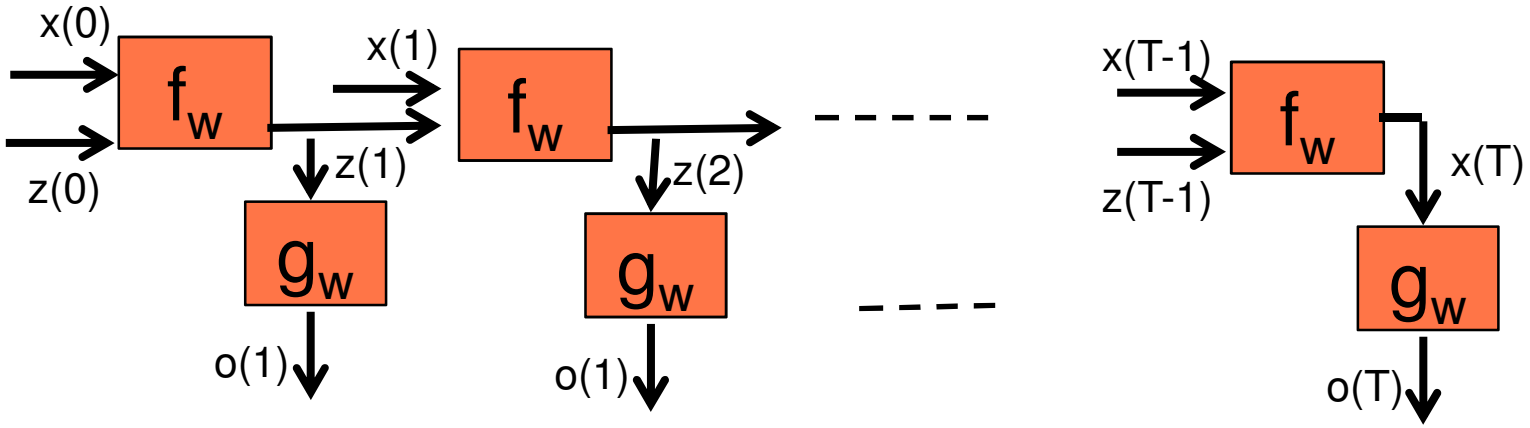
Common explanation

Remember

- $e_w(T) = (l - ow(T))^2$
- $\frac{\partial ew(T)}{\partial x(0)} = \frac{\partial ew(T)}{\partial x(T)} \frac{\partial Z(T)}{\partial Z(T-1)} \frac{\partial z(T-1)}{\partial z(T-2)} \cdots \cdots \frac{\partial z(1)}{\partial x(0)}$

The term $\frac{\partial z(t+1)}{\partial z(t)}$

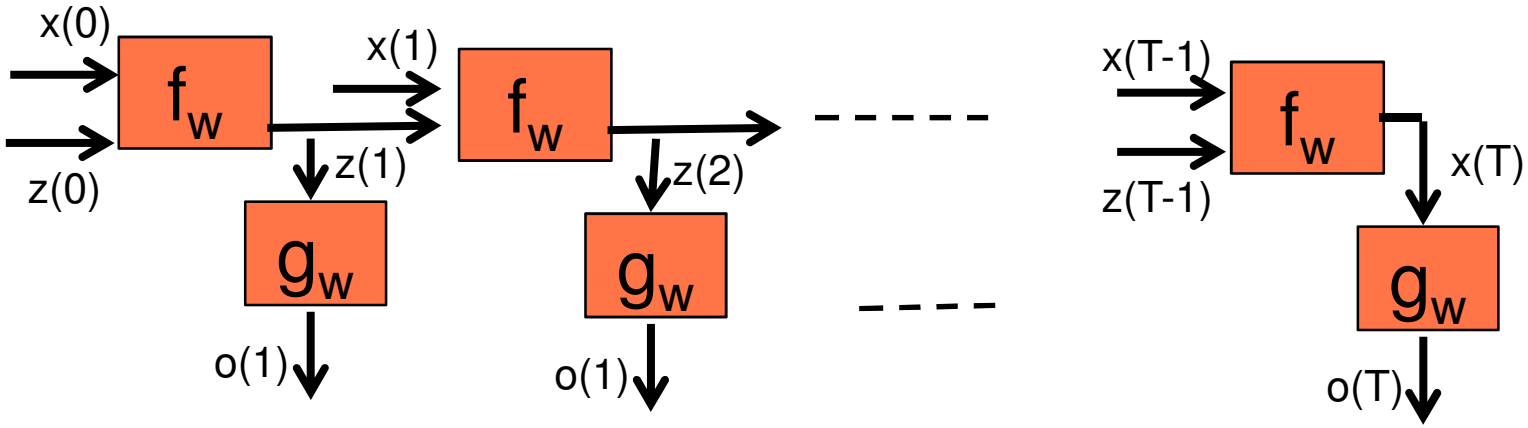
- A sxs matrix measuring how the input of f_w affects its output
- When $\left\|\frac{\partial z(t+1)}{\partial z(t)}\right\|$<1 1 and T is large $\left\|\frac{\partial e_w(T)}{\partial x(0)}\right\|$ is close to 0



Common explanation

$\left\| \frac{\partial z(t+1)}{\partial Z(t)} \right\|$ is small where the function f_w is flat

- saturated neurons
- small weights

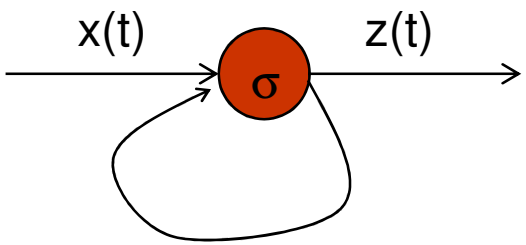


UNIVERSITÀ DI SIENA 1240

Long-term dependencies: some experiments

Bengio et alt.

- The problem: learning to latch
  - An input sequence x(0)…x(T)
  - Two classes to be reconignized which depend only on x(0)
  - The expected output at time T o(T)>0 for class A, o(t)<= fo class B
- The network
  - A RNN with a single hidden neuron
  - σ=tanh
  - z(t)=w σ(z(t-1))+x(t)

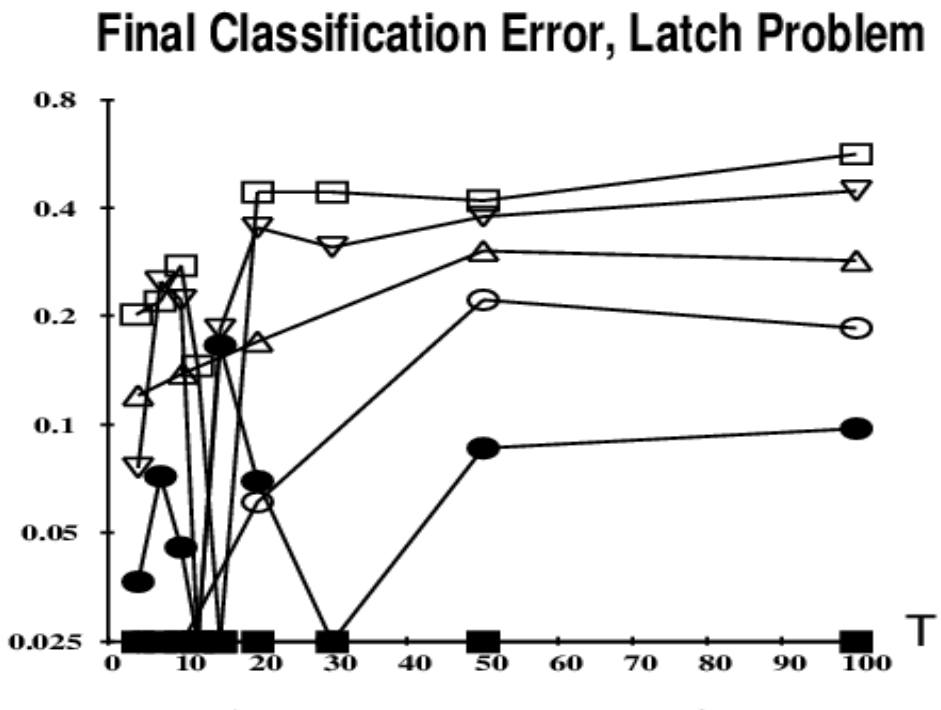


UNIVERSITÀ DI SIENA 1240

Long-term dependencies: some experiments

Bengio et alt.

