Competitive Learning

15-496/782: Artificial Neural Networks
David S. Touretzky

Spring 2004

"Self-Organizing Systems"

Problem: find the (or some) underlying structure in a complex environment.

Approach: group input points into clusters.

Examples:

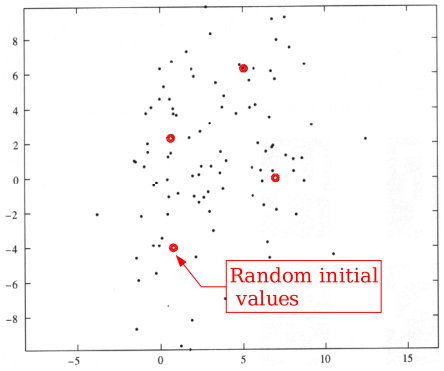
- Concept learning: group animals into "cats", "dogs", "people"
- Visual structure: group pixels into edges of various orientations

How Do We Find the Clusters?

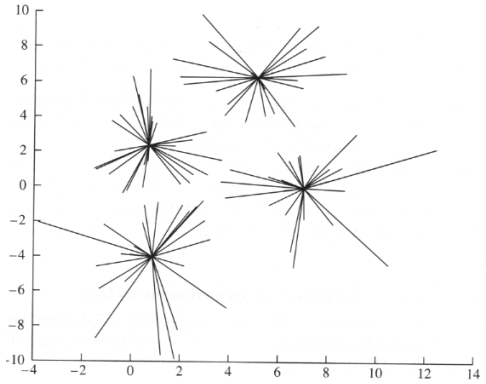
The k-Means Algorithm

1. Create k clusters, with each mean μ_i initialized to a randomly chosen point from the training set.
2. Label each point by the cluster that "captures" it (closest μ_i value.)
3. Recompute each μ_i as the mean of all points captured.
4. Repeat until labels stop changing.

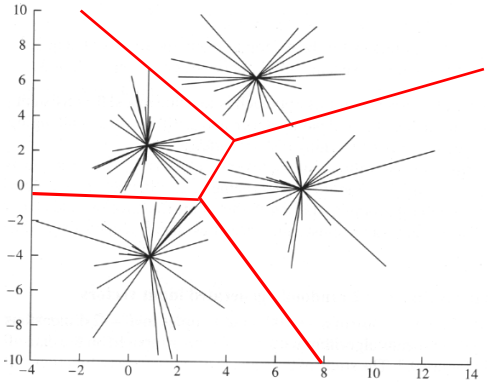
k-Means with k=4



k-Means Computes Four Clusters



Nearest Neighbor Voronoi Diagram



Problems with k-Means

1. Not an on-line learning algorithm (although on-line variants exist: competitive learning.)

2. Must remember the label of each point in order to adjust the means.

Note: once you have computed the k cluster centers, you can discard the training data and classify new points using just the μ_i. Much more efficient than nearest neighbor. (But you don't get to choose the class labels!)

Competitive Learning

An unsupervised training method, like k-means.
No teacher. No error signal.

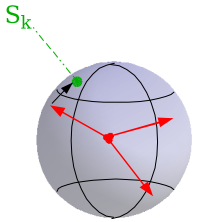
Unsupervised learning systems are often referred to as "self-organizing systems".

More biologically plausible than backprop, but computationally weaker.

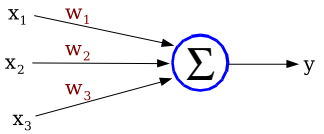
See HK&P sections 9.1, 9.2, 9.4, and 9.7.

Competitive Learning Algorithm

1. Initialize units so they have different weights.
2. Pick an input point S_k, and find the closest matching unit (the "winner") for that point.
3. Adjust the winner's weight vector in the direction of the input S_k.
4. Repeat, for all input points, for many epochs.



"Strongest Match" = Largest Dot Product



$$y = \sum_i x_i \cdot w_i$$

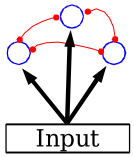
Neurons could do this!

$$\mathbf{x} \cdot \mathbf{w} = x_1 w_1 + x_2 w_2 + x_3 w_3$$

Picking A Winner by "Lateral Inhibition"

Each unit inhibits the others in proportion to its current activation level.

