Competitive Learning

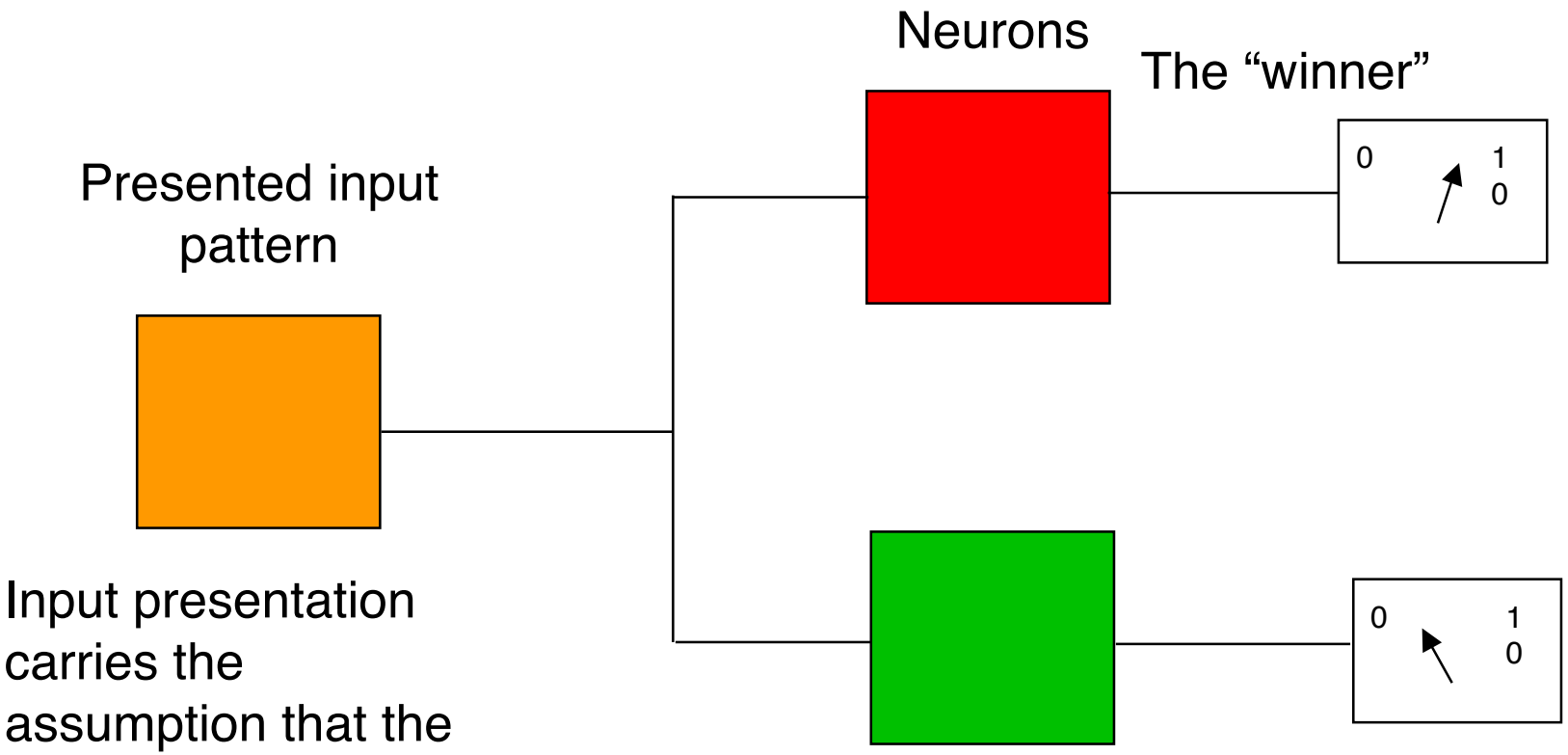
Some Types of Learning

- **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

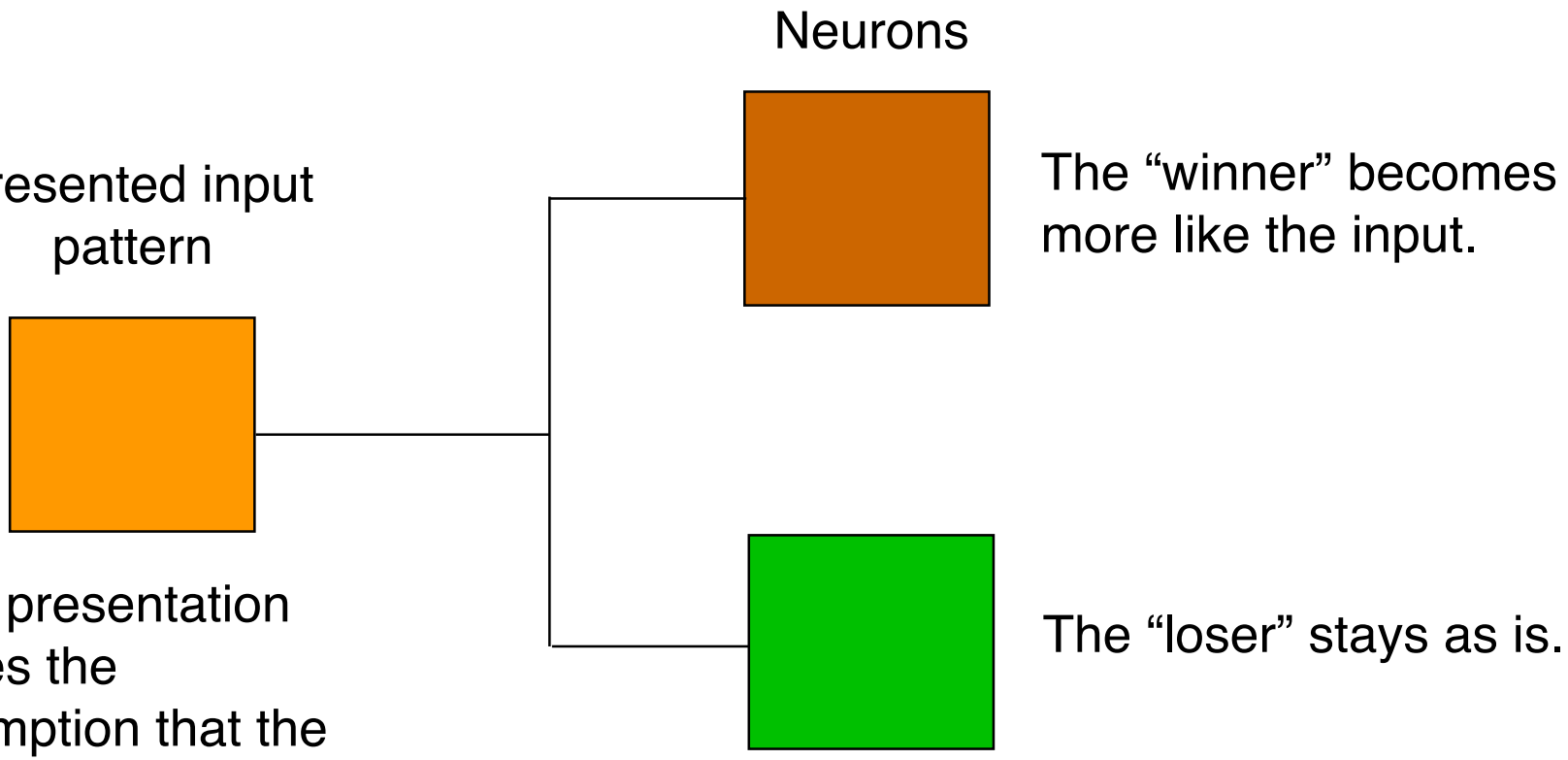
Competitive Learning

- 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



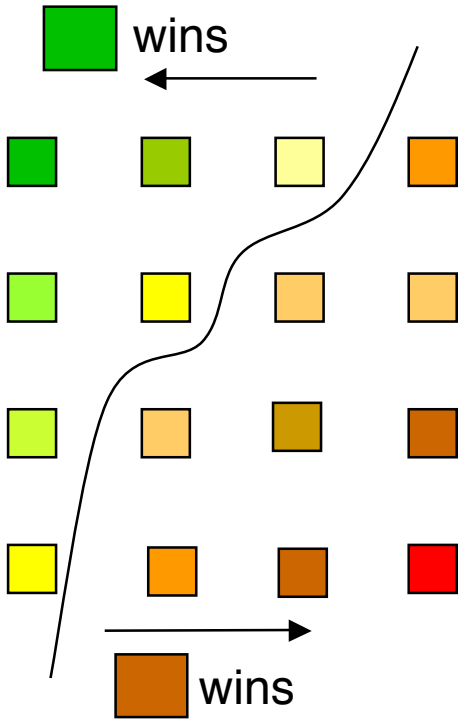
2-way competition



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

- Displaying an image file with “millions of colors” using a palate of, say, 256 colors.
- Each color in the image has to be mapped into one of the colors in the palate.
- Map each image color into the closest one of the 256 palate colors.
- In this case, a **competitive network** can **learn** a reasonable set of colors to use for a given image.

