OTDM / MIRI

Neural Networks classifier with first order unconstrained optimization algorithms

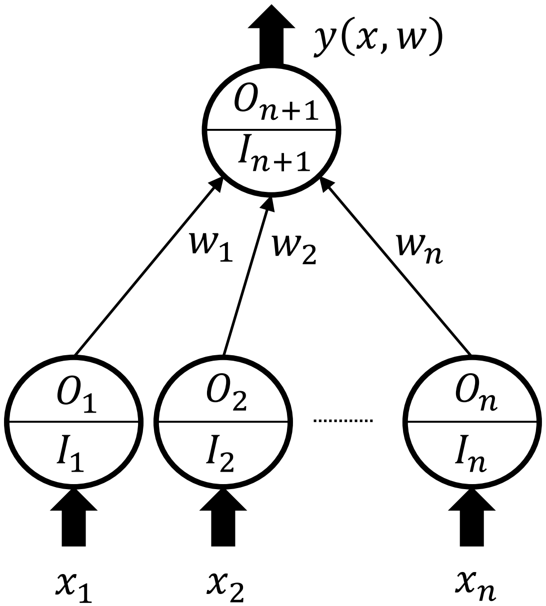
Prof. F. Javier Heredia (f.javier.heredia@upc.edu)

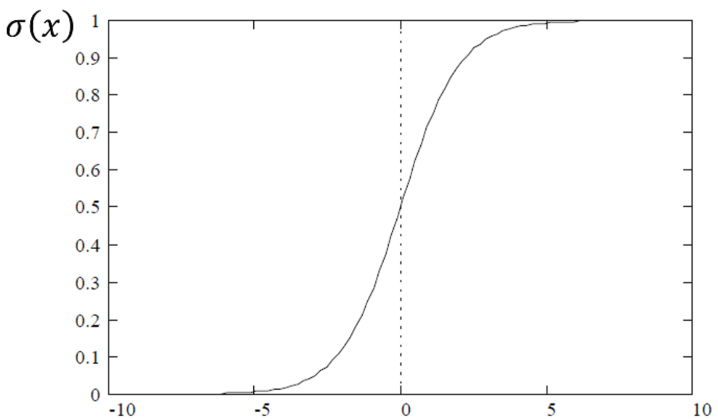
Dept. of Statistics and Operational Research

Universitat Politècnica de Catalunya (UPC)

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH

Single layer Neural Network architecture





- **Input signal:**

$$I_i = x_i, i = 1,2,\ldots,n \quad ; \quad I_{n+1} = \sum_{i=1}^{n} w_i \cdot O_i$$

- **Activation function (sigmoid function) :**

$$O_i = \sigma(I_i) \ , \qquad \sigma(x) = 1/1 + e^{-x}$$

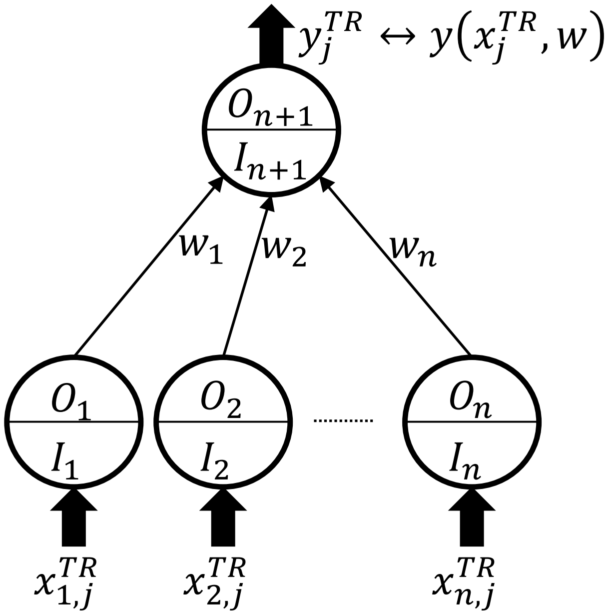
- **Output signal:** assumed to be binary

$$y(x,w) = \quad \sigma(I_{n+1}) = \sigma\left(\sum_{i=1}^{n} w_i O_i\right) = \sigma\left(\sum_{i=1}^{n} w_i \cdot \sigma(x_i)\right)$$

$$= \left(1 + e^{-\left(\sum_{i=1}^{n} w_i \cdot \sigma(x_i)\right)}\right)^{-1}$$

$$= \left(1 + e^{-\left(\sum_{i=1}^{n} w_i \cdot (1+e^{-x_i})^{-1}\right)}\right)^{-1}$$

