L16: competitive learning

Competitive learning

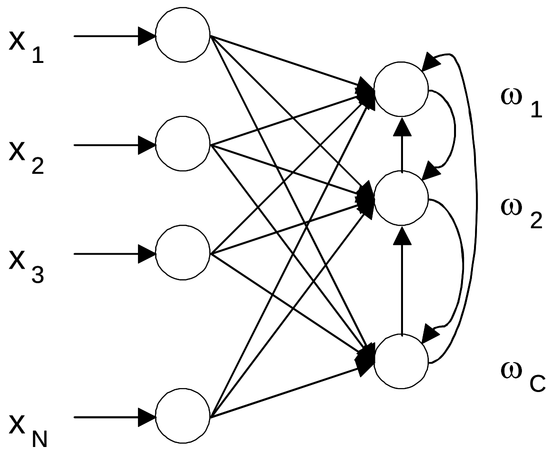
Adaptive resonance theory

Kohonen self-organizing maps

Competitive learning

A form of unsupervised training where output units are said to be in competition for input patterns

- During training, the output unit that provides the highest activation to a given input pattern is declared the winner and is moved closer to the input pattern, whereas the rest of the neurons are left unchanged
- This strategy is also called <u>winner-take-all</u> since only the winning neuron is updated
  - Output units may have lateral inhibitory connections so that a winner neuron can inhibit others by an amount proportional to its activation level



Neuron weights and input patterns are typically normalized

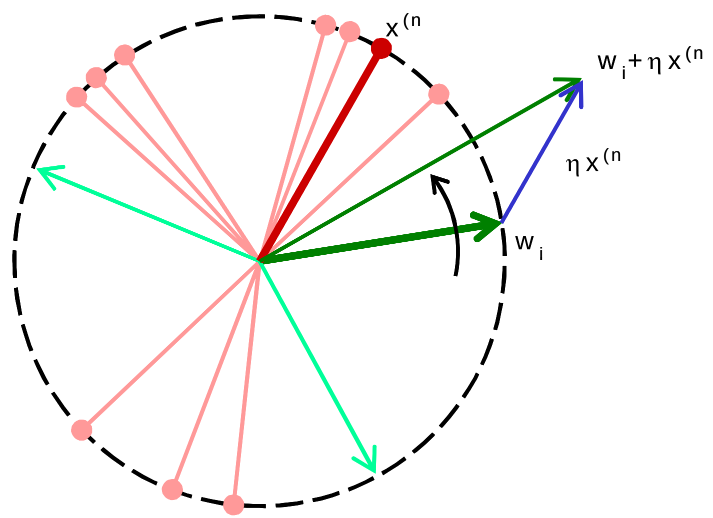
- With normalized vectors, the activation function of the i^{th} unit is the inner product of its weight vector w_i and the input pattern $x^{(n}$

$$g_i\left(x^{(n}\right) = w_i^T x^{(n}$$

- The neuron with largest activation is then adapted to be more like the input that caused the excitation

$$w_i(t+1) = w_i(t) + \eta x^{(n}$$

- Following adaptation, the weight vector is renormalized ($||w|| = 1$)

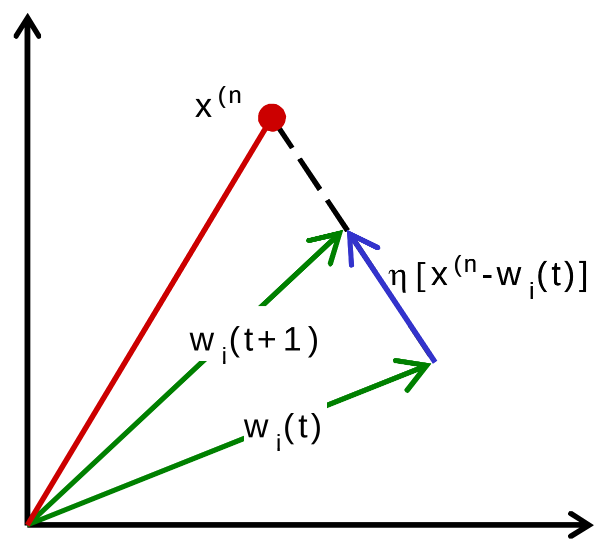


If weights and input patters are un-normalized, the activation function becomes the Euclidean distance

$$g_i\left(x^{(n)}\right) = \left\|w_i - x^{(n}\right\|$$

- In a neural-network implementation, we would use radial units instead of the conventional inner-product unit
- The learning rule then becomes

$$w_i(t+1) = w_i(t) + \eta\left(x^{(n} - w_i(t)\right)$$



Two implementations of competitive learning are presented

- A basic competitive learning scheme with fixed number of clusters
- The Leader-Follower algorithm of Hartigan, which allows a variable number of neurons