Related Applications

- 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 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: dissimilarity or 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**:

$$\| x - w_i \| = \text{sqrt}(\text{sum}_i((x_j - w_{ij})^2))$$

$$= \text{sqrt}((x_j - w_{ij})*(x_j - w_{ij})^T)$$

(vector inner product)

Measures of distance

- The discrete metric:

$$d(x, w_i) = 0 \text{ if } x = w_i, 1 \text{ otherwise}$$

Measures of distance

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

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

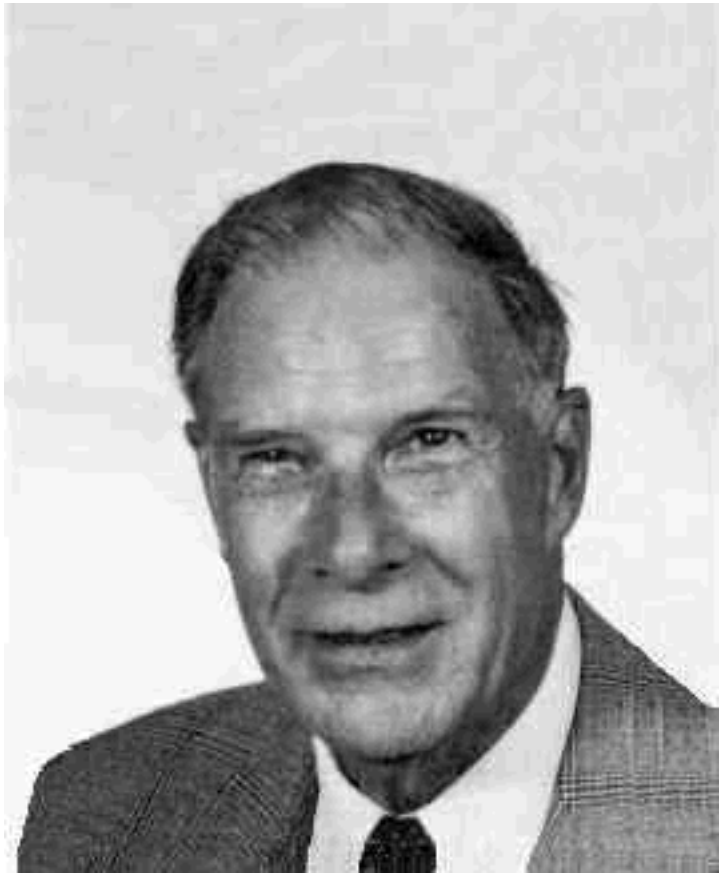
Measures of distance

- 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)



Properties of Distance (i.e. a Metric)

- $d: P \times P \rightarrow R$
  where P is the space of patterns and
  R is the set of real numbers

- $(\forall x, y)\ d(x, y) \geq 0$

- $(\forall x)\ d(x, y) = 0$ iff $x = y$

- $(\forall x, y)\ d(x, y) = d(y, x)$

- $(\forall x, y, z)\ d(x, z) \leq d(x, y) + d(y, z)$

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...
- Discrete metric = 1
- Manhattan distance = 0 + 2 + 0 + 2 = 4
- Hamming distance = 0 + 1 + 0 + 1 = 2

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.

Therefore inner product is **not a metric**.

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

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

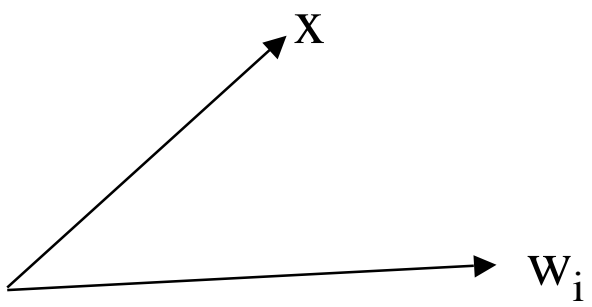
In this case, $(1 - xw_i)/2n$ is a distance, where n is the dimension..

Inner Product is not a Metric

- [1 1 -1 1] * [1 1 -1 1] = 4
- [1 1 -1 1] * [-1 -1 1 -1] = -4
- [1 1 -1 1] * [1 -1 -1 -1] = 0

Inner product as cosine

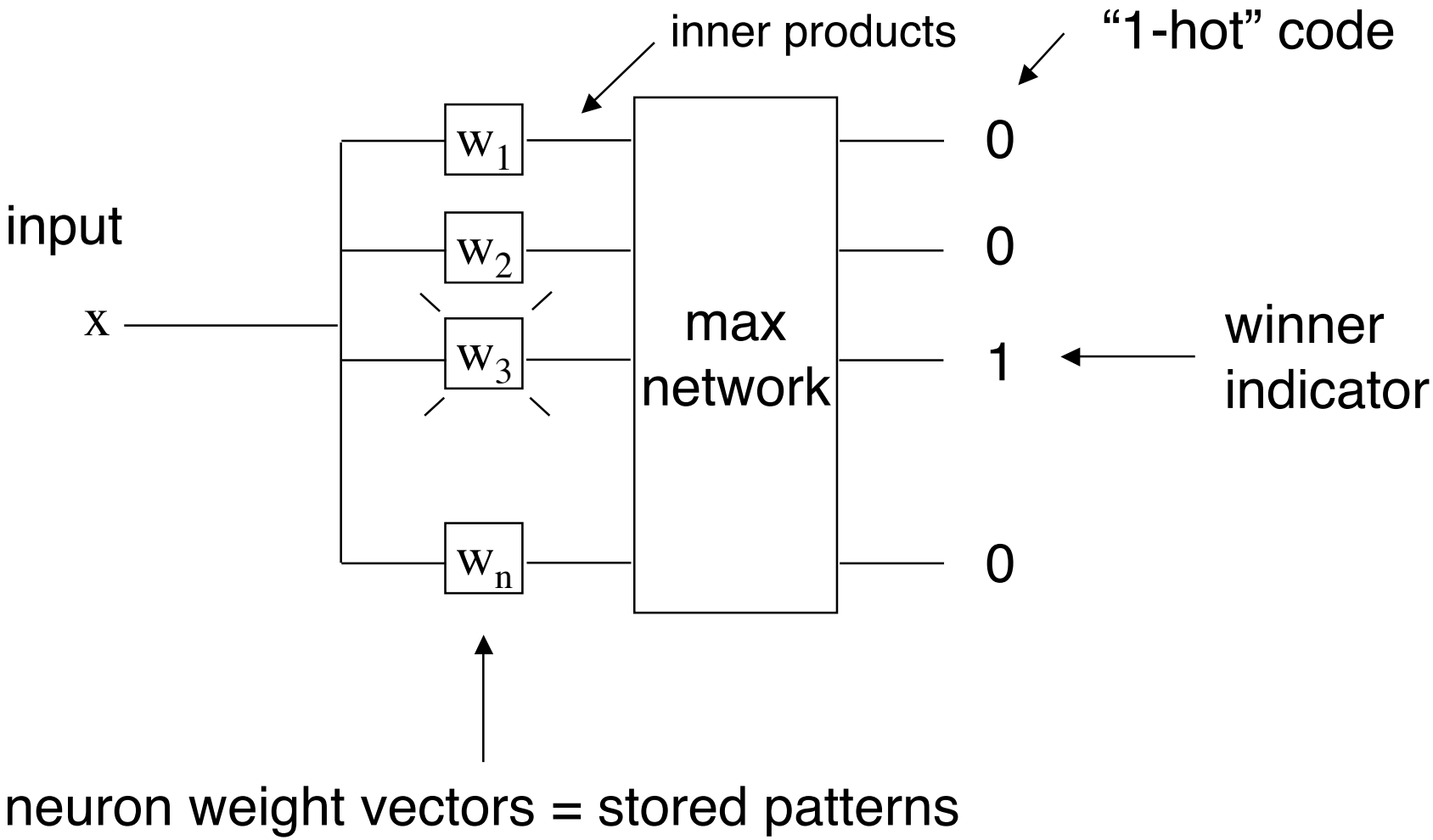
- When the pattern space consists of vectors of numbers, the normalized inner product is the *cosine* of the angle between x and w_i as *vectors*.



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, non-learning)



"Neural" Implementation of a Max Sub-Network

- a recurrent neural net that cycles values through neurons, eliminating one loser each cycle until only the winner is left.
- (This is totally unnecessary from a strictly computational viewpoint.)
- Each neuron has as inputs the outputs of all neurons including itself.
- Self-weights are 1;
  Weights from other neurons are $-\varepsilon$, 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 $\varepsilon = 0.2 < 1/4$

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

step

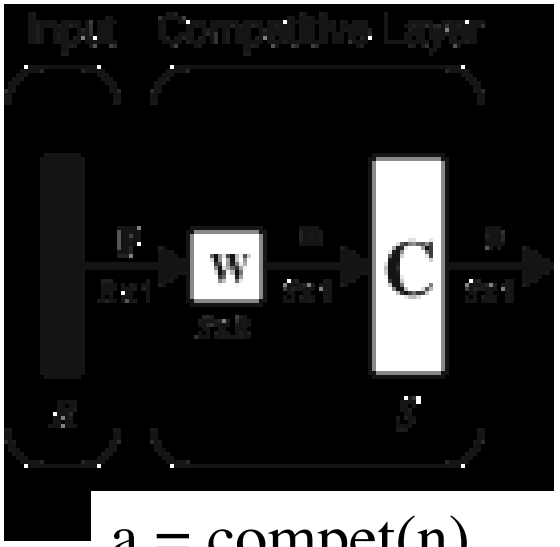
y 1	y 2	y 3	y 4	sum	epsilon
3.0000	1.0000	4.0000	2.0000	10.0000	0.2000
1.6000	0.0000	2.8000	0.4000	4.8000	
0.9600	0.0000	2.4000	0.0000	3.3600	
0.4800	0.0000	2.2080	0.0000	2.6880	
0.0384	0.0000	2.1120	0.0000	2.1504	
0.0000	0.0000	2.1043	0.0000	2.1043	
0.0000	0.0000	2.1043	0.0000	2.1043	

