Theoretical fundamentals of artificial neural networks

Franco Scarselli

▶ DEPARTMENT OF INFORMATION ENGINEERING AND MATHEMATICS

1

OUTLINE

- About the course (dates, exams, credits…)
- (Very) short introduction to perceptron neural networks
- Aproximation capability
- Learning capability
- Generalization capability
- Autoencoders
- Recurrent neural networks
- Deep networks
- Graph neural network

2

The course

UNIVERSITÀ DI SIENA 1240

- Four day course, every Tuesday
- On September 6th,7th, 8th, 9th,10-13, on 20th 9-13
- I will upload the material after each lecture on my home page, https://www3.diism.unisi.it/~franco/

For students requiring credits

UNIVERSITÀ DI SIENA 1240

- The last day will include an exam
  - The exam will be written
  - I will give you/send you by email with a set closed questions you have to answer
  - You will return the answers (by email)
  - After the course, I will mark them
- Just after the end of the exam, I will show you the solution and we will sum up the topics tought during the course
- The goal is that of assessing the students but also that of fisxing and summing up our work

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!

5

UNIVERSITÀ DI SIENA 1240

Machine learning

Can you describe how you recognize an apple in images?

- It just looks like a red circle

Red? Circle?

6

UNIVERSITÀ DI SIENA 1240

Machine learning

Red circle?

7

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

8

UNIVERSITÀ DI SIENA 1240

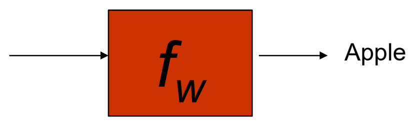
Machine learning: models

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



9

UNIVERSITÀ DI SIENA 1240

(Supervised) 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

10

UNIVERSITÀ DI SIENA 1240

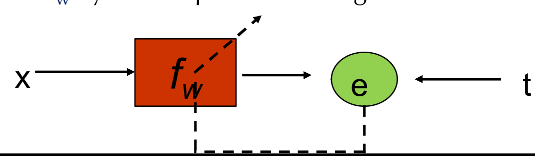
Machine learning: supervised training

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



11

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

12

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, …
- ….

13

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

14

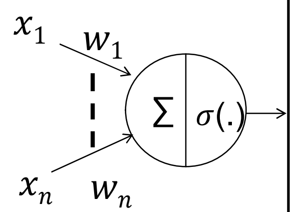
UNIVERSITÀ DI SIENA 1240

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

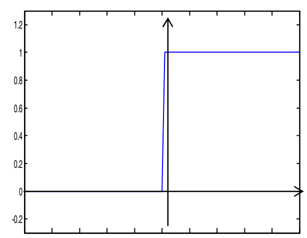
15

UNIVERSITÀ DI SIENA 1240

Types of neurons

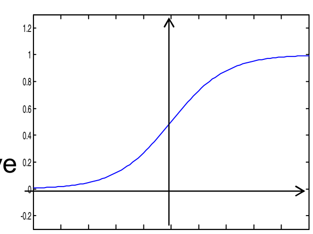
Step

- 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

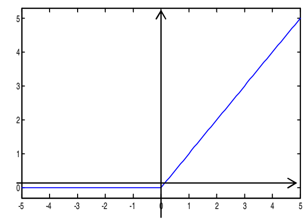


16

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

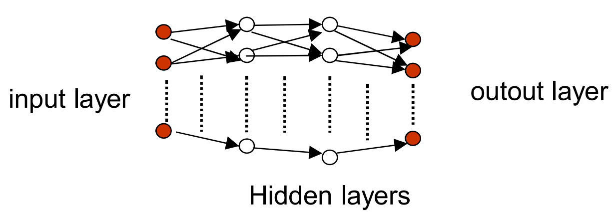


17

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



18

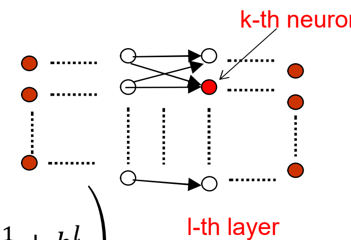
Multilayer feedforward neural networks (FNNs)

UNIVERSITÀ DI SIENA 1240

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

19

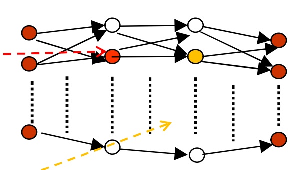
Multilayer feedforward neural networks (FNNs)

UNIVERSITÀ DI SIENA 1240

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

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

20

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

Approximation capability

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

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, ….