Basic competitive learning

1. Normalize all input patterns
2. Randomly select a pattern $x^{(n}$
   2a. Find the winner neuron
   $$i = argmax_j\left[w_j^T x^{(n}\right]$$
   2.b. Update the winner neuron
   $$w_i = w_i + \eta x^{(n}$$
   2c. Normalize the winner neuron
   $$w_i = \frac{w_i}{\|w_i\|}$$
3. Go to step 2 until no changes occur in N_{EX} runs

Leader-follower clustering

1. Normalize all input patterns
2. Randomly select a pattern $x^{(n}$
   2a Find the winner neuron
   $$i = argmax_j\left[w_j^T x^{(n}\right]$$
   2.b If $||x^{(n} - w_i|| < \theta$ (cluster and example are close)
   then update the winner neuron
   $$w_i = w_i + \eta x^{(n}$$
   else add a new neuron
   $$w_{new} = x^{(n}$$
   2c Normalize the neuron
   $$w_k = \frac{w_k}{\|w_k\|} \; where \; k \in \{i, new\}$$
3. Go to step 2 until no changes occur in N_{EX} runs

Adaptive Resonance Theory

Adaptive Resonance Theory (ART) is a family of algorithms for unsupervised learning developed by Carpenter and Grossberg

- ART is similar to many clustering algorithms where each pattern is processed by
  - finding the "nearest" cluster (a.k.a. prototype or template) to that example
  - updating that cluster to be "closer" to the example

What makes ART different is that it is capable of determining the number of clusters through adaptation

- ART allows a training example to modify an existing cluster only if the cluster is sufficiently close to the example (the cluster is said to “resonate” with the example); otherwise a new cluster is formed to handle the example
- To determine when a new cluster should be formed, ART uses a vigilance parameter as a threshold of similarity between patterns and clusters

There are several architectures in the ART family

- ART1, designed for binary features
- ART2, designed for analog features
- ARTMAP, a supervised version of ART

We will describe the algorithm called ART2-A, a version of ART2 that is optimized for speed

The ART2 algorithm

Let: α: positive number $\alpha \leq 1/\sqrt{N_{EX}}$

β: small positive number

θ: normalization parameter $1 < \theta < 1/\sqrt{N_{EX}}$

ρ: vigilance parameter $0 \leq \rho, 1$

0. For each example $x^{(n}$ in the database
   - 0a. Normalize $x^{(n}$ to have magnitude 1
   - 0b. Zero out coordinates $x^{(n} < \theta$ (remove small noise signals)
   - 0c. Re-normalize $x^{(n}$
1. Start with no prototype vectors (clusters)
2. Perform iterations until no example causes any change. At this point quit because stability has been achieved. For each iteration, choose the next example $x^{(n}$ in cyclic order
3. Find the prototype w_k(cluster) not yet tried during this iteration that maximizes $w_k^T x^{(n}$
4. Test whether w_k is sufficiently similar to $x^{(n}$

$$w_k^T x^{(n} \geq \alpha \sum_{j=1}^{N_{DIM}} x^{(n}(j)$$

   - 4a. If not then
     - 4a1. Make a new cluster with prototype set to $x^{(n}$
     - 4a2. End this iteration and return to step 2 for the next example
   - 4b. If sufficiently similar, then test for vigilance acceptability $w_k^T x^{(n} \geq \rho$
     - 4b1. If acceptable then $x^{(n}$ belongs to w_k. Modify w_k to be more like $x^{(n}$

$$w_k = \frac{(1-\beta)w_k + \beta x^{(n}}{\|(1-\beta)w_k + \beta x^{(n}\|}$$

       and go to step 2 for the next iteration with the next example
     - 4b2. If not acceptable, then make a new cluster with prototype set to $x^{(n}$

[Gallant, 1993]

The “stability-plasticity” dilemma

- A term coined by Grossberg that describes the problems endemic to competitive learning
- The network’s adaptability or plasticity causes prior learning to be eroded by exposure to more recent input patterns

ART resolves this problem by creating a new cluster every time an example is very dissimilar from the existing clusters

- **Stability**: previous learning is preserved since the existing clusters are not altered and
- **Plasticity**: the new example is incorporated by creating a new cluster

However, ART lacks a mechanism to avoid overfitting

- It has been shown that, in the presence of noisy data, ART has a tendency to create new clusters continuously, resulting in “category proliferation”
- Notice that ART is very similar to the leader-follower algorithm!

