Neural networks for derivative-free optimization

Milagros Garcia
mgarcia@cerfacs.fr

Joint work with S. JAN and M. MASMOUDI

Plan

- Motivations
- Learning with neural networks
  - numerical examples
- Optimization using a neural network
  - numerical examples
- Conclusion remarks

Plan

- Motivations
- Learning with neural networks
  - numerical examples
- Optimization using a neural network
  - numerical examples
- Conclusion remarks

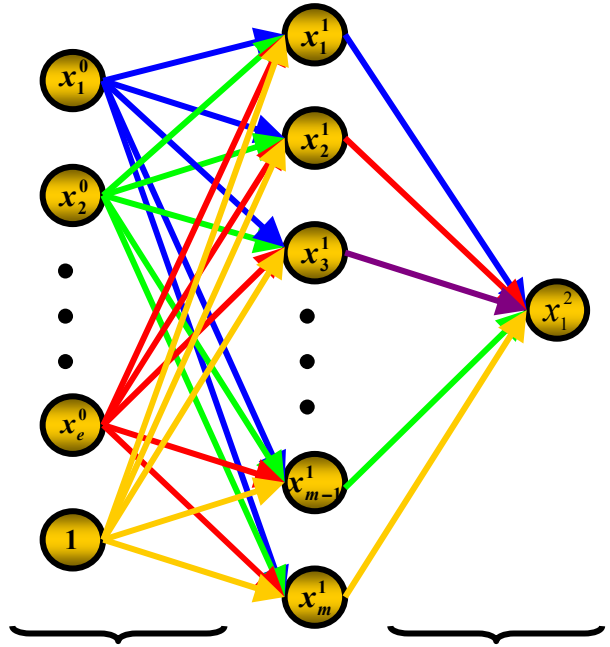
Motivations

- Build a model that represents well a phenomenon which is expensive to simulate or to experiment
- Find the best set of parameters for a criteria which is expensive to compute and / or for which we have no derivatives
- Find the global minimum of functions with local minima

Plan

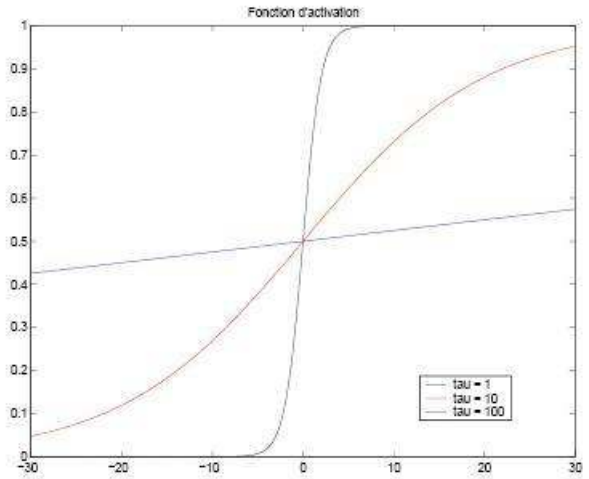
- Motivations
- Learning with neural networks
  - numerical examples
- Optimization using a neural network
  - numerical examples
- Conclusion remarks

Structure of a neural network



Building of basis functions

Search for the coefficients



$$f_0(x) = \frac{1}{1 + e^{\frac{x}{\tau}}}$$

Building of the basis functions: $x_j^1 = f_0\left(\sum_{i=1}^{e+1} W_{ij}^0 x_i^0\right)$

Approximation in this basis: $x_l^2 = \sum_{j=1}^{m} W_{jl}^1 x_j^1$

We build the basis functions adapted to the problem!

…an optimization problem

Notations

Given X^l input signal and their associated target Y^l, l = 1,….nbc,
we look for the nbW weights W_{ij} such that

$$\min_{W \in R^{nbW}} J(W) \quad \text{where} \quad J(W) = \frac{1}{2}\left\| E(W) \right\|^2$$

$$E : \mathbb{R}^{nbW} \rightarrow \mathbb{R}^{nbc}$$

$$W \mapsto \left(R\left(X^1, W\right) - Y^1, ..., R\left(X^{nbc}, W\right) - Y^{nbc} \right)^T$$

and $R(X, W)$ is the network response in the inputs signals X and weights W.

Gauss-Newton or Levenberg Marquardt

$$\min_{W \in R^{nbW}} J(W) \quad \text{where} \quad J(W) = \frac{1}{2}\left\| E(W) \right\|^2$$

- **Newton's method:** $\nabla^2 J(W)\, d = -\nabla J(W)$

$$\nabla^2 J(W) = DE(W)^T\, DE(W) + \sum_{l=1}^{nbc} E_i(W) \nabla^2 E_i(W)$$

where DE is the Jacobian matrix (nbc x nbW) of E.

- **Gauss-Newton:** $DE(W)^T\, DE(W)\, d = -DE(W)^T\, E(W)$

- **Levenberg-Marquardt:**
when ***nbc*** << ***nbW***, the matrix $DE(W)^T\, DE(W)$ is not invertible.

$$(DE(W)^T\, DE(W) + \alpha I)\, d = -DE(W)^T\, E(W)$$

Linear systems are solved using the conjugated gradient method which requires only matrix-vector products.

