Competitive Learning

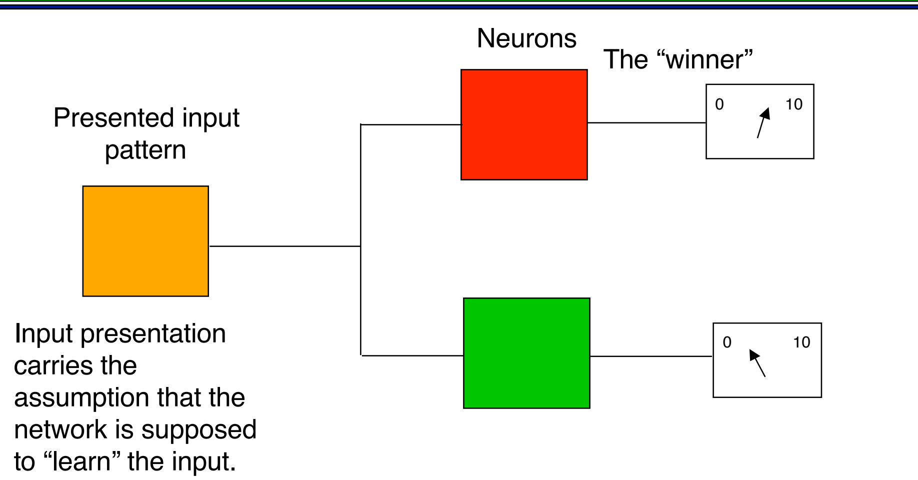
Some Types of Learning

- **Supervised learning**: training using desired response for given stimuli ("rote" learning)
- **Unsupervised learning**: classification by "clustering" of stimuli, without specified response
- **Hybrid**: e.g. unsupervised to form cluster, supervised to learn desired response to class

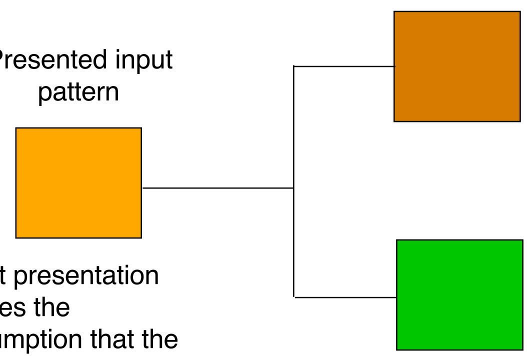
Competitive Learning

- A form of **unsupervised** learning, but **combinable with supervised** learning.
- Neurons "compete" based on proximity to input pattern.
- Neuron closest to pattern (the "**winner**") adjusts its weight to be still closer.

2-way competition



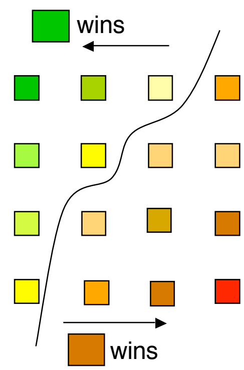
2-way competition



Why not make the winner *exactly* like the input?

- There may be many more distinct input patterns than neurons.
- By “averaging” its behavior, a neuron can put a large number of distinct, but similar inputs into the same category.

Categorizing Inputs by 2 neurons



An Application

- Display an image file with “millions of colors” on a graphic display with, say, 256 colors.
- Each color in the image has to be mapped into one of the colors.
- Map each image color into the closest one of the 256.

An Application, continued

- The actual choice of the 256 might not be fixed; it is likely a limitation of some hardware table (of RGB values) rather than a limitation of the screen itself.
- In this case, a competitive network can **learn** a reasonable set of colors to use for a given image.

A Related Application

- Use the reduction in number of colors of the image to **store** a version of the image more **compactly**
  (1M color → 256 colors reduces the number of bits by a factor of 2.5),

  or to **transmit** the version image over a slow channel.

A Competitive Neural Network

- when presented with patterns from the same selection of inputs repeatedly, will tend to **stabilize** so that its neurons are **representatives** of clusters of closer **inputs**.
- Each neuron will tend to be similar to inputs in its cluster (like a chameleon, perhaps)

Measures of similarity or closeness (opposite: distance)

- Suppose x is an input vector and w_i the weight vector of the i^{th} neuron.
- One measure of distance is the **Euclidean distance**:

$$|| x - w_i || = sqrt(sum_i((x_j - w_{ij})^2))$$
$$= sqrt((x_j - w_{ij})*(x_j - w_{ij})^T)$$

(vector inner product)

