Connectionist Computing
COMP 30230/41390

Gianluca Pollastri

office: E0.95, Science East.

email: gianluca.pollastri@ucd.ie

Credits

- **Geoffrey Hinton, University of Toronto.**
  - **borrowed some of his slides for "Neural Networks" and "Computation in Neural Networks" courses.**

- **Ronan Reilly, NUI Maynooth.**
  - **slides from his CS4018.**

- **Paolo Frasconi, University of Florence.**
  - **slides from tutorial on Machine Learning for structured domains.**

Lecture notes on Brightspace

- **Strictly confidential...**
- **Slim PDF version will be uploaded later, typically the same day as the lecture.**
- **If there is demand, I can upload onto Brightspace last year's narrated slides.. (should be very similar to this year's material)**

Books

- No book covers large fractions of this course.

- Parts of chapters 4, 6, (7), 13 of Tom Mitchell's "Machine Learning"
- Parts of chapter V of Mackay's "Information Theory, Inference, and Learning Algorithms", available online at:

http://www.inference.phy.cam.ac.uk/mackay/itprnn/book.html

- Chapter 20 of Russell and Norvig's "Artificial Intelligence: A Modern Approach", also available at:

http://aima.cs.berkeley.edu/newchap20.pdf

- More materials later..

Marking

- **3 landmark papers to read, and submit a 10-line summary on Brightspace about: each worth 6-7%**
- **a connectionist model to build and play with on some sets, write a report: 30%**
- **Final Exam in the RDS (50%)**

Hebbian learning

"When an axon of cell A is near enough to excite cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic changes takes place in one or both cells such that A's efficiency as one of the cells firing B, is increased ." **(Hebb, 1949)**

Rosenblatt's perceptron ('57)

- **A binary neuron, inputs possibly real-valued:**

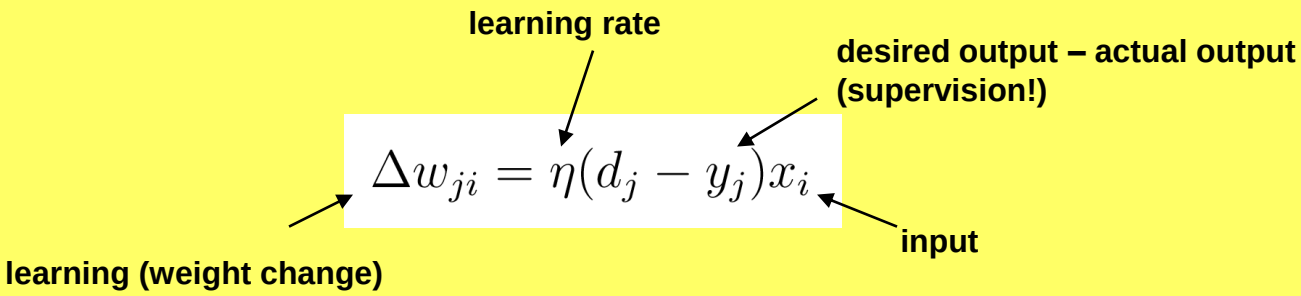
$$z_j = \sum_i w_{ji} x_i$$

$$y_j = \begin{cases} 1 & \text{if } z_j \geq 0 \\ 0 & \text{if } z_j < 0 \end{cases}$$

- **This is really just a McCulloch-Pitts neuron.. but Rosenblatt actually implemented it on a computer..**

Rosenblatt's perceptron

- **This rule assumes that we have examples**
- **Notice that, since this is a binary neuron, (d-y) can only be: -1, 0, 1**