Single layer Neural Network : training



- **Training data set, size p:**

$$X^{TR} = [x_1^{TR}, x_2^{TR}, \dots, x_p^{TR}] = \begin{bmatrix} x_{1,1}^{TR} & x_{1,2}^{TR} & \cdots & x_{1,p}^{TR} \\ x_{2,1}^{TR} & x_{2,2}^{TR} & \cdots & x_{2,p}^{TR} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n,1}^{TR} & x_{n,2}^{TR} & \cdots & x_{n,p}^{TR} \end{bmatrix}$$

$$y^{TR} = \begin{bmatrix} y_1^{TR} & y_2^{TR} & \cdots & y_p^{TR} \end{bmatrix}^T$$

- **Loss function:** for a given (X^{TR}, Y^{TR})

$$L(X^{TR}, y^{TR}) = \min_{w \in \mathbb{R}^n} L(w; X^{TR}, y^{TR}) = \sum_{j=1}^{p} \left(y(x_j^{TR}, w) - y_j^{TR} \right)^2$$

- **Loss function with L2 regularization with param. λ:**

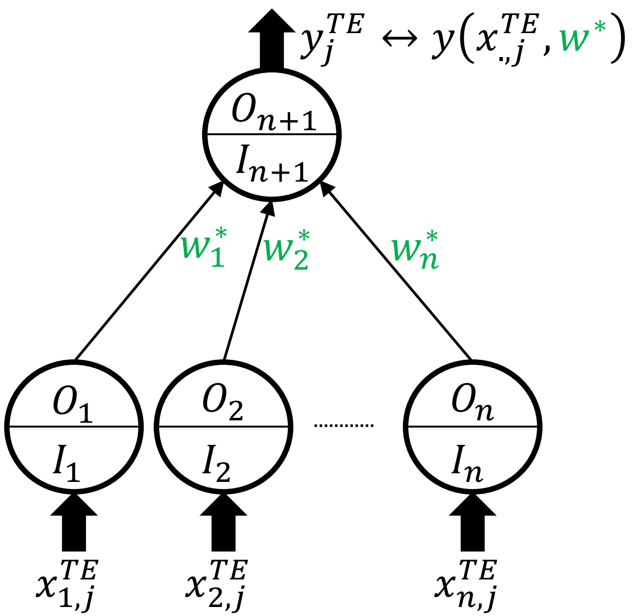
$$\tilde{L}(X^{TR}, y^{TR}, \lambda)$$
$$= \min_{w \in \mathbb{R}^n} \tilde{L}(w; X^{TR}, y^{TR}, \lambda) = L(w; X^{TR}, y^{TR}) + \lambda \cdot \frac{\|w\|^2}{2}$$

- **Training accuracy (%):** $w^* = \operatorname{argmin}_{w \in \mathbb{R}^n} \tilde{L}(w; X^{TR}, y^{TR}, \lambda)$

$$\mathbf{Accuracy}^{TR} = \frac{100}{p} \cdot \sum_{j=1}^{p} \delta_{\underbrace{[y(x_j^{TR}, w^*)]}_{round(\cdot)}, y_j^{TR}}$$

where $\delta_{x,y} = \begin{cases} 1 & if\ x = y \\ 0 & if\ x \neq y \end{cases}$ (*Kronecker delta*).

Single layer Neural Network : testing



- **Test data set, size q:**

$$X^{TE} = \left[x_1^{TE}, x_2^{TE}, \dots, x_q^{TE}\right] = \begin{bmatrix} x_{1,1}^{TE} & x_{1,2}^{TE} & \cdots & x_{1,q}^{TE} \\ x_{2,1}^{TE} & x_{2,2}^{TE} & \cdots & x_{2,q}^{TE} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n,1}^{TE} & x_{n,2}^{TE} & \cdots & x_{n,q}^{TE} \end{bmatrix}$$

$$y^{TE} = \left[y_1^{TE} \quad y_2^{TE} \quad \cdots \quad y_q^{TE}\right]^T$$

- **Test accuracy (%):**

$$\text{Accuracy}^{TE} = \frac{100}{p} \cdot \sum_{j=1}^{p} \delta_{\left[y\left(x_j^{TE}, w^*\right)\right], y_j^{TE}}$$