A method that uses low-memory

$$DE(W)^T \underbrace{DE(W)d}_{z}$$

We calculate neither $DE(W)$, nor the product $DE(W)^T DE(W)$,

we store neither $DE(W)^T DE(W)$, nor its factorization.

- $DE(W)d = \left(\frac{\partial}{\partial \varepsilon} E(W + \varepsilon d) \right)_{\varepsilon=0}$

  differentiation of a vectoriel function w.r.t. a single parameter

  ⇒ **direct mode of AD**

- $DE(W)^T z = D_W (E(W)^T z)$

  differentiation of a scalar function w.r.t. many parameters

  ⇒ **reverse mode of AD**

Training algorithm

- Given W^0
- for k = 0,1,2,…
  - for $\alpha > 0$, use conjugate gradient to solve

$$(DE(W)^T DE(W) + \alpha I)d = -DE(W)^T E(W)$$

  - and set $W^{k+1} = W^k + d$
- end

Generalization

$\{1, 2, ..., , nbc\} = I_L \cup I_G$ with $I_L \cap I_G = 0$

- $\Omega_L = \{ (X^l, Y^l), l \in I_L \}$ to train the network

$$e_L = \frac{1}{2} \sum_{i \in I_L} \left(R\left(X^i, W\right) - Y^i \right)^2$$

- $\Omega_G = \{ (X^l, Y^l), l \in I_G \}$ to measure the network capability to give a response that approaches the real values for data which have not been learned yet

$$e_G = \frac{1}{2} \sum_{i \in I_G} \left(R\left(X^i, W\right) - Y^i \right)^2$$

Regularization

- Gauss-Newton
- Tikhonov regularization
- Number of hidden neurons reduction
- Early stopping

Regularization

- Gauss-Newton
- Tikhonov regularization
- Number of hidden neurons reduction
- Early stopping

Tikhonov regularization

To avoid oscillations in the basis function, we solve

$$\min_{W \in R^{nbW}} \frac{1}{2}\left(\left\| E(W) \right\|^2 + \beta \left\| W^0 \right\|^2 \right)$$

Strategy to choose the parameter β

- $\{1, 2, ..., nbc\} = I_L \cup I_G \cup I_V$
- **Phase 1:**
  - training network using I_L with $\beta = 0$
    - while $\left\| E(W) \right\| \geq tol$
      - increase the number m of hidden neurons and train the network
  - Train the network using I_L with β which minimizes the error in the set I_G
- **Phase 2 :**
  - train the network using sets I_L and I_G with m and β calculated in phase 1 and verify it on I_V

Regularization

- Gauss-Newton
- Tikhonov regularization
- Number of hidden neurons reduction
- Early stopping

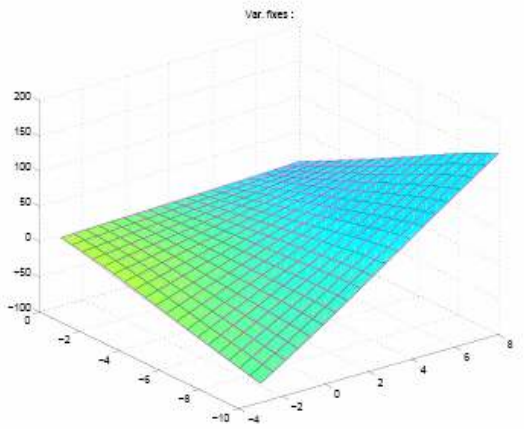
Regularization

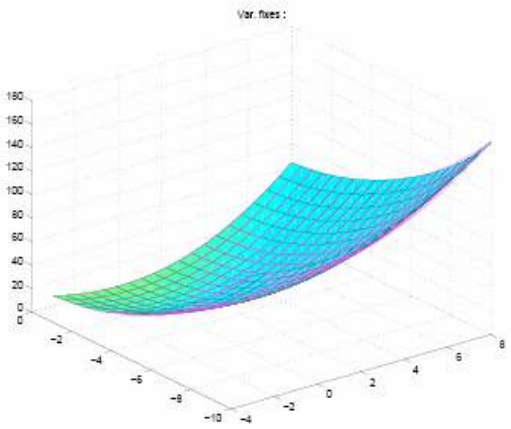
- Gauss-Newton
- Tikhonov regularization
- Number of hidden neurons reduction
- Early stopping

