Competitive Learning

Reference

- NAS, Chapter 7

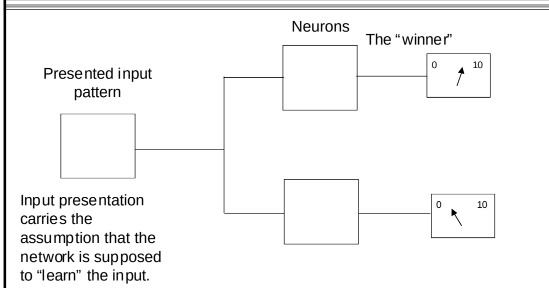
Types of Learning

- **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

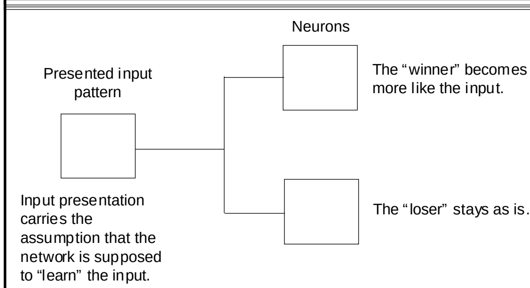
Competitive Learning

- 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



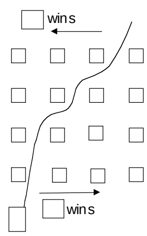
Why not make the winner *exactly* like the input?

- 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.

Grandmother Cells

- A neuron that recognizes exactly one pattern is called a "grandmother cell".
- It is based on the folklore that everyone has a neuron that fires when, and only when, he/she sees his/her grandmother.

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 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 12),

  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**:

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

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

$$x \, w_i$$

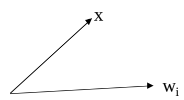
is **larger** when x is closer to w_i.

Usually it is best if x and are **normalized** before using this measure, so that

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

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

larger is farther

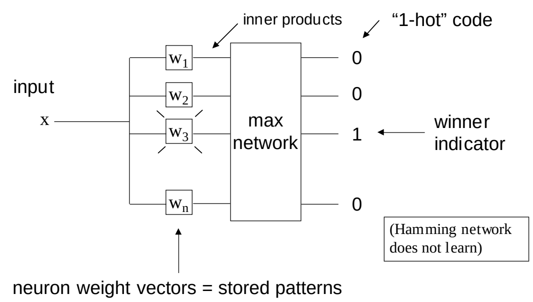
- inner product = [1 1 -1 1] $[1\ -1\ -1\ -1]^T = 0$

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



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

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

- These weights are designed so that:
  - all but one output is non-zero after n cycles
  - all outputs persist at the same value after n cycles

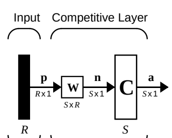
MaxNet Example

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

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

Competitive Learning

Instar Rule (Stephen Grossberg)

$${}_i\mathbf{w}(q) = {}_i\mathbf{w}(q-1) + \alpha \underbrace{a_i(q)}\overbrace{(\mathbf{p}(q) - {}_i\mathbf{w}(q-1))}^{\text{pattern - weight}}$$

1 for i = winner
0 otherwise

learning rate

Only winner learns

Similar to the Adaline rule, with:
input = whether a winner
output = weight

Kohonen Rule

(when specialized to single winner = Instar Rule)

input

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

index of winners

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

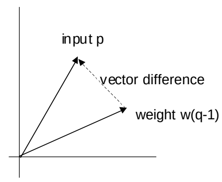
$${}_i\mathbf{w}(q) = {}_i\mathbf{w}(q-1) \qquad i \neq i^*$$

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

Teuovo Kohonen

Dr. Eng., Professor of the Academy of Finland;
Head, Neural Networks Research Centre,
Helsinki University of Technology, Finland

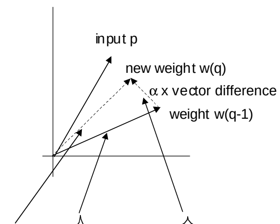
Graphical Representation



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

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

Graphical Representation

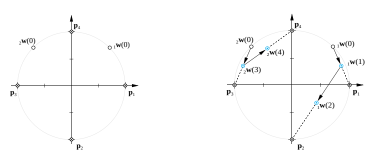


$$_{i^*}\mathbf{w}(q) = \underbrace{{}_{i^*}\mathbf{w}(q-1)}_{} + \underbrace{\alpha(\mathbf{p}(q) - {}_{i^*}\mathbf{w}(q-1))}_{}$$