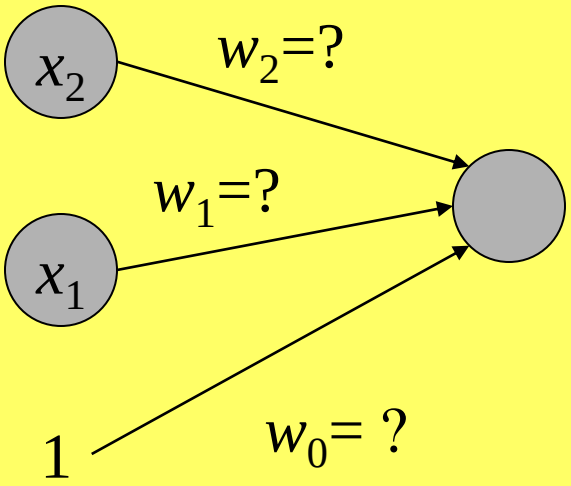


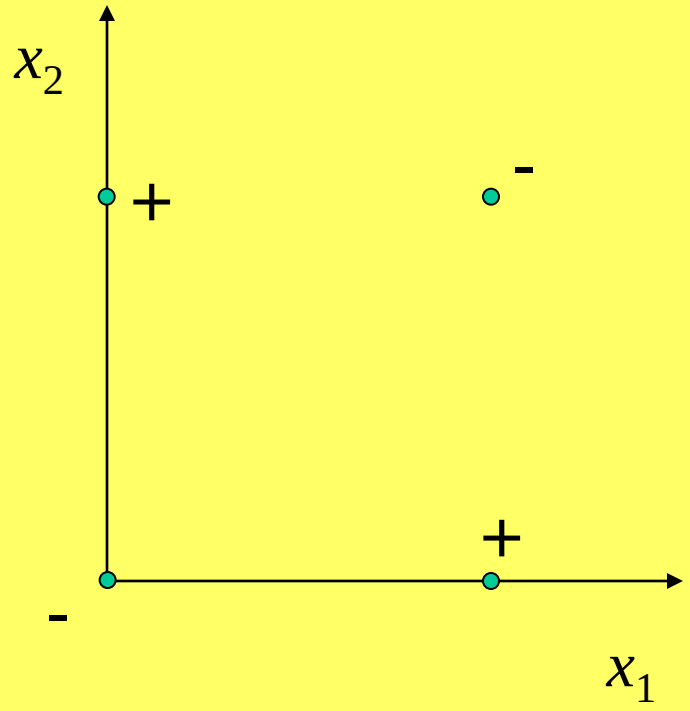
Rosenblatt's contributions

- McCulloch-Pitts neuron ('43) and Hebbian learning ('49) implemented in practice.
- Computer simulations to study the behaviour of perceptrons
- Mathematical analysis of their properties
- Rosenblatt was probably the first who talked of "connectionism".
- He studied also perceptrons with multiple layers (see next slide..)
- He called "error backpropagation" a procedure to extend the Hebbian idea to multiple layers.

Minsky & Papert: XOR (2)

x_1	x_2	$f(x)$
0	0	0
0	1	1
1	0	1
1	1	0



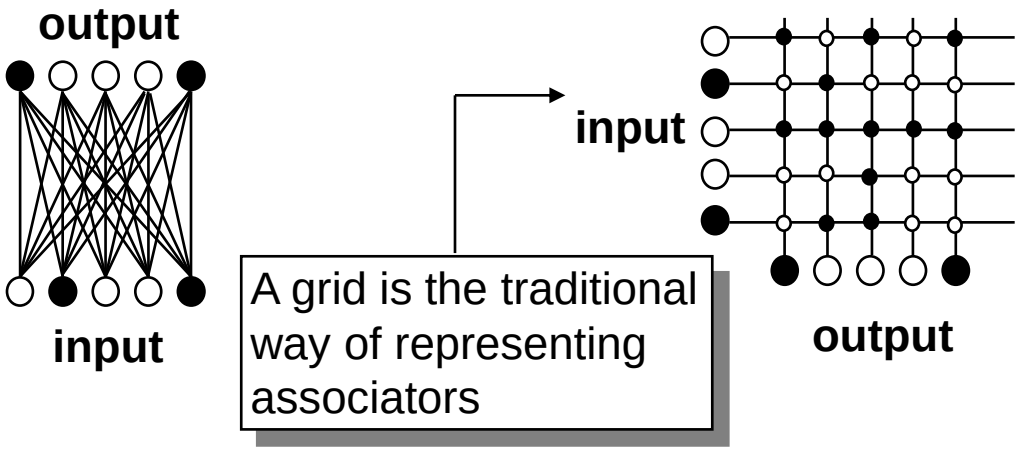


No linear separation surface exists!!