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)

$$\Sigma^3{}_\sigma = \{f(x) | f(x) = W^2\, \sigma(W^1 x + b_1) + b_2\}$$

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

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

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

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

25

Approximation capability: sketch of the proof

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

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

Two step proof

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

26

Sketch of the proof: single input functions

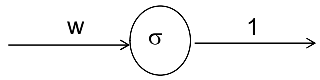
The main idea: four step proof

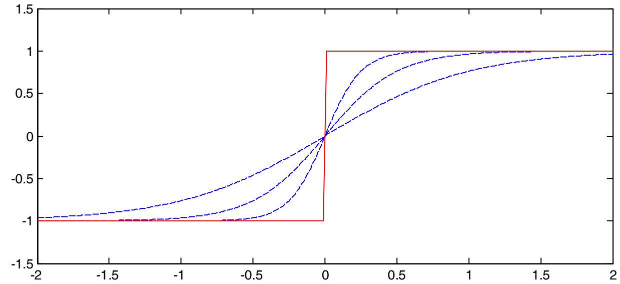
1. A single neuron implements sigmoid function
2. A sigmoid can approximate a step function
3. Many step functions can approximate a staircase function
4. Staircase function can approximate any continuous function

27

Sketch of the proof: single input functions

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



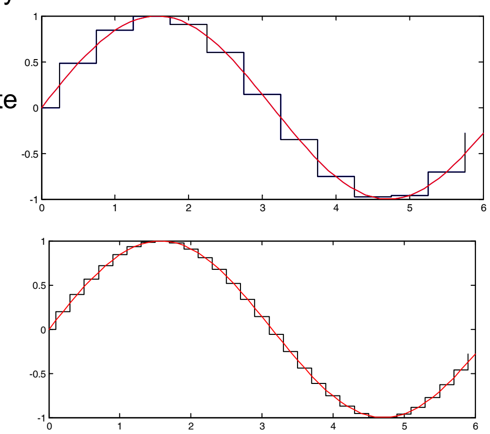


28

Sketch of the proof: single input functions

UNIVERSITÀ DI SIENA 1240

- 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



29

Sketch of the proof: another approach

UNIVERSITÀ DI SIENA 1240

The main idea: two step proof

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

30

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

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

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

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

31

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 R^{T}(x-x0), which is smaller than O($(x\text{-}x0)^{T}$)
  - $\sigma(x) = \sigma(x-x0) + \frac{(x-x0)\sigma'(x-x\ )}{1!} + \frac{(x-x0)^{2}\sigma'(x-x0)}{2!} + \cdots.\frac{(x-x0)^{T-1}\sigma^{(T-1)}(x-x0)}{(T-1)!} + RT(x-x0)$

- Thus, a neuron with an analytic activation function looks like a polynomial…
- … then, an FNN looks like a combination of polynomials

32

Sketch of the proof: approximating polynomials

Go back to the original goal …

- for any polynomial $p(x)=c_0+c_1x+c_2x^{2}+\ldots+c_rx^r$, 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 $\lim_{\alpha \to 0} p_\alpha(x) = p(x)$, where**

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

r neurons

This is a FNN

Second layer weight

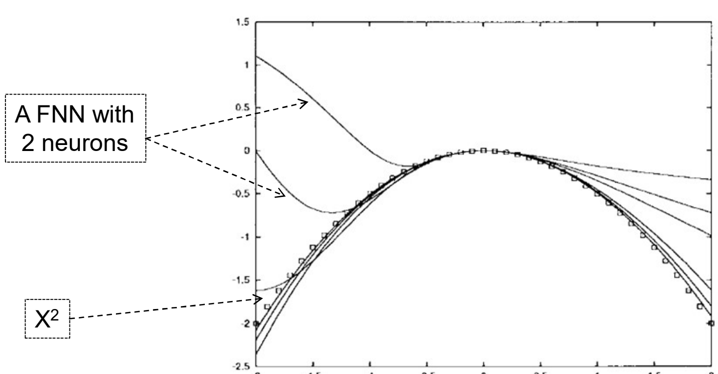
First layer weight

Hidden bias

33

Sketch of the proof: approximating polynomials

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



34

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 a simple case: when the target function is a ridge function