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

Matlab Demos

- nnd14cl (competive learning)

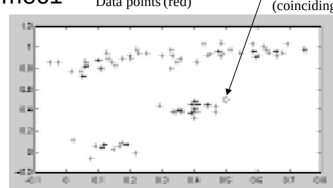


Matlab Demos

- democ1

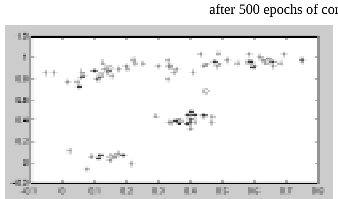
Data points (red)

2D weights of neurons
(coinciding initially)



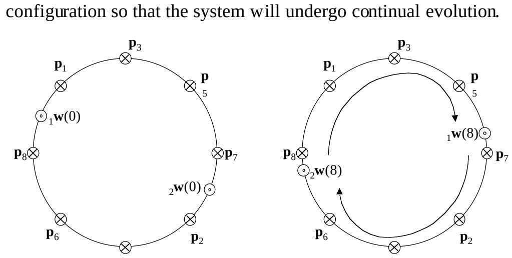
democ1

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



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.



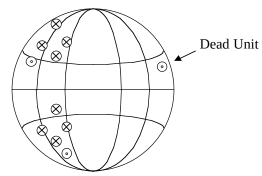
Possible Instability

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.

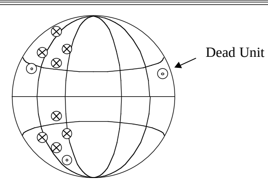
Solution: Gradually **decrease the learning rate**.

"Dead" Units / Starvation

One problem with competitive learning is that neurons with initial weights far from any input vector may never win.



Have a Heart



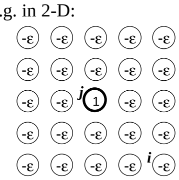
Solution: Add a negative bias to each neuron, and increase the magnitude of the bias as the neuron wins.
This will make it harder to win if a neuron has won often.
This is called the "**conscience**" method.

Mutual Weights by Distance On-Center/Off-Surround again

e.g. weights in the *competitive layer* of the Hamming network:

$$w_{i,j} = \begin{cases} 1, & \text{if } i = j \\ -\varepsilon, & \text{if } i \neq j \end{cases}$$

Weights can be regarded as being assigned based on *distance*, e.g. in 2-D:

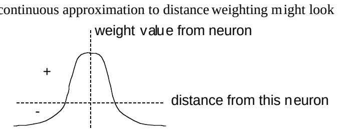


In this formula, "distance" is very discrete.

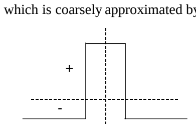
$$w_{i,j} = \begin{cases} 1, & \text{if } d_{i,j} = 0 \\ -\varepsilon, & \text{if } d_{i,j} > 0 \end{cases}$$

Mexican-Hat Function

A continuous approximation to distance weighting might look like:



which is coarsely approximated by:



Self-Organizing Maps

Kohonen Nets

Feature Maps

Self-Organizing Maps (SOMs)

Update weight vectors in a ***neighborhood*** of the winning neuron.

Kohonen Rule: $${}_i\mathbf{w}(q) = {}_i\mathbf{w}(q-1) + \alpha(\mathbf{p}(q) - {}_i\mathbf{w}(q-1))$$

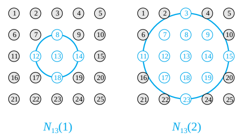
$${}_i\mathbf{w}(q) = (1-\alpha)\,{}_i\mathbf{w}(q-1) + \alpha\mathbf{p}(q) \qquad i \in N_{i^*}(d)$$

$$N_i(d) = \{ j, d_{i,j} \le d \}$$

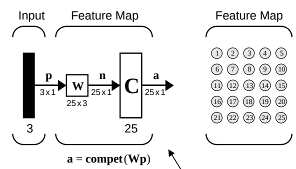
$$N_{13}(1) = \{8, 12, 13, 14, 18\}$$

$$N_{13}(2) = \{3, 7, 8, 9, 11, 12, 13, 14, 15, 17, 18, 19, 23\}$$

Two possible neighborhoods of neuron 13



SOM as a matlab Competitive Network



$\mathbf{a} = \mathbf{compet}(\mathbf{Wp})$

Does not show neighborhood computation. Neighborhood starts large, gradually decreases, along with learning rate.