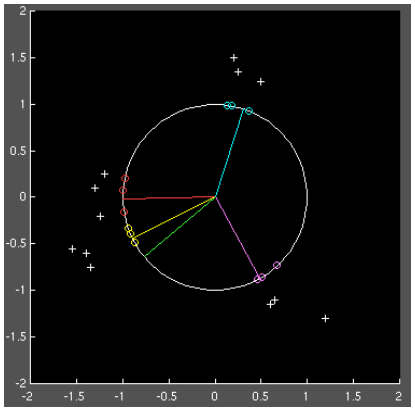
"Winner take all" dynamics.



Self-excitatory connections enhance stability (Grossberg).

Can skip WTA dynamics and just use "max" to find the winner directly.

Competitive Learning Demo



Initializing the Weights

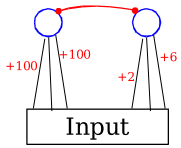
1. Initialize to random values.
2. Initialize to randomly-selected training points.
3. Initialize to constant vector **v**, and "turn on" the input patterns gradually during learning:

$$\alpha \xi_i + (1-\alpha)\mathbf{v}$$

where α goes from $0 \rightarrow 1$

As input patterns separate, so will the weight vectors.

Potential Problem: Starvation



Problem: one unit could win all the time.

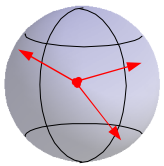
Solution: limit the sum of a unit's input weights.

For all units j, $\sum_i |w_{ij}| = B$

Assume w.o.l.o.g. that B = 1.

Starvation Averted

Now the weight vectors correspond to points on a hypersphere of radius B.



No one unit can win for all possible patterns.

Learning Algorithm: Simple Version

1. Move winning unit's weight vector in the direction of the input pattern S_k.

$$w_{ij} \leftarrow w_{ij} + c_{ik} \cdot g$$

c_{ik} = bit i of pattern S_k
g = learning rate constant

2. Renormalize the weights: $w_{ij} \leftarrow \frac{w_{ij}}{\sum_i w_{ij}}$

Is weight normalization biologically plausible? Maybe.

But it's easier to normalize the inputs instead. Grossberg does this by an extra input layer.

Rumelhart & Zipser also do this.

Learning Algorithm: Rumelhart & Zipser Version

$$\Delta w_{ij} = \begin{cases} 0 & \text{if unit j loses for } S_k \\ g\frac{c_{ik}}{n_k} - g\,w_{ij} & \text{if unit j wins for } S_k \end{cases}$$

c_{ik} = bit i of pattern S_k

$n_k = \sum_i c_{ik}$ = number of 1 bits

g = learning rate constant, e.g., 0.01

Maintaining Normalized Weights

When j wins: $\Delta w_{ij} = g\frac{c_{ik}}{n_k} - g\,w_{ij}$

Since weights are normalized:

$\sum_i w_{ij} = 1$ for any unit j

$\sum_i \frac{c_{ik}}{n_k} = 1$ for any input pattern

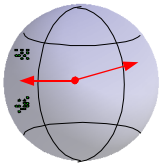
So $\sum_i \Delta w_{ij} = 0$ for any unit j

So $\sum_i (w_{ij} + \Delta w_{ij}) = 1$ weights stay normalized!

Leaky Learnning

Due to an unfortunate choice of initial weights, some units may never be winners.

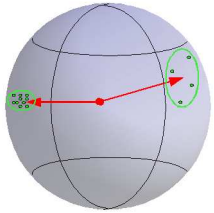
Solution: "leaky learning."



$$\Delta w_{ij} = \begin{cases} g_w\frac{c_{ik}}{n_k} - g_w w_{ij} & \text{if j wins} \\ g_l\frac{c_{ik}}{n_k} - g_l w_{ij} & \text{if j loses} \end{cases} \quad \text{where } g_l \ll g_w$$

Competition with a "Conscience"

Alternative to leaky learning: each unit adjusts its sensitivity (via a bias term b_j) so that it wins a reasonable amount of the time.



$$y_j = \sum_i w_{ij} \cdot c_{ik} + b_j$$

What is a "Good" Partitioning

$$M_i^\xi = \begin{cases} 1 & \text{if unit i wins pattern } \xi \\ 0 & \text{otherwise} \end{cases}$$

Lyapunov cost function: $E(\mathbf{w}) = \frac{1}{2} \sum_{i,j,\xi} M_i^\xi (\xi_j - w_{ij})^2$

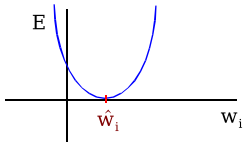
Sum of distances between patterns and their winning unit's weight vector.

