Complexity in social dynamics : from the micro to the macro Laboratory 1

Franco Bagnoli

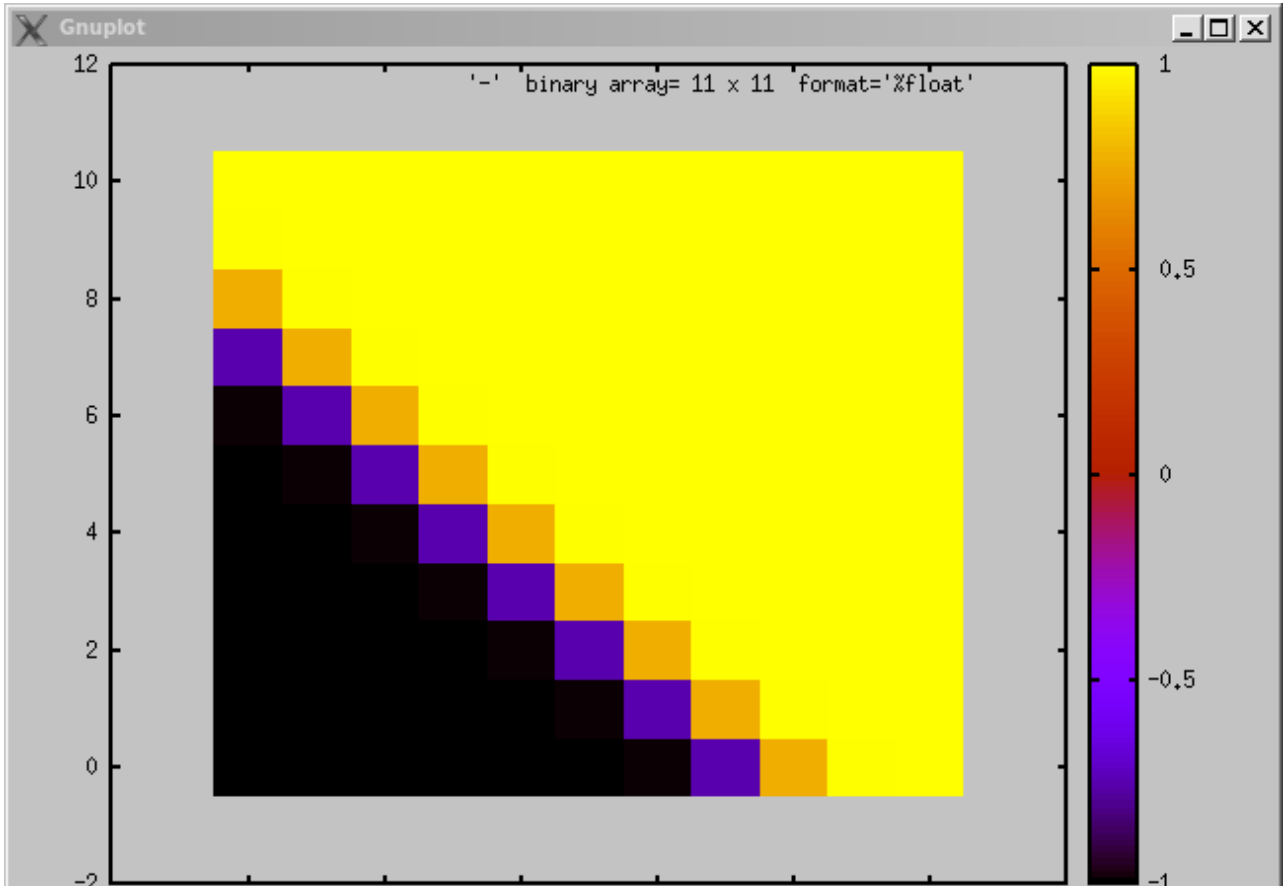
Namur 7-18/4/2008

Neural networks

1. Perceptron. Linear classification.
2. Perceptron learning.
3. Multi-layer perceptron.
4. Backpropagation
5. Unsupervised learning (to do).
6. Hopfield model (to do).