35

UNIVERSITÀ DI SIENA 1240

Ridge functions

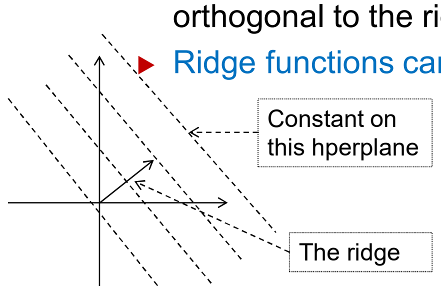
- A ridge function g:R^n->R, can be written as

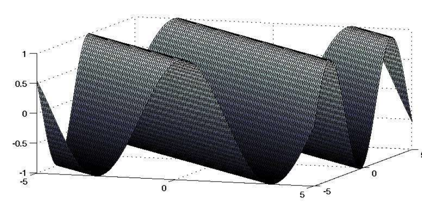
$$g(x) = h(wx)$$

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





36

Sketch of the proof: functions with several inputs

How can we extend our results to the general case of 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

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

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

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

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

Solution 1: the Radon transform

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

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

This is a ridge function

- The result is a sum of ridge functions!!

39

Solution 2: combination of polynomials

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

- Restrict your attention to polynomials and prove that

Theorem For any target polynomial t:R^n->R, there exist ridge polynomial g_1,..,g_k, such that

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

40

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

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

41

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 of n variables and degree r!!

42

UNIVERSITÀ DI SIENA 1240

Other solutions

- Other solutions exist, e.g. based on Fourier transform, …

43

UNIVERSITÀ DI SIENA 1240

Possible extensions

Universal approximation

- Activation functions
  - tanh, arctan, ReLU,   step, any analytic function, any step function
  - Not enough: polynomials
    - Used alone, they can implement only other polynomials
- Error norm
  - Sup, L1, L2, … sobolev,…
- Architecture
  - Any number of inputs and outputs
  - At least one hidden layer is required!

44

Back to practice

- In general, the FNNs described by the mentioned results on approximation theory are not encountered in practical experiments
  - learning algorithm often produce FNNs without an intuitive explanation

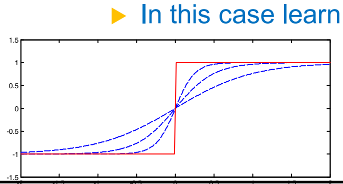
- But in particular cases, the consequences of approximation theory are evident also in practice.

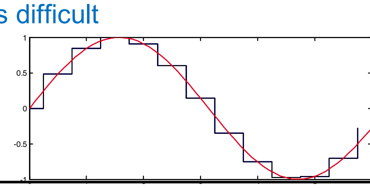
45

Back to practice: often seen in experiments with three layer networks

About weight dimension approximating staircase

- 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





46

Back to practice: often seen in experiments with three layer networks

About weight dimension, approximating polynomials

- With sigmoids 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!!

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

47

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
- ReLU does not suffers from the problem with polynomials: this a good new for the modern architectures that tend to use ReLUs!!
  - Notice however that if the polynomial to be approximated is very high order then also ReLUs are not a good solution may still suffer from large weights !!!
- Modern architectures and learning algorithms use a number of tricks that alleviate/control/change the effect of weight dimension so that the mentioned problem may be change consequently: eg weigh decay, batch normalization, …

48

Back to practice: ridge direction

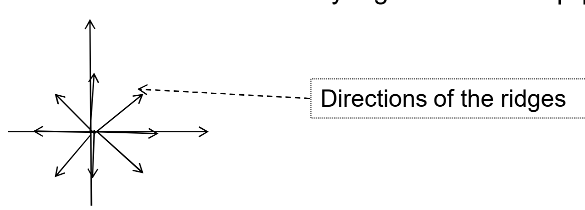
What about the directions (not dimension) of ridges $w_1^1, \ldots, w_n^1$?