Associators (70s)

- **Linear associators**
  - **Simple generalisation of the Perceptron to networks with more than one output.**
  - **Networks with n inputs and m outputs. Nothing hidden, only 1 layer: direct connections (synapses) between any input and any output.**
  - **As in the perceptron case, any output is a combination of all the inputs.**
  - **Linear output function (was binary-threshold in the perceptron).**

Associators



Learning in associators

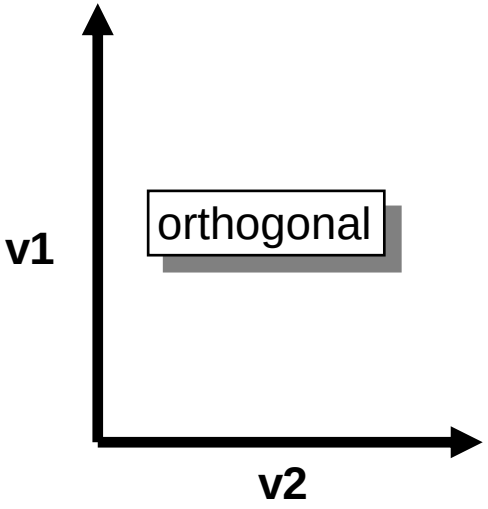
- **Learning involves a variation of Hebb's rule:**

$$\Delta w_{ji} = \eta y_j x_i$$

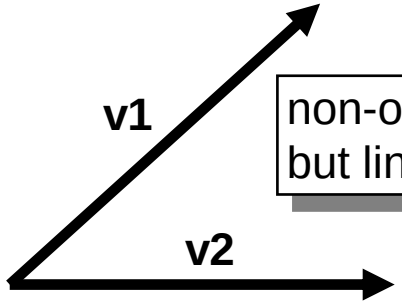
- **y_j : j-th output**
- **x_i : i-th input**

Learning in associators

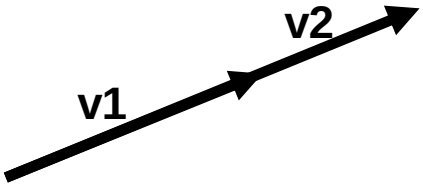
- **Under certain conditions learning in LAs can be "one-shot". After showing a set of patterns once, LAs memorise them exactly. The conditions for this to happen relate to the degree of interdependence among the input vectors.**
- **Orthogonality: This is the strongest form of independence. Two vectors are said to be orthogonal if their dot-product is 0.**
- **Linear independence: A set of vectors is linearly independent if no members of the set is a linear combination of the other set members.**
- **Linear dependence: If at least one vector in set can be written as a linear combination of the others, the set of vectors is said to be linearly dependent.**



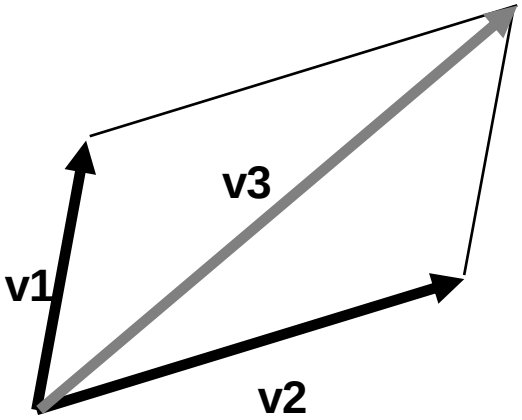
orthogonal



non-orthogonal
but linearly independent



linearly dependent:
v2 = c v1



linearly dependent:
v3 = v1+v2

Learning from a set of examples

- **We have a set of examples (input-desired output couples).**
- **For each couple compute the weight updates associated with it according to Hebb's rule, add them to the weights.**
- **Hopefully at the end of a learning cycle over all the examples the associator will output the correct values for each possible input.**

Learning in associators

- **Learning involves a variation of Hebb's rule:**

$$\Delta w_{ji} = \eta y_j x_i$$

- **y_j : j-th output**
- **x_i : i-th input**

some notation

- **The network**

$$y_j = \sum_{i=1}^{N} w_{ji} x_i$$

$$j = 1..M$$

- **Can rewrite as:**

$$\mathbf{y} = \mathbf{W} \cdot \mathbf{x}$$

- **If we define:**

$$\mathbf{x} = (x_1, .., x_n)^T$$
$$\mathbf{y} = (y_1, .., y_m)^T$$

$$\mathbf{W} = \begin{pmatrix} w_{11} & .. & w_{1i} & .. & w_{1n} \\ w_{j1} & .. & w_{ji} & .. & w_{jn} \\ w_{m1} & .. & w_{mi} & .. & w_{mn} \end{pmatrix}$$

some notation

- **Set of examples that we want to learn from:**

$$(x^{(1)}, y^{(1)}), .., (x^{(p)}, y^{(p)}), .., (x^{(P)}, y^{(P)})$$
$$x^{(p)} = (x_1^{(p)}, .., x_n^{(p)})^T$$
$$y^{(p)} = (y_1^{(p)}, .., y_m^{(p)})^T$$