- The classification task is learned only for small T (T<20)
- Some tricks with learning algorithms improved only a few the results

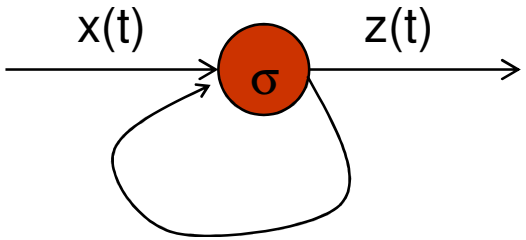


UNIVERSITÀ DI SIENA 1240

Long-term dependencies: the theory

Bengio et alt.

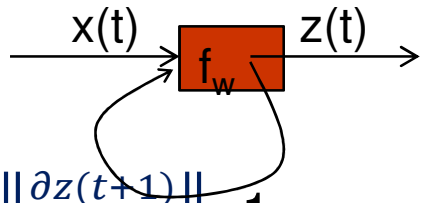
- .If w>1 then the system has two attracting stable states, a positive z^+, and a negative stable state z^-
  - Intuitively: the system is able to store some information only if w>1
  - This is true in general, if the weights are too small then the RNN has only one stable point!!|!
- If z(0)>0, z(0) is large enough and |x(t)| not too large then z(T)>0 (the converse hold if z(0)<0)
  - Intuitively: the system resets the stored information when the input is large or small enough
  - The information storing is robust for small inputs

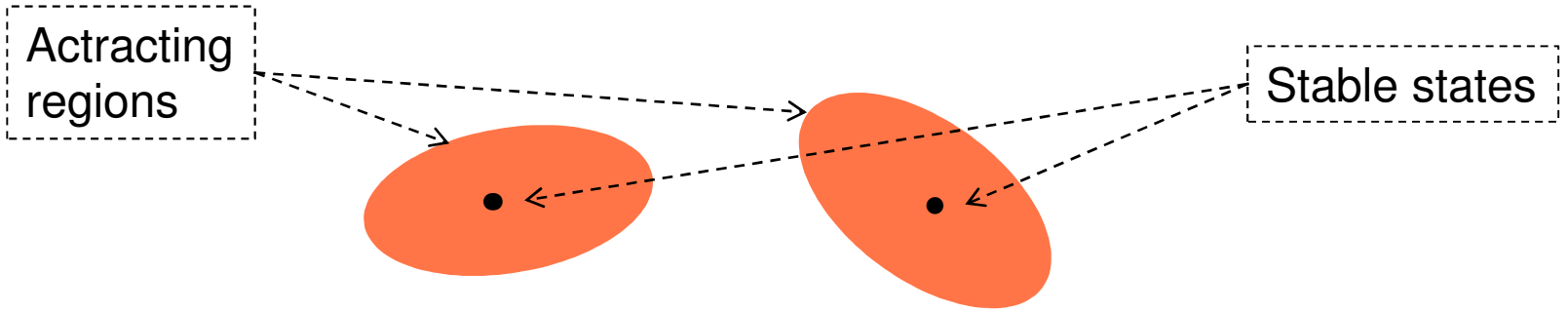


Long-term dependencies: the theory

Bengio et alt.

- A more general network with any number of neurons
  - $z(t)=f_w(z(t-1))+x(t)$
- The system may have several attracting points
- There are regions close to those points where $\left\|\frac{\partial z(t+1)}{\partial z(t)}\right\|<1$
- Those regions
  - make information latching robust
  - But if inputs do not move the state from an attracting regions, then $\lim_{T\to\infty}\frac{\partial e(T)}{\partial x(0)}=0$





UNIVERSITÀ DI SIENA 1240

Long-term dependencies: other explanations

Loading problem is difficult with many inputs

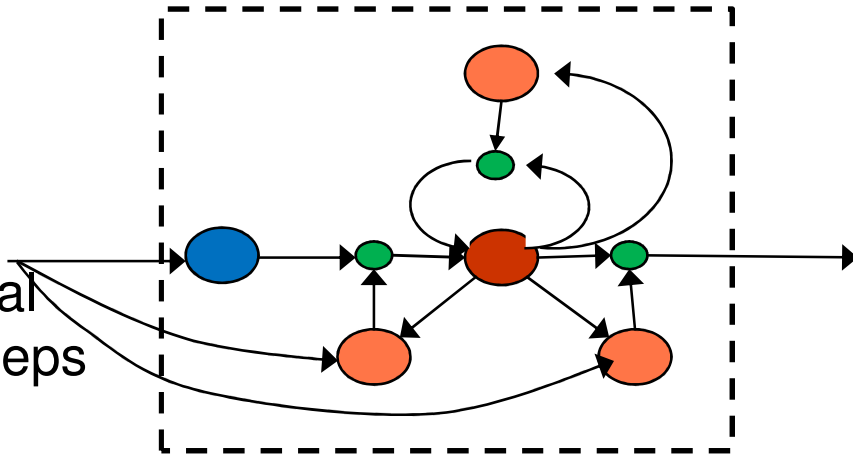
- The learning algorithm is a search algorithm in network space: with many inputs, the search space is large
  - which are the inputs affecting the output?

Antagonist goals

- weights must be large to have several attracting regions:
  - but large weights decrease generalization capability and/or saturate sigmoids and/or make gradient oscillate
- There must exist regions where $\left\|\frac{\partial z(t+1)}{\partial z(t)}\right\|$<1 holds tor ,robustness
  - But $\left\|\frac{\partial z(t+1)}{\partial z(t)}\right\|$<1 makes make gradient small
  - Small regions are difficult to reach

UNIVERSITÀ DI SIENA 1240

Long short-term memory: a solution



The idea

- keep $\left\|\frac{\partial z(t+1)}{\partial z(t)}\right\|$ equal to 1
  - this makes the error signal to remain through time steps