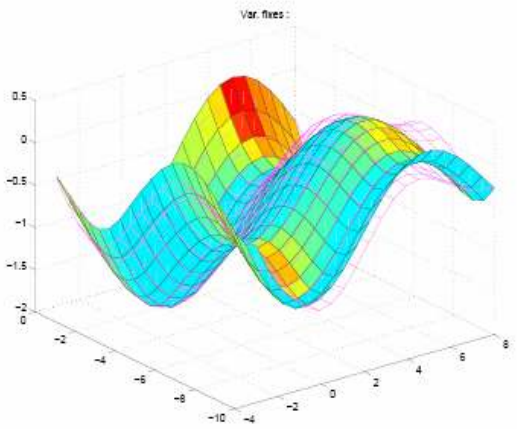
A first example

$$f \;:\; [-3,8] \times [-10,0] \;\rightarrow\; R^3$$

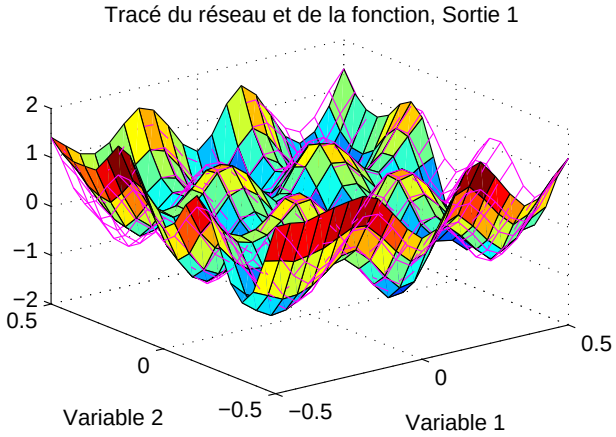
$$(x,y) \;\mapsto\; \left(x+y-2xy,\; x^2+y^2,\; -\frac{\sin(x)}{x} - \frac{\sin(y+2)}{(y+2)} \right)$$