- **Overfitting:** if $\text{Accuracy}^{TR} \ll \text{Accuracy}^{TE}$

Single layer Neural Network : gradient (1/2)

- **Loss function (objective function):**

$$\tilde{L}(w; X^{TR}, y^{TR}, \lambda) = \sum_{j=1}^{p} \left(y\left(x_j^{TR}, w\right) - y_j^{TR}\right)^2 + \frac{\lambda}{2} \cdot \sum_{i=1}^{n} w_i^2$$

- **Gradient:**

$$\frac{\partial \tilde{L}(w; X^{TR}, y^{TR}, \lambda)}{\partial w_i} = \sum_{j=1}^{p} 2 \cdot \left(y\left(x_j^{TR}, w\right) - y_j^{TR}\right) \cdot \frac{\partial y\left(x_j^{TR}, w\right)}{\partial w_i} + \lambda \cdot w_i \quad (1)$$

with

$$y\left(x_j^{TR}, w\right) = \left(1 + e^{-\left(\sum_{i=1}^{n} w_i \cdot \left(1 + e^{-x_{i,j}^{TR}}\right)^{-1}\right)}\right)^{-1} \quad (2)$$

Single layer Neural Network : gradient (2/2)

- Let us find $\frac{\partial y\left(x_j^{TR},w\right)}{\partial w_i}$:

$$\frac{\partial y\left(x_j^{TR},w\right)}{\partial w_i} = \frac{\partial}{\partial w_i}\left(1+e^{-\left(\sum_{i=1}^{n} w_i\cdot\left(1+e^{-x_{i,j}^{TR}}\right)^{-1}\right)}\right)^{-1} =$$

$$= \overbrace{-\left(1+e^{-\left(\sum_{i=1}^{n} w_i\cdot\left(1+e^{-x_{i,j}^{TR}}\right)^{-1}\right)}\right)^{-2}}^{-y\left(x_j^{TR},w\right)^2}\cdot \overbrace{e^{-\left(\sum_{i=1}^{n} w_i\cdot\left(1+e^{-x_{i,j}^{TR}}\right)^{-1}\right)}}^{\left(y\left(x_j^{TR},w\right)^{-1}-1\right)}\cdot\left(-\left(1+e^{-x_{i,j}^{TR}}\right)^{-1}\right)$$

$$= y\left(x_j^{TR},w\right)^2\cdot\left(y\left(x_j^{TR},w\right)^{-1}-1\right)\cdot\left(1+e^{-x_{i,j}^{TR}}\right)^{-1}$$

$$= \left(y\left(x_j^{TR},w\right)-y\left(x_j^{TR},w\right)^2\right)\cdot\left(1+e^{-x_{i,j}^{TR}}\right)^{-1}$$

Therefore:

$$\boxed{\frac{\partial\tilde{L}\left(w;X^{TR},y^{TR},\lambda\right)}{\partial w_i} = \sum_{j=1}^{p} 2\cdot\left(y\left(x_j^{TR},w\right)-y_j^{TR}\right)\cdot\left(y\left(x_j^{TR},w\right)-y\left(x_j^{TR},w\right)^2\right)\cdot\left(1+e^{-x_{i,j}^{TR}}\right)^{-1}+\lambda\cdot w_i}$$

Single layer Neural Network : first classification problem

- **Classification rule 1:**

$$x_i \in \{0,1\}, \qquad y = \begin{cases} 1 & \text{if } x_2 > x_1 \\ 0 & \text{otherwise} \end{cases}, \qquad i = 1,2$$

- **Populating the training dataset $\left(X^{TR}, y^{TR}\right)$ in `Matlab`:**

```
training_seed = 123456;
rng(training_seed);
xtr = round(rand(2,p));
ytr = zeros(p,1)';
for j=1:p
    if xtr(2,j) > xtr(1,j)
        ytr(j) = 1;
    end
end
```