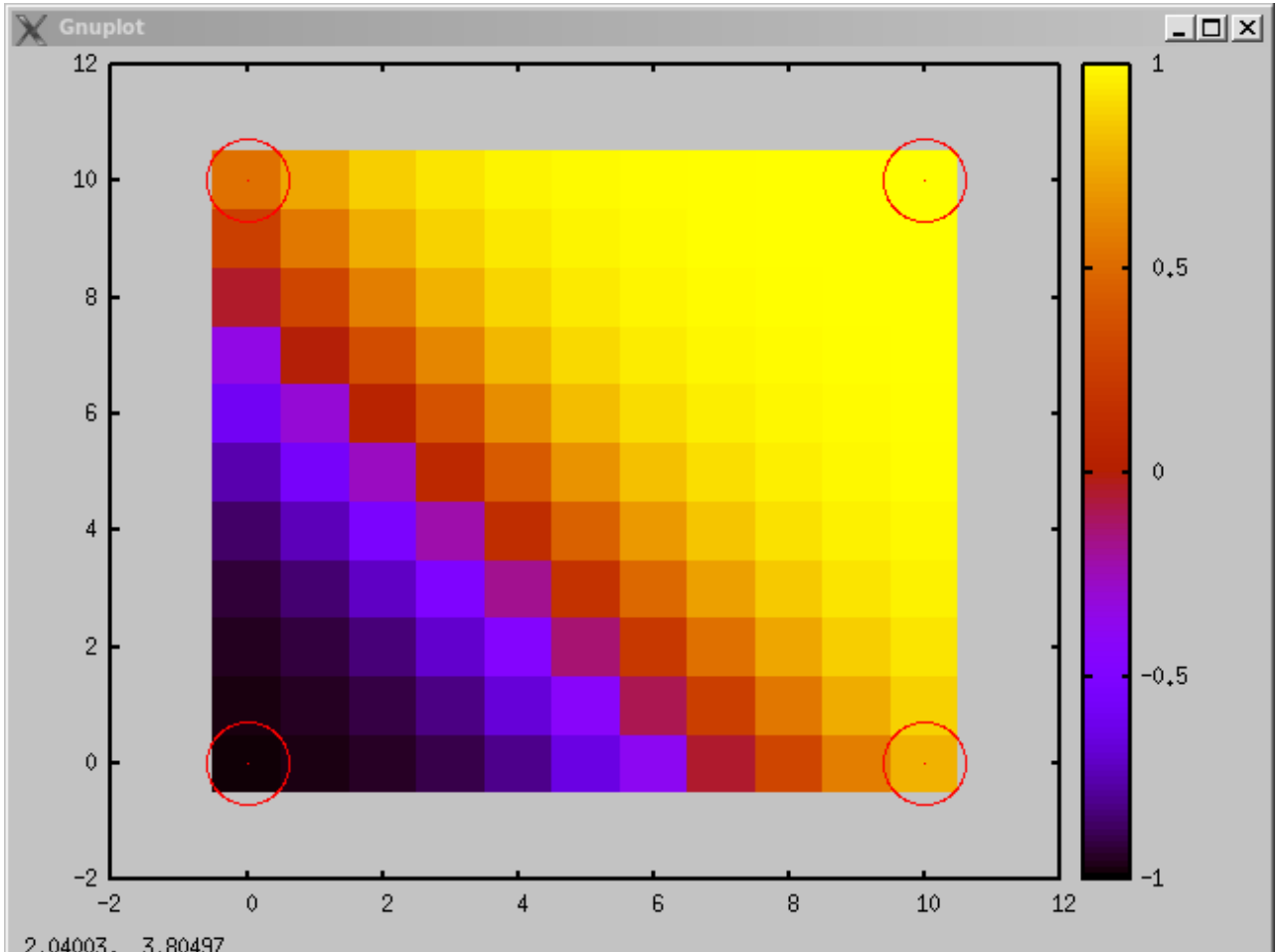
Perceptron

- The simple perceptron with two inputs is characterized by the two weights w_1 and w_2, and the threshold b (`perceptron.f90`)
- Experiment with the program and discover the role of parameters.