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

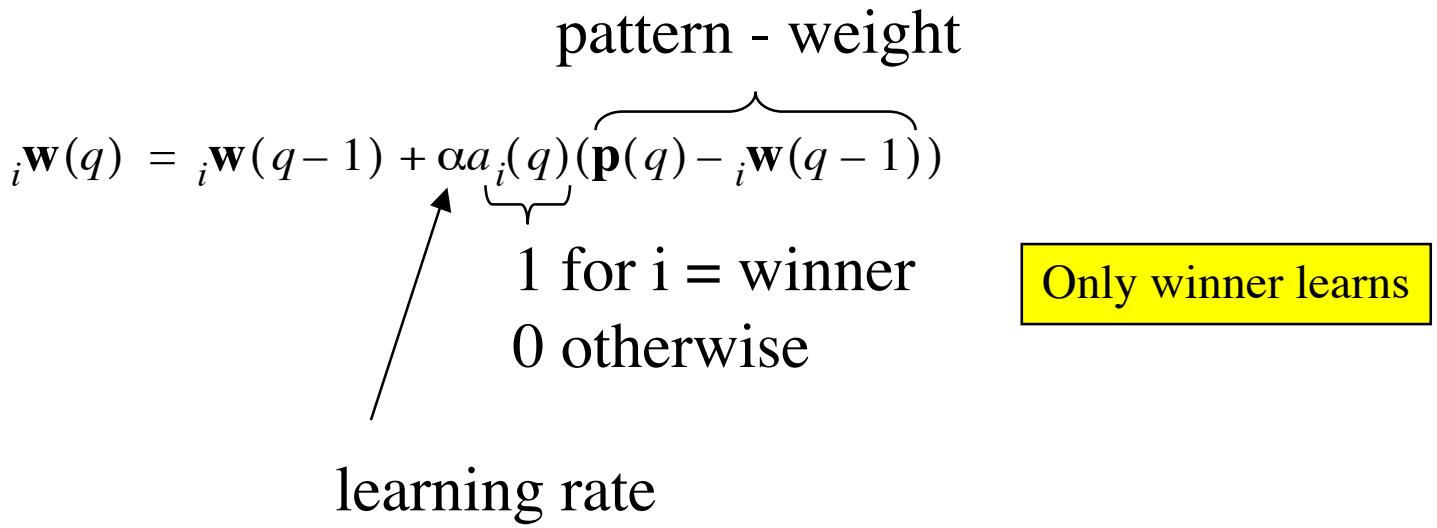
Using Competition in Conjunction with Learning

- Input presented
- Winner selected
- The winner learns

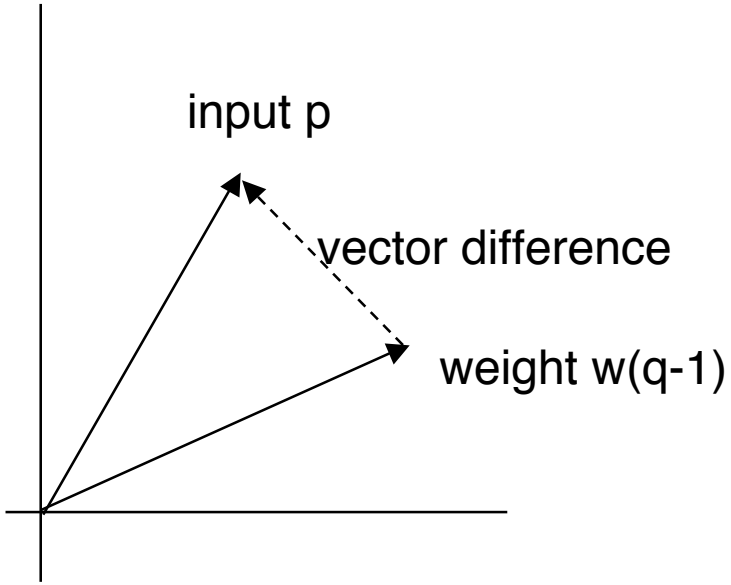
- Others “close to” winner may learn as well.

Instar Rule (seen before)

Instar Rule (Stephen Grossberg)



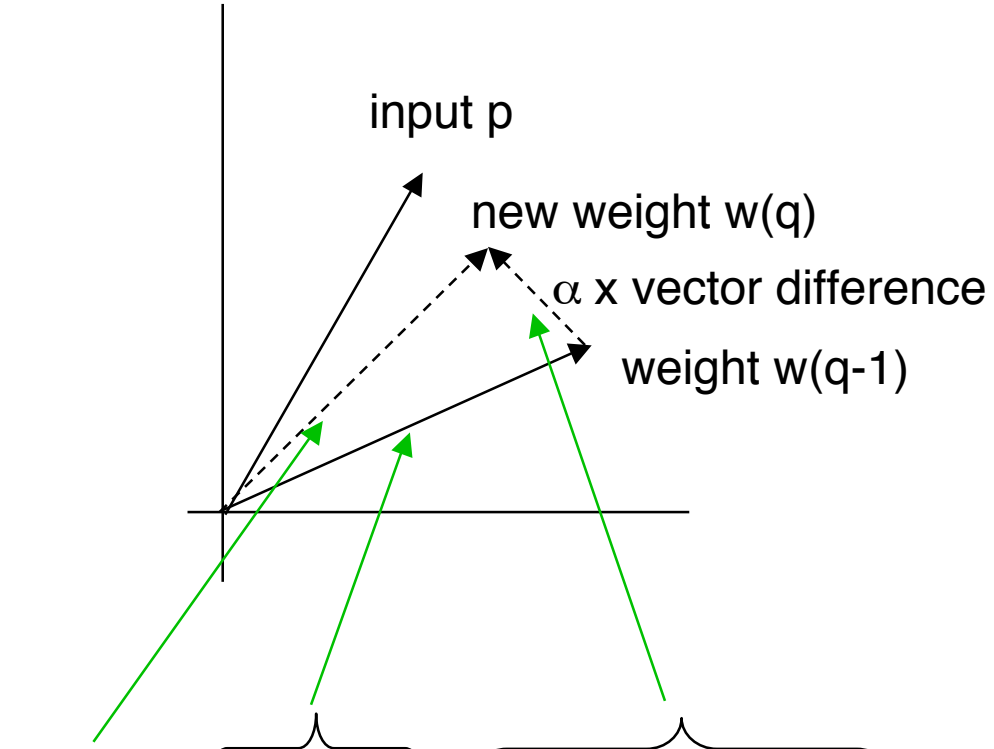
Graphical Representation



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

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

Graphical Representation

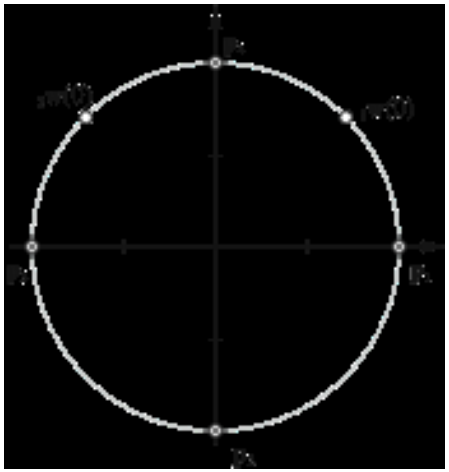


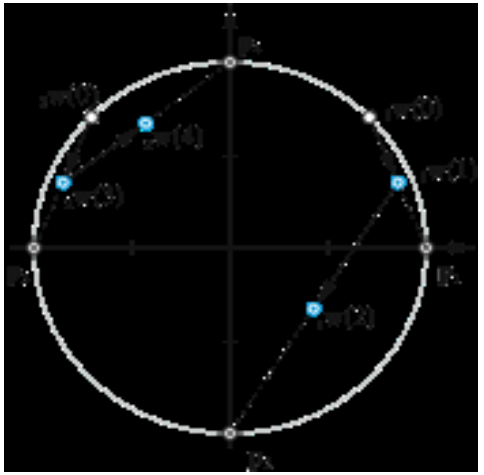
$$ {}_{i^*}\mathbf{w}(q) = {}_{i^*}\mathbf{w}(q-1) + \alpha(\mathbf{p}(q) - {}_{i^*}\mathbf{w}(q-1)) $$

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

Matlab Demos

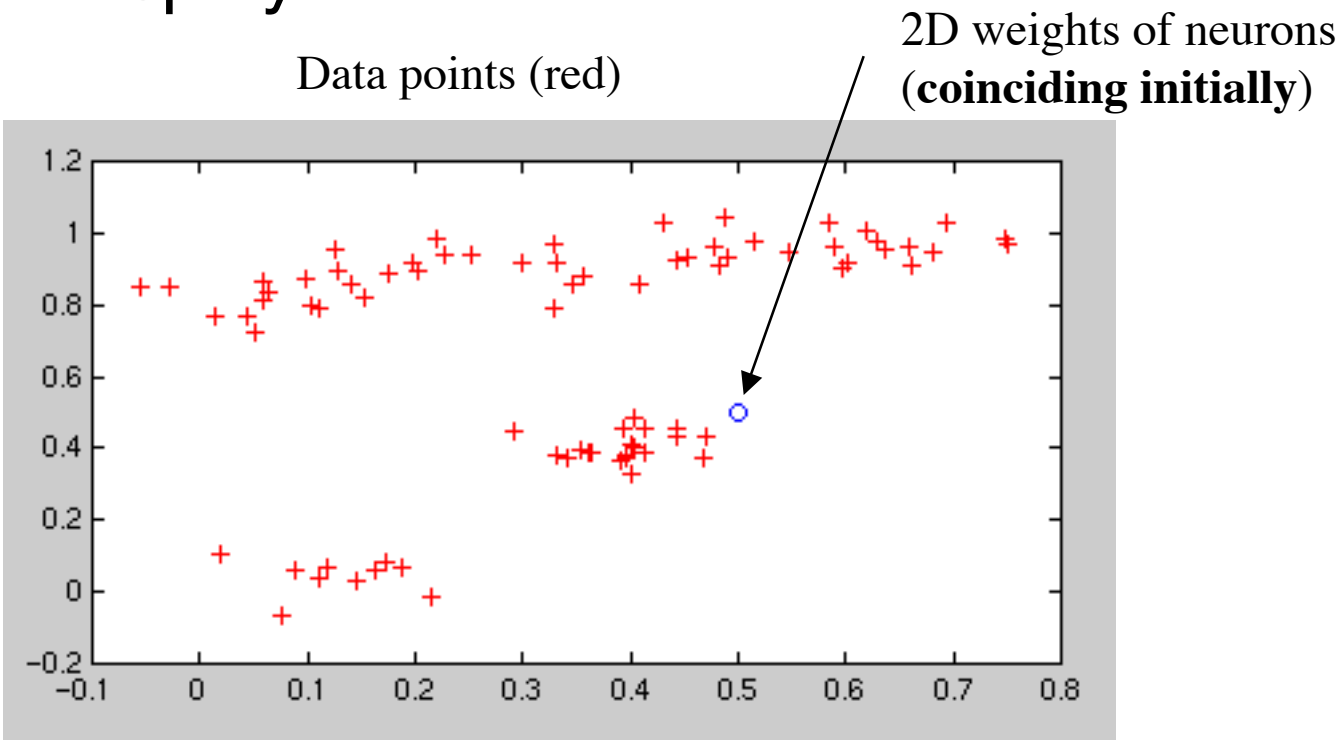
- nnd14cl (competitive learning)
- A vector input is chosen on the circumference of a circle.
- A competition is held among the neurons.
- The closest neuron learns by moving closer to the input.
- Other neurons stay put.





