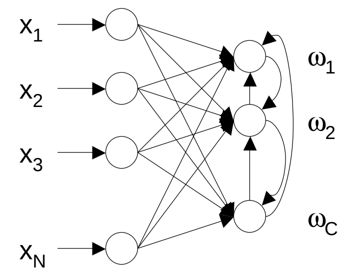
Lecture 15: Unsupervised learning II

- **Competitive Learning**
- **Adaptive Resonance Theory**
- **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 **winner-take-all** 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

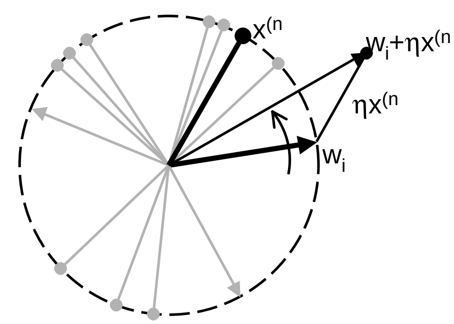


  - Neuron weights and input patterns are typically normalized
    - With normalized vectors, the activation function of the i^{th} unit can be computed as the inner product of the unit's weight vector w_i and a particular input pattern $x^{(n}$

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

      - Note: the inner product of two normal vectors is equal to the cosine of the angle between the vectors
    - The neuron with largest activation is then adapted to be more like the input that caused the excitation

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

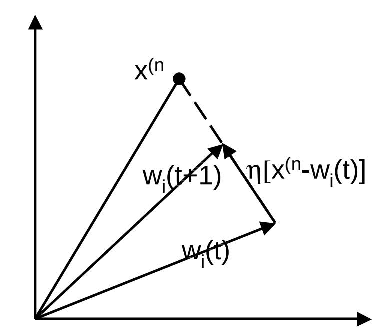


  - If unnormalized weights and input patters are used
    - The activation function becomes the Euclidean distance, and we loose the neural-network-alike forward propagation of inputs thought the weights (the inner product)

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

    - The learning rule then becomes

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



Competitive learning (2)

- **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
   $$i = \arg\max_j \left[w_j^T x^{(n} \right]$$
   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
   $$i = \arg\max_j \left[w_j^T x^{(n} \right]$$
   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
   $$w_k = \frac{w_k}{\|w_k\|} \quad \text{where } k \in \{i, new\}$$
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 Gail Carpenter and Steve Grossberg**
  - ART is similar to many iterative 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 $\alpha \le 1/\sqrt{N_{EX}}$

β: small positive number

θ: normalization parameter $0 < \theta < 1/\sqrt{N_{EX}}$

ρ: vigilance parameter $0 \le \rho < 1$

0. For each example $x^{(n}$ in the database
   - 0a. Normalize $x^{(n}$ to have magnitude 1
   - 0b. Replace coordinates of $x^{(n}$ that are $<\theta$ by 0 (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 $w_k^T x^{(n}$ (inner product of two normal vectors is equal to the cosine of the angle between the vectors)
4. Test whether w_k is sufficiently similar to $x^{(n}$

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

   - 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

$$w_k^T x^{(n} \ge \rho$$

     - 4b1. If acceptable then $x^{(n}$ belongs to w_k. Modify w_k to be more like $x^{(n}$

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

       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 to $x^{(n}$

Adaptive Resonance Theory (2)

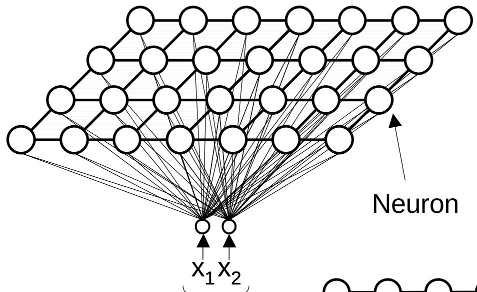
- **ART was motivated by the "stability-plasticity dilemma", a term coined by Grossberg that describes the problems endemic to competitive learning**
  - The network's adaptability or plasticity causes prior learning to be eroded by exposure to more recent input patterns
  - ART resolves this problem 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
- **The main limitation of ART is that it 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!