LSTM cell

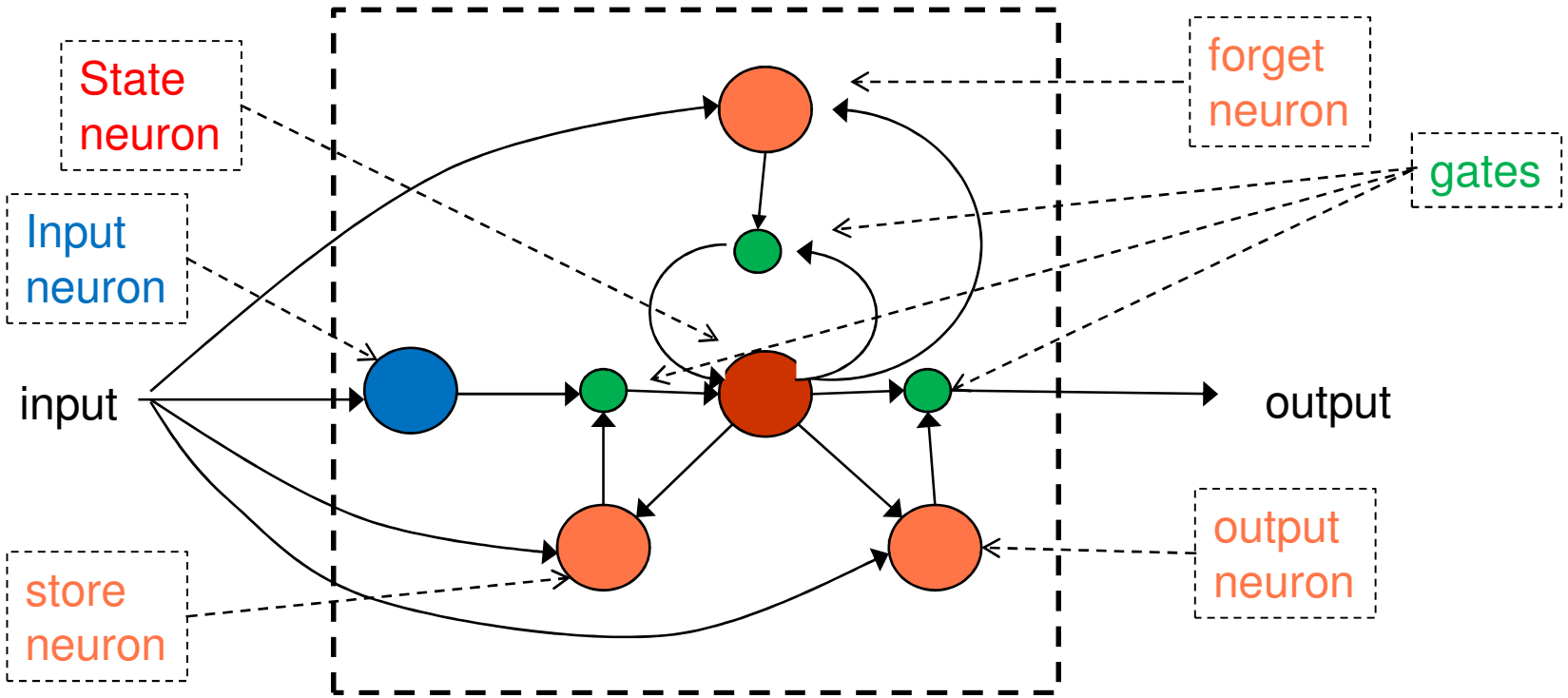
- a neuron to store the state
- an input neurons
- a store neuron and a gate to decide whether to store input
- a forget neuron and a gate to decide whether to reset the state
- An output neuron and gate to decide whether to output the state

UNIVERSITÀ DI SIENA 1240

Long short-term memory

LSTM cell

- neurons use sum and sigmoidal activation
- gates use just product

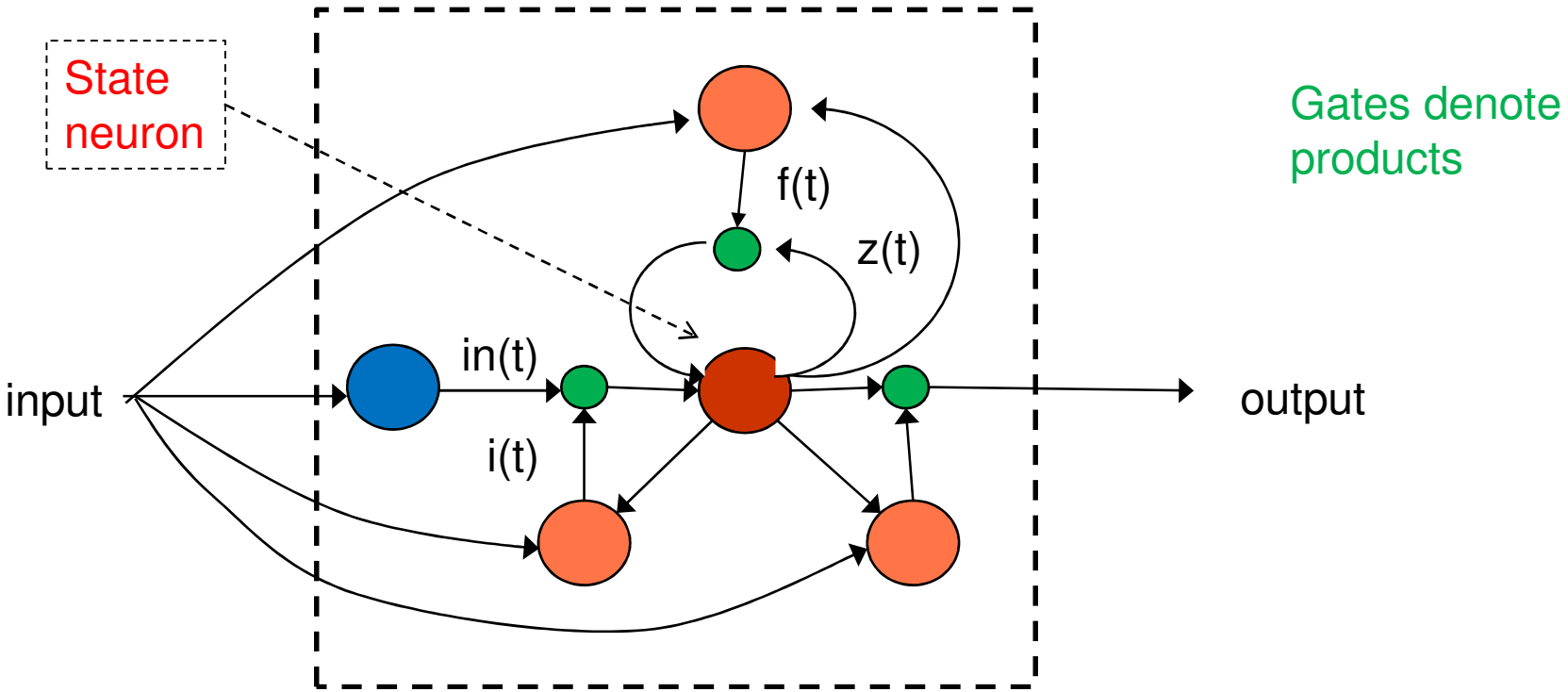


Long short-term memory

The state

- f(t) is 0 to forget, it is 1to remember
- i(t) is 1 to store input, it is 0 to skip

$z(t+1) = f(t)z(t)$+i(t) in(t)



