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

- Presented input pattern
- Input presentation carries the assumption that the network is supposed to "learn" the input.
- Neurons
- The "winner" adjusts to be still closer to the input pattern
- The "loser" stays as is.
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:
  $$\| x - w_i \| = \sqrt{\sum_j (x_j - w_{ij})^2}$$

Measures of distance

- Another measure of distance, used when the values are integer, is the "Manhattan" or "city-block" distance:
  $$\| x - w_i \| = \sum_j |x_j - w_{ij}|$$

- Another measure of distance, used when the values are 2-valued, is the "Hamming distance":
  $$\sum_j (|x_j - w_{ij}|)$$

  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 $x \cdot w_i$ 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.

![](image1.png)

**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 = \( 0 + 1 + 0 + 1 = 2 \)
- Inner product = \( [1 \ 1 \ -1 \ 1] [1 \ -1 \ -1 \ -1]^T = 0 \)

**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 \(-\varepsilon\), where \(\varepsilon\) is any quantity < \(1/\text{(# of neurons)}\).

**Max Network**

- Activation functions are “poslin”:
  - poslin\((x) = x \text{ if } x > 0, \ 0 \text{ otherwise}\)
- 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
  \[ y_i := \text{poslin}(y_i - \varepsilon \sum_{j \neq i} y_j) \]

- 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 $\varepsilon = 0.2 < 1/4$

<table>
<thead>
<tr>
<th>step</th>
<th>$y_1$</th>
<th>$y_2$</th>
<th>$y_3$</th>
<th>$y_4$</th>
<th>sum epsilon</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.6000</td>
<td>0.0000</td>
<td>2.8000</td>
<td>0.4000</td>
<td>4.8000</td>
</tr>
<tr>
<td>2</td>
<td>0.9600</td>
<td>0.0000</td>
<td>2.4000</td>
<td>0.0000</td>
<td>3.3600</td>
</tr>
<tr>
<td>3</td>
<td>0.4800</td>
<td>0.0000</td>
<td>2.0800</td>
<td>0.0000</td>
<td>2.6880</td>
</tr>
<tr>
<td>4</td>
<td>0.0384</td>
<td>0.0000</td>
<td>2.1120</td>
<td>0.0000</td>
<td>2.1504</td>
</tr>
<tr>
<td>5</td>
<td>0.0000</td>
<td>0.0000</td>
<td>2.1043</td>
<td>0.0000</td>
<td>2.1043</td>
</tr>
<tr>
<td>6</td>
<td>0.0000</td>
<td>0.0000</td>
<td>2.1043</td>
<td>0.0000</td>
<td>2.1043</td>
</tr>
</tbody>
</table>

Matlab `compet` function (non-learning)

- `COMPET(N)` takes one input argument, $N$ - $S \times Q$ matrix of net input (column) vectors.
- Returns output vectors with 1 where each net input vector has its maximum value, and 0 elsewhere.

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

Competitive Learning

Instar Rule (Stephen Grossberg)

\[
\text{pattern} \cdot \text{weight} \ldots
\]

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)

\[
\omega(q) = \omega(q-1) \alpha \delta_{i,w} \omega(q) - \omega(q-1)
\]

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

Teuovo Kohonen

Dr. Eng., Professor of the Academy of Finland, Head, Neural Networks Research Cente, Helsinki University of Technology, Finland
Graphical Representation

\[ \text{weight } w(q-1) \]

Matlab Demos

**nnd14cl (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.

Matlab Demos

**democ1**

Data points (red)

3D weights of neurons (coinciding initially)

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.

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.

Solution: Add a negative bias to each neuron, and increase the magnitude of the bias as the neuron wins. 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:

\[
\begin{align*}
    w_{ij} &= \begin{cases} 
    1, & \text{if } i = j \\
    -\epsilon, & \text{if } d_{ij} > 0 
    \end{cases}
\end{align*}
\]

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

In this formula, “distance” is very discrete.

Mexican-Hat Function

A continuous approximation to distance weighting might look like:

Self-Organizing Maps

Kohonen Nets
Feature Maps
Self-Organizing Maps (SOMs)

Update weight vectors in a neighborhood of the winning neuron.

Kohonen Rule:

\[
W_i(q) = W_i(q-1) + \alpha(t) \cdot \Delta_i \cdot \Delta_i(q)
\]

\[
N_i(d) = \{ j \mid d_{ij} \leq d \}
\]

Two possible neighborhoods

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

\[
N_{13}(2) = \{ 3, 7, 8, 9, 11, 12, 13, 14, 15, 17, 18, 19, 20 \}
\]

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

SOM as a matlab Competitive Network

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

Human 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.

Regions of your the Brain
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:

“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 less than 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
    double min = neuron[winpoint].distance(input);
    // find the min distances 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


Related Demos on the Web

- 3D data & neurons -> 2D overlay

http://ifhs801.2.fh-regensburg.de/~saj39122/jfroehl/diplome/index.html
Related Demos on the Web

- 2D data & neurons -> 2D overlay
  
  http://www.cis.hut.fi/research/javasomdemo/demo2.html

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

- 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 $f:D \rightarrow R$ is done by treating the sample space as a set of $(d, r)$ pairs.
  
  Example: Classifying news messages on Neural Nets by terms occurring within

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