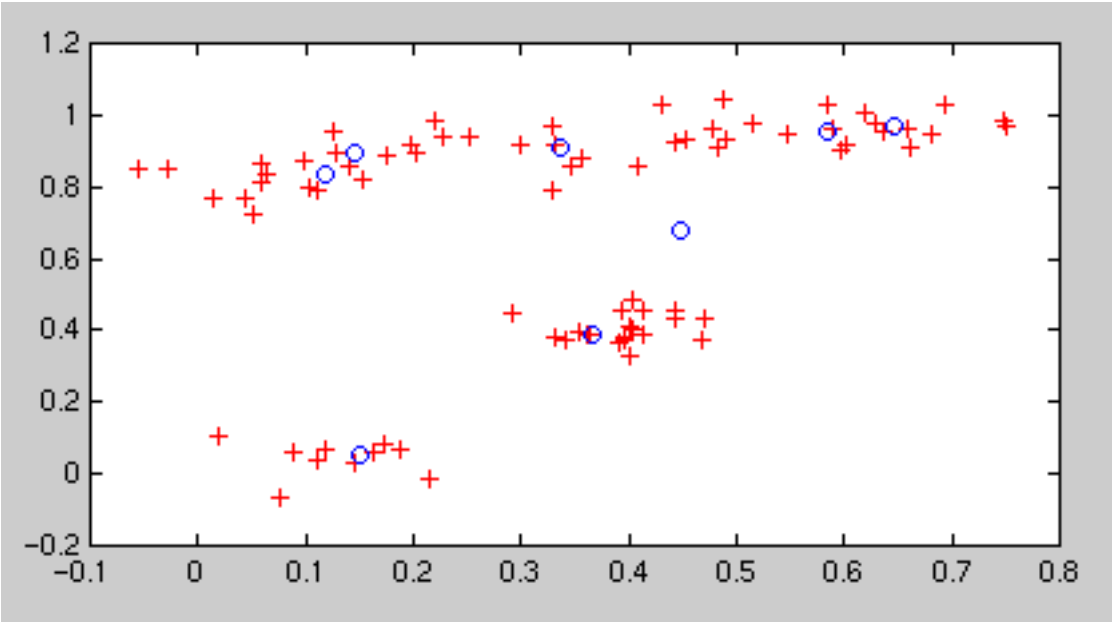
Matlab Demos

- democ1
- initial display:



democ1

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



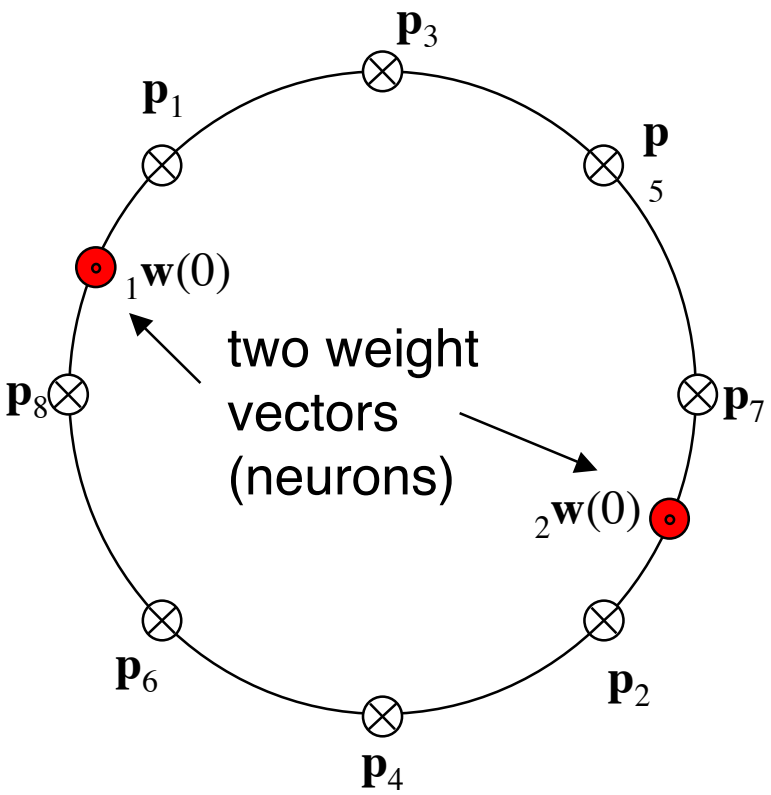
The neurons are now better *representatives* of the data.

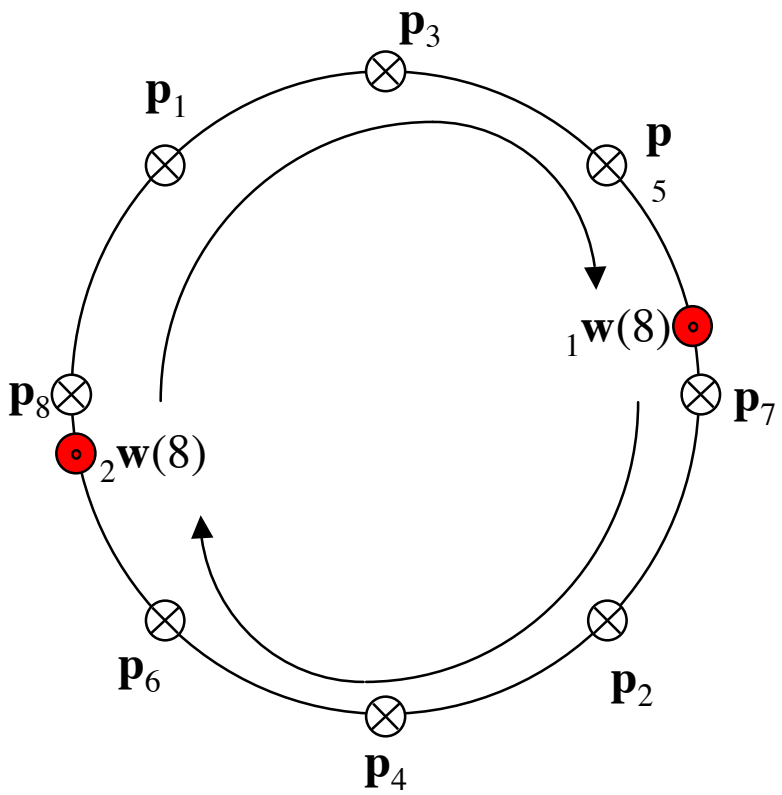
Competitive Learning vs. Clustering

- The preceding example shows that competitive learning is one way to achieve clustering.
- We discussed the **k-means clustering** algorithm earlier for example.
- In k-means, the computation of new centers can be viewed as a kind of competition, since the new centers are computed by summing the inputs **closest** to a given old center.

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.





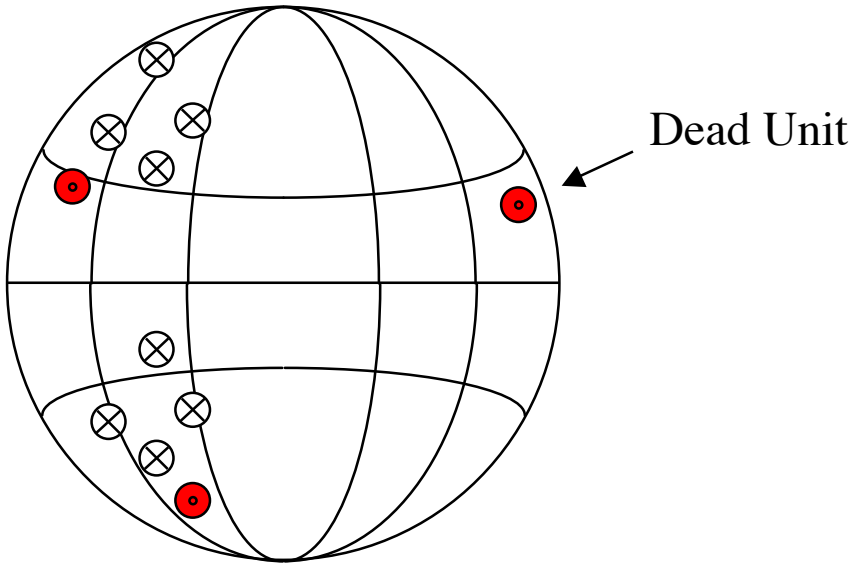
Possible Instability

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.

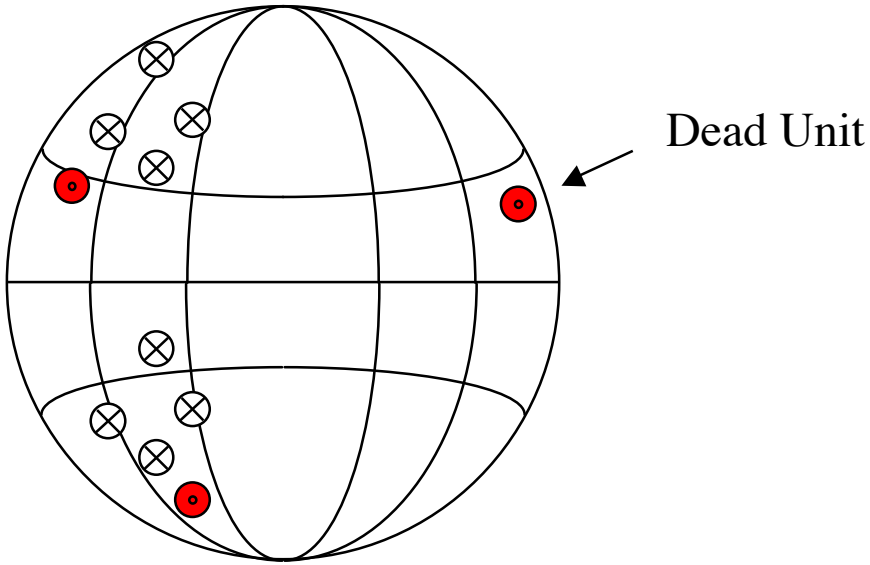
Solution: Gradually **decrease the learning rate ("annealing")**.