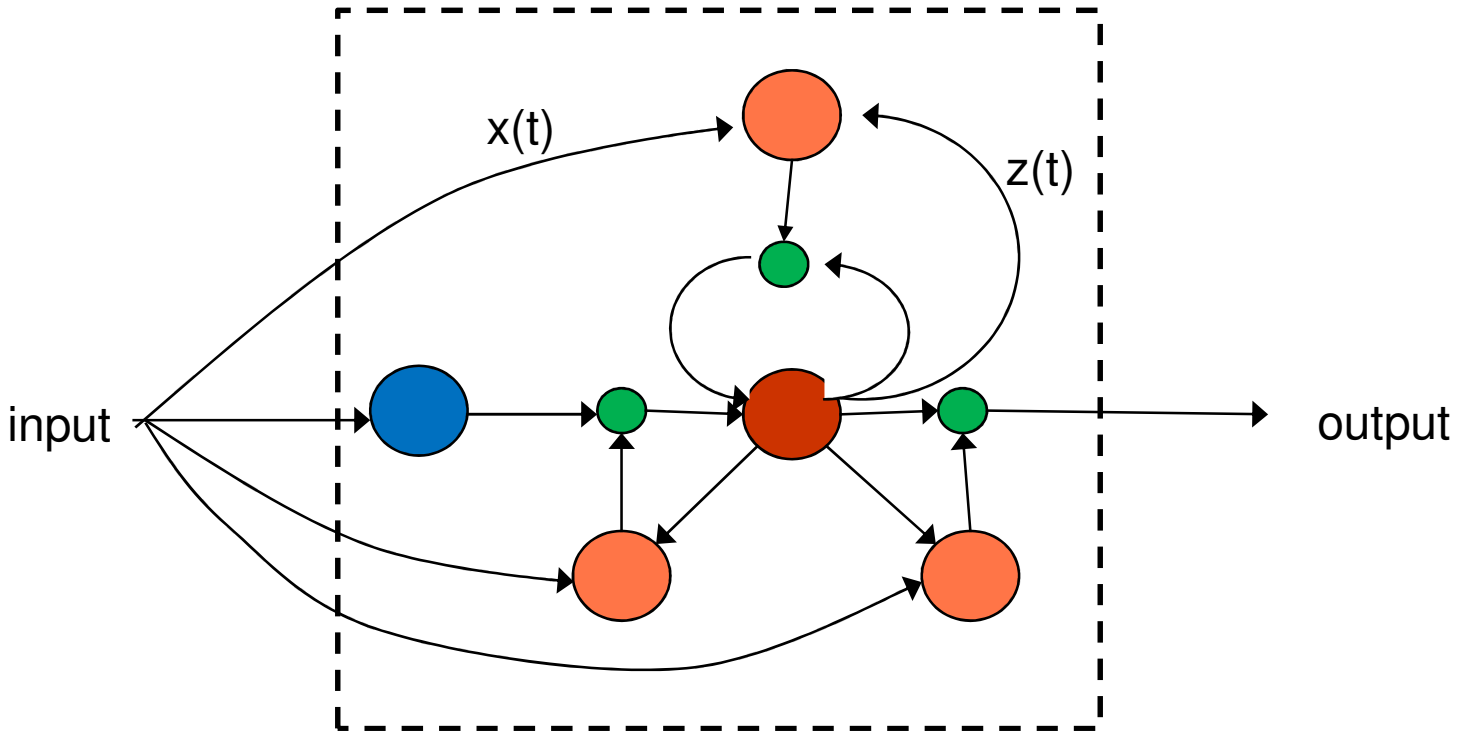
UNIVERSITÀ DI SIENA 1240

Long short-term memory

Store, forget and output neurons

- They are standard neurons with sigmoidal activation function and weights, e.g.

$$f(t) = \sigma(w_z z(t) + w_x x(t))$$

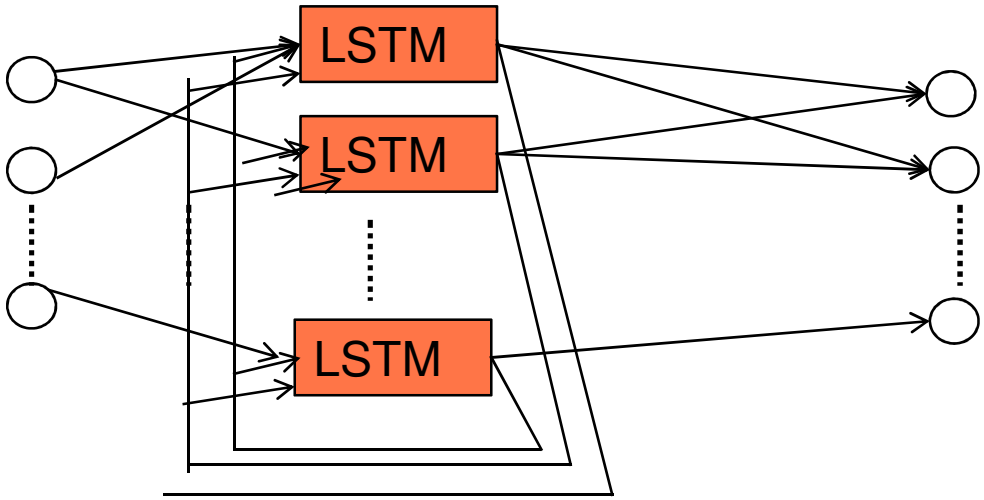


UNIVERSITÀ DI SIENA 1240

Long short-term memory

LSTM networks

- constructed connecting LSTM cell
- They can be mixed with standard neurons
- common architecture: a standard input layer, a hidden layer of LSTM cells, a standard output layer



UNIVERSITÀ DI SIENA 1240

Long short-term memory

LSTM

- can store input for hundreds/thousands of instances
- Successfully applied in several application domains, e.g. language translation

UNIVERSITÀ DI SIENA 1240

Long short-term memory: why do they work?

Explanation ver 1.0

- When the forget gate is 1, then we have $\left\|\frac{\partial z(t+1)}{\partial z(t)}\right\|=1$

$z(t+1) = f(t)z(t)$+i(t) in(t)

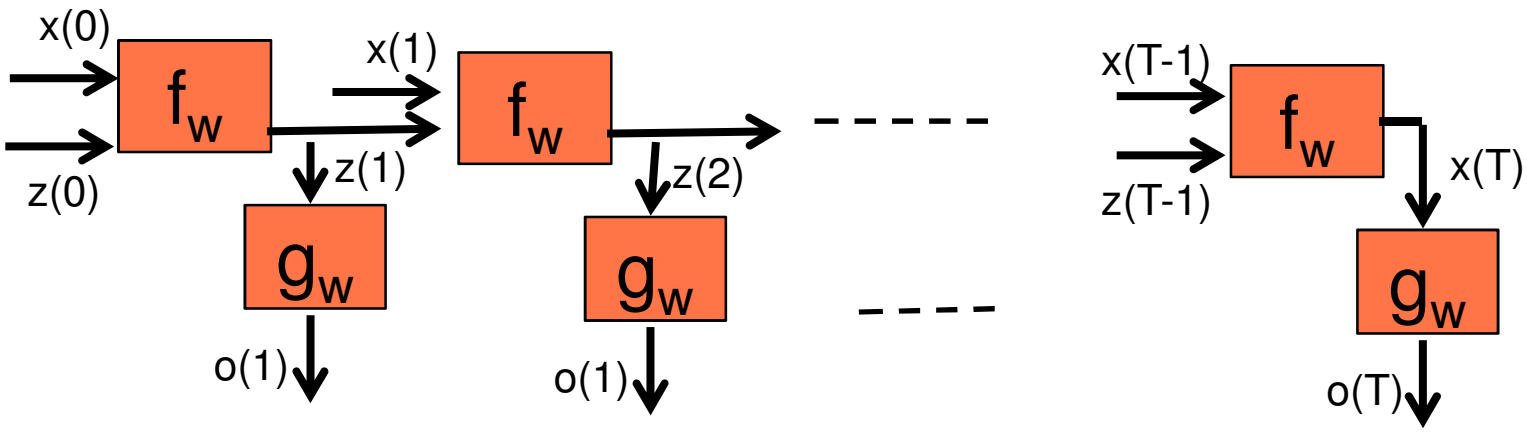
Another Explanation

- LSTM assumes that
  - the problem can be solved by storing/forgetting inputs/states
  - The decision whether to store/forget is simple (linearly separable, implementable by a single layer net)
- RNN does not make any assumption
- LSTM are better than RNN if the assumption is satisfied
  - The search space is reduced!