Unfortunately, ART also uses an obscure (biologically-inspired) terminology that clouds its simplicity

- Data are called an "arbitrary sequence of input patterns”
- The current training case stored in "short term memory" and clusters are "long term memory”
- A cluster is called a "maximally compressed pattern recognition code"
- The two stages of finding the nearest cluster are performed by an "Attentional Subsystem" and an "Orienting Subsystem”
  - The latter is said to perform "hypothesis testing", which simply refers to the comparison with the vigilance threshold, not to hypothesis testing in the statistical sense
- "Stable learning" means that the algorithm converges
- The claim that ART is "capable of rapid stable learning of recognition codes in response to arbitrary sequences of input patterns" simply means that ART converges to a solution
  - It does not mean that the clusters are insensitive to the sequence in which the training patterns are presented --quite the opposite is true. Extracted from [comp.ai.neural-nets FAQ]

Kohonen Self Organizing Maps

Kohonen Self-Organizing Maps (SOMs) produce a mapping from a multidimensional input space onto a lattice of clusters

- The key feature in SOMs is that the mapping is topology-preserving, in that neighboring neurons respond to "similar" input patterns
- SOMs are typically organized as 1D or 2D lattices (i.e., a string or a mesh) for the purpose of visualization and dimensionality reduction

Unlike MLPs trained with back-propagation, SOMs have a strong neurobiological basis

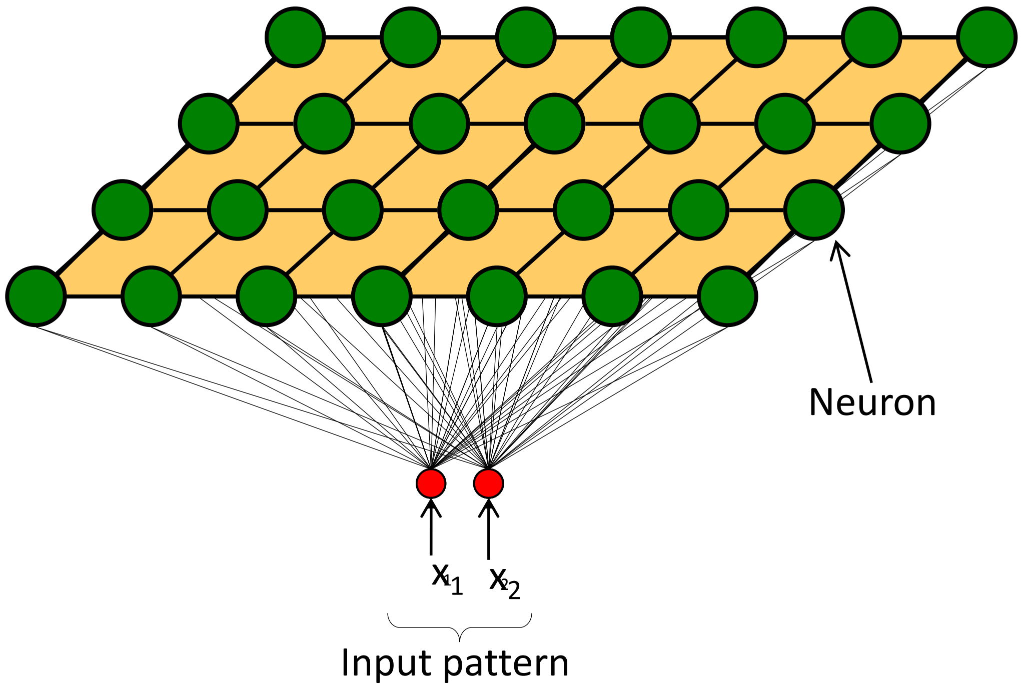
- On the mammalian brain, visual, auditory and tactile inputs are mapped into a number of "sheets" (folded planes) of cells [Gallant, 1993]
- Topology is preserved in these sheets; for example, if we touch parts of the body that are close together, groups of cells will fire that are also close together