“Dead” Units / Starvation

Another 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 for a neuron to win if it has won often.

This is called the "**conscience**" method.

Competitive Learning with Explicit Output Classification

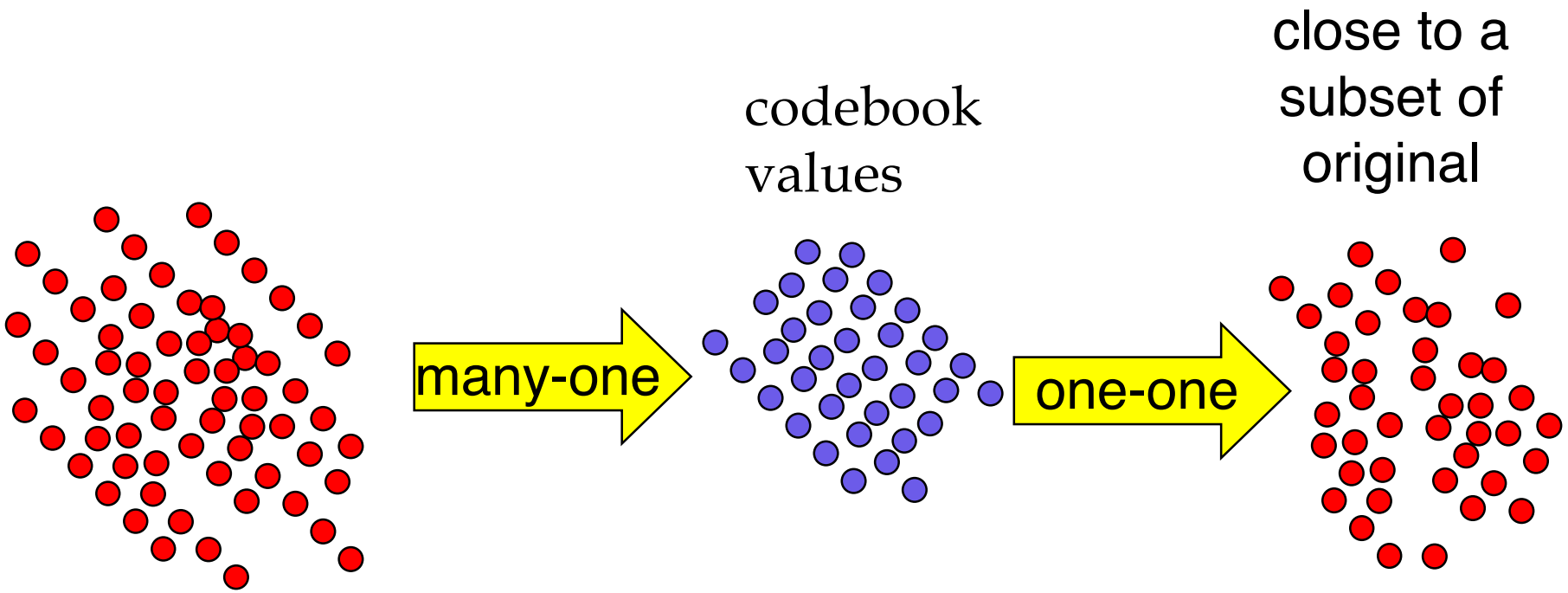
Learning Vector-Quantization

Counterpropagation

Vector Quantization

- VQ is a technique used to **reduce the dimensionality of data**.
- The original data is a set of vectors, of dimension n, say.
- The vectors are mapped into a set of smaller dimension, m, of **codebook values**.
- The codebook values are used for storage or transmission (some information content is lost).
- The codebook values, upon receipt or retrieval, are re-translated into values close to the original data values (will not be exact).

Codebook

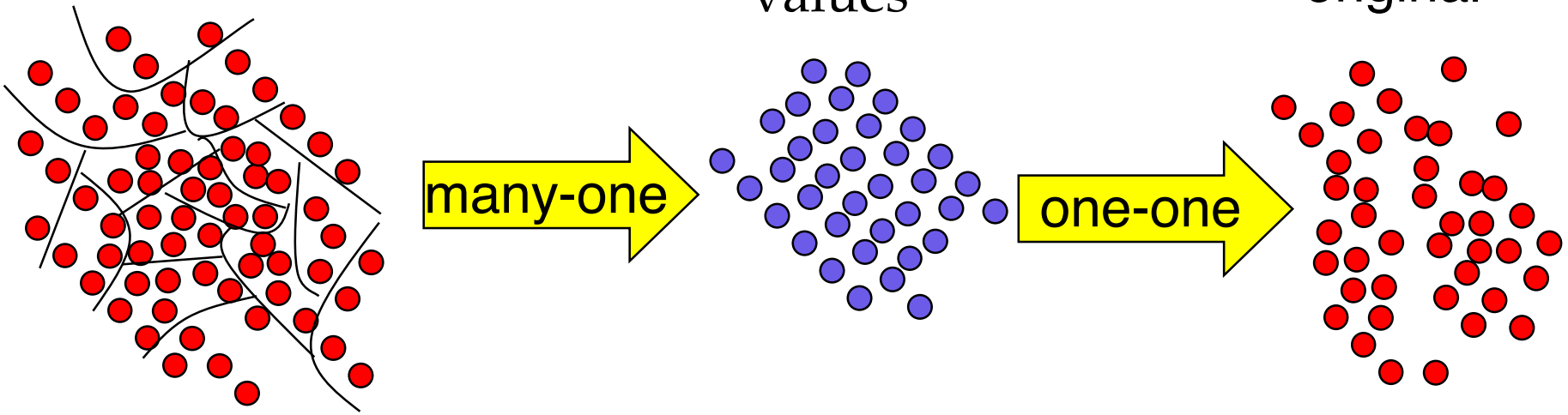

codebook
values
close to a
subset of
original
many-one
one-one

Codebook

inverse-image = some kind of clustering

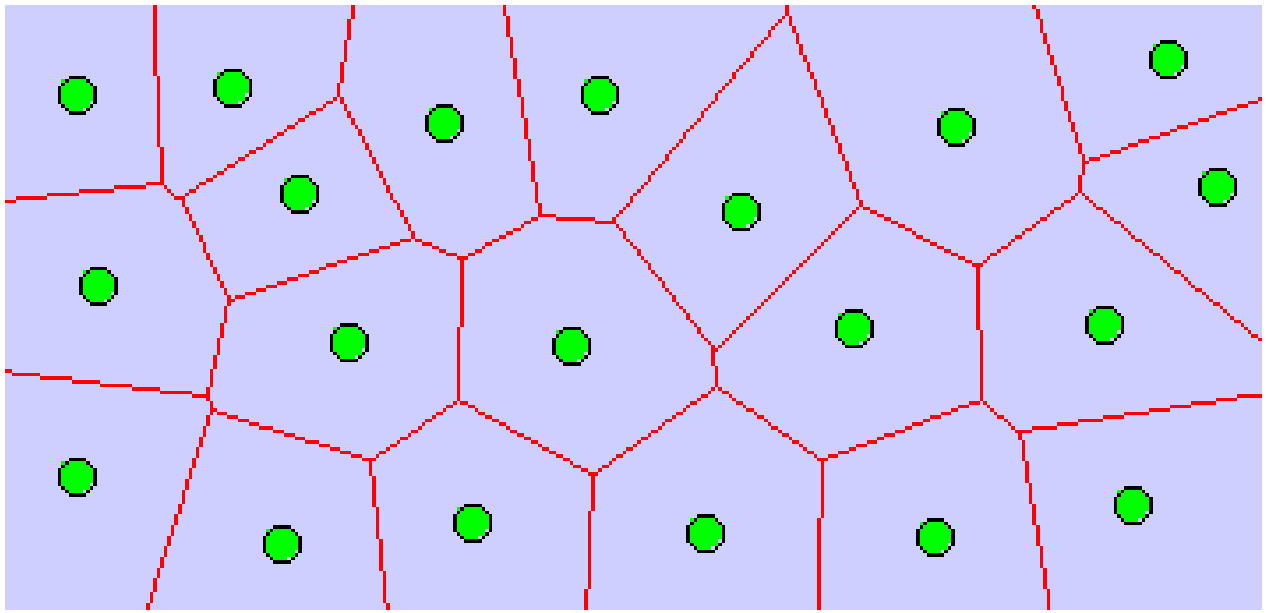
codebook values

close to a subset of original



Voronoi Regions

Typically codebook code representatives are centers of ***Voronoi Regions*** in the data space. Voronoi regions are sets of points **closest** to one representative vs. another. Below each polygon is a Voronoi region, with representatives as dots.



closest ⇒ error is minimized

Chicken vs. Egg

- Which comes first, the Voronoi regions or the representatives?
- Initially we just have a set of points with neither.
- We specify the maximum size of the codebook (assume the number of points is larger).
- Then use a clustering technique, such as **k-means clustering**.

k-means clustering in more detail

- Choose an **initial** set of representatives by some method.
- Iterate the following:
  - For each point, find its closest representative; this determines a set of Voronoi regions.
  - Compute the *centroids* of the regions and use them as the new representatives.
- Until the desired error bound is reached.
- Note: The representatives need not be actual data points.

Termination Condition

- Compute MSE = $\frac{\Sigma\{\text{distance-from-rep}(p)^2 \mid p \text{ in region}\}}{\#\text{regions}}$

  where the sum is over all regions.

- We want MSE to be below a certain bound.

Computational Cost

- $O(n^2)$ per iteration, where n is the number of points.