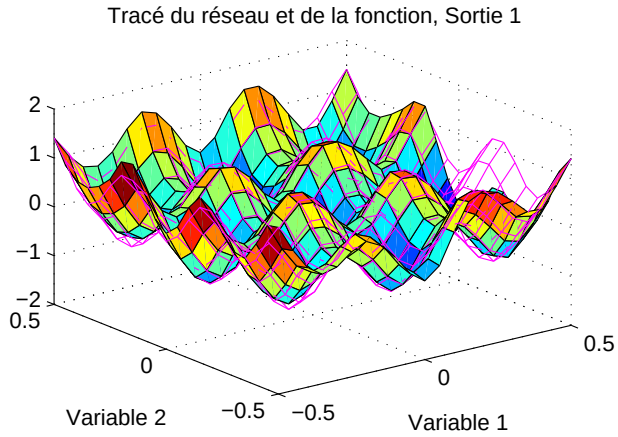


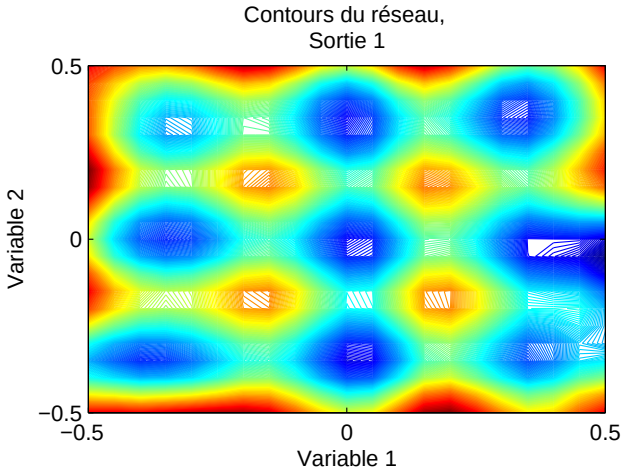


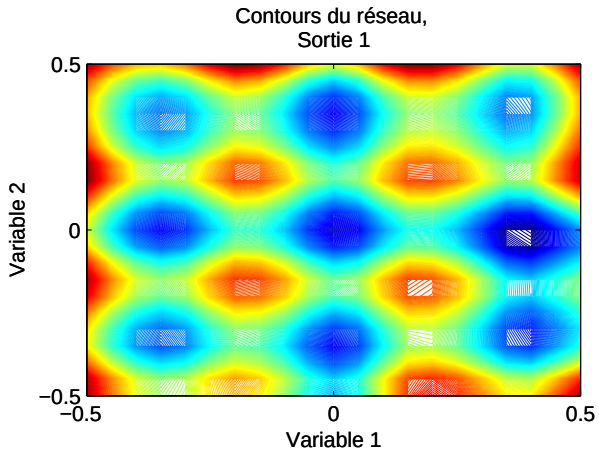


Another example









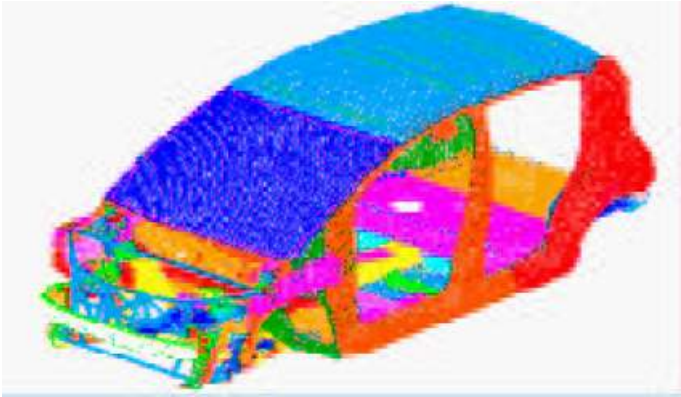
A real case

Consortium MONASTIR: CADOE, MECALOG, MICHELIN, MIP, PSA, RENAULT, SNCF

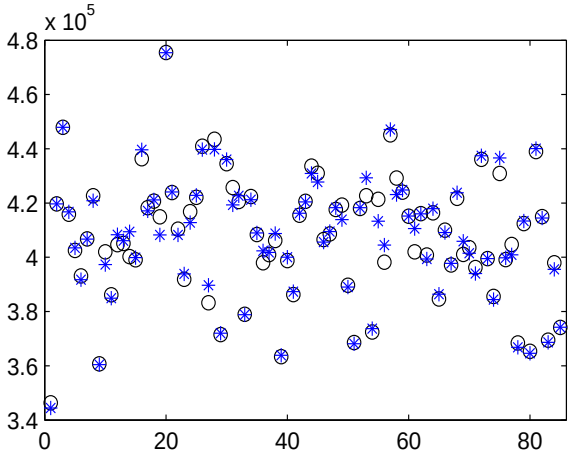
Model of CRASH on a distortable barrier

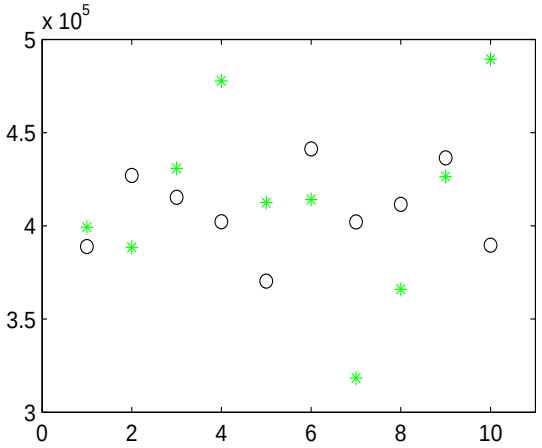
1 output that depends on 7 parameters

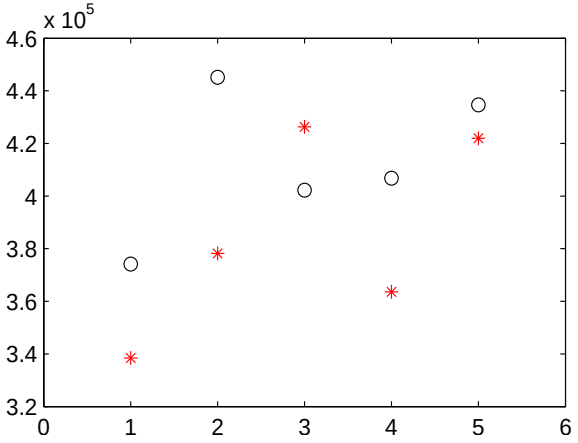
x_1	stiffness of the bumper	x_5	stiffness of the barrier
x_2	horizontal position of the barrier	x_6	vertical position of the barrier
x_3	incidence angle of the vehicle	x_7	Incidence speed
x_4	seating of the vehhicle		