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

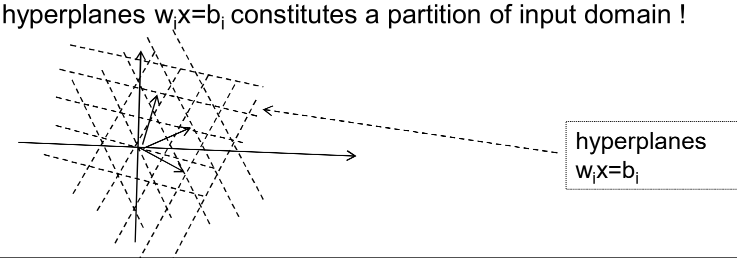
49

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 !



50

Back to practice: ridge direction

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

51

A constructive algorithm from theory

A simple algorithm (old work by me and Tsoi)

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

52

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 matrix form
- $T = [t_1,..t_k]$ Set of targets in matrix form

The network output Y

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

53

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

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

$=\|vec(T - W^2H + b^2\mathbf{1})\|^2$

$=\|vec(T) - vec(W^2H + b^2\mathbf{1})\|^2$

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

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

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

54

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
- …

55

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!
- We have to discuss about resource usage, not only about universal approximation!

56

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/parameters ….

57

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 quality changes with the number of neurons

58

UNIVERSITÀ DI SIENA 1240

Going beyond universal approximation

Barron proved that

- Let *t* be 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

59

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 adds a neuron at each step

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

60

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 constant

Thus,

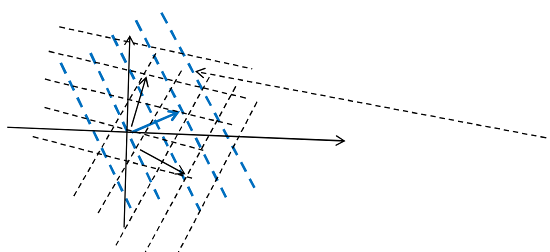
- There are target functions for which approximation by ELM, polynomials, … is much worse than approximation by FNN!
  - It is O(1/k) in FNN wrt $O(1/\sqrt[n]{k})$
  - When the input space is large, the difference is huge

61

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

62

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?

63

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

64

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

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

65

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)

66

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!!!
  - Back on this later
  - this is about deep learning !!!!

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 suggestion is the most obvious, the best architecture depends on the problem ☹

UNIVERSITÀ DI SIENA 1240

Learning capability

69

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?

70

Learning in neural networks

Optimization of error function ver 1.0

- by gradient descent
- update the parameters according to

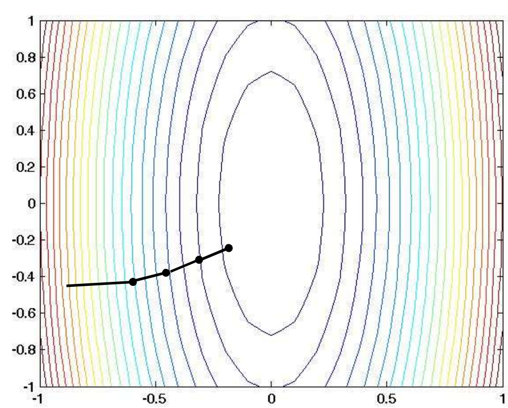
$$w(t+1) = w(t) - \lambda \frac{\partial ew}{\partial w}$$

until a desired minimal error is obtained

71

Learning in neural networks

Gradient descent



72

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: adam, resilient backpropagation, conjugate gradient, Newton, …
- Not matter of this course

73

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

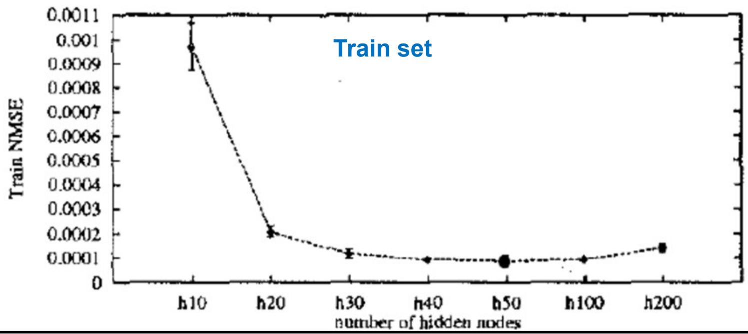
- When k2>=k1, the best minina has cost 0!