Measures of distance

- Another measure of distance, used when the values are integer, is the "**Manhattan**" or "**city-block**" distance:

$$|| x - w_i || = \text{sum}_i(|x_j - w_{ij}|)$$

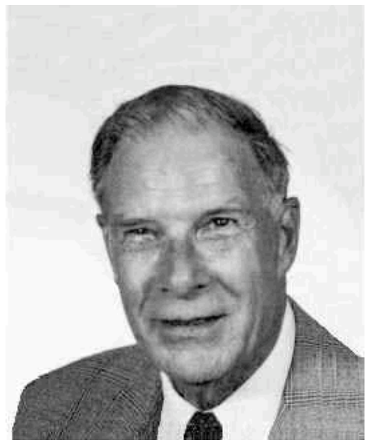
Measures of distance

- Another measure of distance, used when the values are **2-valued**, is the **"Hamming distance":**

$$\text{sum}_i(|x_j == w_{ij}|)$$

0 when the values are equal, 1 otherwise

Richard Hamming (1915-1998)



A measure of similarity is given by the **inner product**

The inner product

$$x \; w_i$$

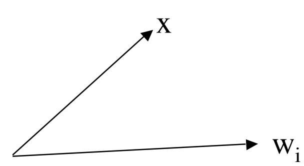
is **larger** when x is "closer to" w_i.

Usually it is best if x and w_i are **normalized** before using this measure:

$$\| x \| = \| w_i \| = 1$$

Inner product as cosine

- The normalized inner product is the *cosine* of the angle between x and w_i as *vectors*.



Example for Different Metrics

- Suppose x = [1 1 -1 1], w = [1 -1 -1 -1]
- Euclidean distance = sqrt($0^2 + 2^2 + 0^2 + 2^2$) = 2.83...
- Manhattan distance = 0 + 2 + 0 + 2 = 4
- Hamming distance = 0 + 1 + 0 + 1 = 2

smaller is closer

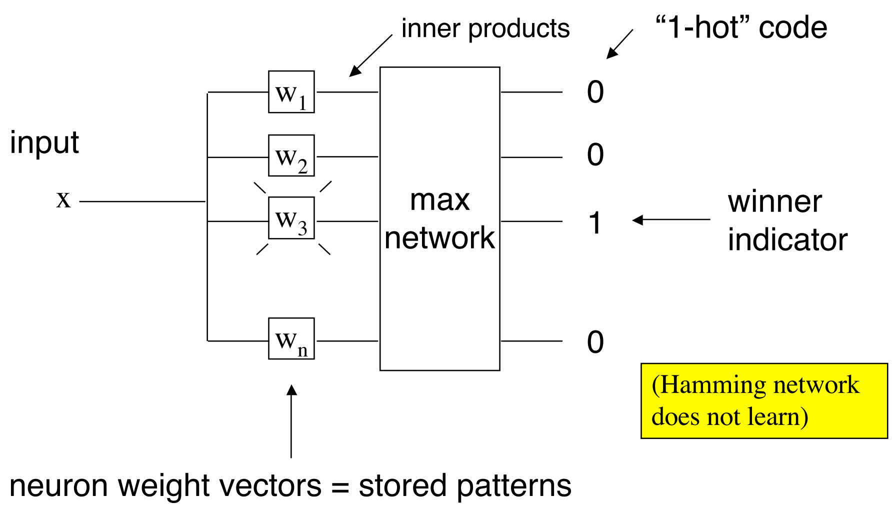
- inner product = [1 1 -1 1] $[1\ -1\ -1\ -1]^T$ = 0

larger is closer

Determining a Winner

- The winner is the neuron with weight either:
  - the smallest distance to the input, or
  - the largest inner product with the input.
- Again, if inner products are used, it is best to normalize the weight and input first, or use only normalized values.

Example: Hamming Network (competitive)



Max Sub-Network

- a recurrent neural net that cycles values through neurons, eliminating one loser each cycle until only the winner is left.

- Each neuron has as inputs the outputs of all neurons including itself.

- Self-weights are 1;
  Weights from other neurons are $-\epsilon$, where ϵ is any quantity < 1/(# of neurons).

Max Network

- Activation functions are “poslin”:

  poslin(x) = x if x > 0, 0 otherwise

- The network is operated *synchronously*.
- The initial outputs are forced to those of the input values.
- On each cycle, each neuron computes
  poslin(weighted inputs).