- **x = inputs, y=outputs**

Weight update

- **Here is the Hebbian learning rule. We can rewrite it for the k-th input-output example using vectors:**

$$\Delta w_{ji} = \eta y_j x_i$$

$$\Delta W^{(k)} = \eta y^{(k)} \cdot x^{(k)T}$$

Weight matrix

- **At the end of a training cycle the overall weight matrix of the associator will be the sum of the updates for each example:**

$$W = \sum_k \Delta W^{(k)}$$

Estimated output

- **The estimated output for pattern p after a round of weight updates can be written as:**

$$y^{(p)'} = W x^{(p)}$$

Expanding..

$$y^{(p)'} = \sum_k \Delta W^{(k)} x^{(p)} =$$

$$\Delta W^{(p)} x^{(p)} + \sum_{k \neq p} \Delta W^{(k)} x^{(p)} =$$

$$\eta y^{(p)} \cdot x^{(p)T} \cdot x^{(p)} +$$

$$\eta \sum_{k \neq p} y^{(k)} \cdot x^{(k)T} \cdot x^{(p)} =$$

.. expanding ..

- **By an appropriate selection of η (or if the input vectors are normalised):**

$$\eta = \frac{1}{x^{(p)T} \cdot x^{(p)}}$$

$$y^{(p)\prime} = y^{(p)} + \eta \sum_{k \neq p} y^{(k)} \cdot x^{(k)T} \cdot x^{(p)}$$

.. finally

- **If the input patterns are orthogonal the associator shows perfect memory:**

$$x^{(k)T} \cdot x^{(p)} = 0$$

$$y^{(p)'} = y^{(p)}$$

- **If not orthogonal, “crosstalk”**

An example

Below is an example of some input-output pattern pairs for a 10x10 associator. The input vectors have been normalised to unit length and are almost orthogonal.

input vectors

i1	i2	i3
.00	.00	-.23
-.15	.00	-.23
-.29	-.06	-.23
.88	-.09	.14
-.29	-.30	.56
-.15	.00	.56
.00	.89	.14
.00	.00	-.23
.00	-.30	-.23
.00	-.15	-.23

required output vectors

o1	o2	o3
1.00	.00	.00
1.00	.00	.00
1.00	.00	.00
1.00	1.00	.00
.00	1.00	.00
.00	1.00	.00
.00	1.00	1.00
.00	.00	1.00
.00	.00	1.00
.00	.00	1.00
.00	.00	1.00

Resistence to noise

An important property of this simple associative memory is that it is relatively resistent to noise in the input vectors or in the memory matrix itself. If input vectors are auto-associated, the associative memory can be used to **reconstruct** the input from partial information.

actual output vectors

o1	o2	o3
1.00	.03	-.02
1.00	.03	-.02
1.00	.03	-.02
1.03	1.03	.04
.03	1.00	.06
.03	1.00	.06
.00	1.06	1.06
-.02	.06	1.00
-.02	.06	1.00
-.02	.06	1.00

crosstalk effects

output vector o3 with corrupt i3

o3 with i3[6]=0	o3 with i3(7-9)=0
.06	-.02
.06	-.02
.06	-.02
.13	-.15
.06	-.13
.06	-.13
.75	.74
.68	.87
.68	.87
.68	.87

Iterative learning procedure

- When the number of input patterns increases and the patterns themselves are not orthogonal, the "one-shot" learning approach ceases to be optimal.
- An alternative (suggested by Kohonen, 1977) is to repeatedly iterate through a set of patterns, making small weight changes, and attempting to minimise some error measure.

Minimise squared error

- **A possible error function:**

$$E = \frac{1}{2} \sum_{p=1}^{P} \sum_{k} (y_k^{(p)} - y_k^{(p)'})^2$$

Gradient descent

- **We can measure the slope of the error function with respect to the parameters, and follow the direction of steepest descent:**

$$\nabla_w E = \left(\frac{\partial E}{\partial w_{ji}} \right)$$