Learning rule: $\Delta w_{ij} = -\eta \frac{\partial E}{\partial w_{ij}} = -\eta M_i^\xi \xi_j$

Optimal partition: minimal $E(\mathbf{w})$.
There are many local minima.

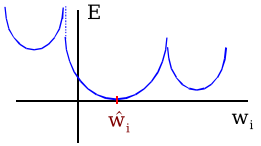
Local Minima

LMS learning has a quadratic error surface with no local minima.



Competitive learning does have local minima because the "winner" is a discontinuous function.

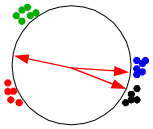
$$M_i^\xi = \begin{cases} 1 & \text{if unit i wins pattern } \xi \\ 0 & \text{otherwise} \end{cases}$$



$$E(\mathbf{w}) = \frac{1}{2} \sum_{i,j,\xi} M_i^\xi (\xi_j - w_{ij})^2$$

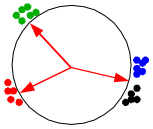
Example of a Local Minimum

Suboptimal Partitioning



Local minimum

Optimal Partitioning



Global minimum

Convergence

Stochastic competitive learning (process training points one at a time) might not converge.

Grossberg: weight vectors can go through cycles if the training patterns are presented cyclically.

The batch version always converges.

(But batch learning not neurophysiologically plausible.)

Stability

Will the weight vectors shift around forever?

The model reaches an "equilibrium state" when the weights stop changing, i.e. the average Δw_{ij} is zero.

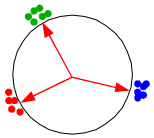
Under what conditions is the model stable?

Stability Measure

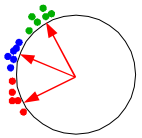
$$T = \sum_k p_k \sum_{i,j} v_{jk}(\alpha_{jk} - \alpha_{ik})$$

p_k = probability of pattern k
v_{jk} = probability unit j wins for pattern k
α_{jk} = activation level of unit j on pattern k

T = for each pattern, how much does the activation level of the winner exceed that of the losers?



High T (stable): winner "wins big". Widely separated regions.



Low T (unstable)

Characterizing the Stable State

Let's assume binary patterns...

p_k = probability of seeing pattern S_k
v_{jk} = probability that unit j wins for pattern S_k

What's the chance until j will win on the next trial?

$$\sum_k p_k v_{jk}$$

What's the chance the next input pattern will have bit i on?

$$\frac{1}{k}\sum_k p_k c_{ik}$$

Weights Don't Change On Average

$$\sum_k p_k v_{jk} \Delta w_{ij} = 0 \quad \text{for all i,j}$$

By substitution:

$$0 = g\sum_k \frac{c_{ik}}{n_k} p_k v_{jk} - g\sum_k w_{ij} p_k v_{jk}$$

$$w_{ij}\underbrace{\sum_k p_k v_{jk}}_{\text{prob. that unit j wins}} = \sum_k \frac{c_{ik} p_k v_{jk}}{n_k}$$

Nature of the Stable State

$$w_{ij} = \frac{\sum_k \frac{p_k c_{ik} v_{jk}}{n_k}}{\sum_k p_k v_{jk}}$$

prob. that bit i is on and unit j wins

prob. that unit j wins

Assume n_k is a constant N for all patterns S_k. Then:

$$w_{ij} = \frac{1}{N} \cdot P[\text{bit i} = 1 | \text{unit j wins}]$$

Unit j allocates the most weight to input lines that are most often on when j wins.

Local Fluctuations

The preceding analysis is based on averages over all training patterns.

But if patterns are presented randomly, there will be local fluctuations in p_k (variance from the mean value.)

Stability = insensitivity to small fluctuations.

How can we measure this?