Generalization capability of RNNs

Intuitively

- VCD of RNNs can be studied by observing the unfolding network
- However, such an assumption does not consider the implications of the weight sharing
- Current results may suffer of this limitation



UNIVERSITÀ DI SIENA 1240

Generalization capability of RNNs

Neural networks with *p* parameters, T time steps, bounds for the order of growth of VCD(f_w)

- RNNs with piecewise polynomial activation function

  Upper bound: $O(Tp^2)$,

  Lower bound $o(p^2)$
- RNNs with, tanh, logsig, atan activation function

  Upper bound: $O(T^2p^4)$

  Lower bound $o(p^4)$

Deep neural networks (DNNs)

UNIVERSITÀ DI SIENA 1240

Deep neural networks

Definition ver 1.0

- Just networks with many layers

Intuitive explanation

- Each layer produces a more abstract representation of inputs simplifying difficult problems
- Difficult problems require the use of several layers
- Layers in animal brains are a justification of this fact

UNIVERSITÀ DI SIENA 1240

Current DNNs are much more complex than old DNNs

Current DNNs exploits a lot of peculiarities

- Different types of layers
- Weight sharing
  - Some neurons share the same weights
- Modularization
  - A sub-module of the network is applied on different subset of the inputs
- particular activation functions
  - Rectifier, drop out, max out, …

UNIVERSITÀ DI SIENA 1240

An example of DNNs: convolutional neural networks

Convolutional neural networks

- For image classification
- Originally used for handwritten digit recognition
- Currently, the best tools for object recognition sin images are based on evolutions of convolutional neural networks

The layers of a convolutional network

- Convolutional layers
- Pooling layers
- Fully connected layers

UNIVERSITÀ DI SIENA 1240

Convolutional neural networks: convolutional layer

Kernel filters in image processing

- Filter kernels are matrices, which can be applied to an image by convolution

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$ Edge Detection $$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$ Blur

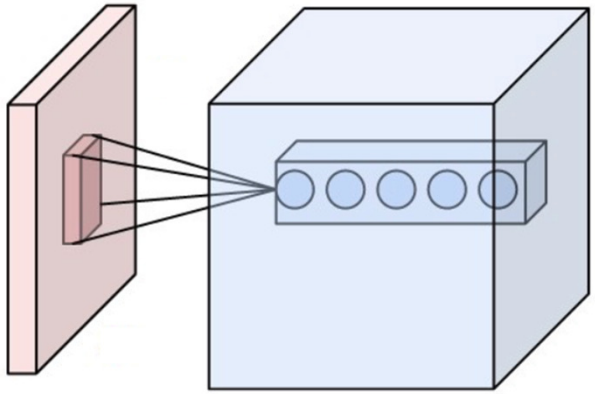
- By convolution with a 3x3 matrix M, the pixels of each 3x3 window are multiplied by M

- A convolutional layer is almost equivalent to the application of a kernel whose parameters are trained!!

UNIVERSITÀ DI SIENA 1240

Convolutional neural networks: convolutional layer

Convolutional layers

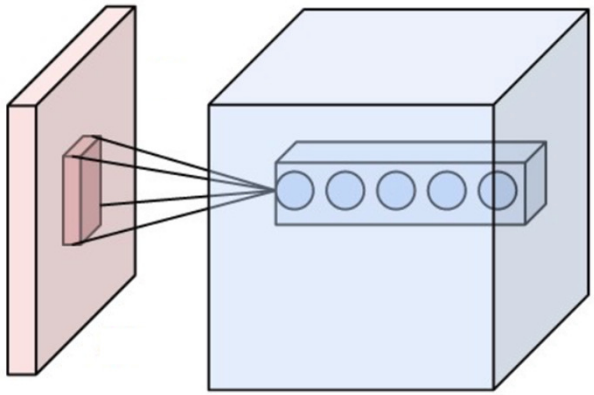


- A neuron of a convolutional layer implements a kernel
  - The neuron is connected to a window (receptive field) of the original image (f.i., a 3x3(x3) square)
  - The kernel matrix is defined by the weights of the connections
- The kernel is convoluted over the input
  - There is a neuron for each window receptive field
  - All the neurons share the same weight sets

UNIVERSITÀ DI SIENA 1240

Convolutional neural networks: convolutional layer

- A convolutional layer has the dimension of the input image (width and height)



- Intuitively, convolutional layers extract low level features from the input image

UNIVERSITÀ DI SIENA 1240

Convolutional neural networks: pooling layer

Pooling layers

- Each neuron of a pooling layer summarize the result of a small window of the image
  (f.i., a 2x2 receptive fields in the previous layer)
- Summarization can be by taking maximum, a random value, …
- Pooling layers decrease the size of the features (decrease width and height)

- Intuitively, pooling layers are used to simplify the problem by reducing the features to be considered

UNIVERSITÀ DI SIENA 1240

Convolutional neural networks: fully connected layer

Fully connected layers

- Common layers in which each neurons connected to all the neurons of the previous layer

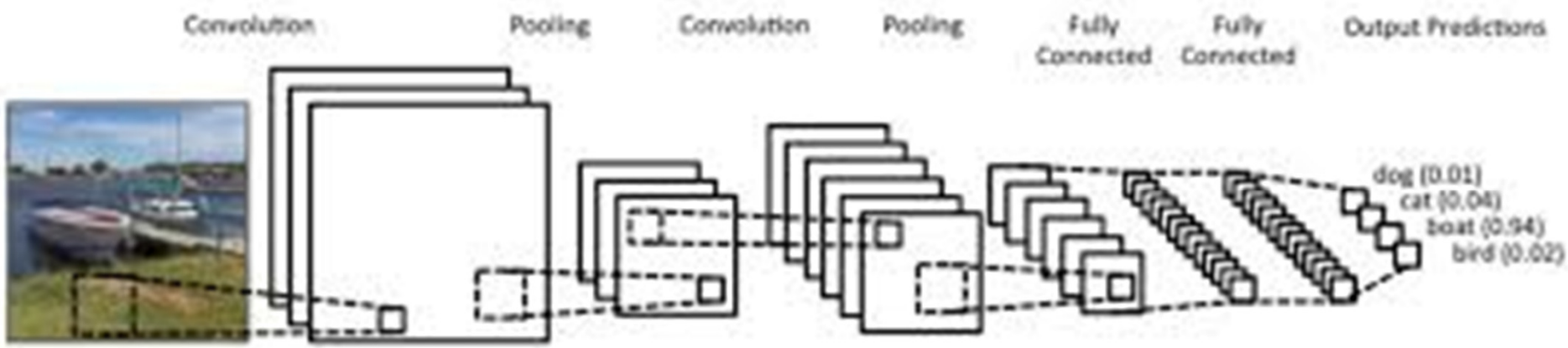
- Intuitively, fully connected layers allow to combine and reason about the extracted features in order to take the final decision

UNIVERSITÀ DI SIENA 1240

Convolutional neural networks

The architecture

- A convolution layer followed by a pooling layer
- One or several fully connected layer
- The pair (convolutional, pooling) may be repeated several times
- Several convolutional layers can be used in parallel



UNIVERSITÀ DI SIENA 1240

Approximation capability of DNNs

Approximation capability ver 1.0

- DNNs are universal approximators (with mild constraints on architecture)

- An example of the proof, whenthe network has more hiddens than outputs starting from the thirds layer
  - Given a target function t,
  - the first three layers of the DNN approximate t
  - remaining layers copy the ouptut of the third layer

UNIVERSITÀ DI SIENA 1240

The role of depth in DNNs

Approximation capability ver 2.0: the idea

- "(using the same of amount of resources), deep architectures can implement more complex functions than shallow networks"

A new tool was required to evaluate the complexity of the implemented classifiers

- The following complexity measures were not useful
  - Number of neurons
  - VC-dimension
  - Approximation capability

UNIVERSITÀ DI SIENA 1240

The underlining idea

- N: a neural network
- f: the function implemented by the network

- S_N: the set of the non-negative patterns, i.e.