Maps in Neurobiology

- Related neural functions ***map*** onto identifiable regions of the brain
  - retinotopic map: vision, *superior colliculus*
  - phonotopic map: hearing, auditory cortex
  - somatotopic map: touch, somatosensory cortex

Human Brain

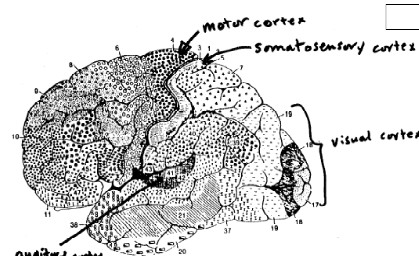
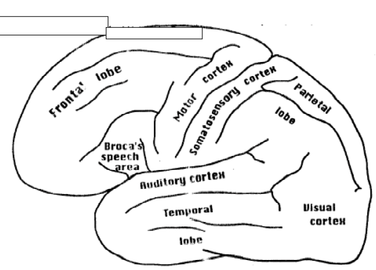


FIGURE 10.1 Cytoarchitectural map of the cerebral cortex.

Regions of your the Brain



Desired Properties of Maps

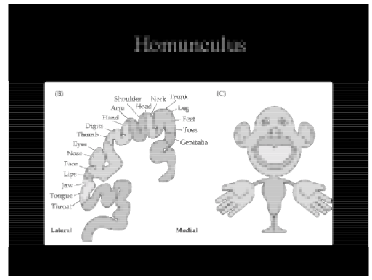
- **Approximation of input space**: generally a many-one mapping of input space into weight space
- **Topology-preserving**: point close together in input space should map to points close together in weight space.
- **Density-preserving**: regions of similar density should map to regions of proportional density.

Somatotopic Map Illustration: The "Homunculus"

Cartoon map of the relationship between body surfaces and the regions of the brain that control them

(somewhat different from the original "little person inside" meaning).

Another Depiction



Project: Map your own homunculus:

http://www.woodrow.org/teachers/biology/institutes/1991/homunculus.html

"Imposed Dimensionality" of SOM

- In a general SOM, an n-dimensional graph, n $\geq$ 1, can be *overlaid* with the neurons as nodes.
- The edges connecting the neurons that *constrain the neighborhood for updating* (rather than using Euclidean distance).
- Only nodes a specified *graphical* diameter away are in the neighborhood.

Uses of "Imposed Dimensionality" in Data Analysis

- Data points may have an unknown dimensionality, e.g. the underlying phenomenon or process that has several dimensions of attributes (such as color dimensions, various size or rate dimensions, etc.)
- By training a network with an *imposed* dimensionality, the relationships among the data may become visualizable, especially if the imposed number of dimensions is $\leq$ the actual dimensions in the data.

Uses of "Imposed Dimensionality" in Data Analysis

- In other words, training the map "organizes" or "sorts" the data according to similarity in each dimension.

Demo Applets

- /cs/cs152/kohonen/ demo1, demo2, demo3
- Legend:
  - black = input data point (random over a 2-D region; 2-dimensional data)
  - red = neuron in winner neighborhood; learns by Kohonen rule
  - blue = other neuron
- Learning rate and neighborhood both decrease with time; demo speeds up over time.

Competition Code

```
// compete finds the neuron with weights closest to the input.
// It sets winpoint to the indices of that neuron.

  public void compete(Input input)
    {
    // initialize min with an distance to an arbitrary neuron
    winpoint = 0;
    double min = neuron[winpoint].distance(input);

    // find the min distance among all neurons

    for( int point = 0; point < points; point++ )
      {
      double dist = neuron[point].distance(input);
      if( min > dist )
        {
        min = dist;            // update the min distance
        winpoint = point;
        }
      }
```

Competition Code

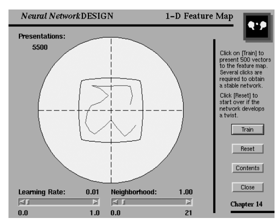
```
// learn updates all neurons within current neighborhood
// by applying the Kohonen learning rule against the current
// input.

  public void learn(Input input, double learningRate)
    {
    for( int point = 0; point < points; point++ )
      {
      if( inWinnersNeighborhood(point) )
        {
        neuron[point].learn(input, learningRate);
        }
      }
    }
```