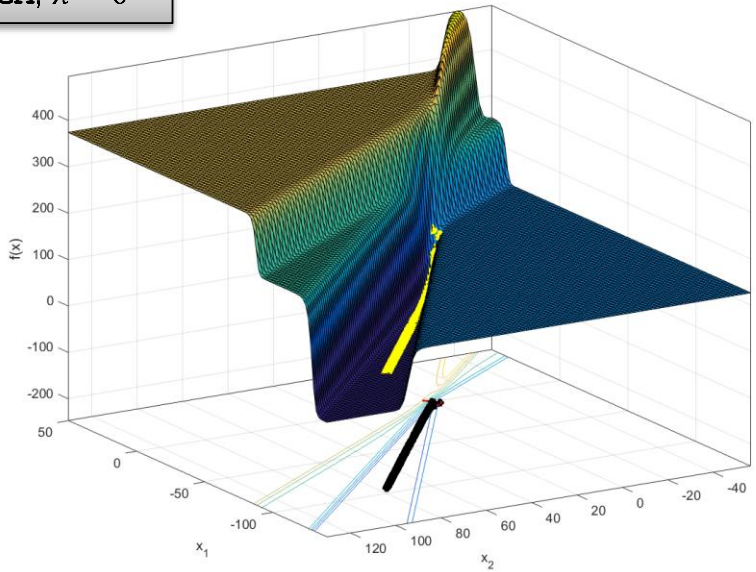
Single layer Neural Network : classification problem #1

- **Results(*)**:

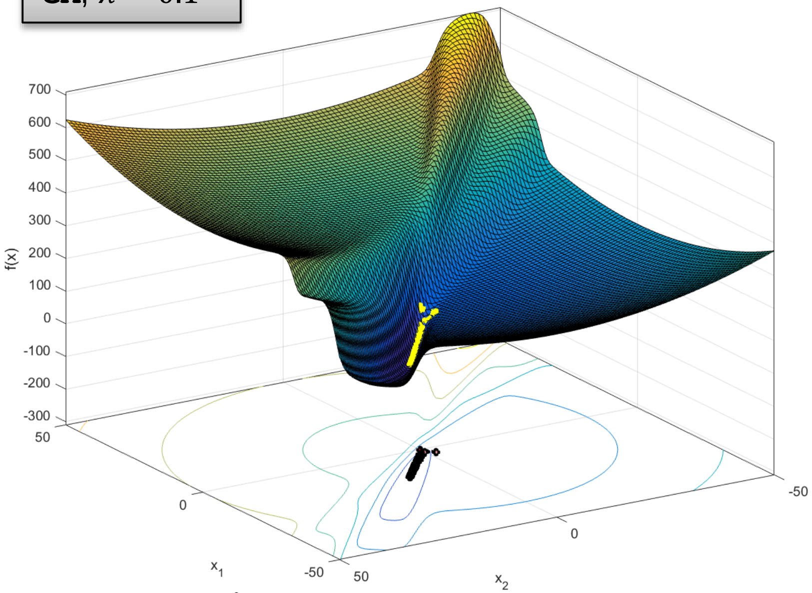
r	tr_p	tr_seed	la	w1(1)	w1(2)	ls	c2	sdm	iout	iter	w*(1)	w*(2)	L*	tr_acc	te_acc	te_q	te_seed
1	500	1234566	0.000	0.0	0.0	1	0.5	GM	0	756	-93.5	76.1	5.771e-06	100.0	100.0	50000	7891016
1	500	1234566	0.000	0.0	0.0	1	0.5	BFGS	2	3	-5.4	2.5	6.888e+01	74.8	74.9	50000	7891016
1	500	1234566	0.000	0.0	0.0	1	0.9	BFGS	2	3	-5.4	2.5	6.888e+01	74.8	74.9	50000	7891016
1	500	1234566	0.000	0.0	0.0	2	0.5	BFGS	0	6	-9946.6	7695.9	0.000e+00	100.0	100.0	50000	7891016

r	tr_p	tr_seed	la	w1(1)	w1(2)	ls	c2	sdm	iout	iter	w*(1)	w*(2)	L*	tr_acc	te_acc	te_q	te_seed
1	500	1234566	0.100	0.0	0.0	1	0.5	GM	0	445	-13.6	10.9	2.930e+01	100.0	100.0	50000	7891016
1	500	1234566	0.100	0.0	0.0	2	0.5	BFGS	0	9	-13.6	10.9	2.930e+01	100.0	100.0	50000	7891016

GM, $\lambda = 0$



GM, $\lambda = 0.1$



(*) r = rule; tr_p= training dataset size; tr_seed= seed for populating the training set; la= λ; w1= initial point; ls= options(20)= 1=> bactracking with SW, 2=> minfbd; c2= c_2; sdm= search direction (1=> GM; 4=> BFGS); iout= exit status of the optimization (0=> optimal, 1=>too many iterations, 2=> ascent direction, 3=> linesearch failure); iter= # iteration optim.; w*= w^*; L*= $\tilde{L}(x^*)$; tr_acc= $Accuracy^{TR}$; te_acc= $Accuracy^{TE}$; te_p= test dataset size; te_seed= seed for populating the test dataset.