Other Clustering Methods

- There are many.
- For a survey, see this tutorial: http://www.pitt.edu/~csna/reports/janowitz.pdf which was listed on the website of the:

  Classification Society of North America

- (k-means clustering is mentioned on page 30).

Learning Vector-Quantization (LVQ)

Learning Vector Quantization (LVQ)

- Combine **competitive** learning with **supervision**.
- Competitive learning achieves clusters.
- Assign a **class** (or output value) to each cluster.
- Reinforce cluster representative (a neuron) when it **classifies** input in the **desired** class:
  - **Positive reinforcement**: pull the neuron weights toward the input.
  - **Negative reinforcement**: push the weights away.

Learning Vector Quantization

Training examples:

a is actual, t is target

$$\{\mathbf{p}_1, \mathbf{t}_1\}, \{\mathbf{p}_2, \mathbf{t}_2\}, \ldots, \{\mathbf{p}_Q, \mathbf{t}_Q\}$$

If the input pattern is classified ***correctly***, then move the *winning* weight ***toward*** the input vector according to the rule:

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

$$a_{k^*}^2 = t_{k^*} = 1$$

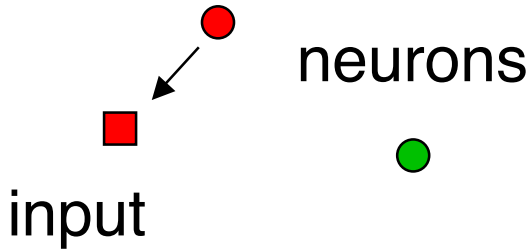
If the input pattern is classified ***incorrectly***, then move the *winning* weight ***away*** from the input vector:

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

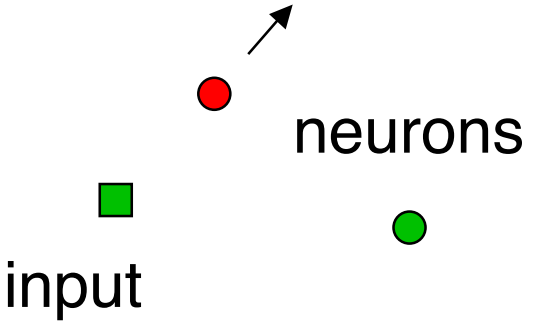
$$a_{k^*}^2 = 1 \neq t_{k^*} = 0$$

LVQ Rules

Input pattern
correctly classified



Input pattern
incorrectly classified



Learning Vector Quantization: Second Layer

- The classification mapping from selection of cluster representative to output can be accomplished by a second layer, following the competitive layer.

- This second layer is essentially a matrix multiply, so can be represented by neural weights as usual. These weights may be pre-assigned and fixed.

LVQ xor Example

Example patterns with targets:

$$\left\{\mathbf{p}_1 = \begin{bmatrix}0\\1\end{bmatrix}, \mathbf{t}_1 = \begin{bmatrix}1\\0\end{bmatrix}\right\} \quad \left\{\mathbf{p}_2 = \begin{bmatrix}1\\0\end{bmatrix}, \mathbf{t}_2 = \begin{bmatrix}1\\0\end{bmatrix}\right\} \quad \left\{\mathbf{p}_3 = \begin{bmatrix}1\\1\end{bmatrix}, \mathbf{t}_3 = \begin{bmatrix}0\\1\end{bmatrix}\right\} \quad \left\{\mathbf{p}_4 = \begin{bmatrix}0\\0\end{bmatrix}, \mathbf{t}_4 = \begin{bmatrix}0\\1\end{bmatrix}\right\}$$

Competitive layer weights:

$$\mathbf{W}^1(0) = \begin{bmatrix}({}_1\mathbf{w}^1)^T\\({}_2\mathbf{w}^1)^T\\({}_3\mathbf{w}^1)^T\\({}_4\mathbf{w}^1)^T\end{bmatrix} = \begin{bmatrix}0.25 & 0.75\\0.75 & 0.75\\1 & 0.25\\0.5 & 0.25\end{bmatrix}$$

Classification layer weight (fixed):

$$\mathbf{W}^2 = \begin{bmatrix}1 & 1 & 0 & 0\\0 & 0 & 1 & 1\end{bmatrix}$$

Iteration 1, first layer

$$\mathbf{a}^1 = \mathbf{compet}(\mathbf{n}^1) = \mathbf{compet}\left(\begin{bmatrix} -\|_1\mathbf{w}^1 - \mathbf{p}_1\| \\ -\|_2\mathbf{w}^1 - \mathbf{p}_1\| \\ -\|_3\mathbf{w}^1 - \mathbf{p}_1\| \\ -\|_4\mathbf{w}^1 - \mathbf{p}_1\| \end{bmatrix}\right)$$

$$\mathbf{a}^1 = \mathbf{compet}\left(\begin{bmatrix} -\left\|\begin{bmatrix}0.25 & 0.75\end{bmatrix}^T - \begin{bmatrix}0 & 1\end{bmatrix}^T\right\| \\ -\left\|\begin{bmatrix}0.75 & 0.75\end{bmatrix}^T - \begin{bmatrix}0 & 1\end{bmatrix}^T\right\| \\ -\left\|\begin{bmatrix}1.00 & 0.25\end{bmatrix}^T - \begin{bmatrix}0 & 1\end{bmatrix}^T\right\| \\ -\left\|\begin{bmatrix}0.50 & 0.25\end{bmatrix}^T - \begin{bmatrix}0 & 1\end{bmatrix}^T\right\| \end{bmatrix}\right) = \mathbf{compet}\left(\begin{bmatrix} -0.354 \\ -0.791 \\ -1.25 \\ -0.901 \end{bmatrix}\right) = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Iteration 1, second layer

$$\mathbf{a}^2 = \mathbf{W}^2\mathbf{a}^1 = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

This is the **correct** class, therefore the weight vector is moved **toward** the input vector.

$$_1\mathbf{w}^1(1) = {}_1\mathbf{w}^1(0) + \alpha(\mathbf{p}_1 - {}_1\mathbf{w}^1(0))$$

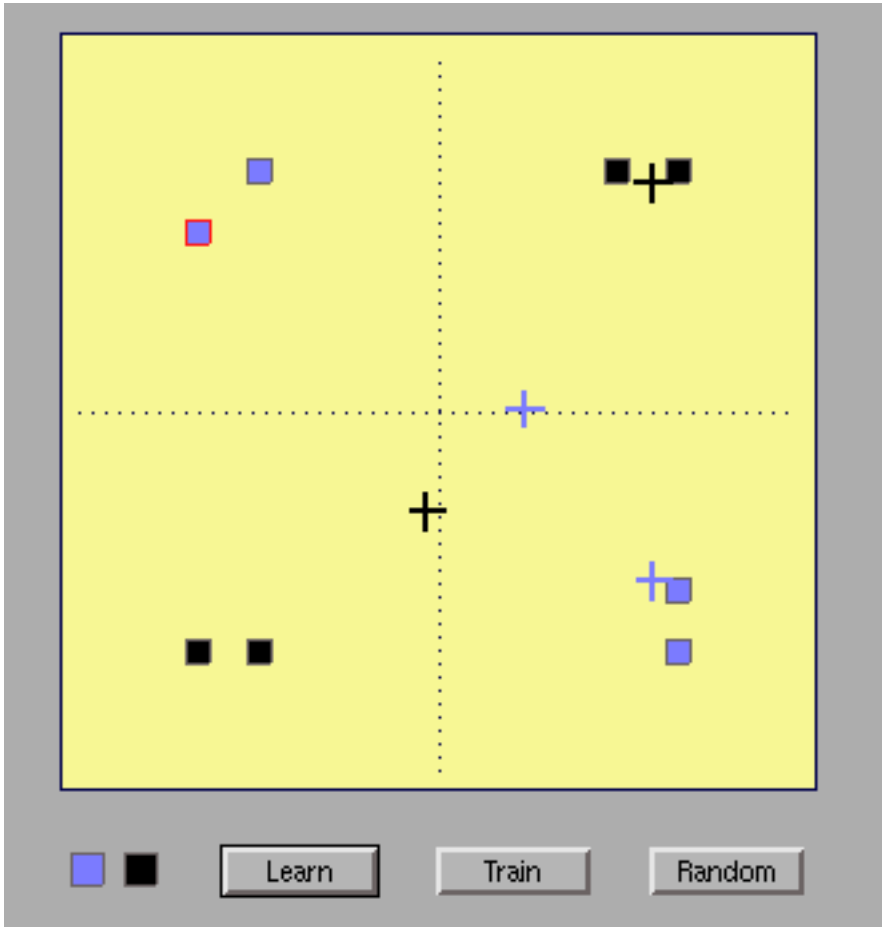
$$_1\mathbf{w}^1(1) = \begin{bmatrix} 0.25 \\ 0.75 \end{bmatrix} + 0.5\left(\begin{bmatrix} 0 \\ 1 \end{bmatrix} - \begin{bmatrix} 0.25 \\ 0.75 \end{bmatrix}\right) = \begin{bmatrix} 0.125 \\ 0.875 \end{bmatrix}$$

Matlab LVQ demos

- nnd14lv1, nnd14lv2

Blue vs. black represent class of neuron, desired class of input sample.

Selected input sample changes to green if correctly classified, red if not; weights adjusted.



Variant: LVQ2

- If the winning neuron in the hidden layer **correctly** classifies the current input, the response is the same as in LVQ1.

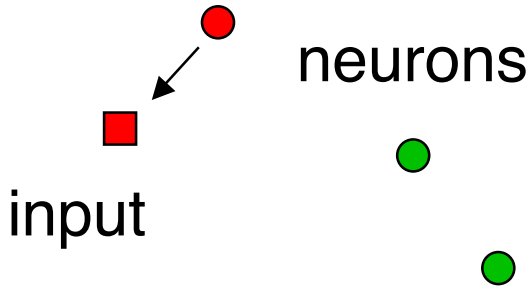
- If the winning neuron in the hidden layer **incorrectly** classifies the current input, we move its weight vector **away** from the input vector, as in LVQ,

  and

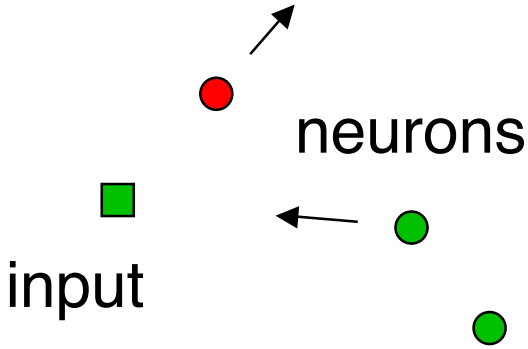
- move the weights of the **closest** neuron to the input vector that **would have correctly classified** the input **toward** the input vector.