Kohonen Self Organizing Maps (1)

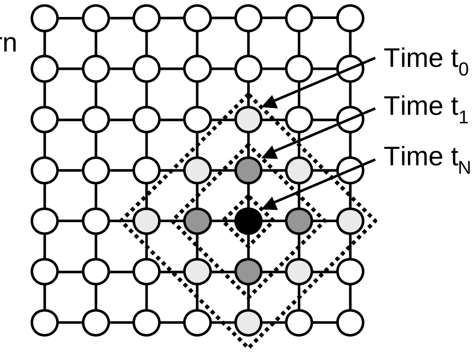
- **The purpose of Kohonen Self-Organizing Maps is to produce a mapping from a multidimensional input space onto a topology-preserving map of neurons**
  - The key feature in Kohonen SOMs is to preserve a topological order in the map so that neighboring neurons respond to "similar" input patterns
  - The topology of SOMs is typically a one- or two- dimensional lattice (a mesh) for the purpose of visualization and dimensionality reduction
- **Unlike MLPs trained with the back-propagation algorithm, SOMs have a strong neuro-biological 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
- **Kohonen SOMs result from the combination of three basic processes**
  - **Competition**: For a given input pattern, all the neurons compute a activation function, and the neuron with largest activation is declared a winner
  - **Cooperation**: In order to stimulate a topological ordering, the winner spreads its activation over a neighborhood of neurons in the map
  - **Adaptation**: Winner and neighboring neurons adapt their activation function in order to become more sensitive to that particular input pattern

Kohonen Self Organizing Maps (2)

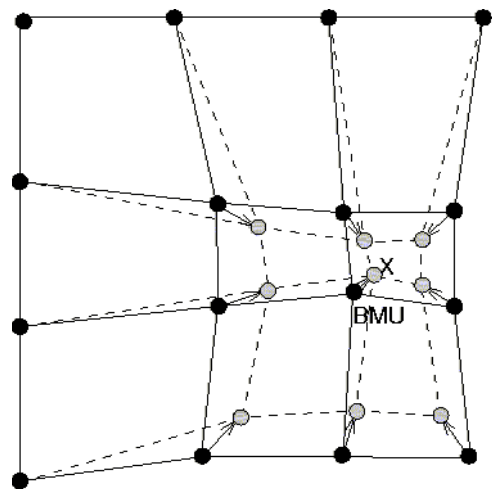
- **Competition**
  - Each neuron in a SOM is assigned a weight vector with the same dimensionality N as the input space
  - Any given input pattern is compared to the weight vector of each neuron and the neuron that is closest (Euclidean distance) is declared the winner



- **Cooperation**
  - The activation of the winning neuron is spread around neurons in its immediate neighborhood so that topologically close neurons will become sensitive to similar patterns
  - The neighborhood of the winner is determined from the topology of the lattice, in other words, it is a function of the number of lateral connections to the winner (i.e., city block distance)
  - The size of the neighborhood is initially large in order to preserve the topology, but shrinks over time to allow 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
  - Neurons that are closer to the winner will adapt more heavily than neurons that are further away
  - The adaptation rule is similar to the one used in Learning Vector Quantization: “neurons are moved a bit closer to the input pattern”
  - The magnitude of the adaptation is controlled with a learning rate, which decays over time to ensure convergence of the SOM



Kohonen Self Organizing Maps (3)

- **Define**
  - A learning rate decay rule $\eta(t)=\eta_0 \exp\left(-\frac{t}{\tau_1}\right)$
  - A neighborhood kernel function $h_{ik}(t)=\exp\left(-\frac{d_{ik}^2}{2\sigma(t)^2}\right)$
    - where d_{ik} is the lattice distance between w_i and w_k
  - A neighborhood size decay rule $\sigma(t)=\sigma_0 \exp\left(-\frac{t}{\tau_2}\right)$