Single layer Neural Network : classification problem #2

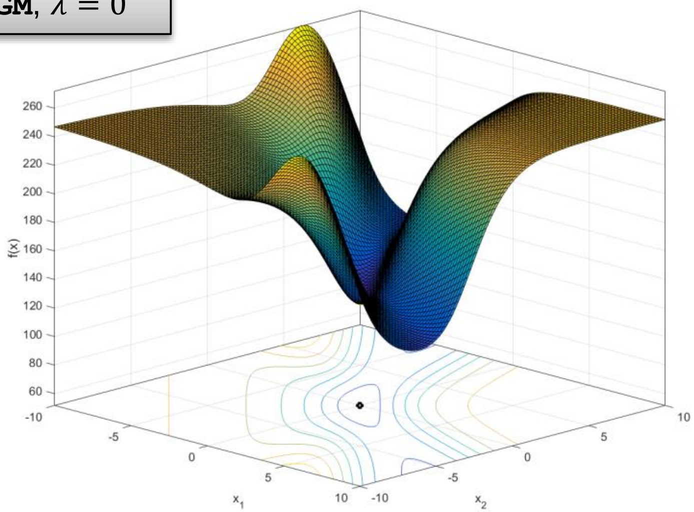
- **Classification rule 2:** $x_i \in \{-1,1\},\ y = \begin{cases} 1 & \text{if } x_2 = x_1 \\ 0 & \text{otherwise} \end{cases},\ i = 1,2$

- **Results(*):**

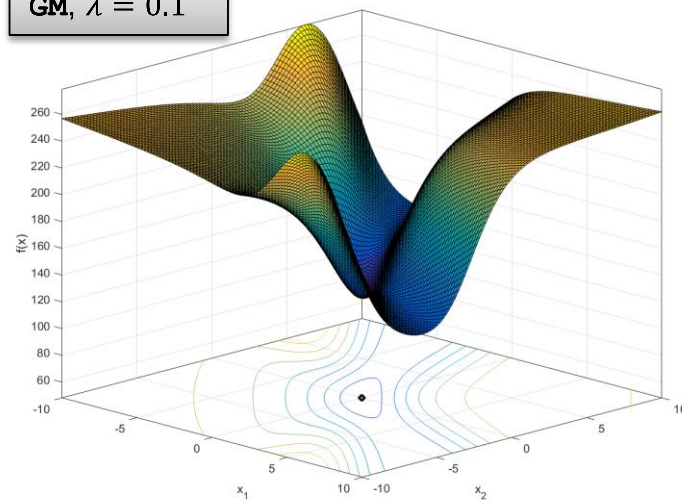
r	tr_p	tr_seed	la	w*(1)	w*(2)	ls	c2	sdm	iout	iter	w*(1)	w*(2)	L*	tr_acc	te_acc	te_q	te_seed
2	500	1234566	0.000	+0.00e+00	+0.00e+00	1	0.5	GM	0	13	-4.40e-03	-4.40e-03	1.250e+02	50.4	49.9	50000	7891016
2	500	1234566	0.000	+0.00e+00	+0.00e+00	1	0.5	BFGS	0	8	-4.40e-03	-4.40e-03	1.250e+02	50.4	49.9	50000	7891016

r	tr_p	tr_seed	la	w*(1)	w*(2)	ls	c2	sdm	iout	iter	w*(1)	w*(2)	L*	tr_acc	te_acc	te_q	te_seed
2	500	1234566	1.000	+0.00e+00	+0.00e+00	1	0.5	GM	0	8	-4.28e-03	-4.28e-03	1.250e+02	50.4	49.9	50000	7891016
2	500	1234566	1.000	+0.00e+00	+0.00e+00	1	0.5	BFGS	0	8	-4.28e-03	-4.28e-03	1.250e+02	50.4	49.9	50000	7891016

GM, $\lambda = 0$



GM, $\lambda = 0.1$



(*) r = rule; tr_p= training dataset size; tr_seed= seed for populating the training set; la= λ; w1= initial point; ls= options(20)= 1=> bactracking with SW, 2=> minfbd; c2= c_2; sdm= search direction (1=> GM; 4=> BFGS); iout= exit status of the optimization (0=> optimal, 1=>too many iterations, 2=> ascent direction, 3=> linesearch failure); iter= # iteration optim.; w*= w^*; L*= $\tilde{L}(x^*)$; tr_acc= $Accuracy^{TR}$; te_acc= $Accuracy^{TE}$; te_p= test dataset size; te_seed= seed for populating the test dataset.