High Stability From Pattern Overlap

Let α_{jl} = response of unit j to pattern S_l: $\alpha_{jl} = \sum_i w_{ij} c_{il}$

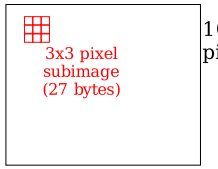
Let r_{lk} = overlap between S_l and S_k: $r_{lk} = \sum_i \frac{c_{ik} c_{il}}{n_k}$

Then at equilibrium: $\alpha_{jl} = \frac{\sum_k p_k v_{jk} r_{lk}}{\sum_k p_k v_{jk}}$

Conclusion: max stability (high α) when unit captures patterns that are highly overlapped.

What is Competitive Learning Good For?

Codebooks for compressing speech and image data.



1000 x 1000 pixel image

Create a codbook of 512 subimages. Can transmit a subimage in 9 bits instead of 216. Big savings!

Some loss of image quality.

Dimensionality Reduction by Competitive Learning on a Grid

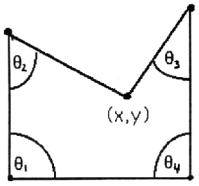
Establish an M-dimensional neighborhood relation among competitive units.

Ususally M=2 and the units are arrayed on a grid.

Map N-dimensional space of input points onto simpler M-dimensional space, without loss of information.

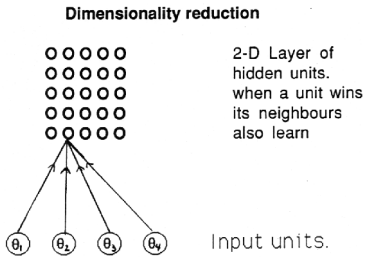
“Captures the structure” of a complex world.

Dimensionality Reduction

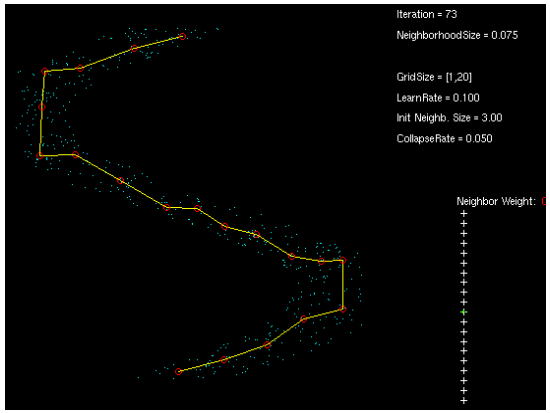


A two-armed robot with two degrees of freedom

Learns the topology of the robot's state space.

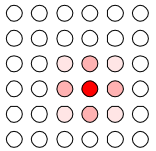


Dimensionality Reduction: Mapping a Curve in the Plane



Kohonen's Self-Organizing Feature Maps

1. Adjust weights of the winner, as in standard competitive learning.



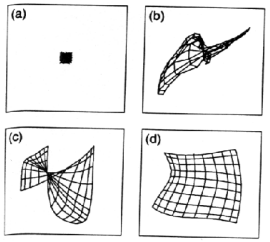
2. Adjust weights of neighboring units as well, by lesser amount.

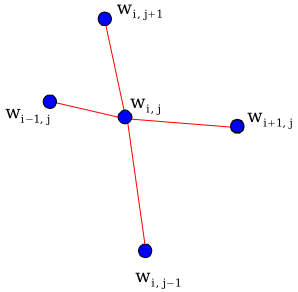
This is spatially localized “leaky learning”.

3. Slowly shrink the neighborhood function as learning progresses.

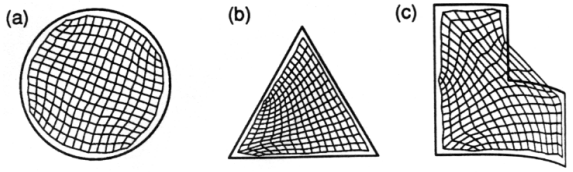
Nearby regions of feature space will be captured by adjacent units. Learns the topological structure of feature space.