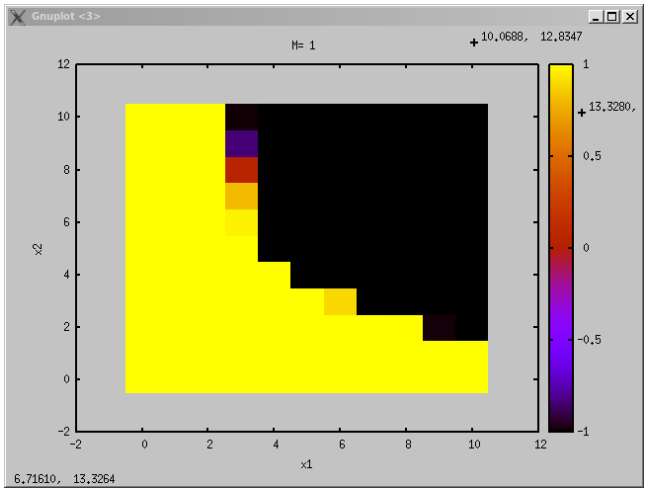
Perceptron learning

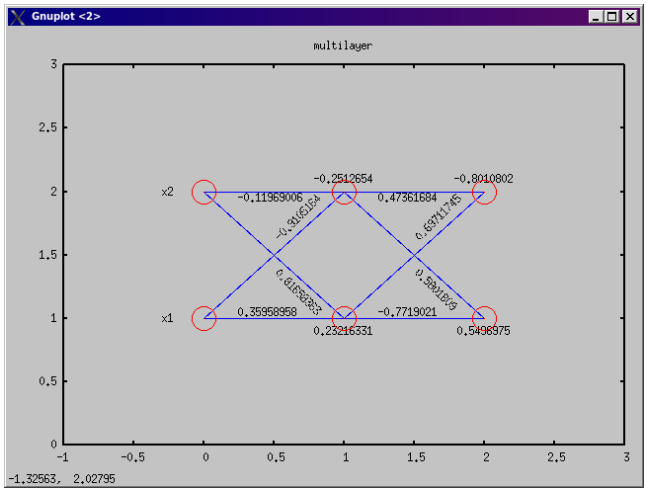
- Perceptron can "learn" how to discriminate about linearly separable examples (`perceptronLearning.f90`)
- However, it is not able to learn how to separate the inputs into disconnected sets (the "XOR" problem).



Multi-layer perceptron

- By adding more layer one can classify arbitrary regions (`multiPerceptron.f90`)
- The problem is how to design the weights and the thresholds.
- Notice that when the number of layer is large, the output is homogeneous. This is because the random weights and threshold adds to give about zero, so the last b dominates.





Supervised learning

- Supervised learning means changing weights and threshold in order to minimise the error (difference between the expected output $\overline{y}^{(k)}$ and the actual one $y^{(k)}(x_1^{(k)}, x_2^{(k)})$).
- It is a minimization problem similar to least squares. Since however the "perceptron" function is in general not invertible, one has to use iterative techniques.
- One of these techniques is gradient descent. Compute the gradient of the function to be minimized and take a step in the opposite direction (backpropagation). Another possibility is simulated annealing.
- In gradient descent there are two possible problems: the choice of the step and the "insensibility" of weights that have value near 1 or -1.

Backpropagation 1

The increase in error $\Delta^{(k)}\xi^2 = \left(\overline{y}^{(k)} - y^{(k)}\right)^2$ due to example k is (neglecting k)

$$\frac{\partial \Delta\xi^2}{\partial w} = 2(\overline{y} - y)\frac{\partial y}{\partial w},$$

where

$$x_i(\ell) = \tanh\left(\sum_j w_{ij}x_j(\ell - 1) + b_i(\ell)\right),$$

and $x_i = x_i(0)$, $y = x_1(L+1)$ and $\ell \in \{1, \ldots, L\}$ numbers the layers.