LVQ2 Rules

Input pattern
correctly classified



Input pattern
incorrectly classified



Counterpropagation Networks

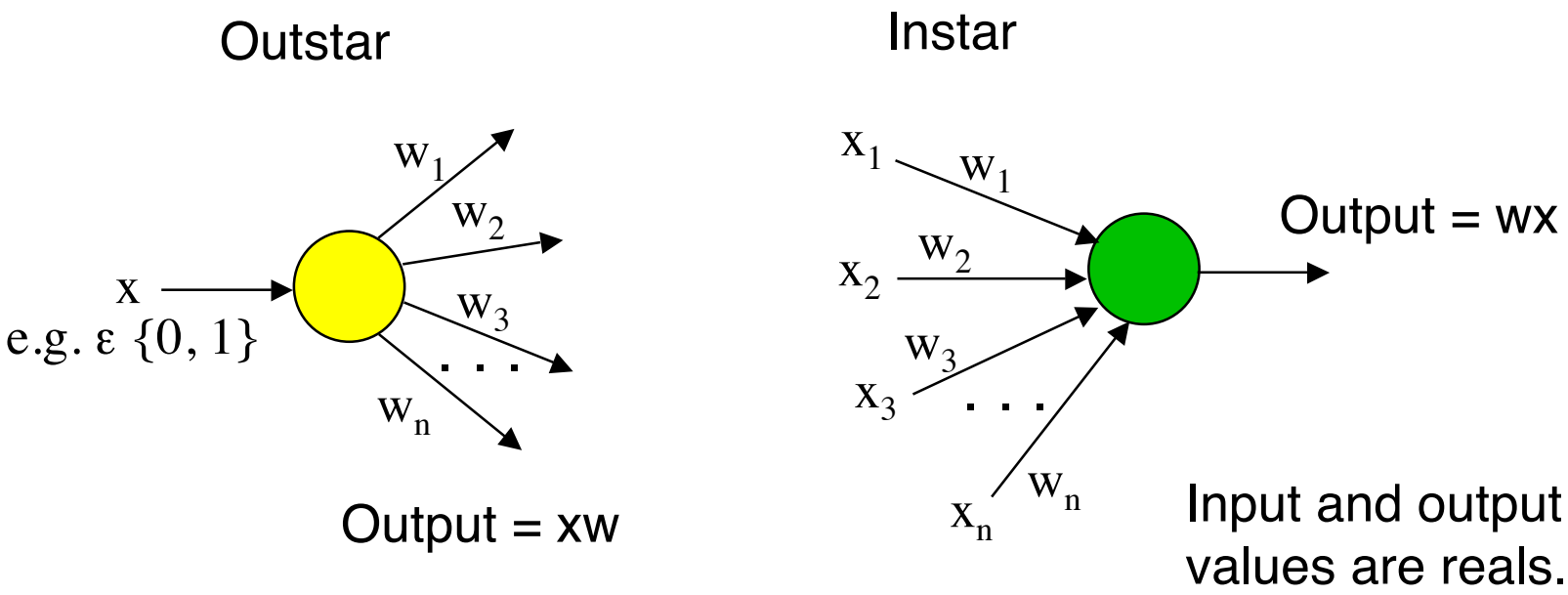
Counterpropagation Networks

- Invented by **Robert Hecht-Nielson**, founder of HNC Inc., acquired by Fair Isaac Co., Inc.
- Consists of **two** opposing sub-networks, one for learning a **function**, the other for learning its **inverse**.
- Each network has two layers:
  - A first layer that clusters inputs (sometimes stated to be a self-organizing map, see later).
  - An “outstar” second layer to provide the output values for each cluster.

Outstar and Instar (due to Stephen Grossberg)

- An **instar** responds to a single input vector.
- An **outstar** produces a single output **vector** when stimulated with a binary value.

Outstar and Instar



Biologically, outstar would be synaptic weights, while instar would have **dendritic** ones.
However, it is common to refer to both weights as “synaptic”.

Simplified Instar and Outstar

- Outstar learning rule:

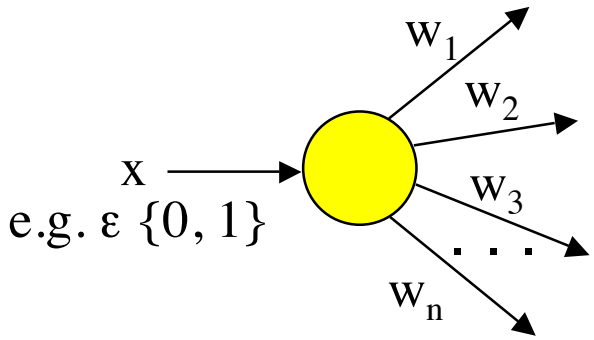
  $\Delta w = \alpha\, x\, (d - w)$

  (d = desired)

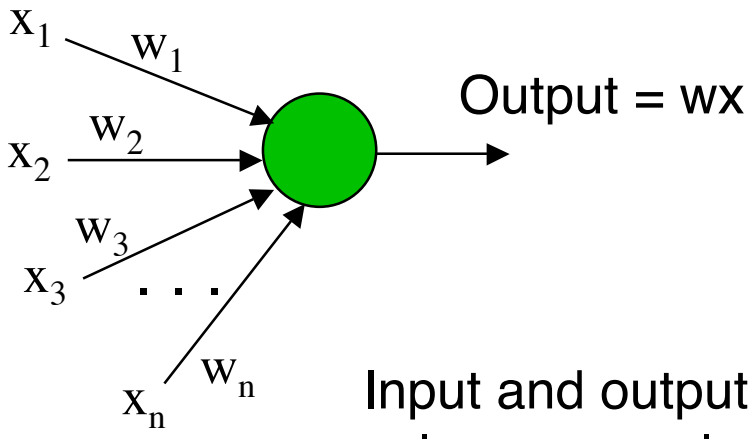
Instar learning rule:

$\Delta w = \alpha (x - w)\, d$

(d = desired)



Output = xw



Input and output values are reals.

Outstar and Instar

- Variations are possible, e.g. weight-decay.

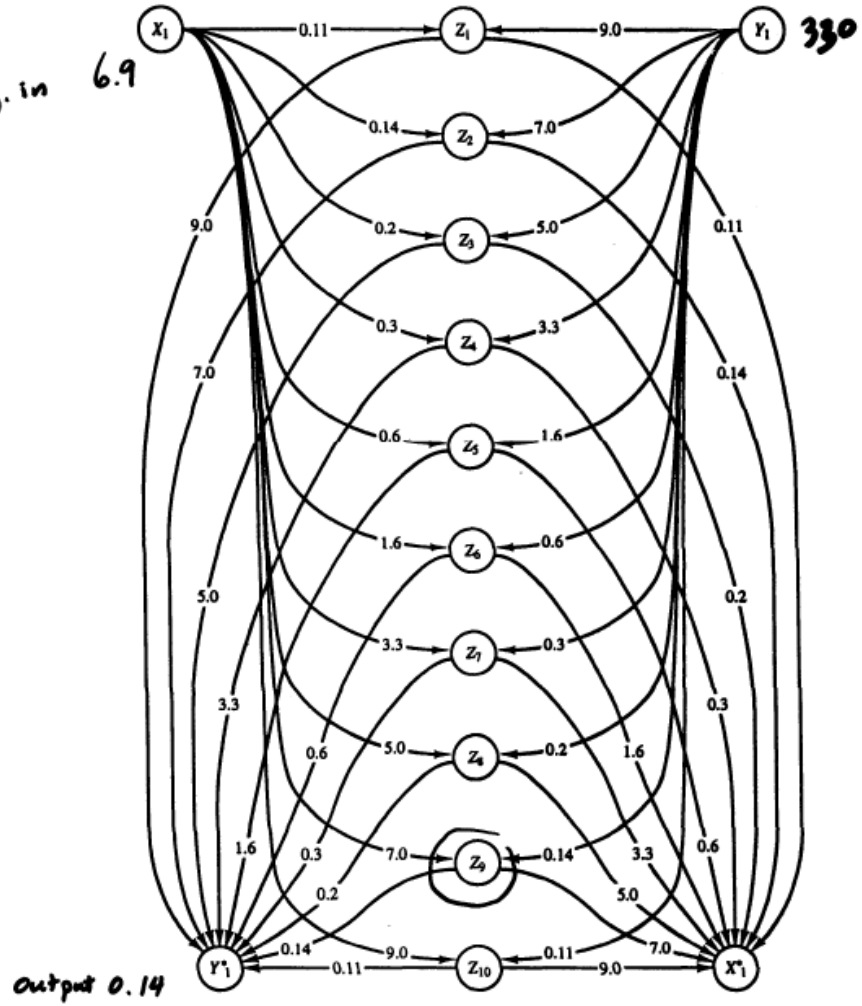
Counterpropagation Operation

- An outstar neuron is associated with each cluster representative.
- Given an input, the winner is found.
- The outstar is then stimulated to give the output.

Counterpropagation Notes

- Since these networks operate by **recognizing input patterns** in the first layer, one would generally use ***lots*** of neurons in this layer.
- There is a certain similarity to the approach of a radial-basis function network.
- In a full counterpropagation network, if the relationship being learned is a function, then ***inverse*** of the function can be learned at the same time.

Counterpropagation Network Computing Numeric Reciprocal



e.g. in 6.9

330

output 0.14

Figure 4.44 Full counterpropagation network for $y = 1/x$.