K-SOMs result from the synergy of three basic processes

- Competition
- Cooperation
- Adaptation

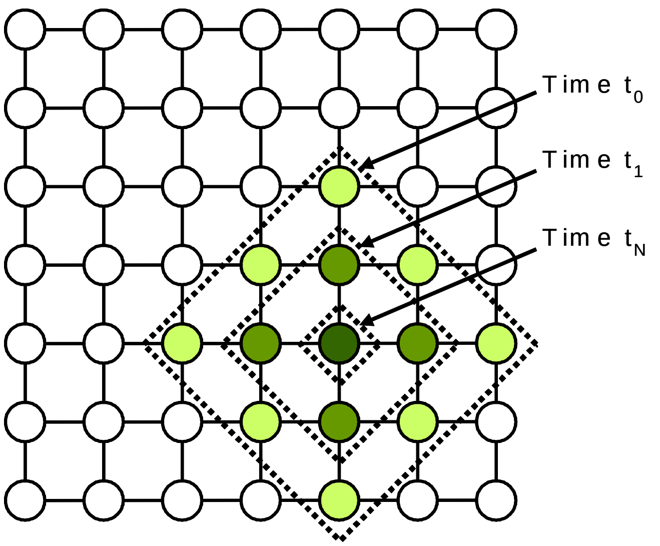
Competition

- Each neuron in a SOM is assigned a weight vector with the same dimensionality N as the input space
- Any given input pattern is compared to the weight vector of each neuron and the closest neuron is declared the winner
- The Euclidean norm is commonly used to measure distance



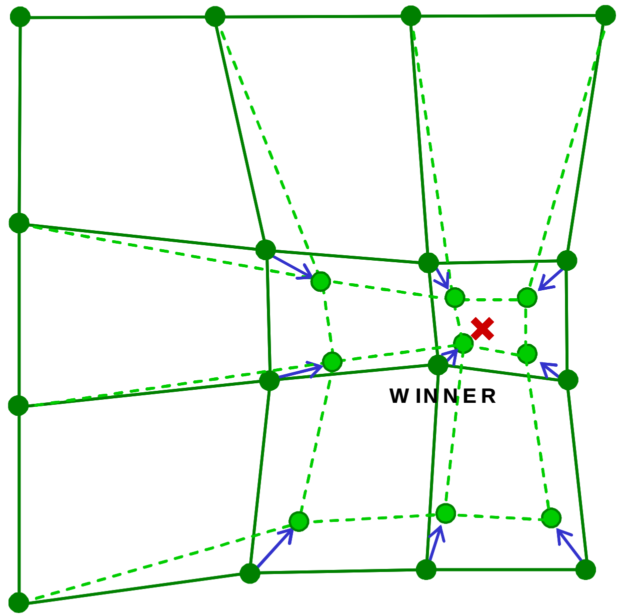
Cooperation

- The activation of the winning neuron is spread to neurons in its immediate neighborhood
  - This allows topologically close neurons to become sensitive to similar patterns
- The winner's neighborhood is determined on the lattice topology
  - Distance in the lattice is a function of the number of lateral connections to the winner (as in city-block distance)
- The size of the neighborhood is initially large, but shrinks over time
  - An initially large neighborhood promotes a topology-preserving mapping
  - Smaller neighborhoods allows neurons to specialize in the latter stages of training



Adaptation

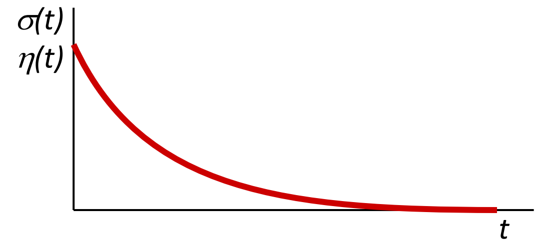
– During training, the winner neuron and its topological neighbors are adapted to make their weight vectors more similar to the input pattern that caused the activation
  - The adaptation rule is similar to the one presented in slide 4
  - Neurons that are closer to the winner will adapt more heavily than neurons that are further away
  - The magnitude of the adaptation is controlled with a learning rate, which decays over time to ensure convergence of the SOM



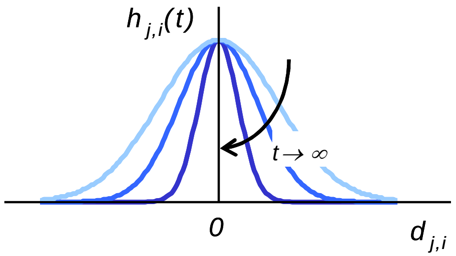
K-SOM algorithm

Define

- A learning rate decay rule $\eta(t) = \eta_0 e^{-t/\tau_1}$



- A neighborhood kernel function $h_{ik}(t) = e^{-\frac{d_{ik}^2}{2\sigma^2(t)}}$
  - where d_{ik} is the lattice distance between w_i and w_k
- A neighborhood size decay rule $\sigma(t) = \sigma_0 e^{-t/\tau_2}$