  $S_N=\{x/\ f_N(x)\geq 0\}$

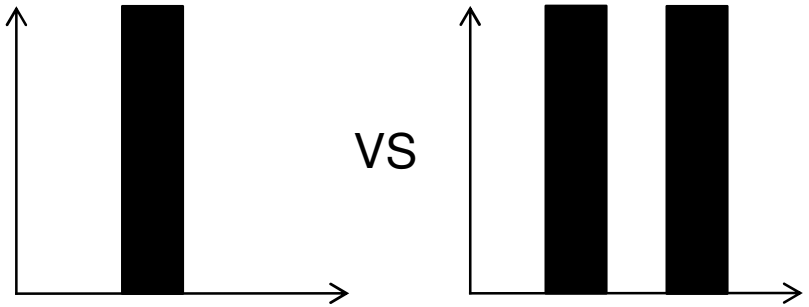
The idea

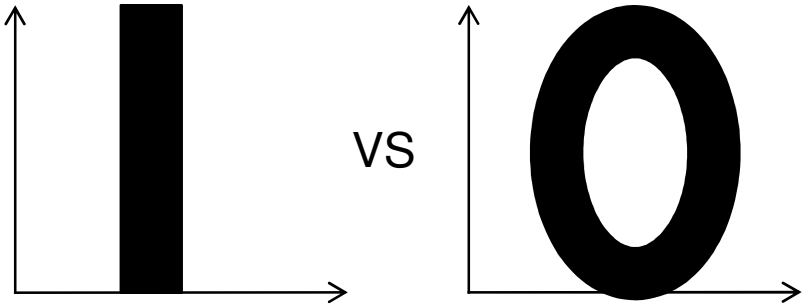
- To measure the topological complexity of the set S_N
- It is reasonable when N is used for classification purposes

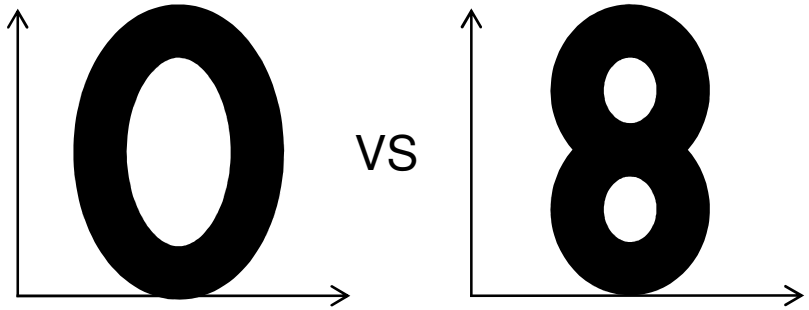
UNIVERSITÀ DI SIENA 1240

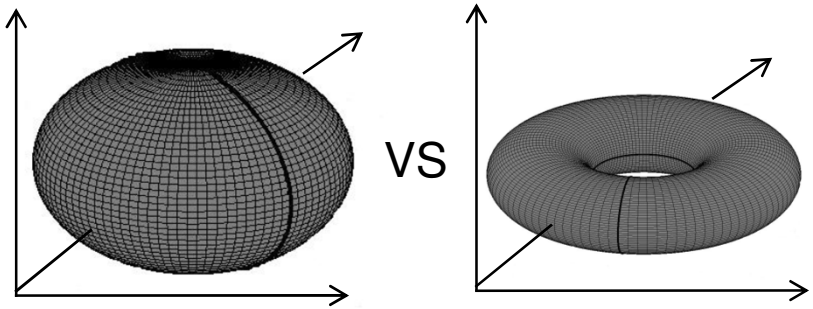
Topological complexity: an intuitive viewpoint

- Black regions represent the non-negative patterns, i.e,, S_N
- Can you say which set is more complex in each couple?









UNIVERSITÀ DI SIENA 1240

Betti numbers: a topological concept

Betti numbers

- used to distinguish spaces with different topological properties
- for any subset $S \subset R^n$, there exist n Betti numbers,

  $b_0(S), b_1(S), \ldots b_{n-1}(S)$

Formally

- $b_k(S)$: is rank of the k–th homology group of the space S.

Intuitively

- $b_0(S)$: is the number of connected components of the set S
- $b_k(S)$: counts the number of $(k+1)$–dimensional holes in S

UNIVERSITÀ DI SIENA 1240

The proposed measure of complexity

- In topology, the sum of the Betti numbers

  $B(S)=\Sigma_k\, b_k(S)$,

  is used to evaluate the complexity of the set S

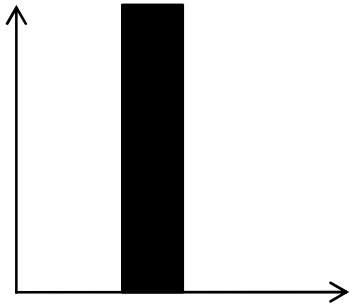
- We will measure the complexity of the function f_N implemented by a neural network N by

  $B(S_N)=\Sigma_k\, b_k(S_N)$,

  where of $S_N=\{x/\ f_N(x)\geq 0\}$.