Single layer Neural Network : improving classification problem #2

- The accuracy of the NN classification can be usually improved with a **new architecture** incorporating some additional información.

- **Classification rule 2.1:** same rule #2 but with an **additional third neuron**

$$x_i \in \{-1,1\}, i = 1,2,\mathbf{3} \quad \text{with} \quad \boldsymbol{x_3 = x_1 \cdot x_2}$$

$$y = \begin{cases} 1 & \text{if } x_2 = x_1 \\ 0 & \text{otherwise} \end{cases}$$

- **Results(*)**:

r	tr_p	tr_seed	la	w1(1)	w1(2)	w1(3)	ls	c2	sdm	iout	iter	w*(1)	w*(2)	w*(3)	L*	tr_acc	te_acc	te_q	te_seed
21	500	1234566	0.000	+0.00e+00	+0.00e+00	+0.00e+00	1	0.5	GM	0	439	-2.75e+01	-2.75e+01	+6.76e+01	3.283e-06	100.0	100.0	50000	7891016
21	500	1234566	0.000	+0.00e+00	+0.00e+00	+0.00e+00	1	0.5	BFGS	0	6	-3.88e+01	-3.88e+01	+9.76e+01	3.285e-09	100.0	100.0	50000	7891016

r	tr_p	tr_seed	la	w1(1)	w1(2)	w1(3)	ls	c2	sdm	iout	iter	w*(1)	w*(2)	w*(3)	L*	tr_acc	te_acc	te_q	te_seed
21	500	1234566	0.100	+0.00e+00	+0.00e+00	+0.00e+00	1	0.5	GM	0	264	-5.39e+00	-5.39e+00	+1.32e+01	1.904e+01	100.0	100.0	50000	7891016
21	500	1234566	0.100	+0.00e+00	+0.00e+00	+0.00e+00	1	0.5	BFGS	0	15	-5.39e+00	-5.39e+00	+1.32e+01	1.904e+01	100.0	100.0	50000	7891016

(*) r = rule; tr_p= training dataset size; tr_seed= seed for populating the training set; la= λ; w1= initial point; ls= options(20)= 1=> bactracking with SW, 2=> minfbd; c2= c_2; sdm= search direction (1=> GM; 4=> BFGS); iout= exit status of the optimization (0=> optimal, 1=>too many iterations, 2=> ascent direction, 3=> linesearch failure); iter= # iteration optim.; w*= w^*; L*= $\tilde{L}(x^*)$; tr_acc= $Accuracy^{TR}$; te_acc= $Accuracy^{TE}$; te_p= test dataset size; te_seed= seed for populating the test dataset.

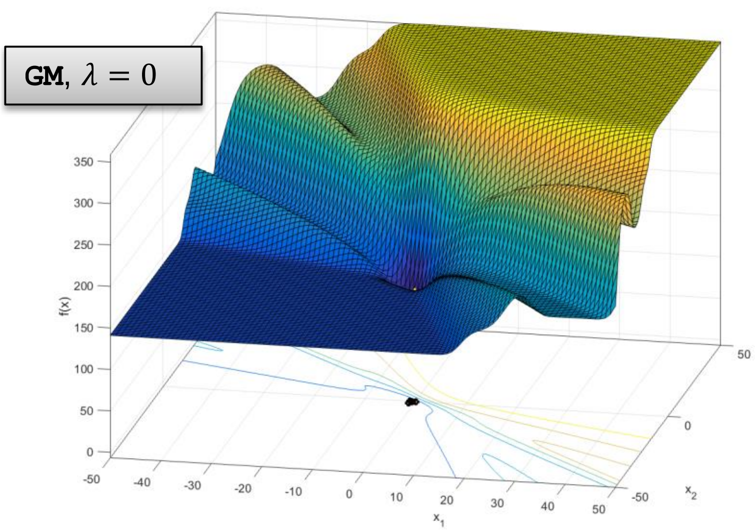
Single layer Neural Network : classification problem #3