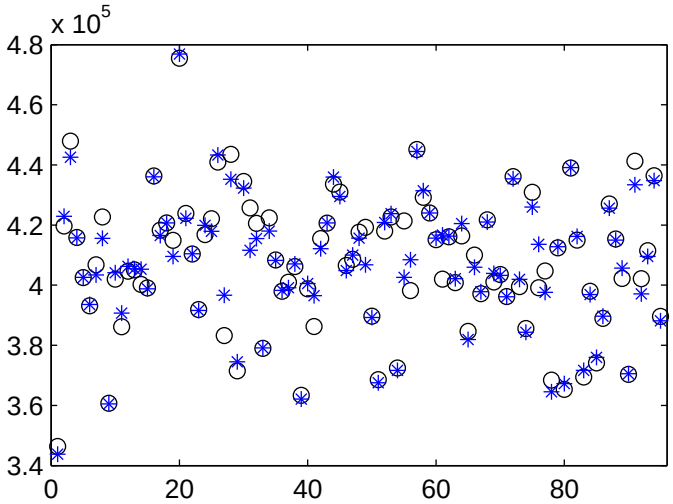


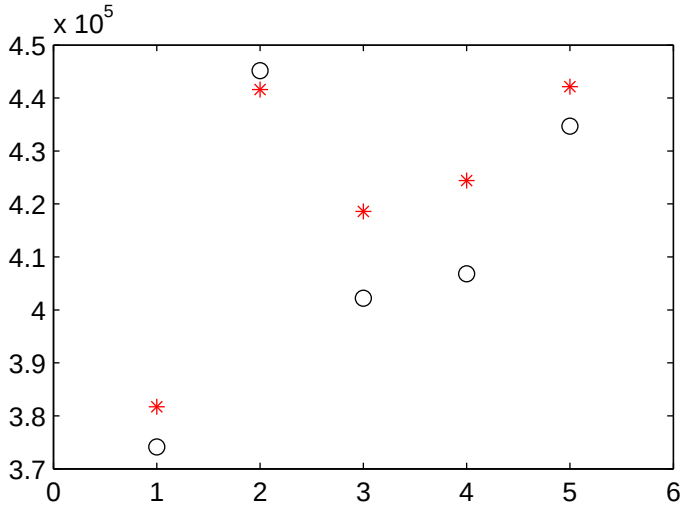
First and final responses



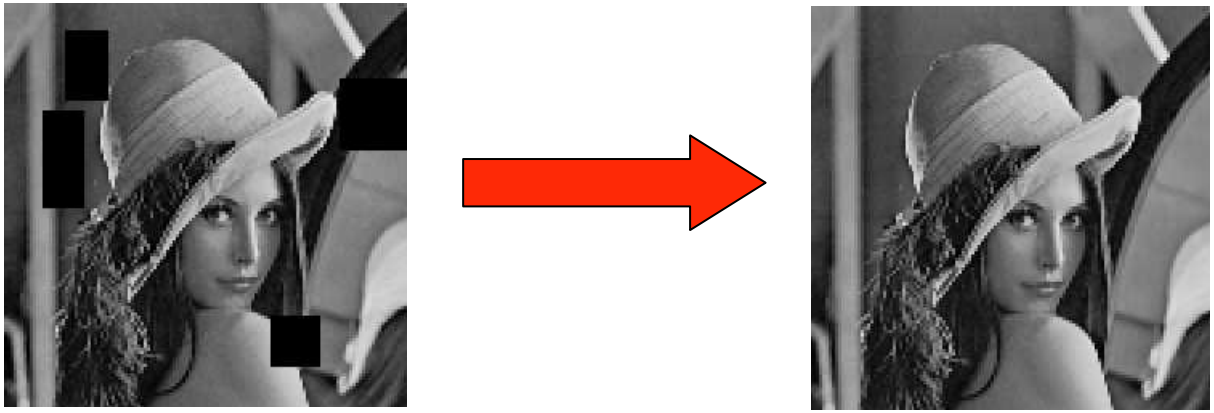








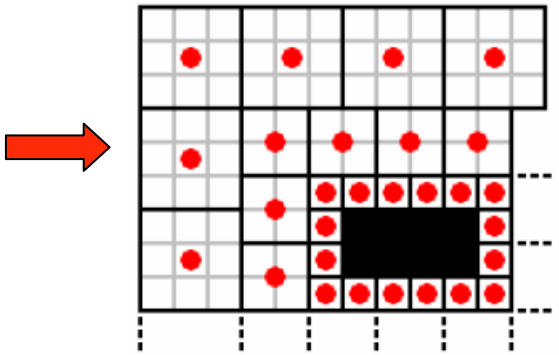
Another example : image restoration



We have a missing image, we want to rebuild the original image