74

An old experiment

Results

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

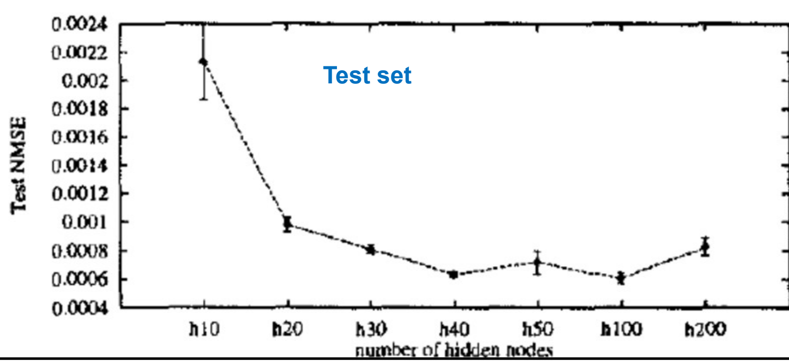


75

An old experiment

Results

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



76

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

77

UNIVERSITÀ DI SIENA 1240

Practical question: learning capability

Why does training fail

Answer (ver 1.0): local minima

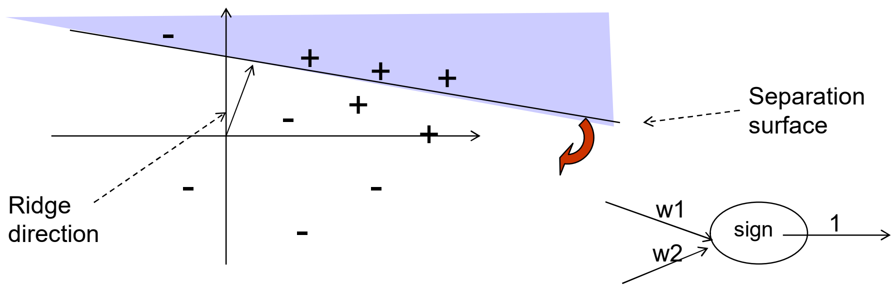
- learning capability depends on the presence/absence of local minima in error function
- Theoretical results on this are few and incomplete … let us review some of them for three layer networks
  - Later we will discuss about deep architectures

78

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

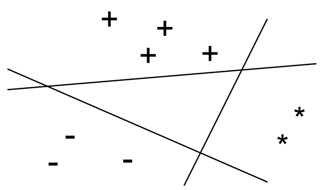


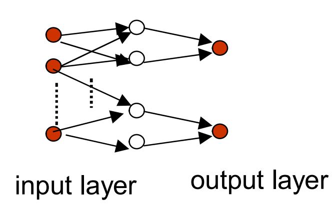
79

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





80

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

81

So many neurons as patterns

Proof… the idea

Remember that

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

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

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

- 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)!

82

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

83

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

84

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 k

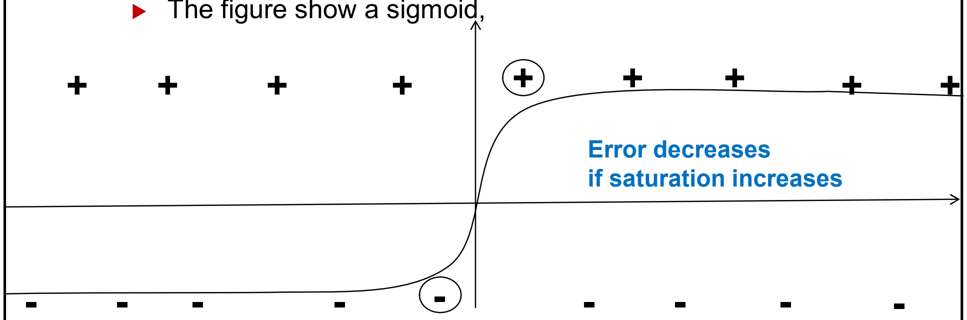
85

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
- The figure show a sigmoid,

