Selforganizing Maps (Kohonen Network)

and

“Friends”

Competetive Learning: cont´d

$$\Delta w_i = \eta \left( x_j - w_i \right)$$

Learning rule can be derived from this cost function:

$$E(w_i) = \frac{1}{2} \sum_j \left( x_j - \tilde{u}_i \right)^2$$

minimize sum of squared errors.

Note 1: may be seen as online version of k-means clustering
Note 2: can modify learning rule to make all units fire equally often

Simple Competetive Learning

$$h_i = \sum_j w_{ij} x_j = w_i \cdot x$$

simple linear units,
input is $x$

Assume weights $>0$ and possibly weights normalized.
If weights normalized, $h_i$ maximal if $w_i$ has smallest Euclidean distance to $x_i$.

output of unit: $o_i = \begin{cases} 1 & i = i^* \\ 0 & i \neq i^* \end{cases}$, sparse code!

Winner-Take-All Mechanism:
may be implemented through lateral inhibition

Vector Quantization

Idea: represent input by weight vector of the winner
(can be used for data compression: just store/transmit label representing the winner)

Question: what is set of inputs that “belong” to a unit $i$,
i.e. for which unit $i$ is winner?

Answer: Voroni tessellation (note: matlab command for this)

Competetive Learning: cont´d

$$\Delta w_i = \eta \left( x_j - w_i \right)$$

learning rule is Hebb-like
plus decay

Weight of winning unit is moved towards current input $x_j$.
Similar to Oja’s and Sanger’s rule.

Geometric interpretation: units (their weights) move in input space.
Winning unit moves towards current input.

Self-Organizing Maps (Kohonen)

Idea: Output units have a priori topology (1-dim. or 2-dim.),
e.g. are arranged on a chain or regular grid.
Not only winner get`s to learn but also its neighbors.

SOMs transform incoming signal patterns of arbitrary dimension
on 1-dim. or 2-dim. map. Neighbors in the map respond to similar
input patterns.

Competition: again with winner-take-all mechanism
Cooperation: through neighborhood relations that are exploited
during learning.
Self-Organizing Maps (Kohonen)

example of 1-dim. topology of output nodes

Neighboring output units learn together! Update winner’s weights but also those of neighbors.

SOM algorithm

\[ h_j = \sum_i w_{ij} \xi_i = \tilde{w}_j \cdot \xi \] again, simple linear units, input is \( \xi \)

Define winner: \( i^* = \arg \max_i h_i \)

Output of unit \( j \) when winner is \( i^* \): \( \sigma_{ij} = \exp\left( -\frac{d_{ij}}{2\sigma_0} \right) \), \( d_{ij} = \| j - i^* \| \)

Same learning rule as earlier:

\[ \Delta \tilde{w}_j = \eta_0 \sigma_{i^*j} (\xi - \tilde{w}_j) \]

Usually, neighborhood shrinks with time: \( \sigma(t) = \sigma_0 \exp\left(-\frac{t}{\tau_0} \right) \) (for \( \sigma \to 0 \): competitive learning)

Usually, learning rate decays, too: \( \eta(t) = \eta_0 \exp\left(-\frac{t}{\tau_1} \right) \)

2 Learning Phases

Time dependence of parameters:

\[ \sigma(t) = \sigma_0 \exp\left(-\frac{t}{\tau_0} \right) \quad \eta(t) = \eta_0 \exp\left(-\frac{t}{\tau_1} \right) \]

1. Self-organizing or ordering phase
   - topological ordering of weight vectors
   - use:
     \( \sigma_0 = \text{radius of layer} \), \( \tau_0 = \frac{1000}{\log \sigma_0} \), \( \eta_0 = 0.1 \), \( \tau_1 = 1000 \)

2. Convergence phase
   - fine tuning of weights
   - use: very small neighborhood, learning rate around 0.01

Examples

2 Learning Phases

Figure taken from Haykin

Figure taken from Haykin

Figure taken from Hertz et.al.