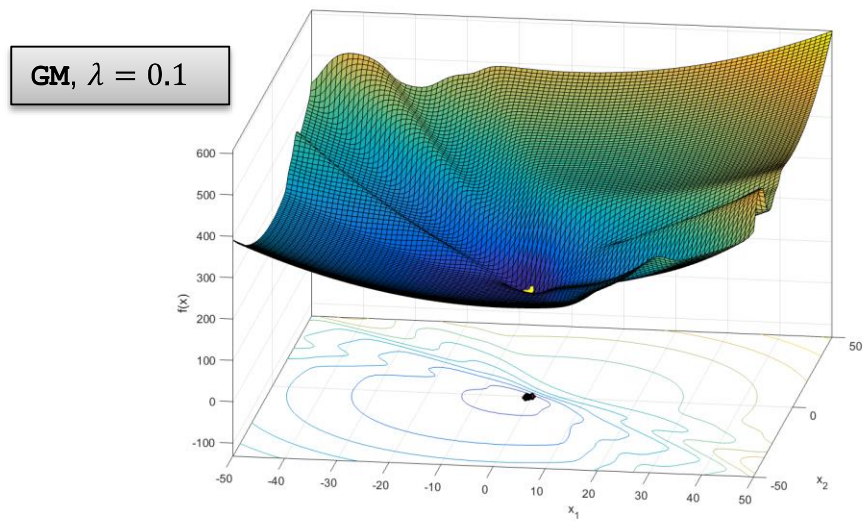
- **Classification rule 3:** $x_i \in \{-1,0,1\},\ y = \begin{cases} 1 & \text{if } x_2 + x_1 > 0 \\ 0 & \text{otherwise} \end{cases},\ i = 1,2$

- **Results(*):**

r	tr_p	tr_seed	la	w1(1)	w1(2)	ls	c2	sdm	iout	iter	w*(1)	w*(2)	L*	tr_acc	te_acc	te_q	te_seed
3	500	1234566	0.000	+0.00e+00	+0.00e+00	1	0.5	GM	0	40	-1.16e+00	-8.38e-02	1.146e+02	71.8	68.5	50000	7891016

r	tr_p	tr_seed	la	w1(1)	w1(2)	ls	c2	sdm	iout	iter	w*(1)	w*(2)	L*	tr_acc	te_acc	te_q	te_seed
3	500	1234566	0.100	+0.00e+00	+0.00e+00	1	0.5	BFGS	0	14	-1.13e+00	-1.15e-01	1.147e+02	71.8	68.5	50000	7891016





(*) r = rule; tr_p= training dataset size; tr_seed= seed for populating the training set; la= λ; w1= initial point; ls= options(20)= 1=> bactracking with SW, 2=> minfbd; c2= c_2; sdm= search direction (1=> GM; 4=> BFGS); iout= exit status of the optimization (0=> optimal, 1=>too many iterations, 2=> ascent direction, 3=> linesearch failure); iter= # iteration optim.; w*= w^*; L*= $\tilde{L}(x^*)$; tr_acc= $Accuracy^{TR}$; te_acc= $Accuracy^{TE}$; te_p= test dataset size; te_seed= seed for populating the test dataset.

Laboratory assignment

- The aim of the assignment is to implement a neural network to solve the classification problem #3 and to improve the accuracy modifying the architecture. The code must:
  1. Generate the training data set (X^{TR}, y^{TR}).
  2. Find the value of w^* minimizing $\tilde{L}(w; X^{TR}, y^{TR}, \lambda)$ with your implementation of function **`otdm_uo_students`**.
  3. Calculate Accuracy^{TR}.
  4. Generate some test dataset (X^{TE}, y^{TE}) and calculate Accuracy^{TE}.
- Use your code to:
  - a) Solve the classification problem #2 with all the first order methods studied at class (GM, CGM-PR, CGM-FR, BFGS, DFP), with $\lambda = 0$ and $\lambda = 0.1$. Check that the results of your code coincide with the results in the slides.
  - b) Find an improvement of the NN architecture including a third neuron and solve the modified classification problem with the GM and BFGS. Analyze the change in Accuracy^{TE}
- The assignment is in groups of two. You must upload to Atenea a file with the name **`surname-student-1_surname-student-2.zip`** containing:
  - a) A report (.pdf file) with your results and comments of sections a) and b).
  - b) All the source code used in this assignment.