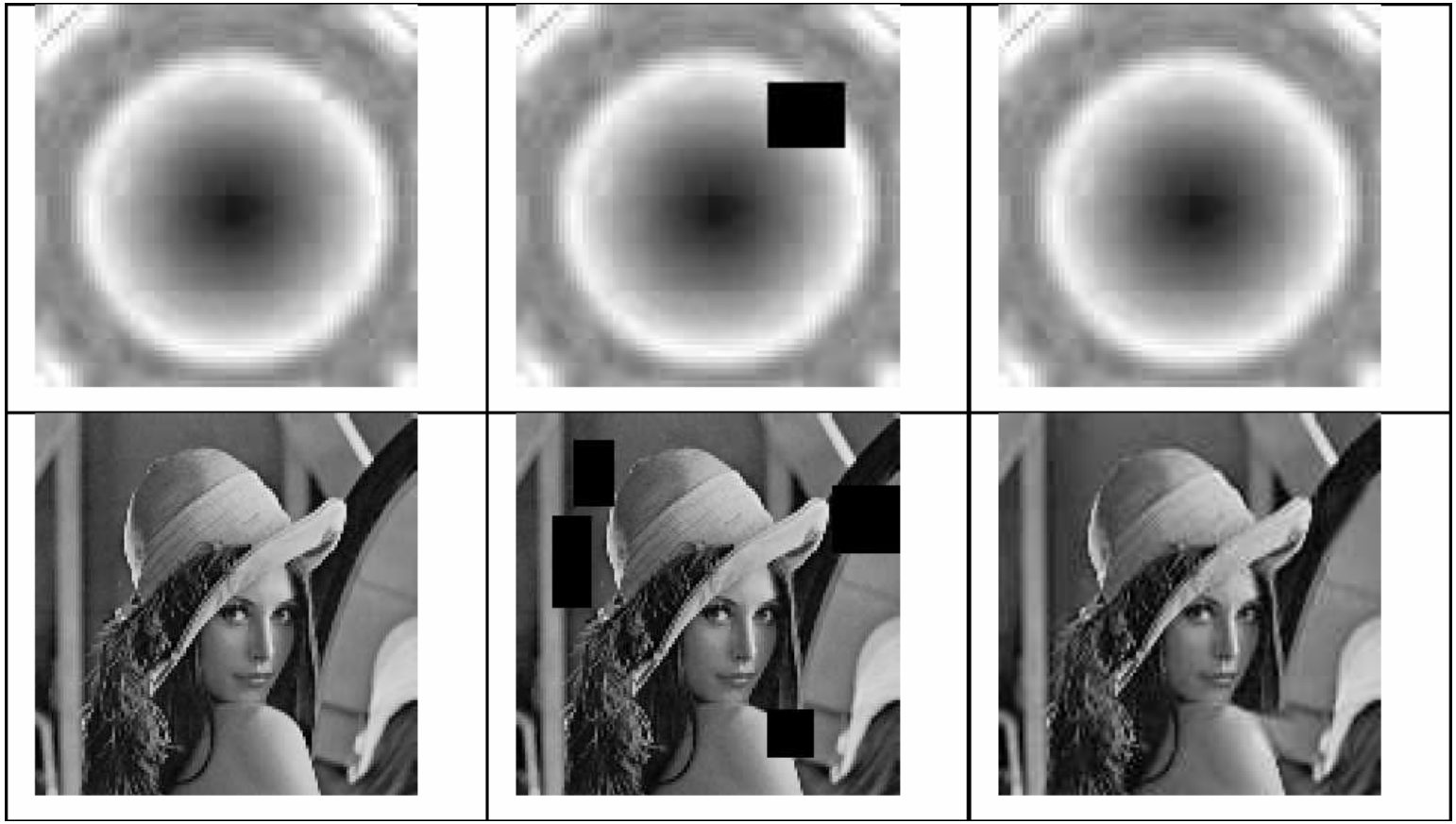
We use an image in gray levels, each point in the image (i,j) is associated with a number I(i,j) such as $0 \leq I(i,j) \leq 255$

choose patterns in a very sensible way



Another example : image restoration

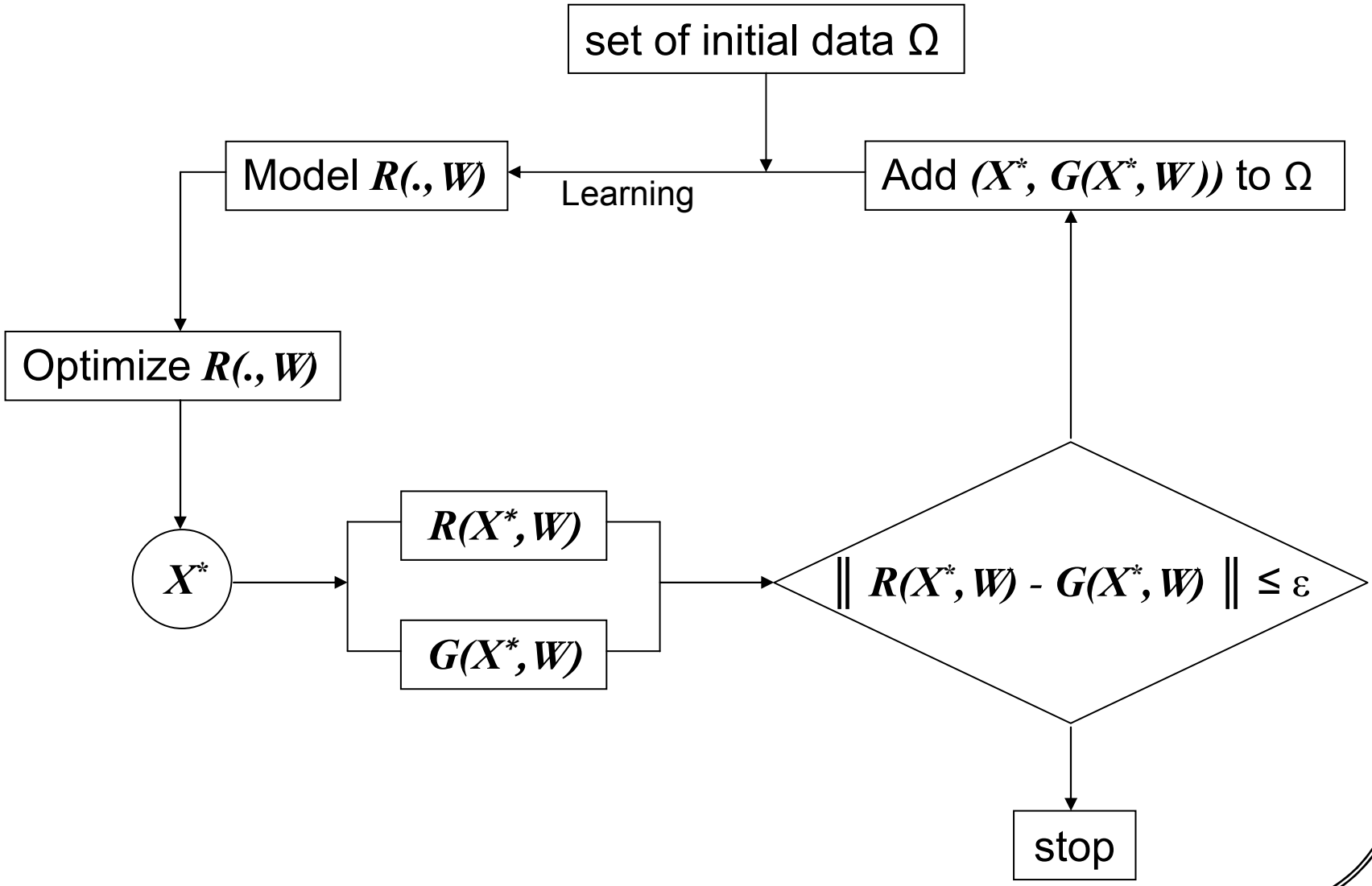
Original image | Image to restore | Image restored