Backpropagation 2

The gradient descent gives for the w a recursive formula. First compute the errors backwards

$$\delta_j(\ell) = (1 - x_j^2(\ell)) \sum_i w_{ij}(\ell+1)\delta_i(\ell+1),$$

(remember that the derivative of $\tanh(x)$ is $1 - tanh^2(x)$), and then

$$w'_{ij}(\ell) = w_{ij}(\ell) + \eta x_j(\ell-1)\delta_i(\ell).$$

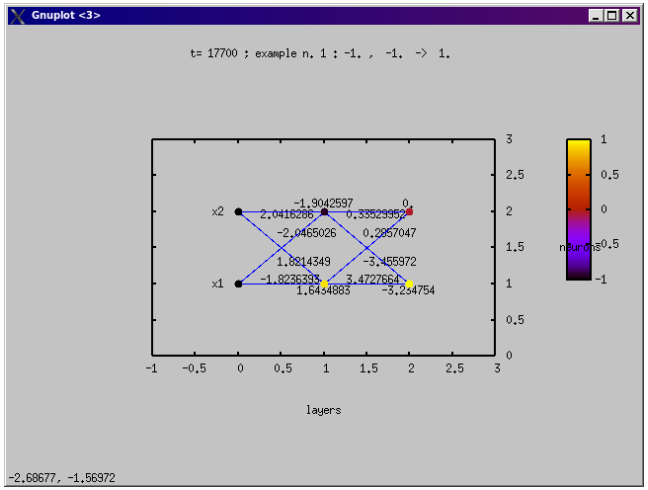
For the weights

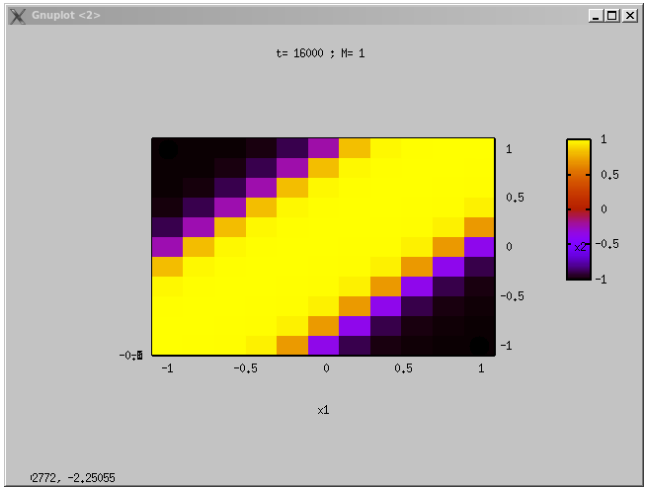
$$b'_i(\ell) = b_i(\ell) + \eta\delta_i(\ell).$$

The parameter η is the learning speed, and determines the width of a step in the direction of gradient descent.

Backpropagation implementation

- Since there are many inputs (the examples) we read them from a file (`multiPerceptronLearning.f90`, `xor.inp`, `and.inp`, etc.)
- One can check that with just an hidden layer, one can learn the xor rule.
- However, if one starts with a network with weights -1 or 1, the convergence is quite slow (insensitivity).
- Sometimes, backpropagation fails...





Unsupervised learning

- In unsupervised learning, examples are classified using the correlations among them.
- One popular implementation is the Kauhonen competitive network.
- still to be done...

Attractor neural networks

- Higher brain functions (the cortex) is better represented by a highly connected network.
- The Hopfield model is a symmetric, fully connected network.
- The Hebbian rule allows to store patterns ($\xi^{(k)}$):

$$w_{ij} = \sum_k \xi_i^{(k)} \xi_j^{(k)}$$

- still to be done...