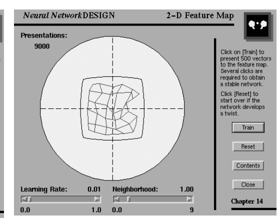
Similar matlab demos

Same input spaces, different imposed dimensions

1D

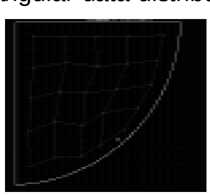


2D



Related Demos on the Web

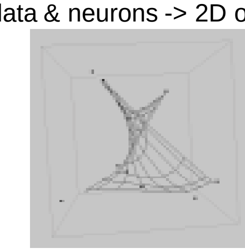
- Non-rectangular data distribution



http://www.patol.com/java/fill/index.html

Related Demos on the Web

- 3D data & neurons -> 2D overlay



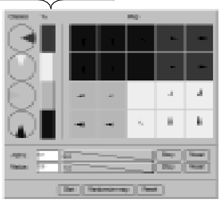
http://rfhs8012.fh-regensburg.de/~saj39122/jfroehl/diplom/e-index.html

Related Demos on the Web

- 2D data & neurons -> 2D overlay

fraction of samples of each color

distribution of direction vectors corresponding to colors

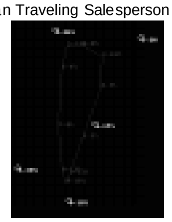


neurons, showing weights by direction vectors and corresponding color

http://www.cis.hut.fi/research/javasomdemo/demo2.html

Related Demos on the Web

- Finding a heuristic (not necessarily optimal) solution to the Euclidean Traveling Salesperson Problem using Kohonen net



red points = neurons,
red lines = overlay
circles = cities
neurons travel toward cities

http://www.patol.com/java/TSP/index.html

Related Demos on the Web

- Eight competitive models in one applet, including:
  - Kohonen
  - Neural Gas
  - Growing Neural Gas (GNG): Number of neurons grows
- Thirteen choices of input distribution
- Worth visiting

http://www.sund.de/netze/applets/gng/full/GNG_0.html

Applications of Kohonen Nets

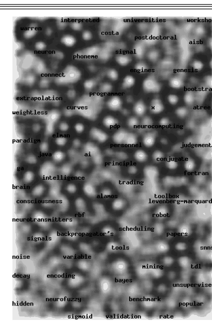
- Most applications that can be treated with MLP's can also be treated in some way by variants of Kohonen nets.
- Example: Learning a function f:D→R is done by treating the sample space as a set of (d, r) pairs.

Applications of Kohonen Nets

- Some applications more suited to Kohonen nets
- Data-mining applications, discovering similarites in data.

Example: Classifying news messages on Neural Nets by terms occurring within

(seehttp://websom.hut.fi/websom/comp.ai.neural-nets-new/html/root.html)

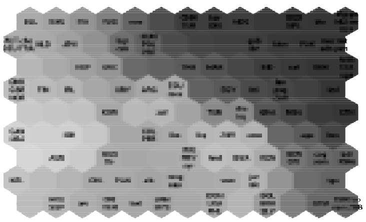


Automatically generated labels and examples of titles in which the labels hav

ai - SubSymbolic AI :-O
aisb - AISB and ECCS 1997 - Call for Registrations
alamos - Graduate Research Assistantship in Los Alamos National Laborato
atree - Atree 3.0 EK mirror sites
backpropagator's - Backpropagator's review, by Donald R. Tveter
bayes - nn and stat decisions for med: Bayes vs neorotic nets
benchmark - Benchmark data for signal processing
bootstrap - Bootstrapping vs. Bayesian
brain - Brain usage Was:Re: Function of sleep
conjugate - Moller's Scaled conjugate gradient (Re: Questions about my new
connect - Direct Connect Problems
consciousness - electromagnetics, brain waves and consciousness
costa - New paper available "The Application of Fuzzy Logic in Automatic M
curves - Predicting Learning Curves (Ph.D. thesis, online)
decay - weight decay
elman - Elman (or Edelman?) nn, references please.
encoding - Question on Encoding Input Data
engines - IC ENGINES & NEURAL NETS
extrapolation - Generalization (Interpolation & Extrapolation)
fortran - Neural Net Fortran Code
ga - GA for learning in ANN
genesis - GENESIS 2.0
hidden - Hidden layer heuristics
intelligence - Commercial Intelligence?
interpreted - Lisp is not an interpreted language
java - NN in Java?

Example: Classifying World Poverty from a 39-dimension indicator

(http://www.cis.hut.fi/research/som-research/worldmap.html)



The colors were automatically assigned after 2-D clustering using a Kohonen map.

Colors transferred to a world map

Other Interesting Applications

- Tree experiment
- Animal characterization experiment
- Phonetic typewriter
- Function approximation
  - Robot Hand-Eye coordination
  - Master/Slave learning technique