UNIVERSITÀ DI SIENA 1240

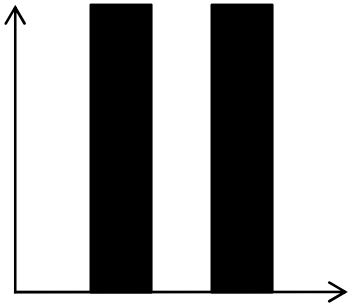
Betti numbers: examples



$b_0(S)=1\ b_1(S)=0$

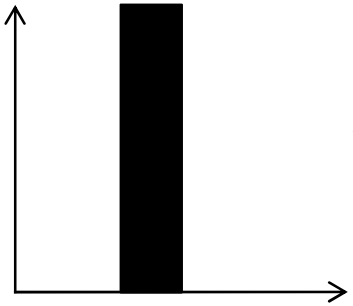
B(S)=1

VS



$b_0(S)=2\ b_1(S)=1$

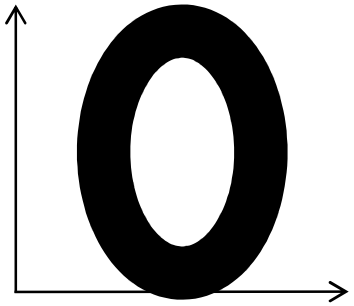
B(S)=3



$b_0(S)=1\ b_1(S)=0$

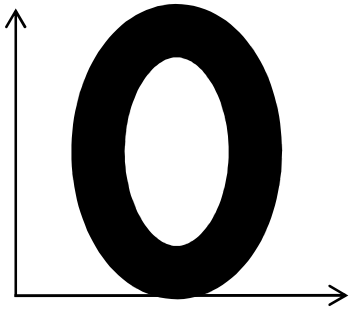
B(S)=1

VS



$b_0(S)=1\ b_1(S)=1$

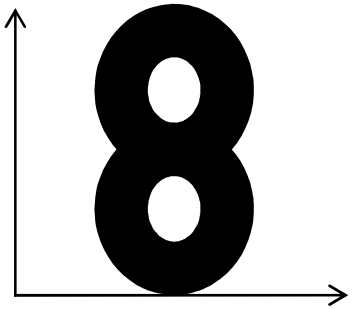
B(S)=2



$b_0(S)=1\ b_1(S)=1$

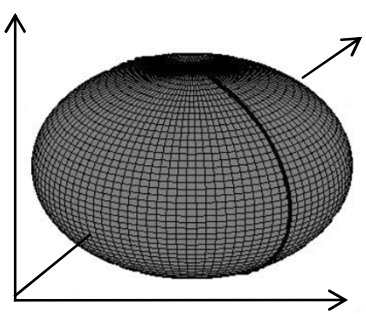
B(S)=2

VS



$b_0(S)=1\ b_1(S)=2$

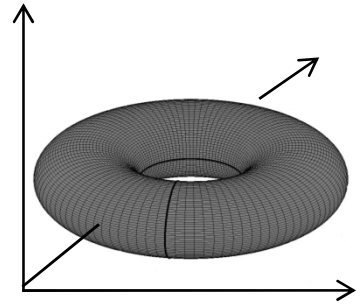
B(S)=3



$b_0(S)=1\ b_1(S)=1$
$b_2(S)=1$

B(S)=3

VS

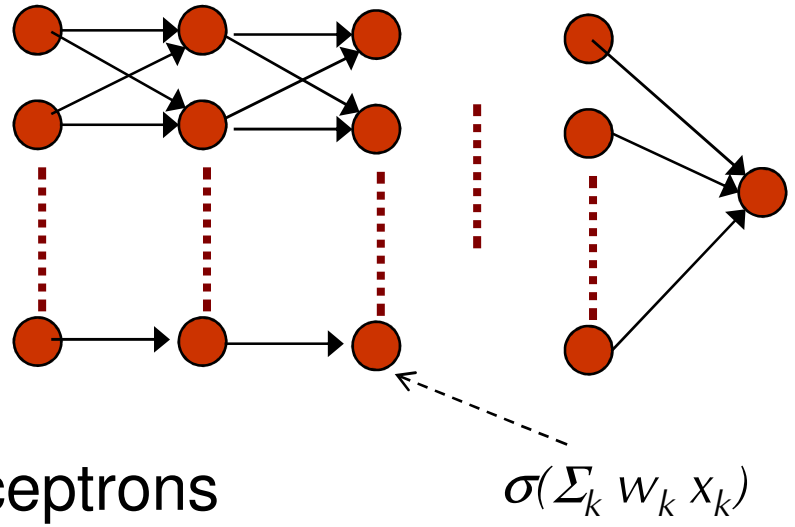


$b_0(S)=1\ b_1(S)=2$
$b_0(S)=1$

B(S)=4

UNIVERSITÀ DI SIENA 1240

The study



The considered networks

- feedforwad layered perceptrons
- with sigmoidal (ridge) activation functions in the hidden layers and linear in the output layer

Upper and lower bounds on $B(S_N)$ *varying*

- The number of hidden layers *l*
- The number of hidden units *h*
- The number of inputs *n*
- The activation functions: tanh, arctan, polynmial of degree *r*, generic sigmoids

UNIVERSITÀ DI SIENA 1240

The results

- l: number of hidden layers
- h: number of hidden units
- n: number of inputs
- r: degree of the polynomial

Layers	Activation	Bound on $B(S_N)$	Exponential	Polynomial
Upper bounds				
1	threshold	h^n	n	h
1	polynomial	$(2 + r)(1 + r)^{n-1}$	n	r
1	arctan	$(n + h)^{n+2}$	n	h
many	arctan	$2^{h(2h-1)} \bullet (nl + n)^{n+2h}$	n,h,l	
many	tanh	$2^{h(h-1)/2} \bullet (nl + n)^{n+h}$	n,h,l	
many	polynomial	$(2 + r^l)(1 + r\ l)^{n-1}$	n, l, h	r
Lower bounds				
1	sigmoid	$((h-1) /n)^n$	n	h
many	sigmoid	2^{l-1}	l, h	
many	polynomial	2^{l-1}	l, h	

UNIVERSITÀ DI SIENA 1240

Analysis of the results

W.r.t. the number of inputs n, the complexity

- always grows exponentially

UNIVERSITÀ DI SIENA 1240

The results

- l: number of hidden layers
- h: number of hidden units
- n: number of inputs
- r: degree of the polynomial

Layers	Activation	Bound on $B(S_N)$	Exponential	Polynomial
Upper bounds				
1	threshold	h^n	**n**	h
1	polynomial	$(2 + r)(1 + r)^{n-1}$	**n**	r
1	arctan	$(n + h)^{n+2}$	**n**	h
many	arctan	$2^{h(2h-1)} \bullet (nl + n)^{n+2h}$	**n**,h,l	
many	tanh	$2^{h(h-1)/2} \bullet (nl + n)^{n+h}$	**n**,h,l	
many	polynomial	$(2 + r^l)(1 + r\ l)^{n-1}$	**n**, l, h	r
Lower bounds				
1	sigmoid	$((h-1) /n)^n$	**n**	h
many	sigmoid	2^{l-1}	l, h	
many	polynomial	2^{l-1}	l, h	

UNIVERSITÀ DI SIENA 1240

Analysis of the results

w.r.t. the number of hidden neurons h, the complexity

- grows polynomially, for shallow networks
- grows exponentially, for deep networks

UNIVERSITÀ DI SIENA 1240

The results

- l: number of hidden layers
- h: number of hidden units
- n: number of inputs
- r: degree of the polynomial

Layers	Activation	Bound on $B(S_N)$	Exponential	Polynomial
Upper bounds				
1	threshold	h^n	n	**h**
1	polynomial	$(2 + r)(1 + r)^{n-1}$	n	r
1	arctan	$(n + h)^{n+2}$	n	**h**
many	arctan	$2^{h(2h-1)} \bullet (nl + n)^{n+2h}$	n,**h**,l	
many	tanh	$2^{h(h-1)/2} \bullet (nl + n)^{n+h}$	n,**h**,l	
many	polynomial	$(2 + r^l)(1 + r\ l)^{n-1}$	n, l, **h**	r
Lower bounds				
1	sigmoid	$((h-1) /n)^n$	n	**h**
many	sigmoid	2^{l-1}	l,**h**	
many	polynomial	2^{l-1}	l,**h**	

UNIVERSITÀ DI SIENA 1240

Analysis of the results

Summing up

- with the same amount of resources, deep networks can realize more complex classifiers than shallow networks!!

Remarks

- *This does mean that all the functions can be better approximated by deep networks!!*
- *Only some functions with particular symmetries will have benefict from being approximated by deppe networks*

UNIVERSITÀ DI SIENA 1240

An intuitive explanation of the advantage of deep architectures

Notice that

- A layered network implements a function

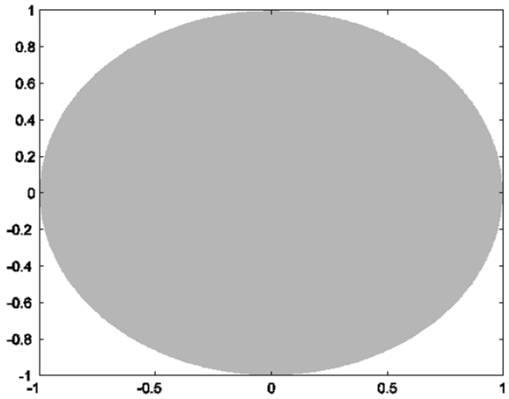
  $f_N = g_1 \circ g_2 \,..\, \circ\, g_l \circ t$

  - g_k is the function implemented by layer k
  - $\circ$ is the function composition operator
- If $f_N = g_1 \circ t$, then f_N behaves as t on all the regions $A_{1,..}, A_s$ where $g_1(A_k) = R^n$
- With several layers, the number of regions A_k such that $g_l(\ldots g_1(A_k)) = R^n$ can grow exponentially

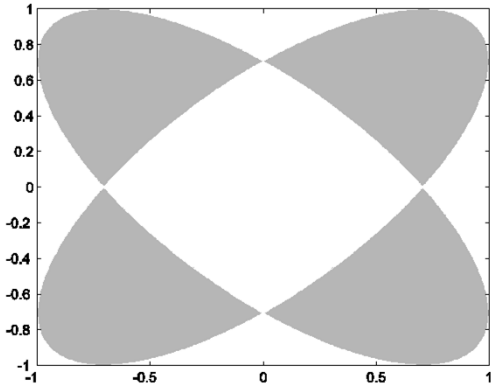
Deep networks can replicate more easily the same behavior on different regions of the input domain