Plan

- Motivations
- Learning with neural networks
  - numerical examples
- Optimization using a neural network
  - numerical examples
- Conclusion remarks

Optimization based on a model

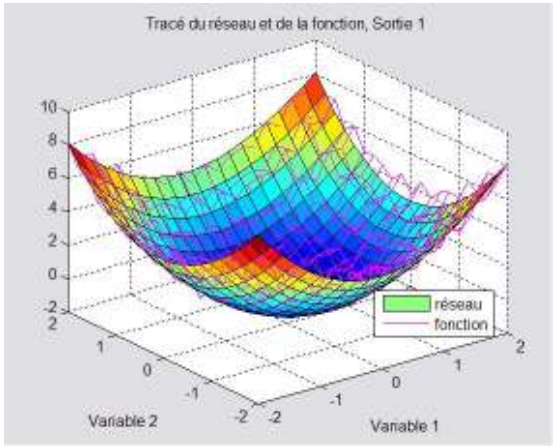


Rastrigin function

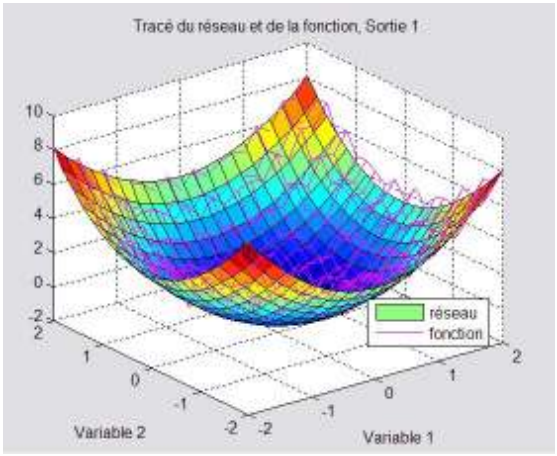
$$f:[-2,2]\times[-2,2]\to\mathbb{R} \quad ; \quad f(x,y)=x^2+y^2-\frac{1}{2}\left(\cos(18x)+\cos(18y)\right)$$

Network : 2 - 3 - 1
function evaluations : 20

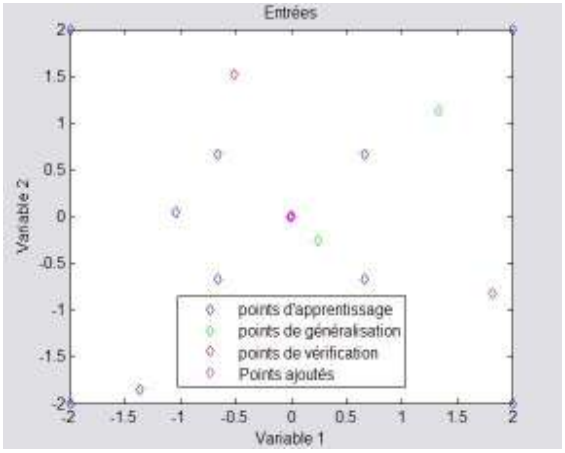
Initial data : 10
calculated minimum : (0.0006, 0.0002)