Max Network

- For the i^{th} neuron

$$y_i := \text{poslin}(y_i - \varepsilon \sum_{j \neq i} y_j)$$

$$= (1+\varepsilon) y_i - \varepsilon \sum y_j$$

- These weights are designed so that:
  - all but one output is non-zero after n cycles (assuming inputs were originally distinct)
  - all outputs persist at the same value after n cycles

MaxNet Example

- n = 4 neurons, take $\varepsilon = 0.2 < 1/4$

$$y_i := (1+\varepsilon) y_i - \varepsilon \sum y_j$$

step ↓

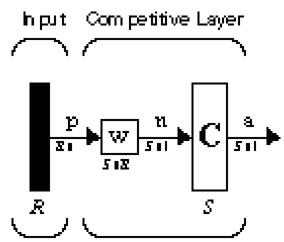
y 1	y 2	y 3	y 4	sum	epsilon
3.0000	1.0000	4.0000	2.0000	10.0000	0.2000
1.6000	0.0000	2.8000	0.4000	4.8000	
0.9600	0.0000	2.4000	0.0000	3.3600	
0.4800	0.0000	2.2080	0.0000	2.6880	
0.0384	0.0000	2.1120	0.0000	2.1504	
0.0000	0.0000	2.1043	0.0000	2.1043	
0.0000	0.0000	2.1043	0.0000	2.1043	

Matlab *compet* function (non-learning)

COMPET(N) takes one input argument,

N - SxQ matrix of net input (column) vectors.

and returns output vectors **with 1 where each net input vector has its maximum value, and 0 elsewhere**.



a = compet(n)
= compet(Wp)

compet([-3; -1; 5; 2; -9]) ==> (3,1) 1

column separators

column vector

sparse matrix notation: row 3, col 1 = 1

Using Competition in Conjunction with Learning

- Input presented
- Winner selected
- The winner learns
- Others "close to" winner may learn as well.

Instar Rule (seen before)

Instar Rule (Stephen Grossberg)

$$ {}_i\mathbf{w}(q) = {}_i\mathbf{w}(q-1) + \alpha \underbrace{a_i(q)}_{\substack{1 \text{ for i = winner} \\ 0 \text{ otherwise}}} \overbrace{(\mathbf{p}(q) - {}_i\mathbf{w}(q-1))}^{\text{pattern - weight}} $$

learning rate

Only winner learns

Kohonen Rule

(when specialized to single winner = Instar Rule)

input

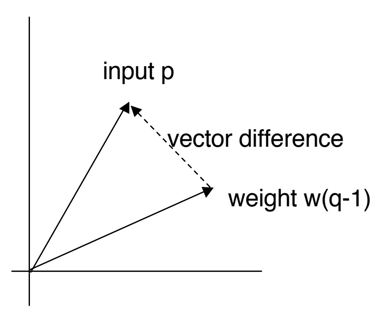
$${}_{i\square}\mathbf{w}(q) = {}_{i\square}\mathbf{w}(q-1) + \square(\mathbf{p}(q) - {}_{i\square}\mathbf{w}(q-1))$$

index of
winners

$$\boxed{\begin{array}{l} {}_{i\square}\mathbf{w}(q) = (1-\square)\,{}_{i\square}\mathbf{w}(q-1) + \square\mathbf{p}(q) \\ \\ {}_{i}\mathbf{w}(q) = {}_{i}\mathbf{w}(q-1) \qquad i \neq i\square \end{array}}$$

In the general Kohonen rule, there can be multiple “winners”.

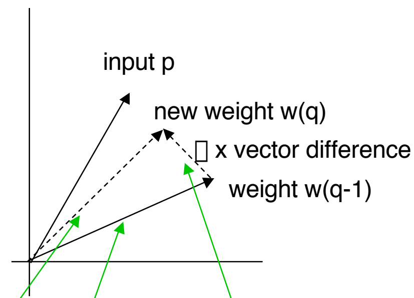
Graphical Representation



$${}_{i\square}\mathbf{w}(q) = {}_{i\square}\mathbf{w}(q-1) + \square(\mathbf{p}(q) - {}_{i\square}\mathbf{w}(q-1))$$

$${}_{i\square}\mathbf{w}(q) = (1-\square)\,{}_{i\square}\mathbf{w}(q-1) + \square\mathbf{p}(q)$$

Graphical Representation

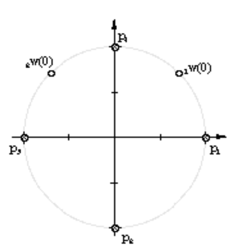


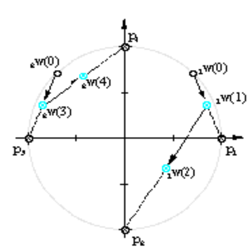
$$_{i^*}\mathbf{w}(q) = \overbrace{_{i^*}\mathbf{w}(q-1)} + \overbrace{\alpha(\mathbf{p}(q) - {}_{i^*}\mathbf{w}(q-1))}$$

$$_{i^*}\mathbf{w}(q) = (1-\alpha)\,{}_{i^*}\mathbf{w}(q-1) + \alpha\mathbf{p}(q)$$