UNIVERSITÀ DI SIENA 1240

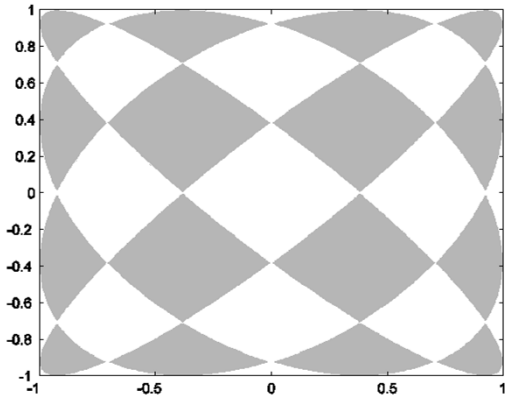
An example of the advantages of deep networks



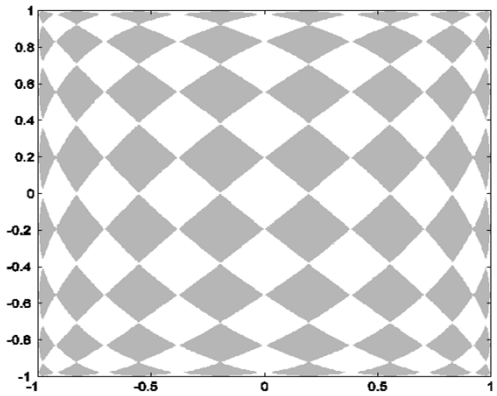
t



g ∘ t



g ∘ g ∘ t



g ∘ g ∘ g ∘ t

$$g(x) = 1 - \|x\|^2$$

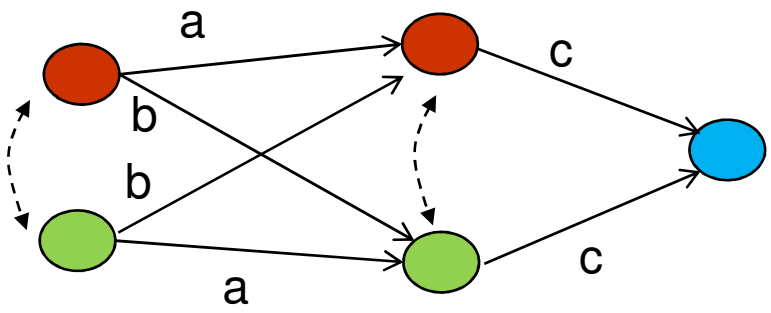
$$t(x) = [1 - x_1^2, 1 - x_2^2]$$

UNIVERSITÀ DI SIENA 1240

The role of convolutional layers

Symmetry networks (Shaw-taylor)

- Networks for which there exist a set of group of automorphisms which maps an network in an equivalent network where neurons are permutated
- It can be proved that the output of symmetry network is a invariant under permutations of its inputs



UNIVERSITÀ DI SIENA 1240

The role of convolutional layers: invariance

Convolutional networks

- Convolutional layers are automorphic w.r.t. translations of the neurons in inputs on those on feature maps
  - But this is formally true only if infinitive or circular inputs are considered
  - Convolutional layers mixed with pooling layers have also this property
- Full connected layers are not invariant under the above automorphisms

UNIVERSITÀ DI SIENA 1240

The role of convolutional layers: invariance

Thus

- Convolutional networks are not symmetric networks and are not invariant under input translations
- They could be invariant using symmetric full connected layers, e.g., a one layer network sharing the weights

UNIVERSITÀ DI SIENA 1240

The role of convolutional layers: approximation capability

A question: Are network composed by convolutional layers universal approximators?

- That consider a network composed of convolutional layers and consider a feature among those in feature maps: can be the map from inputs to such a feature approximate any function?

The answer

- Yes, with the following remarks
  - The feature map can be only a function of its receptive field
  - A sufficient number of feature maps in previous layers must be used
    (feature maps play the role of hidden nodes in feedforward neural networks)

UNIVERSITÀ DI SIENA 1240

The role of convolutional layers: approximation capability

Notice

- From approximation viewpoint, the single features of a feature map are not restricted
  - In theory, a single feature is sufficient to compute any function of the input image!
  - In practice, in order to compute a complex function of the input maps would require a large number of feature maps
    - Better tp have a single large full connected network!

UNIVERSITÀ DI SIENA 1240

Bibliography

- Auer, P., Herbster, M., & Warmuth, M. K. (1996). Exponentially many local minima for single neurons. In *Advances in neural information processing systems* (pp. 316-322).
- Baldi, P., & Hornik, K. (1989). Neural networks and principal component analysis: Learning from examples without local minima. *Neural networks*, *2*(1), 53-58.
- Bartlett, P. L. (1998). The sample complexity of pattern classification with neural networks: the size of the weights is more important than the size of the network. *IEEE transactions on Information Theory*, *44*(2), 525-536.

UNIVERSITÀ DI SIENA 1240

Bibliography

- Bengio, Y., Simard, P., & Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, *5*(2), 157-166.
- Bianchini, M., Gori, M., & Maggini, M. (1994). On the problem of local minima in recurrent neural networks. *IEEE Transactions or Neural Networks*, *5*(2), 167-177.
- Bianchini, M., & Gori, M. (1996). Optimal learning in artificial neural networks: A review of theoretical results. *Neurocomputing*, *13*(2-4), 313-346.
- Bianchini, M., & Scarselli, F. (2014). On the complexity of neural network classifiers: A comparison between shallow and deep architectures. *IEEE transactions on neural networks and learning systems*, *25*(8), 1553-1565.
- Barron, A. R. (1993). Universal approximation bounds for superpositions of a sigmoidal function. *IEEE Transactions on Information theory*, *39*(3), 930-945.

UNIVERSITÀ DI SIENA 1240

Bibliography

- Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, *2*(4), 303-314.
- Hammer, B. (2000). On the approximation capability of recurrent neural networks. *Neurocomputing*, *31*(1-4), 107-123.
- Hornik, K., Stinchcombe, M., & White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural networks*, *2*(5), 359-366.
- Funahashi, K. I. (1989). On the approximate realization of continuous mappings by neural networks. *Neural networks*, *2*(3) 183-192.

UNIVERSITÀ DI SIENA 1240

Bibliography

- Gori, Marco, and Alberto Tesi. "On the problem of local minima in backpropagation." *IEEE Transactions on Pattern Analysis and Machine Intelligence* 14.1 (1992): 76-86.
- Gori, M., & Scarselli, F. (1998). Are multilayer perceptrons adequate for pattern recognition and verification?. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *20*(11), 1121-1132.
- Huang, G. B., Zhu, Q. Y., & Siew, C. K. (2006). Extreme learning machine: theory and applications. *Neurocomputing*, *70*(1-3), 489-501.
- Judd, J. S. (1990). *Neural network design and the complexity of learning*. MIT press.

UNIVERSITÀ DI SIENA 1240

Bibliography

- Lawrence, S., Tsoi, A. C., & Giles, C. L. (1996, June). Local minima and generalization. In *Neural Networks, 1996., IEEE International Conference on* (Vol. 1, pp. 371-376). IEEE.
- Hochreiter, S., Bengio, Y., Frasconi, P., & Schmidhuber, J. (2001). Gradient flow in recurrent nets: the difficulty of learning long-term dependencies.
- Scarselli, F., & Tsoi, A. C. (1998). Universal approximation using feedforward neural networks: A survey of some existing methods, and some new results. *Neural networks*, *11*(1), 15-37.
- Shawe-Taylor, J. (1993). Symmetries and discriminability in feedforward network architectures. *IEEE Transactions on Neural Networks*, *4*(5), 816-826.
- Yu, X. H., & Chen, G. A. (1995). On the local minima free condition of backpropagation learning. *IEEE Transactions on Neural Networks*, *6*(5), 1300-1303.
- Vapnik, V. N. (1999). An overview of statistical learning theory. *IEEE transactions on neural networks*, *10*(5), 988-999.

UNIVERSITÀ
DI SIENA
1240

Thank you for your attention!