Initial network



optimal network



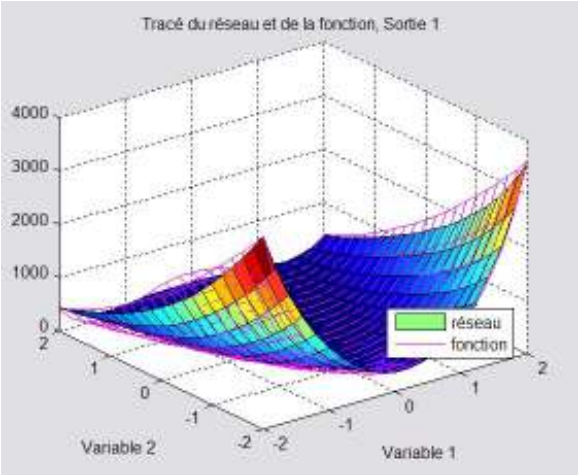
Total points

Rosenbrock function

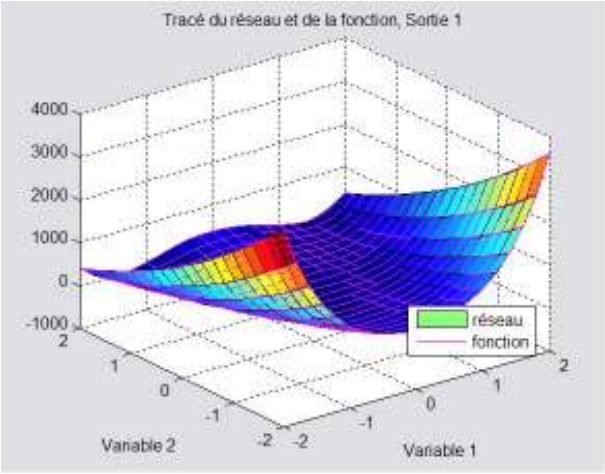
$$f : [-2,2] \times [-2,2] \to \mathbb{R} \qquad f(x,y) = 100\left(y - x^2\right)^2 + \left(1 - x\right)^2$$

Network : 3 - 15 - 1
function evaluations : 46

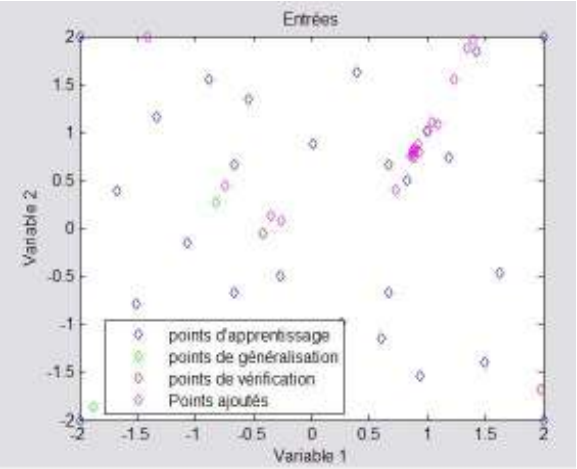
Initial data : 25
calculated minimum : (1.0002,1.001)



Initial network



optimal network



Total points

Functions in two dimensions

Fonction	Domaine	# F	Sol. Exacte	Sol. Appr.
$(x-2)^2+(y-2)^2$	$-3 \le x, y \le 3$	13	(2,2)	(1.999,1.999)
$-\frac{\sin(x-3)}{(x-3)}-\frac{\sin(y+5)}{(y+5)}$	$-3 \le x \le 8$ $-10 \le y \le 0$	29	(3,-5)	(2.999,-5.00)
Branin	$-5 \le x \le 10$ $0 \le y \le 15$	30	(3.142,2.275)	(3.146,2.275)
$(30+x\sin(x))(4+e^{-y})$	$0 \le x, y \le 5$	12	(5,5)	(5,5)
$e^x(4x^2+2y^2+4xy+2y+1)$	$-1 \le x, y \le 1$	13	(0.5,-1)	(0.5,-1)
Griewank	$-100 \le x, y \le 100$	20	(0.0)	(-0.0001, -0.007)
Freudenstein	$0 \le x \le 15$ $2 \le y \le 5$	22	(5,4)	(5.004,3.991)

Functions in two dimensions

Fonction	Domaine	#F	Sol. exacte	Sol. appr.
Rosenbrock	$-2 \le x, y \le 2$	46	(1,1)	(1.0002,1.001)
Goldstein	$-2 \le x, y \le 2$	46	(0,1)	(-0.08,1.01)
Hamma	$0 \le x, y \le 5$	24	(2,2)	(1.99,2.00)
Rastrigin	$-2 \le x, y \le 2$	20	(0,0)	(-0.0001, -0.0002)
Chameau	$-1 \le x, y \le 1$	25	(0.08,0.71)	(0.081,0.710)

Functions in higher dimension

Function	Input var.	Output var.	# Initial F	# total F
Carredec	6	1	16	56
	10	1	32	82
Trigo	6	1	16	56
	10	1	32	82
Neumaier	6	1	16	56
Rastrigin	5	1	40	52
Compressor	8	2	32	62
Turbine	10	108	45	60

Carredec function: $f(X) = \sum_{i=1}^{n} \left(x_i - 2 + 4\frac{i-1}{n-1} \right), \quad X \in [-3,3]^n$

Neumaier function: $f(X) = \sum_{i=1}^{n} (x_i - 1)^2 - \sum_{i=2}^{n} x_i x_{i-1}, \quad X \in \left[-n^2, n^2\right]^n$

Trigo function: $f(X) = n - \sum_{i=1}^{n} \sin(x_i), \quad X \in [-\pi, \pi]^n$

Concluding remarks

- Development of a learning method without Jacobian calculations
- Use of regularization techniques
- Optimization model which enhances itself in each iteration