Figure taken from Hertz et.al.
Feature Map Properties

Property 1: feature map approximates the input space

Property 2: topologically ordered, i.e. neighboring units correspond to similar input patterns

Property 3: density matching: density of output units corresponds qualitatively to input probability density function

Property 4: feature selection: SOM selects best features to approx. input probability density function.

Lissom equations

input activities are elongated Gaussian blobs:

$$E_{ij} = \exp \left( -\frac{1}{2} \left( \frac{d_i - c_j}{h} \right)^2 \right)$$

initial map activity is nonlinear function of input activity, weights $\mu$:

$$\mu \left( \sum E_{ij} \right)$$

time evolution of activity $\mu, E, I$ are all weights)

$$\frac{d\mu}{dt} = \sum \left( \mu - \sum E_{ij} \right)$$

Hebbian style learning with weight normalization all weights learn but with different parameters:

$$\mu_{j,i} = \sum E_{ij} \mu_i$$

Contextual Maps

Different way of displaying SOM: label output nodes with a class label describing what output node represents. Can be used to display data from high dimensional input spaces in 2d.

Input vector is concatenation of attribute vector and symbol code: (symbol code “small” and free of correlations)

$$z = z_s, \bar{z}_s, 1$$

Figure taken from Haykin.

RF-LISSOM model

Receptive Field Laterally Interconnected Synergistically Self-Organizing Map

Figures taken from http://www.cs.texas.edu/users/nn/web-pubs/htmlbook96/sirosh/

Understanding Cortical Maps

primary visual cortex: orientation selectivity, retinal position, ocular dominance all mapped onto 2-dim. map.

Figure taken from Tanaka lab web page

Demo: (needs supercomputer power)
Contextual Maps cont’d.
Contextual Map trained just like standard SOM.
Which unit is winner for symbol input only?

![Figure 9.17 Feature map containing labeled neurons with strongest responses to their respective inputs.](image)

Contextual Maps cont’d.
For which symbol codes fires unit the most?

![Figure 9.18 Semantic map obtained through the use of simulated electrode penetration mapping. The map is divided into three regions representing: birds, peaceful species, and hunters.](image)

Learning Vector Quantization
Idea: supervised add-on for the case that class labels are available for the input vectors
How it works: first do unsupervised learning, then label output nodes

![Figure 9.13 Block diagram of adaptive pattern classification, using a self-organizing feature map and learning vector quantizer.](image)

Learning rule: $C_x$: desired class label for input $x$
$C_w$: class label of winning unit with weight vector $w$

- In case $C_x = C_w$: $\Delta w = a(t) [x - w]$ (move weight towards input)
- In case $C_x \neq C_w$: $\Delta w = -a(t) [x - w]$ (move away from input)

$a(t)$: decaying learning rate

![Figure 9.14 LVQ data:](image)

![Figure 9.15 LVQ result:](image)

Topology Learning: neural gas
Idea: no fixed underlying topology

![Figure 9.16 Competitive Topology Learning](image)

**Algorithm 6.3 Competitive Topology Learning**

1. Initialize the prototypes $w_j, j = 1, \ldots, R$.
2. Initialize the connectivity matrix $C_{ij}$ of connection strengths to 0.
3. For each input pattern $x$ and for each $w_j$, do the following:
   - Enhance $l_j$, the number of prototypes closest to $w_j$, and find the two closest, $w_i$ and $w_j$.
   - For each $w_i$, $w_j$, and $l_j$, do the following:
     - Compute the connectivity $C_{ij}$.
     - Update the weight for those units according to $\Delta w = -\Delta w_i + \Delta w_j$.
   - Update the connectivity information.
   - Set $C_{ij} = 1$ and $l_j = 0$.
   - For all $l_j$ such that $C_{ij} = 1$ and $l_j < T$, set $C_{ij} = 0$.
   - Set $\Delta w$ is sufficiently small.
SOM Summary

“simple algorithms” with complex behavior.

Can be used to understand organization of response preferences for neurons in cortex, i.e. formation of cortical maps.

Used for visualization of high dimensional data.

Can be extended for supervised learning.

Some notable extensions, e.g. neural gas and growing neural gas, that are generalizations to 2-dim topology. Proper (local) topology is discovered during the learning process.