Mapping the Plane





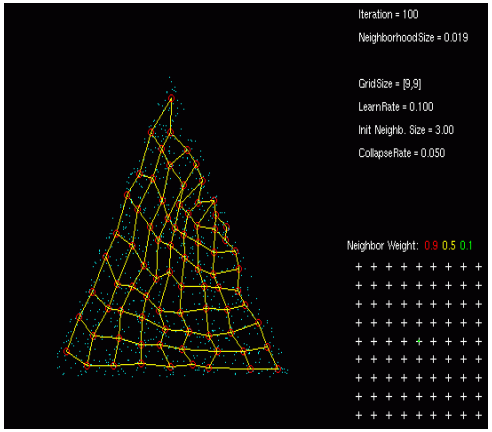
Mapping Planar Regions with a 15x15 Grid of Units



This makes for neat demos.

But Kohonen nets are most useful when the input space is of higher dimensionality than 2D.

Kohonen Demo

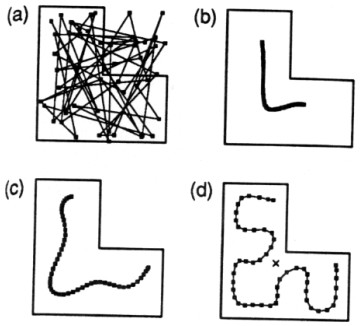


In the directory matlab/kohonen

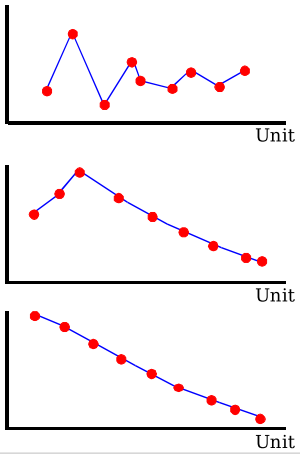
Set KohDataSet = 4 for triangle.

For instructions: help kohdemo

Mapping 2D to 1D: Space-Filling Curve



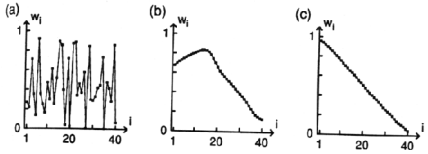
Kohonen's Ordering Theorem



Learning Theory (1D Case) of Kohonen Nets

Only two stable solutions exist: weights all increasing, or all decreasing.

No local minima.



Weight updates preserve monotonic regions of weights, except at the boundaries.

So: kinks may be eliminated, but are never added.

Theory of Kohonen Nets (cont.)

Two phases of learning:

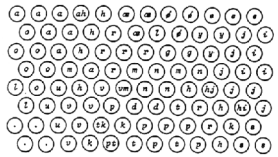
- "untangling"
- close fitting

Weight distribution is as:

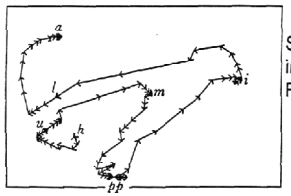
$$P(\xi)^{2/3}$$

Under-samples dense regions.

Encoding Speech (Kohonen 1988)



2-D map of phonemes



Sequence of activation in recognizing a Finnish word

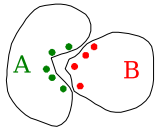
Competitive Networks as Classifiers

Suppose you want a fast nearest neighbor classifier.

You have labeled training data.

Solution: use a competitive network, with each unit assigned a class label.

Convex classes?
Use several units per class.



Kohonen's LVQ (Learning Vector Quantization)

LVQ can be used to train a prototype set automatically in order to produce a nearest neighbor classifier.

Compare the class of the training point with the class of the winning unit i.

$$\Delta w_i = \begin{cases} +g\cdot(\xi - w_i) & \text{if correct class} \\ -g\cdot(\xi - w_i) & \text{if wrong class} \end{cases}$$

Move prototype vector toward the training point if class is correct. Else move away.

Number of prototypes is fixed. Can we correct that?

Variants of LVQ

LVQ2: if winning unit i is wrong class but next closest unit j is correct class, then update both. Trains faster.

DSM (Decision Surface Mapping):
Adds new prototypes as needed.

If winning unit i is correct class, do nothing.

Otherwise, let j be the closest unit in correct class.

If j is "close enough", update units i and j.

Else create a new prototype of the same class as j, and place it at the location of the training point.