Matlab Demos

- nnd14cl (competive learning)



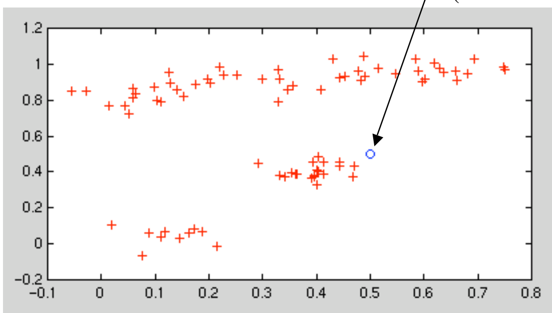


Matlab Demos

- democ1

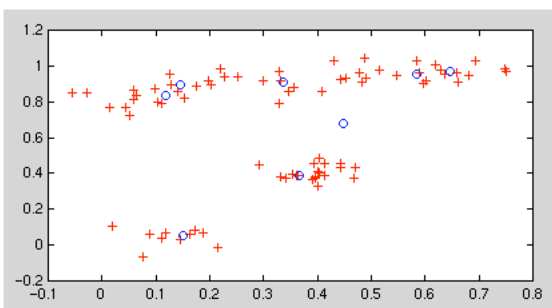
Data points (red)

2D weights of neurons
(**coinciding initially**)



democ1

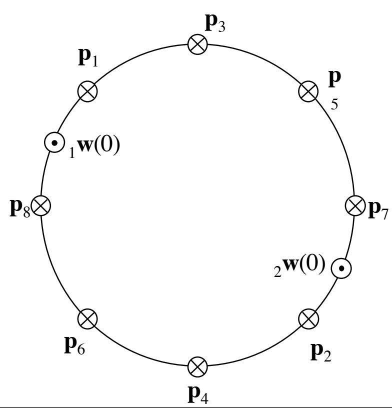
2D weights of neurons (blue)
after 500 epochs of competitive learning

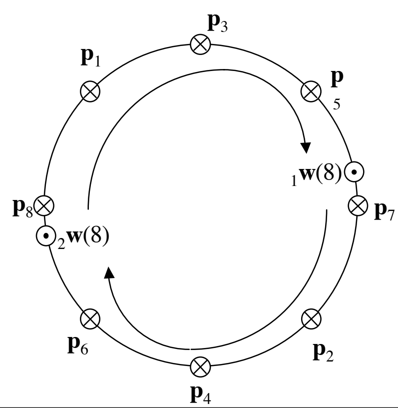


The neurons are now better *representatives* of the data.

Possible Instability

If the input vectors don't fall into nice clusters, then for large learning rates the presentation of each input vector may modify the configuration so that the system will undergo continual evolution.





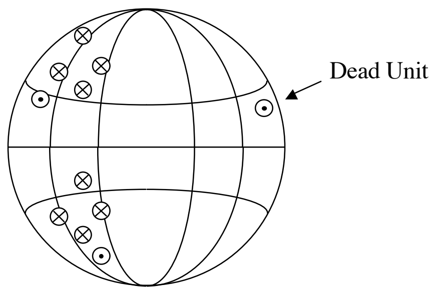
Possible Instability

If the input vectors don't fall into nice clusters, then for large learning rates the presentation of each input vector may modify the configuration so that the system will undergo continual evolution.

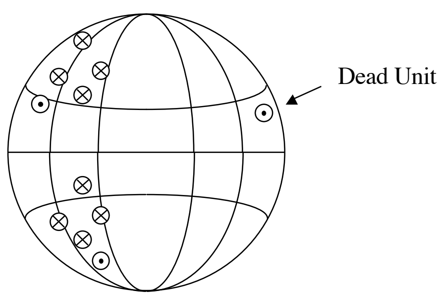
Solution: Gradually **decrease the learning rate ("annealing")**.

"Dead" Units / Starvation

One problem with competitive learning is that neurons with initial weights far from any input vector **may never win** and thus become **useless**.



Have a Heart



Solution: Add a negative bias to each neuron, and increase the magnitude of the bias as the neuron wins.
This will make it harder to win if a neuron has won often.
This is called the "**conscience**" method.