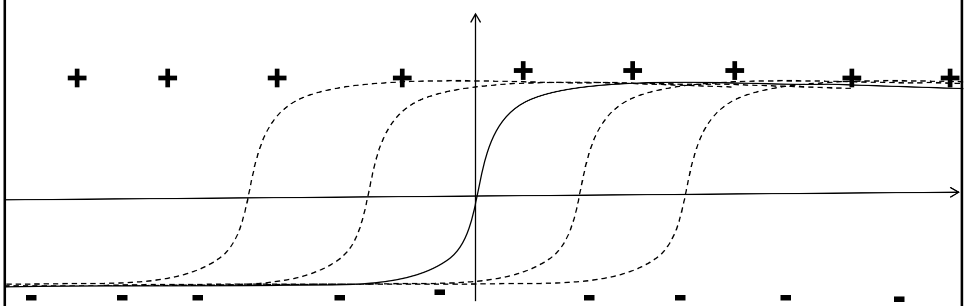


86

Presence of local minima

A sketch of the proof for a single input perceptron

- There are many minima



87

Presence of local minina

Extended XOR (Bianchini)

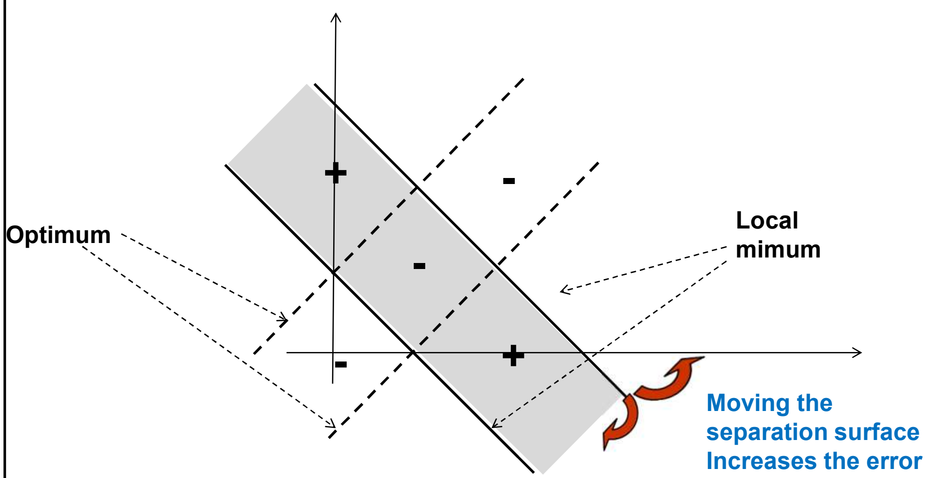
- Network
  - One hidden layer with 2 neurons
  - Sigmonidal output activation function
- Patterns
  - 5 patterns

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

88

Presence of local minina

Extended XOR (Bianchini 96)



89

Practical question: learning capability

Why does training fail?

- Does really training fail for local minima?

We may have disregarded the role of

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

90

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

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

91

UNIVERSITÀ DI SIENA 1240

Attracting regions

Networks with only ReLU activations

- Some attracting regions may due to the approximation of discontinuous or unbounded functions (recall approximation by step functions)
- Other may due to soft max loss
- ….

Networks with sigmoidal activation

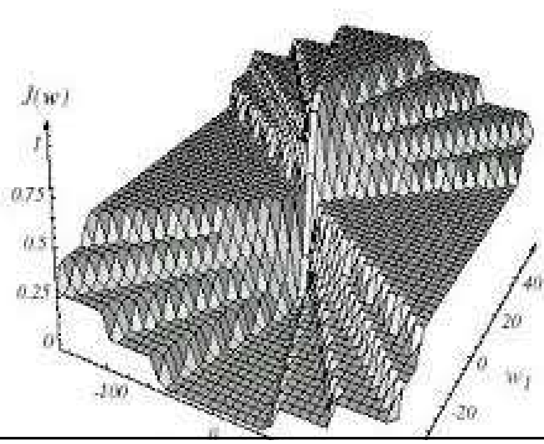
- They may have a large number of attracting regions, due to the saturation of the sigmoid when the weights are large
- If output neurons use sigmoids and the error function is MSE
  - a perfect learning of the train set requires for infinitive weights!