1. Initialize weights to some small, random values
2. Repeat until convergence
   - 2a. Select the next input pattern $x^{(n}$ from the database
     - 2a1. Find the unit w_i that best matches the input pattern $x^{(n}$

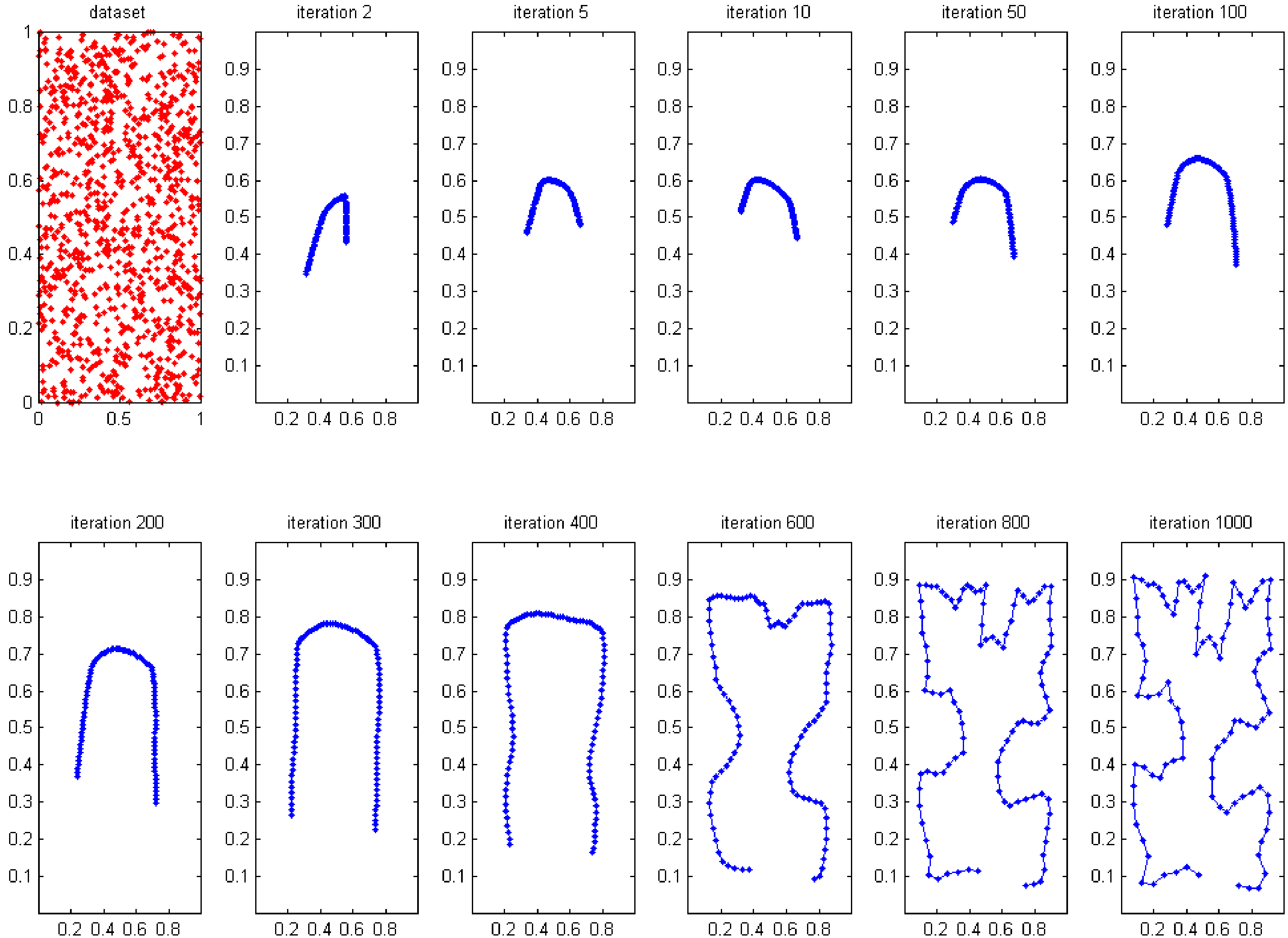
$$i = argmin_j \| x^{(n} - w_j \|$$

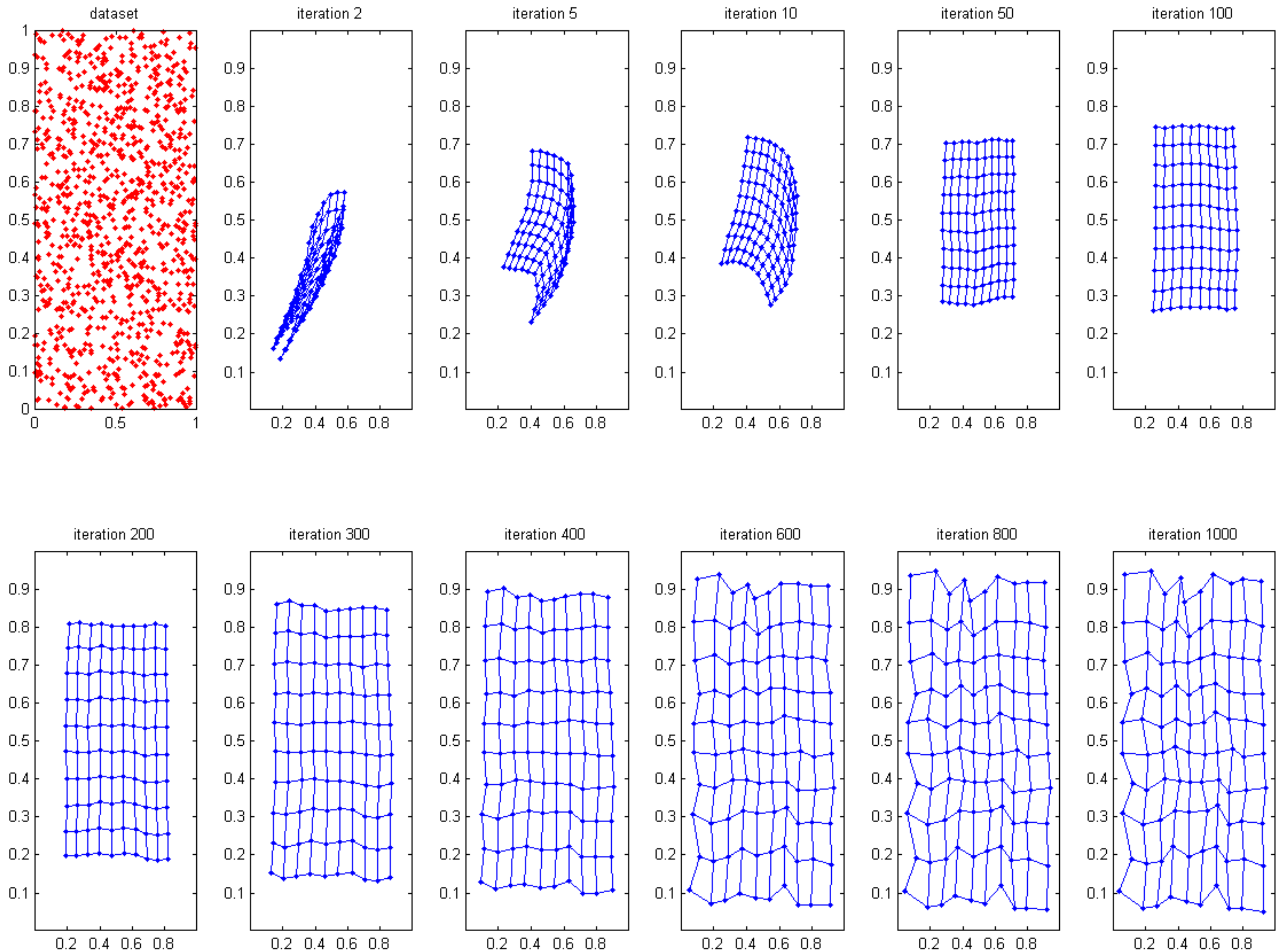
     - 2a2. Update the weights of the winner w_i and all its neighbors w_k

$$w_k = w_k + \eta(t) h_{ik}(t) \left(x^{(n} - w_k \right)$$

   - 2b. Decrease learning rate $\eta(t)$
   - 2c. Decrease neighborhood size $\sigma(t)$

K-SOM examples





dataset
iteration 2
iteration 5
iteration 10
iteration 50
iteration 100
iteration 200
iteration 300
iteration 400
iteration 600
iteration 800
iteration 1000