Competitive layer;
only winner fires.

outstar weights determine output

Counterpropagation vs. LVQ

Both combine Supervise and Unsupervised learning.

	Counterpropagation	**LVQ**
Uses	Classification and regression	Classification
Training phases	Two	One
Output Weights	Trained in phase two	Fixed *a priori*

Counterpropagation Applications

- TeleSpec: IR Spectrum of Molecular Structures
  www2.chemie.uni-erlangen.de/services/telespec/english/StructureInput.html
- A trained neural network is able to predict an infrared spectrum for a molecule **it has not seen before**.
- The structure code of the query structure is compared with the structure block of the neural network.
- The similarity criterion is the MSE between the query structure code and the neuron weights.
- The most similar neuron, the winning neuron, acts as a pointer to the corresponding information in the output block. The output block provides the simulated spectrum

TeleSpec Operation

- The first step in simulating an IR spectrum is the input of a query structure. TeleSpec provides an integrated Java-based structure editor that converts a graphical input into a so-called **SMILES** string, a standardized ASCII description of the connectivity of a molecule.

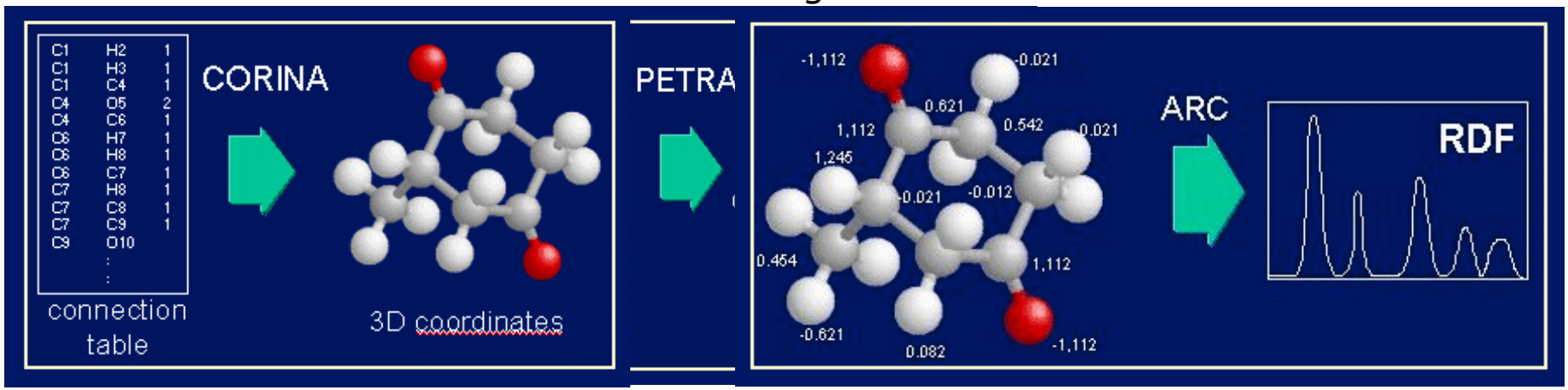
- The SMILES string is then converted into 3D coordinates of the atoms with the integrated structure generator CORINA, (**CO**o**R**d**INA**tes), a rule and data based system, that automatically generates three-dimensional atomic coordinates from the constitution of a molecule as expressed by a connection table or linear code, and which is powerful and reliable to convert large databases of several hundreds of thousand or even millions of compounds.

  http://www2.chemie.uni-erlangen.de/software/corina/index.html

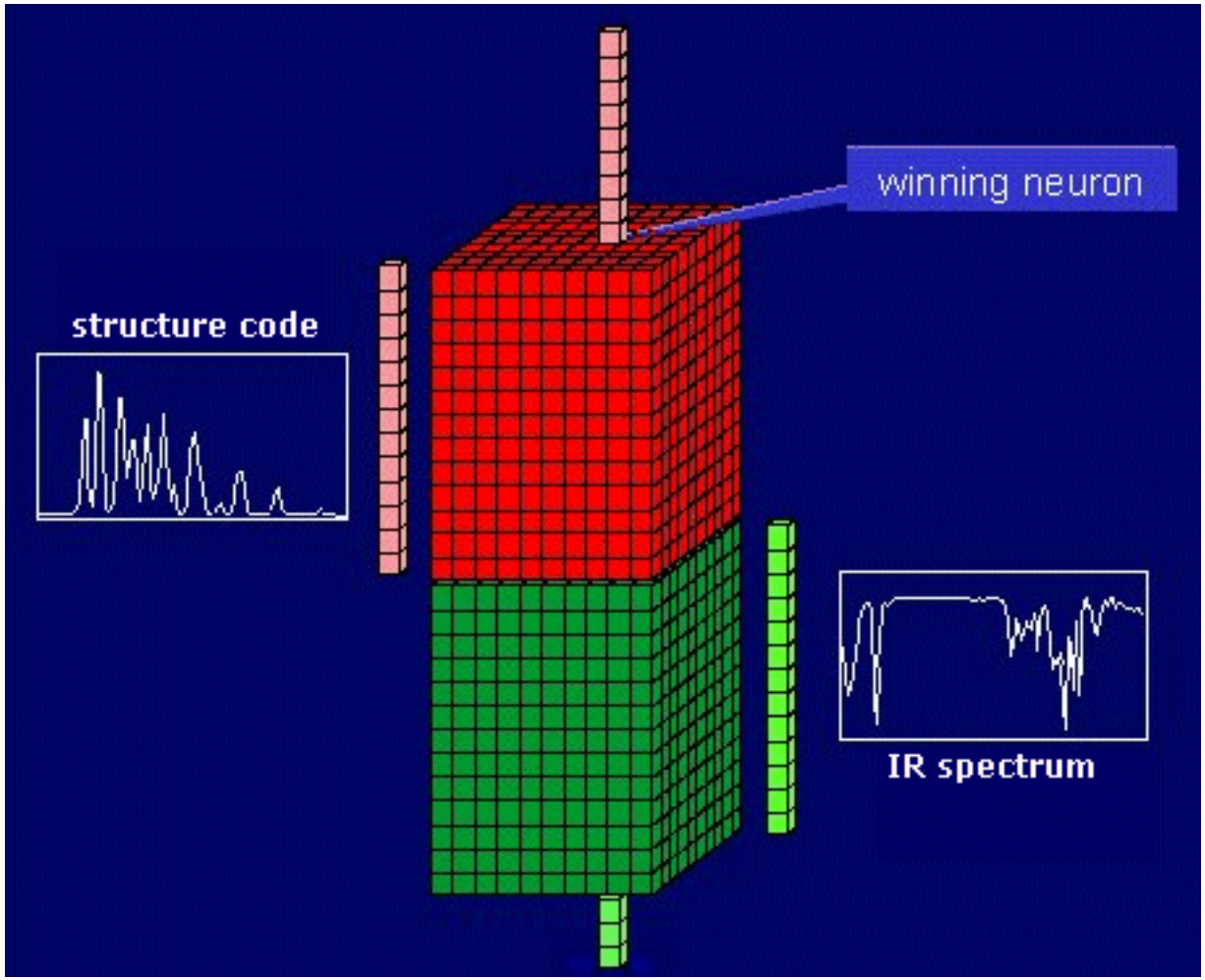
TeleSpec Operation

- In addition to the three-dimensional arrangement of the atoms in a molecule, physicochemical properties are needed to describe molecules in more detail. This is achieved by integrating a series of empirical methods for the calculation of electronic and energy effects, such as charge distribution or polarizability effects. This program package is called PETRA (Parameter Estimation for the Treatment of Reactivity Applications).

- This is followed by a structure coding step (shown as ARC) which encodes into a numeric vector using either of two methods.



TeleSpec use of Counterpropagation



TeleSpec Network Training

- The network learns the relationship between structure and spectrum by analyzing a set of pairs of molecules and spectra in a training phase. For each data point, a pair of structure and spectrum, two steps are performed:
  - **Determining the most similar neuron** For each data point, the most similar neuron is determined by calculating the rms error between the structure code (red) of the training structure and the input (upper) block of the neural network.
  - **Adjustment of the neuron weights** The weights of the winning neuron are adjusted to become more similar to the training data point. The weights of the neighbor neurons are also adjusted with the rate of adjustment decreasing with increasing distance from the winning neuron. The adjustment is applied to the structure (green) block and the spectrum (red) block of the network. This training step establishes the correlation between structures and spectra and stores it in the network.
- A trained counterpropagation network acts as an interpolator and is able to predict a spectrum for a molecule it has never seen before.

Another Counterpropagation Application: Dolphin Echolocation

- Roitblat, H. L., P. W. B. Moore, P. E. Nachtigall, R. H. Penner, and W. W. L. Au. 1989. Dolphin Echolocation: Identification of Returning Echoes Using a Counterpropagation Network. International Joint Conf. on Neural Networks, Vol. 1, IEEE and International Neural Network Society, Piscataway, NJ, pp. 295-30.

- Describes preliminary work on using a counterpropagation artificial neural network to recognize echoes from objects ensonified in a test pool by an artificial dolphin click and in Kaneohe Bay by a dolphin during performance of a delayed matching- to-sample task. Selected echoes were analyzed and successfully recognized by the network. Target recognition abilities of an echolocating dolphin and the neural network were also compared. **In a noisy natural environment, the dolphin was 94.5 percent correct and the network was 96.7 percent correct.** Possible applications of neural networks to echolocation studies are discussed.

Counterpropagation vs. MLP

Robocup soccer application “a4ty”
(http://dssg.cs.rtu.lv/download/robocup/a4ty2003_long.pdf)

“We applied [counterpropagation] for **pass selection**. An agent trained with offline learning needs to decide whether it should give a pass to a partner or not. If it makes an incorrect pass (the ball is intercepted by an opponent) in online mode in the given situation, the network updates weights so as next time the probability of choosing this action for the same (or similar) situation would be lower.

However, this approach has not been implemented in the final version of a4ty team since the **overall performance (approximation capabilities) of counterpropagation network is worse than, for example, multilayer perceptron**. Thus, counterpropagation neural network has been replaced by multilayer perceptron (MLP)”.