92

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 $a>0$
    - ….

Regions with deep valleys

Condition number for a matrix A

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

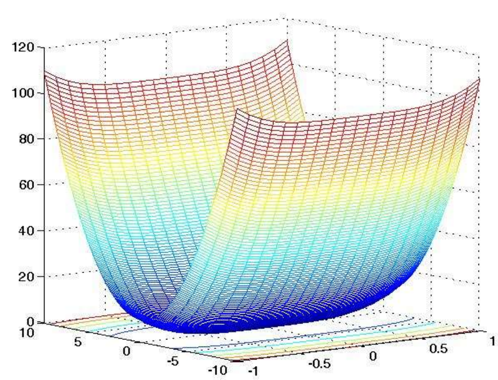
An error function e_w having Hessian matrix $\nabla^2 e_w$ with a large condition number

- 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!!

95

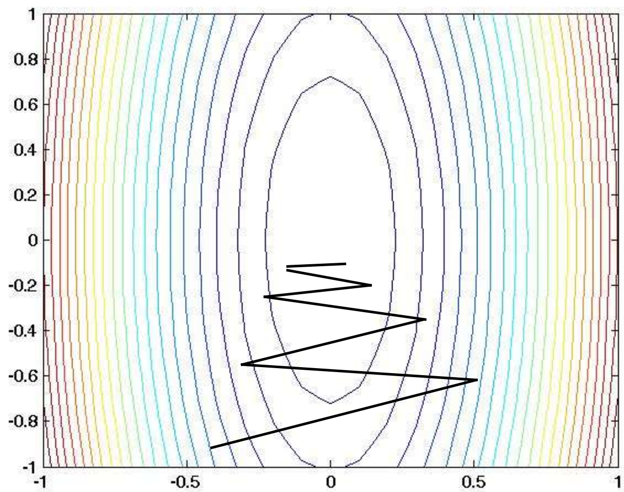
Regions with deep valleys



96

UNIVERSITÀ DI SIENA 1240

Regions with deep valleys



97

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

98

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, the converge rate for error ε is

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

Regions with deep valleys: a theoretical viewpoint

- In a general case, the function may not be strongly convex (some eigenvalue is 0)
  - 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 $O\left(\sqrt{L/\varepsilon}\right)$

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

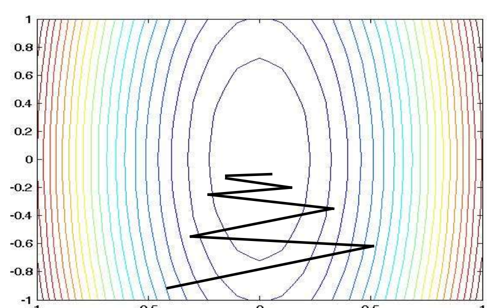
$$O\left(\sqrt{L/\varepsilon}\right)$$

steps

101

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!!

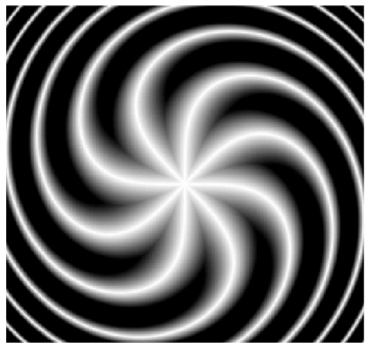


102

UNIVERSITÀ DI SIENA 1240

Intuitive message 2

- Learning may bed difficult even if the error function has only a minimum!!



103

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 been proven that the following Judd

- Loading problem is NP-complete!!

The theorem holds for

- binary functions
- network with threshold activation functions

104

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 capability

In practice

- even if a model can approximate a target function, such a model may not be easy to learn
- learning capability depends
  - on the problem (train set)
  - the adopted model
- A general rule works in practice!!!
  - the smallest the data, the simplest the learning
  - the larger the model, the simplest the learning

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

107

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

108