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 $x^{(n}$

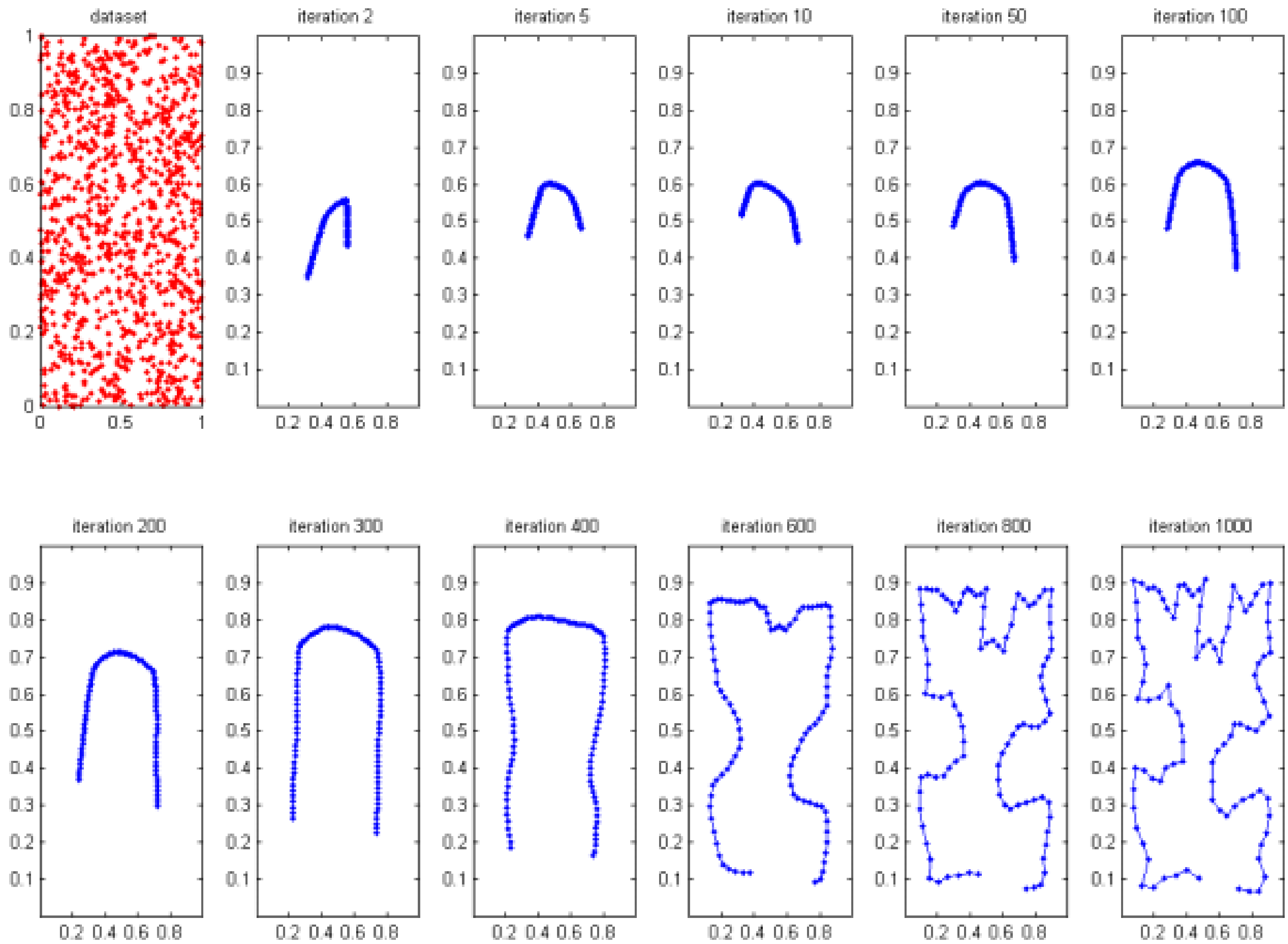
       $$i\left(x^{(n}\right)=\arg\min_j \left\| x^{(n}-w_j \right\|$$

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

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

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

SOM examples (1)



SOM examples (2)